UNIVERSIDAD DE GUADALAJARA
CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

Procesamiento digital de Señales

2023A

Producto integrador: Implementación de Visión Artificial

Docente:	Stewart Rene Santos Arce
Integrantes del equipo:	Cristopher Jesus Rubio Sanchez: 217639714
	Edwin Omar Arellano Madera: 217734504
Fecha:	18 de mayo de 2023, Guadalajara

Contenido

Introducción

¿Qué es la visión artificial?

La visión artificial, también conocida como visión por computadora (del inglés computer vision) o visión técnica, es una disciplina científica que incluye métodos para adquirir, procesar, analizar y comprender las imágenes del mundo real con el fin de producir información numérica o simbólica para que puedan ser tratados por un ordenador. Tal y como los humanos usamos nuestros ojos y cerebros para comprender el mundo que nos rodea, la visión artificial trata de producir el mismo efecto para que los ordenadores puedan percibir y comprender una imagen o secuencia de imágenes y actuar según convenga en una determinada situación. Esta comprensión se consigue gracias a distintos campos como la geometría, la estadística, la física y otras disciplinas [4].

El objetivo último de la visión artificial es conseguir el desarrollo de estrategias automáticas para el reconocimiento de patrones complejos en imágenes de múltiples dominios. En la actualidad, muchos son los campos que se han visto beneficiados por este conjunto de técnicas. Uno de los más conocidos es el de la robótica, ya que los robots con cierta autonomía deben reconocer con precisión la localización de los objetos de su entorno para no colisionar contra ellos, por ejemplo. A menudo, esto lo consiguen por medio de sensores o de cámaras, siendo estos últimos dispositivos idóneos para la aplicación de las estrategias de visión por ordenador.

A la hora de aplicar los conceptos teóricos de la visión por ordenador encontraremos siempre interferencias y problemas relacionados con el mundo que nos rodea. Esto es por el mero hecho de que nuestro mundo no es perfecto y los aparatos de medición y captura tampoco lo son. Estos introducen siempre (a mayor o menos cantidad) una distorsión o ruido que contamina la muestra o imagen con la que deseamos trabajar, existen muchos tipos de ruidos para las imágenes pero los más conocidos son: salt and pepper, ruido uniforme, ruido Gaussiano, y muchos más [4].

Interferencias debidas al contexto: Punto de vista, Oclusión, Escala, Deformación, Fondo desordenado, Variaciones dentro de una misma clase.

La detección de objetos es la parte de la visión artificial que estudia cómo encontrar la presencia de objetos en una imagen sobre la base de su apariencia visual, bien sea atendiendo al tipo de objeto (una persona, un coche) o a la instancia del objeto (mi coche, el coche del vecino). Generalmente se pueden distinguir dos partes en el proceso de detección: la extracción de características del contenido de una imagen y la búsqueda de objetos basada en dichas características.

Objetivo

En este documento se desarrollara nuestro proyecto el cual debe de cumplir con una serie de lineamientos como lo son encender la cámara para que de algún modo tenga movimiento ya sea con motores o cualquier otro componente, debe de tener un control el cual suavice los movimientos y la implementación de visión artificial.

La cámara debe de ser capaz de reconocer un color en especifico el cual nosotros seleccionaremos por medio de la misma para después seguirlo en todo momento y responder a los movimientos que este haga. Es importante tener en cuenta los límites de nuestros componentes como lo son la capacidad máxima de grados que puede moverse los motores que utilizaremos.

El objeto que usaremos debe de ser de un color llamativo el cual no se confunda con los objetos y preferentemente en una superficie que no refleje la luz, el fondo del mismo debe de ser uniforme también para evitar alguna intervención del ambiente, por ultimo debe de ser un objeto lo suficientemente grande para que la cámara lo detecte.

El código debe de cumplir con lo siguiente, en cuanto se corra debe abrir la cámara para seleccionar el objeto el cual vamos a segmentar, inmediatamente después de segmentar el color debe de abrir la pantalla donde se observara el seguimiento, en este código debe de estar un control PID el cual suavice y elimine el error lo mayor posible siempre y cuando no afecte a los motores, el seguimiento no debe de pasar los 180 grados en los ejes x y y para el cuidado de los motores, por ultimo debe de apreciarse el movimiento en la pantalla.

Cumpliendo con todo lo anterior la practica habrá sido un éxito logrando así la visión artificial en nuestro dispositivo.

Desarrollo

Segmentación de color

Comenzamos por la creación del algoritmo para segmentar colores en una imagen, en nuestro caso, elegimos *Matlab* para el desarrollo del código ya que personalmente nos pareció más sencillo por las operaciones con imágenes que es capaz de realizar.

Esta primera parte la dividimos en los siguientes pasos:

- Obtener imagen con color a segmentar.
- Selección del color.
- Obtener las referencias (valores) del color.
- Buscar el color seleccionado y eliminar los otros.
- Mostrar resultados.

Configuración de la cámara

Para la obtención de la imagen haremos uso de una cámara web como se ve en la Figura 1.

Figura 1: Cámara web utilizada para capturar las imágenes.

Configuramos la cámara utilizando la función de *imageAcquisitionExplorer* que incluye *Matlab*, en la cual pudimos configurar la cámara que ibamos a usar, seleccionar la resolución con la que trabajaremos y obtener las siguientes lineas de código que usaremos más adelante:

```
1 v = videoinput("winvideo", 1, "MJPG_640x480");
2 v.FramesPerTrigger = Inf;
3 v.ReturnedColorspace = "rgb";
```

```
recording1 = getdata(v, v.FramesAvailable);
snapshot1 = getsnapshot(v);
```

Básicamente lo que hacemos con estas líneas de código es seleccionar nuestra cámara web y asignarla a un objeto de *Matlab*, con esta variable podemos comenzar a grabar y capturar imágenes de la misma para guardarlas en alguna variable ("snapshot1.[en] nuestro caso) además de editar sus propiedades como lo son la resolución o el color (RGB por defecto pero igual lo definimos). La resolución que escogimos es de 640x480 ya que no necesitamos de demasiada información en las imágenes, además de que pueden alentar el proceso en pasos posteriores resoluciones más altas.

Algo que es importante mencionar es que nos encontramos con un problema, al tomar la captura desde *Matlab* este lo hace inmediatamente después de abrir la cámara por lo que la imagen salia muy oscura ya que a esta le costaba algo de un segundo para ajustar el brillo a la escena como se puede ver en la Figura 2. Nuestra primera solución fue aumentarle el brillo a la imagen por medio de las propiedades que *Matlab* le puede dar a la cámara pero estos nos producía más problemas posteriores ya que al estabilizarse el color el brillo era muy alto para la segmentación de color en tiempo real. A la solución final que llegamos fue hacer que de algún modo la cámara se iniciara antes de tomar la captura para que se ajuste al color y trabajar con la imagen correcta. Para hacer esto encontramos dos funciones muy útiles de *Matlab*:

```
set(v, 'TriggerFrameDelay' 5);
implay(recording1);
```

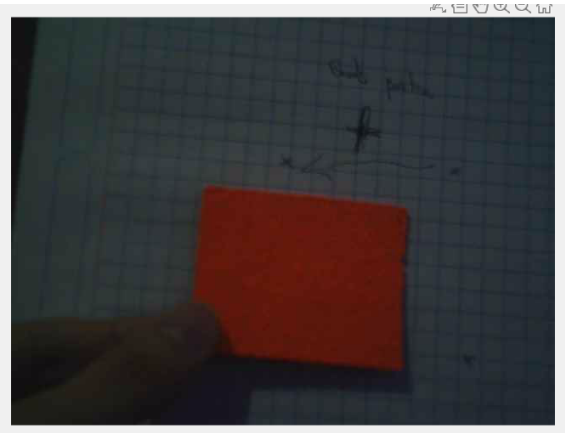

Figura 2: Si se toma la captura inmediatamente después de activar la cámara esta resulta muy oscura.

Lo que hace la primer linea de código es establecer un retraso de 5 frames a la cámara antes de tomar la captura, la segunda inicializa una ventana con la vista previa de la cámara por lo que al terminar de ejecutar estas dos líneas de código la imagen ya tenia el brillo establecido para trabajar con ella.

Selección del color y segmentación

Para la selección del color usaremos otra función de *Matlab* llamada *roipoly*, con esta función podemos delimitar el área de nuestro color para poder guardarlo en alguna variable como se puede

observar en la Figura 3.

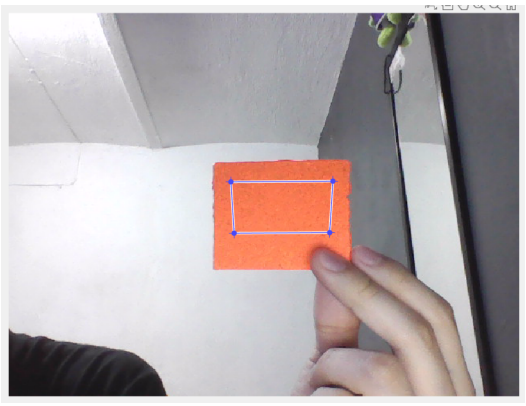

Figura 3: Funcionamiento de roipoly en *Matlab*.

Una vez teniendo la referencia del color que queremos segmentar de la imagen, necesitamos el valor de referencia para R, G y B por lo que cada uno de ellos sera un promedio de los valores de referencia que hay en nuestra imagen capturada ya que es con esto con los que segmentaremos los cada región con los valores de referencia de nuestra imagen.

Para la segmentación de la imagen definimos un umbral de 30 normalizado para 255, de este modo los colores que dejamos pasar son más y la segmentación es más exitosa. Con el umbral definido ahora de las referencias obtenidas anteriormente realizamos una búsqueda en R, G y B para segmentar cada una de estas regiones que contengan valores del color que vamos a segmentar, finalmente unimos estas 3 búsquedas con una multiplicación elemento por elemento para mostrar el resultado final como se puede observar en la Figura 4

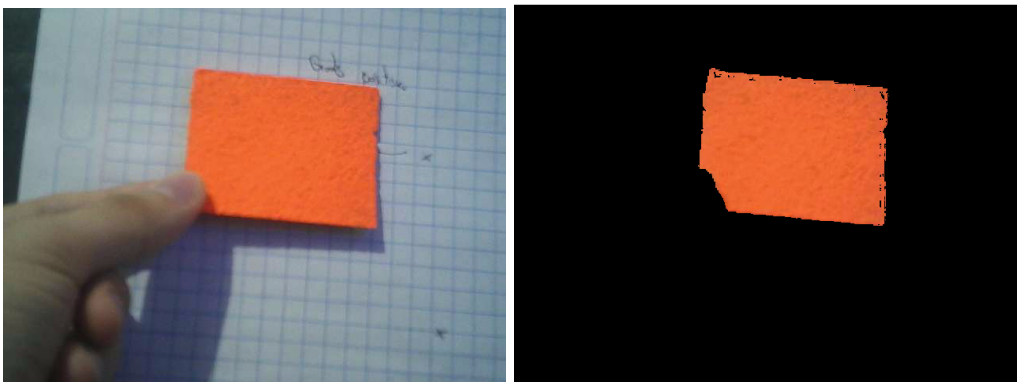

Figura 4: Resultado de segmentar la imagen.

Centroide y diferencias

Necesitamos conocer el centroide de nuestro objeto segmentado para posteriormente conocer la diferencia del centro de nuestra imagen con el centroide del objeto, de este modo tenemos la diferencia en píxeles lo que nos ayudara más adelante para el control de los motores. Por ahora, el modo de dibujar el centro de nuestro objeto segmentado es muy simple: solo necesitamos conocer la posición en x y y del promedio de los valores donde se segmento el color comparado con la imagen completa seg-

mentada, de este modo se aísla el objeto segmentado y obtenemos las coordenadas de este solamente.

Para las coordenadas del centro de la imagen solo hay que tomar en cuenta la resolución con la que estamos trabajando (640x480) y dibujar la mitad de cada eje (320, 240). La diferencia es las coordenadas del centro de nuestra imagen menos las coordenadas del centroide antes calculado. El dibujo de estos se observa de en la Figura 5.

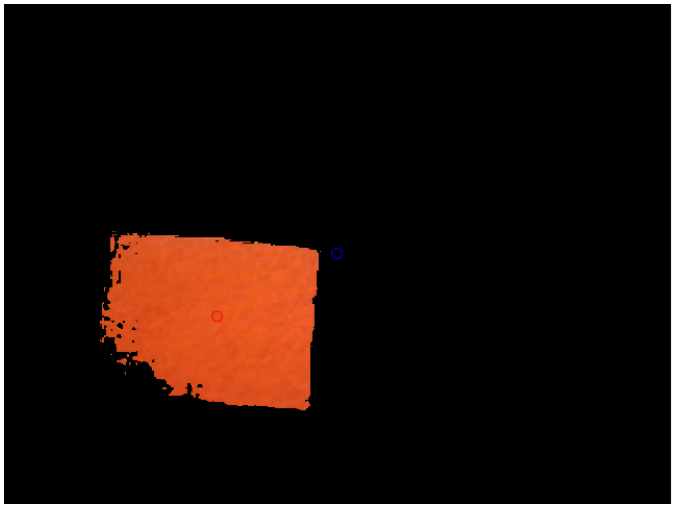

Figura 5: Dibujo de centroides.

Finalmente, para siempre estar capturando imágenes en tiempo real y segmentarlas encerramos todo el código realizado hasta ahora en un *while* con la condición *true* para que nunca termine.

Algo que también se hizo fue el añadirle ruido a la imagen para probar algún tipo de filtro que nos elimine lo más posible este defecto en la imagen. Si trabajamos con el ruido la segmentación del objeto es muy mala, además de confundirse con el fondo y segmentar partes de este por el mismo ruido. El filtro que usamos fue una ventana Gaussiana (pasa bajas). De este modo la imagen salia con perdida de información, pero el filtro llenaba esa información suavizando la imagen y eliminando el ruido. Es así que la segmentación se logro de manera exitosa una vez más (ver sección de Resultados para ver las imágenes con ruido, con filtro y segmentadas).

Construcción de la base con motores y cámara

Si queremos que la cámara pueda seguir al objeto segmentado en un plano de x y y necesitaremos de dos motores los cuales unidos a una base se encarguen de girar uno en x y otro en el eje y según sea lo necesario. En nuestro caso, estuvimos buscando en tiendas de electrónicos algún tipo de base ya armada o para armar que nos diera los requerimientos anteriormente descritos, afortunadamente encontramos una base para armar en la cual se montan dos servomotores, el servomotor que gira en x se pega a la base para que pueda girar esta libremente 180°, el servomotor y va unido al otro motor por medio de una bisagra que ayuda a sostener la cámara que montaremos, la base armada sin la cámara puede observarse en la Figura 6.

Figura 6: Base que sostendrá la cámara armada.

El problema fue montar la cámara, ya que el espacio que este incluía para la cámara era muy pequeño y la nuestra venía con un clip muy grande, Figura 7. En nuestro caso la cámara podía desprenderse de este clip quitando un tornillo pero aún así la cámara era muy grande para el espacio que teníamos.

Figura 7: Cámara web que utilizamos.

Solo pegarla con silicona no iba a ser suficiente, necesitábamos de algo más fuerte que sostenga la cámara. Nuestra solución fue hacerle dos agujeros a las pinzas donde iría la cámara para después

ingresar un alambre que pase por el orificio donde iba el tornillo que habíamos retirado y que se sostuviera de los dos agujeros previamente hechos, para al final solo pegarlo con silicona y asegurar un agarre perfecto, se puede observar el resultado de este en la Figura 8.

Figura 8: Cámara unida a la base giratoria.

Movimiento de los motores

Primeramente para poder mover nuestros motores con la señal de control deseada, necesitamos de un microcontrolador, nosotros optamos el *Arduino UNO* ya que es en el que tenemos más experiencia en uso además de ser completamente compatible con *Matlab*. La inicialización del Arduino es muy sencilla, con tan solo previamente haber instalado el paquete de instalación *MATLAB Support Package para Arduino* e inicializar un objeto como se observa a continuación.

```
p = arduino('COM3','UNO','Libraries','Servo');
```

Básicamente estamos dándole las propiedades del modelo de nuestro Arduino, el puerto donde esta conectado y las librerías que utilizaremos. Utilizamos la librería *Servo* para controlar la posición de los servomotores.

Los servomotores deben de inicializarse en *Matlab* especificando el nombre que le queramos dar, al Arduino que esta conectado y el pin que le asignemos. Como necesitamos de señales PMW utilizamos los pines 6 y 9.

```
servo_x = servo(p,'D6');
servo_y = servo(p,'D9');
```

El movimiento de los servomotores fue un poco más complicado de entender, ya que estos se mueven de grados teniendo un punto fijo para los 0°, 90° y 180°, pero además de esta condición, estos tienen que darse en el código de manera normalizara para un rango de 180°, es decir, el valor que le daremos para moverse en grados debe de ser un valor de 0 a 1, donde 1 representa 180°.

```
angulox = 90/180;
anguloy = 90/180;
writePosition(servo_x,angulox);
writePosition(servo_y,anguloy);
```

En este caso, 0.5 representa un movimiento a 90°, por lo que nuestro primer ajuste fue establecer ambos motores a 90° y después ajustar la base para que a estos grados la base se encuentre completamente derecha.

Una vez establecido el ángulo de inicio de nuestros motores, necesitábamos encontrar una relación que partiendo de la diferencia de los centroides previamente calculados, nos diera el valor en grados que los motores necesitan moverse para eliminar esta diferencia de centros. Nuestra solución fue hacer una regla de tres en la cual sabiendo cuantos píxeles equivale el movimiento de un grado de nuestro motor, podamos saber cuantos grados equivale x cantidad de píxeles. Para lograr esto, sabiendo la posición del centro de nuestra imagen, movimos el motor en x 10° grados y utilizando la función de *Matlab* llamada *imtool()* medimos la cantidad de píxeles que recorrió, siendo en este caso 161px, entonces para 1° son 16.1px.

$$\begin{aligned} 1^\circ &\rightarrow 16.1\text{px} \\ X &\rightarrow \text{Diferencia de píxeles en } x \end{aligned} \tag{1}$$

donde X es la cantidad de grados que tiene que moverse, por lo que $X = \frac{\text{Diferencia de píxeles en } x}{16.1\text{px}}$.

Para el eje y ejecutamos la misma tarea y encontramos que para 10° el servomotor se mueve 180px, por lo que para 1° el motor recorre 18px:

$$\begin{aligned} 1^\circ &\rightarrow 18\text{px} \\ Y &\rightarrow \text{Diferencia de píxeles en } y \end{aligned} \tag{2}$$

donde Y es la cantidad de grados que tiene que moverse, por lo que $Y = \frac{\text{Diferencia de píxeles en } y}{18\text{px}}$.

Con las relaciones encontradas para la cantidad de movimiento que deben realizar los motores solo nos falta establecer la dirección de su movimiento, es decir, que condiciones requiere para moverse hacia arriba, abajo, izquierda o derecha.

Las relaciones que encontramos son las siguientes:

- Moverse hacia arriba
  1. La diferencia de píxeles en eje y debe ser positiva.
  2. Este valor debe de restarse al ángulo actual que tenga el servomotor.
- Moverse hacia abajo
  1. La diferencia de píxeles en eje y debe ser negativa.
  2. Este valor debe de sumarse al ángulo actual que tenga el servomotor.
- Moverse hacia la derecha
  1. La diferencia de píxeles en eje x debe ser negativa.
  2. Este valor debe de restarse al ángulo actual que tenga el servomotor.
- Moverse hacia la izquierda
  1. La diferencia de píxeles en eje x debe ser positiva.
  2. Este valor debe de sumarse al ángulo actual que tenga el servomotor.

Por lo que solo definimos distintas condiciones partiendo de si la diferencia tanto en x como en y sea positiva o negativa, las condiciones finales para establecer los ángulos fueron:

```
%% Movimiento en x
if diferenciax > 0
    angulox = (anguloAntesx + (diferenciax/16.1)/180) ; % IZQUIERDA
else
    angulox = (anguloAntesx - (-diferenciax/16.1)/180) ; % DERECHA
end

%% Movimiento en y
if diferenciay > 0
    anguloy = (anguloAntesy - (diferenciay/18)/180); % ARRIBA
else
    anguloy = (anguloAntesy + (-diferenciay/18)/180); % ABAJO
end
```

Algo que agregamos fue una condición extra en la cual verifica si el valor de la diferencia en x o en y no es un valor numérico se establezca la posición de los servomotores a 90°. De este modo si no encuentra el objeto seleccionado anteriormente la posición del motor se regresa a los 90°. Al final no nos convenció mucho esta opción ya que por lo cambios de la iluminación a veces el objeto no era detectado a pesar de estar frente a la cámara, por lo que ingresamos una condición más, si después de 10 frames el objeto no es detectado de nuevo los motores vuelven a los 90°, esto le dio más tiempo a la cámara para volver a detectar el objeto y no restablecer la posición cuando no es necesario.

Control PID para los motores

Un controlador PID esta compuesto de tres elementos principales, una acción proporcional, otra integral y una derivativa [3]. Con una retroalimentación conseguimos el error entre el estado deseado y el conseguido, este es el error que tomara parte den nuestra acción de control para eliminar el error completamente [3].

Para la acción de control proporcional buscamos hacer más rápida la respuesta del sistema y eliminar el error que encontramos en estado estacionario, el problema con esta acción de controlo es que si la aumentamos demasiado el sistema puede volverse inestable [1]. Con la acción integral disminuimos aun más el error que encontramos y para la integral le damos más estabilidad al sistema.

Para implementar el control PID en nuestro sistema de motores necesitamos hacer uso del error que tenemos en la posición deseada en grados a la alcanzada de nuestros motores. Como comienzo definimos una ganancia K_p muy pequeña para hasta hacer que el sistema se comporte a la velocidad que queremos, para nuestro caso después de prueba y error la ganancia K_p la definimos como 0.5 para el eje x y de 0.35 para el eje y. De este modo visualizamos un respuesta mucho más suave al movimiento de la cámara pero un poco lento a la hora de hacer movimiento buscos al objeto pero no mucho. Para la ganancia integral y derivativa, para ambos ejes, escogimos una ganancia muy baja ya que al hacerla más grande el movimiento de los motores se descontrolaba demasiado.
Ahora solo queda calcular la acción de controlo. El termino P solo se multiplica el error por nuestra ganancia K_p, nuestro componente integral es la multiplicación de la ganancia K_i por el error que hemos acumulado hasta ese momento y finalmente para el componente derivativo necesitamos multiplicar nuestra ganancia K_d por el cambio de error de un movimiento a otro.

$$
\begin{aligned}
Kp_x &= 0.5 \\
Ki_x &= 0.0001 \\
Kd_x &= 0.001 \\
Kp_y &= 0.35 \\
Ki_y &= 0.0001 \\
Kd_y &= 0.0004
\end{aligned}
$$

Las ganancias para la acción de control en el eje y fue menor ya que el motor recorre más píxeles sobre este mismo. El calculo de cada termino, P, I y D se realizo de la siguiente forma:

```
%% CALCULO COMPONENTES PID x
Px = diferenciax * Kp_x;
Ix = errorAcumx * time_ex * Ki_x;
Dx = ((diferenciax - errorIntx)/time_ex) * Kd_x;
controlx = Px + Ix + Dx;

%% CALCULO COMPONENTES PID y
Py = diferenciay * Kp_y;
Iy = errorAcumy * time_ex * Ki_y;
Dy = ((diferenciay - errorInty)/time_ex) * Kd_y;
controly = Py + Iy + Dy;
```

donde la variable *time_ex* representa el tiempo que lleva actualmente en ejecución, la acción de controlo a la que llegamos es el nuevo movimiento que realizaran nuestros motores.

Resultados

Imagenes

La cámara montada en la base con los motores finalmente nos quedo como se ve en la Figura 9.

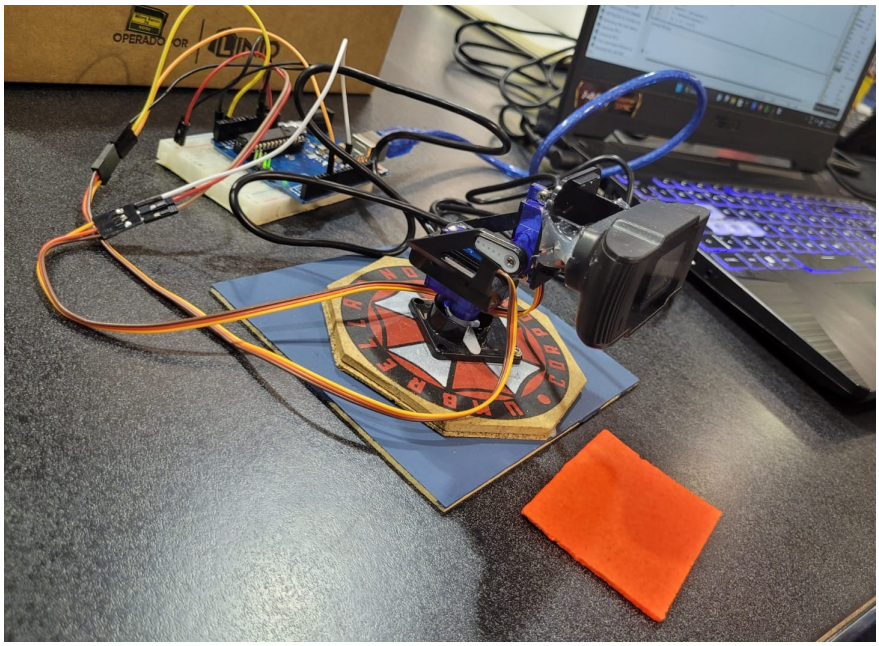

Figura 9: Prototipo final.

La imagen de entrada es (Figura 10):

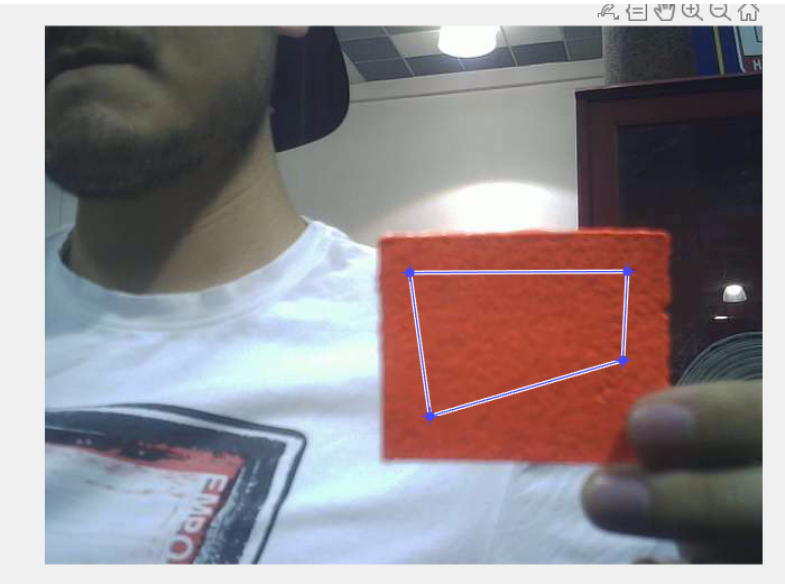

Figura 10: Imagen de entrada.

Cuando a la imagen se le aplica ruido, la imagen filtrada y con ruido es (Figura 11):
La imagen de salida ya filtrada y segmentada (Figura 12):

Figura 11: Imagen con ruido y filtrada respectivamente.

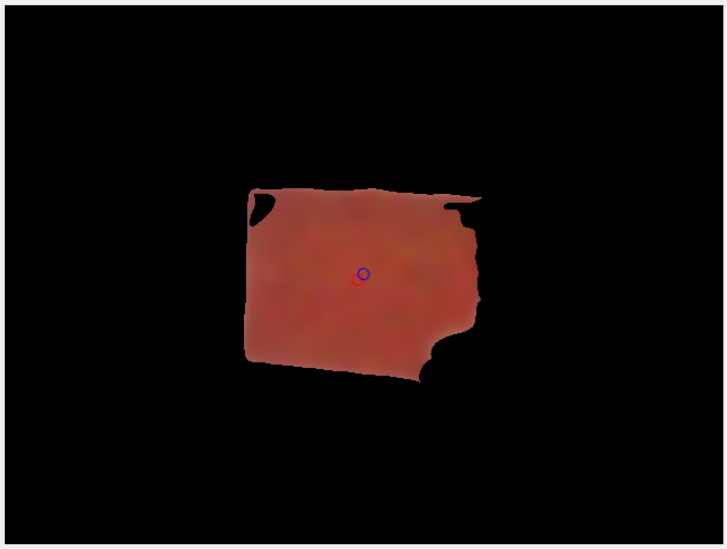

Figura 12: Imagen de salida.

Graficas

Las gráficas de error tanto en el eje x como en y representan la posición a la que llegamos y la posición de referencia:

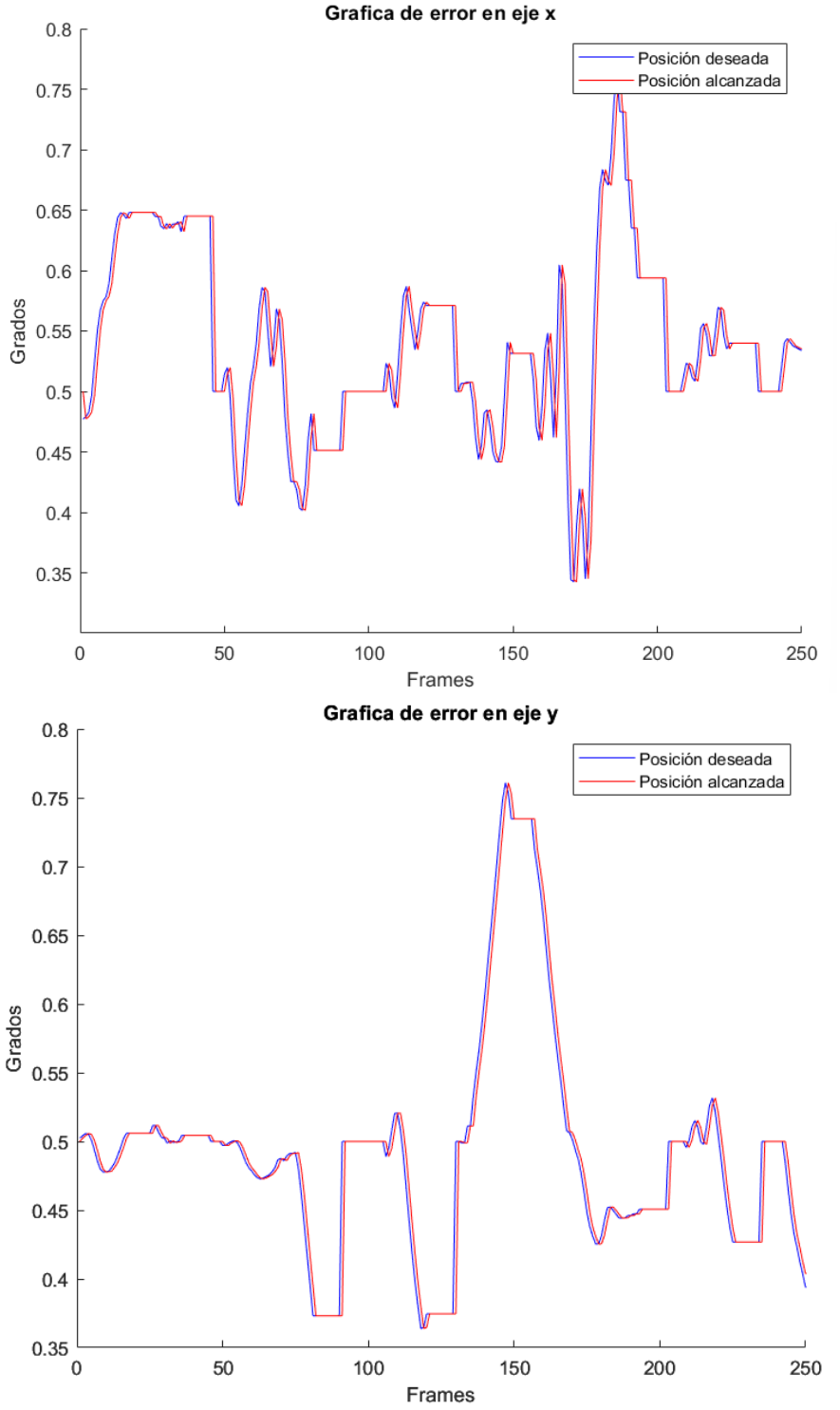

Figura 13: Gráficas de error en x y y.

Videos

Para ver el proceso de la construcción del proyecto y del resultado final en vídeo, visite los siguientes enlaces:

- Movimiento de motores sin control:
  `https://drive.google.com/file/d/1V1rJRi1JonNvFab4kbHfTpClcq5dklbI/view?usp=sharing`
- Movimiento con control:
  `https://drive.google.com/file/d/1KLsqcHHeqxyybiV6KHMxBxNjjsvVZved/view`
- Base armada y conectada al *Arduino UNO*:
  `https://drive.google.com/file/d/1KEkQwk9Yc4K-ShVTqXDDyz3YIozlfj26/view?usp=share_link`

Conclusiones y puntos de mejora

Error: Al final obtuvimos un error bastante pequeño y aceptable pero creemos que podemos hacer algo mas eficiente, de manera que su corrección sea mas rápida, el error aún más pequeño y por lo tanto un seguimiento más estable.

Base: Nuestra base fue creada con caucho y madera, quedo una base bastante ligera y poco anti-derrapante, por lo que debemos sujetarla para evitar algún movimiento no deseado, seria un punto a mejorar colocarle una base con mayor peso y una mejor superficie para evitar cualquier tipo de movimiento a causa del seguimiento de los motores.
Esto no afecto en nuestros resultados finales pero es algo que si mejoramos no debemos de preocuparnos de que sufra movimientos externos no deseados.

Control: Si bien logramos tener un buen control, para errores pequeños nos generaba oscilaciones de manera que fue prudente agregar una función para evitar movimientos pequeños, de no haber puesto esta función observaríamos muchas movimientos bruscos en todo momento. Las ganancias jugaron una parte muy larga y nos dejo con dudas de si la implementación fue la correcta o no.

Objeto a seguir: En este caso al igual que la base nuestro objeto esta hecho de caucho, no nos percatamos hasta el final que en ambientes de mucha luz se veía afectada la percepción que tenia en la cámara, por lo que movía el centroide dependiendo de como estuviera recibiendo la luz, esto también nos afecto en el error, pero en un ambiente controlado daba buenos resultados.

Conclusión general: Logramos un proyecto que nos diera buenos resultados, si bien hay varios puntos a mejorar en base a nuestras investigaciones, clases y conocimiento logramos aplicar la visión artificial a nuestro sistema con puntos a mejorar más que nada estéticos.

Referencias y Bibliografía

[1] Franklin Electric. *¿Qué es el control PID?* 2022, 17 febrero. URL: https://franklinelinkmx.wordpress.com/2022/02/16/que-es-el-control-pid/.

[2] MATLAB Simulink - MathWorks América Latina. *Funciones básicas de representación gráfica.* (s.f.) URL: https://la.mathworks.com/help/matlab/learn_matlab/basic-plotting-functions.html.

[3] Picuino. *Controlador PID - Control Automático.* (s.f.) URL: https://www.picuino.com/es/control-pid.html.

[4] Colaboradores de Wikipedia. *Visión artificial.* 2022b. URL: https://es.wikipedia.org/wiki/Visi%5C%C3%5C%B3n_artificial.