DETECCIÓN Y SEGUIMIENTO DE OBJETOS EN SECUENCIA DE IMÁGENES POR MEDIO DE PROCESAMIENTO DE IMÁGENES Y FILTRO DE KALMAN

Jhoan Sebastian Colmenares Garcia

ID 000268655

Universidad Pontificia Bolivariana – Seccional Bucaramanga

Escuela de Ingeniería

Bucaramanga

2021

DETECCIÓN Y SEGUIMIENTO DE OBJETOS EN SECUENCIA DE IMÁGENES POR MEDIO DE PROCESAMIENTO DE IMÁGENES Y FILTRO DE KALMAN

Jhoan Sebastian Colmenares Garcia

ID 000268655

Proyecto de grado presentado como requisito para optar al título de:

INGENIERO ELECTRONICO

Director del Proyecto

Luis Ángel Silva, PhD

Universidad Pontificia Bolivariana – Seccional Bucaramanga

Escuela de Ingeniería

Bucaramanga

2021

Tabla de Contenidos

Lista de figuras

Universidad Pontificia Bolivariana
SECCIONAL BUCARAMANGA

RESUMEN GENERAL DE TRABAJO DE GRADO

TITULO: DETECCIÓN Y SEGUIMIENTO DE OBJETOS EN SECUENCIA DE IMÁGENES POR MEDIO DE PROCESAMIENTO DE IMÁGENES Y FILTRO DE KALMAN

AUTOR(ES): Jhoan Sebastian Colmenares Garcia

PROGRAMA: Facultad de Ingeniería Electrónica

DIRECTOR(A): Luis Angel Silva

RESUMEN

En este proyecto se desarrolló una aplicación de escritorio usando el IDE Python 3.9 enlazado a la librería OpenCV, implementando el algoritmo del filtro de Kalman, logrando la detección, el seguimiento y la predicción de la posición del objeto dada por el centroide de este. Inicialmente, se realizó una breve revisión bibliográfica del algoritmo del filtro de Kalman y su uso en la visión artificial y el procesamiento de imágenes. La aplicación de escritorio resultante contiene cuatro opciones en el menú principal, donde la primera fue llamada "UMBRALIZACIÓN", en la cual por medio de la captura de imágenes en tiempo real se convierte estas imágenes del espacio de color RGB al espacio de color HSV y por medio de deslizadores se encuentran los umbrales que contienen el objeto y esos datos de los umbrales son guardados para las pruebas futuras. La segunda opción del menú principal fue llamada "PREDICCIÓN / SEGUIMIENTO", donde se seleccionan los datos de los objetos que se desean detectar tomando los valores de los umbrales anteriormente encontrados. Con estos datos se realiza el procesamiento de imagen, inicialmente aplicando transformaciones morfológicas y luego extrayendo características como los contornos y centroides de los objetos, tomando los centroides como la posición del objeto en el plano coordenado (x, y). Por medio de ecuaciones lineales relacionando el diámetro en pixeles del objeto con distancias medidas manualmente, se halla un aproximado de la posición en el eje z. Posteriormente con los datos de las posiciones de los objetos, el filtro de Kalman el cual se encarga de predecir la posición futura en los tres ejes manejando así tres dimensiones. Los datos tomados son guardados en un documento plano para luego ser graficados y analizados. La tercera opción del menú principal fue llamada "GRAFICAS RESULTADO", donde por medio de la librería Matplotlib se realizaron las gráficas de comparación de trayectorias en cada uno de los ejes, la gráfica de movimiento en el plano coordenado (x, y) y la gráfica en el espacio (x,y,z). La cuarta y última opción del menú principal fue llamado "SALIR" encargada de terminar el proceso y cerrar la aplicación. Finalmente se realizaron pruebas en tiempo real capturando las imágenes mediante la cámara web del computador portátil y con un ambiente no controlado entregando resultados de detección, seguimiento y predicción de la posición con resultados considerablemente aproximados, con una detección eficiente de acuerdo al color de cada objeto y valores muy similares entre la posición del objeto y la posición predicha por el filtro de Kalman como también en las trayectorias entregando mayor suavidad en los cambios de dirección en la posición predicha en comparación con la posición medida.

PALABRAS CLAVE:

Detección, seguimiento, procesamiento de imagenes, filtro Kalman

V° B° DIRECTOR DE TRABAJO DE GRADO

Universidad Pontificia Bolivariana
SECCIONAL BUCARAMANGA

GENERAL SUMMARY OF WORK OF GRADE

TITLE: DETECTION AND TRACKING OF OBJECTS IN IMAGE SEQUENCE BY MEANS OF IMAGE PROCESSING AND KALMAN FILTER

AUTHOR(S): Jhoan Sebastian Colmenares Garcia

FACULTY: Facultad de Ingeniería Electrónica

DIRECTOR: Luis Angel Silva

ABSTRACT

In this project, a desktop application was developed using the Python 3.9 IDE linked to the OpenCV library, implementing the Kalman filter algorithm, achieving the detection, monitoring and prediction of the object's position given by its centroid. Initially, a brief literature review of the Kalman filter algorithm and its use in computer vision and image processing was conducted. The resulting desktop application contains four options in the main menu, where the first one was called "UMBRALIZACIÓN", in which by means of the capture of images in real time these images are converted from the RGB color space to the HSV color space and by Middle of sliders are the thresholds that contain the object and that threshold data is saved for future testing. The second option on the main menu was called "PREDICCIÓN / SEGUIMIENTO", where the data of the objects to be detected are selected taking the values of the previously found thresholds. Image processing is carried out with these data, initially applying morphological transformations and then extracting characteristics such as the contours and centroids of the objects, taking the centroids as the position of the object in the coordinate plane (x, y). By means of linear equations relating the diameter in pixels of the object with manually measured distances, an approximation of the position on the z axis is found. Subsequently with the data of the positions of the objects, the Kalman filter which is in charge of predicting the future position in the three axes, thus managing three dimensions. The data collected is saved in a flat document to later be plotted and analyzed. The third option of the main menu was called "GRAFICAS RESULTADO", where by means of the Matplotlib library the graphs of comparison of trajectories in each of the axes, the graph of movement in the coordinate plane (x, y) and the graph in space (x, y, z). The fourth and last option of the main menu was called "SALIR" in charge of ending the process and closing the application. Finally, tests were carried out in real time, capturing the images through the webcam of the laptop and with a non-controlling environment, delivering results of detection, monitoring and prediction of the position with considerably approximate results, with efficient detection according to the color of each object. and very similar values between the position of the object and the position predicted by the Kalman filter as well as in the trajectories giving greater smoothness in the changes of direction in the predicted position compared to the measured position

KEYWORDS:

DETECTION, TRACKING, IMAGE PROCESSING, KALMAN FILTER

V° B° DIRECTOR OF GRADUATE WORK

1. INTRODUCCION

Con el paso del tiempo, la humanidad ha sido testigo de los avances tecnológicos que se van logrando de manera acelerada, estos avances han presentado grandes beneficios para la investigación y la industria, convirtiéndose en herramientas de gran utilidad para tareas rutinarias en estos campos. En la industria, esto se ha traducido en ahorro en mano de obra y tiempos de ejecución en los procesos, como a su vez mejora en la calidad de producción, en estas herramientas se encuentra aspectos como el aprendizaje automático, la inteligencia artificial, el internet de las cosas entre otros, puntos base de la transformación digital conocida como Industria 4.0.

La inteligencia artificial, en su defecto, es la inteligencia realizada por maquinas manteniendo un aprendizaje automático el cual les permite ir actuando de acuerdo con las tareas, datos y requerimiento que van recibiendo, superando obstáculos y encontrando problemas y defectos de una manera eficiente. La visión artificial y el procesamiento de imágenes han tenido gran influencia en el avance tecnológico de los últimos años, aplicadas tanto en la industria como también con un enfoque comercial, en ámbitos de seguridad, entretenimiento como también en producción en masa, control automático, entre otros.

La detección de objetos y la caracterización de parámetros por medio de la visión por computador permite la ejecución de algoritmos para el tratamiento de dichos datos logrando entre otros, el seguimiento de objetos y predicción de posición es futuras, punto importante en el objetivo general de este proyecto, donde se propone por medio del lenguaje de programación Python, el uso de la librería OpenCv y un estimador de estado como el filtro de Kalman, desarrollar una aplicación de escritorio con interfaz gráfica en tiempo real del seguimiento de dos objetos esféricos de color azul y verde, y a su vez, presentar los resultados para el análisis en gráficas y recopilados en un documento plano de Excel.

2. PLANTEAMIENTO DEL PROBLEMA

La visión artificial teniendo como objetivo el modelamiento matemático de los procesos de percepción visual y la generación de programas que permitan simular estas capacidades visuales en un computador. Permite como herramienta establecer una relación entre el mundo tridimensional (3D) y las vistas bidimensionales que se toman de él. A su vez, el procesamiento de imágenes ha tenido gran influencia en el desarrollo de nuevas estrategias para la optimización en el funcionamiento de maquinaria y robots en la industria con usos como encontrar defectos en productos, control de calidad, desalineación en producción en masa, entre otros. En estos procesos la necesidad de automatizar en mayor medida el trabajo de los robots y maquinaria, presentan limitaciones en la velocidad de reacción ante imprevistos, en estos casos, la posibilidad de seguimiento y predicción de situaciones futuras toma importancia. Ante dicha posibilidad, el uso de estimadores matemáticos que permitan de alguna manera intuir y adelantarse a los imprevistos no habituales en procesos industriales, proponen una solución a minimizar los riesgos que estos conllevan.

Con el paso del tiempo, han sido desarrollados un significativo número de algoritmos con base a estimadores matemáticos como lo es el estimador de estado, el filtro de partículas y el filtro de Kalman. En el caso del filtro de Kalman, es importante resaltar que es uno de los estimadores ampliamente usado en la industria por su sencillez y eficiencia en las tareas de predicción y seguimiento de objetos. ¿Puede el algoritmo del filtro de Kalman ser una herramienta que permita la predicción y seguimiento de objetos en tres dimensiones?

3. JUSTIFICACIÓN DEL PROBLEMA

Con la necesidad presente de la optimización de sistemas robóticos autónomos e inteligentes, el procesamiento de imágenes ha tomado gran importancia, permitiendo el desarrollo de aplicaciones para localizar objetos, seguirlos, predecir posición es futuras, etc.

El Filtro de Kalman con el cual se logra predecir datos futuros, lo cual ayuda entre otros, a minimizar sobrecargas causadas por movimientos bruscos de los robots, menor tiempo y anticipación de reacción entre controlador y actuador. Además, permite la detección, seguimiento y predicción de la posición y movimiento de los objetos, con el fin de obtener información valiosa para luego poder implementar estrategias matemáticas que ofrecen diferentes algoritmos diseñados para predecir posición es futuras e ir corrigiendo anomalías que se vayan presentando, logrando que el robot sea lo más autónomo posible, ahorrando inversiones en tiempo, dinero y personal.

En este proyecto se enmarca en la línea de investigación de reconocimiento de patrones del grupo de control industrial de la universidad Pontificia Bolivariana y permitirá desarrollar algoritmos de detección y predicción de objetos en movimiento para futuras aplicaciones de captura de objetos por parte del robot presente en el laboratorio de control de procesos industriales.

4. OBJETIVOS

4.1 Objetivo general

Detectar y seguir dos objetos circulares de diferente color en una secuencia de imágenes adquiridas por medio de una cámara web utilizando el filtro de Kalman

4.2 Objetivos específicos

- Desarrollar un algoritmo para la captura de imágenes a través de una cámara web y procesamiento básico de imágenes en el entorno de programación Python, empleando la librería OpenCV.

- Detectar dos objetos circulares de color azul y verde en ambientes de iluminación controlada por medio de la librería OpenCV y extraer características relevantes relacionadas con el centroide y el área.

- Implementar el algoritmo del filtro de Kalman con las características extraídas de los objetos circulares para realizar su seguimiento en una secuencia de imágenes y analizar el resultado en tareas de detección y predicción.

- Diseñar y desarrollar interfaz gráfica de visualización de resultados empleando la librería Tkinter.

5. MARCO TEÓRICO

En esta sección se abarcará temas relacionados con el procesamiento digital de imágenes, algunos modelos de color, el lenguaje de programación Python y la librería OpenCV como herramienta en visión artificial.

5.1 Procesamiento de imágenes

El objetivo principal del procesamiento de imágenes es hacer evidentes detalles y características propias de la imagen o imágenes en cuestión, a su vez, permite mejorar el aspecto de esta. Una imagen está representada en dos dimensiones por una función f (x, y), la cual describe la amplitud en cada punto llamada nivel de gris o intensidad de la imagen y donde x e y representan las coordenadas en un plano. Estas imágenes pueden ser generadas de diferentes maneras, como, por ejemplo, fotográfica o electrónicamente, por medio de monitores de televisión. Generalmente el procesamiento de imágenes se realiza en medios digitales (computadora), donde la composición de cada imagen digital es un número finito de elementos y cada uno tiene una posición y valor particular. A dichos elementos se les denomina pixeles (Aguirre, N. 2013). Al momento de realizar un procesamiento de imagen se debe tener presente los diferentes espacios de color, entre los cuales se encuentran:

- Modelo RGB, siendo uno de los más usados por los sistemas informáticos para crear y reproducir colores en monitores y pantallas. Está apoyado en la llamada "síntesis aditiva", donde se constituye por la suma de las intensidades de luz rojas, verdes y azules consiguiendo diferentes colores en rango mínimo negro y rango máximo blanco.
- Modelo HSV, se define como tono (Hue), saturación (saturation) y valor (value) respectivamente. De forma cilíndrica es su sistema coordenado y una pirámide de base hexagonal para su subconjunto del espacio el cual define su color.

5.2 Lenguaje de programación Python

Python es un lenguaje de programación a código abierto e interpretado cuya filosofía primeriza una sintaxis que hace favorable un código legible. Se trata de un lenguaje de tipo multiparadigma, por lo que soporta programación orientada a objetos, programación imperativa y, en menor medida, programación funcional. Es catalogado como un lenguaje

de alto nivel, así como lo son también C, C++, Perl y java. Fue creado a principios de los noventa por Guido van Rossum en los Países Bajos (González, R. 2015).
Presenta una sintaxis sencilla contando con una muy buena cantidad de bibliotecas como herramientas que lo hacen un lenguaje de programación muy útil y único. Estas bibliotecas permiten el desarrollo de aplicaciones en cantidad infinitas de tareas, tales como desarrollo de app web, inteligencia artificial, procesamiento de imágenes, investigación, big data, etc.

5.3 OpenCV

OpenCV (Open Source Computer Vision Library) es una biblioteca de software de visión artificial y aprendizaje automático de código abierto. OpenCV proporciona una herramienta para diferentes aplicaciones de visión por computador que permite una aceleración del uso de la precepción de las maquinas en diferentes productos comerciales. Al ser un producto con licencia BSD, esta librería permite que empresas y personas puedan usar y modificar el código con libertad.

La biblioteca tiene más de 2500 algoritmos, entre estos se encuentra un conjunto de algoritmos de visión por computador y aprendizaje automático clásicos y de última generación. Dichos algoritmos, pueden ser usados en la detección y reconocimiento de rostros, rastrear objetos en movimiento, extraer modelos 3D de objetos, seguir los movimientos oculares, etc. Tiene interfaces C ++, Python, Java y MATLAB y es compatible con Windows, Linux, Android y Mac OS.

Muy seguramente, OpenCV sea la biblioteca de visión artificial más importante y usada del mundo. Siendo usada por universidades, empresas y personas del común teniendo a favor ser un software libre (Casares, C. et al., 2017).

5.4 Tkinter

Tkinter es una librería por defecto instalada en el lenguaje de programación, al ser un paquete estándar de creación de interfaz gráfica (GUI), funciona en un amplio porcentaje de sistemas operativos. Dependiendo a la versión de Python que este en uso, el método de llamar elementos de la librería puede variar.

Presenta simplicidad a la hora de diseñar interfaces y se complementa presentando herramientas para la construcción de estas. Entre dichas herramientas se encuentran botones, graficas, textos, ventanas, etc. facilitando la interacción hombre – software.

5.5 Filtro de Kalman

El filtro de Kalman es en esencia, un grupo de ecuaciones matemáticas seriales donde implementan un estimador del tipo predictor – corrector, que minimiza el error estimado de la covarianza, en ciertas condiciones dadas. (Rodríguez Muñoz, P. 2003.)

Como objetivo principal del filtro, busca en un sistema dinámico obtener un pronosticador de sus variables de estado, basado en un modelo de incertidumbre de la dinámica del sistema y observaciones ruidosas.

El filtro de Kalman trata de estimar el estado X perteneciente a Rn de un proceso controlado en tiempo discreto, gobernado por una ecuación x_{k+1} (1).

$$x_{k+1} = Ax_k + B\mu_k + W_k \quad (1)$$

Con una medición $z \in \mathbb{R}n$ como se muestra en la ecuación (2):

$$z_k = Hx_k + V_k \quad (2)$$

Las variables aleatorias $Vk\ y\ Wk$ expresan respectivamente la medición y el ruido del proceso. Cuando el filtro de Kalman se aplica a la Visión Artificial, la posición corresponde al estado x siendo este el vector de posición del objeto detectado donde se determina por coordenadas en el eje x y y, donde las coordenadas (xx, xy) y (vx, vy) constituyen el contenido de dicho vector. La medida z, está conformado por un vector de dos componentes (zx, zy), definida como la posición observada del objeto.
La matriz $ANxM$ relaciona el estado presente (k) con el siguiente estado (k+1). Esta relación se manifiesta en las siguientes ecuaciones, dando como resultado la matriz A como se muestra en la ecuación (3).

$$\begin{aligned} x_{x_{k+1}} &= x_{x_k} + v_x t \\ x_{y_{k+1}} &= x_{y_k} + v_y t \\ v_{x_{k+1}} &= v_{x_k} \\ v_{y_{k+1}} &= v_{y_k} \end{aligned} \qquad A = \begin{Bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{Bmatrix} \quad (3)$$

La matriz $BNx1$ donde se vincula la entrada unitaria de control (u) que pertenece a los reales, con el vector de estado (x) y la matriz $HNxM$ (4) relaciona el vector de estado (x) con la medida (zk).

$$H_k = \begin{Bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{Bmatrix} \quad (4)$$

La descripción del filtro de Kalman con sus ecuaciones puede verse en la figura 1:

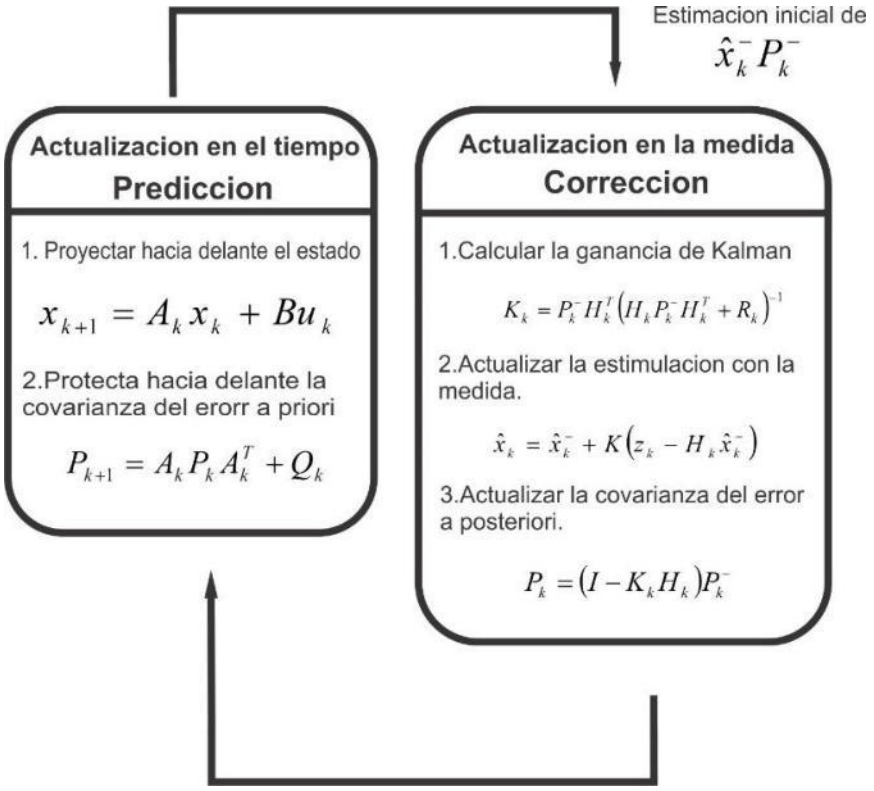


Figura 1. Descripción Algoritmo Filtro de Kalman

6. ESTADO DEL ARTE

Se presenta una breve revisión de investigaciones anteriores con referencia a los temas de relevancia del proyecto, puntualmente son detección y seguimiento de objetos, visión por computador, procesamiento digital de imágenes. Enfocando como herramientas principales lenguaje de programación Python, la librería OpenCV y el algoritmo del Filtro de Kalman.

- Tracking automático de objetos en secuencias de imágenes usando Filtro de Partículas (Chaparro Laso, 2017).

Se realiza por medio del lenguaje de programación c++ y la librería OpenCV la segmentación por umbralizado, sustrayendo fondo de la imagen para así poder aplicar el filtro de partículas a dos objetos diferentes. La toma de imágenes se realizó por medio de una cámara Sony DCR-HC18E para obtener dichas imágenes de mejor calidad y formato y por medio de Gantt Project se desarrolla el diagrama de Gantt para planificar los objetivos del proyecto. Se comprueba que el Filtro de Kalman estima mejor la posición del objeto de interés debido a su carácter matemático, frente al carácter probabilístico del Filtro de Partículas.

- Aplicación del filtro de Kalman al seguimiento de objetos en secuencias de imágenes (Rodríguez Muñoz, 2003).

Se detectan objetos en diferentes ambientes determinando posición y extrayendo los datos necesarios en cada una de las pruebas. Donde se usa el software Matlab y la librería OpenCV con ayuda de la aplicación TRIPOD realizando un procesamiento de imágenes online para ir revisando los resultados de cada uno de los procesamientos efectuados a la imagen. Se pudo afirmar que el filtro de Kalman como método, es una herramienta eficiente en tareas de predicción en gran parte del tiempo en un amplio rango de sucesos evolutivos.

- Diseño de un sistema de seguimiento utilizando filtros de Kalman y filtros de correlación adaptativos (Ontiveros Gallardo, 2015).

Se realiza seguimiento realizando múltiples detecciones de la figura de interés en espacios delimitados de estudio. Uso de funciones discriminantes sintéticas (SDF) y un valor de capacidad de discriminación deseado. Finalmente se evalúa rendimiento. Se demuestra un nivel de confianza alto en términos de

eficiencia en la detección y precisión en el seguimiento del objeto. El sistema demuestra que es posible seguir y estimar el estado de un objeto con gran precisión aun en la presencia de ruido moderado en las imágenes e incluso a través de fondos complejos.

- Sistema de seguimiento de objetos usando OpenCV, ArUco y Filtro de Kalman extendido (Jiménez Bravo, 2018)

Estimación de posición de un objeto mediante visión artificial, involucrando detección y seguimiento de dicho objeto. El proyecto fue desarrollado en C++ con uso de librerías OpenCV y ArUco. Por medio de dos cámaras y un marcador se implementa un bloque de posición amiento, implementación de un módulo de localización y un módulo estimador (Filtro de Kalman). Los resultados obtenidos (posición del objeto) presentan errores en bajo porcentaje comprobando la eficiencia del sistema.

- Computer Vision Face Tracking For Use in a Perceptual User Interface (Bradski, Gary, 1998).

Desarrollo de un algoritmo de seguimiento de color por medio de visión por computador y lo aplicaron al seguimiento de rostros humanos. El proyecto consto con una interfaz de usuario la cual tenía como objetivo ser rápida y eficiente, siendo capaz de rastrear en tiempo real con un pequeño porcentaje de recursos computacionales. La propuesta era encontrar el modo de una distribución de color dentro de una escena de video, por lo que el algoritmo de desplazamiento medio se modificó para trabajar con distribuciones de probabilidad de color que cambian dinámicamente.

- Detección y seguimiento de personas basado en estereovisión y Filtro de Kalman (Garcia J., 2012).

Se presenta un sistema de conteo de personas a través de un sistema de estereovisión ubicado en cierta posición, pretendiendo que el sistema resuelva errores de conteo con perturbaciones que son expuestas en el desarrollo del proyecto. El sistema propuesto se puede adaptar a diferentes entornos interiores como al igual en diferentes alturas por lo que los parámetros se pueden ajustar para mantener los resultados de detección.
El proyecto consta de 4 etapas de actualización de fondo, preprocesamiento, detección de cabezas y aplicación del filtro de Kalman

- Detección y seguimiento de objetos móviles en secuencias de video (Ormaechea Rafael, 2015).

Se propone el desarrollo de un sistema de vigilancia propio aplicando bancos móviles en secuencias de video, que se puede convertir en una aplicación para dispositivos móviles. Se realiza una caracterización del comportamiento del algoritmo de detección SSIM y el

filtro de Kalman al aplicarse sobre imágenes que se desplazan en todas las direcciones, realizando la detección en solo una parte de estas. Desarrollando un prototipo en Matlab donde se realiza un proceso de entrenamiento, validación y corrección de problemas.

- Object detection and tracking using deep learning and artificial intelligence for video surveillance applications (Ravish Aradhya, 2019).

En este proyecto, se diseña un algoritmo eficiente para la detección y el seguimiento de objetos para la videovigilancia en entornos complejos. Se realiza un entrenamiento con un banco de datos, donde el 80% se usan para esta actividad y el 20% restante para las pruebas. Se considera que la imagen encuentra objetos en ella mediante el uso de algoritmos CNN y YOLOv3.

- Real time object detection and tracking using deep learning and OpenCV (Chandan G., Harsh J., 2018)

En el artículo se implementan algoritmos para la detección y seguimiento en el entorno Python. Estos algoritmos son Single Shot Detector (SSD) y MobileNets algorithm. La detección de objetos presenta diferentes métodos como la diferenciación de cuadros y resta de fondo, donde se detecta y separa primer plano o persona del fondo de la imagen para su posterior procesamiento. El seguimiento de objetos se realiza en secuencias de video como cámaras de seguridad y CCTV.

7. METODOLOGÍA

Se realizaron 5 etapas importantes en el desarrollo de este proyecto, las cuales se relacionan a continuación:

ETAPA 1: Inicialmente se realizó una breve revisión bibliográfica sobre investigaciones y trabajos anteriores en los cuales se haya hecho uso de las herramientas Python, OpenCV y el algoritmo de Kalman, como también, de tareas como visión por computador, procesamiento de imágenes y el seguimiento de objetos.

ETAPA 2: Se continuo con la familiarización con el entorno Python 3.8.3 el cual es la base en la escritura del código, el uso de las librerías OpenCV y la herramienta "KalmanFilter". Realizando las respectivas instalaciones del software.
Por desarrollo matemático se encontraron las matrices que alimentaron dicha herramienta para el tracking del o los objetos a procesar.

ETAPA 3: Se realizo la adquisición de datos por medio de una cámara web, donde por medio de un algoritmo diseñado para modificar manualmente los valores hasta encontrar los valores del rango de cada objeto en modelo de color HSV. A ellos se le aplicaron técnicas de umbralización y segmentación por color y con los valores obtenidos y ayuda de la librería OpenCV se hallarán bordes, contornos y centroides de los objetos de importancia.

ETAPA 4: Se implemento el algoritmo del filtro de Kalman con la herramienta "KalmanFilter" incluida en la librería OpenCV, con la cual se predijeron posición es futuras de los objetos. Alimentando la función con los datos hallados en la ETAPA 2, teniendo en cuenta que el sistema de predicción se hizo para 2 dimensiones, donde se evaluaron los planos (X, Y) y (X, Z) para así encontrar los valores en 3 dimensiones.

ETAPA 5: Se evaluaron los resultados mediante revisión de las gráficas de los datos obtenidos, donde también se crearon las tablas compilando los resultados obtenidos para comparar los encontrados en diferentes pruebas llegando con estos a las conclusiones del proyecto.

ETAPA 6: Se hizo entrega de la aplicación funcional y realización del documento final de tesis.

8. DESARROLLO METODOLÓGICO

En el diagrama de la figura 2, se describe los procesos realizados cronológicamente, para el desarrollo de la aplicación de escritorio para la detección, seguimiento y predicción de posición de objetos.

Figura 2. Desarrollo metodológico.

8.1 Instalación software Python

El software para instalar es Python en su versión 3.9.2, el cual se encuentra de manera gratuita en la página web oficial de Python (Python.org).

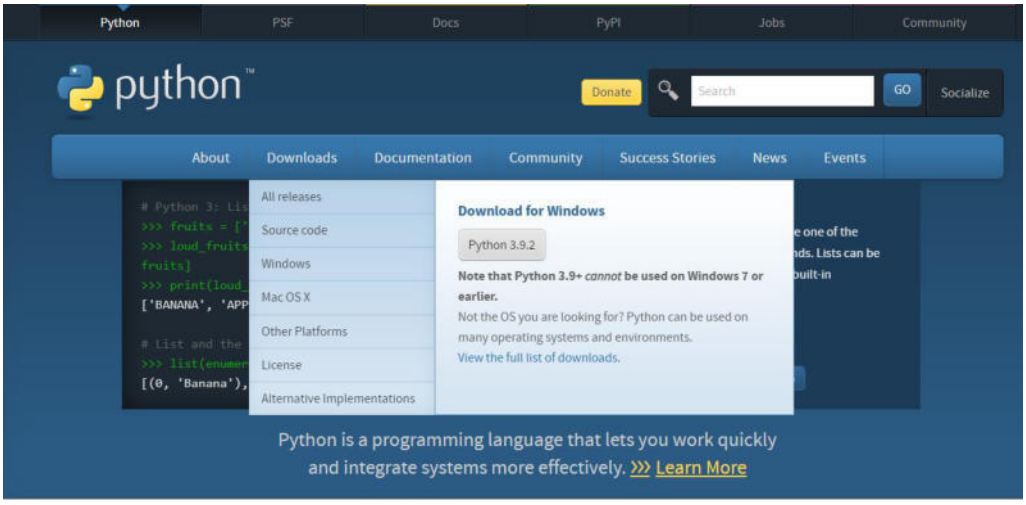


Figura 3. Pantalla principal Python.org (Python.org,2021)

Luego de obtener el instalador “python-3.9.2-amd64.exe”, se procede a ejecutarlo y completar la instalación :

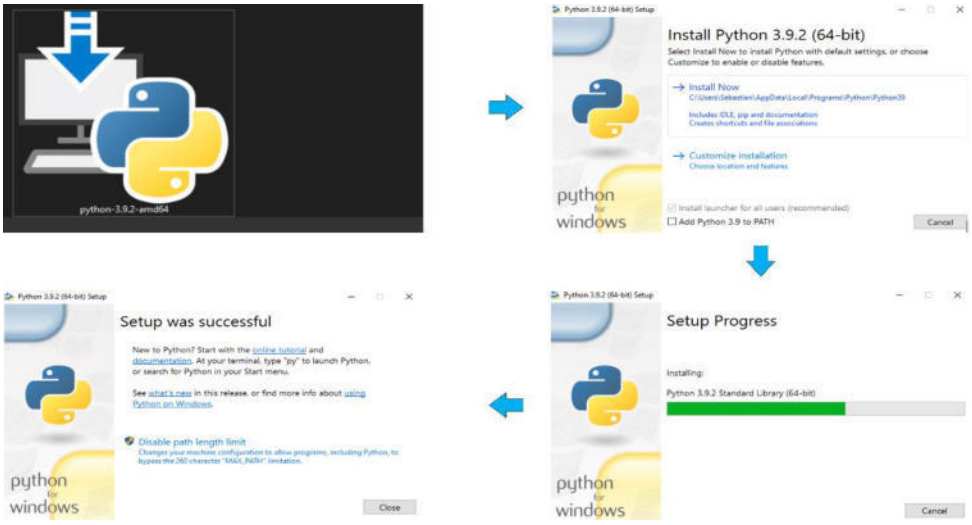


Figura 4. Proceso de instalación Python 3.9.2

8.2 Instalación e importación de librerías (OpenCV, Openpyxl, matplotlib)

Una de las opciones más usada en el proceso de instalación de librerías para Python es por medio del módulo “pip” y la ventana del símbolo de sistema (cmd), como se ve en la Figura 7. Para proceder en la instalación es necesario dirigirse a la carpeta de

instalación de Python 3.9 y luego en la subcarpeta “Scripts”, copiando la ruta de dirección de dicha carpeta como se muestra a continuación:

- Desde la carpeta de instalación del interprete Python, en este caso ubicada en la ruta “C:\Users\”Nombre_del_equipo”\AppData\Local\Programs\Python\Python39”, se procede a buscar e ingresar a la carpeta “Scripts”.

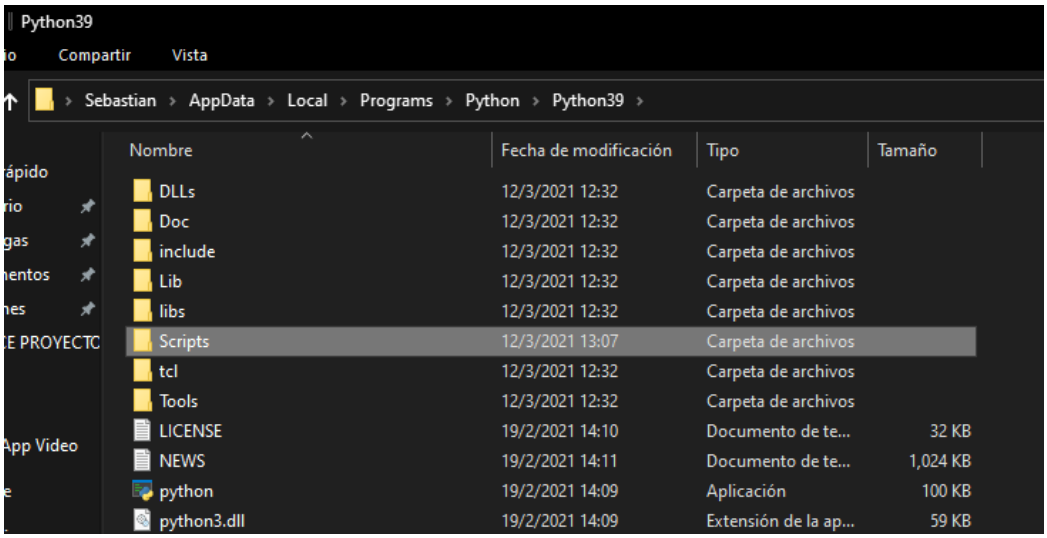


Figura 5. Carpeta principal Python 3.9

- Ya dentro de la carpeta “Scripts”, se selecciona y se copia la ruta de dirección de esta y luego pegarla en el símbolo de sistema (cmd) antecedida del comando “cd”, el cual, cambia el directorio a la carpeta de la ruta seleccionada, como se muestra en la Figura 6 y la Figura 7, respectivamente.

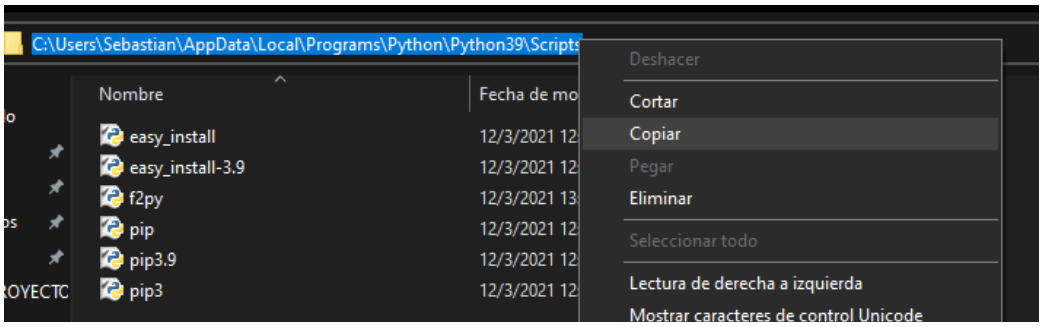


Figura 6. Copia ruta dirección carpeta “Scripts”

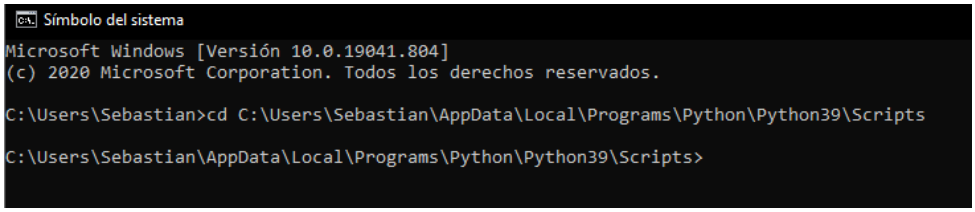


Figura 7. Ruta instalación librerías con el símbolo de sistema

Se procede a la instalación de cada una de las librerías que se usaran en el proyecto, las cuales son OpenCv, Openpyxl, matplotlib y xlwt. La instalación de estas librerías se realiza mediante las rutas otorgadas por el índice de paquetes de Python "PyPI", el cual es un repositorio de software para el lenguaje de programación Python.

- OpenCV

  Se realiza la búsqueda del paquete contenedor para enlaces de Python OpenCV, ubicado en la dirección web https://pypi.org/project/opencv-python/, desde donde se debe copiar el comando otorgado para la descarga e instalación mediante el módulo "pip" y el símbolo de sistema de Windows como se muestra en la Figura 8.

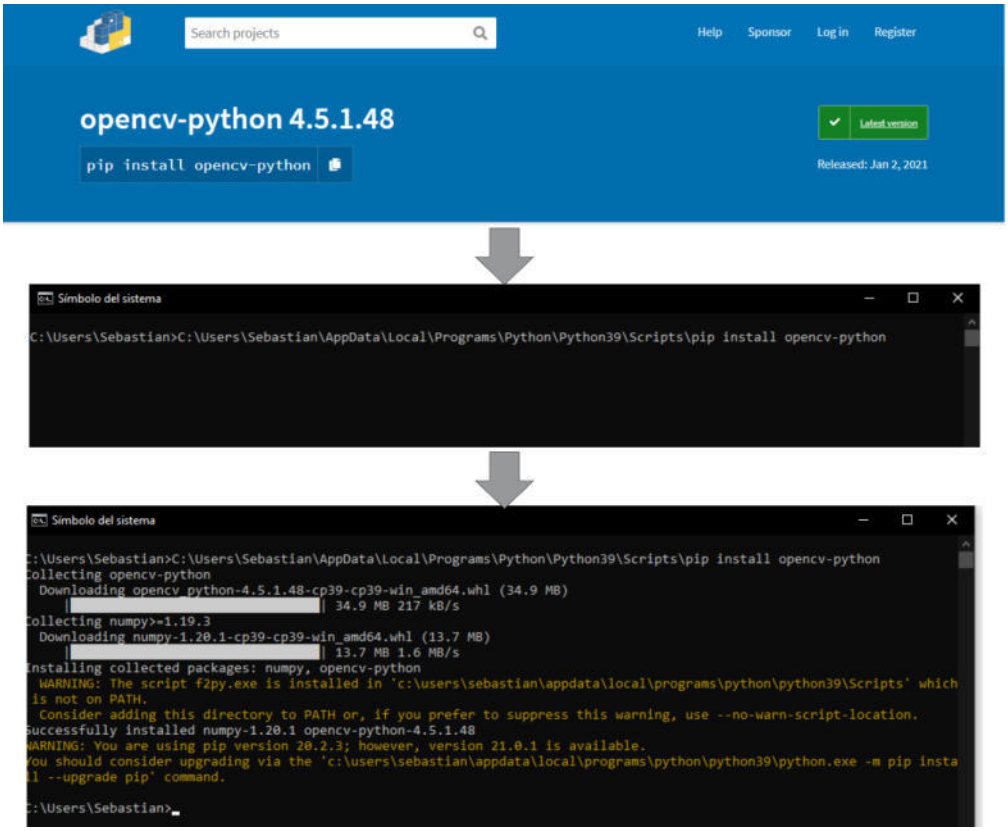


Figura 8. Instalación librería OpenCV

- Openpyxl.

  Se realiza la búsqueda del paquete contenedor para enlaces de Python Openpyxl, ubicado en la dirección web https://pypi.org/project/openpyxl /, desde donde se debe copiar el comando otorgado para la descarga e instalación mediante el módulo "pip" y el símbolo de sistema de Windows como se muestra en la Figura 9.

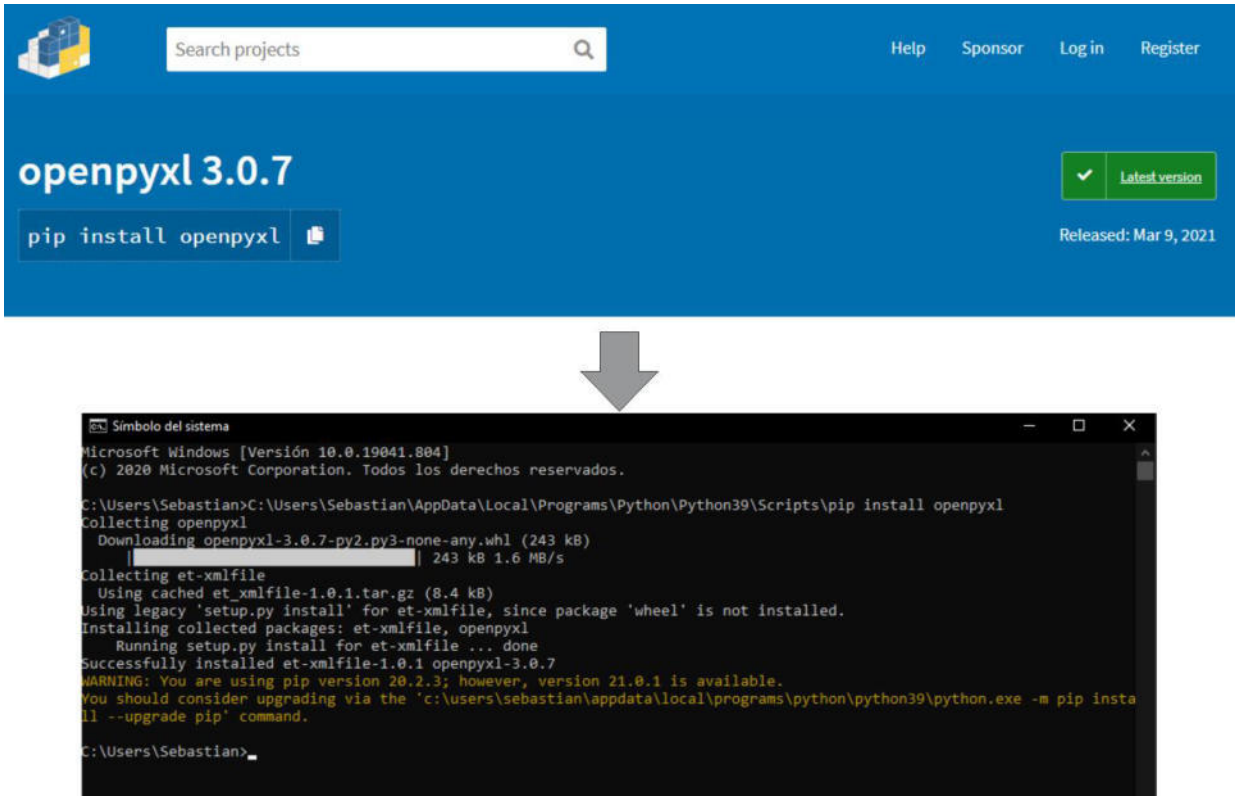


Figura 9. Instalación librería Openpyxl

- Matplotlib

  Se realiza la búsqueda del paquete contenedor para enlaces de Python Matplotlib, ubicado en la dirección web https://pypi.org/project/ Matplotlib/, desde donde se debe copiar el comando otorgado para la descarga e instalación mediante el módulo "pip" el símbolo de sistema de Windows como se muestra en la Figura 10.

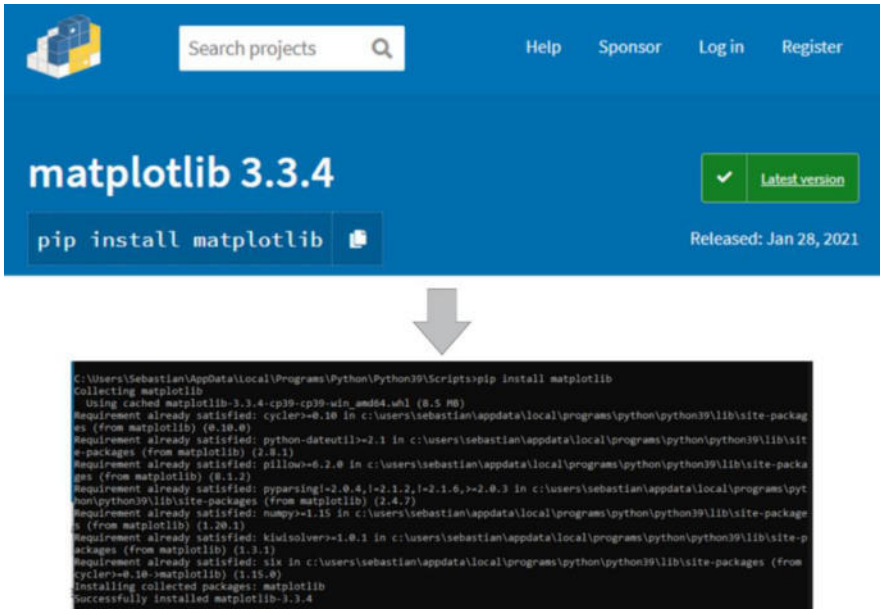


Figura 10. Instalación librería Matplotlib

8.3 Entorno Python 3.9 (IDLE)

Python 3.9 es un entorno grafico para desarrollo elemental que permite editar y ejecutar códigos escritos en el lenguaje de programación Python, a su vez, presenta un entorno interactivo para ejecutar instrucciones sueltas del mismo. El IDLE (Integrated DeveLopment Environment for Python) está diseñado para uso en diferentes plataformas, en caso de Windows, viene incluido en la instalación de Python3.9, caso contrario en Linux, en el que es una aplicación separada que se puede instalar desde los repositorios de cada distribución.

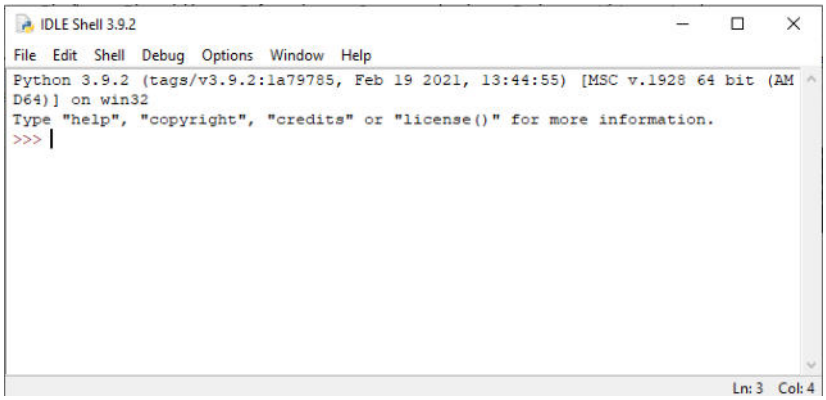


Figura 11. IDLE, Python 3.9 entorno interactivo

Luego se procede a crear el archivo en el cual se realizará la escritura del algoritmo, creando así el primer proyecto en el proceso de familiarización con el entorno.

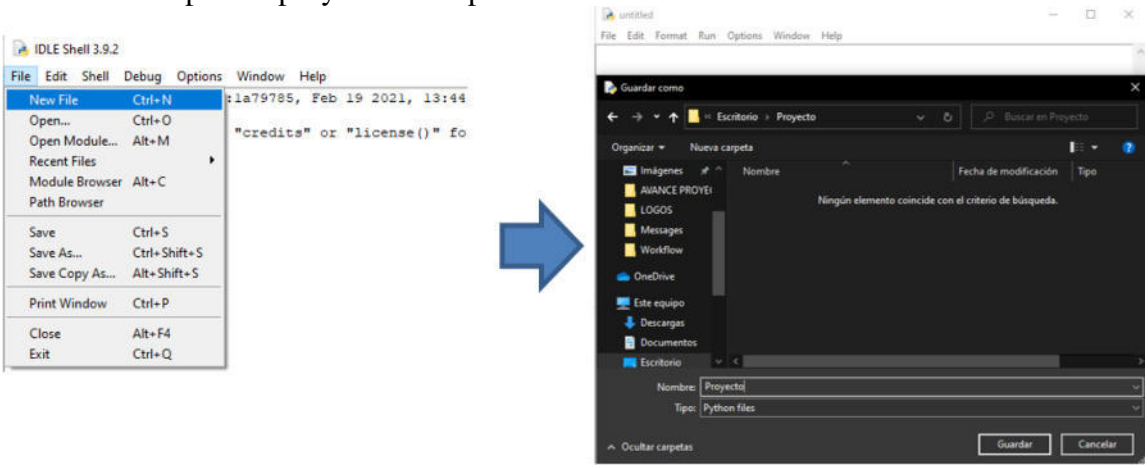


Figura 12. creación archivo IDLE Python3.9

Importación y manejo librería OpenCV

Se realiza la importación de la librería OpenCV para el uso de sus módulos y funciones para el procesamiento de imágenes digitales por medio del comando "import cv2". Inicialmente se realizan pruebas cambiando entre diferentes modelos de color como RGB, HSV, escala de grises, etc. El algoritmo para dicha transformación de modelos de color se muestra en la Figura 13. Donde inicialmente se realiza la importación de la librería cv2 (Línea 2), para luego cargar la imagen a convertir, verificando que no hay errores en el proceso (Línea 5-8). Por medio del comando "cv2.cvtColor", indicando la variable que guarda la imagen RGB y la función que indica la conversión que se desea hacer (BGR2HSV, BGR2GRAY) para convertir a HSV y escala de grises respectivamente.

```python
#Importacion libreria OpenCV
import cv2

#Abrimos la imagen
imagen = cv2.imread('IMAGEN_DE_PRUEBA_1.jpg')
if(imagen is None):
    print("Error: no se ha podido encontrar la imagen")
    quit()


#Conversion imagen a MODELO HSV
IMGHSV = cv2.cvtColor(imagen, cv2.COLOR_BGR2HSV)

#Conversion imagen a ESCALA DE GRISES
IMGGRIS = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)

#Impresion en pantalla imagenes resultado
cv2.imshow('IMAGEN EN RGB', imagen)
cv2.imshow('IMAGEN EN HSV', IMGHSV)
cv2.imshow('IMAGEN ESCALA DE GRISES', IMGGRIS)
```

Figura 13. Algoritmo cambio modelo de color

Finalmente, se muestran en pantalla los resultados obtenidos (Figura 14), (RGB), la imagen de conversión RGB – HSV y la imagen de conversión RGB – Escala de grises.

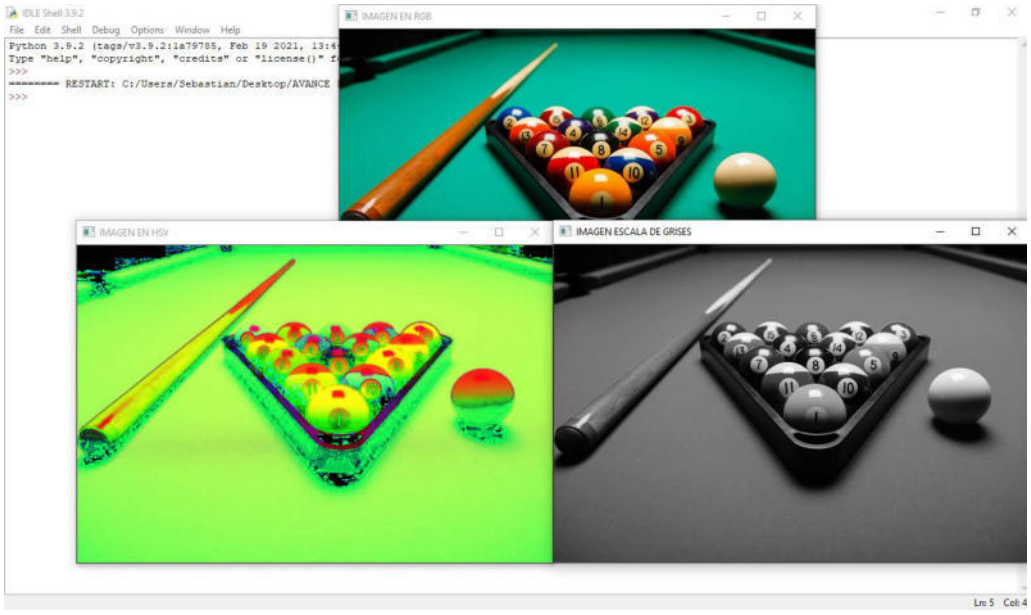


Figura 14. Resultados algoritmo cambio de modelo de color

8.4 Aplicación de técnica de umbralización y segmentación por color en imagen

Una de las tareas principales en el procesamiento de imágenes es la segmentación, el cual permite extraer información relevante de una imagen. Una de las metodoligias empleadas en la segmentación es la umbralización en la cual, se extrae información teniendo en cuenta determinados umbrales aplicados a cada una de las componentes de una imagen en color. En este proyecto se ha empleado una imagen en el espacio de color HSV, en la cual se identifican experimentalmente los diferentes intervalos de umbralización de cada componente empleando deslizadores de acuerdo a las características del objeto a segmentar.

Se realiza la aplicación de la tecnica de umbralización y segmentación, seleccionando la imagen luego de cambiarla al modelo de color HSV, la cual, se usa como base y por medio del uso de deslizadores se modificar los valores mínimos y máximos del rango de las variables H, S y V, de acuerdo al objeto que se desea diferenciar.

Para el proceso de segmentación, se toma como base el codigo del ejemplo anterior (Figura 13), el cual permite cargar una imagen, verificar que fue cargada correctamente y luego convertirla del espacio de color RGB al espacio de color HSV.A continuación se crean las variables que guardarn los datos de los rangos máximos y mínimos de umbralización (Linea 13 – 18) (Figura 15).

```
1  #Importacion libreria OpenCV
2  import cv2
3
4  #Abrimos la imagen
5  imagen = cv2.imread('IMAGEN_DE_PRUEBA_2.jpg')
6  if(imagen is None):
7      print("Error: no se ha podido encontrar la imagen")
8      quit()
9
10 #Conversion imagen a MODELO HSV
11 IMGHSV = cv2.cvtColor(imagen, cv2.COLOR_BGR2HSV)
12
13 min_H = 0
14 min_S = 0
15 min_V = 0
16 max_H = 180
17 max_S = 255
18 max_V = 255
```

Figura 15. Importación librería e imagen y creación de variables

Se continua con la creación de las funciones para guardar y actualizar cada uno de los valores mínimos y máximos de las variables H, S y V (Lineas 20 - 55). El comando "cv2.setTrackbarPos" actualiza y guarda el dato en la variable interna de cada función . Se indica el nombre que tendra la ventana donde estaran los deslizadores con el comando "cv2.namedWindow" y los deslizadores son creados mediante la función el comando "cv2.createTrackbar", al cual se le ingresan los datos (Nombre del deslizador, ventana donde se imprimiran, variable), este proceso de muestra en la Figura 16 .

Con el uso de la instruccion "while" (estructura de control repetitiva), se logra realizar el cambio de los rangos e ir visualizando en tiempo real el resuldato de la umbralización del objeto en cuestion. Por medio del comando "cv2.inRange" en donde se le ingresan los datos (Nombre de la imagen a umbralizar, datos mínimos H,S y V, datos máximos H,S y V), esta nos entrega una imagen binaria, mostrando en color blanco los pixeles que esten dentro del rango ingresado en la función . Por ultimo se muestran en pantalla la imagen original (RGB) y la imagen binaria resultante, adicionando una llave de salida del programa al oprimir las teclas "q" y "Esc" del teclado. A manera de ejemplo, en la Figura 18, se muestra el resultado de umbralizar el color azul.

```
21 def MIN_H(val):
22     global min_H,max_H
23     min_H = val
24     min_H = min(max_H-1, min_H)
25     cv2.setTrackbarPos('min H', "DESLIZADORES", min_H)
26 #Funcion para guardar y actualizar valor maximo variable H
27 def MAX_H(val):
28     global min_H,max_H
29     max_H = val
30     max_H = max(max_H, min_H+1)
31     cv2.setTrackbarPos('max H', "DESLIZADORES", max_H)
32 #Funcion para guardar y actualizar valor minimo variable S
33 def MIN_S(val):
34     global min_S,max_S
35     min_S = val
36     min_S = min(max_S-1, min_S)
37     cv2.setTrackbarPos('min S', "DESLIZADORES", min_S)
38 #Funcion para guardar y actualizar valor maximo variable S
39 def MAX_S(val):
40     global min_S,max_S
41     max_S = val
42     max_S = max(max_S, min_S+1)
43     cv2.setTrackbarPos('max S', "DESLIZADORES", max_S)
44 #Funcion para guardar y actualizar valor minimo variable V
45 def MIN_V(val):
46     global min_V,max_V
47     min_V = val
48     min_V = min(max_V-1, min_V)
49     cv2.setTrackbarPos('min V', "DESLIZADORES", min_V)
50 #Funcion para guardar y actualizar valor maximo variable V
51 def MAX_V(val):
52     global min_V,max_V
53     max_V = val
54     max_V = max(max_V, min_V+1)
55     cv2.setTrackbarPos('max V', "DESLIZADORES", max_V)
56
57 cv2.namedWindow("DESLIZADORES") #Asignacion nombre a la pantalla de los deslizadores
58 cv2.createTrackbar('min H', "DESLIZADORES" , min_H, 180, MIN_H)
59 cv2.createTrackbar('max H', "DESLIZADORES" , max_H, 180, MAX_H)
60 cv2.createTrackbar('min S', "DESLIZADORES" , min_S, 255, MIN_S)
61 cv2.createTrackbar('max S', "DESLIZADORES" , max_S, 255, MAX_S)
62 cv2.createTrackbar('min V', "DESLIZADORES" , min_V, 255, MIN_V)
63 cv2.createTrackbar('max V', "DESLIZADORES" , max_V, 255, MAX_V)
```

Figura 16. Creación de deslizadores y funciones para actualización de variables

```
while True:

    imagen_Range = cv2.inRange(IMGHSV, (min_H, min_S, min_V), (max_H, max_S, max_V))
    cv2.imshow('IMAGEN EN RGB', imagen)
    cv2.imshow('IMAGEN BINARIA RESULTANTE', imagen_Range)

    key = cv2.waitKey(30)
    if key == ord('q') or key == 27:
        break
```

Figura 17 Instrucción while, comando inRange e impresión de imágenes

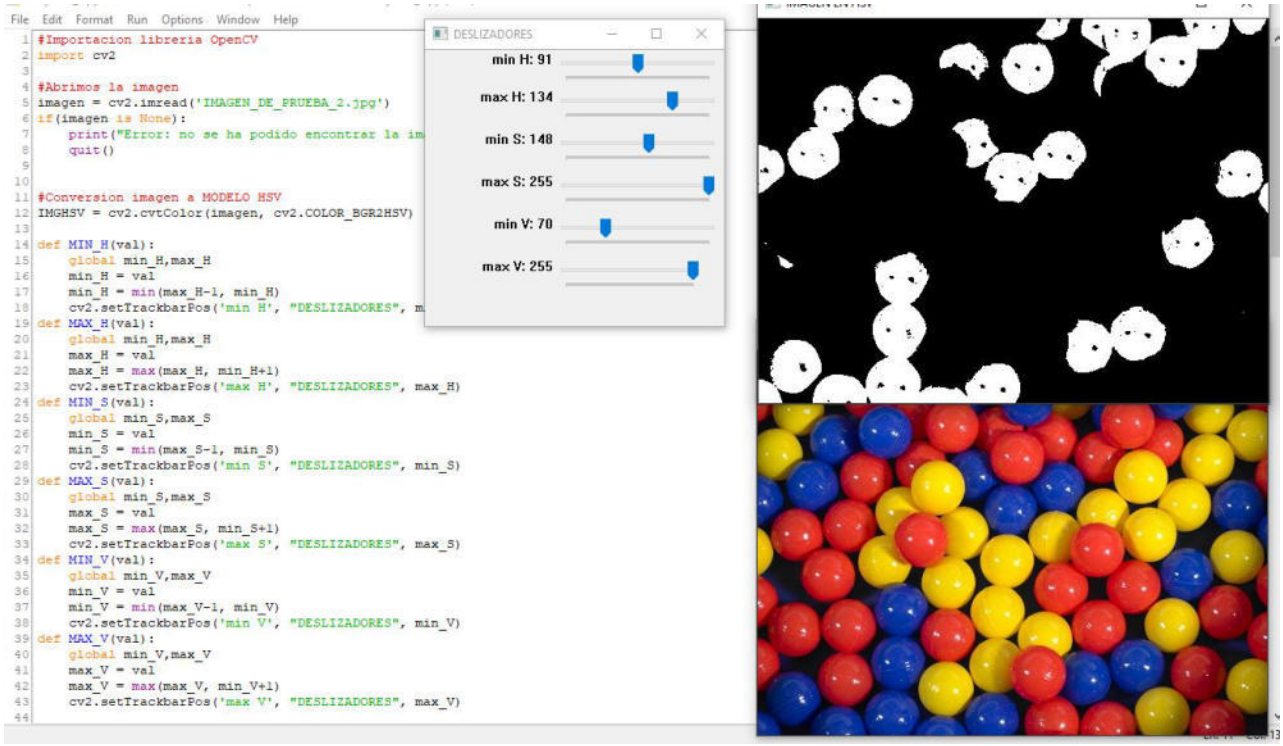


Figura 18. Resultados umbralización figuras de color azul

8.5 Aplicación de técnica de umbralización y segmentación por color en video online.

Con base en el código del ejemplo anterior de la umbralización en una imagen, se realiza el cambio aplicándolo en una secuencia de imágenes obtenidas de las capturas tomadas on-line por medio de la cámara web del computador.
Con el comando “VideoCapture(0)”, siendo el número entre los paréntesis el índice que especifica el uso de la cámara web, se realiza la captura de las imágenes. Luego, por medio de un ciclo “while”, se genera la lectura de dichas capturas mediante la instrucción

"cap.read()", donde "cap" es la dirección donde se guardó el comando de captura de las imágenes y a su vez, permite realizar la umbralización online de la secuencia de imágenes, así como se muestra en las Figuras 19 y 20, el resultado al segmentar los objetos de color azul y rojo respectivamente.

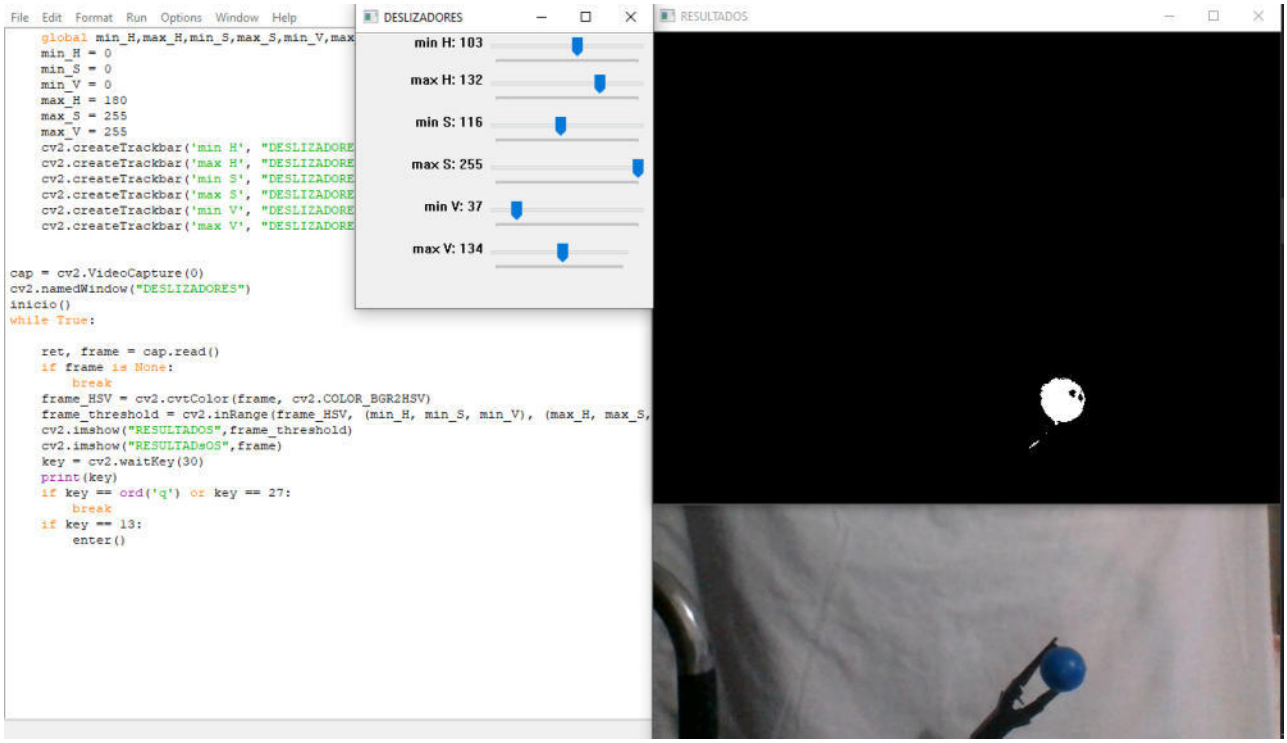


Figura 19. Umbralización online objeto color azul

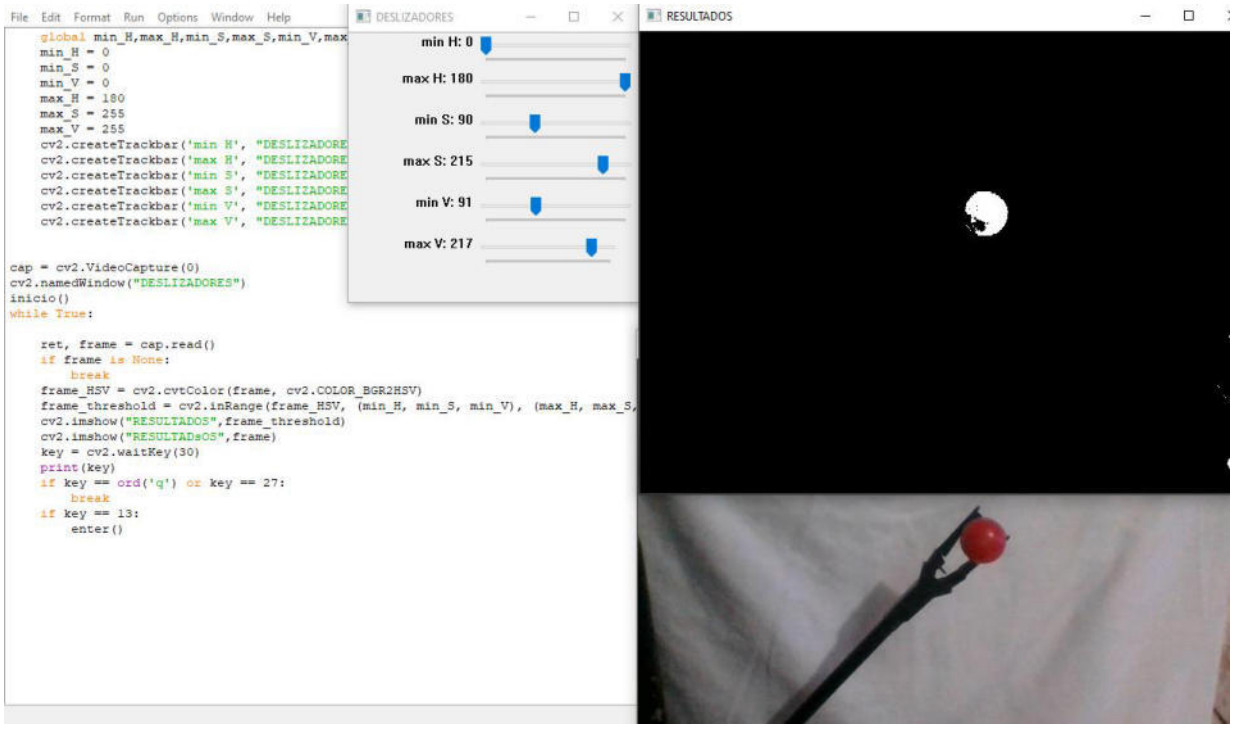


Figura 20. Umbralización online objeto color rojo

8.6 Detección de bordes, contornos y centroides a objetos

En la detección de los bordes de los objetos en cuestión, se hace necesario el uso del algoritmo de umbralización anterior, con el cual se toma la información de los umbrales de color para uno de los objetos (en este caso la esfera de color azul), para las respectivas pruebas.

Luego de esto, se comprueba que la segmentación del objeto sea de forma correcta y se revisa visualmente corriendo el algoritmo (Figura 21). Al revisar detenidamente los resultados, se encuentra que la imagen binaria resultante presenta ruido (Figura 22), el cual, con el paso del tiempo se puede convertir en un inconveniente en el algoritmo final.

En este caso mediante herramientas de transformaciones morfológicas las cuales son operaciones simples basadas en la forma de la imagen, que normalmente se aplican en imágenes binarias. Dos operadores morfológicos básicos son dilatación y erosión, teniendo estas formas variantes como apertura, cierre, gradiente, etc. Estas transformaciones morfológicas están incluidas como métodos presentes en la librería OpenCV, con uso del comando “morphologyEx”, el cual solicita los datos de imagen, tipo de transformación a realizar, elemento estructurante o núcleo (Kernel).

El tipo de transformación seleccionado en este caso fue de tipo “Apertura”, el cual realiza una operación de erosión seguida de dilatación, donde el proceso de erosionado se realiza a través de la tarea de deslizar el kernel por la imagen donde un píxel de la imagen original (1 o 0) sólo se considerará 1 si todos los píxeles que caen dentro de la ventana del kernel son 1, de lo contrario se erosiona (se hace a cero). Por tanto, todos los bordes de los objetos de la imagen serán descartados, esto directamente relacionado al tamaño del kernel, el cual para el algoritmo se definió como una matriz de unos de tamaño 3x3. A continuación, el comando realiza el proceso de dilatación, el cual no es más que el contrario de la erosión, así definido en caso de un píxel estar dentro de la ventana del kernel, es tomado como un 1.

```
import cv2
import numpy as np

#UMBRAL HSV VALORES MINIMOS Y MAXIMOS COLOR AZUL
min_H = 97
min_S = 157
min_V = 56
max_H = 113
max_S = 255
max_V = 255

#TOMA DE CAPTURAS CAMARA WEB
cap = cv2.VideoCapture(0)


while True:

    #LECTURA CAPTURAS CAMARA WEB
    ret, frame = cap.read()

    #CONFIRMACION DE DATO NO NULO
    if frame is None:
        break

    #CONVERSION IMAGEN DE RGB A HSV
    frame_HSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    #APLICACION IN RANGE PARA LA SEGMENTACION
    frame_Range = cv2.inRange(frame_HSV, (min_H, min_S, min_V), (max_H, max_S, max_V))

    #IMPRESION EN PANTALLA IMAGEN ORIGINAL
    cv2.imshow("VIDEO_ONLINE_ORIGINAL",frame)

    #IMPRESION EN PANTALLA IMAGEN RESULTANTE
    cv2.imshow("VIDEO_ONLINE_BINARIA_SEGMENTADA",frame_Range)
```

Figura 21. Algoritmo segmentación de objeto color azul

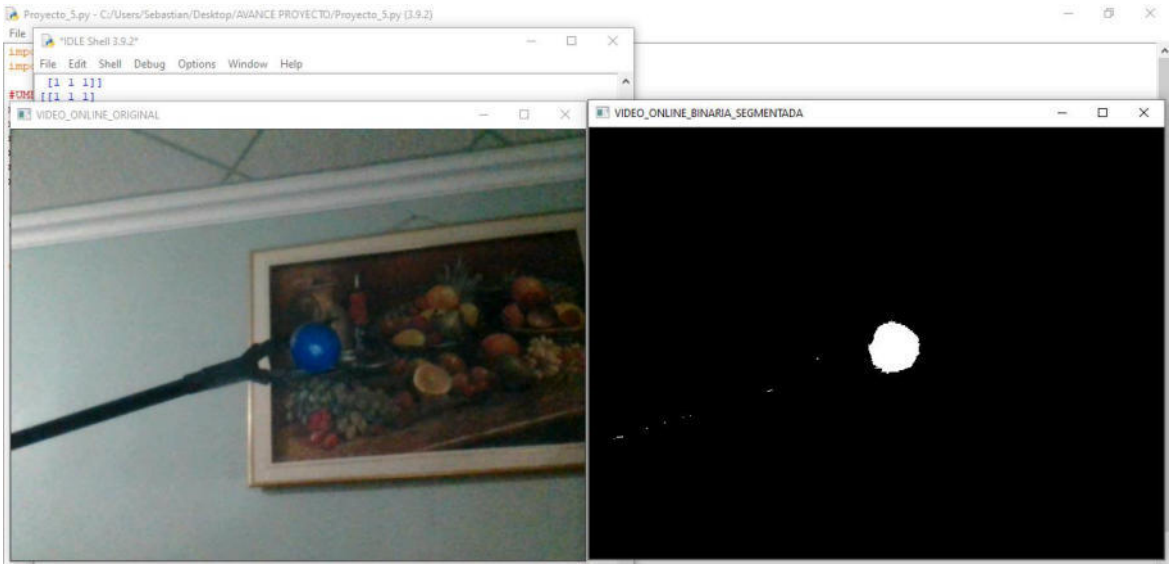


Figura 22. Imagen real y binaria resultante después de segmentación

Luego se crea la variable “kernel”, en la cual se le asigna una matriz de unos de tamaño 3 x 3, luego mediante el comando “cv2.morphologyEx (frame_Range, cv2.MORPH_OPEN, kernel)”, se asigna dentro de él, los datos de imagen binaria resultante de la segmentación, el tipo de transformación y la matriz kernel (Figura 23). El proceso de operación morfológica se realiza dos veces para obtener una imagen que elimine los objetos no deseados del fondo.

En la Figura 24, se muestra los resultados, donde en pantalla se encuentran la imagen original (superior izquierda), la imagen binaria resultante después de los procesos morfologicos (superior derecha), la imagen resultante del proceso de apertura(inferior izquierda) y la imagen resultante del proceso de cierre (inferior derecha).

```
#CONVERSION IMAGEN DE RGB A HSV
frame_HSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

#APLICACION IN RANGE PARA LA SEGMENTACION
frame_Range = cv2.inRange(frame_HSV, (min_H, min_S, min_V), (max_H, max_S, max_V))

#FILTRADO DE IMAGEN
#ELEMENTO ESTRUCTURANTE O NUCLEO (KERNEL)
kernel = np.ones((3,3),np.uint8)

#APLICACION APERTURA (VARIACION MORFOLOGICA)
Filtro_Apertura=cv2.morphologyEx(frame_Range,cv2.MORPH_OPEN,kernel)

#APLICACION APERTURA DE NUEVO (VARIACION MORFOLOGICA)
Filtro_Apertura_II=cv2.morphologyEx(Filtro_Apertura,cv2.MORPH_OPEN,kernel)

#IMPRESION EN PANTALLA IMAGEN ORIGINAL
cv2.imshow("VIDEO_ONLINE_ORIGINAL",frame)

#IMPRESION EN PANTALLA IMAGEN RESULTANTE
cv2.imshow("VIDEO_ONLINE_BINARIA_SEGMENTADA",frame_Range)

#IMPRESION EN PANTALLA IMAGEN FILTRO APERTURA
cv2.imshow("VIDEO_ONLINE_FILTRO_APERTURA",Filtro_Apertura)

#IMPRESION EN PANTALLA IMAGEN FILTRO CIERRE
cv2.imshow("VIDEO_ONLINE_FILTRO_APERTURA_II",Filtro_Apertura_II)
```

Figura 23. Algoritmo aplicación de filtrado de la imagen

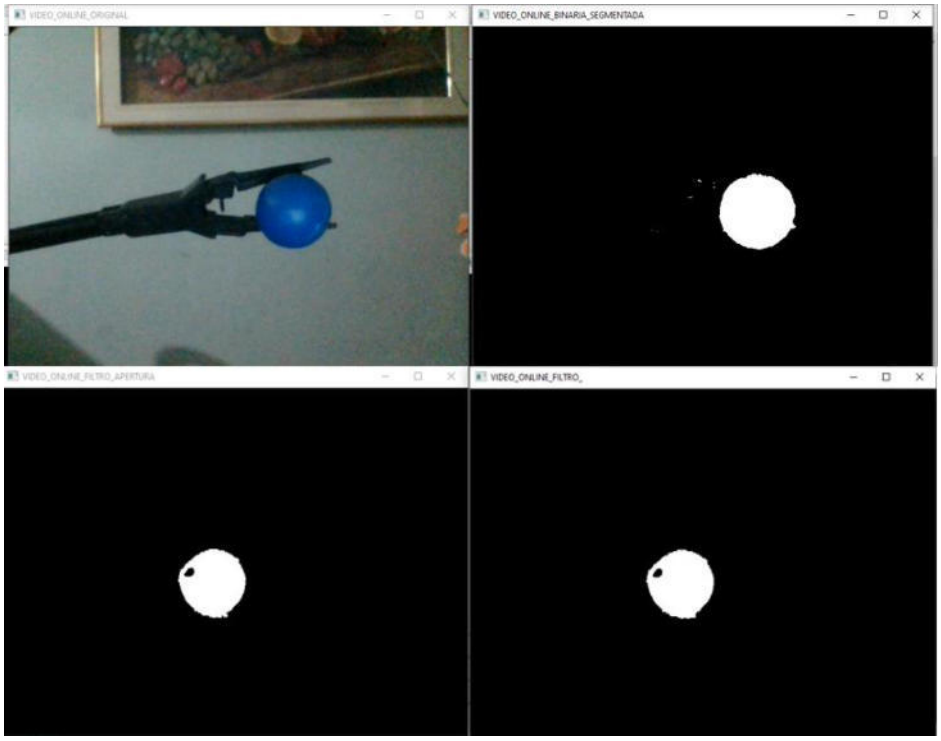


Figura 24. Resultados después de proceso de transformaciones morfológicas

- Detección Contornos y Centroide

Un contorno se define como una curva que une todos los puntos continuos a lo largo de un límite dependiente del mismo color o intensidad. Para encontrar los contornos, se toma como base la imagen resultante del filtrado, en la cual con el uso la función "findContours", donde solicita los datos de imagen a tratar, modo de recuperación de contorno y un método de aproximación (Este último permite escoger si se desea puntos quiebres o todos los puntos del contorno). La función devuelve, la imagen, los contornos y la jerarquía. El dato relevante en este caso, son los contornos, los cuales son devueltos como una matriz con datos de las coordenadas de los contornos. Dicha matriz contiene los datos de los contornos de todos los objetos encontrados de acuerdo con las especificaciones dadas. Puesto que el fondo no fue controlado al 100%, se obtienen resultados de varios objetos de tamaño mínimo por efectos de brillo, reflejos, etc. Por tal razón, es necesario evaluar la información y tomar los contornos del objeto de mayor tamaño . Al tener la información en la matriz, se realiza un recorrido dentro de ella, en el cual se ejecuta una comparación en la longitud de los contornos obtenidos, para así tomar el objeto con la mayor longitud del contorno, el cual será el del objeto en cuestión. El algoritmo resultante de este paso se muestra en la Figura 25.

```
#ELEMENTO ESTRUCTURANTE O NUCLEO (KERNEL)
kernel = np.ones((3,3),np.uint8)

#APLICACION APERTURA (VARIACION MORFOLOGICA)
Filtro_Apertura=cv2.morphologyEx(frame_Range,cv2.MORPH_OPEN,kernel)

#APLICACION APERTURA DE NUEVO (VARIACION MORFOLOGICA)
Filtro_Apertura_II=cv2.morphologyEx(Filtro_Apertura,cv2.MORPH_OPEN,kernel)

#APLICACION DEFICION DE CONTORNOS
Contorno , _ = cv2.findContours(Filtro_Apertura_II , cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)

#COMPARACION PARA TOMAR SOLO DATO DEL CONTORNO MAYOR
Comparador = 0
Contorno_Mayor = []
for j in range(len(Contorno)):
    if len(Contorno[j]) > Comparador:
        Comparador = len(Contorno[j])
        Contorno_Mayor = Contorno[j]
Contorno = Contorno_Mayor
```

Figura 25. Algoritmo detección de contornos y selección contorno mayor

Para el centroide, se detecta mediante el uso de los momentos de la imagen, con el comando "moments" que permite calcular algunas características como el centro de masa del objeto, el área del objeto, etc. Dicho comando, devuelve un diccionario de todos los valores de los momentos calculados, con estos datos se encuentra el valor de los centroides que están definidos por:

$$cx = \frac{M_{10}}{M_{00}} \; y \; cy = \frac{M_{01}}{M_{00}}$$

En la Figura 26, se realiza el cálculo de la ecuación anterior en el algoritmo en Python, iniciando con un condicional "if", con el cual se asegura de tener los datos del contorno del objeto para evitar errores en la compilación del código.
Al final, se dibuja el centroide del objeto con la función "cv2.circle" seleccionando la imagen, coordenadas del centroide, tamaño, color y el contorno del objeto con la función "cv2.drawContours" seleccionando la imagen original, matriz de contornos, "-1" (todos los datos de contornos), color y tamaño. En la Figura 27, se muestra en pantalla el resultado del cálculo de los contornos y centroide.

```
#CALCULO DEL CENTROIDE A PARTIR DE LOS MOMENTOS
if Comparador != 0:
    Momento = cv2.moments(Contorno_Mayor)
    cx = int(Momento['m10']/Momento['m00'])
    cy = int(Momento['m01']/Momento['m00'])
    print(cx,cy)
    #DIBUJO DEL CENTROIDE
    cv2.circle(frame , (cx , cy) , 5 , (0,200,255) , -1)


#DIBUJO DE CONTORNOS EN LA IMAGEN
cv2.drawContours(frame, Contorno, -1, (255,255,0), 3)
```

Figura 26. Algoritmo cálculo de centroides del objeto

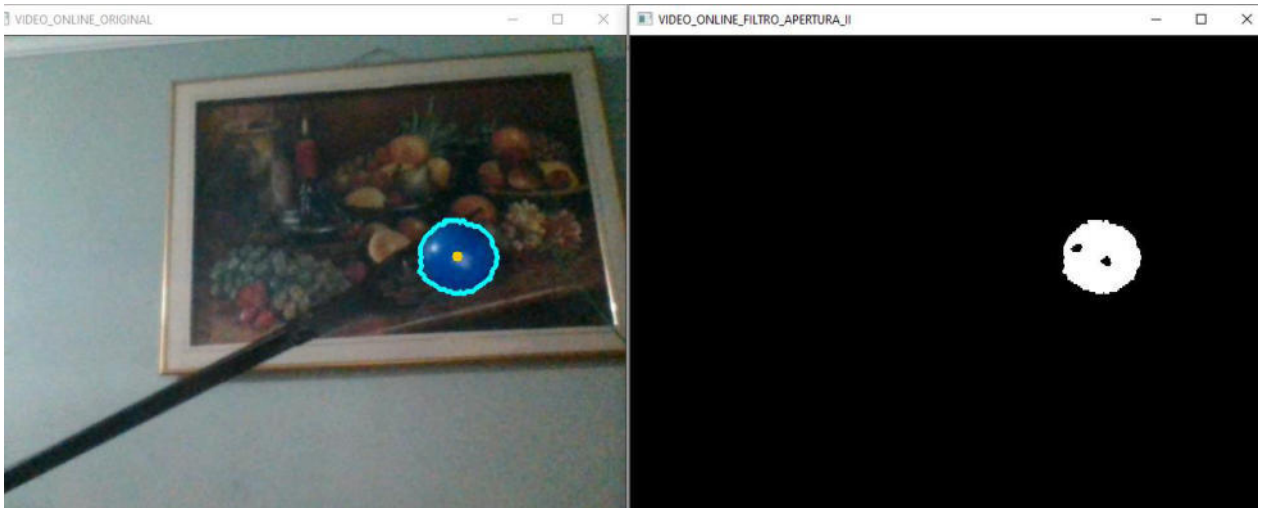


Figura 27. Imagen resultada del cálculo y dibujo contorno y centroide

Para la medida y predicción de la posición de un objeto en el eje z, se realiza la toma de datos a diferentes distancias desde la cámara al objeto y por medio de ecuaciones de recta entre los puntos tomados se realiza una aproximación de la coordenada en el eje z del objeto. La figura 28, muestra la tabla de los datos tomados para un objeto de diámetro 4 cm.

PUNTOS TOMADOS PRUEBAS EN OBJETO DE DIAMETRO 4 CM		
PUNTO1 (DIAMETRO EN PIXELES (d), DISTANCIA (cm) ENTRE CAMARA Y OBJETO(z))	PUNTO2 (DIAMETRO EN PIXELES (d), DISTANCIA (cm) ENTRE CAMARA Y OBJETO(z))	Ecuacion de la recta entre los dos puntos
(300 , 4)	(120 , 20)	z = (-0.1 * d) + 32
(120 , 20)	(80 , 30)	z = (-0.25 * d) + 50
(80 , 30)	(58 , 40)	z = ((-5/11) * d) + (730/11)
(58 , 40)	(34 , 60)	z = ((-5/6) * d) + (265/3)
(34 , 60)	(25 , 70)	z = ((-10/9)* d) + (880/9)
(25 , 70)	(10 , 80)	z = (-2 * d) + 120

Figura 28. Tabla de datos recolectados diámetros, distancia y ecuación de la recta objetos de 4cm de diámetro.

Al relacionar los datos tomados con los objetos de diámetros 4cm (Figura 28) y 8cm (Figura 29), las ecuaciones para el objeto de diámetro 4 cm, pueden usarse para los objetos de diferentes medidas, multiplicando el resultado de las ecuaciones por un coeficiente para cada una de ellas determinado por $c = T/4$, donde T es el diámetro en centímetros de la figura a relación y el número "4" es el diámetro de cuatro centímetros del objeto con el cual se hallaron ecuaciones de la recta.

PUNTOS TOMADOS PRUEBAS EN OBJETO DE DIAMETRO 8 CM	
PUNTO1 (DIAMETRO EN PIXELES (d), DISTANCIA (cm) ENTRE CAMARA Y OBJETO(z))	PUNTO2 (DIAMETRO EN PIXELES (d), DISTANCIA (cm) ENTRE CAMARA Y OBJETO(z))
(242 , 20)	(163 , 30)
(163 , 30)	(126 , 40)
(126 , 40)	(75 , 60)
(75 , 60)	(53 , 70)
(53 , 70)	(25 , 80)
(25 , 80)	(12 , 90)

Figura 29. Tabla de datos recolectados diámetros, distancia y ecuación de la recta objetos de 8cm de diámetro.

8.7 Implementación filtro de Kalman

En la implementación del filtro de Kalman, se utilizó la clase "KalmanFilter" incluida en la librería OpenCV, la cual implementa un filtro Kalman estándar, que permite modificar "transitionMatrix", "controlMatrix" y "measurementMatrix". Al realizar dicha modificación, se obtiene la funcionalidad de un filtro de Kalman extendido.
Las matrices se definieron como:

$$measurementMatrix = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad (5)$$

$$transitionMatrix = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6)$$

$$processNoiseCov = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7)$$

Se crea un objeto donde se declara el filtro de Kalman, por medio de la función "cv2.KalmanFilter()" incluida en la clase "Kalman filter", encargada de crear el objeto en la función encargada de recibir los datos de las medidas y luego proceder a realizar la predicción. Dentro del paréntesis de la función "KalmanFilter()", se especifica como se ve en la segunda línea de la figura 30 los valores "(4, 2)", donde el número 4 indica el número de estados, incluyendo (x, y, dx, dy) posición y velocidad a lo largo del eje x y el eje y; el número 2, especifica los valores de las coordenadas que se pueden visualizar. Luego se declaran las matrices "measurementMatrix", "transitionMatrix", "processNoiseCov" con los valores presentados en las ecuaciones (5), (6) y (7) respectivamente.

```
#DEFINICION FILTRO DE KALMAN
kalman = cv2.KalmanFilter(4,2)
#MATRIX DE MEDIDA
kalman.measurementMatrix = np.array([[1,0,0,0],[0,1,0,0]],np.float32)
#MATRIX DE TRANSICION
kalman.transitionMatrix = np.array([[1,0,1,0],[0,1,0,1],
                                    [0,0,1,0],[0,0,0,1]],np.float32)
#MATRIX DE RUIDO PROCESO
kalman.processNoiseCov = np.array([[1,0,0,0],[0,1,0,0],
                                   [0,0,1,0],[0,0,0,1]],np.float32) * 0.03
```

Figura 30. Definición filtro de Kalman

Luego como se observa en la Figura 31, en un arreglo 2x11 llamado “medición” se agregan los valores obtenidos de las coordenadas del centroide (cx ,cy) del objeto en cada frame de la prueba. El arreglo “medición ” es el valor que se le entrega a la función “kalman.correct()”, la cual realiza una actualización del estado previsto de la medición y a continuación se ejecuta la función “kalman.predict()” encargada de realizar la predicción de la posición (x,y) futura del objeto; esta devuelve los valores de las coordenadas de la predicción realizada por el filtro de Kalman. Luego la posición (x,y) obtenida y la posición(x,y) resultante de la predicción, se dibujan en pantalla mediante un círculo de tamaño 5 pixeles, de color amarillo y fucsia respectivamente .Luego con el dato de la medida en z, se procede a aplicar el Filtro de Kalman para la predicción de la posición futura de la misma, así como se hizo en los ejes (x, y), se ejecuta en el eje (x, z), se crean las variables necesarias y se ejecuta la predicción, se guarda los resultados en un nuevo arreglo “prediccionz”.

```
#MATRIX DE MEDIDA EJES X,Y
    medicion = np.array([[np.float32(cx),
                          np.float32(cy)]])
    kalman.correct(medicion)
#PREDICCION DE POSICION FUTURA EJES X,Y
    Prediccion = kalman.predict()
#DIBUJO DE LA PREDICCION EJES X,Y
    cv2.circle(frame,(Prediccion[0] ,Prediccion[1]) , 5 , (200,0,255) , -1)
 #MATRIX DE MEDIDA EJES X,Z
    medicionz = np.array([[np.float32(cx),
                          np.float32(cz)]])
    kalmanz.correct(medicionz)
#PREDICCION DE POSICION FUTURA EJES X,Z
    Prediccionz = kalmanz.predict()
```

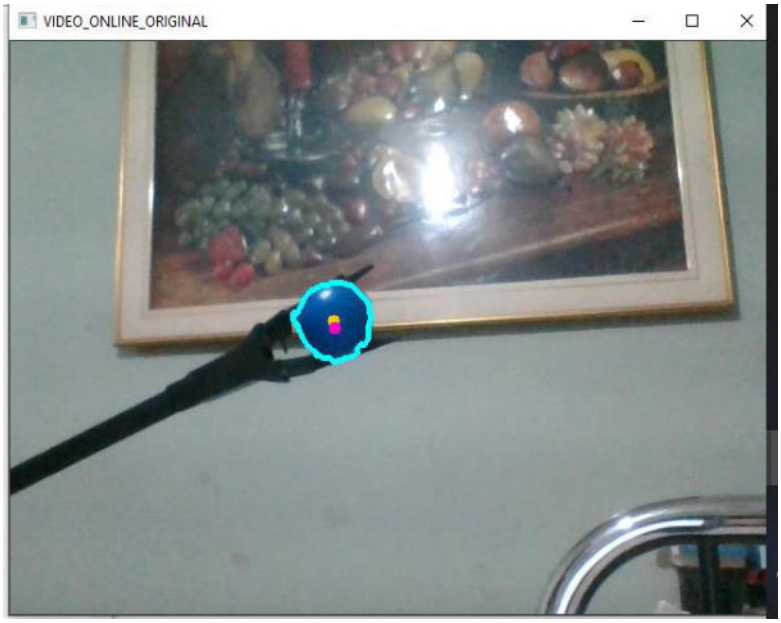


Figura 31. Calculo y dibujo de la predicción del filtro de Kalman plano (x,y).

8.8 Representación gráfica de resultados

La representación gráfica de los resultados de cada proceso de predicción se realiza mediante el uso de la librería matplotlib (Figura 32). Esta librería permite graficar en 2d y 3d sobre planos coordenados definidos por las entradas que se le entreguen.

```
import matplotlib.pyplot as plt
```

Figura 32. Importe librería matplotlib

Los datos tomados de las pruebas para graficarlos se guardan en arreglos o vectores (centroides y predicciones de los centroides), los cuales son declarados inicialmente como vacíos (Figura 33) para luego ir concatenando la información obtenida. De igual manera, se crea un contador para llevar un control en la cantidad de datos tomados y así poder graficar los resultados de las medidas y predicciones.

```
#VARIABLES Y VECTORES VACIOS PARA GRAFICAR
medidasx = []
medidasy = []
prediccionx = []
predicciony = []
contador = 0
tamaño = []
```

Figura 33. Declaración de variables y vectores vacíos

En la Figura 34, se observa un contador el cual va aumentando cada vez que un nuevo dato se agregara en los vectores de medidas y predicción.

```
#DATOS PARA GRAFICAR RESULTADOS
contador = contador + 1
tamaño.append(contador)
medidasx.append(cx)
medidasy.append(cy)
prediccionx.append(int(Prediccion[0]))
predicciony.append(int(Prediccion[1]))
```

Figura 34. Almacenamiento de datos en variables para graficar

Posteriormente, se toman como información importante para la comparacion en las graficas los datos de medición y predicción tomados en el eje x y el eje y, asi mismo, se grafica el movimiento del objeto en dichos ejes. Para esto en necesario antes de graficar, señalar la figura en la que se graficara, esto se realiza por medio del comando “plt.figure()” donde entre parentesis se ingresa el número de la figura donde se desea imprimir la información. Luego mediante el comando “plt.plot()” se realiza la impresión en pantalla. Este comando solicita los datos que contendra el eje x y eje y y el nombre del grupo de puntos que se graficaran. Finalmente, el comanto “plt.show” crea las ventanas con las respectivas figuras que se ha graficado.

```
plt.figure(0)
plt.plot(tamaño,medidasx,label="medidax")
plt.plot(tamaño,prediccionx,label="prediccionx")
plt.legend()
plt.figure(1)
plt.plot(tamaño,medidasy,label="mediday")
plt.plot(tamaño,predicciony,label="predicciony")
plt.legend()
plt.figure(2)
plt.plot(medidasx,medidasy,label="medida")
plt.plot(prediccionx,predicciony,label="prediccion")
plt.legend()
plt.show()
```

Figura 35. Método de dibujo de graficas con matplotlib

En la Figura 36, se muestran tres pruebas realizadas para comprobar el funcionamiento de la toma de datos y del dibujo de las gráficas, donde se tomaron dos movimientos geométricos (circulo y rombo) y también un movimiento en "U". De izquierda a derecha están las figuras del movimiento, medida en x comparada con la predicción del filtro de Kalman en x y medida en y comparada con la predicción del filtro de Kalman en y.
Los datos ingresados en las dos primeras pruebas fueron entradas predeterminadas con posición es fijas que al recorrerlas logran interpretar la forma del movimiento como las figuras geométricas circulo y rombo, para así poder analizar el funcionamiento del filtro de Kalman con información sin tantas perturbaciones. La tercera prueba se realizó con captura online, donde se intentó recrear un movimiento en U.

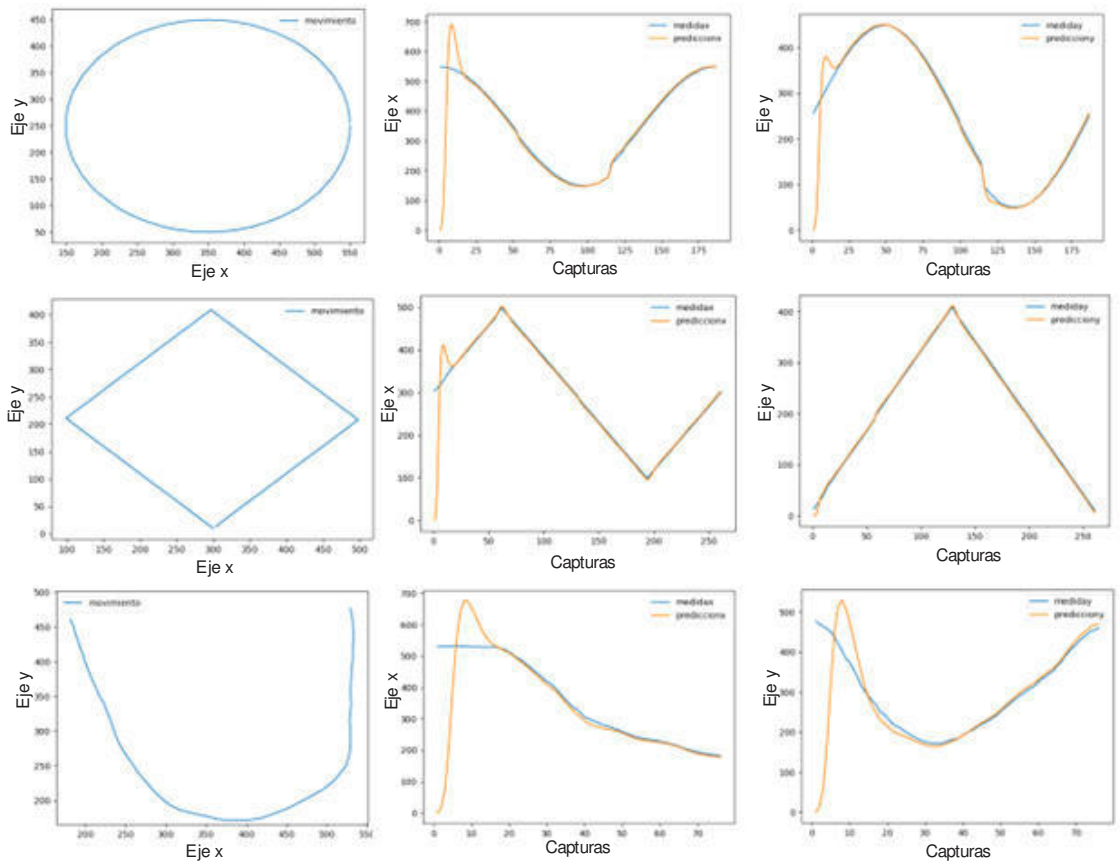


Figura 36. Graficas de medición y predicción para tres trayectorias

8.9 Manejo de interfaz gráfica para usuario

En la Figura 37, se presenta el diagrama general que manejara la interfaz gráfica para el usuario final del programa:

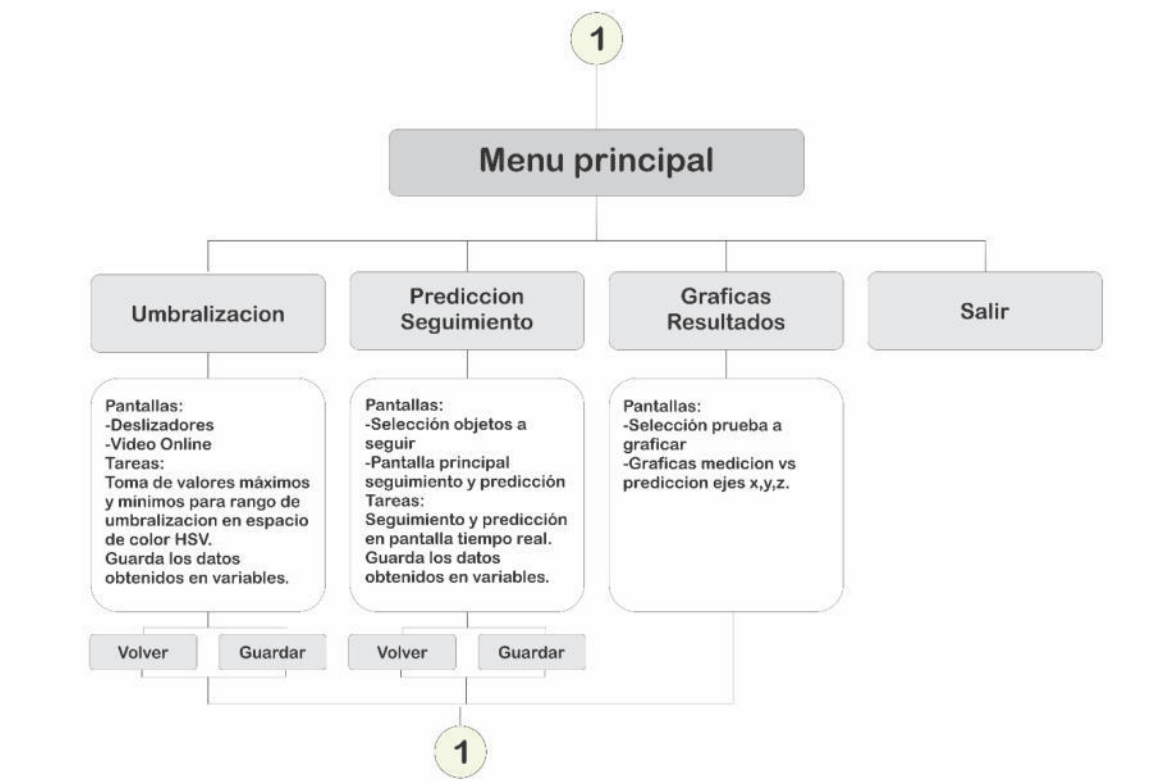


Figura 37. Diagrama general interfaz grafica

Para la construcción de la interfaz grafica, se uso la librería tkinter, ya incluida en la instalación de Python. Esta librería permite la creación de ventanas con operadores sencillos como cuadros de texto, botones, listas desplegables, botones de chequeo, etc.

Inicialmente se creo una función llamada "menu_inicial()", la cual contiene cuatro botones los cuales estaran enlazados a diferentes funciones que se crearan en base al codigo desarrollo previamente (umbralización, predicción y seguimiento, graficas resultados). Para la creación de la pantalla se asigna a una variable el comando "Vmenu = Tk()" y para evitar el cambio de tamaño de la ventana se uso el comando "Vmenu.resizable(0,0)", luego con en el comando "Vmenu.geometry()" se configura el tamaño y posición en la pantalla de la ventana creada, y por ultimo asignamos un titulo a la ventana con el comando "Vmenu.title()", esto se muestra en la figura 38.

```
def menu_inicial():
    Vmenu=Tk()
    Vmenu.resizable(0,0)
    Vmenu.geometry("+%d+50" % ((Vmenu.winfo_screenwidth()-350)/2))
    Vmenu.title("Menu Principal")
```

Figura 38. Creación y configuración ventana Tkinter

Para la creación del menú se usaron cuatro botones llamados: UMBRALIZACIÓN, PREDICCIÓN – SEGUIMIENTO, GRAFICAS RESULTADOS Y SALIR, en orden descendente y a cada uno de ellos se les asignaron las funciones A1, A2, A3 Y A4 respectivamente, como se muestra en la figura 39 y 40. A su vez, se configuro color de texto azul (fg = "blue"), tipo de fuente (Font = ("Yu Gothic", 20, "bold")), justificación centrada (anchor = "center"), y en tamaño un ancho de 20 y un alto de 2 (width = 20, height = 2). Para la ubicación en la ventana, se usó el estilo de ubicación ".grid", el cual particiona la pantalla en una especie de matriz por filas y columnas (row, column).

```
btn1=Button(Vmenu, command=A1,  fg="blue",  text="UMBRALIZACION",
            font=("Yu Gothic",20,"bold") ,  anchor="center",
            width=20,height=2).grid(row=0,column=0)
btn2=Button(Vmenu, command=A2,  fg="blue",  text="PREDICCION \nSEGUIMIENTO",
            font=("Yu Gothic",20,"bold"),anchor="center",
            width=20,height=2).grid(row=1,column=0)
btn3=Button(Vmenu, command=A3,  fg="blue",  text="GRAFICAS \nRESULTADOS",
            font=("Yu Gothic",20,"bold"),anchor="center",
            width=20,height=2).grid(row=2,column=0)
btn4=Button(Vmenu, command=salir, fg="blue",text="SALIR",
            font=("Yu Gothic",20,"bold"),anchor="center",
            width=20,height=2).grid(row=4,column=0)
Vmenu.mainloop()
```

```
def A1():
    Vmenu.destroy()
    umbral()
def A2():
    Vmenu.destroy()
    prediccion()
def A3():
    Vmenu.destroy()
    grafica()
def salir():
    global a
    a = 1
    Vmenu.destroy()
```

Figura 39. Creación botones y sus respectivas funciones en ventana menú

Como se observa en la Figura 39, cada función contiene el comando “Vmenu.destroy()”, el cual se encarga de destruir la ventana principal del menú y luego ejecutar una nueva tarea.
La ventana resultante de este proceso se muestra en la Figura 40.

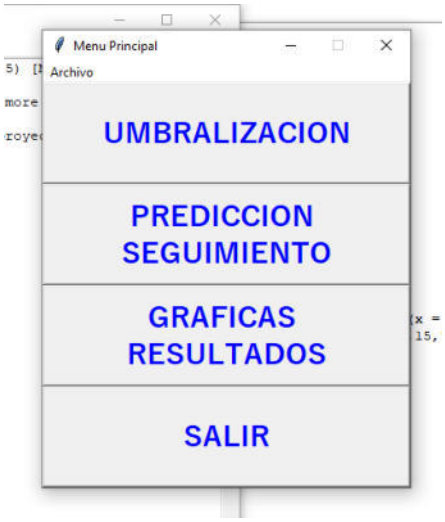


Figura 40. Ventana Menú principal

Siguiendo con el desarrollo de la interfaz, se procede a la creación de las sub ventanas de cada botón, iniciando por el botón “UMBRALIZACIÓN”, el cual como se observa en la Figura 39, llama la función umbral() donde se incluye el código descrito en las secciones 11.5 y 11.6, donde al terminar la umbralización y al oprimir la tecla “ESC”, se toman los valores que se hayan ajustado en la umbralización, se realiza el llamado de la función guardar_umbral() y se destruye las ventanas creadas por la librería OpenCV con el comando “cv2.destroyAllWindows()” (Figura 41).

```
if key == ord('q') or key == 27:
    guardar_umbral(min_H, min_S, min_V, max_H, max_S, max_V)
    cv2.destroyAllWindows()
    break
```

Figura 41. Función para guardar umbrales

La función guardar_umbral(), solicita como entradas los valores de la umbralización en HSV mínimos y máximos. Se realiza la configuración de la pantalla, entre ellos el título de “Ventana guardar”. Para guardar los umbrales es necesario el manejo de documentos planos, en este caso el nombre del archivo es “Objetos.csv” guardado en la ruta 'Data/Objetos/Objetos.csv', el cual con el comando “Open” fue abierto en modo lectura “r”

para conocer la cantidad de objetos guardados existentes después de umbralizarlos en el documento. Con el comando “Label” de Tkinter, se crean en la ventana los textos que indicaran información relevante antes de guardar los datos, siendo ubicados en la ventana con el comando “.place()” por medio de una ubicación (x, y) en pixeles. (Figura 42)

```
def guardar_umbral(Hm,Sm,Vm,HM,SM,VM):

    #ESPECIFICACIONES INICIALES VENTANA
    Vguardar=Tk()
    Vguardar.resizable(0,0)
    Vguardar.geometry("270x330+%d+50" % ((Vguardar.winfo_screenwidth()-350)/2))
    Vguardar.title("Ventana guardar")

    #LECTURA DEL DOCUMENTO DONDE ESTA GUARDADO LOS DATOS DE LOS OBJETOS
    archivo = open('Data/Objetos/Objetos.csv','r')

    #VARIABLES PARA CONTAR OBJETOS Y GUARDAR INFORMACION
    objetos = 0
    lista = []

    #LECTURA POR LINEA DE LA INFORMACION EN EL DOCUMENTO
    while True:
        hoja = archivo.readline()
        if not hoja:
            break
        datos = list(hoja.split(','))
        lista.append(datos)
        objetos = objetos + 1
    archivo.close()

    #TEXTO EN PANTALLA
    Lab1=Label(Vguardar,text="Desea guardar el objeto # "+str(objetos+1),
               fg="blue",  font=("Yu Gothic",12,"bold"),
               anchor="center").place(x = 25,y = 20)
    Lab2=Label(Vguardar,text="H minimo "+str(Hm), fg="blue",
               font=("Yu Gothic",12,"bold"),anchor="center").place(x = 25,y = 50)
    Lab3=Label(Vguardar,text="H maximo "+str(HM), fg="blue",
               font=("Yu Gothic",12,"bold"),anchor="center").place(x = 150,y = 50)
    Lab4=Label(Vguardar,text="S minimo "+str(Sm), fg="blue",
               font=("Yu Gothic",12,"bold"),anchor="center").place(x = 25,y = 80)
    Lab5=Label(Vguardar,text="S maximo "+str(SM), fg="blue",
               font=("Yu Gothic",12,"bold"),anchor="center").place(x = 150,y = 80)
    Lab6=Label(Vguardar,text="V minimo "+str(Vm), fg="blue",
               font=("Yu Gothic",12,"bold"),anchor="center").place(x = 25,y = 110)
    Lab7=Label(Vguardar,text="V maximo "+str(VM), fg="blue",
               font=("Yu Gothic",12,"bold"),anchor="center").place(x = 150,y = 110)
    Lab8=Label(Vguardar,text="Color "           , fg="blue",
               font=("Yu Gothic",12,"bold"),anchor="center").place(x = 25,y = 140)
```

Figura 42. Función guardar_umbral parte 1

En la Figura 43 se muestra, la definicion de la función salida(), que se encarga de destruir la ventana y volver al menu principal, esto al oprimir el boton “Volver”. Adicionalmente, se crean dos cuadros de texto, en lo que se solicita información del color y diametro de la figura que se umbralizó. Al final los botones “Volver” y “Guardar”, permiten omitir la umbralización y guardar la información llamando la función guarda() respectivamente (Figura 44).

```
#FUNCION PARA BOTON "SALIDA"
def salida():

    #DESTRUCCION VENTANA
    Vguardar.destroy()

#VARIABLES PARA CUADROS DE TEXTO
color = StringVar()
diametro = StringVar()

#CUADROS DE TEXTO PARA PANTALLA
Ent1=Entry(Vguardar,textvariable=color).place(x = 25,y = 170)
Ent2=Entry(Vguardar,textvariable=diametro).place(x = 25,y = 230)

#BOTONES PARA PANTALLA
btn1=Button(Vguardar, command=guarda,  fg="blue",  text="Guardar",
            font=("Yu Gothic",12,"bold") ,anchor="center",
            width=10,height=1).place(x = 140, y = 270)
btn2=Button(Vguardar, command=salida,  fg="blue",  text="Volver",
            font=("Yu Gothic",12,"bold") ,anchor="center",
            width=10,height=1).place(x = 20, y = 270)

#DESTRUCCION VENTANA
Vguardar.mainloop()
```

Figura 43. Función guardar_umbral parte 2

La función “guarda()” inicialmente toma los valores ingresados en los cuadros de texto de la ventana “Ventana guardar” y los guarda en las variables “colore” y “diametroe” para luego agregar la información en la lista de los valores tomados del documento “Objetos.csv”. Para guardar los datos en dicho documento, se abre en modo de escritura, por medio de un ciclo for se escribe con el comando “archivo.write()” el cual lo realiza línea por línea recorriendo la lista. Al terminar se cierra la lista y se destruye la ventana.

```
#FUNCION PARA BOTON "GUARDAR"
def guarda():

    #VARIABLE PARA GUARDAR DATO DE LOS CUADROS DE TE
    colore = color.get()
    diametroe = diametro.get()

    #AGREGARNDO INFORMACION NUEVA AL VECTOR
    numero = len(lista)
    lista.append([])
    lista[numero].append(str(Hm))
    lista[numero].append(str(Sm))
    lista[numero].append(str(Vm))
    lista[numero].append(str(HM))
    lista[numero].append(str(SM))
    lista[numero].append(str(VM))
    lista[numero].append(str(colore))
    lista[numero].append(str(diametroe))
    lista[numero].append(str("objeto"+str(objetos+1)))
    lista[numero].append(str(""))

    #ARCHIVO ABIERTO EN MODO DE ESCRITURA
    archivo = open('Data/Objetos/Objetos.csv','w')

    #ESCRITURA EN ARCHIVO
    for i in lista:
        archivo.write(str(i[0]))
        archivo.write(",")
        archivo.write(str(i[1]))
        archivo.write(",")
        archivo.write(str(i[2]))
        archivo.write(",")
        archivo.write(str(i[3]))
        archivo.write(",")
        archivo.write(str(i[4]))
        archivo.write(",")
        archivo.write(str(i[5]))
        archivo.write(",")
        archivo.write(str(i[6]))
        archivo.write(",")
        archivo.write(str(i[7]))
        archivo.write(",")
        archivo.write(str(i[8]))
        archivo.write(",")
        archivo.write(str(i[9]))
        archivo.write(",")
        archivo.write("\n")
    archivo.close()

    #DESTRUCCION VENTANA
    Vguardar.destroy()
```

Figura 44. Función ventana guardar umbral

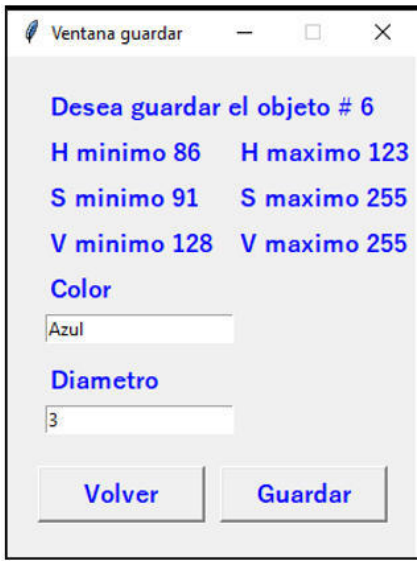


Figura 45. Ventana guardar umbralización.

En la opción “PREDICCIÓN SEGUIMIENTO” del Menú Principal (Figura 46), se incluyen los códigos descritos en las secciones 11.6 y 11.7, en los cuales se realiza la detección, seguimiento y predicción de la posición de un objeto mediante la aplicación del Filtro de Kalman.

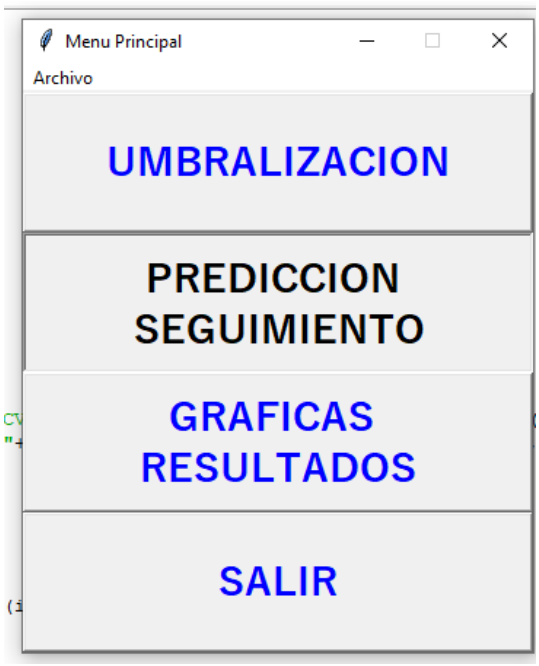


Figura 46. Selección botón PREDICCIÓN / SEGUIMIENTO

Al ingresar a dicha opcion, se crea una subventana, en la cual se muestra información de los objetos umbralizados guardados en el archivo “Objetos.csv”, junto con un “Checkbutton”, con el que se da la opcion al usuario de escoger las figuras que se desean seguir en la prueba de predicción y seguimiento que se realizara.
Esta subventana incluye tambien dos botones “Volver” y “Seleccionar”, donde el primero tiene como función volver al menu principal y el otro, ingresa a la función “selec()” para continuar con el proceso. Las Figuras 47 y 48, muestran el codigo que se describio anteriormente y la Figura 49 representa el resultado de la ventana que se muestra en pantalla al usuario.

```
def prediccion():
    global medidasx,medidasy,prediccionx,predicciony,tamaño

    Vprediccion=Tk()
    Vprediccion.resizable(0,0)
    Vprediccion.geometry("450x500+%d+50" % ((Vprediccion.winfo_screenwidth()-350)/2))
    Vprediccion.title("Ventana prediccion")
    archivo = open('Data/Objetos/Objetos.csv','r')
    objetos = 0
    lista = []
    while True:
        hoja = archivo.readline()
        if not hoja:
            break
        datos = list(hoja.split(','))
        lista.append(datos)
        objetos = objetos + 1
    archivo.close()
    posicion = 30
    for i in range(len(lista)):
        globals()["CV"+str(i)] = IntVar()
        Chel = Checkbutton(Vprediccion, variable = globals()["CV"+str(i)],
                           onvalue = 1, offvalue = 0).place(x = 400, y = 75 + posicion)
        Labl=Label(Vprediccion,text=lista[i][8] + "  color :  "+
                   lista[i][6]+ "  diametro :  "+ lista[i][7], fg="blue",
                   font=("Yu Gothic",15,"bold"),
                   anchor="center").place(x = 10, y = 70 + posicion)
        posicion = posicion + 30
        total_check = i+1
```

Figura 47. Función predicción, ventana predicción parte 1.

```
def volver():
    Vprediccion.destroy()
Labl=Label(Vprediccion,text="Seleccione los objetos \na seguir ", fg="blue",
           font=("Yu Gothic",14,"bold"),anchor="center").place(x=120,y=10)

btn1=Button(Vprediccion, command=select,  fg="blue",  text="Seleccionar",
            font=("Yu Gothic",10,"bold") ,anchor="center",
            width=10,height=1).place(x = 260,y = 420)
btn2=Button(Vprediccion, command=volver,  fg="blue",  text="Volver",
            font=("Yu Gothic",10,"bold") ,anchor="center",
            width=10,height=1).place(x = 130,y = 420)
Vprediccion.mainloop()
```

Figura 48. Función predicción, ventana predicción parte 2.

Figura 49. Ventana predicción.

El botón "Seleccionar" está enlazado con la función "select()", en la cual, se utiliza el código representado en la sección 11.7 adicionando el uso del método "globals()", el cual devuelve el diccionario de la tabla de símbolos global actual, es decir, se puede usar para modificar variables asignando valores a cada una de ellas con un nombre especifico indicado como un string. Esto permite ahorrar líneas de códigos en variables con prefijos y un número consecutivo, por ejemplo, Var1, Var2, Var3, …, VarN. Por medio de un ciclo (for o while), se puede asignar valores diferentes a la cantidad de variables que se desee. Como se puede apreciar en la Figura 50, se agregó un ciclo for donde el rango es dependiente a la cantidad de objetos seleccionados en la "Ventana predicción" y se crean las variables necesarias para el futuro seguimiento.

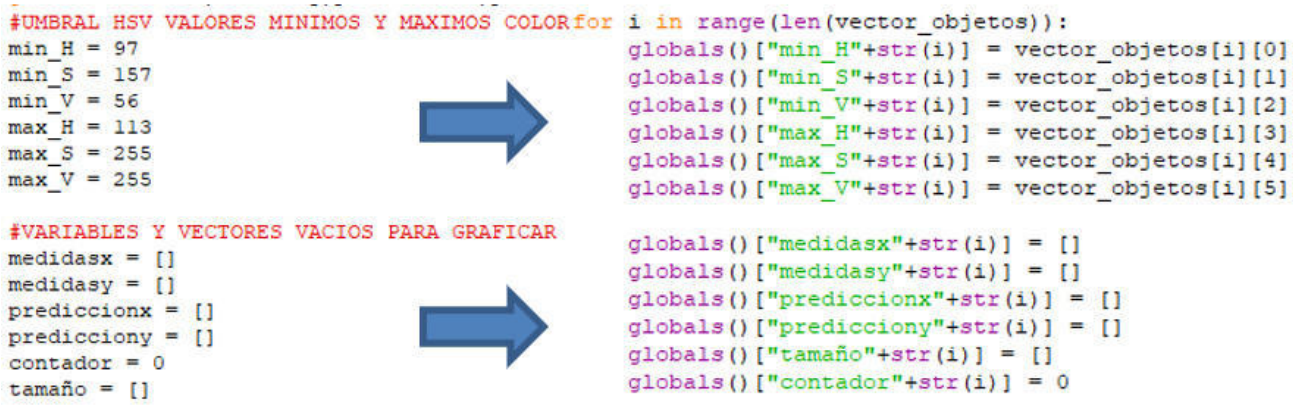


Figura 50. Implementación método globals()

Asi permite realizar el seguimiento y predicción de varios objetos al tiempo (sujeto a la velocidad del procesador del computador). El usuario decide mediante entrada del teclado el momento de terminar con el proceso, oprimiento la tecla “ESC”, el algoritmo entra en el condicional que destruye las pantallas creadas por la librería OpenCV y llama la función “guardar_ prediccion()” ingresando como entrada el dato “vector_objetos” que contiene la información de los objetos a los que se les realizo el seguimiento y predicción.

```
def guardar_prediccion(vector):

    #BUSQUEDA LISTADO DE ARCHIVOS DEL DIRECTORIO DE LAS PRUEBAS
    contenido = os.listdir('Data/Documentos')

    #ESPECIFICACIONES INICIALES VENTANA
    Vguardar=Tk()
    Vguardar.resizable(0,0)
    Vguardar.geometry("270x150+300+50")
    Vguardar.title("Ventana guardar")

    #FUNCION PARA BOTON "VOLVER"
    def salida():

        #DESTRUCCION VENTANA
        Vguardar.destroy()

    #TEXTO EN PANTALLA
    Lab1=Label(Vguardar,text="Desea guardar la Prueba # "+str(len(contenido)+1), fg="blue",
               font=("Yu Gothic",12,"bold"),anchor="center").place(x=25,y=20)

    #BOTONES PARA PANTALLA
    btn1=Button(Vguardar, command=guarda,  fg="blue",  text="Guardar",font=("Yu Gothic",12,"bold") ,
                anchor="center",width=10,height=1).place(x = 140, y = 70)
    btn2=Button(Vguardar, command=salida,  fg="blue",  text="Volver",font=("Yu Gothic",12,"bold") ,
                anchor="center",width=10,height=1).place(x = 20, y = 70)

    Vguardar.mainloop()
```

Figura 51. Función guardar_prediccion

La función "guardar_ prediccion()" (Figura 51), inicialmente crea una ventana la cual pregunta al usuario si desea guardar la prueba y le otorga a la misma un número para su identificacion. Contiene dos botones "Volver" y "Guardar", el primero permite volver al menu principal ignorando los datos obtenidos en la prueba realizada, y el segundo, llama la función "guarda()"(Figura 52). Esta función crea un archivo donde el nombre se asigna "Prueba" + el número de la prueba, guardando el mismo en la dirección "Data/ Documentos/" y luego escribe los datos obtenidos en el proceso de deteccion y seguimiento de los objetos seleccionados. Por medio de los bloques "try" y "except", al presentarse un error, aparece en pantalla una ventana de advertencia, donde se especifica el error al guardar y en caso contrario, se muestra en la misma que el archivo se guardo correctamente (Figura 53).

```
def guarda():
    try:
        #CREACION DEL DOCUMENTO DONDE QUEDARA GUARDADO LOS DATOS DE LA PRUEBA
        archivo = open('Data/Documentos/Prueba'+str(len(contenido)+1)+'.csv','w')

        #ESCRITURA EN EL DOCUMENTO CSV
        for j in range(len(vector)):
            for i in range(len(globals()["tamaño"+str(j)])):
                archivo.write(str(globals()["medidasx"+str(j)][i]))
                archivo.write(',')
            archivo.write(str(vector[j][6])+'\n')
            for i in range(len(globals()["tamaño"+str(j)])):
                archivo.write(str(globals()["medidasy"+str(j)][i]))
                archivo.write(',')
            archivo.write(str(vector[j][6])+'\n')
            for i in range(len(globals()["tamaño"+str(j)])):
                archivo.write(str(globals()["prediccionx"+str(j)][i]))
                archivo.write(',')
            archivo.write(str(vector[j][6])+'\n')
            for i in range(len(globals()["tamaño"+str(j)])):
                archivo.write(str(globals()["predicciony"+str(j)][i]))
                archivo.write(',')
            archivo.write(str(vector[j][6])+'\n')
            for i in range(len(globals()["tamaño"+str(j)])):
                archivo.write(str(globals()["tamaño"+str(j)][i]))
                archivo.write(',')
            archivo.write(str(vector[j][6])+'\n')
        archivo.close()

        #VENTANA ADVERTENCIA DE GUARDADO EXITOSO
        messagebox.showinfo("Advertencia","Guardo correctamente")

        #DESTRUCCION VENTANA
        Vguardar.destroy()
    except:

        #VENTANA ADVERTENCIA DE ERROR AL GUARDAR
        messagebox.showinfo("Advertencia","Error al guardar")

        #DESTRUCCION VENTANA
        Vguardar.destroy()
```

Figura 52. Función de creación de archivo para guardado de datos obtenidos

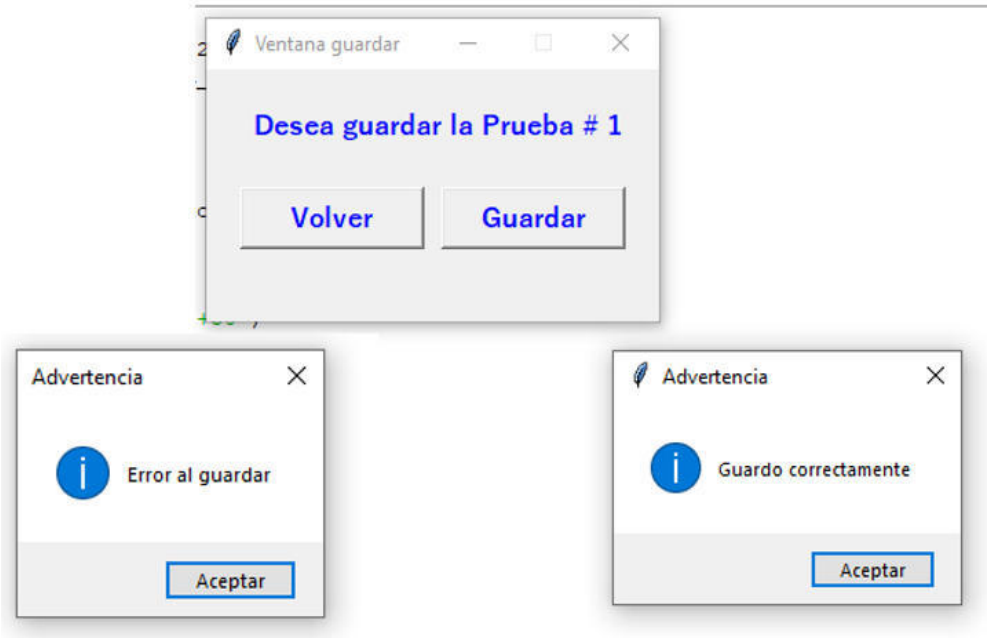


Figura 53. Ventanas de advertencia al guardar

En la opcion “GRAFICAS RESULTADOS” del Menu Principal (Figura 54), se incluyen los codigos descritos en la seccion 11.8, en los cuales se realiza la representacion grafica de los resultados obtenidos en pruebas anteriores.

Figura 54. Selección botón GRAFICAS RESULTADOS

Al ingresar en dicha opción, se crea una subventana, como se muestra en la Figura 55. Esta opción llama la función "grafica()", en la cual se configura una ventana "Vgrafica", donde se agregan textos informativos, una lista desplegable contenedora de los nombres de las pruebas realizadas y cuatro botones, dos de ellos para volver al menú principal, uno con llamado a la función "abrir()" la cual está encargada de tomar la última prueba realizada y graficarla y el otro, llama la función "select()" encargada de tomar el valor de la lista desplegable que se ha seleccionado y así tomar los datos de la prueba y graficarlas.

```
def grafica():
    contenido = os.listdir('Data/Documentos')
    Vgrafica=Tk()
    Vgrafica.resizable(0,0)
    Vgrafica.geometry("280x300+300+50")
    Vgrafica.title("Menu Principal")

    def volver():
        Vgrafica.destroy()
    seleccion = StringVar()
    lista_desplegable = ttk.Combobox(Vgrafica,width=30,values=contenido)
                                     textvariable = seleccion).place(x

    btn1=Button(Vgrafica, command=abrir,  fg="blue",  text="Abrir",font
                anchor="center",width=10,height=1).place(x = 130,y = 70
    btn2=Button(Vgrafica, command=volver,  fg="blue",  text="Volver",fo
                anchor="center",width=10,height=1).place(x = 30,y = 70)

    Lab1=Label(Vgrafica,text="Ultima Prueba realizada # "+str(len(conte
               font=("Yu Gothic",12,"bold"),anchor="center").place(x=25,
    Lab1=Label(Vgrafica,text="o", fg="blue",  font=("Yu Gothic",14,"bold"),anchor="center").place(x=120,y=110)
    Lab1=Label(Vgrafica,text="Seleccione otra Prueba ", fg="blue",  font=("Yu Gothic",12,"bold")
               ,anchor="center").place(x=25,y=140)

    btn1=Button(Vgrafica, command=select,  fg="blue",  text="Seleccionar",font=("Yu Gothic",10,"bold") ,
                anchor="center",width=10,height=1).place(x = 130,y = 220)
    btn2=Button(Vgrafica, command=volver,  fg="blue",  text="Volver",font=("Yu Gothic",10,"bold") ,
                anchor="center",width=10,height=1).place(x = 30,y = 220)
    Vgrafica.mainloop()
```

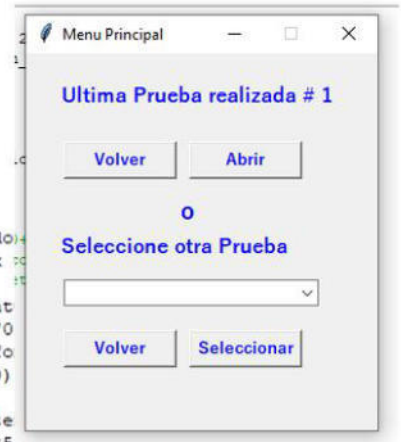
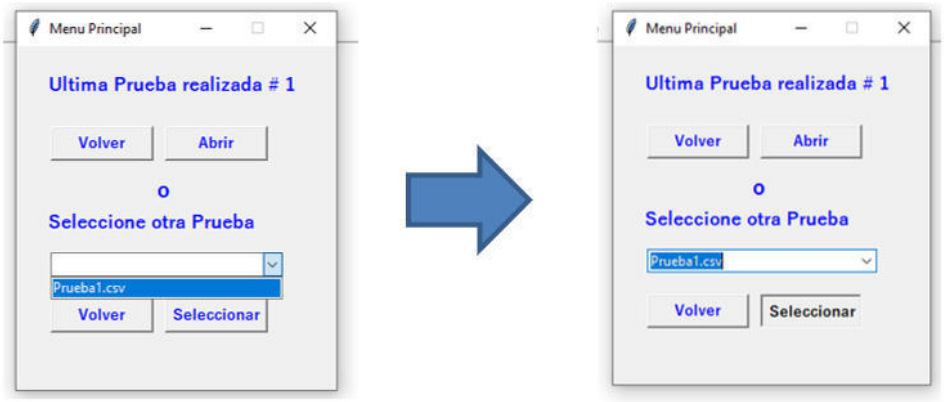


Figura 55. Función grafica() y sub-ventana para graficar

En el proceso para graficar, se toman los datos guardados de la prueba seleccionada, se guardan en listas para poder graficarlos. Por medio de un ciclo for y el metodo globals(), se recorre la información de cada uno de los objetos a los que se le realizo el seguimiento y predicción de su posición en la prueba seleccionada. Como resultado se muestran en pantalla las graficas para comparar medición eje x con predicción del filtro de kalman eje x, medición eje y con predicción del filtro de kalman eje y y movimiento en el plano (x,y). Los resultados obtenidos se muestran en la Figura 56 y Figura 57.




```
for i in range(total_objetos):
    plt.figure("Objeto "+str(globals()["medidasxstr"+str(i)][len(globals()["medidasxstr"+str(i)])-1]))
    plt.subplot(1,3,1)
    plt.plot(globals()["tamaño"+str(i)],globals()["medidasx"+str(i)],label="medidax")
    plt.plot(globals()["tamaño"+str(i)],globals()["prediccionx"+str(i)],label="prediccionx")
    plt.legend()
    plt.subplot(1,3,2)
    plt.plot(globals()["tamaño"+str(i)],globals()["medidasy"+str(i)],label="mediday")
    plt.plot(globals()["tamaño"+str(i)],globals()["predicciony"+str(i)],label="predicciony")
    plt.legend()
    plt.subplot(1,3,3)
    plt.plot(globals()["medidasx"+str(i)],globals()["medidasy"+str(i)],label="medida")
    plt.plot(globals()["prediccionx"+str(i)],globals()["predicciony"+str(i)],label="prediccion")
    plt.legend()
    plt.show()
```

Figura 56. Selección de prueba listado desplegable y función para graficar

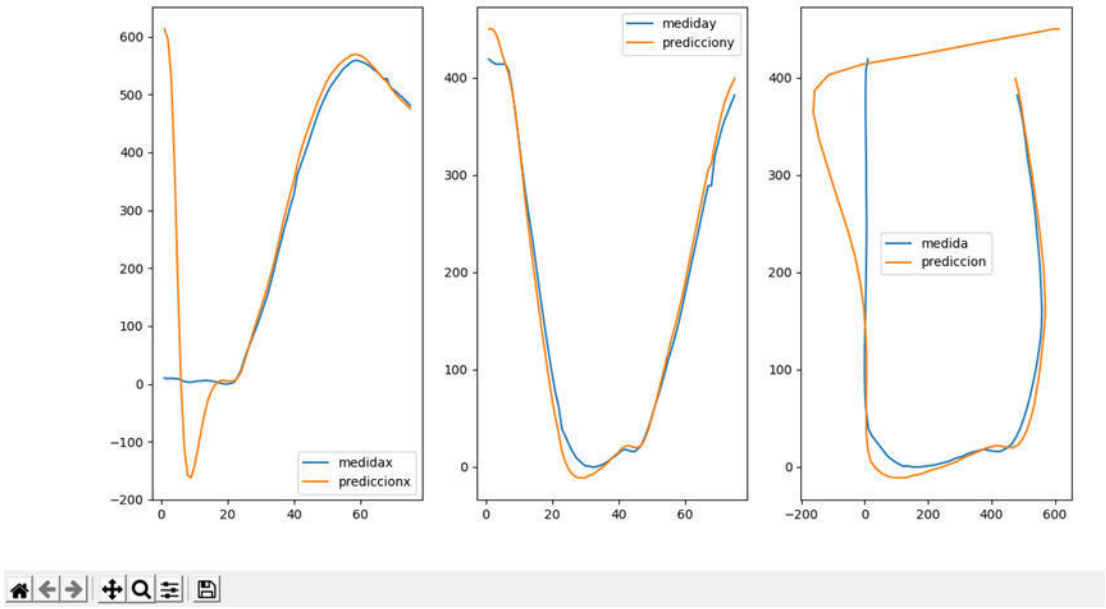


Figura 57. Representación gráfica resultados

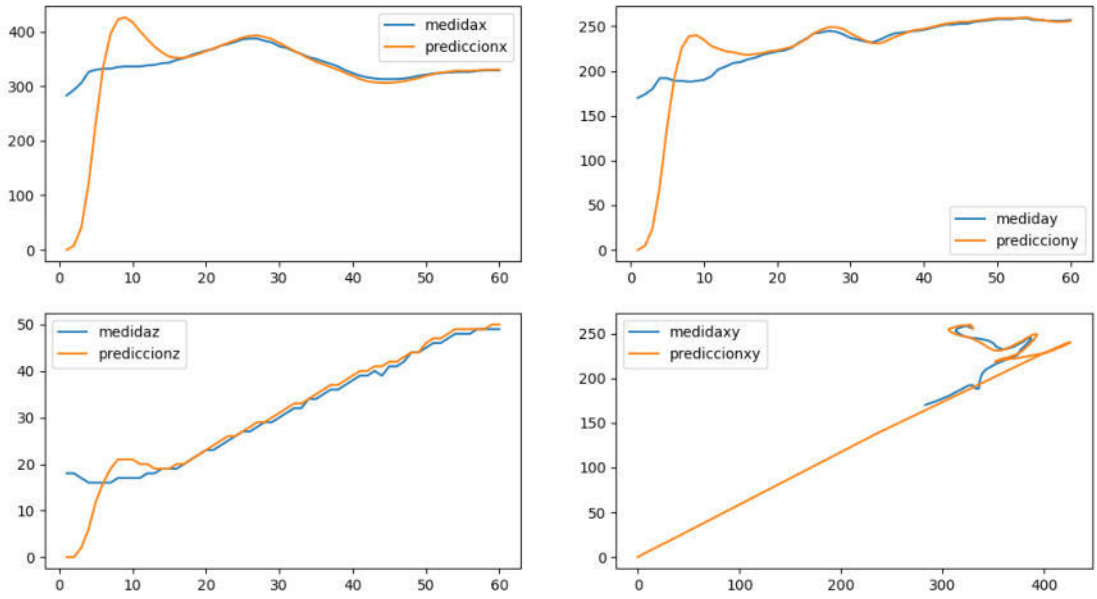


Figura 58. Graficas de resultados

Luego, para graficar los datos suministrados en la medición y los datos obtenidos en la predicción con el filtro del Kalman, se genera la gráfica en 3 dimensiones de los ejes x,y y

z, identificando la medición con el color verde la medición y el color rojo la predicción. El código y el resultado grafico se muestra en la Figura 59 y Figura 60 respectivamente.

```
#Creacion del objeto figura
fig = plt.figure()
#Configuracion para proyeccion en 3d
ax1 = fig.add_subplot(111,projection='3d')
#Titulo de la grafica
plt.title("Medidas (verde) vs Prediccion (rojo)",
        fontdict={'family': 'serif',
        'color' : 'darkblue',
        'weight': 'bold',
        'size': 18})
#Dibujo de datos de medicion
ax1.plot_wireframe(np.array([globals()["medidasx"+str(i)]]),
                   np.array([globals()["medidasy"+str(i)]]),
                   np.array([globals()["medidasz"+str(i)]]),
                   color="g",label="medidas")
#Dibujo de datos de prediccion
ax1.plot_wireframe(np.array([globals()["prediccionx"+str(i)]]),
                   np.array([globals()["predicciony"+str(i)]]),
                   np.array([globals()["prediccionz"+str(i)]]),
                   color="r",label="prediccion")
#Titulo para eje X y Y
plt.ylabel("Eje y")
plt.xlabel("Eje x")
# Muestra del gráfico
plt.show()
```

Figura 59. Código para graficar resultados en 3d

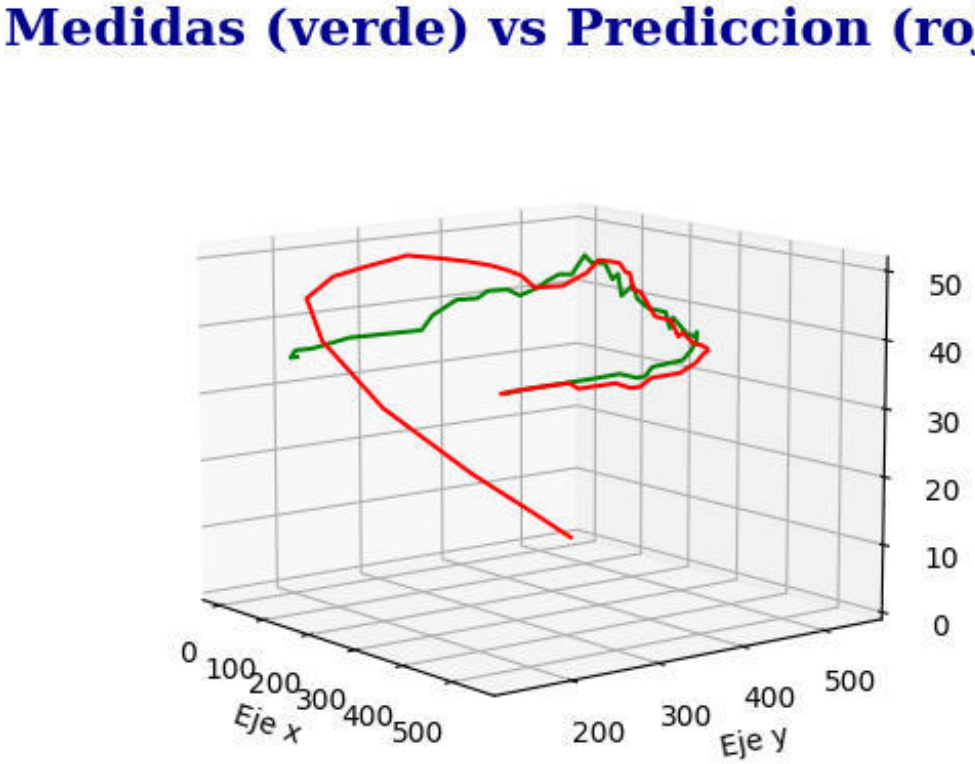


Figura 60. Resultado grafica 3d

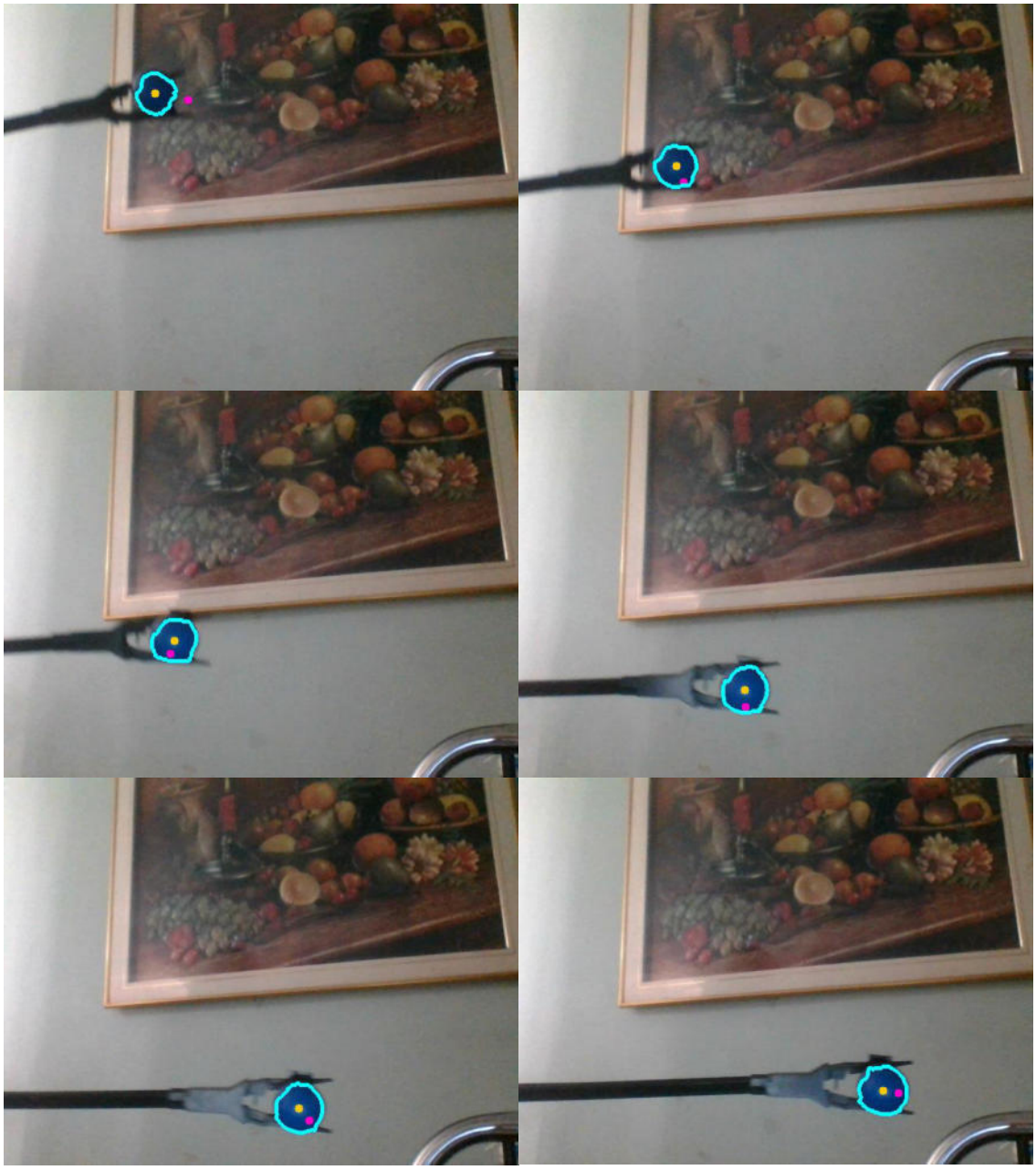
9.ANÁLISIS Y RESULTADOS

En el análisis de los resultados, se tomaron cinco pruebas entre las cuales en se utilizaron en su orden: objeto azul diámetro 4cm, objeto fucsia diámetro 4cm, objeto amarillo diámetro 4cm, objeto verde diámetro 4cm y los 4 objetos anteriores al tiempo. Para cada prueba se tomaron 60 imágenes online a través de una cámara web con el fin de obtener el tiempo total de muestreo y realizar la comparación entre este tiempo y el total de las imágenes capturadas .
En las graficas de resultados, se realiza el análisis de cinco graficas las cuales consta de medición y predicción de la posición del centroide en el eje x, en el eje y, en el eje z, en el plano cartesiano (x,y) y en el espacio (x,y,z). En las graficas de los ejes, las medidas de los datos están dadas por posición en pixeles en el eje vertical versus número de imagen capturada; en la grafica del plano cartesiano (x,y) tanto el eje vertical y el eje horizontal los datos son medidos en pixeles. Para la grafica en tres dimensiones, los datos del eje x y eje y están dados en pixeles y para el eje z, están dados en centímetros.
Los resultados de estas pruebas se encuentran a continuación:

- Prueba con objeto azul:

Una vez capturada cada imagen, en esta prueba el algoritmo se encarga de obtener los datos de los contornos del objeto azul (perímetro color azul celeste) y la posición del centroide del objeto (punto amarillo) y con esta información y por medio del filtro de Kalman obtener la predicción de la posición futura del centroide del objeto (punto fucsia). El movimiento que se intentó recrear con el objeto fue un recorrido en "U". En la Figura 61 se muestran diez imágenes en intervalos de cada seis imágenes mostrando el cambio de la posición del objeto. Se observa que la posición inicial medida del centroide difiere de la posición predicha por el filtro de Kalman, pero a medida que el algoritmo tiene mayores imágenes de evaluación, las dos posiciones tienden a coincidir.

En la Figura 62, se muestra las gráficas que se generaron al terminar el proceso de seguimiento y predicción del objeto azul, donde se realiza la comparación de las medidas y las predicciones obtenidas en cada uno de los ejes (x,y,z) vs la cantidad de datos (imágenes capturadas) tomadas y una cuarta grafica donde se muestra el movimiento en el plano (x,y) del objeto. En el encabezado se identifica el valor del tiempo total de la prueba, en este caso de 8.458232 segundos.



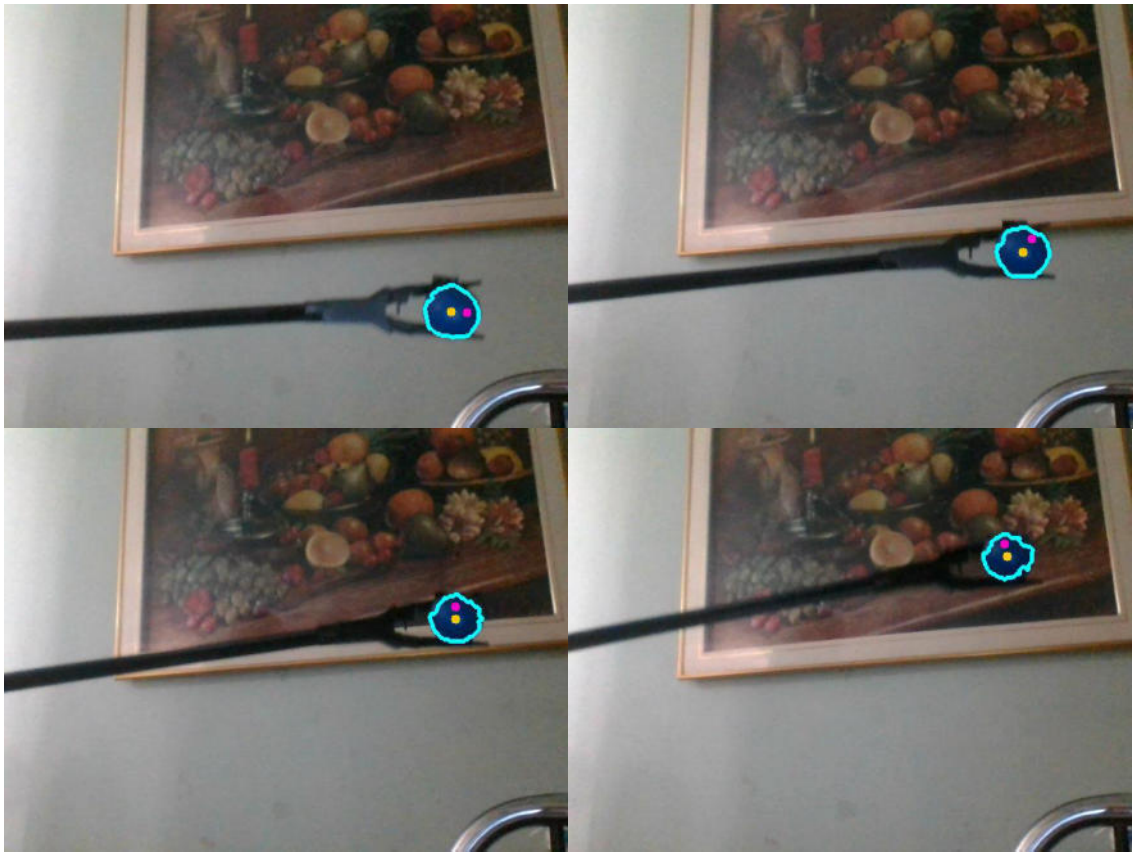


Figura 61. Capturas del seguimiento y predicción objeto azul movimiento en "U"

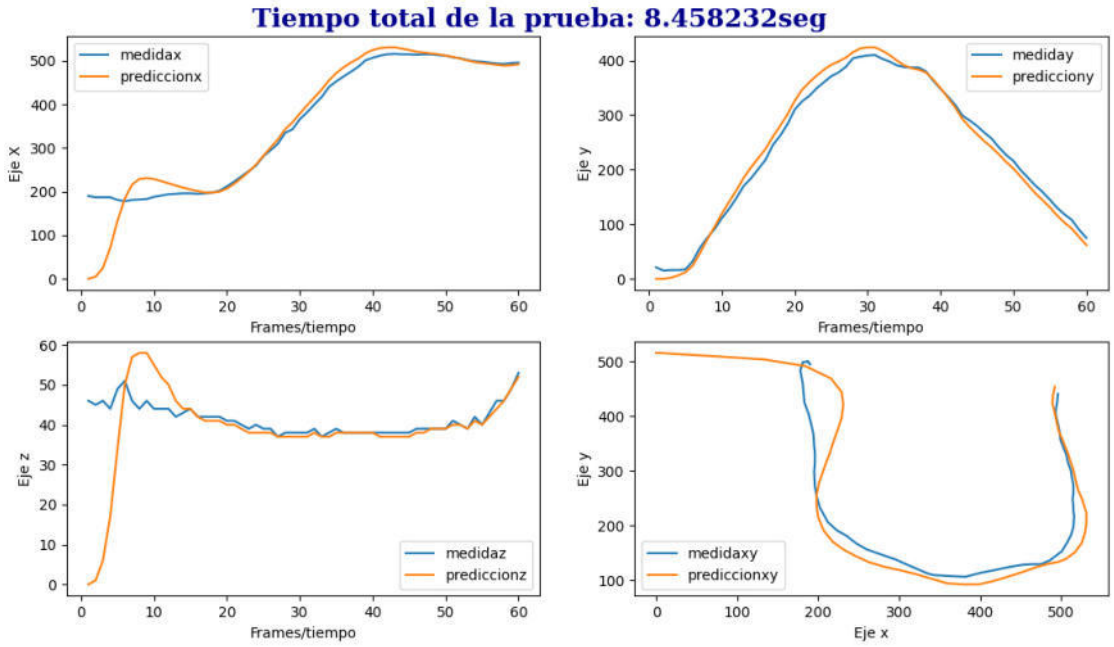


Figura 62. Graficas resultados obtenidos prueba objeto azul

En la gráfica de comparación de la medida en el eje x y predicción de la posición del centroide en el eje x (Figura 63), se observa como luego de aproximadamente de 16 imágenes capturadas (2.114558 segundos) la predicción logra estabilizarse, esto debido a que su valor inicial en el eje x es la posición 0 y el de la medida la posición 200. A su vez, después de la imagen capturada número 40, luego de ejercer un cambio en la posición medida, la predicción logra realizar la corrección con un movimiento suave, pero estabilizándose aproximadamente en 10 imágenes capturadas después (1.40970 segundos).

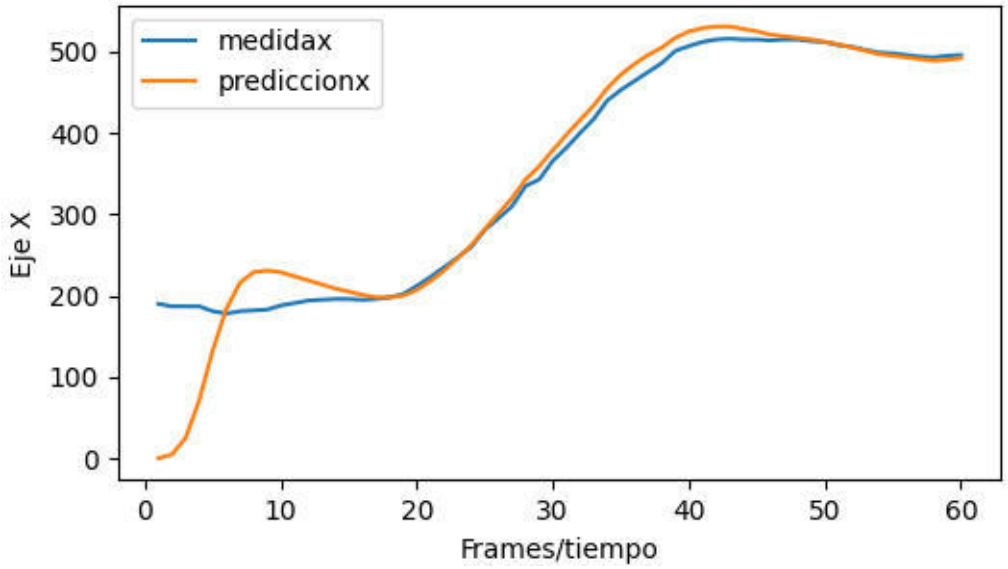


Figura 63. Grafica comparación medida y predicción eje x vs cantidad de capturas

En la Figura 64 se observan los resultados obtenidos en los valores de la medida en el eje y y los valores obtenidos en la predicción de la posición en el eje y. Donde los resultados muestran valores aproximados entre ellos, teniendo en cuenta, que los valores en color naranja son la predicción de la posición siguiente de la medida anterior.

La Figura 65 muestra los resultados de la medición y la predicción en la posición en el eje z, de manera similar que en la predicción en el eje x, esta toma cierta cantidad de imágenes capturadas en estabilizarse, ya que la predicción es dependiente del dato de la medida y el dato inicial de la predicción es 0.

En la Figura 66, se muestra el movimiento que realizo el objeto sobre el plano cartesiano x,y, a lo largo de la prueba realizada, en color naranja muestra la predicción futura de la posición de dicho movimiento. Para la preducion del movimiento se hace notorio el tiempo de estabilización sobre el eje x por lo explicado anteriormente en la Figura 63.

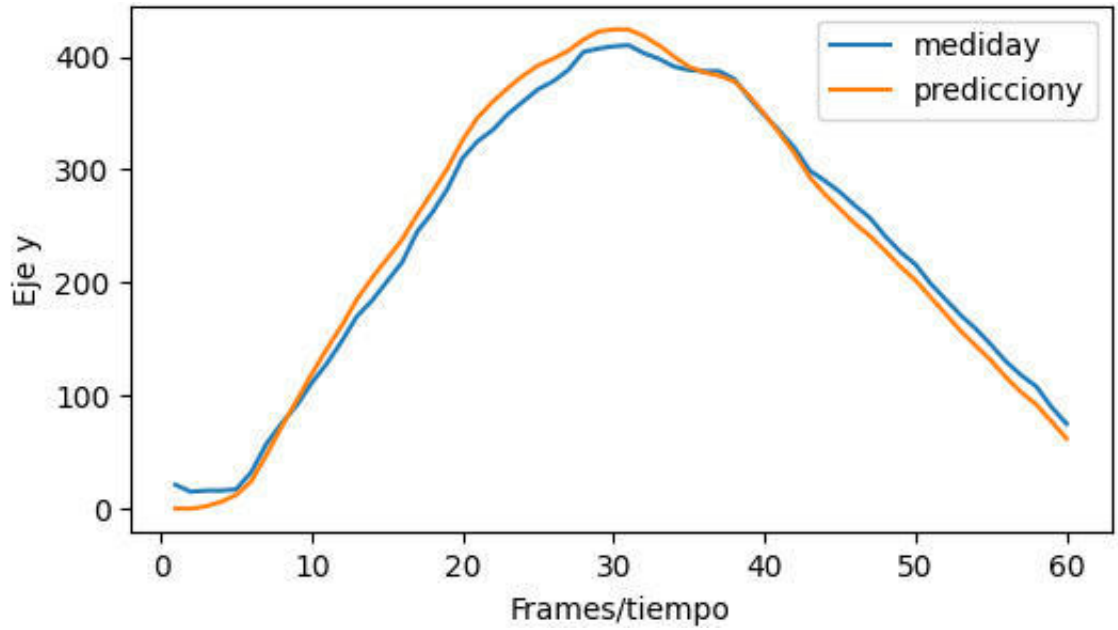


Figura 64. Grafica comparación medida y predicción eje y vs cantidad de capturas

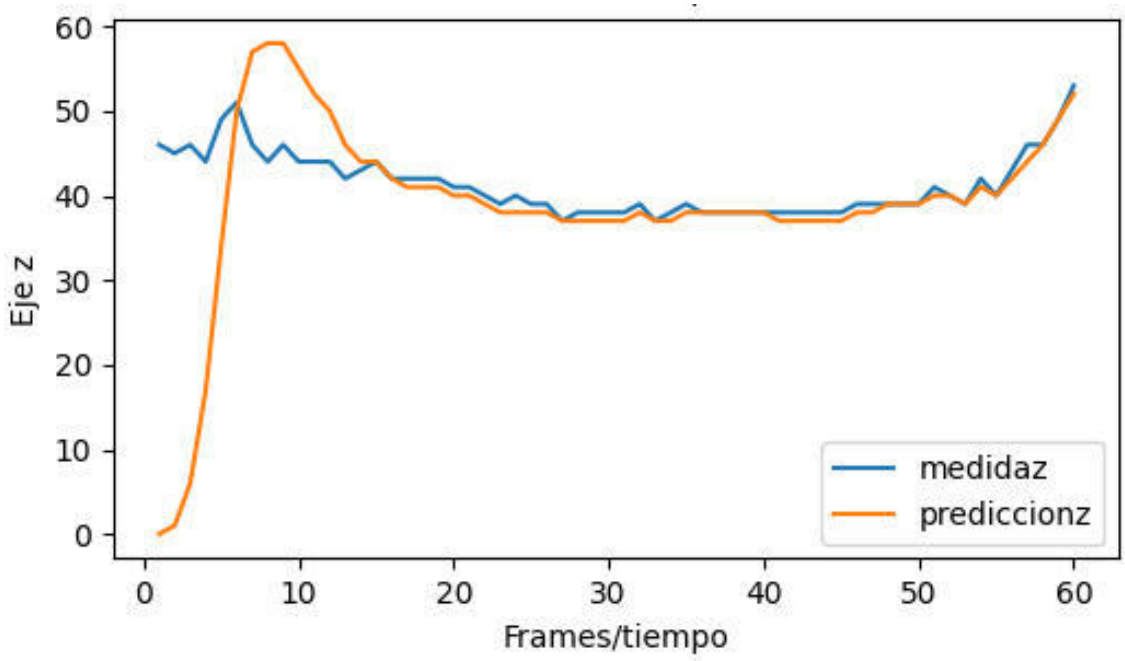


Figura 65. Grafica comparación medida y predicción eje z vs cantidad de capturas

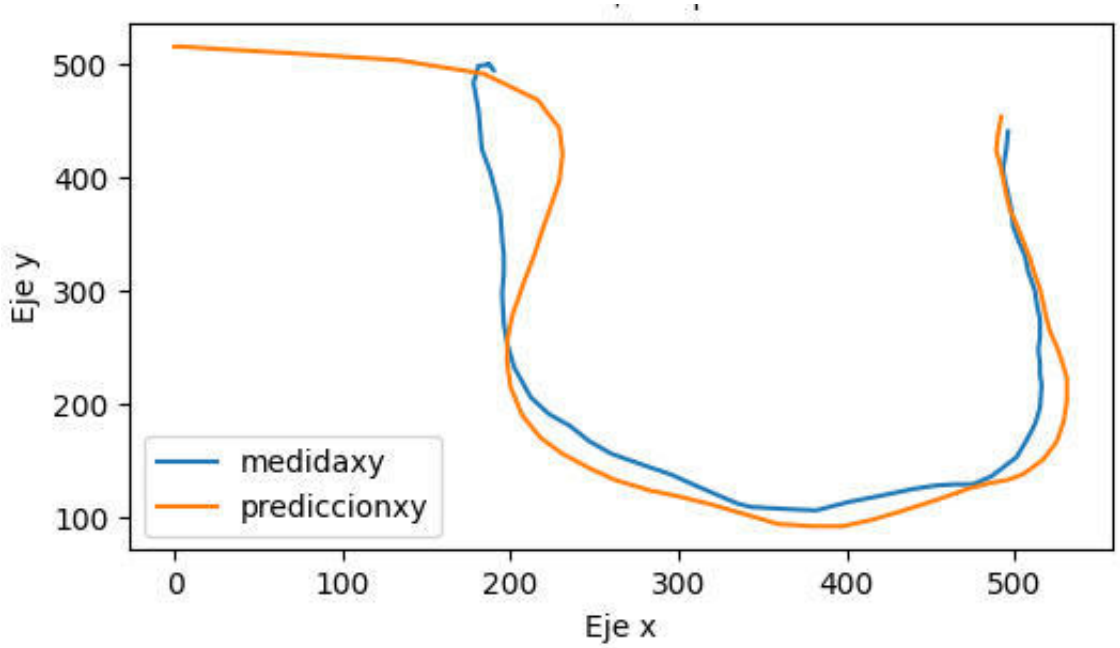


Figura 66. Grafica movimiento objeto en 2d sobre plano (x,y)

En la figura 67, se realizo la grafica en tres dimensiones de los ejes x,y y z, donde las medidas de las posición es son graficadas en color verde y la predicción de las posición es en color rojo. Se nota la suavidad en el movimiento que genera la trayectoria de la predicción pero aun asi manteniendo aproximadamente la dirección de las medidas de posición tomadas de los centroides.

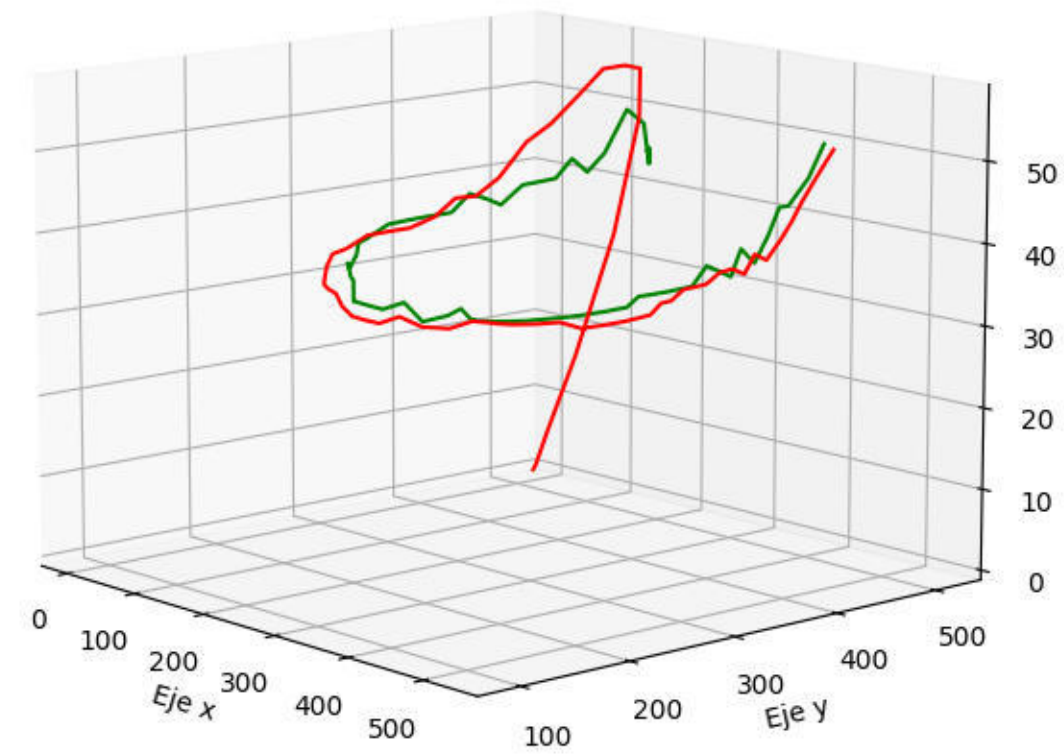
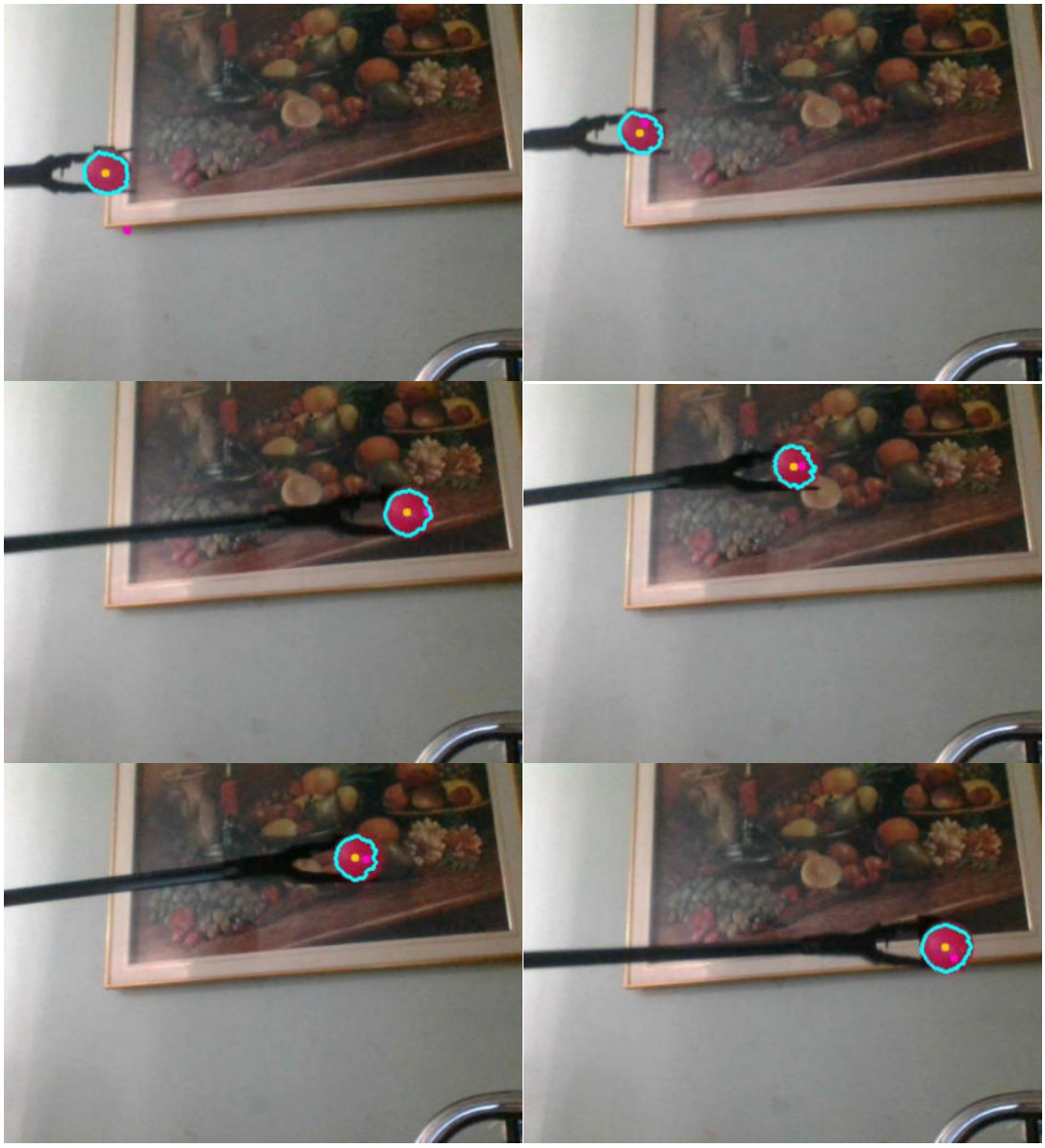


Figura 67. Grafica movimiento objeto azul en 3d en el espacio (x,y,z)

- Prueba con objeto de color fucsia

Luego se realizó la misma prueba relacionada anteriormente, pero empleando un objeto de color fucsia. Algunas imágenes capturas de esta prueba se muestran en la Figura 68. Se tomaron 60 imágenes en un tiempo 8.45593 segundos. Se observa que inicialmente los dos centroides (medido y predicho) están separados, pero luego de la captura de mas imágenes, los dos centroides tienden a coincidir.



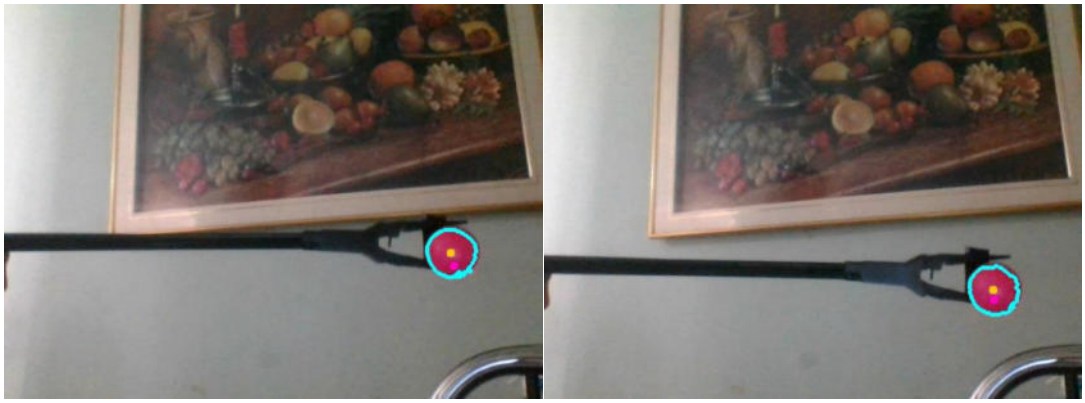


Figura 68. Capturas del seguimiento y predicción objeto fucsia movimiento en "U" invertida

En la Figura 69, se puede observar que, por la naturaleza del movimiento del objeto en esta prueba, en los 3 ejes (x,y y z), la predicción de la posición toma un tiempo de estabilizar la trayectoria debido al cambio brusco en las posiciones iniciales del objeto en relación con el dato inicial de la predicción que es cero en los 3 ejes.

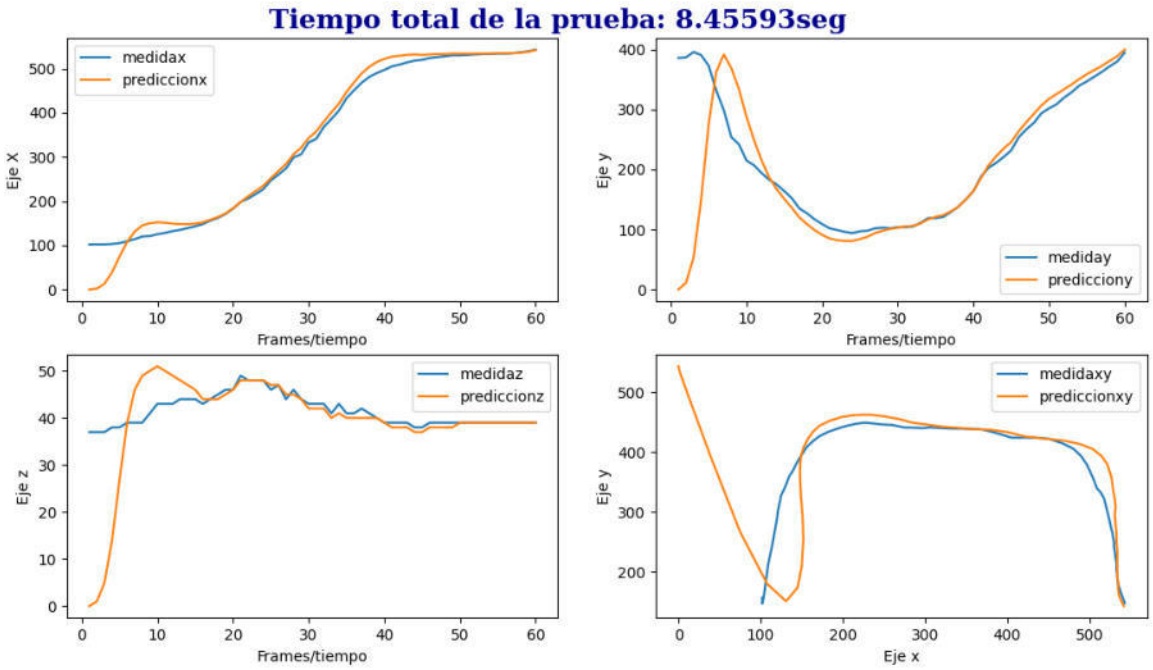


Figura 69. Graficas resultados obtenidos prueba objeto fucsia

En la Figura 70 se muestra la gráfica en tres dimensiones de la prueba anterior. De manera similar a los resultados obtenidos en la prueba con el objeto azul, la trayectoria de la predicción tiene la misma dirección que la trayectoria de las medidas, con la diferencia en la suavidad del movimiento en la predicción.

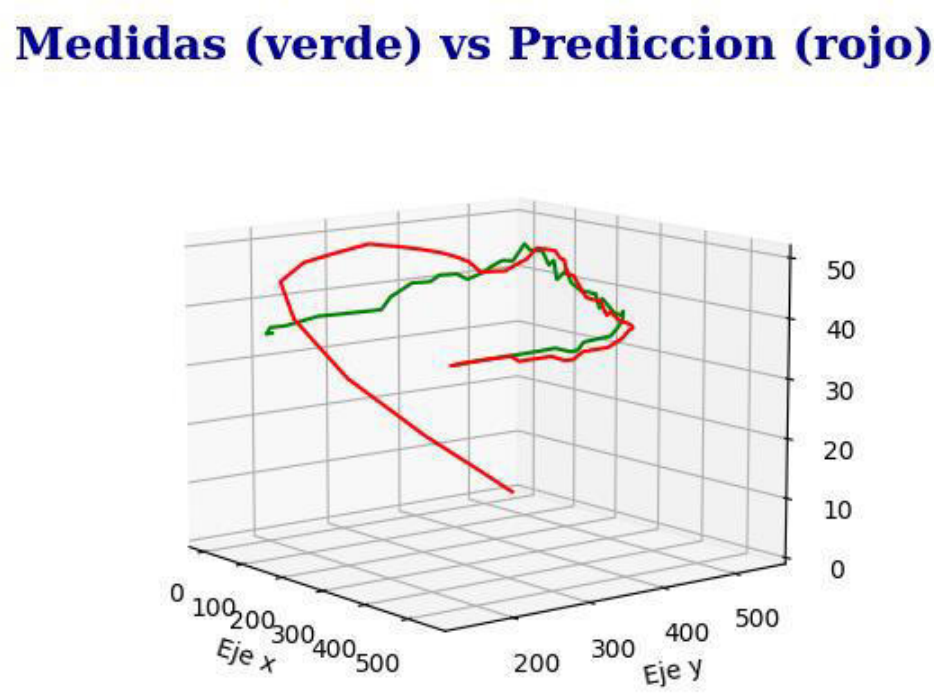
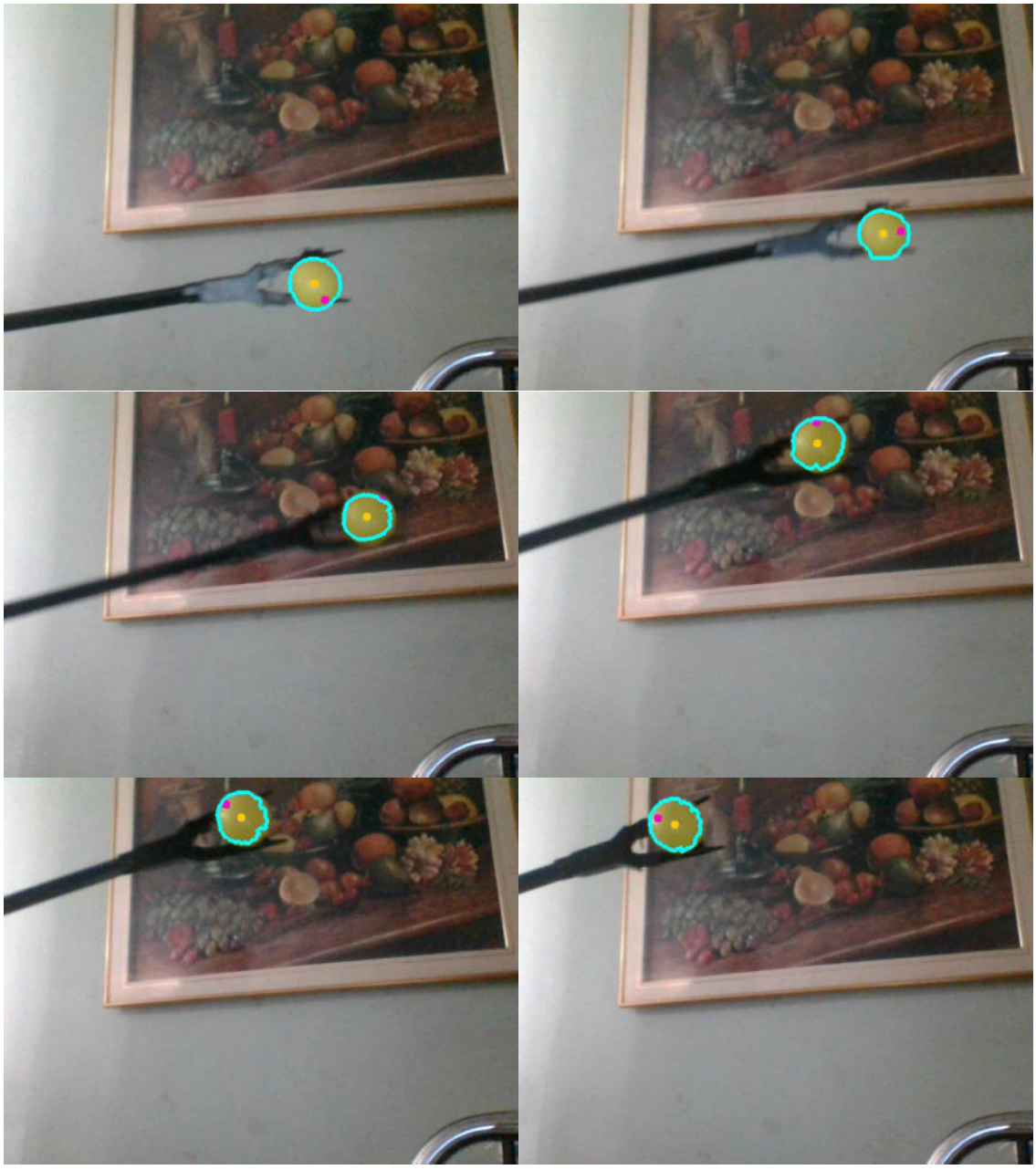


Figura 70. Grafica movimiento objeto fucsia en 3d en el espacio (x,y,z)

- Prueba objeto de color amarillo

Para la prueba con el objeto de color amarillo, se recreó un movimiento de tipo elíptico, así como se muestra en las imágenes capturadas en la Figura 71 y las gráficas en el plano cartesiano de la Figura 72. En esta prueba, el algoritmo presenta un funcionamiento eficiente y muy parecido a las anteriores pruebas, sin mayores alteraciones. En la Figura 73, se muestra el movimiento del objeto y la trayectoria de la predicción en las tres dimensiones, donde se puede observar un cambio mayor en la posición en el eje z en comparación a las pruebas anteriores, teniendo en cuenta que el eje z es el eje de medida y predicción más propenso al error, debido a que es tomado de acuerdo con ecuaciones lineales que entregan datos aproximados a los reales.



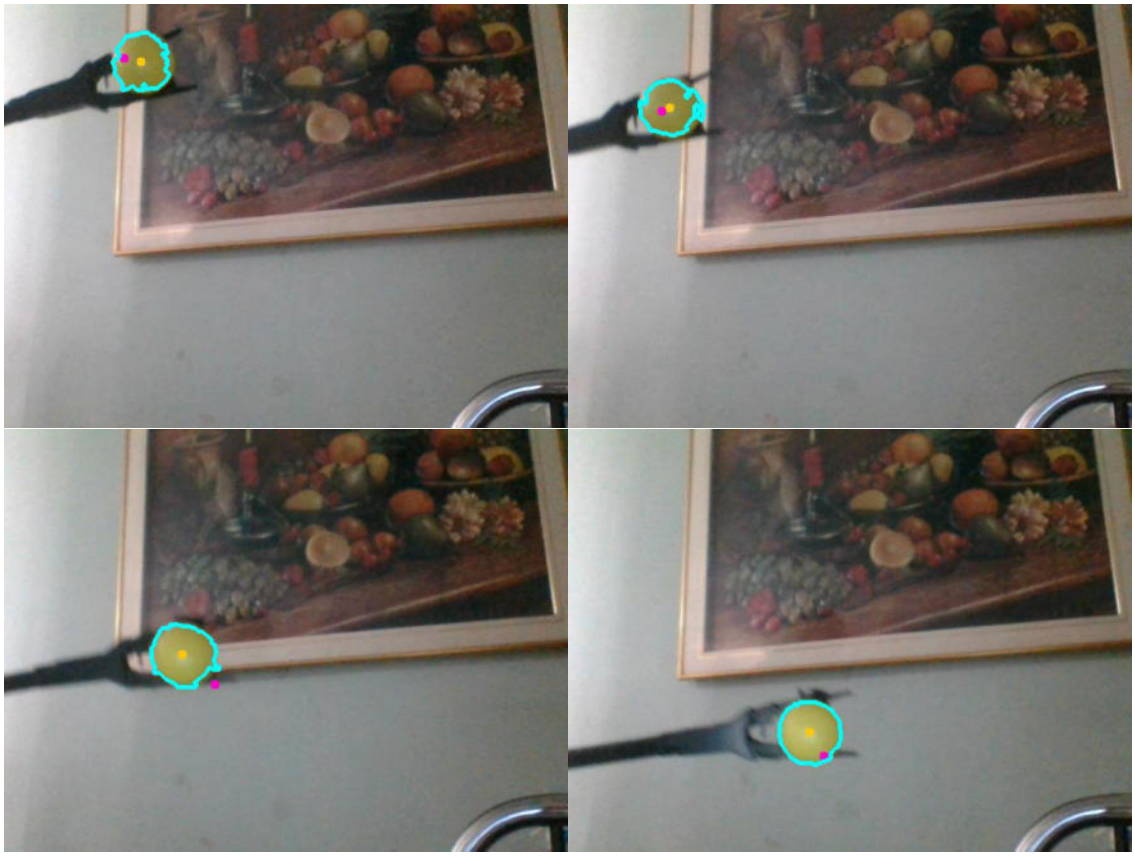


Figura 71. Capturas del seguimiento y predicción objeto amarillo movimiento en forma de elipse.

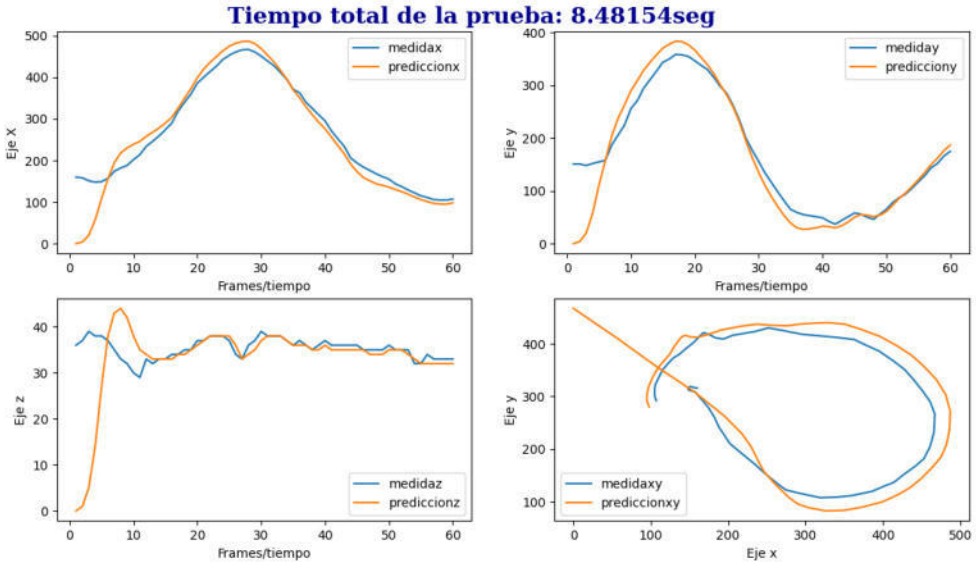


Figura 72. Graficas resultados obtenidos prueba objeto amarillo

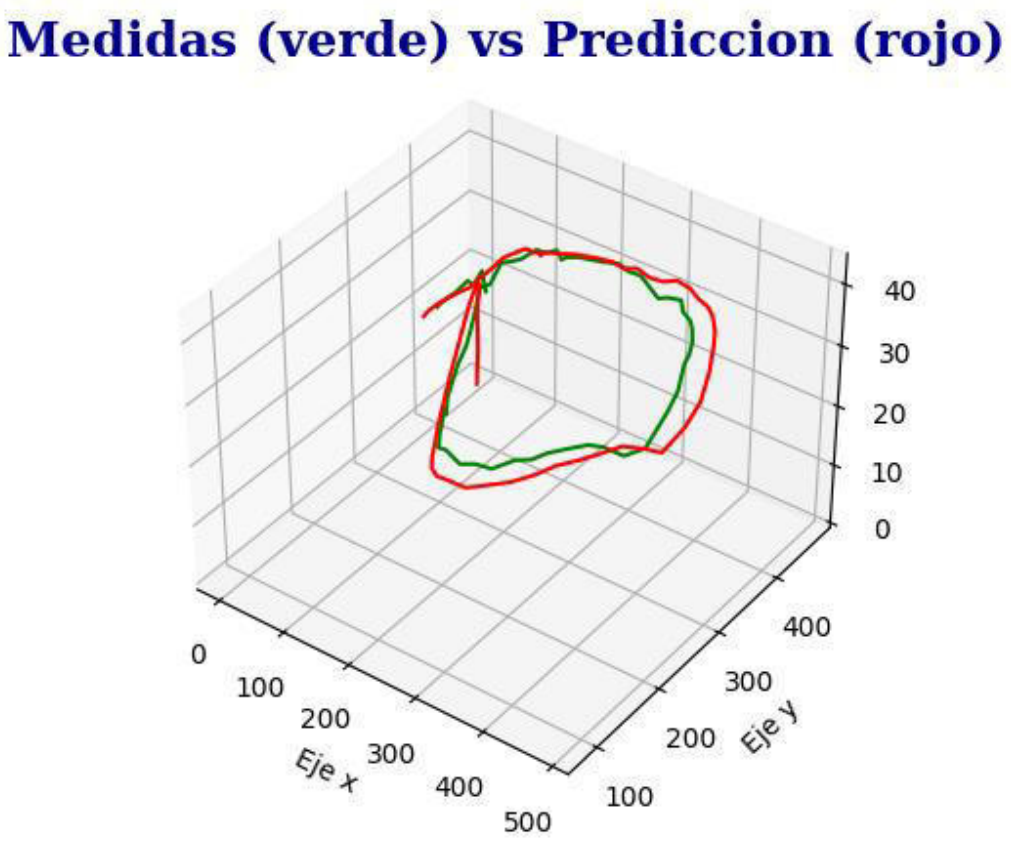


Figura 73. Grafica movimiento objeto amarillo en 3d sobre el espacio (x,y,z).

- Prueba objeto color verde

Para la prueba con el objeto de color verde, se intentó recrear un movimiento de tipo sinusoidal, así como se muestra en las imágenes capturadas en la Figura 74 y las gráficas en el plano cartesiano de la Figura 75. En esta prueba, el algoritmo presenta un funcionamiento eficiente y muy parecido a las anteriores pruebas, con alteraciones en los datos correspondientes al eje y, en los cuales se puede apreciar en la segunda grafica de la Figura 75, como al cambio bruscos en el movimiento del objeto en dicho eje, la trayectoria de la predicción intenta volver a la trayectoria de la medida. En la Figura 76, se muestra el movimiento del objeto y la trayectoria de la predicción en las tres dimensiones, donde se puede observar el seguimiento y predicción del objeto sin alteraciones notorias.

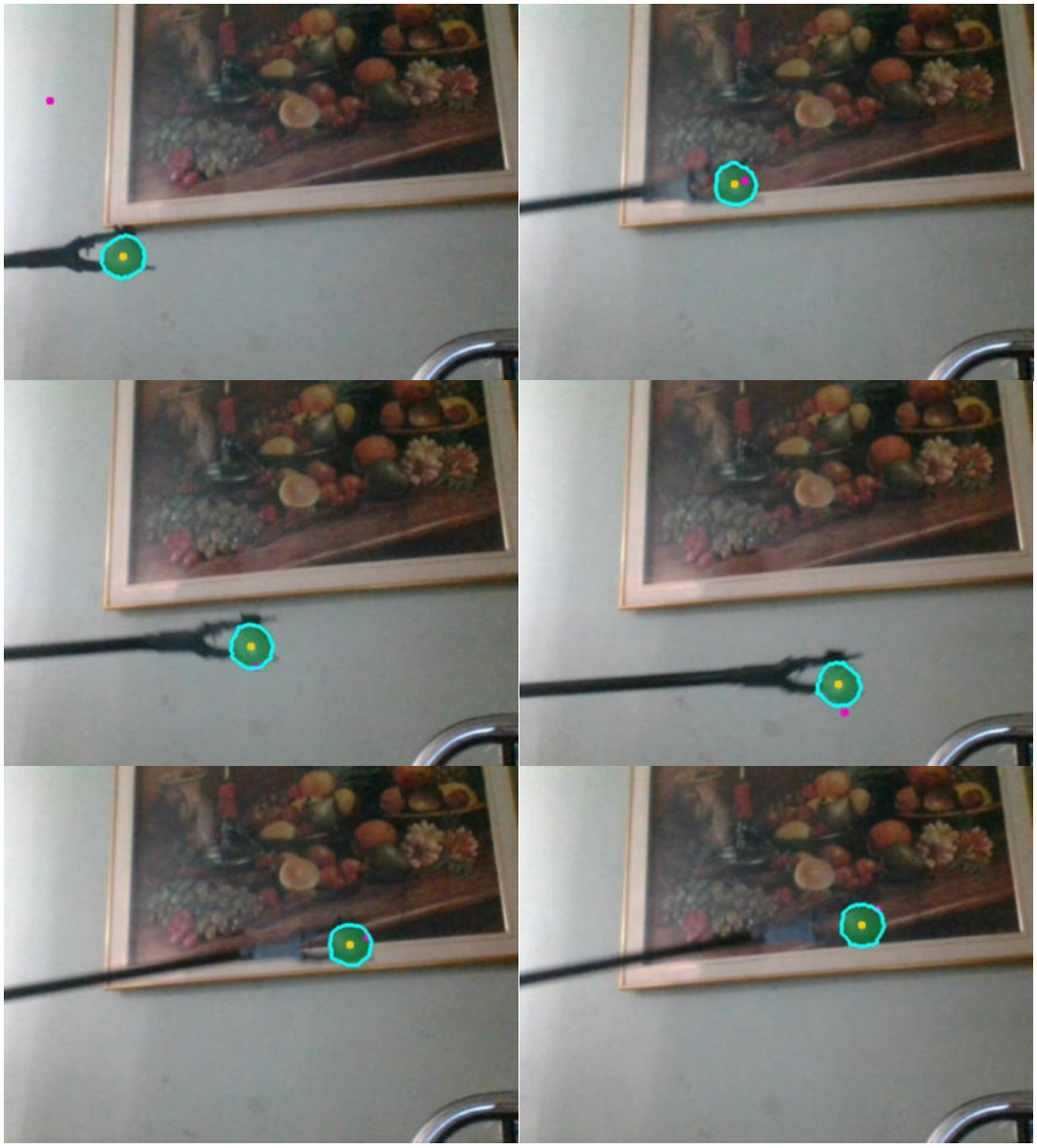


Figura 74. Capturas del seguimiento y predicción objeto verde movimiento sinusoidal.

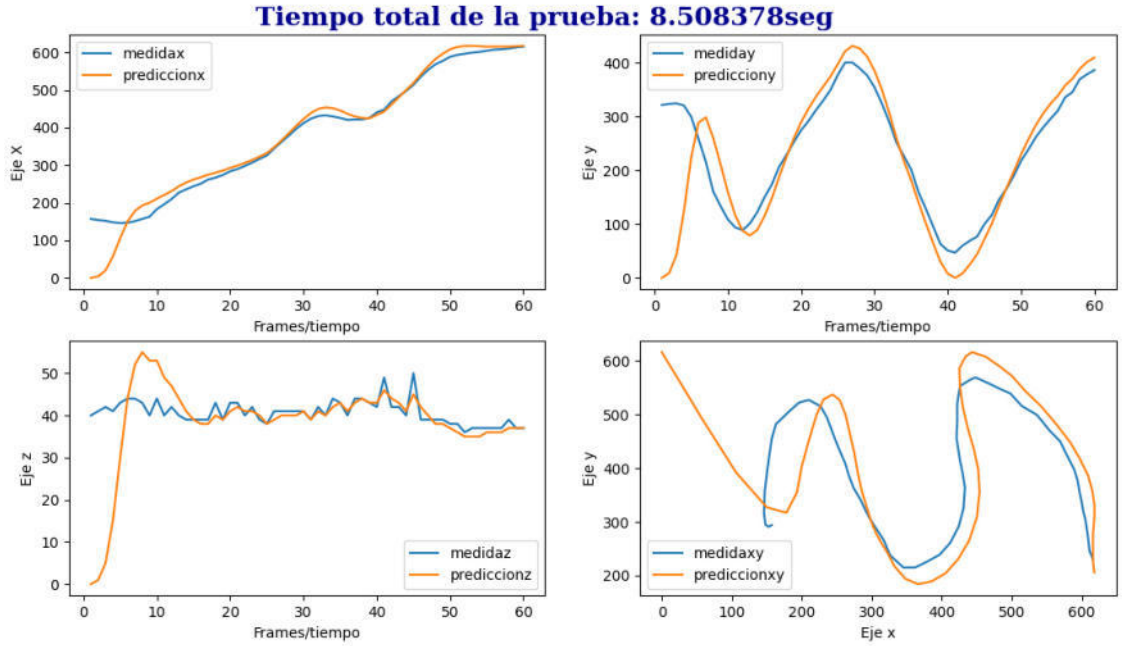


Figura 75. Graficas resultados obtenidos prueba objeto verde

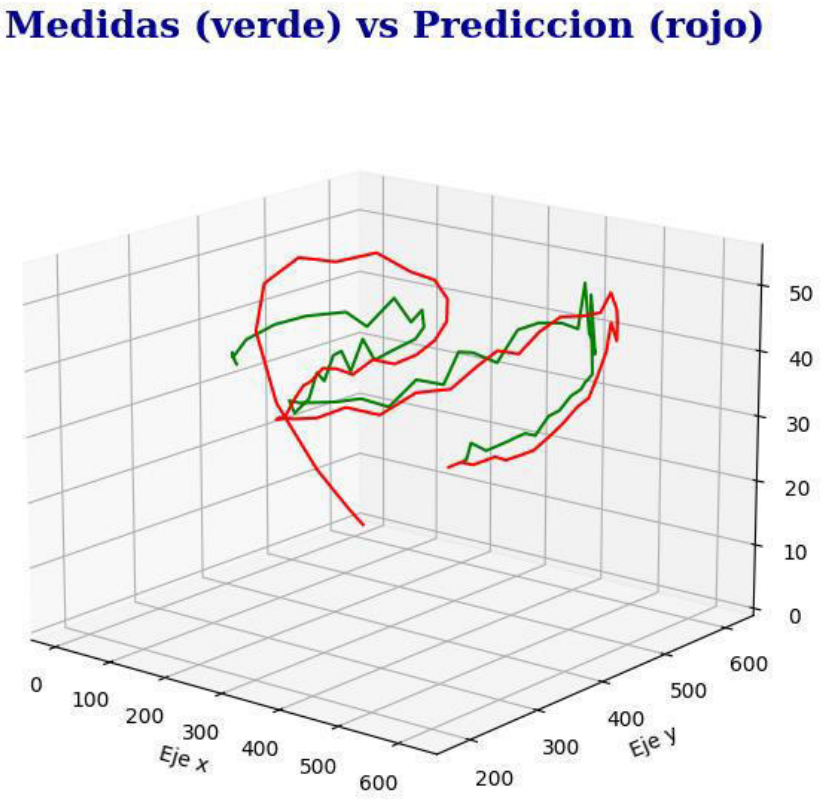
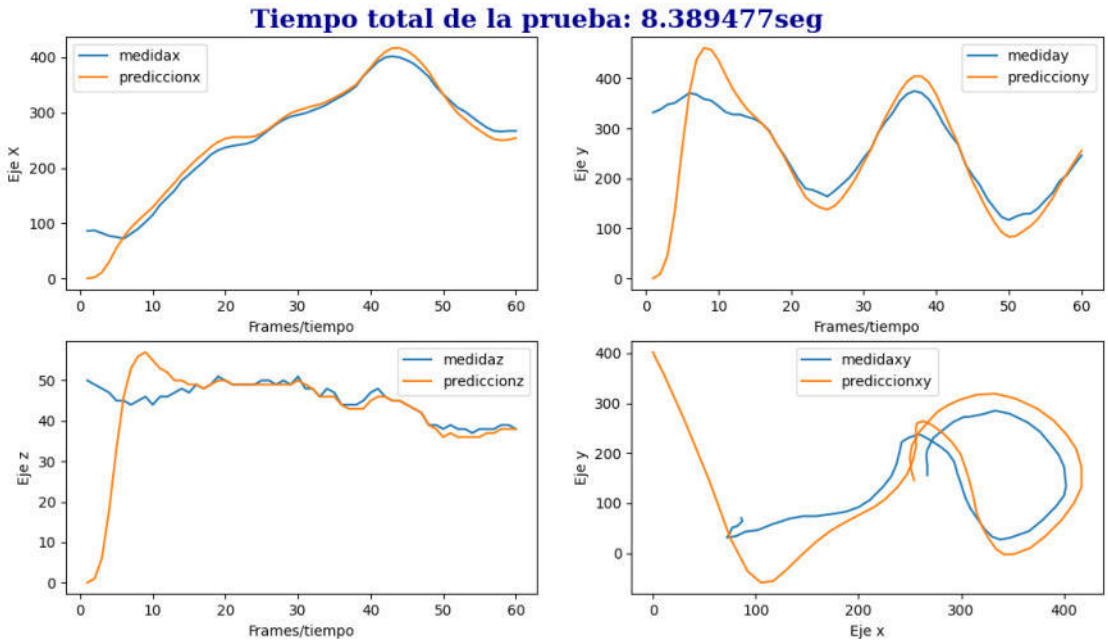


Figura 76. Grafica movimiento objeto verde en 3d sobre el espacio (x,y,z)

Para finalizar se realizó una quinta prueba, la cual consto de realizar el proceso de seguimiento y predicción para los cuatro objetos al tiempo, donde se pudo comprobar que el cambio de uno a cuatro objetos al tiempo, no aumento el tiempo de ejecución para lograr las 60 imágenes capturadas. La Figura 77 muestra algunas imágenes capturadas tomadas de la prueba, en las cuales se logra apreciar el correcto función amiento en el dibujo de los puntos indicadores de seguimiento y predicción de la posición de cada uno de los objetos. En las Figuras 78, 79, 80 y 81, se observa la aproximación en las trayectorias entre los objetos, los cuales se fijaron a un soporte a través del cual realizo el movimiento de los cuatro objetos al tiempo. La grafica del movimiento en el plano (x,y) del objeto de color amarillo presenta un movimiento adicional en comparación a los otros objetos, debido a que en la prueba fue el primer objeto en aparecer en pantalla y por esa razón tuvo un movimiento inicial diferente al de los otros objetos.

Figura 77. Capturas del seguimiento y predicción cuatro objetos al tiempo.



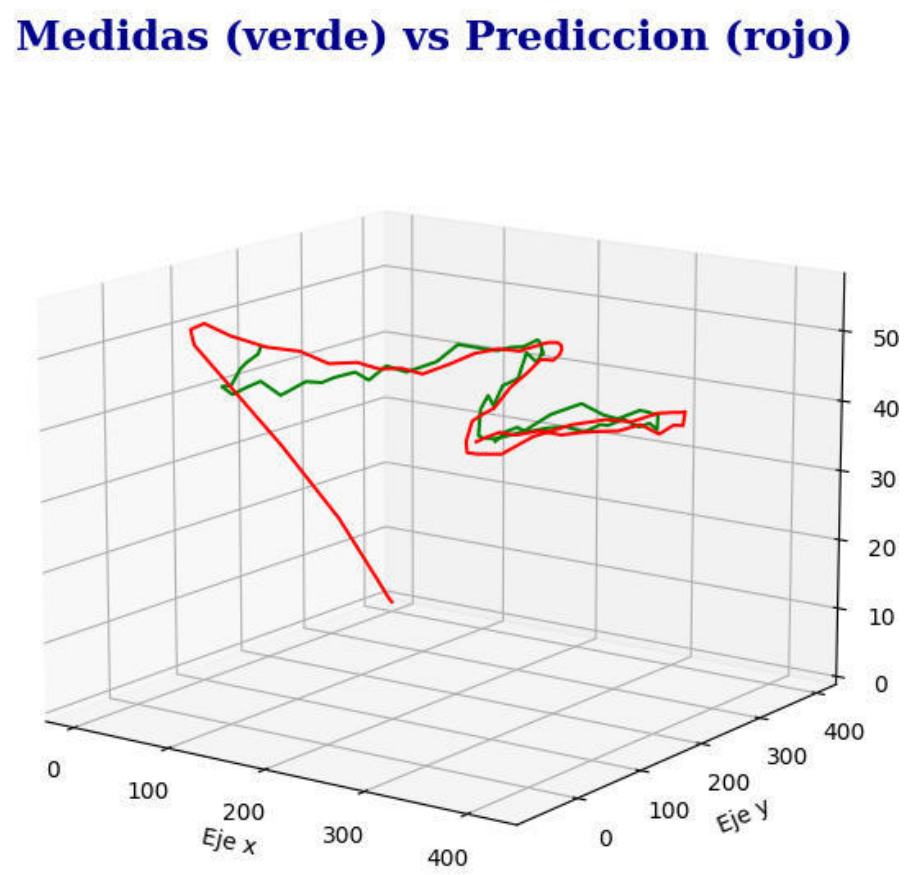
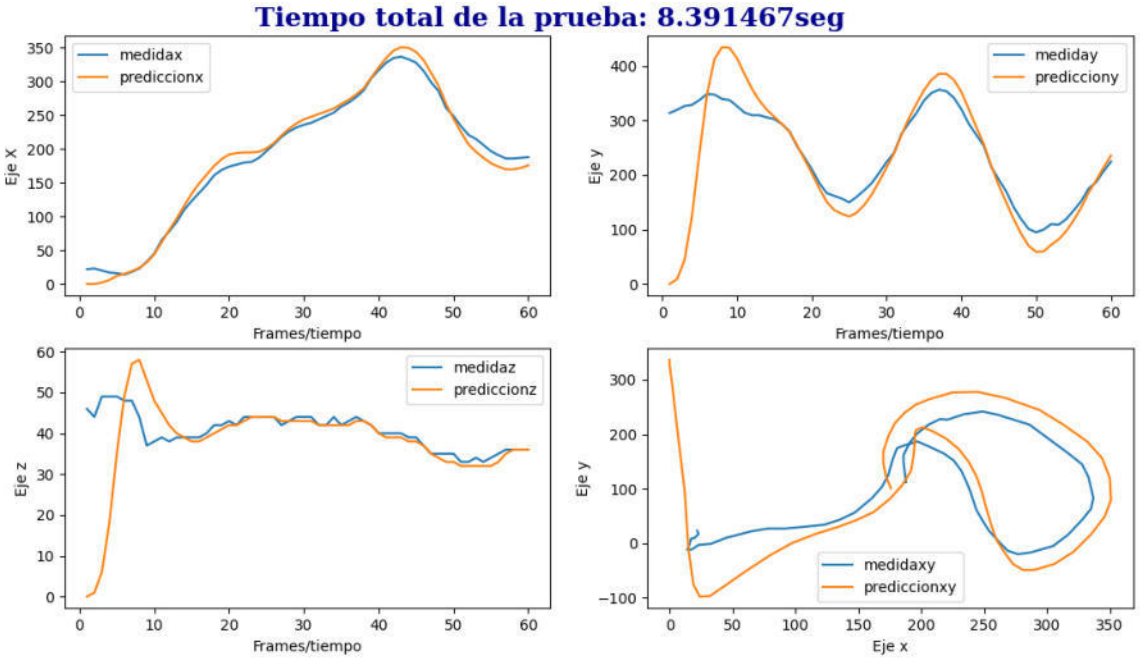


Figura 78. Graficas resultados obtenidos prueba cuatro objetos objeto azul



Medidas (verde) vs Prediccion (rojo)

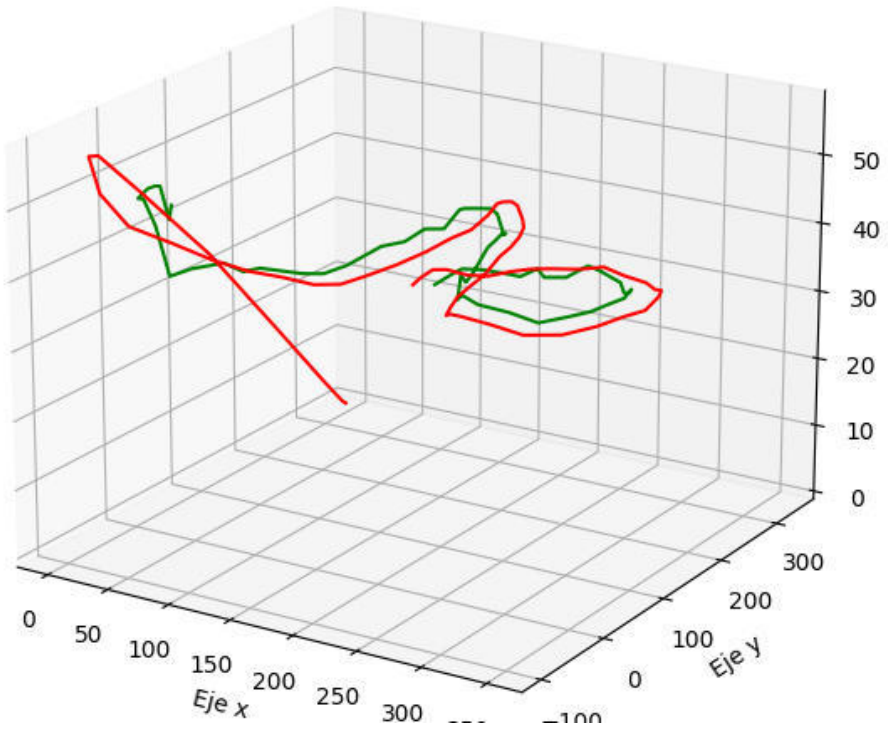
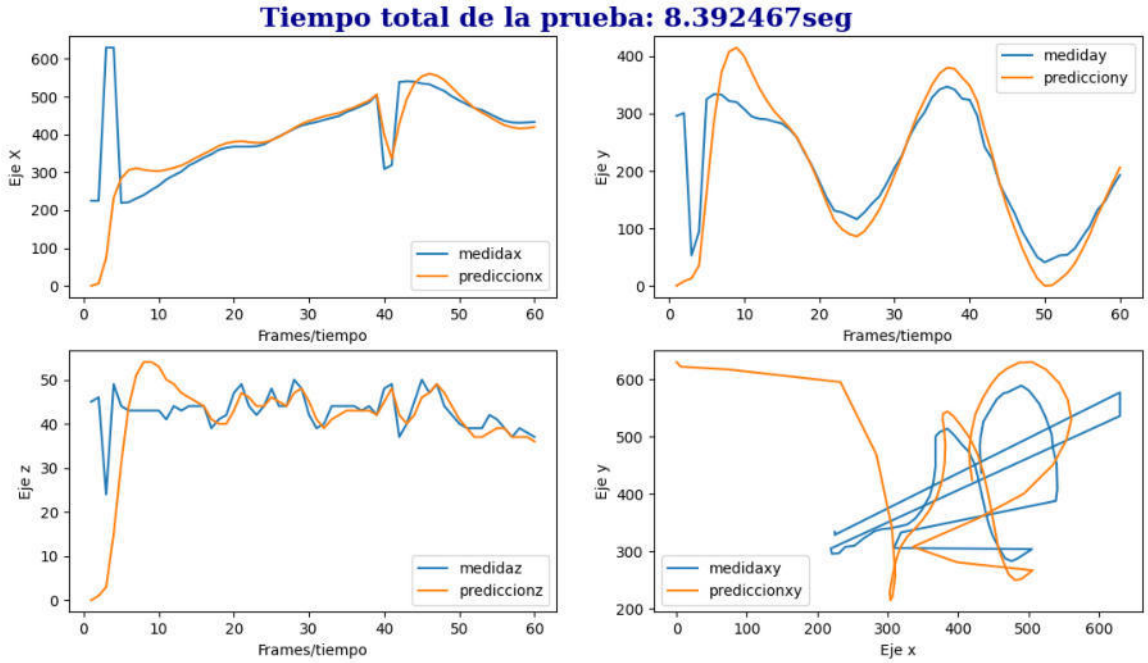


Figura 79. Graficas resultados obtenidos prueba cuatro objetos objeto verde


Tiempo total de la prueba: 8.392467seg
Eje X
Frames/tiempo
medidax
prediccionx
Eje y
Frames/tiempo
mediday
predicciony
Eje z
Frames/tiempo
medidaz
prediccionz
Eje y
Eje x
medidaxy
prediccionxy

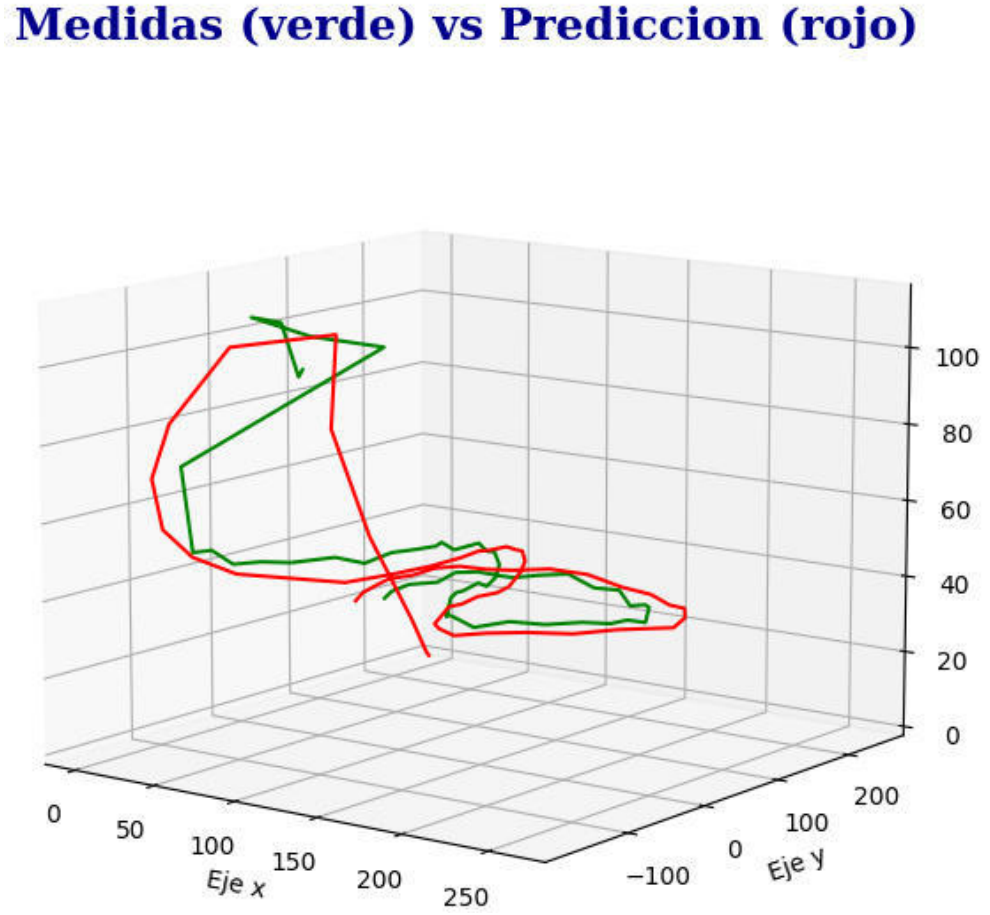
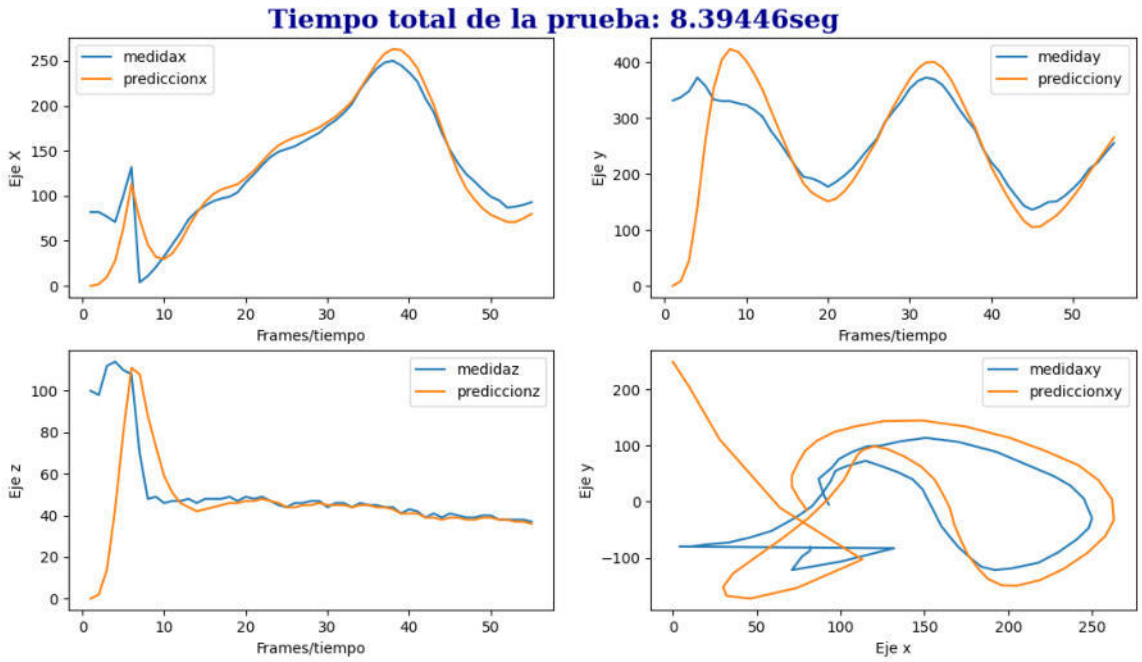


Figura 80. Graficas resultados obtenidos prueba cuatro objetos objeto amarillo


Tiempo total de la prueba: 8.39446seg
medidax
prediccionx
Eje X
Frames/tiempo
mediday
predicciony
Eje y
Frames/tiempo
medidaz
prediccionz
Eje z
Frames/tiempo
medidaxy
prediccionxy
Eje y
Eje x

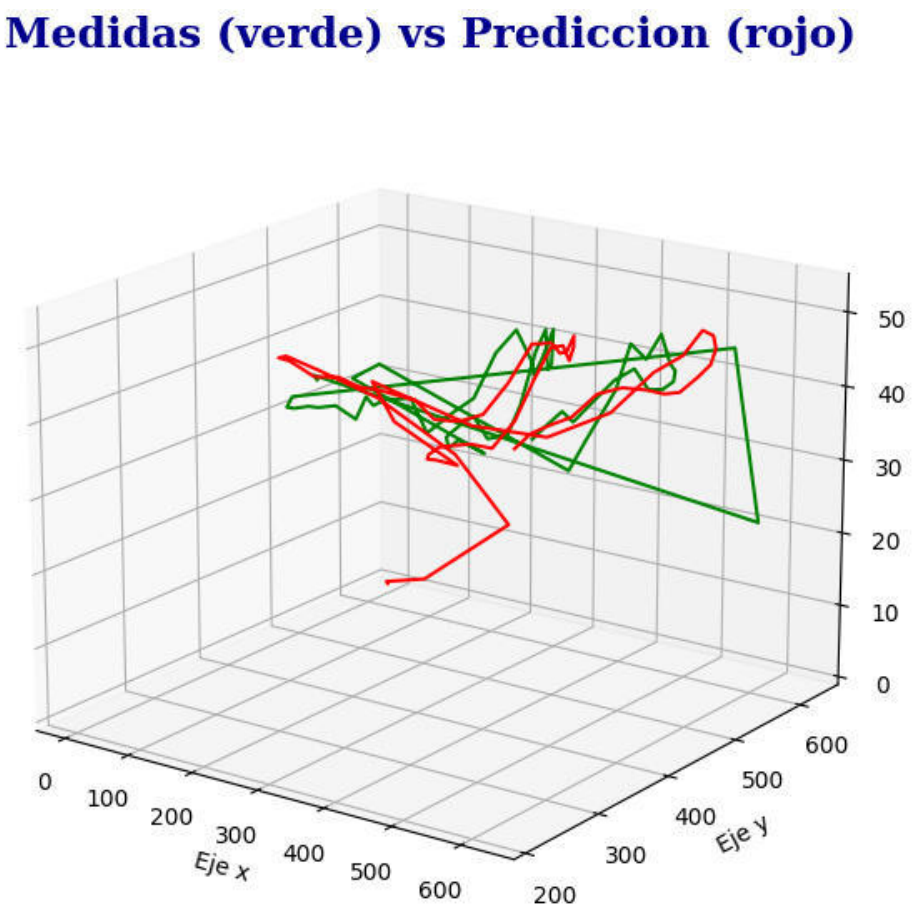


Figura 81. Graficas resultados obtenidos prueba cuatro objetos objeto fucsia

10. CONCLUSIONES

- El desarrollo del proyecto permitió abarcar los procesos que se deben realizar en el diseño de un algoritmo con uso de la visión artificial relacionados con la captura de imágenes, procesamiento de imágenes, segmentación en espacios de color, obtención de contornos y centroides, detección de características propias de cada imagen, etc.

- Luego de una breve revisión bibliográfica se encontró información suficiente para el desarrollo de este proyecto en la implementación del filtro de Kalman incluido como una clase de la librería OpenCV, logrando un conocimiento del funcionamiento del algoritmo.

- Por medio del entorno de programación Python, se desarrollo un algoritmo eficiente para la captura de imágenes a través de una cámara web permitiendo el procesamiento de esta imagen empleando la librería OpenCV. A su vez, el algoritmo diseñado da las bases para el desarrollo de algoritmos de seguimiento y predicción de objetos más robustos, que pueden ser usados en la realización de futuros proyectos de investigación y una posible inclusión en la industria.

- Por medio de la captura de imágenes y el procesamiento de estas, se logró la extracción de características relevantes como el centroide, área y contornos de los objetos de interés. Al algoritmo permitir la umbralización de los objetos didácticamente, se logro aumentar la cantidad de objetos que se pueden procesar en comparación a los propuestos en los objetivos (2), realizando pruebas con cuatro objetos al tiempo, obteniendo los datos y resultados de cada uno sin aumentar el tiempo de ejecución del algoritmo.

- Se realizo la implementación del algoritmo del filtro de Kalman con las características extraídas de los objetos circulares, realizando el seguimiento en una secuencia de imágenes y el posterior análisis de los resultados gráficos de las tareas de detección y predicción.

- La interfaz gráfica diseñada es agradable al usuario, facilitando su uso sin necesidad de conocimientos avanzados en el tema, así mismo, el algoritmo en general, el cual para trabajos futuros permite mejoras tanto de hardware (procesador, cámara web) como de propiedades del ambiente y los objetos, como variaciones en tamaño, color, iluminación, formas, etc.

- Como una parte principal en cualquier proyecto que incluya visión artificial, la segmentación de imágenes tuvo gran importancia en este, así como también la aplicación de transformaciones morfológicas, los cuales combinados dieron resultados eficientes teniendo en cuenta las variaciones de iluminación, presencia de ruidos externos al realizar pruebas en sitios abiertos con fondos que podían afectar el correcto funcionamiento.

- Por medio del dibujo de graficas de resultados en 2d y 3d con el uso de herramientas de la librería Matplolib se pudo realizar el análisis de los resultados obtenidos de manera visual con la comparación de los datos recolectados en la medición con los obtenidos en la predicción, siendo estos aproximados entre sí.

- Como trabajos futuros se plantean la implementación del filtro de Kalman para el seguimiento de objetos por parte de un sistema robótico. Este trabajo fortalecerá la línea de investigación en robótica industrial en el grupo de control industrial de la Universidad Pontificia Bolivariana.

- En la implementación del filtro de Kalman se observó en las primeras capturas de imágenes diferencias notorias entre el valor de los datos de las medidas iniciales del centroide difieren de las posición es predichas por el filtro de Kalman, pero a medida que el algoritmo tiene mayor imágenes de evaluación, las dos posición es tienden a coincidir, en donde la trayectoria de los datos de la posición predicha presentan suavidad en los cambios de acuerdo a las perturbaciones presentes en las medidas.

11. BIBLIOGRAFÍA

Munuera, M. (2018). Filtro de Kalman y sus aplicaciones. Universidad de Barcelona.

Open-Source Computer Vision Library. (2020). Home OpenCV. Recuperado de https://opencv.org

Becerra, A. (2009). Introducción a la programación con Python. Pontificia Universidad Javeriana de Cali.

Roldan, D. (2015). Sistema multiplataforma de gestión integral de huertos urbanos. Universidad Politécnica de Madrid.

Lázaro, M. (2001). Procesado Digital de Señales en Comunicaciones, 4.6 El Filtro de Kalman.

Python. (2020). documentación. Recuperado de https://www.python.org
Microsoft. (2020).¿Qué es Python? Recuperado de https://docs.microsoft.com/es-es/learn/modules/python-introduction

González, R. (2015). Python para todos. España. Universidad Tecnológica Intercontinental

Marzal y García. (s, f.). Introducción a la programación con Python. Universitat Jaume-1.

Arévalo et al. (2012). La librería de visión artificial OpenCV aplicación a la docencia e investigación. Universidad de Málaga.

Mogena, A. (2014). Informe de OpenCV y Tratamiento de Imágenes. Open CV.

Cea, E. (2018). Estudio y aplicación de la librería OpenCV sobre la arquitectura ARM, para el control de agentes robóticos móviles usando visión artificial. Universidad del Bio-Bio.

Viera, G. (2017). Procesamiento de imágenes usando OpenCV aplicado en raspberry pi para la clasificación del cacao. Universidad de Pihura.

Chaparro Laso, (2017) E. Tracking automático de objetos en secuencias de imágenes usando Filtro de Partículas. Universidad de Extremadura.

Rodríguez Muñoz, P. (2003). Aplicación del filtro de Kalman al seguimiento de objetos en secuencia de imágenes. Universidad Rey Juan Carlos.

Ontiveros Gallardo, S. (2015). Diseño de un sistema de seguimiento utilizando filtros de Kalman y filtros de correlación adaptativos. Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California.

Casares, C. Farias, N. Garcia, N. Garcia, A. (2017). Procesamiento de imágenes de plantas ornamentales multi escala para calcular su crecimiento.

Martínez, J., Fernández, A. (2018). Sistema de seguimiento de objetos usando OpenCV, ArUco y Filtro de Kalman extendido. Universidad de Sevilla.

Gary R. Bradski. Computer Vision Face Tracking For Use in a Perceptual User Interface. Microcomputer Research Lab, Santa Clara, CA, Intel Corporation.

Aguirre, N. (2013). Implementación de un sistema de detección de señales de tráfico mediante visión artificial basado en fpga. Universidad de Sevilla.

Garcia J., Gardel A., Bravo I., Lázaro J., Martínez M., Rodríguez D. (2012). Detección y seguimiento de personas basado en estereovisión y Filtro de Kalman. Universidad de Alcalá.

Ormaechea, R. (2015). Detección y seguimiento de objetos móviles en secuencias de video.

Ravish Aradhya, (2019). Object detection and tracking using deep learning and artificial intelligence for video surveillance applications.

Chandan G., Harsh J., (2018). Real time object detection and tracking using deep learning and OpenCV.

Baptista, M., Martinez, C., Losada, C. Marron, M. Sistema robusto para la detección y seguimiento de personas en aplicaciones videovigilancia. Universidad de Alcalá.

Yépez, H. (2019). Plataforma abierta para desarrollo de dispositivos de tarifación vehicular: Software para ajustes. Universidad Técnica del Norte.