

# Control Seguimiento Laser vía PID y OpenCV

UTN FRA - LEIFRA

1<sup>st</sup> Sergio Pereira  
sergio\_pereira\_89@hotmail.com,

*Abstract*—El contexto actual de inseguridad nos plantea implementar nuevas soluciones a la seguridad hogareña dentro del marco de la ley y que nos brinde tranquilidad en el uso de armas no letales como medio de disuasión.

*Index Terms*—Control PID, Control Posicion, Motor DC, Laser, Encoder Magnetico, Seguridad.

## CONTENTS

<b>I Introducción</b>	1	<b>IX Pruebas de seguimiento</b>	14
<b>II Marco Teórico</b>	1	<b>IX-A Motores</b> . . . . .	14
<b>II-A Control PID</b>	2	<b>IX-B Variables mostradas</b> . . . . .	14
<b>II-B Acción proporcional</b>	2	<b>IX-C Seguimiento con opencv</b> . . . . .	15
<b>II-C Acción integral</b>	2		
<b>II-D Acción derivativa</b>	2		
<b>III Hardware</b>	3	<b>X Conclusiones</b>	15
<b>III-A Motores</b>	3	<b>References</b>	15
<b>III-B Encoders</b>	3		
<b>III-C Microprocesador</b>	3	<b>Appendix</b>	15
<b>III-D Puente H</b>	4	<b>A Programa Microprocesador ESP32</b> . . . . .	16
<b>III-E Adaptador de niveles</b>	4	<b>B Programa Python Opencv</b> . . . . .	22
<b>III-F Fuente Stepdown</b>	4	<b>C Esquemático</b> . . . . .	23
<b>III-G Display VFD</b>	5		
<b>IV Seguimiento</b>	5		
<b>IV-A Laser</b>	5	<b>I. INTRODUCCIÓN</b>	
<b>IV-B Cámara</b>	5	Se plantea como solución un sistema de seguimiento laser	
<b>IV-C OpenCV</b>	5	utilizando como posicionamiento dos motores de corriente	
<b>V Planta</b>	5	continua controlados vía PID con lazo de realimentación,	
<b>V-A Lazos de control</b>	5	encoders de posicionamiento angular vertical y horizontal.	
<b>V-B Esquemático</b>	6	A sí mismo, el sistema de posicionamiento es controlado	
<b>VI Controlador PID</b>	6	vía software con lazo de realimentación, una cámara web	
<b>VI-A Modelización</b>	6	implementando detección de objetos.	
<b>VI-B Diseño de Controlador PID con MATLAB</b>	7		
<b>VI-C Control en cascada</b>	9		
<b>VI-D Control PID directo</b>	10		
<b>VII Software</b>	10		
<b>VII-A Código ESP32</b>	10		
<b>VII-B Código Python</b>	12		
<b>VIII Construcción del Proyecto</b>	13		
<b>VIII-A Soporte as5600</b>	13		
<b>VIII-B Soporte Motores</b>	14		
<b>VIII-C Montaje del proyecto</b>	14		

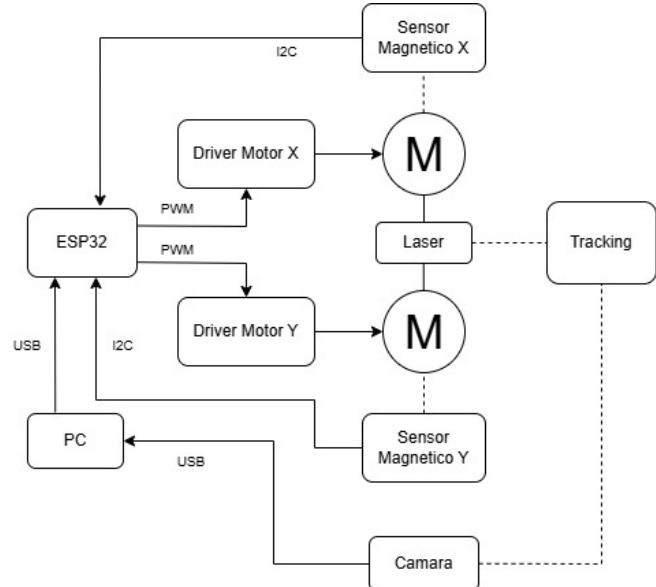


Fig. 1. Diagrama en bloques

## II. MARCO TEÓRICO

El controlador PID es la solución más común a los problemas prácticos de control. Aunque controladores con acción

proporcional e integral han sido utilizados desde la época en que los molinos de viento y las máquinas de vapor eran las tecnologías dominantes, la forma actual del controlador PID emergió con los controladores neumáticos en los años 30 del siglo pasado. Una razón fue que las realizaciones con computadores hizo posible añadir características tales como capacidad de auto sintonía y diagnóstico, que son muy beneficiosas para los usuarios.

### A. Control PID

El algoritmo PID se puede describir como:

$$u(t) = K \left( e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right),$$

donde:

- $u(t)$  es la señal de control
- $e(t)$  es el error de control  $e = y_{sp} - y$

La señal de control es así una suma de tres términos:

- el término-P (que es proporcional al error)
- el término-I (que es proporcional a la integral del error)
- el término-D (que es proporcional a la derivada del error)

Los parámetros del controlador son:

- la ganancia proporcional K
- el tiempo integral  $T_i$
- el tiempo derivativo  $T_d$

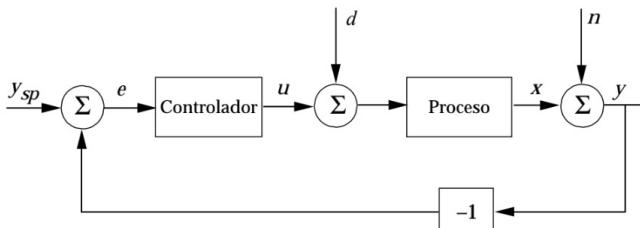


Fig. 2. Diagrama de bloques de un lazo de realimentación simple.

### B. Acción proporcional

En el caso del control proporcional puro, la ley de control dada por la Ecuación 3.1 se reduce a  $u_{(t)} = K * e_{(t)} + u_b$  la acción de control es simplemente proporcional al error de control. La variable  $u_b$  es una polarización o un reset. Cuando el error de control  $e$  es cero, la señal de control toma el valor  $u_{(t)} = u_b$ . La polarización  $u_b$  a menudo se fija a  $\frac{(u_{max}+u_{min})}{2}$ , pero puede algunas veces ajustarse manualmente de forma que el error de control estacionario es cero en un punto de consigna dado.

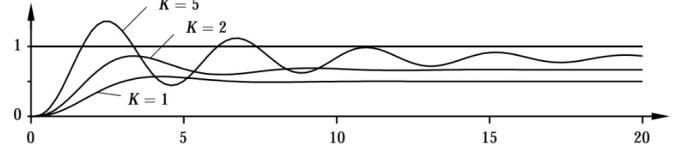


Fig. 3. Simulación de un sistema en lazo cerrado con control proporcional.

### C. Acción integral

La función principal de la acción integral es asegurarse de que la salida del proceso coincida con el punto de consigna en estado estacionario. Con control proporcional, hay normalmente un error de control en estado estacionario. Con acción integral, un pequeño error positivo conducirá siempre a una señal de control creciente, y un error negativo dará una señal de control decreciente sin tener en cuenta lo pequeño que sea el error. El siguiente argumento simple muestra que el error en estado estacionario siempre será cero con acción integral. Suponga que el sistema está en estado estacionario con una señal de control constante  $u_0$  y un error constante  $e_0$ . Se sigue de la ecuación del algoritmo PID que la señal de control viene entonces dada por

$$u_0 = K \left( e_0 + \frac{e_0}{T_i} t \right).$$

Mientras  $e_0 = 0$ , esto claramente contradice la hipótesis de que la señal de control  $u_0$  es constante. Un controlador con acción integral siempre dará error cero en estado estacionario.

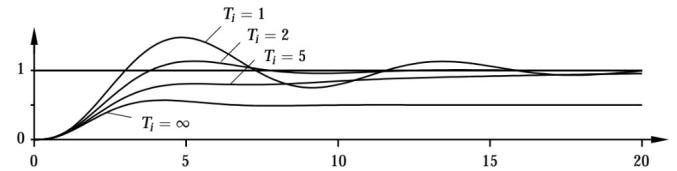


Fig. 4. Simulación de un sistema en lazo cerrado con control proporcional e integral.

y encontramos que:

$$u = K \left( e + \frac{1}{T_i} \int e(\tau) d\tau \right),$$

es un controlador PI.

### D. Acción derivativa

El objetivo de la acción derivativa es mejorar la estabilidad en lazo cerrado. El mecanismo de inestabilidad se puede describir intuitivamente como sigue.

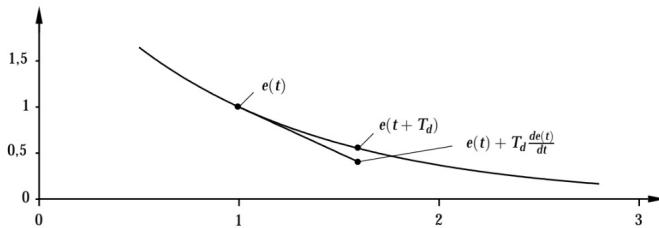


Fig. 5. Interpretación de la acción derivativa como control predictivo, donde la predicción se obtiene por extrapolación lineal

A causa de la dinámica del proceso, llevará algún tiempo antes de que un cambio en la variable de control sea detectable en la salida del proceso. Así, el sistema de control tardará en corregir un error. La acción de un controlador con acción proporcional y derivativa se puede interpretar como si el control se hiciese proporcional a la salida predicha del proceso, donde la predicción se hace extrapolando el error por la tangente a la curva del error. La estructura básica de un controlador PD es:

$$u(t) = K \left( e(t) + T_d \frac{de(t)}{dt} \right).$$

La señal de control es así proporcional a una estimación del error de control en el tiempo  $t_d$  hacia adelante, donde se obtiene la estimación por extrapolación lineal. En la siguiente figura se ilustran las propiedades de la acción derivativa, que muestra una simulación de un sistema con control PID, donde la ganancia del controlador es  $K = 3$ , y el tiempo integral es  $t_i = 2$ .

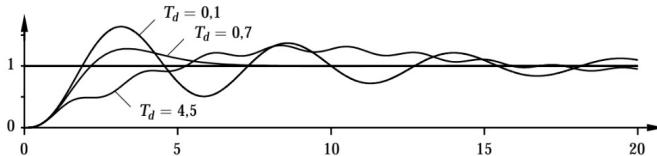


Fig. 6. Simulación de un sistema en lazo cerrado con control proporcional, integral, y derivativo.

### III. HARDWARE

#### A. Motores

Se utiliza Motores de corriente continua con escobillas de carbón, modelo RS-555SH que funciona a 24v con aplicaciones en compresores de aire, destornilladores eléctricos, molinillos de cafeteras, etc.



Fig. 7. RS-555SH

#### TORQUE SPEED CURVE

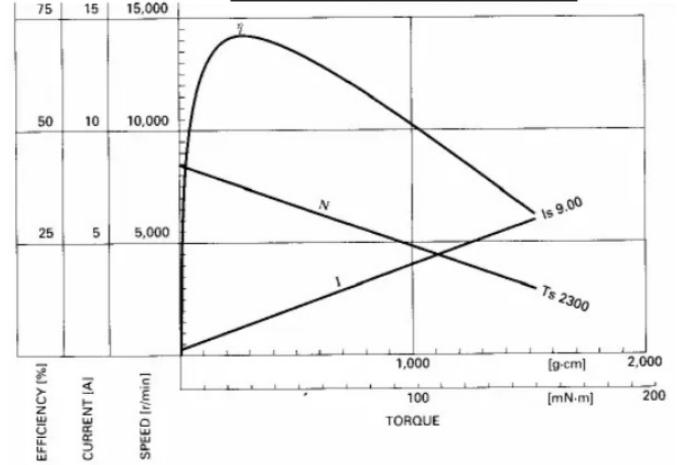


Fig. 8. Curva velocidad/torque motor

#### B. Encoders

El AS5600 es un sensor de posición giratorio magnético fácil de programar con una salida analógica o PWM de 12 bits de alta resolución. Este sistema sin contacto mide el ángulo absoluto de un imán diametralmente magnetizado en el eje. El AS5600 está diseñado para aplicaciones de potenciómetro sin contacto y su diseño robusto elimina la influencia de cualquier campo magnético externo homogéneo.

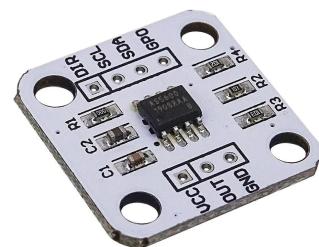


Fig. 9. Sensor AS5600

#### C. Microprocesador

El ESP32 WROOM-32E es un módulo versátil y potente construido en torno al chipset ESP32 de Espressif. Ofrece

procesamiento de doble núcleo Xtensa® 32-bit LX6, conectividad integrada Wi-Fi y Bluetooth, y cuenta con una amplia gama de interfaces periféricas. Conocido por su bajo consumo de energía, el módulo es ideal para aplicaciones de IoT, lo que permite una conectividad inteligente y un rendimiento robusto en formatos compactos.

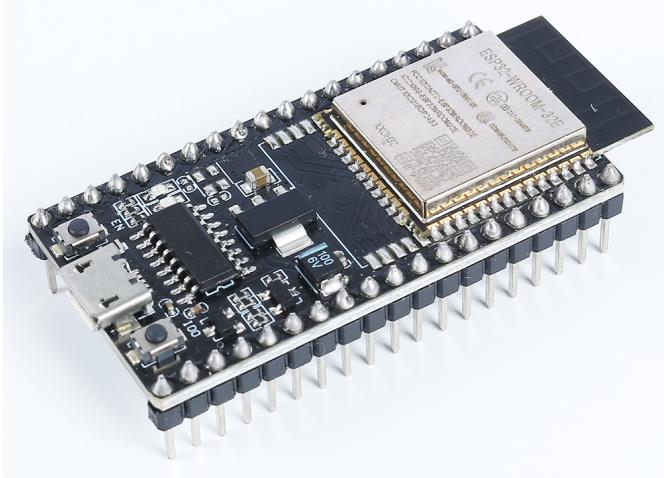


Fig. 10. ESP32 WROOM 32E

#### D. Puente H

El L298 [3] es un circuito monolítico integrado en un circuito de 15 pines en encapsulado Multiwatt. Es un Controlador de puente completo dual, de alto voltaje y alta corriente diseñado para aceptar niveles lógicos TTL estándar y conducir cargas inductivas como relés, solenoides, CC y motores paso a paso. Se proporcionan dos entradas de habilitación para habilitar o deshabilitar el dispositivo independientemente de las señales de entrada.



Fig. 11. Puente H L298n

#### E. Adaptador de niveles

Debido a que los encoders tienen una tensión de trabajo adecuada de 5V y el microcontrolador de 3.3V es necesario el montaje de un adaptador de niveles lógicos para evitar

quemar las salidas del micro y aun así lograr una correcta comunicación de los dispositivos.

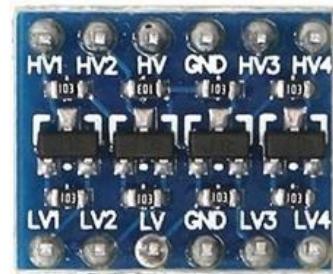


Fig. 12. Level Shifter

Módulo que utilizamos para adaptar niveles lógicos de tensión de 5v a 3.3v.

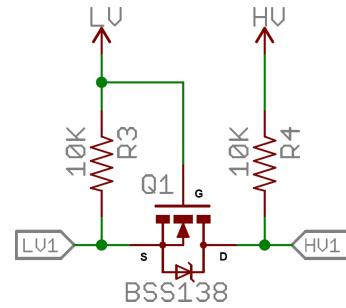


Fig. 13. Circuito Level Shifter

El circuito consta de un D-MOSFET canal N funcionando como llave entre las entradas de tensión. Con LV=3.3v, HV=5v, su lógica es que el gate está a un potencial constante, a un nivel alto de LV1 no hay diferencia de potencial entre gate y source, por lo cual el mosfet no conduce logrando un nivel alto en HV1. Lo mismo ocurre viceversa con un nivel alto de HV1. Cuando LV1 está a nivel bajo, se aplica un potencial a Vgs y el mosfet conduce logrando un nivel bajo en HV1. Por otro lado con un nivel bajo de HV1, el diodo de sustrato del MOSFET permite que el source también se desactive parcialmente debido a una pequeña cantidad de voltaje que cae a través del diodo, el VGS se eleva por encima del umbral y el MOSFET comienza a conducir de manera efectiva sin pasar por el diodo del sustrato logrando un nivel bajo en LV1.

#### F. Fuente Stepdown

Este circuito te permite tener un voltaje regulado a partir de una fuente de alimentación con un voltaje mayor. El módulo convertidor LM2596 es un regulador de tipo conmutado, así que su eficiencia es significativamente mayor en comparación con los populares reguladores lineales de tres terminales, especialmente con tensiones de entrada superiores. Especificaciones Técnicas:

- Voltaje de entrada: DC 3,2V-40V.
- Voltaje de salida: DC 1,25V-35V (Ajustable. La salida deberá ser al menos 1,5V mayor a la entrada).

- Corriente de salida: 3A (MAX. 2,5A recomendado para uso prolongado. Usar disipadores si se superan los 2A).
- Potencia de salida: 70W (MAX). Se recomienda disipar a partir de los 50W.
- Eficacia de conversión: el 92
- Ripple:  $\pm 30\text{mV}$ .
- Frecuencia de switching: 65Khz.



Fig. 14. Fuente Step Down

#### G. Display VFD

Con el objetivo de mostrar el seguimiento angular en tiempo real se utilizará un display vfd. El acrónimo VFD, del inglés Vacuum Fluorescent Display, refiere a las pantallas fluorescentes de vacío. Consisten en una ampolla de vidrio que contiene uno o varios filamentos que actúan de cátodo, varios ánodos recubiertos de fósforo y una rejilla por carácter. Al polarizar positivamente los ánodos y las rejillas, los electrones emitidos por el cátodo alcanzan un ánodo, que se ilumina.



Fig. 15. Display VFD

#### IV. SEGUIMIENTO

##### A. Laser

La Gmconn P10 es una mira láser con una combinación ultracompacta de luz y haz verde. A pesar de su diseño compacto, ofrece 650 lúmenes de luz blanca brillante, junto

con un haz verde de alta visibilidad para mayor precisión y más opciones para elegir entre luz, haz verde o haz verde y luz combinados.



Fig. 16. Laser Gmconn P10

##### B. Cámara

El sistema de seguimiento constará de su propia cámara web colocada lo más cercano posible al laser a fin de evitar una mayor corrección por deriva horizontal y vertical.

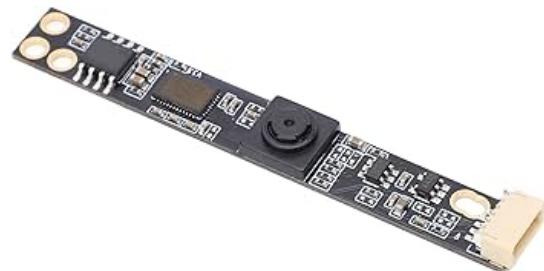


Fig. 17. Camara Web

En nuestra aplicación será una cámara web VGA de 640x480 pixeles.

##### C. OpenCV

OpenCV (Open Source Computer Vision Library) es una biblioteca de funciones de programación, principalmente para visión artificial en tiempo real.



Fig. 18. OpenCV logo

#### V. PLANTA

##### A. Lazos de control

El sistema de control consta de 2 lazos cerrados PID para el posicionamiento X e Y, ambos controlados por el

microprocesador, y un lazo abierto para el seguimiento en tiempo real controlado por la aplicación python utilizando opencv.

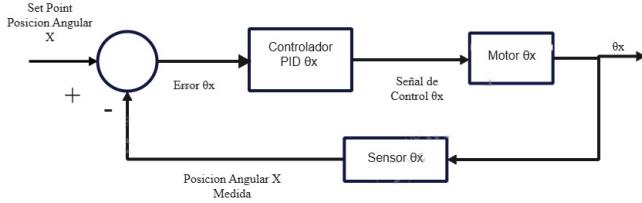


Fig. 19. Lazo de control ejeX

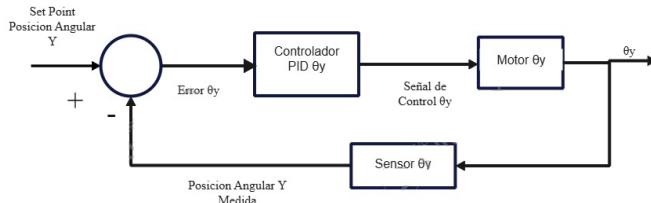


Fig. 20. Lazo de control ejeY

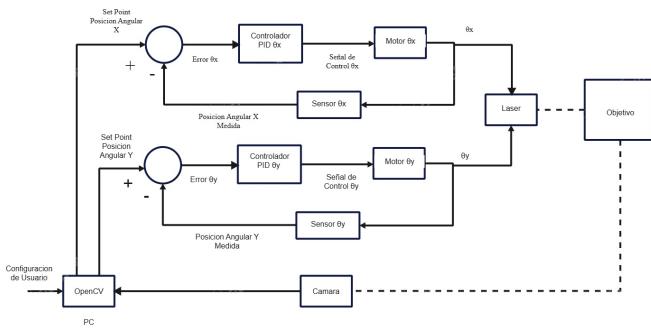


Fig. 21. Lazo de control completo

### B. Esquemático

Se adjunta esquemático como anexo al final del documento y se enlaza el hipervínculo en la siguiente imagen al github del proyecto para poder visualizar/descargar el archivo pdf.

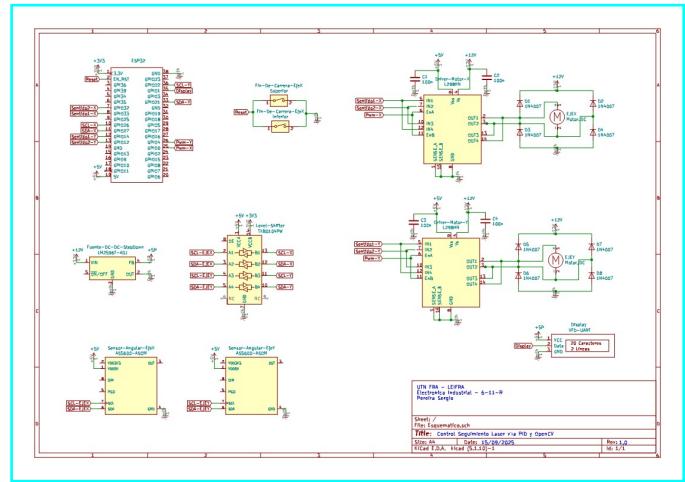
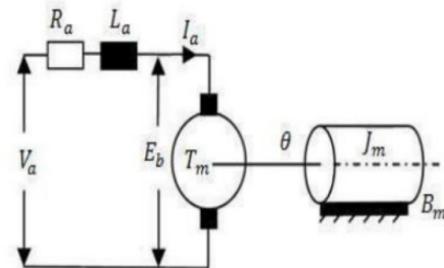


Fig. 22. Esquemático

## VI. CONTROLADOR PID

### A. Modelización

Un actuador común en los sistemas de control es el motor de CC. Este proporciona directamente movimiento rotatorio y, acoplado a ruedas o tambores y cables, puede proporcionar movimiento de traslación. El circuito eléctrico equivalente de la armadura y el diagrama de cuerpo libre del rotor se muestran en la siguiente figura.



$$V_a = R_a * I_a(t) + L_a * \frac{di_a(t)}{dt} + e_b(t) \quad (1)$$

$$e_b(t) = K_b * w(t) \quad (2)$$

$$T_m(t) = K_t * i_a(t) \quad (3)$$

$$T_m(t) = J_m * \frac{dw(t)}{dt} + B_m * w(t) \quad (4)$$

Donde:

- $V_a$  = Voltaje de Armadura (V)
- $R_a$  = Resistencia de Armadura ( $\Omega$ )
- $L_a$  = Inductancia de Armadura (H)
- $i_a$  = Corriente de Armadura (A)
- $e_b$  = Campo electromagnético inverso (V)
- $w$  = Velocidad angular (Rad/s)
- $T_m$  = Torque del motor (Nm)
- $\phi$  = Posicion Angular del rotor (Rad)
- $J_m$  = Inercia del Rotor (Kg-m<sup>2</sup>)
- $B_m$  = Coeficiente de Friccion (Nms/Rad)

- $K_t$  = Constante de Torque (N-m/A)
- $K_b$  = Constante de Campo electromagnético inverso (V/Rad)

Combinando las ecuaciones anteriores:

$$v_a = R_a * i_a(t) + L_a * \frac{di_a(t)}{dt} + K_b * W(t) \quad (5)$$

$$K_t * i_a(t) = J_m * \frac{dW(t)}{dt} + B_m * W(t) \quad (6)$$

Aplicando la transformada de Laplace, la función de transferencia entre la posición del eje y la tensión de armadura en vacío es:

$$\frac{\phi(s)}{v_a(s)} = \frac{K_t}{(L_a J_a)s^3 + (R_a J_m + L_a B_m)s^2 + (R_a B_m + K_b K_t)s} \quad (7)$$

### B. Diseño de Controlador PID con MATLAB

(Basándonos en los ejemplos de tutoriales de control para Matlab y Simulink.)

Identificada la planta se procede al diseño del controlador bajo una referencia de 1 radian y los criterios de diseño son:

- Tiempo de estabilización de 40mseg
- Sobrevalor (overshoot) inferior al 16%
- Sin error en régimen estacionario, incluso ante una perturbación.

En MATLAB creamos un nuevo archivo .m e ingresamos los siguientes comandos para ingresar los valores de la planta y definir la función transferencia.

```
>> J = 3,2284E - 6;
>> b = 3,5077E - 6;
>> K = 0,0274;
>> R = 4;
>> L = 2,75E - 6;
>> s = tf('s');
>> P(motor) = K/(s*((J*s+b)*(L*s+R)+K^2));
```

Recuerde que la función de transferencia para un controlador PID tiene la siguiente forma:

$$C(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s} \quad (8)$$

#### Control Proporcional

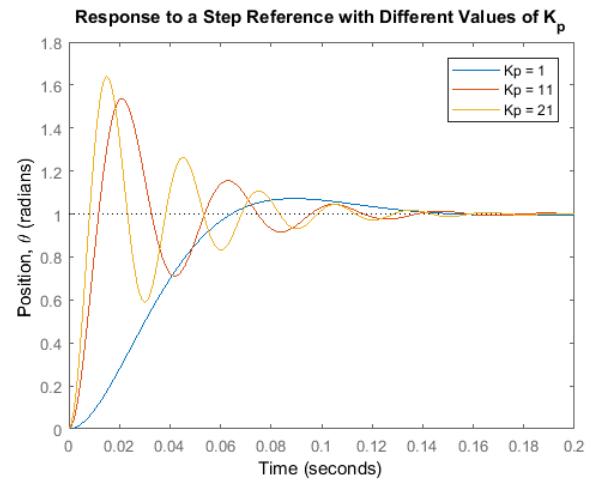
Primero, probemos con un controlador proporcional con una ganancia que va de 1 a 21. Se puede crear un arreglo de modelos LTI (lineales invariantes en el tiempo), cada uno con una ganancia proporcional diferente, usando un bucle ‘for’ . Las funciones de transferencia de lazo cerrado se pueden generar con el comando ‘feedback’.

```
>> Kp = 1;
>> for i = 1 : 3
>>     C(:,:,i) = pid(Kp);
>>     Kp = Kp + 10;
>> end
```

```
>> sys_cl = feedback(C * P(motor), 1);
```

Veamos ahora cómo son las respuestas escalón, armamos la gráfica con los siguientes comandos.

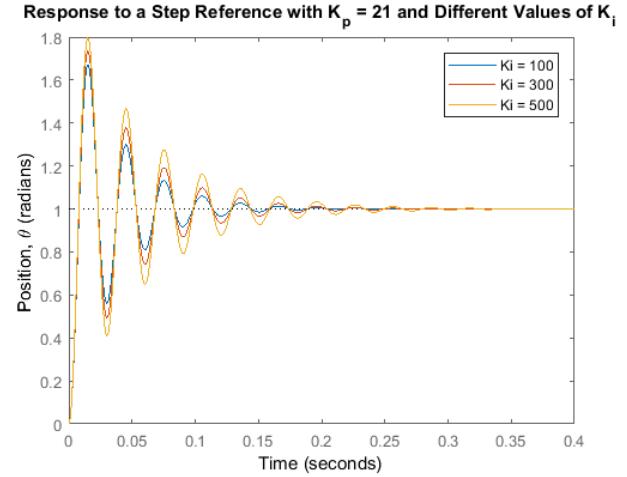
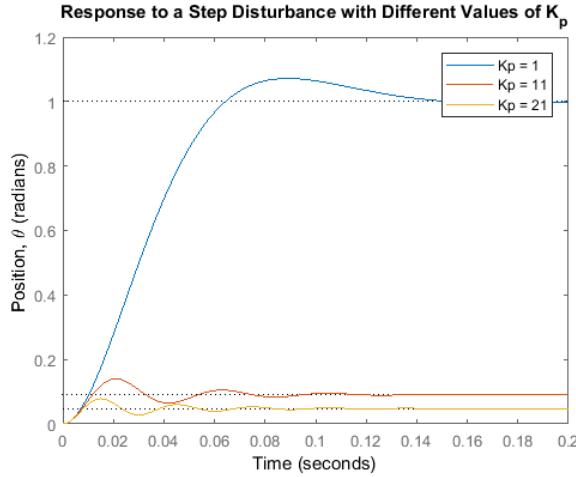
```
>> t = 0 : 0.001 : 0.2;
>> step(sys_cl(:,:,1), sys_cl(:,:,2), sys_cl(:,:,3), t)
>> ylabel('Posición, θ(radianes)')
>> title('Respuesta con diferentes valores de Kp')
>> legend('Kp = 1', 'Kp = 11', 'Kp = 21')
```



Observamos que no presenta error estacionario, independiente del valor de la ganancia proporcional, esto se debe a la planta al ser de tipo 1, se dice que la planta cuenta con un integrador.

Obtenida la respuesta al escalón evaluamos la respuesta frente a una perturbación, tomando como referencia de cero. El comando de realimentación aún puede emplearse para generar la función de transferencia en lazo cerrado, donde persiste la realimentación negativa; sin embargo, ahora solo la función de transferencia de la planta  $P(s)$  se encuentra en la trayectoria directa, mientras que el controlador  $C(s)$  se considera en la trayectoria de realimentación.

```
>> dist_cl = feedback(P(motor), C);
>> step(dist_cl(:,:,1), dist_cl(:,:,2), dist_cl(:,:,3), t)
>> ylabel('Posición, θ(radianes)')
>> title('Respuesta a perturbación valores de Kp')
>> legend('Kp = 1', 'Kp = 11', 'Kp = 21')
```



El sistema presenta un error significativo en estado estacionario al añadir la perturbación. Específicamente, la respuesta ante la referencia y la perturbación aplicadas simultáneamente es igual a la suma de las dos gráficas mostradas. Esto se deduce de la propiedad de superposición que se cumple para sistemas lineales. Por lo tanto, para obtener un error en estado estacionario nulo ante una perturbación, es necesario que la respuesta a la perturbación tienda a cero. Cuanto mayor sea el valor de  $K_p$ , la ganancia proporcional, menor será el error en estado estacionario debido a la perturbación, pero este nunca llega a cero. Además, el uso de valores cada vez mayores de la ganancia proporcional  $K_p$  tiene el efecto adverso de aumentar la sobre oscilación y el tiempo de estabilización, como se observa en la gráfica de la referencia escalón, añadir un término integral elimina el error en estado estacionario y un término derivativo puede reducir la sobre oscilación y el tiempo de estabilización.

### Control PI

Primero, probemos un controlador PI para eliminar el error en estado estacionario debido a la perturbación. Estableceremos  $K_p = 21$  y probaremos ganancias integrales  $K_i$  que van de 100 a 500.

```

>> Kp = 21;
>> Ki = 100;
>> for i = 1 : 5
>>     C(:, :, i) = pid(Kp, Ki);
>>     Ki = Ki + 200;
>> end
>> sys_cl = feedback(C * P_motor), 1);
>> t = 0 : 0.001 : 0.4;
>> step(sys_cl(:, :, 1), sys_cl(:, :, 2), sys_cl(:, :, 3), t)
>> ylabel('Posición, θ(radianes)')
>> title('Respuesta a escalon con Kp = 21 y valores de Ki')
>> legend( 'Ki = 100' , 'Ki = 300' , 'Ki = 500' )

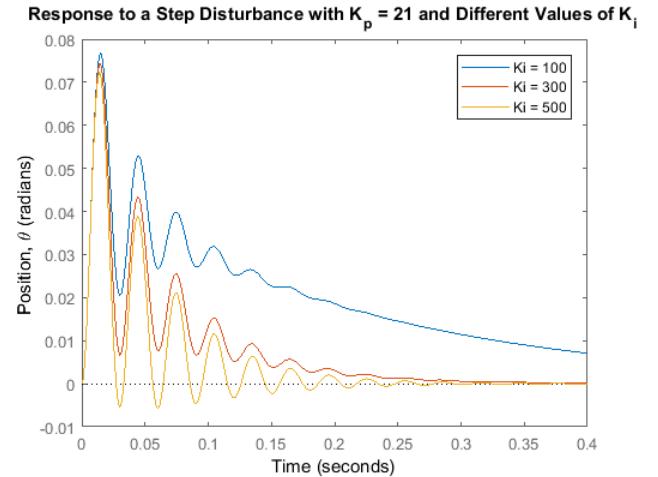
```

Veamos ahora qué ocurrió con la respuesta a la perturbación escalón.

```

>> dist_cl = feedback(P_motor), C);
>> step(dist_cl(:, :, 1), dist_cl(:, :, 2), dist_cl(:, :, 3), t)
>> ylabel( 'Posición, θ(radianes)' )
>> title('Respuesta a perturbacion con Kp = 21 y valores de Ki')
>> legend( 'Ki = 100' , 'Ki = 300' , 'Ki = 500' )

```



El control integral ha reducido el error en estado estacionario a cero, incluso ante una perturbación escalón; este era el objetivo al añadir el término integral. Para la respuesta al escalon, todas las respuestas son similares y la cantidad de oscilación aumenta ligeramente a medida que  $K_i$  se hace más grande. Sin embargo, la respuesta a la perturbación cambia significativamente al modificar la ganancia integral  $K_i$ . En concreto, cuanto mayor es el valor de  $K_i$  empleado, más rápido decae el error a cero. Elegiremos  $K_i = 500$  porque el error debido a la perturbación decae rápidamente a cero, aunque la respuesta al escalon tenga un tiempo de estabilización mayor y una mayor sobre oscilación. Intentaremos reducir el tiempo de estabilización y la sobre oscilación añadiendo un término derivativo al controlador.

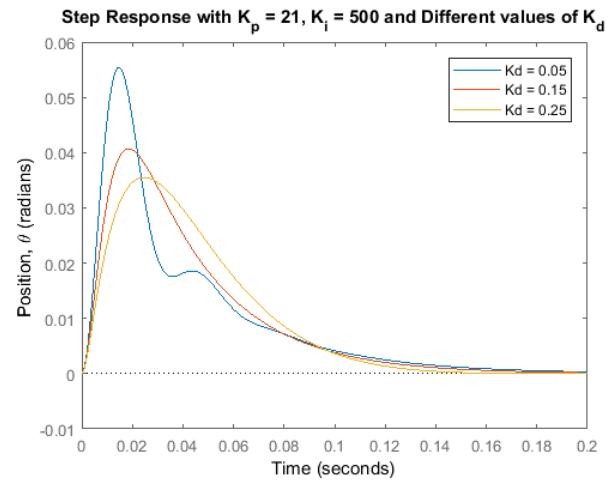
### Control PID

Al agregar un término derivativo al controlador, ahora contamos con los tres términos del controlador PID. Analizaremos ganancias derivativas  $K_d$  entre 0.05 y 0.25.

```

>> Kp = 21;
>> Ki = 500;
>> Kd = 0.05;
>> for i = 1 : 3
>>     C(:,:,i) = pid(Kp, Ki, Kd);
>>     Kd = Kd + 0.1;
>> end
>> sys_cl = feedback(C * P(motor), 1);
>> t = 0 : 0.001 : 0.1;
>> step(sys_cl(:,:,1), sys_cl(:,:,2), sys_cl(:,:,3), t)
>> ylabel('Posición, θ(radianes)')
>> title('Respuesta a escalón con  $K_p = 21$  y con  $K_i = 500$  y valores de  $K_d'$ )
>> legend('Kd = 0.05', 'Kd = 0.15', 'Kd = 0.25')

```



Parece que cuando  $K_d = 0.15$ , podemos cumplir con nuestros requisitos de diseño. Para determinar las características precisas de la respuesta al escalón usamos el comando stepinfo.

```
>> stepinfo(sys_cl(:,:,2))
```

```
ans =
```

estructura con los siguientes campos:

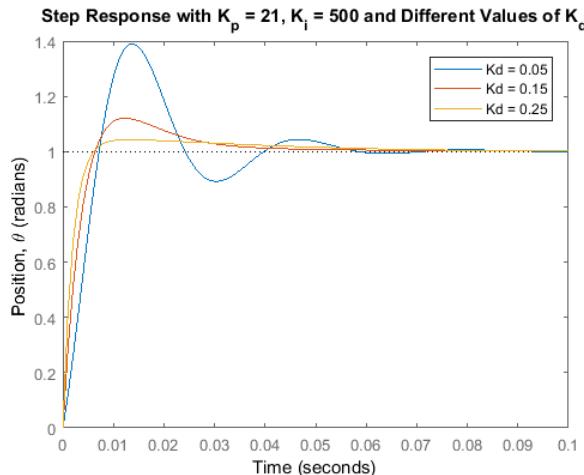
- RiseTime: 0.0046
- SettlingTime: 0.0338
- SettlingMin: 0.9103
- SettlingMax: 1.1212
- Overshoot: 12.1175
- Undershoot: 0
- Peak: 1.1212
- PeakTime: 0.0122

De lo anterior, observamos que la respuesta a una referencia escalón tiene un tiempo de estabilización de aproximadamente 34 ms (<40 ms), una sobre oscilación del 12% (<16%) y ningún error en estado estacionario. Además, la respuesta a la perturbación escalón tampoco presenta error en estado estacionario.

Por lo tanto, ahora sabemos que si utilizamos un controlador PID con

- $K_p = 21$
- $K_i = 500$
- $K_d = 0, 15$

Se cumplirán todos nuestros requisitos de diseño.



Veamos qué sucedió con la respuesta a la perturbación.

```

>> dist_cl = feedback(P(motor), C);
>> t = 0 : 0.001 : 0.2;
>> step(dist_cl(:,:,1), dist_cl(:,:,2), dist_cl(:,:,3), t)
>> ylabel('Posición, θ(radianes)')
>> title('Respuesta a perturbación con  $K_p = 21$  y  $K_i = 500$  y valores de  $K_d'$ )
>> legend('Kd = 0.05', 'Kd = 0.15', 'Kd = 0.25')

```

### *C. Control en cascada*

Esta arquitectura utiliza bucles de realimentación internos para responder rápidamente a las desviaciones, en lugar de esperar a que los efectos se propaguen por todo el bucle de control. Los errores se minimizan antes de que puedan propagarse. Si el bucle de corriente interno es más rápido (y debería serlo), un ligero cambio en la carga tendrá menor influencia en la velocidad. En general, los bucles internos deben tener un mayor ancho de banda que los bucles externos. El motor de CC suele tener una constante de tiempo pequeña. De esta manera, si solo se utilizara el control de velocidad, un pequeño cambio en el voltaje en los terminales del motor podría causar

una gran variación en la corriente (sin control). Por lo tanto, otra ventaja del control en cascada es la posibilidad de limitar la corriente a los valores máximos permitidos.

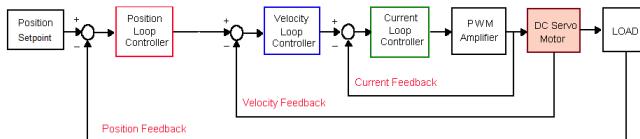


Fig. 23. Lazo de Control en Cascada

En una máquina de CC, el par es proporcional al valor de la corriente de armadura. El valor de consigna del par se obtiene del controlador de velocidad. Por lo tanto, el controlador de velocidad solicita el par necesario para mantener una velocidad determinada. Por lo tanto si se busca un control de posición preciso, se debe calibrar las ganancias de corriente, luego ajustar con precisión las ganancias de velocidad y refinar aún más las ganancias de posición .

#### D. Control PID directo

Es el enfoque que se optó para este proyecto y en el que se basó el diseño del PID, se realizó una aproximación de la respuesta de la planta a un escalón y se terminó con un ajuste fino de las variables PID dependiendo del eje que se maneja y de las variaciones del ángulo. Recordemos que en el caso del eje X, su carga tiende a ser constante de no ser por la inercia del motor y la del eje Y. En el caso del eje Y tiene variación en los dos casos, al no estar el láser sujeto de su centro de masa, a mayor ángulo aumenta la carga en el eje además de la inercia del mismo. Otro punto importante es que varía la respuesta del motor con un mismo PWM dependiendo del sentido de giro, para ambos ejes. Por lo cual tenemos que ajustar las constantes del PID que ponderen estas problemáticas.

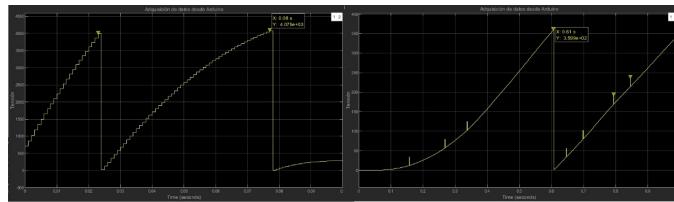


Fig. 24. Respuesta escalon eje X / eje Y

Partiendo de la linealización de las curvas, se fueron adaptando los parámetros PID tomando como referencia la siguiente tabla y observando el efecto de los mismos es las variaciones de los ángulos desde el min al max.

Efectos del parámetro de ganancia PID					
Parámetro aumentado	Tiempo de subida	Excederse	Tiempo de asentamiento	Error en estado estacionario	Estabilidad
$K_p$	Disminuir	Aumentar	Cambio	Disminuir	Degradar
$K_i$	Disminuir	Aumentar	Aumentar	Disminuir significativamente	Degradar
$K_d$	Disminución leve	Disminución leve	Disminución leve	Sin efecto	Mejora (para $K_d$ pequeño)



## VII. SOFTWARE

### A. Código ESP32

En la siguiente sección se explicarán las funciones que componen la programación del microprocesador. Se adjunta el programa como anexo al final del documento y se enlaza el hipervínculo en la siguiente imagen al github del proyecto para poder visualizar/descargar el archivo .ino



```

void setup() {
  Serial.begin(115200);
  I2C1.begin(SDA1, SCL1, I2CFREQ);
  I2C2.begin(SDA2, SCL2, I2CFREQ);
  ledcAttach(PWMPinx, FreqPWM, ResolPWM);
  ledcAttach(PWMPiny, FreqPWM, ResolPWM);
  pinMode(directionPin1y, OUTPUT);
  pinMode(directionPin2y, OUTPUT);
  pinMode(directionPin1x, OUTPUT);
  pinMode(directionPin2x, OUTPUT);
  previousTimedisplay = millis(); }
  
```

En la función setup() configuramos los periféricos usados en el proyecto, se configura el puerto serie a 115200 baudios, el puerto i2c1 y i2c2 para los sensores magnéticos del eje Y y X respectivamente a 400khz, los puertos PWM con una frecuencia 20khz para evitar ruidos en el motor y una

resolución de 10 bits, que es la máx para la freq deseada. Se configuran los pines de salida y se guarda los miliseg actuales para ser usado como timer

```
void SensorMagneticoY() {
I2C1.beginTransmission(0x36);
I2C1.write(0x0D);
I2C1.endTransmission();
I2C1.requestFrom(0x36, 1);
while(I2C1.available() == 0);
lowbyteY = I2C1.read();
I2C1.beginTransmission(0x36);
I2C1.write(0x0C);
I2C1.endTransmission();
I2C1.requestFrom(0x36, 1);
while(I2C1.available() == 0);
highbyteY = I2C1.read();
highbyteY = highbyteY << 8;
rawAngleY = highbyteY|lowbyteY;
degAngleY = rawAngleY * 0.087890625; }
```

La función SensorMagneticoY() es idéntica a la usada para el eje x y su función es obtener el angulo del eje en el motor. Se comunica con la dirección i2c 0x36 del as5600, escribe la dirección del registro del angulo (parte baja) 0x0D y la almacena en lowbyte. Lo mismo ocurre con la parte alta con dirección de registro 0x0C y la almacena en highbyte. Luego se desplaza la parte alta y se concatenan con una operación or y se multiplica or la mínima resolución para finalmente obtener el angulo deseado

```
void CalculoPID()
{
currentTime = micros();
deltaTime = (currentTime - previousTime) / 1000000.0;
previousTime = currentTime;
errorValuex = degAngleX - setpointx;
if (fabs(errorValuex)<0.1)
{errorValue=0;}
errorValuey = degAngleY - setpointy;
if (fabs(errorValuey)<0.1)
{errorValuey=0;}
edotx = (errorValuex - previousErrorx) / deltaTime;
edoty = (errorValuey - previousErrory) / deltaTime;
errorIntegralx = errorIntegralx + (errorValuex * deltaTime);
errorIntegraly = errorIntegraly + (errorValuey * deltaTime);
accionIntegralx=(errorIntegralx*kix);
accionIntegraly=(errorIntegraly*kiy);
controlSignalx = (kpx * errorValuex) + (kdx * edotx) + (accionIntegralx);
controlSignaly = (kpy * errorValuey) + (kdy * edoty) + (accionIntegraly);
previousErrorx = errorValuex;
previousErrory = errorValuey;
if (controlSignalx > MAXPWM)
{controlSignalx = MAXPWM;}
if (controlSignalx < -MAXPWM)
{controlSignalx = -MAXPWM;}
if (controlSignaly > MAXPWM)
{controlSignaly = MAXPWM;}
if (controlSignaly < -MAXPWM)
{controlSignaly = -MAXPWM;}
}
```

Se decidió con objetivos didácticos, usar el algoritmo de un PID discreto y desarrollado en la cátedra de la materia. Algo a mencionar es que a fin de evitar un efecto jitter en la corrección del error para cada setpoint (eje x e y) se acota a una décima de grado en valor absoluto.

```

void DriveMotorY()
{
int valorpwm;
if (controlSignaly < 0)
motorDirectiony = -1;
if (controlSignaly > 0)
motorDirectiony = 1;
if (controlSignaly == 0)
motorDirectiony = 0;
PWMValuey = (int)fabs(controlSignaly);
if (PWMValuey > MAXPWM)
PWMValuey = MAXPWM;
if (PWMValuey < MINPWMYI) errorValuey != 0 motorDirectiony == -1)
PWMValuey = MINPWMYI;
if (PWMValuey < MINPWMYD) errorValuey != 0
motorDirectiony == 1)
PWMValuey = MINPWMYD;
if (motorDirectiony == 1) // -1 == CCW
digitalWrite(directionPin1y, LOW);
digitalWrite(directionPin2y, HIGH);
if (motorDirectiony == -1) // == 1, CW
digitalWrite(directionPin1y, HIGH);
digitalWrite(directionPin2y, LOW);
if (motorDirectiony == 0) // == 0, stop/break
digitalWrite(directionPin1y, LOW);
digitalWrite(directionPin2y, LOW);
PWMValuey = 0;
if (PWMValuey != valorpwm)
ledcWrite(PWMPiny, PWMValuey);
valorpwm=PWMValuey; }

```

La función DriveMotorY() es idéntica a la usada para el eje x y su función es controlar el motor del eje y con el pwm calculado y sentido elegido.

```

void ComunicacionPC()
{
if (Serial.available() > 1)
{
ejex=Serial.parseFloat();
ejey=Serial.parseFloat();
if(ejex<=109 ejex>=51)
{setpointx=ejex;}
if(ejey<=102 ejey>=58)
{setpointy=ejey;}
}
currentTimedisplay=millis();
if (currentTimedisplay<previousTimedisplay+250)
{
previousTimedisplay=currentTimedisplay;
Serial.print(degAngleX);
Serial.print(" ");
Serial.println(degAngleY);
}
}

```

La función ComunicacionPC() se usa para la comunicación

entre el microprocesador con la pc, para obtener los setpoint de los ejes x e y, ademas de enviar los angulos respecto a la horizontal y vertical al display VFD cada 250mseg.

```

void loop()
{
SensorMagneticoY();
SensorMagneticoX();
CalculoPID();
DriveMotorX();
DriveMotorY();
ComunicacionPC();
}

```

Loop principal del programa del microprocesador pasando por las funciones previamente vistas.

#### B. Código Python

En la siguiente sección se explicara la programación de la aplicación de tracking. Se adjunta el programa como anexo al final del documento y se enlaza el hipervínculo en la siguiente imagen al github del proyecto para poder visualizar/descargar el archivo .py



```

import cv2
import serial
import numpy as np

```

Se importan las librerías necesarias para el proyecto, Cv2 (opencv), Serial (puertos serial) y Numpy como np (vectores)

```

cap = cv2.VideoCapture(0)

```

Se le indica a opencv cual es su fuente de video con el numero decimal entre (), como es el caso de una camara web externa se debe probar con 1 o si fuera la integrada con 0.

```

window_name = "Camara"
cv2.namedWindow(window_name, cv2 wnd...)
cv2.setWindowProperty(window_name, cv2 wnd...)

```

Definimos las características de la ventana donde se mostrara la camara, con nombre "camara" y en tamaño pantalla completa.

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

Creamos los arrays para el rango sup e inf de colores para su posterior detección, formato HSV

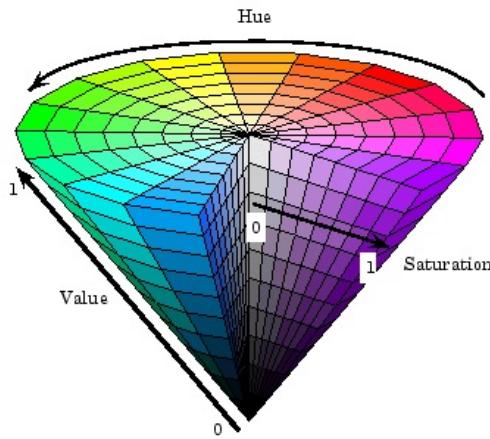


Fig. 25. Formato de Colores HSV (Hue,Saturation,Value)

```
ser=serial.Serial(port='COM8', baudrate=115200)
```

Se setea el uso del puerto serial

```
ret,frame = cap.read()
```

Lectura de un solo frame (cuadro) de la camara, ret es booleano indicando si se pudo leer la fuente de video.

```
frameHSV=cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
mask=cv2.inRange(frameHSV, azulBajo, azulAlto)
contours,hierarchy=cv2.findContours(mask, ...)
```

FrameHSV convierte el frame de formato bgr a formato hsv. Mask nos crea la mascara osea una imagen binaria (en blanco y negro) en el rango de colores indicado. cv2.findContours() busca contornos (bordes) en Mask con contornos externos y metodo de aproximacion simple, los cuales se almacenan en la variable contours.

```
for c in contours:
    area = cv2.contourArea(c)
    if area > 1000:
        M = cv2.moments(c)
        if (M['m00']==0): M['m00']=1
        x = int(M['m10']/M['m00'])
        y = int(M['m01']/M['m00'])
```

Pasando por los contornos detectados se lee el area con cv2.contourArea(c), si es mayor a 1000 (depende de la distancia entre el objeto y la camara) se obtiene el centro (centroide) del mismo con cv2.moments(c).

- M["m00"] es el area del contorno,
- M["m10"] es suma ponderada de las coordenadas X
- M["m01"] es suma ponderada de las coordenadas Y

por lo cual si el area es nula se estaría dividiendo por cero en el calculo de coordenadas del centroide, por eso el condicional que la hace unitaria.

```
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(frame, ',' .format(x,y),... )
nuevoContorno = cv2.convexHull(c)
cv2.drawContours(frame, [nuevoContorno],...)
angulox=round(((0.0920*x)+51),1)
anguloy=round((-0.0936*y)+102),1)
ser.write((str(angulox+2)+str(anguloy+2.5)+").encode())
cv2.imshow(window_name, frame)
```

Utilizamos cv2.putText con la fuente definida en font para dibujar las coordenadas del centroide en el marco superior derecho de la imagen, con cv2.drawContours dibujamos los contornos que fueron cerrados con cv2.convexHull para identificar los objetos que son trackeados. La imagen de la camara de 640x480 pixeles deben ser distribuidos uniformemente en el rango de angulos asignados para los ejes X e Y (35° a 125° grados) Por eso para cada uno se calculo su respectiva ecuación de la recta.

- angulox=round(((0.0920\*x)+51),1)
- anguloy=round((-0.0936\*y)+102),1)

Con la función ser.write escribimos en el puerto serie las coordenadas de X e Y separadas con un espacio y previamente redondeadas a un digito decimal con Round. La suma a cada coordenada responde a la compensación de la deriva que se da al no estar el láser alineado con la camara. cv2.imshow muestra la imagen con todos los dibujos que agregamos previamente.

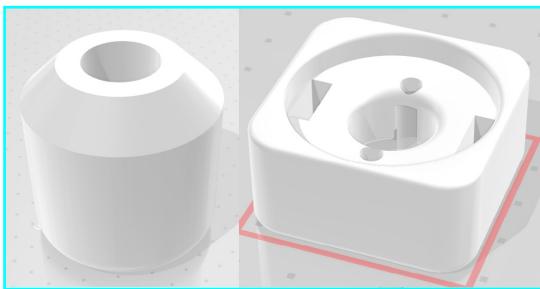
```
if cv2.waitKey(1) == ord('s'):
    break
cap.release()
cv2.destroyAllWindows()
```

El programa corre en un bucle while(1) que se quiebra ingresando en el teclado la letra "S", para luego liberar el uso de la cámara elegida y cerrar las ventanas creadas que mostraban las imágenes.

## VIII. CONSTRUCCIÓN DEL PROYECTO

### A. Soporte as5600

Los archivos de diseño fueron proveídos por la cátedra ya que fueron anteriormente utilizado en otro proyecto, se enlaza el hipervínculo en la siguiente imagen al github del proyecto para poder descargar los archivos .stl



### B. Soporte Motores

El motor del eje X se montó verticalmente con una rueda dentada en el eje donde se monta los soportes del motor del eje Y. Para el eje Y, el láser esta montado directamente en el eje

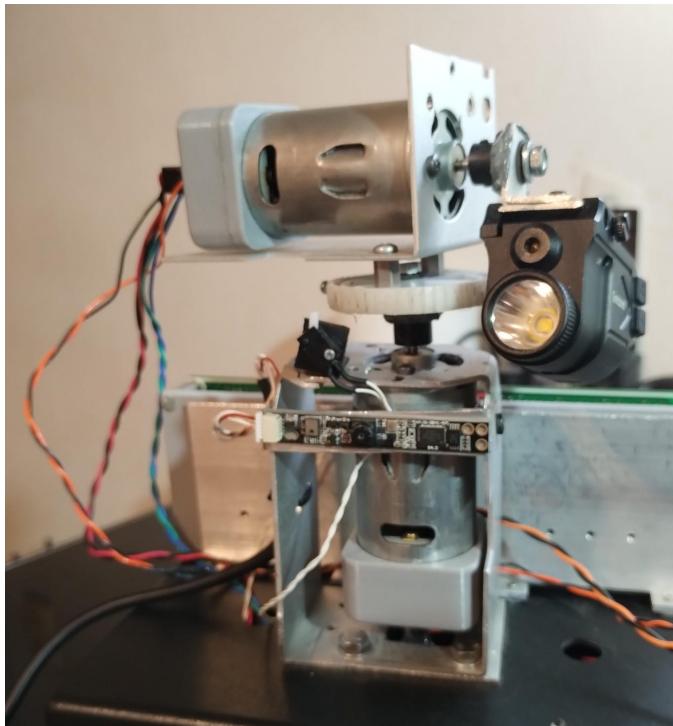


Fig. 26. Montaje de motores

### C. Montaje del proyecto

Se uso una batería vieja y gastada para aportar soporte y mayor estabilidad a todo el conjunto. Las distintas placas fueron montadas con torretas y tornillos, ademas las interconexiones son cableadas ya que las mismas están sujetas a movimiento.

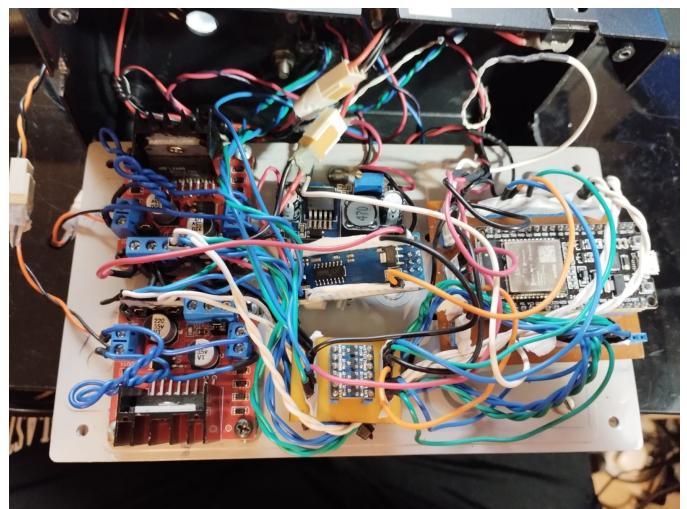


Fig. 27. Montaje de modulos

## IX. PRUEBAS DE SEGUIMIENTO

### A. Motores

Se observa en el video le movimiento de los motores durante el seguimiento del objetivo, es imperioso el ajuste del PID para evitar sobre impulsos en los ejes y desestabilizar el láser en su recorrido.

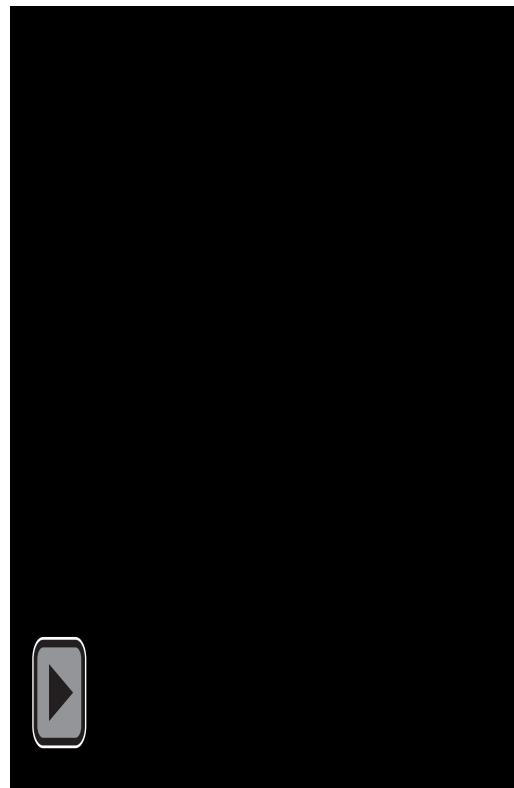


Fig. 28. Movimiento motores

### B. Variables mostradas

Se observa en el video la información mostrada en el display VFD, la variación de los ángulos respecto a la horizontal y vertical de la posición del láser.

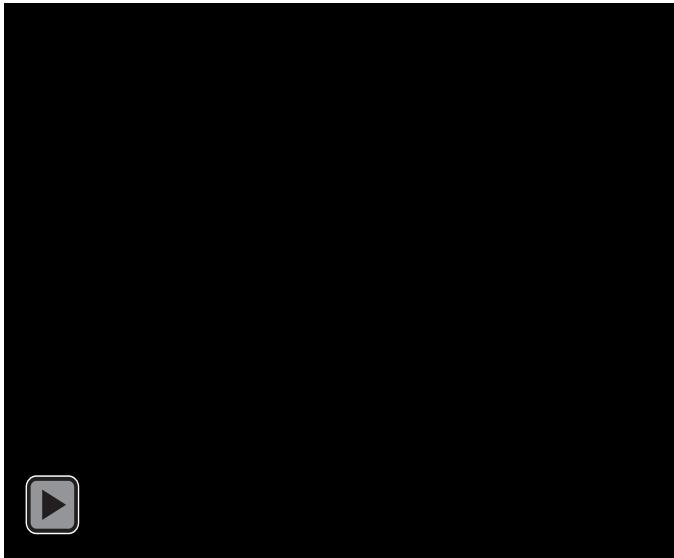


Fig. 29. Informacion mostrada

### C. Seguimiento con opencv

Se observa en el video la implementación de opencv en el seguimiento en tiempo real de un objeto con el color azulado. Se obtiene las coordenadas de la posición del centroide en la esquina superior derecha.



Fig. 30. Tracking con láser

## X. CONCLUSIONES

Como primer proyecto de control con aplicación de un lazo PID me encontré con la dificultad de su implementación en un parámetro como es la posición angular, que no tiene un control directo. En la mayoría de los papers/proyectos que se investigaron en un primer momento, los controles actúan en motores con caja reductora de relaciones 1:10 1:20 1:25, estos tienen la ventaja de tener un mayor control de la velocidad angular frente a su posición, y que la relación de la caja nos reduce la carga en el eje en igual proporción.

Como mejora al proyecto se podría proponer una lazo de control en cascada descripto anteriormente y una cámara con mayor resolución ya que la actual al ser VGA nos reduce la precisión de opencv y esto en consecuencia el área efectiva

de movimiento.

Para finalizar este trabajo quiero agradecer a la cátedra de Electrónica Industrial - LEIFRA- UTN FRA - en especial al Ing.Daniel Graff y al Ing.Ariel Tulian por su dedicación en la tutoría de la materia electiva y su posterior proyecto.

## REFERENCES

- [1] Ingeniería de Control Moderna - OGATA - 3ra edición.
- [2] Control PID avanzado - Astrom/Hagglund - 1ra edición.
- [3] Catedra Leifra - UTN FRA.

## APPENDIX

### A. Programa Microprocesador ESP32

#### B. Programa Python OpenCV

#### C. Esquemático

### A. Programa Microprocesador ESP32

```
#include <Wire.h>
```

```
#define I2C_FREQ 400000
#define SDA_1 21
#define SCL_1 22
#define SDA_2 27
#define SCL_2 26
#define RXD2 18
#define TXD2 17
#define DISPLAY_BAUD 9600
#define FrecPWM 20000
#define ResolucionPWM 10
#define MAXPWM 1023

//VARIABLES SENSOR MAGNETICO ANGULO Y
int lowbyteY; //raw angle 7:0
word highbyteY; //raw angle 7:0 and 11:8
int rawAngleY; //final raw angle
float degAngleY; //raw angle in degrees (360/4096 * [value between 0-4095])
float lastdegAngleY;

//VARIABLES SENSOR MAGNETICO ANGULO X
int lowbyteX; //raw angle 7:0
word highbyteX; //raw angle 7:0 and 11:8
int rawAngleX; //final raw angle
float degAngleX; //raw angle in degrees (360/4096 * [value between 0-4095])

float angulox;
float anguloy;

//VARIABLES MOTOR X
const int PWMPinx = 0;
int PWMValueX = 0;
const int directionPin1x = 33;
const int directionPin2x = 32;
int motorDirectionX = 0;

//VARIABLES MOTOR Y
const int PWMPiny = 4;
int PWMValuey = 0;
const int directionPin1y = 14;
const int directionPin2y = 12;
int motorDirectionY = 0;

//PARAMETROS PID EJE X
float setpointx = 80;
float kpx = 225;
float kix = 117.5;
float kdx = 0;
float controlSignalx = 0;
```

```

//PARAMETROS PID EJE Y

float setpointy = 80;
float kpy = 50;
float kiy = 45;
float kdy = 0.1;
float controlSignaly = 0;

//PID CALCULO TIEMPO
float previousTime = 0; //para calcular delta t
float currentTime = 0; //tiempo actual
float deltaTime = 0; //diferencia de tiempo

float errorValuex = 0; //error x
float edotx = 0; //derivativo x (de/dt)
float previousErrorx = 0; //para calcular el derivativo x (edotx)
float errorIntegralx = 0; //error integral x
float accionIntegralx = 0; //accion integral x

float errorValuey = 0; //error y
float edoty = 0; //derivativo y (de/dt)
float previousErrory = 0; //para calcular el derivativo y (edoty)
float errorIntegraly = 0; //error integral y
float accionIntegraly = 0; //accion integral y

//VARIABLES APPLICACION TRACKING
float ejex;
float ejey;
int valorx;
int valory;

// 
unsigned long previousTimedisplay = 0;
unsigned long currentTimedisplay = 0;

//Variables Debuging
int a=0;
int hab=0;
float salang;

TwoWire I2C_1 = TwoWire(0);
TwoWire I2C_2 = TwoWire(1);
HardwareSerial mySerial(2);

void setup() {

    Serial.begin(115200);

    mySerial.begin(DISPLAY_BAUD, SERIAL_8N1, RXD2, TXD2, 1);

    I2C_1.begin(SDA_1, SCL_1, I2C_FREQ);
    I2C_2.begin(SDA_2, SCL_2, I2C_FREQ);

    ledcAttach(PWMPinX, FrecPWM, ResolucionPWM);
    ledcAttach(PWMPinY, FrecPWM, ResolucionPWM);
    pinMode(directionPin1y, OUTPUT);
    pinMode(directionPin2y, OUTPUT);
    pinMode(directionPin1x, OUTPUT);
    pinMode(directionPin2x, OUTPUT);

    previousTimedisplay=millis();
}

void loop()
{
    SensorMagneticoY();
    SensorMagneticoX();
    CalculoPID();
    DriveMotorX();
    DriveMotorY();
    ComunicacionPC();
    //ComunicacionPCpruebaPID();
}

```

```

void SensorMagneticoY()
{
    I2C_1.beginTransmission(0x36); //connect to the sensor
    I2C_1.write(0x0D); //figure 21 - register map: Raw angle (7:0)
    I2C_1.endTransmission(); //end transmission
    I2C_1.requestFrom(0x36, 1); //request from the sensor
    while(I2C_1.available() == 0); //wait until it becomes available
    lowbyteY = I2C_1.read(); //Reading the data after the request
    I2C_1.beginTransmission(0x36);
    I2C_1.write(0x0C); //figure 21 - register map: Raw angle (11:8)
    I2C_1.endTransmission();
    I2C_1.requestFrom(0x36, 1);
    while(I2C_1.available() == 0);
    highbyteY = I2C_1.read();
    highbyteY = highbyteY << 8; //shifting to left
    rawAngleY = highbyteY | lowbyteY; //int is 16 bits (as well as the word)
    degAngleY = rawAngleY * 0.087890625;
    if (degAngleY<30 || degAngleY>200)
    {degAngleY=30;}
}

void SensorMagneticoX()
{
    I2C_2.beginTransmission(0x36); //connect to the sensor
    I2C_2.write(0x0D); //figure 21 - register map: Raw angle (7:0)
    I2C_2.endTransmission(); //end transmission
    I2C_2.requestFrom(0x36, 1); //request from the sensor
    while(I2C_2.available() == 0); //wait until it becomes available
    lowbyteX = I2C_2.read(); //Reading the data after the request
    //11:8 - 4 bits
    I2C_2.beginTransmission(0x36);
    I2C_2.write(0x0C); //figure 21 - register map: Raw angle (11:8)
    I2C_2.endTransmission();
    I2C_2.requestFrom(0x36, 1);
    while(I2C_2.available() == 0);
    highbyteX = I2C_2.read();
    highbyteX = highbyteX << 8;
    rawAngleX = highbyteX | lowbyteX;
    degAngleX = rawAngleX * 0.087890625;
    if (degAngleX<30 || degAngleX>200)
    {degAngleX=30;}
}

void CalculoPID()
{
    currentTime = micros(); //current time
    deltaTime = (currentTime - previousTime) / 1000000.0; //time difference in seconds
    //deltaTime = (currentTime - previousTime);
    previousTime = currentTime; //save the current time for the next iteration to get the time difference
    //Error en X
    errorValueX = degAngleX - setpointX; //Current position - target position (or setpoint)
    //Serial.println(errorValueX);
    if (fabs(errorValueX)<0.1)
    {errorValueX=0;}
    //Serial.println(errorValueX);
    //Error en Y
    errorValueY = degAngleY - setpointY; //Current position - target position (or setpoint)
    if (fabs(errorValueY)<0.1)
    {errorValueY=0;}
    //Serial.println(errorValueY);
    //Error Der X
    edotX = (errorValueX - previousErrorX) / deltaTime; //edotX = de/dt - derivative term

    //Error Der Y
    edotY = (errorValueY - previousErrorY) / deltaTime; //edotY = de/dt - derivative term

    //Error Int X
    errorIntegralX = errorIntegralX + (errorValueX * deltaTime); //integral term x - Newton-Leibniz, notice, this is a running sum!

    //Error Int Y
    errorIntegralY = errorIntegralY + (errorValueY * deltaTime); //integral term x - Newton-Leibniz, notice, this is a running sum!

    //Accion Integral X
    accionIntegralX=(errorIntegralX*kix);

    //Accion Integral Y
    accionIntegralY=(errorIntegralY*kiy);
}

```

```

//Señal Control X
controlSignalx = (kpx * errorValuex) + (kdx * edotx) + (accionIntegralx); //final sum, proportional term also calculated here

//Señal Control Y
controlSignaly = (kpy * errorValuey) + (kdy * edoty) + (accionIntegraly); //final sum, proportional term also calculated here

//Se guarda el error anterior X
previousErrorx = errorValuex; //save the error for the next iteration to get the difference (for edot)

//Se guarda el error anterior Y
previousErrory = errorValuey; //save the error for the next iteration to get the difference (for edot)

//Anticlamping X
if (controlSignalx > MAXPWM) //fabs() = floating point absolute value
{
    controlSignalx = MAXPWM; //capping the PWM signal - 8 bit
}

if (controlSignalx < -MAXPWM)
{
    controlSignalx = -MAXPWM;
}

//Anticlamping Y
if (controlSignaly > MAXPWM) //fabs() = floating point absolute value
{
    controlSignaly = MAXPWM; //capping the PWM signal - 8 bit
}

if (controlSignaly < -MAXPWM)
{
    controlSignaly = -MAXPWM;
}
//Serial.println(controlSignalx);
//Serial.println(controlSignaly);

}

void DriveMotorX()
{
    //esp_task_wdt_reset();

if (controlSignalx < 0) //negative value: CCW
{
    motorDirectionx = -1;
}
if (controlSignalx > 0) //positive: CW
{
    motorDirectionx = 1;
}
if (controlSignalx == 0) //0: STOP - this might be a bad practice when you overshoot the setpoint
{
    motorDirectionx = 0;
}

PWMValuex = (int)fabs(controlSignalx); //PWM values cannot be negative and have to be integers

if (PWMValuex > MAXPWM) //fabs() = floating point absolute value
{
    PWMValuex = MAXPWM; //capping the PWM signal - 8 bit
}

if (PWMValuex < MINPWMXI && errorValuex != 0 && motorDirectionx == -1)
{
    PWMValuex = MINPWMXI;
}

if (PWMValuex < MINPWMXD && errorValuex != 0 && motorDirectionx == 1)
{
    PWMValuex = MINPWMXD;
}

if (motorDirectionx == 1) // -1 == CCW
{
    digitalWrite(directionPin1x, LOW);
    digitalWrite(directionPin2x, HIGH);
}

```

```

if (motorDirectionx == -1) // == 1, CW
{
    digitalWrite(directionPin1x, HIGH);
    digitalWrite(directionPin2x, LOW);
}
if (motorDirectionx == 0) // == 0, stop/break
{
    digitalWrite(directionPin1x, LOW);
    digitalWrite(directionPin2x, LOW);
    PWMValuex = 0;
}
//Serial.println(motorDirectionx);
ledcWrite(PWMPinx, PWMValuex);
}

void DriveMotorY()
{
int valor pwm;

// esp_task_wdt_reset();
if (controlSignaly < 0) //negative value: CCW
{
    motorDirectiony = -1;
}
if (controlSignaly > 0) //positive: CW
{
    motorDirectiony = 1;
}
if (controlSignaly == 0) //0: STOP - this might be a bad practice when you overshoot the setpoint
{
    motorDirectiony = 0;
}
//-----
//Speed
PWMValuey = (int)fabs(controlSignaly); //PWM values cannot be negative and have to be integers

if (PWMValuey > MAXPWM) //fabs() = floating point absolute value
{
    PWMValuey = MAXPWM; //capping the PWM signal - 8 bit
}

//Calculo PWM MIN PARA EL SETPOINT ELEGIDO

//MINPWMYI=453+((1.4)*setpointy);
//MINPWMYD=842.5-((2.64)*setpointy);

if (PWMValuey < MINPWMYI && errorValuey != 0 && motorDirectiony == -1)
{
    PWMValuey = MINPWMYI;
}

if (PWMValuey < MINPWMYD && errorValuey != 0 && motorDirectiony == 1)
{
    PWMValuey = MINPWMYD;
}

if (motorDirectiony == 1) // -1 == CCW
{
    digitalWrite(directionPin1y, LOW);
    digitalWrite(directionPin2y, HIGH);
}
if (motorDirectiony == -1) // == 1, CW
{
    digitalWrite(directionPin1y, HIGH);
    digitalWrite(directionPin2y, LOW);
}
if (motorDirectiony == 0) // == 0, stop/break
{
    digitalWrite(directionPin1y, LOW);
    digitalWrite(directionPin2y, LOW);
    PWMValuey = 0;
}
if (PWMValuey != valor pwm)
//Serial.println(motorDirectiony);
{ledcWrite(PWMPiny, PWMValuey);
valor pwm=PWMValuey;}

}

```

```

void ComunicacionPCpruebaPID()
{
    if(a==1)
    {
        if (Serial.available() > 1)
        {
            setpointy=Serial.parseInt();
            hab=1;
        }
    }

    //Serial.println("A");

    if(a==0)
    {
        if (Serial.available() > 1)
        {
            a=1;
            //kpx=Serial.parseInt();
            //kix=Serial.parseInt();
            //kdx=Serial.parseInt();
            kpy=Serial.parseFloat();
            kiy=Serial.parseFloat();
            kdy=Serial.parseFloat();
            Serial.println(kpy);
            Serial.println(kiy);
            Serial.println(kdy);
            //ejey=Serial.parseInt();
        }
        /*
        ejex=Serial.parseInt();
        ejey=Serial.parseInt();
        if(ejex<=125 && ejex>=35)
        {setpointx=ejex;}
        if(ejey<=125 && ejey>=35)
        {setpointy=ejey;}
        */
        //setpointx=setpointx+ejex;
        //setpointy=setpointy+ejey;
    }

    if (hab=1 && salang!=degAngleY)
    {
        Serial.println(degAngleY);
        salang=degAngleY;
    }

    //Serial.println(degAngleY);
    //Serial.println(degAngleX);

}

void ComunicacionPC()
{
    if (Serial.available() > 1)
    {
        ejex=Serial.parseFloat();
        ejey=Serial.parseFloat();
        if(ejex<=109 && ejex>=51)
        {setpointx=ejex;}
        if(ejey<=102 && ejey>=58)
        {setpointy=ejey;}

        //setpointx=setpointx+ejex;
        //setpointy=setpointy+ejey;
    }
    currentTimedisplay=millis();
    if (currentTimedisplay>previousTimedisplay+250)
    {
        previousTimedisplay=currentTimedisplay;
        Serial.print(degAngleX);
        Serial.print(" ");
        Serial.println(degAngleY);
    }
}

```

B. Programa Python OpenCV

```
import cv2
import time
import serial
import numpy as np

cap = cv2.VideoCapture(0)
window_name = "Camara"
cv2.namedWindow(window_name, cv2.WND_PROP_FULLSCREEN)
cv2.setWindowProperty(window_name, cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)
azulBajo = np.array([100,100,20],np.uint8)
azulAlto = np.array([125,255,255],np.uint8)

ser = serial.Serial(port='COM8', baudrate=115200, timeout=1)

while True:
    ret,frame = cap.read()
    if ret==True:
        frameHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        mask = cv2.inRange(frameHSV, azulBajo, azulAlto)
        contours, hierarchy = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        for c in contours:
            area = cv2.contourArea(c)
            if area > 1000:
                M = cv2.moments(c)
                if (M["m00"]==0): M["m00"]=1
                x = int(M["m10"]/M["m00"])
                y = int(M['m01']/M['m00'])
                #cv2.circle(frame, (x,y), 4, (0,0,255), -1)
                font = cv2.FONT_HERSHEY_SIMPLEX
                cv2.putText(frame, '{}{}'.format(x,y),(500,50), font, 0.75,(0,255,0),1, cv2.LINE_AA)
                nuevoContorno = cv2.convexHull(c)
                cv2.drawContours(frame, [nuevoContorno], 0, (255,0,0), 3)
                angulox=round(((0.0920*x)+51),1)
                anguloy=round(((0.0936*y)+102),1)
                #cv2.putText(frame, '{}{}'.format(angulox,anguloy),(0,100), font, 0.75,(0,255,0),1, cv2.LINE_AA)

                ser.write((str(angulox+2) +' '+str(anguloy+2.5) +'\n').encode())

        cv2.imshow(window_name,frame)
        if cv2.waitKey(1) & 0xFF == ord('s'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

C. Esquematico

