Proyecto Fin de Grado
Grado de Ingeniería en Tecnologías Industriales

Prototipo de Ball & Plate para control con retroalimentación visual y CUDA

Autor: Jose Francisco Quesada Marañón
Tutor: Daniel Rodríguez Ramírez

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Grado
Grado de Ingeniería en Tecnologías Industriales

Prototipo de Ball & Plate para control con retroalimentación visual y CUDA

Autor:
Jose Francisco Quesada Marañón

Tutor:
Daniel Rodríguez Ramírez
Profesor Titular

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Proyecto Fin de Grado: Prototipo de Ball & Plate para control con retroalimentación visual y CUDA

Autor: Jose Francisco Quesada Marañón

Tutor: Daniel Rodríguez Ramírez

El Tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A los que lo lean

Agradecimientos

A todos los que me han apoyado, en especial a mis padres.

A Ana, por aguantarme.

A mi tutor Daniel por darme la oportunidad de desarrollar este proyecto y facilitar el trabajar de una forma tan amena y sencilla.

Resumen

El presente proyecto consiste en el diseño y la construcción de un sistema multivariable *ball & plate*, compuesto por una plataforma que se mueve con dos grados de libertad, por la que se desplaza un objeto esférico, el cuál se pretende controlar de forma automática.

Las entradas de nuestro sistema serán las coordenadas en las que se encuentra el objeto. Las salidas serán los ángulos que tenemos que modificar con el fin de corregir la posición del objeto hasta llegar a la referencia que deseemos.

El sensor que hemos elegido para la realización de nuestro proyecto es una cámara de vídeo. De esta manera tendremos que calcular la posición del objeto empleando diferentes técnicas de tratamiento de imagen, en concreto, el método de los momentos invariantes. Como actuadores para mover la plataforma contamos con dos servomotores, cada uno se encargará de mover uno de los ejes de la plataforma con un sistema de bielas. Respecto al diseño del mecanismo de las bielas utilizamos las ecuaciones de Freudenstein para la síntesis de máquinas.

En cuanto al control automático, usamos un PID discretizado mediante una aproximación bilineal. El sistema en el que se ejecuta todo el proceso de tratamiento de la imagen y el control automático es una placa Jetson TK1 de NVIDIA. Para el movimiento de los servomotores utilizaremos una placa Arduino MEGA 2560.

Abstract

This project was based on the design and construction of a multivariable ball & plate system, composed of a platform with two degrees of freedom, through which a spherical object moves as it is controlled automatically.

The inputs of our system are the coordinates of the location of the object. The outputs are the angles that we need to modify in order to correct the position of the object until we reach the reference we want.

We chose a video camera as a sensor and calculated the position of the object using different image treatment techniques, specifically, the method of invariant moments. We used two servo motors as actuators to move the platform. Each motor was in charge of moving one of the axles of the platform with a connecting rod system. The crankshaft mechanism was designed using Freudenstein's equations for machine synthesis.

As for automatic control, we used a discrete PID using a bilinear approach. The system that ran the entire imaging process and automatic control is a Jetson TK1 board from NVIDIA. Lastly, we used an Arduino MEGA 2560 board for the movement of the servomotors.

Índice

ÍNDICE DE FIGURAS

Notación

m	Masa de la bola
g	Constante gravitacional
Ft	Fuerza de translación
Fr	Fuerza de rotación
x	Distancia de la bola al centro
r	Radio de la bola
α	Ángulo de la plataforma
J	Momento de inercia
sin	Función seno
cos	Función coseno
Kcu	Ganancia crítica
Pu	Periodo sistema inestable
Kp	Término proporcional PID
Ti	Término integral PID
Td	Término derivativo PID
yk	Salida en el instante k
rk	Referencia en el instante k
ek	Error en el instante k
uk	Actuación de la entrada en el instante k
T	Tiempo de muestreo
q0, q1, q2	Parámetros aproximación tipo Tustin
Xc	Coordenada x del centro geométrico de la bola
Yc	Coordenada y del centro geométrico de la bola
mrs	Momento de orden r,s
fps	Frames por segundo
L4T	Linux for Tegra

1 DESCRIPCIÓN DEL PROBLEMA

1.1 Introducción

La finalidad de este proyecto es el diseño y la construcción de un sistema dinámico con varias entradas en el cual poder aplicar de forma práctica diferentes métodos de control automático. El sistema elegido fue un *ball & plate (plataforma y bola)*, una plataforma con dos grados de libertad en la que poder controlar un objeto esférico.

El sistema *ball & plate* no es más que la suma de dos sistemas *ball & beam* (balancín y bola). Por lo tanto, para entender bien el primero de ellos es importante conocer antes el funcionamiento de este sistema. El modelo *ball & beam* tiene dos grados de libertad, el primero de ellos es el ángulo del balancín, el cual es modificado por el actuador del sistema, y el segundo grado de libertad es la posición de la bola respecto al balancín, que es función del ángulo del balancín.

Como dijimos antes, el sistema *ball & beam* no es más que la suma de dos sistemas *ball & beam.* Por ello, tendrá cuatro grados de libertad: los dos primeros modificados por los actuadores del sistema y los dos segundos serán la posición de nuestro objeto en ambos ejes cartesianos.

Cabe destacar la naturaleza no lineal del sistema, al ser un caso particular del péndulo invertido.

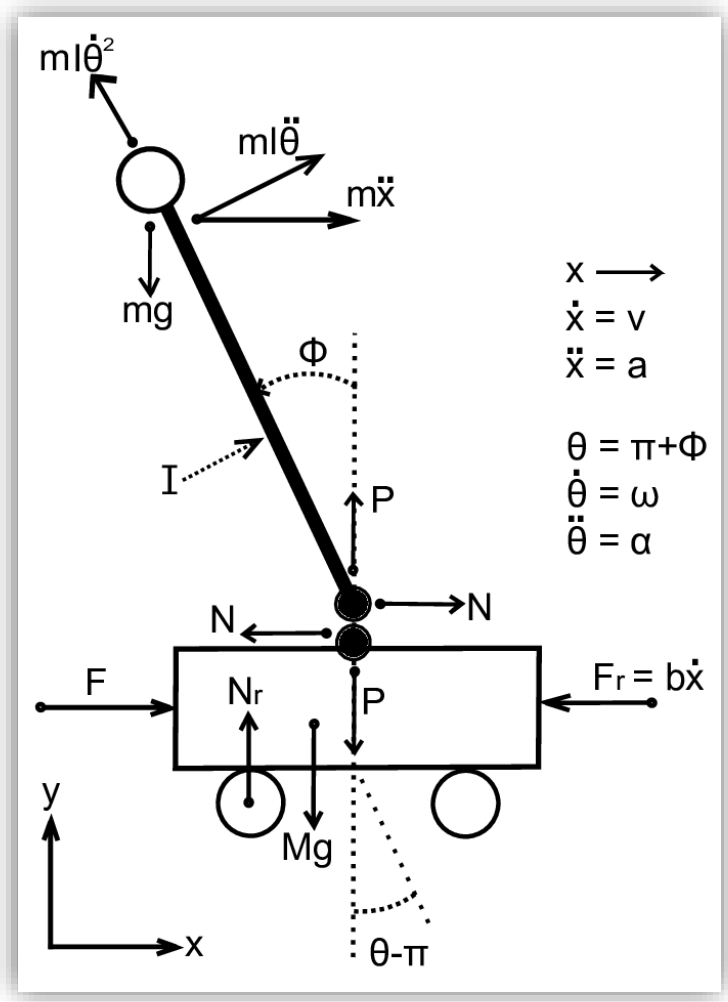


Figura 1-1. Esquema péndulo invertido

1.2 Particularización y objetivos

Para el control de nuestro sistema hicimos uso de una placa NVIDIA Jetson TK1. Para ello, los primeros objetivos del proyecto fueron la puesta a punto de dicha placa y la búsqueda de algún software compatible con el que procesar las imágenes para obtener la posición del objeto a controlar.

Una vez conseguidos estos primeros pasos, la siguiente tarea fue la de diseñar y construir un mecanismo robusto y preciso donde pudiésemos aplicar distintos métodos de control automático sin problemas. En nuestro caso en concreto, dispondríamos de dos salidas: una para inclinar el eje x de la plataforma y otra para el eje y, y dos entradas: la posición x e y del objeto a controlar.

Figura 1-2. Entradas y salidas del sistema

Para calcular la posición de dicho objeto se usó la percepción visual, esto es, el sensor de nuestro sistema es una cámara. Por otro lado, para que la plataforma tenga en todo momento la inclinación que nosotros queramos requerimos de servomotores paso a paso con un mecanismo de bielas. Es decir, nuestros actuadores serán dichos servomotores.

2 FUNDAMENTOS TEÓRICOS

La gravedad explica el movimiento de los planetas, pero no puede explicar quién establece los planetas en movimiento.

Isaac Newton

2.3. Modelo físico-matemático

Para modelar nuestro sistema asumimos varios supuestos para simplificar el análisis:

- Simplificamos a un sistema de partículas compuesto de dos cuerpos rígidos.
- El ángulo de cambio de la plataforma está limitado a 20 grados en cada dirección.
- Despreciamos la fricción entre la bola y la plataforma.

Vamos a centrarnos en uno de los dos ejes, es decir, vamos a estudiar primero el caso del *ball & beam*.

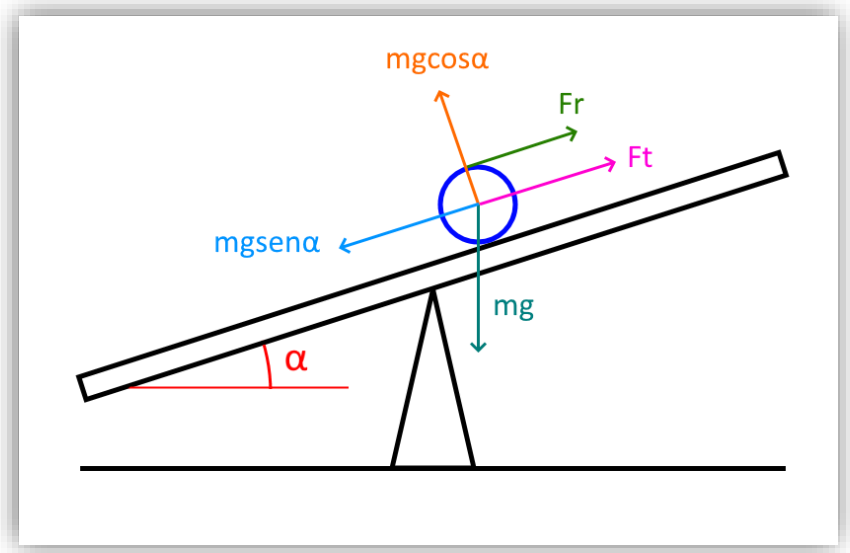


Figura 2-1. Esquema sistema ball & beam

$$m = masa\ de\ la\ bola\ (g)$$

$$g = constante\ gravitacional\ (9.81\frac{m}{s^2})$$

$$F_t = fuerza\ de\ traslación\ (N)$$

$$F_r = fuerza\ de\ rotación\ (N)$$

$$x = distancia\ de\ la\ bola\ al\ centro\ (m)$$

$$R = radio\ de\ la\ bola\ (m)$$

$$\alpha = ángulo\ de\ la\ plataforma$$

La aceleración de la bola viene dada por:

$$\ddot{x} = \frac{d^2x}{dt^2} \tag{2–1}$$

La fuerza resultante del movimiento de translación viene dada por:

$$F_t = m\ddot{x} \tag{2–2}$$

El par que se produce por la rotación de la bola se calcula multiplicando la fuerza de rotación por el radio:

$$T_r = F_r R = J\,\frac{d\omega_b}{dt} \tag{2–3}$$

$$J\frac{d\omega_b}{dt} = J\frac{d\left(\frac{v_b}{R}\right)}{dt} \tag{2–4}$$

$$J\frac{d\left(\frac{v_b}{R}\right)}{dt} = J\frac{d^2\left(\frac{x}{R}\right)}{dt^2} \tag{2–5}$$

$$J\frac{d^2\left(\frac{x}{R}\right)}{dt^2} = \frac{J\ddot{x}}{R} \tag{2–6}$$

Donde:

$$J = momento\ de\ inercia$$

El momento de inercia de un objeto esférico es:

$$J = \frac{2}{5}mR^2 \tag{2–7}$$

Por lo tanto, la fuerza rotacional la podemos calcular dividiendo el torque por el radio:

$$F_r = \frac{T_r}{R} = \frac{J\ddot{x}}{R^2} = \frac{2}{5}m\ddot{x} \tag{2–8}$$

Como ya hemos calculado ambas fuerzas podemos resolver la ecuación:

$$F_r + F_t = mg \sin \alpha \tag{2–9}$$

$$\frac{2}{5} m\ddot{x} + m\ddot{x} = mg \sin \alpha \tag{2–10}$$

$$\frac{2}{5} \ddot{x} + \ddot{x} = g \sin \alpha \tag{2–11}$$

$$\ddot{x} = \frac{5}{7} g \sin \alpha \tag{2–12}$$

Como α va a ser siempre un ángulo pequeño, podemos suponer que:

$$\sin \alpha = \alpha \tag{2–13}$$

Es decir, en radianes el seno del ángulo va a ser el propio ángulo, así que podemos aproximar la ecuación de la siguiente forma:

$$\ddot{x} = \frac{5}{7} g\alpha \tag{2–14}$$

Usando la transformada de Laplace de la posición respecto a los ángulos dados:

$$\ddot{x} = \frac{5}{7} g\alpha \tag{2–15}$$

Usando la transformada de Laplace de la posición respecto a los ángulos dados:

$$H(s) = \frac{X(s)}{\theta\,(s)} = \frac{5g}{7s^2} = \frac{0.91}{s^2} \tag{2–16}$$

2.4. Control automático

Como ya hemos calculado la función de transferencia de nuestro sistema, podemos simular nuestro modelo en Simulink.

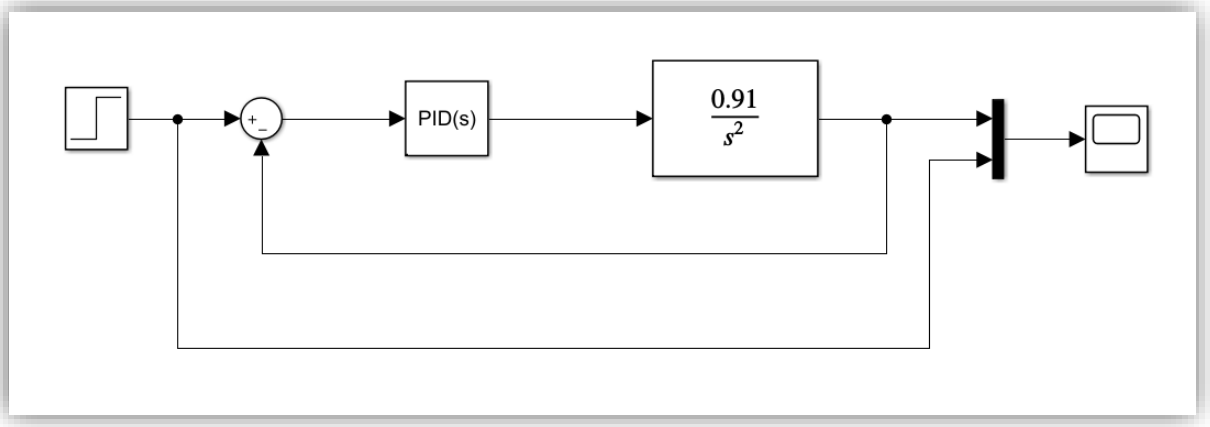


Figura 2-2. Sistema con PID en SIMULINK

Hay varias formas de sintonizar los parámetros de nuestro controlador PID, pero uno de los más usados es el de Ziegler Nichols. Hay dos formas de hacerlo con bucle abierto o con bucle cerrado.

2.2.1 Ziegler-Nichols: bucle abierto

Necesitamos conocer la respuesta de nuestro sistema en bucle abierto frente a un escalón, así que realizamos la simulación en Simulink:

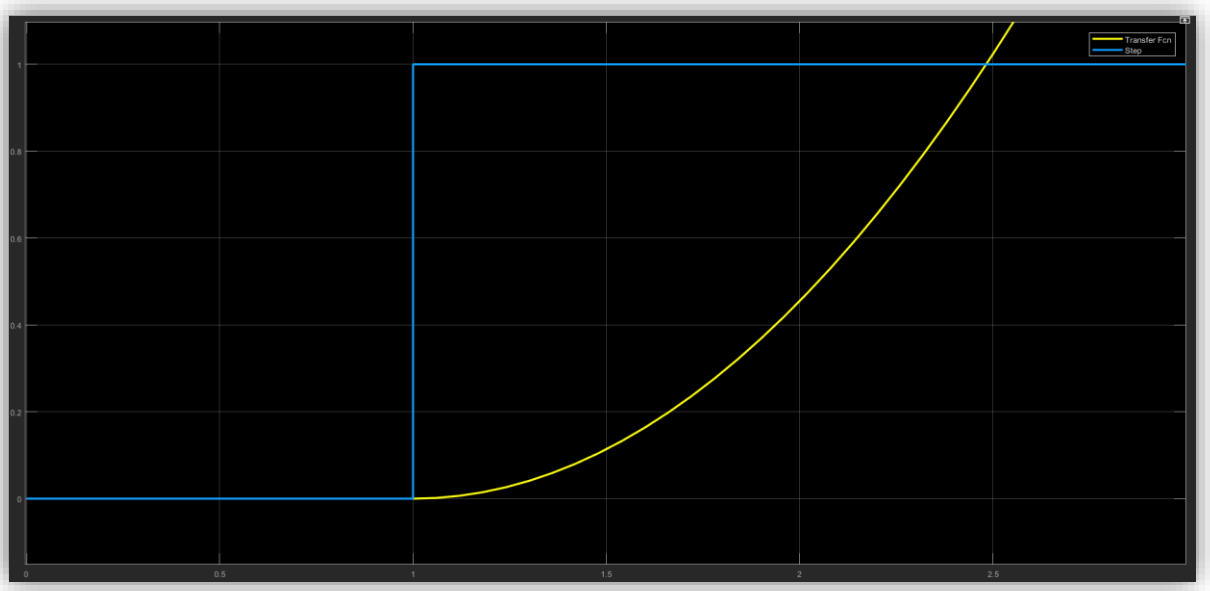


Figura 2-3. Respuesta del sistema en bucle abierto

Para sintonizar nuestro PID con el método de Ziegler-Nichols hacemos uso de esta tabla:

TIPO DE CONTROLADOR	Kp	Ti	Td
P	$\frac{1}{R \cdot L}$		
PI	$\frac{0,9}{R \cdot L}$	$3L$	
PID	$\frac{1,2}{R \cdot L}$	$2L$	$0,5L$

Figura 2-4. Tabla Ziegler-Nichols en bucle abierto

Es decir, necesitamos los valores de R y L, los cuales los obtenemos de esta forma:

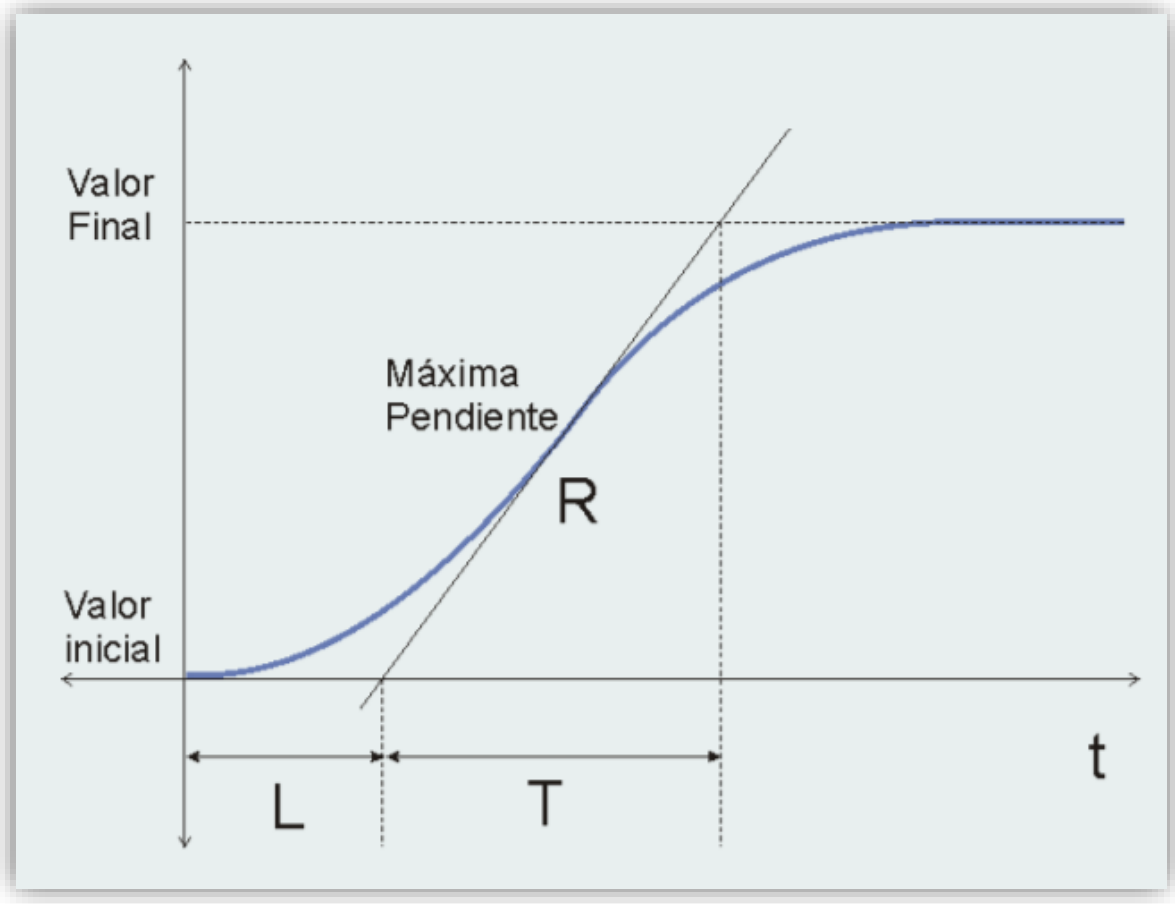


Figura 2-5. Valores R y L en Ziegels-Nichols

Si particularizamos para nuestro sistema queda algo así:

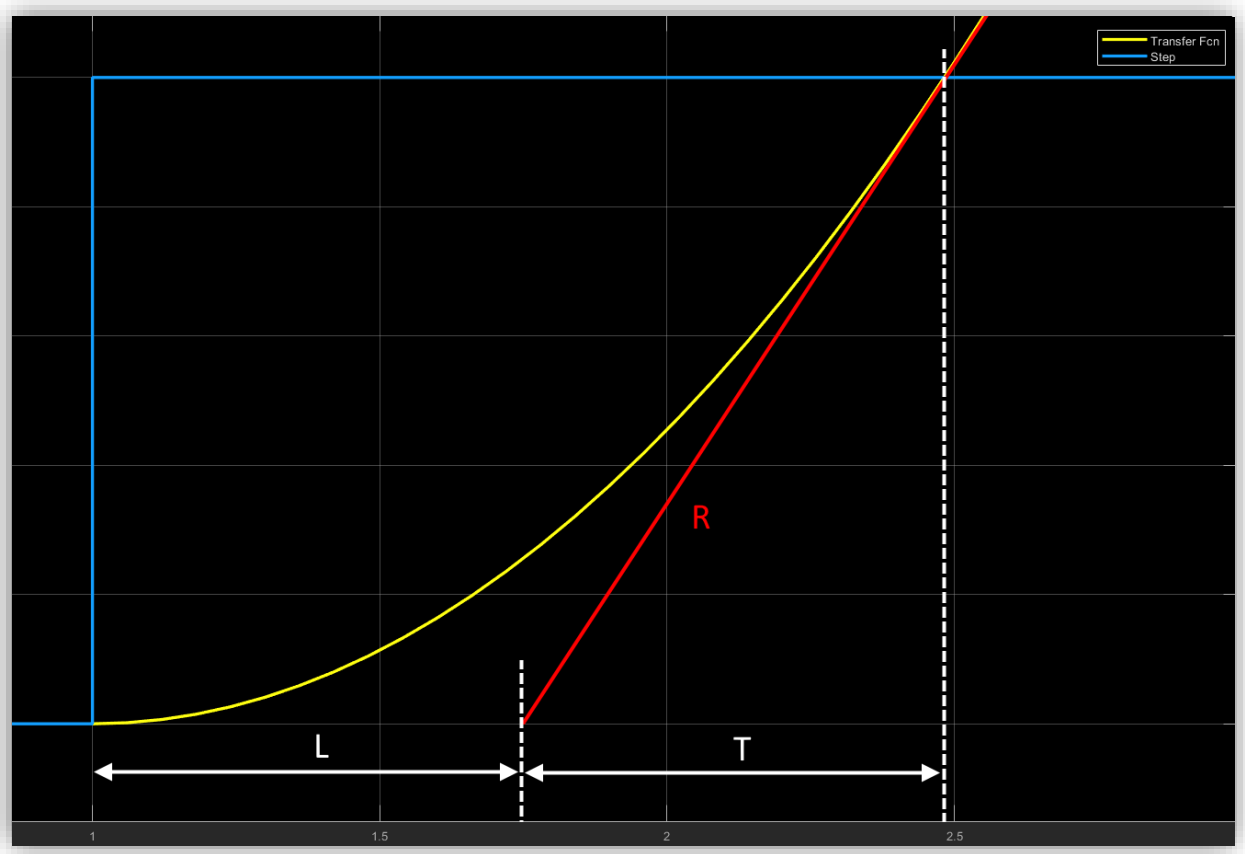


Figura 2-6. Particularización de L y R en nuestro sistema

Por lo tanto, nuestros valores de L y R son los siguientes

$$L = 0.75$$
$$R = 1.37$$

Ahora ya estamos en disposición de calcular los parámetros de nuestro controlador PID:

$$K_p = \frac{1.2}{LR} = 1.168$$

$$T_i = 2L = 1.5$$

$$T_d = 0.5L = 0.375$$

Estos valores no controlan correctamente el sistema, y lo hacen inestable. Como en el sistema es inestable en bucle abierto, deberíamos haber usado el método de Ziegler-Nichols de bucle cerrado.

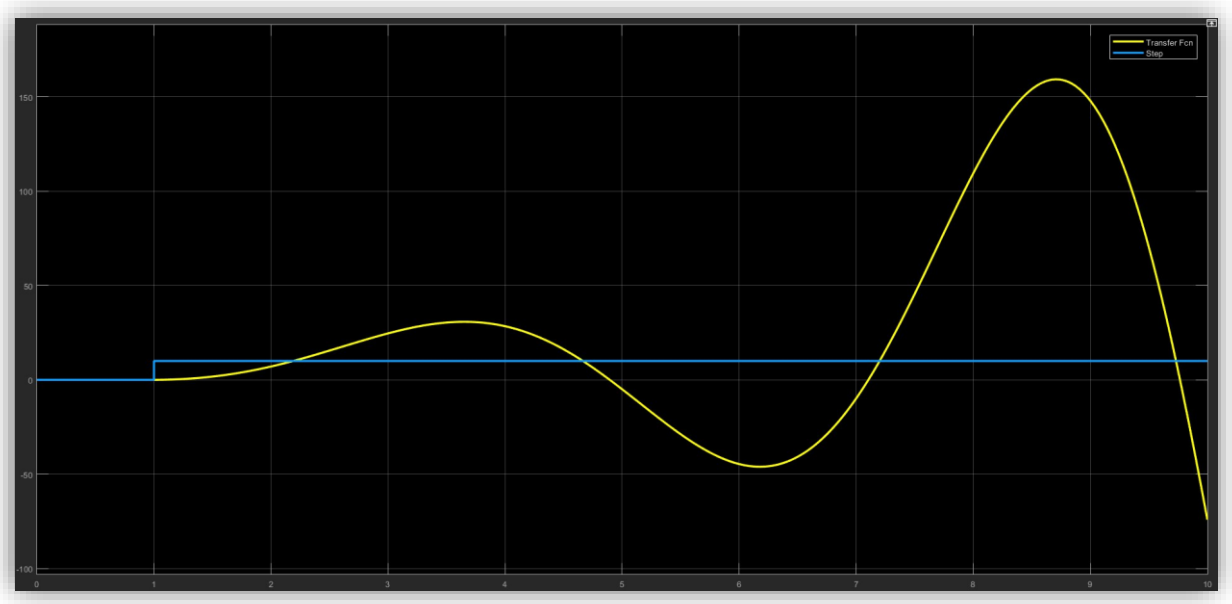


Figura 2-7. Respuesta inestable del sistema con el PID con Ziegler-Nichols en bucle abierto

2.2.2 Ziegler Nichols: bucle cerrado

Tenemos que encontrar una ganancia que haga inestable nuestro sistema en bucle cerrado, la cual se denomina ganancia critica (Kcu) y calcular el periodo (Pu) de dicha inestabilidad. En nuestro caso fueron:

$$Kcu = 3.5$$

$$Pu = 3.52$$

Volvemos a calcular los parámetros de nuestro PID:

$$K_p = 0.6\,Kcu = 2.1$$

$$T_i = \frac{Pu}{2} = 1.76$$

$$T_d = \frac{Pu}{8} = 0.44$$

Pero estos valores tampoco hacen que nuestro sistema sea estable, como podemos ver en la respuesta:

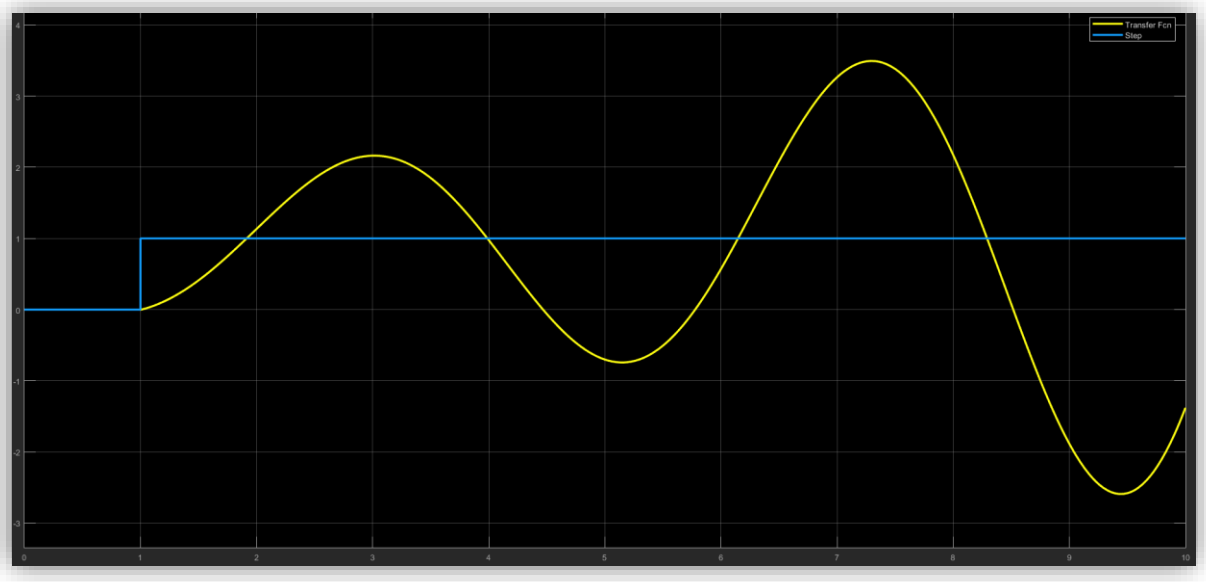


Figura 2-8. Respuesta inestable del sistema con el PID con Ziegler Nichols en bucle cerrado

Con esos valores como referencia sintonizando manualmente los parámetros si llegamos a conseguir estabilizar el sistema, los mejores valores que se encontraron fueron los siguientes:

$$K_p = 2$$
$$T_i = 0.5$$
$$T_d = 2$$

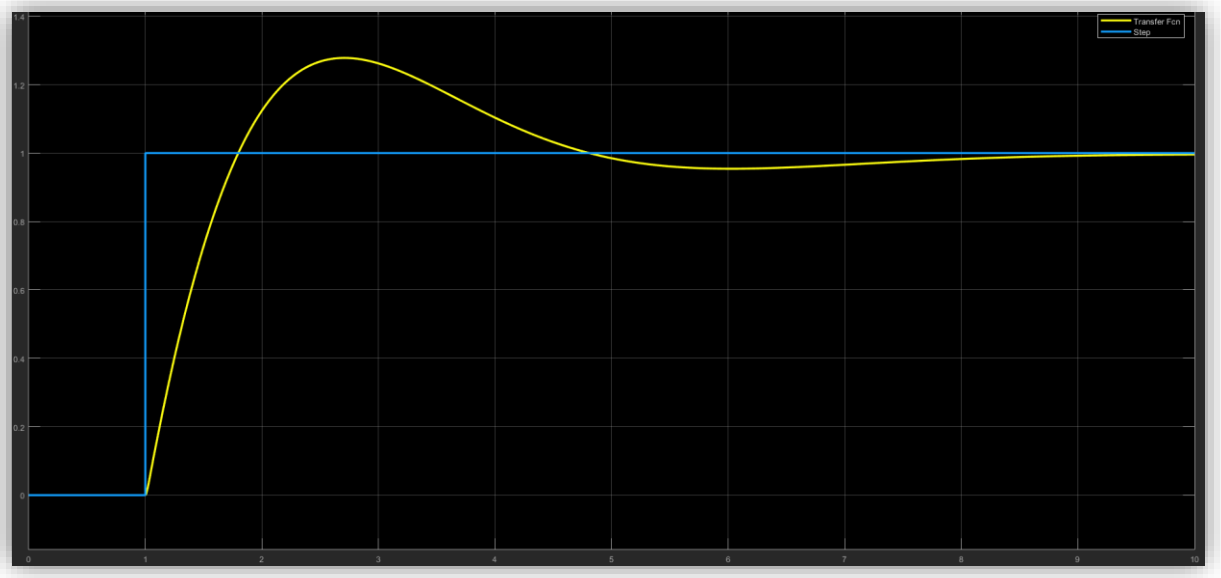


Figura 2-9. Respuesta estable del sistema con el PID con Ziegler Nichols en bucle cerrado

Lo expuesto anteriormente corresponde a la parte más teórica de nuestro proyecto, lo cual nos da unos valores de partida a la hora de sintonizar nuestro PID en el sistema real. El problema es que todo lo que hemos supuesto hasta aquí es en continuo. No obstante, debemos tener en cuenta que los sistemas informáticos funcionan con valores discretos y que, por lo tanto, a la hora de implementar los algoritmos tenemos que discretizar nuestro controlador PID continuo.

Hay varios métodos para realizar esto, nosotros vamos a usar una aproximación bilineal, también llamada trapezoidal o Tustin, puesto que es la aproximación más exacta:

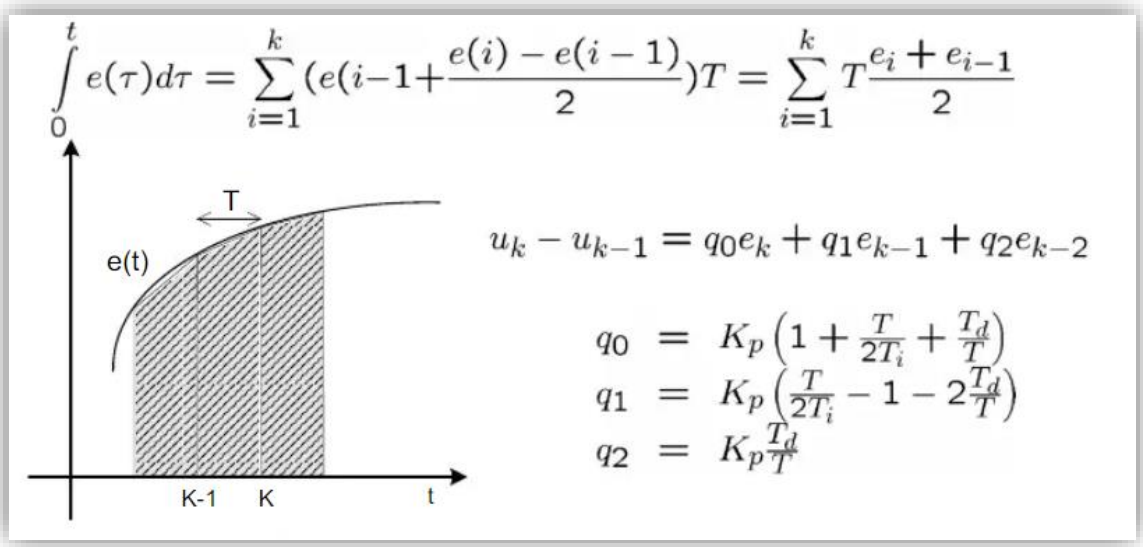


Figura 2-10. Aproximación tipo Tustin

El algoritmo que implementaremos en nuestro código sería el siguiente:

1. Esperar a que se cumpla el tiempo de muestreo T
2. Leer y_k
3. Calcular $e_k = r_k - y_k$
4. Calcular u_k según la siguiente expresión:

$$u_k = u_{k-1} + q_0\, e_k + q_1 e_{k-1} + q_2 e_{k-2}$$

$$q_0 = K_p\left(1+\frac{T_d}{T}\right)$$

$$q_1 = K_p\left(-1-2\frac{T_d}{T}+\frac{T}{T_i}\right)$$

$$q_2 = K_p\left(\frac{T_d}{T}\right)$$

5. Aplicar u_k
6. Actualizar u_{k-1}, e_{k-1} , e_{k-2}

2.5. Entrada: sensores

La entrada de nuestro sistema va a ser la posición del elemento que queremos controlar y la salida el ángulo al que necesitamos mover los servomotores.

Una de las primeras decisiones a la hora de afrontar el proyecto es elegir que sensores vamos a usar para conocer en todo momento la posición del objeto que queremos controlar. Entre las posibilidades que se barajaron estaban una pantalla táctil, sensores de peso y la detección visual. Finalmente, se eligió esta última opción, ya que era mucho más interesante dadas las características de nuestra placa Jetson (compatibilidad con cámara, compatibilidad con CUDA y OpenCV).

Una vez decidido que íbamos a usar una cámara nos enfrentábamos a una nueva dificultad, ¿cómo íbamos a saber dónde se encuentra el objeto en cada frame que capturamos por la cámara? Principalmente se trabajó en dos posibilidades: mediante la Transformada de Hough y calcular los momentos del área del objeto.

2.3.1 Transformada de Hough

La Transformada de Hough es una técnica para la detección de figuras en imágenes digitales, siendo posible encontrar todo tipo de figuras que puedan ser expresadas matemáticamente, tales como rectas, circunferencias o eclipses.

Las principales desventajas de la transformada de Hough es que su eficiencia depende de la calidad de los datos de entrada: los bordes se deben detectar bien, así que por norma general deberemos aplicar algún tratamiento a la imagen para eliminarlo.

Por suerte en OpenCV tenemos una función ya implementada en una de sus librerías básicas así que pudimos implementar el algoritmo sin mucha dificultad, en concreto la función en C++ es ésta:

```
HoughCircles( src_img, circles, CV_HOUGH_GRADIENT, dp, min_dist, up_thr,
                center_thr, min_radius, max_radius );
```

Dicha función tiene 9 argumentos, que expondremos a continuación:

- **src_img**: es la imagen en la que queremos detectar los círculos. Debe de ser una imagen en escala de grises y es bastante recomendable que le hayamos aplicado previamente un filtro de suavizado. En nuestro caso le aplicamos un desenfoque gaussiano
- **circles**: es el vector donde vamos a guardar los círculos que hemos detectado. Cada círculo tiene tres valores: “x”, “y” y el radio
- **CV_HOUGH_GRADIENT**: define el método de detección. Este es el único que existe en OpenCV.
- **dp**: es la relación inversa de la resolución. Suele usarse el valor 1.
- **min_dist**: es la distancia mínima entre los centros de los círculos detectados.
- **up_thr**: es el umbral del detector de contorno.
- **center_thr**: es el umbral del detector de centro.
- **min_radius**: es el radio mínimo de círculos que detectamos.
- **max_radius**: es el radio máximo de círculos que detectamos.

Es interesante destacar que, para nuestro problema en concreto, a min_dist le tendríamos que asignar un valor muy grande puesto que solo necesitamos detectar un círculo en la imagen. Asimismo, con min_radius y max_radius bastaría con aproximar qué valores de radio esperamos detectar en función de la distancia de la cámara al objeto.

A lo largo del proceso encontramos que los valores que nos aportan resultados mas interesantes son con ambos umbrales: up_thr, center_thr. Rápidamente detectamos que, con valores muy bajos, hallamos muy fácilmente los círculos de la imagen, pero también capta otros círculos inexistentes y esto hace que el rendimiento baje muchísimo. Por contra, para valores altos sólo detecta círculos muy perfectos, concluyendo que de esta forma es menos flexible a la hora de detectar, pero el rendimiento es bastante mejor.

Es por ello que se hizo latente que el siguiente paso sería encontrar un punto óptimo en el que el rendimiento no cayera mucho y, a la misma vez, detectara de forma robusta nuestro objeto. En las primeras pruebas este método no nos resultó interesante, porque en general el rendimiento se resentía mucho, y como veremos más adelante, no nos van a sobrar los fps si queremos controlar bien el objeto.

2.3.2 Método de los momentos invariantes

Este método consiste en calcular los momentos de una forma para después obtener el centro de gravedad de dicho objeto. Sin embargo, tiene un pequeño inconveniente, que necesitamos en primer lugar binarizar la imagen solo con el objeto, es decir:

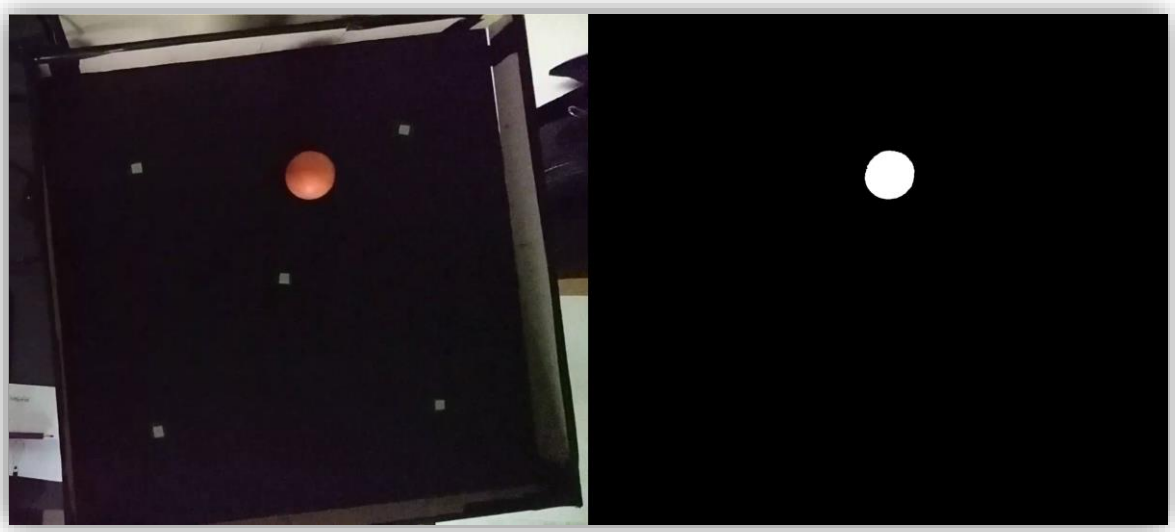


Figura 2-11. Imagen que captura la cámara e imagen binarizada

Por ello, nos enfrentamos a otra dificultad, ¿cómo encontramos que pixeles ocupa el objeto en cada momento?

Tenemos que usar un umbral en RGB, en otras palabras, un umbral para cada uno de los 3 canales de la imagen: rojo, verde y azul. De manera experimental, colocamos dichos umbrales para quedarnos solo con los pixeles del objeto a controlar. Como nuestra plataforma de control es de color negro esto es relativamente sencillo.

Una vez hemos configurado los 3 umbrales ya tenemos nuestra imagen binarizada, consiguiendo una matriz enorme de ceros y unos, siendo los unos el objeto que queremos controlar.

Una vez conseguido esto, ya estamos en disposición de calcular los momentos invariantes y el centro de gravedad del objeto.

Siendo X_c y Y_c las coordenadas cartesianas del centro del objeto, podemos calcularlo de esta forma:

$$X_c = \frac{m_{10}}{m_{00}}$$

$$Y_c = \frac{m_{01}}{m_{00}}$$

Siendo $m_{r,s}$ el momento de orden r, s:

$$m_{r,s} = \sum_{i=1}^{M} \sum_{j=1}^{N} i^r \, j^s \, p_{i,j}$$

Se puede ver fácilmente que m_{00} es el numero total de pixeles de nuestro objeto.

Obsérvese que OpenCV incluye funciones que nos permiten hacer esto sin ninguna dificultad. Al ser un algoritmo sencillo de implementar, se hizo sin usar dichas funciones.

Con los momentos invariantes es bastante sencillo calcular la matriz de inercia. En nuestro caso, con sólo conocer el centro del objeto, nos basta. Esto es así ya que siempre va a ser un objeto esférico y va a tener un radio similar en todo momento.

Como ya mencionamos anteriomente, este método tiene el inconveniente de la luz: un reflejo de luz incidiendo en la plataforma podría hacer que dejásemos de detectar correctamente el objeto y necesitaríamos siempre configurar manualmente los 3 umbrales puesto que, según la luz que haya, puede ir cambiando la forma en la que detectamos el objeto.

La principal ventaja de este método es que es mucho más eficiente y robusto que el anterior. El tiempo de ciclo apenas se ve afectado al realizar estos cálculos y, aunque no detectemos el 100% de los pixeles del objeto, podemos hallar en todo momento una posición aproximada de donde está el objeto con un error muy pequeño. A pesar de todo, con la transformada de Hough o bien lo detectamos o no lo detectamos.

2.6. Salida: actuadores

Para los actuadores estudiamos distintas soluciones, entre las que se encontraban servomotores con un mecanismo biela-manivela, servomotores con un mecanismo de cremallera o incluso actuadores electromagnéticos.

Finalmente nos decantamos por los servomotores con un mecanismo de biela-manivela por sencillez y economía. Para calcular los tamaños necesitamos usar síntesis de mecanismos que describiremos a continuación.

2.7. Síntesis del mecanismo

El objetivo del diseño del balancín era conseguir transmitir un ángulo muy grande a un ángulo muy pequeño. En la Figura 1 podemos ver una simplificación de uno de los mecanismos, podemos apreciar que se trata en realidad de un mecanismo de 4 barras biela-biela.

Para calcular el tamaño de dichas barras hacemos uso de la ecuación de Freudenstein, la cual dándole 3 posiciones nos da las dimensiones de las barras. De esta forma, resolvemos el problema de síntesis.

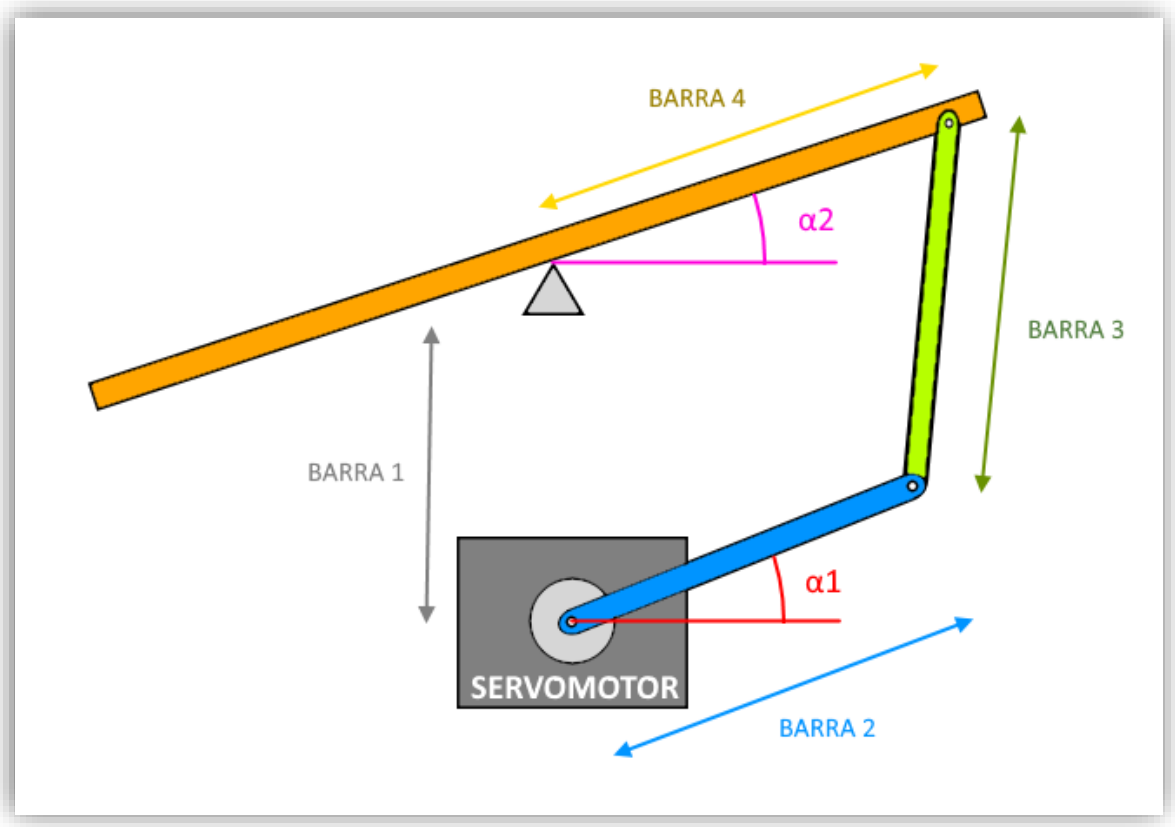


Figura 2-12. Esquema del mecanismo de 4 barras biela-biela

Parece evidente que el rango de movimiento del ángulo de los servomotores tenía que ser lo mayor posible, así obtenemos mucha más definición en la actuación de control. Nuestros servomotores tienen un rango de 180 grados, así que lo óptimo sería usar el máximo rango posible.

También parece evidente que el ángulo de la plataforma tiene que ser más pequeño, esto es, tenemos que transmitir un ángulo grande a un ángulo pequeño. Si bien el ángulo de libertad de la plataforma es muy pequeño, quizás no podremos modificar la posición de la bola tan rápido como nos gustaría. Además, cuanto más grande sea dicho ángulo mayor definición perderemos a la hora de controlar el objeto. Finalmente, se tomó una decisión de diseño y se optó porque tuviese un rango de 40 grados en cada eje.

Con esto ya tenemos definidas nuestras 3 posiciones del mecanismo. Como podemos ver en la siguiente figura, tendríamos 3 ecuaciones con 3 incógnitas, siempre que fijemos ya el tamaño de una de las barras:

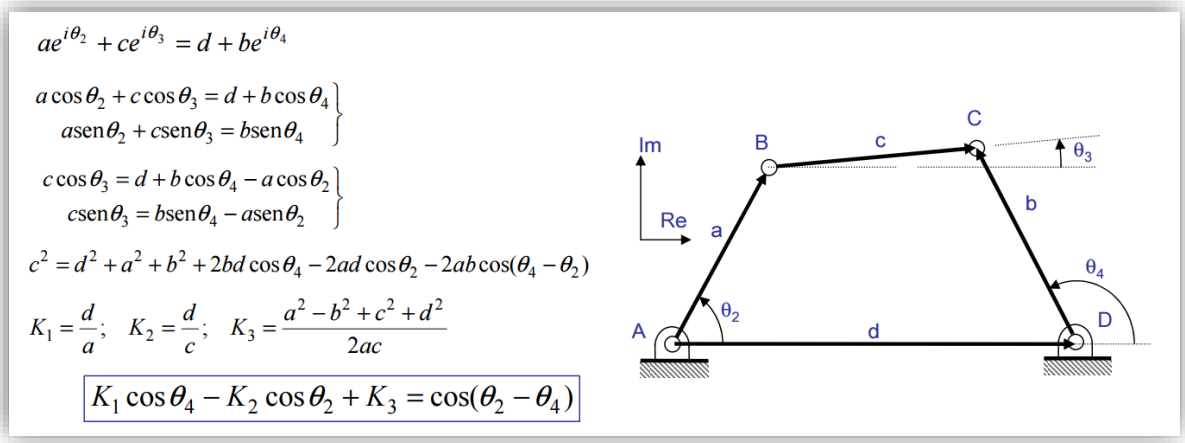


Figura 2-13. Ecuación de Freudestein

La posición que podemos ver en esa figura sería la del punto de equilibrio, donde la plataforma está completamente en horizontal. Por lo tanto, teníamos que diseñar un mecanismo de 4 barras biela-biela.

Para el cálculo de los tamaños de las barras se hizo uso de la ecuación de Freudenstein, la cual dándole 3 posiciones nos da las dimensiones de las barras. De esta forma resolvemos el problema de síntesis.

Necesitábamos que el ángulo conductor tuviese unos 120º y el ángulo conducido unos 10º.

Llamaremos al ángulo conductor $\propto$ y al ángulo conducido β.

De manera, que tenemos estas 3 posiciones preestablecidas: los dos extremos y la posición central.

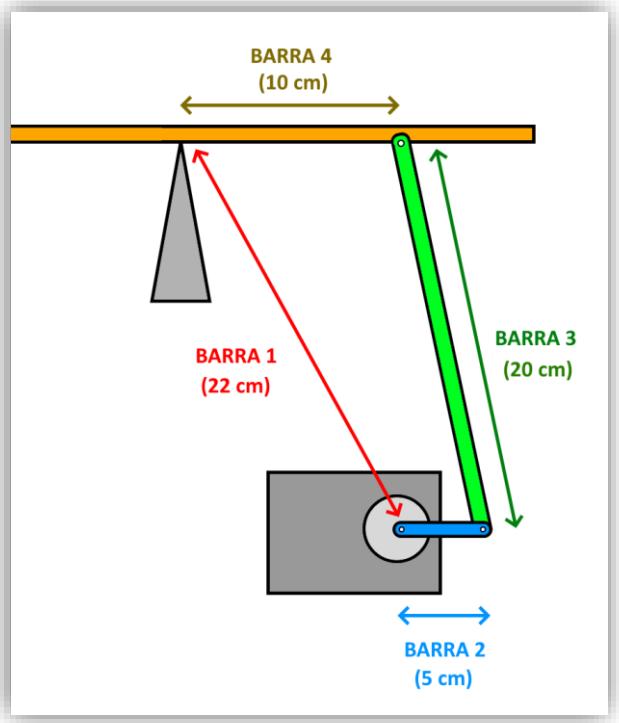


Figura 2-14. Posición intermedia del mecanismo

Aquí podemos ver las dos posiciones extremas de nuestro mecanismo:

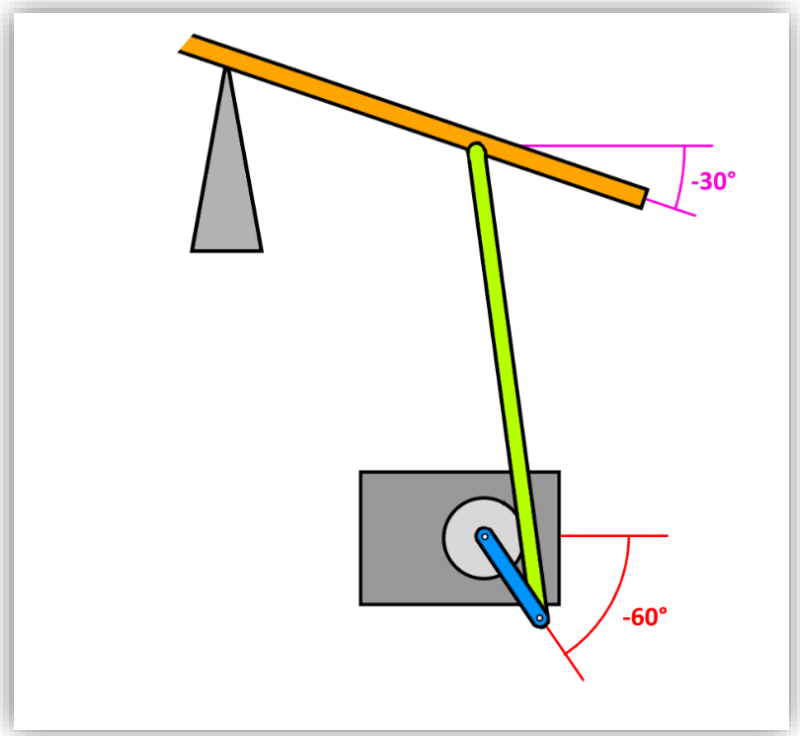


Figura 2-15. Posición extrema inferior del mecanismo

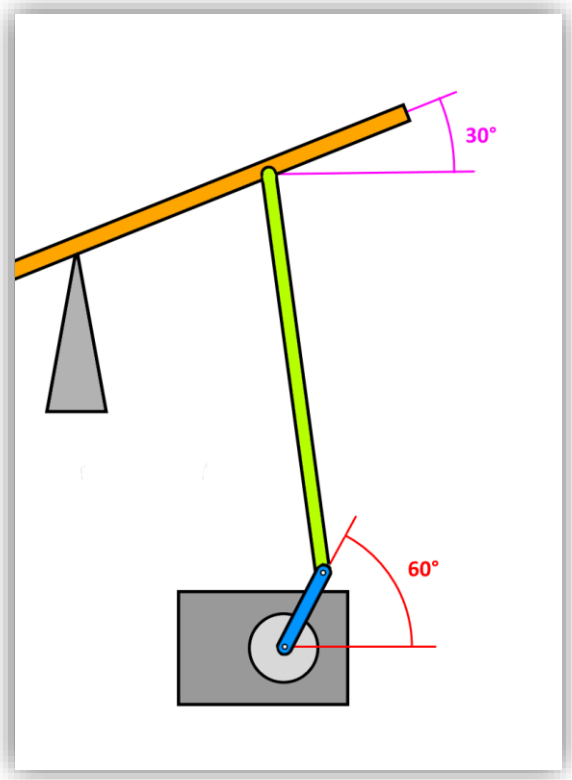


Figura 2-16. Posición extrema superior del mecanismo

3 ARQUITECTURA DE DISEÑO

Jetson TK1 acelera el camino de la computación embebida hacia el futuro, donde las máquinas interactuarán y se adaptarán a sus entornos en tiempo real.

Ian Buck, Vicepresidente de Computación acelerada de NVIDIA

3.1 Diagrama

Nuestro sistema se basa en el movimiento libre de una bola por una plataforma móvil, y nuestro objetivo es controlar su posición. Para ello, tenemos diferentes sensores y actuadores, en nuestro caso tenemos como sensores nuestra cámara y como actuadores nuestros servomotores. La plataforma descansa en el centro en una rotula y la podemos mover mediante los servomotores gracias a un mecanismo de dos bielas en cada eje.

El ciclo entero seria así: la cámara captura un frame de la imagen con la bola y la plataforma y se manda a la Jetson TK1, la cual calcula en que posición se encuentra el centro de la bola y también calcula la nueva acción de control, a saber, cuanto se tienen que modificar los ángulos de los servo-motores para que la bola vaya a la posición de referencia. La Jetson envía ambas acciones de control a la placa Arduino, que se encarga de enviar a los servomotores la modificación en su posición. Después de esto volveríamos a capturar un nuevo frame y repetir todo el proceso. Como podemos ver, una parte muy crítica para que el control de la posición de la bola tenga éxito es que este ciclo se realice de la forma más rápida posible.

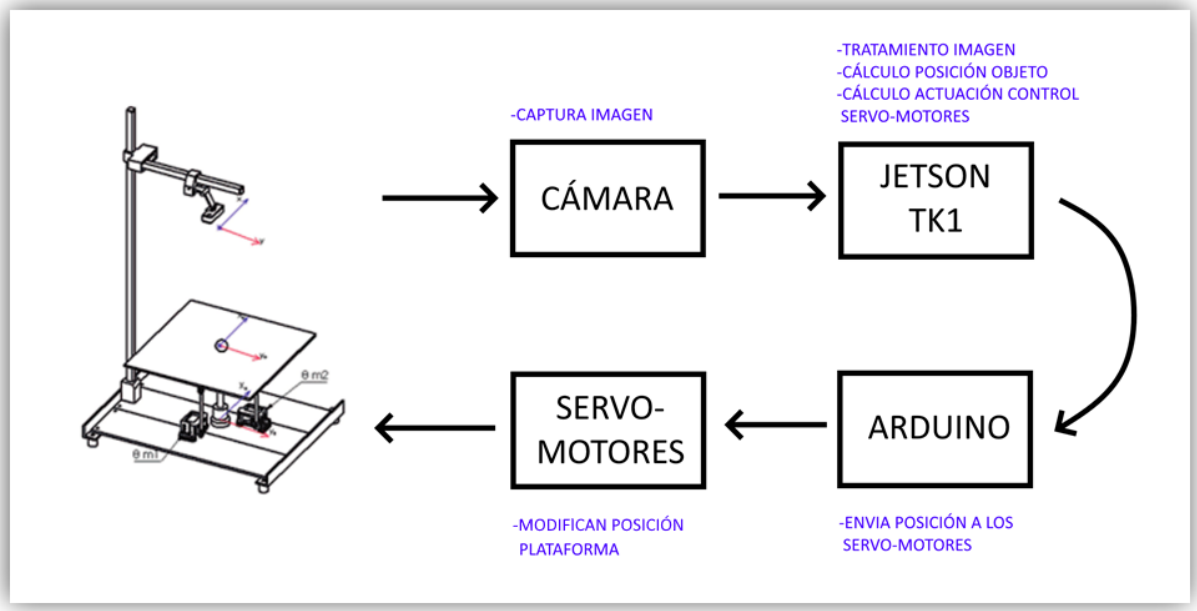


Figura 3-1. Diagrama completo del sistema

3.2 Componentes

3.2.1 Cámara Logitech G920

No todas las cámaras del mercado son compatibles con la Jetson TK1. Aquí ya descartamos una gran cantidad de posibilidades y, de entre las compatibles, finalmente se usó el modelo Logitech G920.

Las especificaciones que interesaban en este proyecto son:

- Resolución de hasta 1080p
- Framerate máximo de 30 fps

Figura 3-2. Cámara Logitech G920

3.2.2 Jetson TK1

Lanzada en 2014 por NVIDIA como "la primera supercomputadora móvil para sistemas móviles", es una plataforma de desarrollo basada en el procesador SoC Tegra K1.

Podemos destacar las siguientes características de nuestra Jetson TK1:

- 192 núcleos CUDA
- GigaFlops/s
- Quad-core ARM Cortex A15 CPU
- 2 GB de RAM
- USB 3.0
- Soporte para cámara
- Soporte oficial con Ubuntu, CUDA, OpenCV

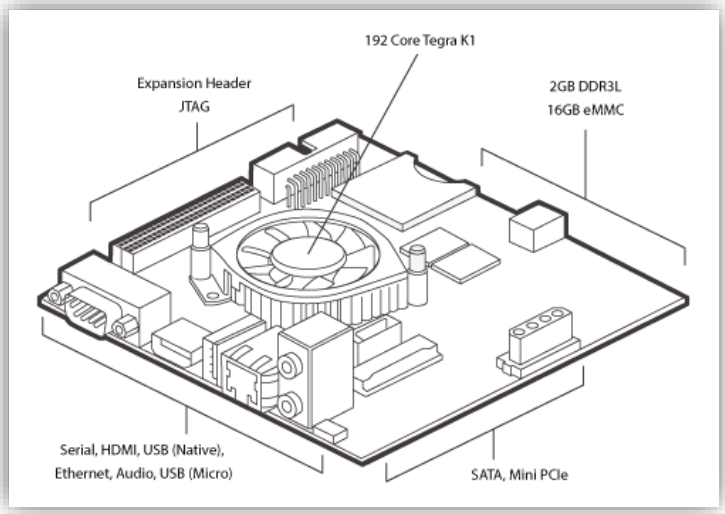


Figura 3-3. Esquema NVIDIA Jetson TK1

Como curiosidad, la última videoconsola de la compañía japonesa Nintendo monta una versión modificada de la NVIDIA Tegra TX1, la hermana mayor de la TK1.

3.2.3 Arduino MEGA 2560

Como teníamos el problema de que nuestra placa Jetson TK1 no puede controlar directamente los servomotores necesitábamos una placa Arduino a modo de esclavo de la Jetson TK1 para controlar los servos.

La placa Arduino que hemos usado es la MEGA 2650. No detallaremos las especificaciones de dicha placa, ya que simplemente se usó la placa Arduino que estaba más a mano, y nos era suficiente con que pudiese controlar un par de servomotores.

Sin embargo, encontrábamos un problema puesto que en cada ciclo de control necesitábamos comunicar el programa que se encarga de mover los servos en la placa Arduino con el programa de nuestra Jetson TK1 que captura la imagen y calcula donde se encuentra el objeto. En concreto, el dato a comunicar era la actuación de ambos servomotores. Por lo tanto, tendríamos que hacer un protocolo de comunicación entre ambos programas.

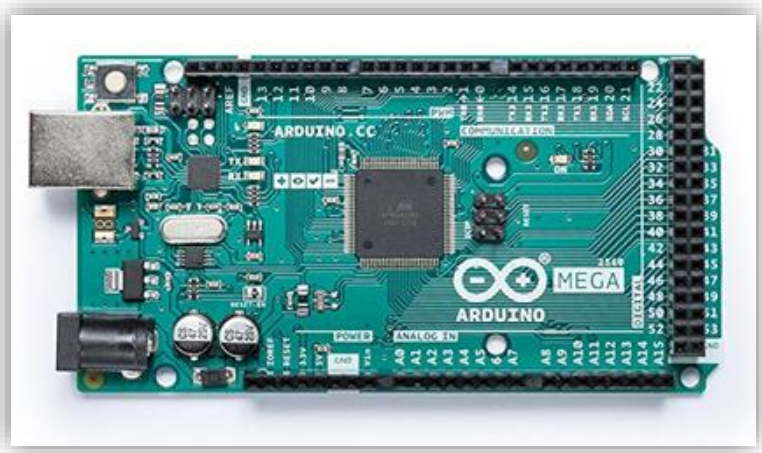


Figura 3-4. Arduino MEGA 2560

3.2.4 Servomotores

Elegimos el modelo 1501MG, que es un servomotor analógico con un extra de torque y es uno de los servomotores de tamaño estándar más populares.

Especificaciones:

- Tamaño: 40,7 mm x 20,5 mm x 39,5 mm
- Peso: 60 g
- Velocidad (6V): 0.14 s/60º
- Torque (6V): 17 kg*cm
- Velocidad (4.8V): 0.16 s/60º
- Torque (6V): 15,5 kg*cm

Figura 3-5. Servomotor 1501MG

Nos gustaría destacar que, en primer lugar, se usaron otros servos que no tenían el torque suficiente para mover la plataforma. Es por esto que el modelo descrito anteriormente contaba con un torque mayor.

Otro de los problemas que nos encontramos fue que la placa Arduino no tiene potencia para alimentar ambos servomotores y se tuvo que contar con una fuente de alimentación externa para los servos.

4 IMPLEMENTACIÓN

¿Cincuenta años de investigación en lenguajes de programación, y acabamos con C++?

Richard A. O'Keefe

4.1 Instalación de la infraestructura tecnológica

4.1.1 Sistema Operativo: Linux For Tegra (L4T)

L4T es la abreviación de Linux for Tegra, que es una distribución de GNU/Linux de NVIDIA para su serie de placas Jetson.

Necesitamos un equipo host con Ubuntu para poder flashear la tarjeta Jetson TK1. En concreto, para la TK1 necesitamos que el host tenga Ubuntu 14.04. Desde la página de NVIDIA descargamos un paquete llamado Jetpack, concretamente la versión 3.1 que es la última compatible con nuestra TK1, que nos servirá para flashear nuestra tarjeta. Una hemos descargado el paquete simplemente tenemos que ejecutarlo desde el host y seguir las instrucciones del programa de instalación.

Para ejecutar el Jetpack tendremos que darle permisos al archivo con esta instrucción:

```
chmod +x JetPack-3.1.run
```

Y luego simplemente ejecutarlo con:

```
./JetPack-3.1.run
```

Cabe señalar algunas dificultades durante la operación de flasheo, como que al poner la tarjeta en una especie de modo *recovery*. Tenemos que tener nuestra Jetson TK1 completamente apagada (no en reposo ni en suspensión) y conectarla al host con un cable micro-usb, pulsar y soltar el botón *POWER*, y luego mientras pulsamos el botón *FORCE RECOVERY* pulsamos el botón *RESET* durante 2 segundos para luego soltamos todos los botones. Dicha operación puede no salirnos a la primera.

Otra dificultad es que es bastante frecuente que el flasheo no se complete al 100% y puede que falle por varios motivos, como por desconexiones o falta de memoria, teniendo que iniciar de nuevo todo el proceso. A pesar de todo, con 3 o 4 intentos conseguiremos flashear nuestra tarjeta.

Una cuestión interesante a mencionar es que, una vez hayamos accedido a L4T 14.04 de nuestra tarjeta, el sistema operativo Ubuntu nos aconsejará actualizar a Ubuntu 16.04 y después a Ubuntu 18.04. Esto no es aconsejable, puesto que en 16.04 se vuelve bastante inestable y en 18.04 directamente no carga el entorno gráfico, y tendremos que empezar otra vez de cero. Por todo esto, nos quedamos en L4T 14.04.

4.1.2 CUDA

"Son las siglas de Compute Unified Device Architecture (Arquitectura Unificada de Dispositivos de Computo) y hace referencia a una plataforma de computación en paralelo incluyendo un compilador y un conjunto de herramientas de desarrollo creadas por NVIDIA que permiten a los programadores usar una variación del lenguaje de programación C para codificar algoritmos en GPU de NVIDIA.

CUDA intenta explotar las ventajas de las GPU frente a las CPU de propósito general, utilizando el paralelismo que ofrecen sus múltiples núcleos, que permiten el lanzamiento de un altísimo número de hilos simultáneos."[1]

El hecho de que OpenCV tenga compatibilidad con CUDA y su aceleración por GPU es muy interesante para nuestro proyecto, ya que nuestra tarjeta cuenta con una GPU NVIDIA y cómo vamos procesar imágenes, a la hora de detectar nuestro objeto en movimiento podríamos llegar a aprovecharnos de dicha aceleración por GPU, pero este es un tema que se tratara más adelante.

Hay múltiples versiones de CUDA, pero nosotros vamos a instalar la 6.5, ya que es la última que tiene compatibilidad con la arquitectura de 32 bits de nuestra tarjeta.

En primer lugar, tenemos que descargar el archivo meta-data del repositorio desde los servidores de NVIDIA:

```
https://developer.nvidia.com/cuda-toolkit-65
```

En concreto tenemos que descargar la versión de L4T que está en el apartado de Linux ARM.

Instalamos el repositorio que acabamos de descargar:

```
sudo dpkg -i cuda-repo-l4t-r21.2-6-5-prod_6.5-34_armhf.deb
```

Actualizamos el repositorio de aplicaciones:

```
sudo apt-get update
```

Instalamos CUDA 6.5:

```
sudo apt-get install cuda-toolkit-6-5
```

Añadimos el usuario 'ubuntu' al grupo vídeo:

```
sudo usermod -a -G video Ubuntu
```

[1] Definición de CUDA de Wikipedia

4.1.3 OpenCV

"Es una biblioteca libre de visión artificial desarrollada por Intel. Se suele usar para sistemas de seguridad con detección de movimiento, aplicaciones de control de procesos donde se requiere reconocimiento de objetos". [2]

OpenCV está desarrollado en C++ y tiene interfaces en C++, C, Python, Java y MATLAB. Hay muchas versiones de OpenCV, pero no todas son compatibles con nuestra tarjeta. En concreto debemos usar una versión optimizada para nuestra Jetson llamada OpenCV4Tegra (basado en la versión de OpenCV 2.4.10.1).

Para poder instalar OpenCV necesitamos tener CUDA instalado en nuestra tarjeta y por supuesto L4T (Linux for Tegra). Descargamos OpenCV4Tegra 2.4.10.1:

```
wget
http://developer.download.nvidia.com/embedded/OpenCV/L4T_21.1/libop
encv4tegra-repo_l4t-r21_2.4.10.1_armhf.deb
```

Instalamos el paquete:

```
sudo dpkg -i libopencv4tegra-repo_l4t-r21_2.4.10.1_armhf.deb
```

Actualizamos el repositorio:

```
sudo apt-get update
```

Instalamos las siguientes depencias:

```
sudo apt-get install libopencv4tegra libopencv4tegra-dev
```

Si queremos comprobar que realmente se ha instalado correctamente en nuestra maquina y la versión que tenemos este comando nos puede ser útil:

```
pkg-config –modversion opencv
```

[2] Definición de OpenCV de Wikipedia

4.2 Diseño y construcción del mecanismo

Uno de los elementos más críticos a la hora de empezar a diseñar nuestro mecanismo era el tamaño y los materiales de la plataforma de control.

Una plataforma muy pequeña facilita su construcción y tiene menos peso por lo que los servomotores pueden moverla más fácilmente, pero en cambio, da menos juego a la hora de hacer pruebas de control.

Finalmente, se decidió que el orden de magnitud de la plataforma fuesen unos 40 cm x 40 cm.

Otra decisión importante era el espesor y el material de dicha plataforma, ya que un espesor demasiado reducido haría que la plataforma se doblase un poco. Si fuese demasiado gruesa aumentaría mucho el peso de la plataforma, haciendo que a los servomotores les costase más mover la plataforma ágilmente (recordamos que la idea es que los servos tienen que mover la plataforma de forma muy rápida).

Figura 4-1. Ball & Plate en construcción

Analizamos diferentes placas metálicas, pero descartamos esta solución por lo que hemos expuesto anteriormente. En cuanto el espesor aumentaba lo suficiente para que la plataforma tuviese rigidez por si sola, el peso aumentaba considerablemente.

Por último, se decidió usar una plataforma cuadrada de madera de unos 3 mm de espesor y 40 cm de lado.

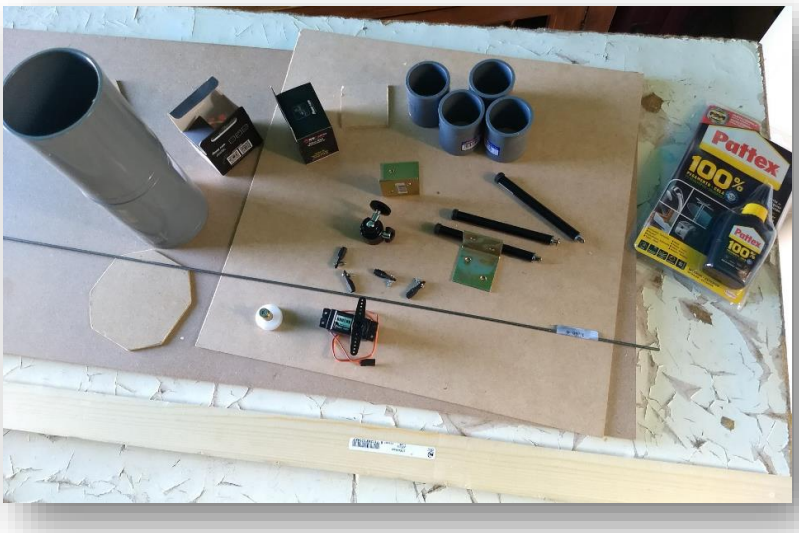


Figura 4-2. Materiales construcción Ball & Plate

Para la base del *plate & ball* se usó también madera, en concreto una tabla de 80 cm por 40 cm y 1 cm de espesor. Para sujetar ambos servos se diseñaron con madera unos pequeños soportes en forma de U. En último lugar, una de las partes más críticas era el apoyo de la plataforma, que tenía que tener 2 grados de libertad. Se usaron un par de secciones de tuberías con una rotula en el extremo.

4.3 Implementación de software y algoritmos

Nuestro sistema ejecutaría dos programas en paralelo. Por un lado, en la Jetson TK1 se ejecutaría el programa principal, el cual haría todo el cálculo de la posición del objeto y algoritmo de control. Por el otro, la placa Arduino estaría esperando en todo momento la llegada de nuevos valores de ángulos para aplicar a los servomotores.

4.3.1 Jetson TK1: tratamiento de la imagen y control automático

Este es el programa principal de nuestro sistema, ya que el programa que se ejecuta en la placa Arduino actúa como esclavo, siempre esperando nuevos ángulos con los que modificar los servomotores.

En este apartado se comentarán los fragmentos de código más relevantes e interesantes de cara al proyecto, la totalidad del código se encuentra en un anexo de este documento.

Lo primero que hacemos en este programa es capturar un frame con nuestra cámara, dicho frame son tres matrices, una para cada uno de los canales RGB y cuyo tamaño es la resolución de la imagen que captura nuestra cámara.

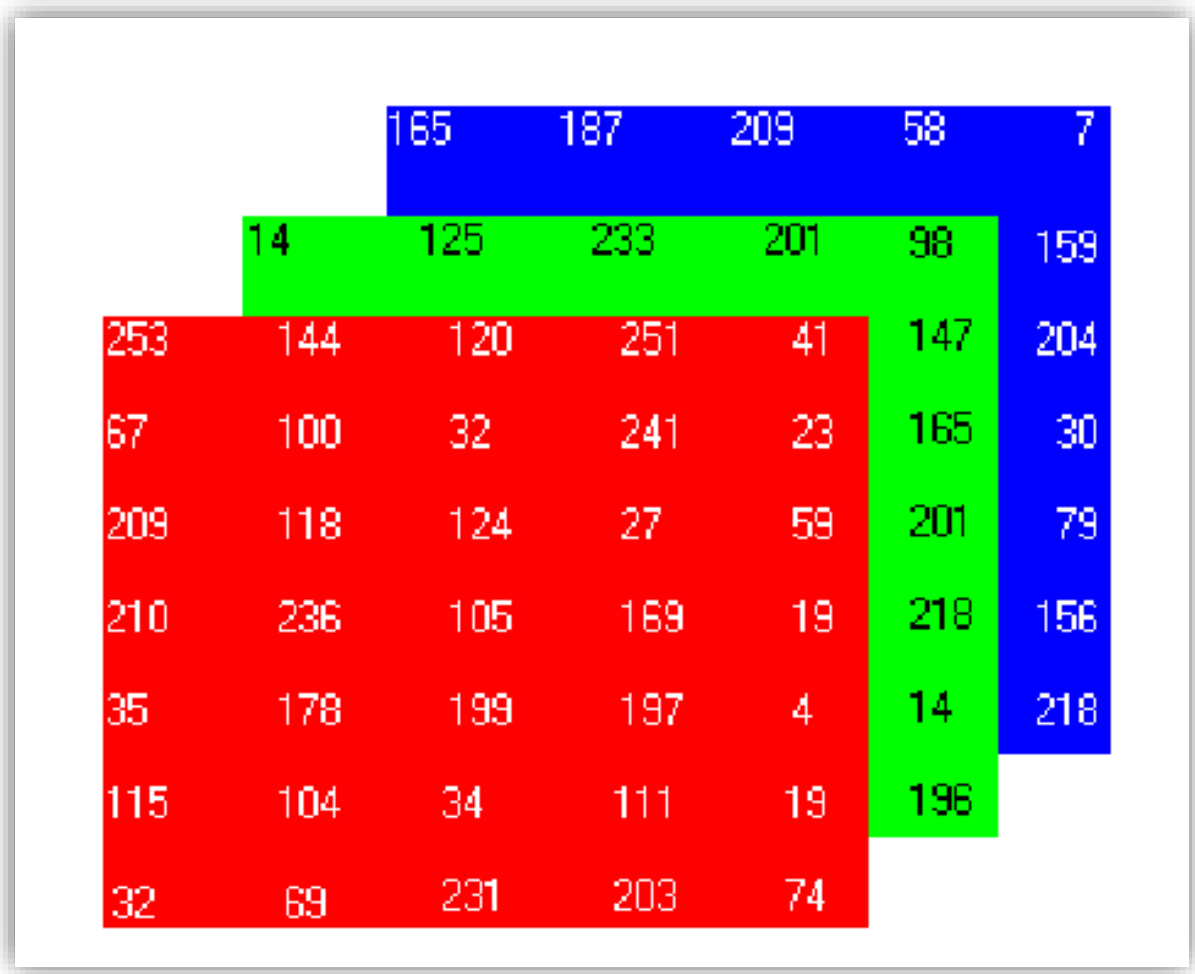


Figura 4-3. Matriz imagen RGB

En nuestro caso, trabajaremos con el rango de color HSV (*Hue, Saturation, Value*), que se trata de una transformación no lineal del espacio de color RGB.

La elección de dicho modelo y no RGB se debe simplemente a que a la hora de elegir nuestro umbral para detectar el objeto era mucho más robusto con HSV que con RGB.

Mediante una función de OpenCV convertiremos nuestra matriz RGB en una matriz HSV.

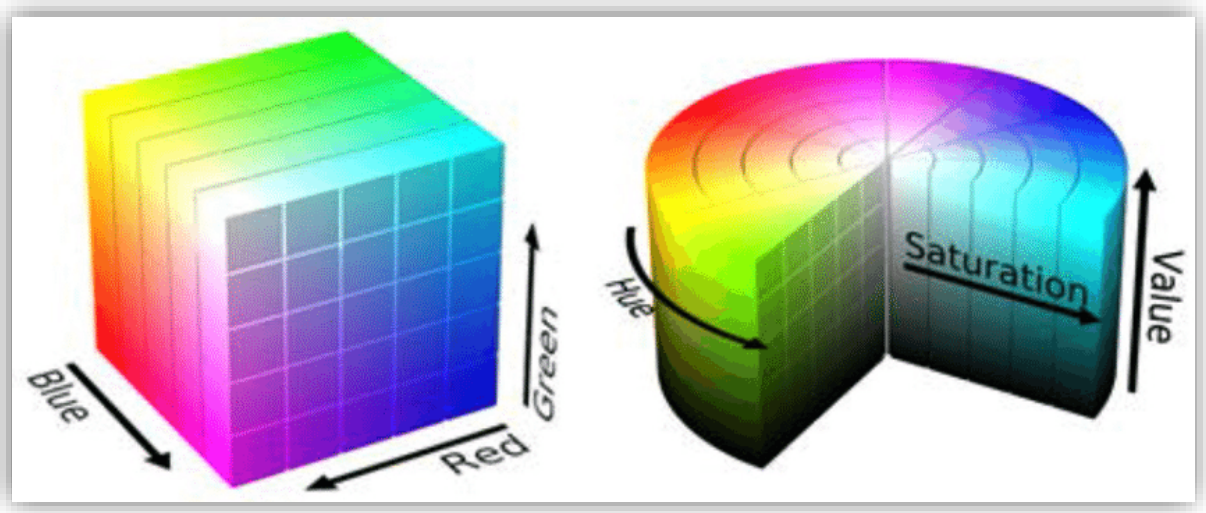


Figura 4-4. Espacio de color RGB y HSV

Aquí podemos ver el código que se encarga de esto:

```
// Inicializamos nuestra cámara y comprobamos si hay algún error

     VideoCapture capture(0);
     if (!capture.isOpened())
     {
          cerr << "ERROR! \n";
          getchar();
          return -1;
     }

//  Establecemos  la  resolución  y  los  fps  que  queremos  en
nuestra cámara

     capture.set(CV_CAP_PROP_FRAME_WIDTH, 100);
     capture.set(CV_CAP_PROP_FRAME_HEIGHT, 100);
     capture.set(CV_CAP_PROP_FPS, 60);

// Guardamos la imagen que está capturando la cámara en este instante
y la guardamos en la variable frame_RGB

     capture >> frame_RGB;
```

```
// Transformamos la imagen de RGB a HSV


     cvtColor(frame_RGB, frame_HSV, COLOR_BGR2HSV);
//Suavizamos la imagen para eliminar ruido, este proceso es opcional,
pero da mejores resultados y no tiene demasiado impacto en el
rendimiento


     medianBlur(frame_HSV, frame_HSV, 5);


// Aplicamos un umbral y guardamos una matriz binarizada en la
varibale binary

     inRange(frame_HSV, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH,
     iHighS, iHighV), binary);
```

Para cada una de las variables HSV, tenemos un umbral inferior y superior, los cuales calculamos de manera experimental en una fase previa de calibración.

Para hacer más fácil esta tarea y poder modificar estos valores en tiempo real y no tener que estar compilando y ejecutando el código cada vez que se modificase cada uno de estos 6 valores, se implementó un sistema de barras:

```
// Valores iniciales filtro HSV

     int iLowH = 0;
     int iHighH = 36;

     int iLowS = 104;
     int iHighS = 219;

     int iLowV = 140;
     int iHighV = 255;

// Barras para graduar el filtro del espacio HSV

     createTrackbar("LowH", "HSV", &iLowH, 255);
     createTrackbar("HighH", "HSV", &iHighH, 255);

     createTrackbar("LowS", "HSV", &iLowS, 255);
     createTrackbar("HighS", "HSV", &iHighS, 255);

     createTrackbar("LowV", "HSV", &iLowV, 255);
     createTrackbar("HighV", "HSV", &iHighV, 255);
```

Con todo esto, habremos capturado nuestra imagen con la cámara, y transformado en una matriz binarizada que contiene solo los pixeles de nuestro objeto.

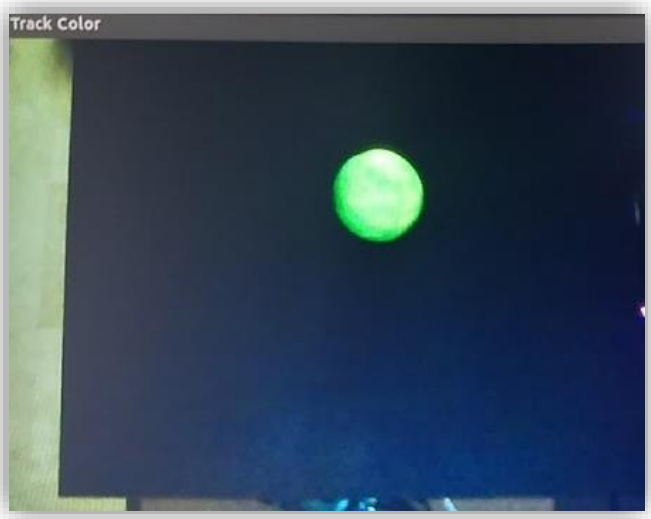


Figura 4-5. Imagen en RGB

Figura 4-6. Imagen en HSV

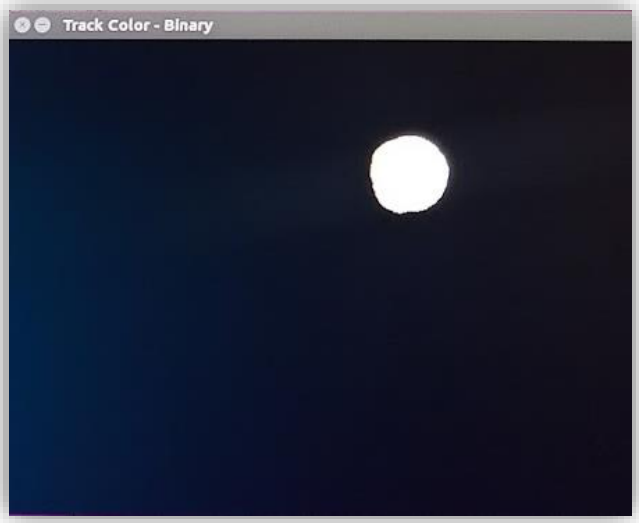


Figura 4-7. Imagen binarizada

Una vez que tenemos esta matriz binarizada que contiene solo los pixeles que ocupa nuestro objeto en la imagen tenemos que calcular el centro de dicha masa de pixeles.

Como ya se comentó en el apartado 3.3.2, se usará el método de los momentos invariantes. En OpenCV existen funciones que realizan este método, pero aquí se implementó de forma directa:

```
// Calculamos los momentos geométricos de la imagen binarizada

      m00 = 0;
      m01 = 0;
      m10 = 0;

      for (j = 1; j<columna; j++)
            for (i = 1; i<fila; i++)
            {
                  valor = (binary.at<char>(i, j));
                  m00 += valor;
                  m10 += i * valor;
                  m01 += j * valor;
            }
      if (m00 > 15000)
      {
            center.x = round((int)(m01 / m00));
            center.y = round((int)(m10 / m00));
            printf("LA BOLA ESTA EN LA PLATAFORMA\n");
      }
      else
      {
            center.x = 400;
            center.y = 260;
            printf("LA BOLA NO ESTA EN LA PLATAFORMA\n");
      }
```

Es interesante comentar que el valor m00 no es más que la cantidad de pixeles total de nuestro objeto. Por lo tanto, si dicho objeto a la distancia que esta de la cámara suele tener un valor mayor de 15000, podemos saber si el objeto esta delante de la pantalla o no. Esto es bastante útil, ya que como podemos ver en el código, para calcular el centro tenemos que dividir por m00, y si m00 tiene un valor nulo da problemas. De forma que si el objeto no está en la plataforma damos el centro de la plataforma para que los actuadores dejen la plataforma horizontal.

Una vez que ya tenemos nuestra coordenada que nos permite saber dónde se encuentra el centro de nuestro objeto en la plataforma, ya hemos acabado con la fase de percepción y cálculo de la entrada y ya estamos por tanto en disposición de calcular nuestras salidas en función de dichas entradas.

Como se trató en el apartado 3.2, usaremos un PID para controlar nuestro sistema y lo discretizaremos mediante una aproximación bilineal, aquí podemos ver la implementación en el código de dicho algoritmo.

```
// Transformamos pixeles a m

     posicion_x = (center.x *0.2) /230;
     posicion_y = (center.y *0.2) /230;

// Calcular errores actuales

     error_x[0] = posicion_x - referencia_x_d;
     error_y[0] = posicion_y - referencia_y_d;


// Calcular q0, q1, q2

     Ti_x_d = (double)Ti_x/1000;
     Ti_y_d = (double)Ti_y/1000;
     Td_x_d = (double)Td_x/1000;
     Td_y_d = (double)Td_y/1000;


     q0_x = Kp_x*((1+(T/(2*Ti_x_d))+(Td_x_d/T)));
     q1_x = Kp_x*((T/(2*Ti_x_d))-1-(2*(Td_x_d/T)));
     q2_x = Kp_x*(Td_x_d / T);

     q0_y = Kp_y*((1+(T/(2*Ti_y_d))+(Td_y_d/T)));
     q1_y = Kp_y*((T/(2*Ti_y_d))-1-(2*(Td_y_d/T)));
     q2_y = Kp_y*(Td_y_d/T);


// Calcular la acción de control en función de q0, q1, q2 y los errores
de los últimos 3 ciclos

     uk_x = q0_x * error_x[0] + q1_x * error_x[1] + q2_x *
     error_x[2]+last_uk_x;
     uk_y = q0_y * error_y[0] + q1_y * error_y[1] + q2_y * error_y[2] +
     last_uk_y;


//  Calcular los ángulos, PF son los ángulos en el punto de
funcionamiento, es decir plataforma horizontal

     angulo_x = uk_x + PF_x;
     angulo_y = uk_y + PF_y;

// Limitamos la modificación de los ángulos a +- 10, para evitar cambios
muy bruscos en la plataforma y el sobre control que haga inestable al
sistema


     if (angulo_x > (PF_x + 10))
          angulo_x = PF_x + 10;
     if (angulo_x < (PF_x - 10))
          angulo_x = PF_x - 10;
     if (angulo_y > (PF_y + 10))
          angulo_y = PF_y + 10;
     if (angulo_y < (PF_y - 10))
     angulo_y = (PF_y - 10);
```

```
// Actualizar uk-1, ek-1, ek-1

      last_uk_x = uk_x;
      last_uk_y = uk_y;

      error_x[2] = error_x[1];
      error_x[1] = error_x[0];
      error_y[2] = error_y[1];
      error_y[1] = error_y[0];
```

Después de esto ya tenemos los valores de los ángulos que el servomotor debe aplicar para reducir el error en la posición del objeto. Los servomotores están controlados por la placa Arduino, así que necesitamos enviar los valores de los ángulos de la placa Jetson TK1 a la placa Arduino.

4.3.2 Protocolo comunicación placa Arduino

Para comunicar ambas placas usaremos un puerto serie, aquí podemos ver la inicialización del puerto serie en el programa que ejecuta la Jetson TK1:

```
// Descriptor de fichero para el puerto serie de comunicación


      #define    DEV   "/dev/ttyACM0"
      int fd;
      struct termios toptions;

// Abrir el puerto serie de comunicación con Arduino

      fd = open(DEV, O_RDWR | O_NOCTTY);

      if (fd < 0)
      {
            printf("Error abriendo el canal de comunicación en
                        %s\n",DEV);
            exit(-1);
      }

      printf("Canal de comunicación en %s abierto correctamente: %i\n",
             DEV, fd);

// Establecimiento de conexión con Arduino

      printf("Esperando a Arduino para arrancar\n");
      usleep(3500000);
      printf("Conectado correctamente con Arduino\n");
```

Tras el cálculo de los ángulos visto en el apartado anterior los enviaremos de esta forma:

```
      data[0] = angulo_y;
      data[1] = '\0';
      write(fd, data, 1);
      data[0] = angulo_x;
      data[1] = '\0';
      write(fd, data, 1);
```

4.3.3 Arduino: movimiento servomotores

Este programa es mucho más sencillo, simplemente estará en todo momento esperando que le lleguen valores por el puerto serie y los aplica a cada uno de los servomotores.

Aquí está el código completo que ejecuta la placa Arduino:

```
// Variable global para control del servomotor

      Servo servoMotor_1;
      Servo servoMotor_2;
      int motor = 0;

// Método de configuración inicial

      void setup()

// Inicialización de la comunicación por el puerto serie

      Serial.begin(9600);

// Asignación del servomotor al pin 9

      servoMotor_1.attach(9);
      servoMotor_2.attach(10);

// Método de ejecución continua

      void loop()
            byte angulo;
// Detectamos si hemos recibido un dato (byte)
// en el puerto serie

      if (Serial.available() > 0)

// Leemos el byte como valor de la variable ángulo

      angulo = Serial.read();

// Indicamos al servomotor que realice el movimiento
// según el valor de ángulo leído

      if(motor == 0) {
            servoMotor_2.write(angulo);
            motor = 1;
      } else {
            servoMotor_1.write(angulo);
            motor = 0;
      }
```

5 ANÁLISIS EXPERIMENTAL

Se realizaron tres pruebas diferentes: la más básica era simplemente llevar el objeto al centro de la plataforma, llevar el objeto a las cuatro esquinas de la plataforma y, por último, usar una referencia dinámica para que el objeto describiese una circunferencia en la plataforma.

Se consiguió controlar el objeto en dichas tres pruebas, aquí podemos ver algunas graficas de los resultados:

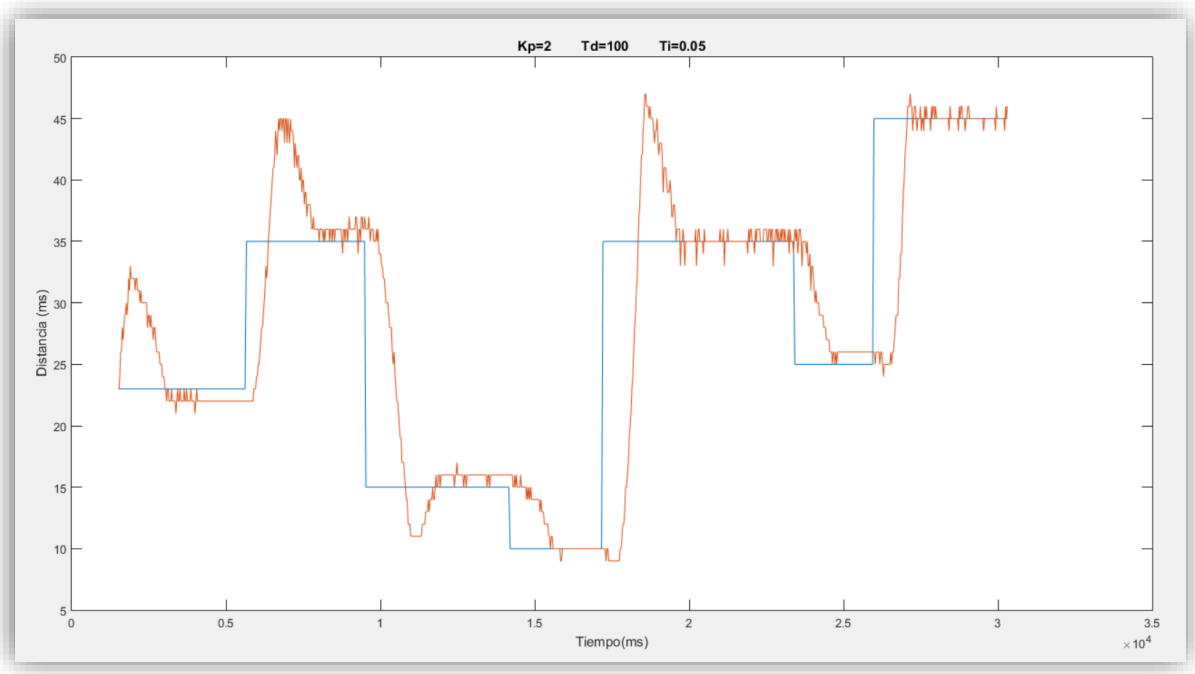


Figura 5-1. Gráfica del resultado de la prueba cambiando la referencia de posición I

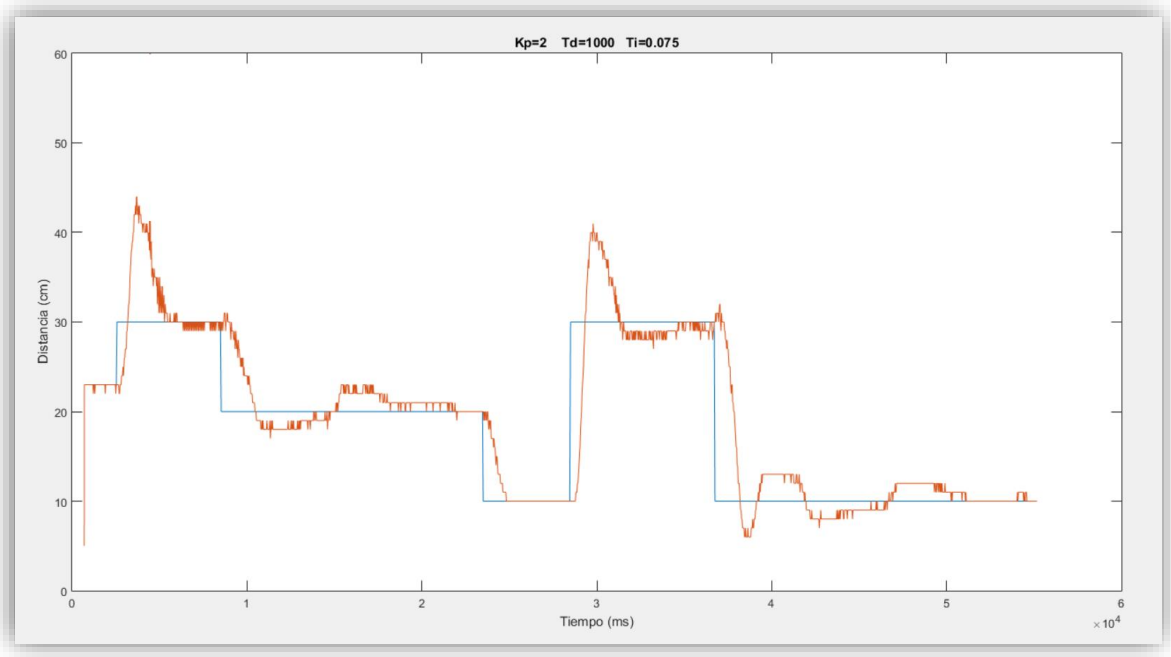


Figura 5-2. Gráfica del resultado de la prueba cambiando la referencia de posición II

El resultado de estas tres pruebas se puede ver en los vídeos que se anexan a este proyecto.

Aquí podemos ver una captura de la primera de las pruebas en las que simplemente la referencia es el centro de la plataforma:

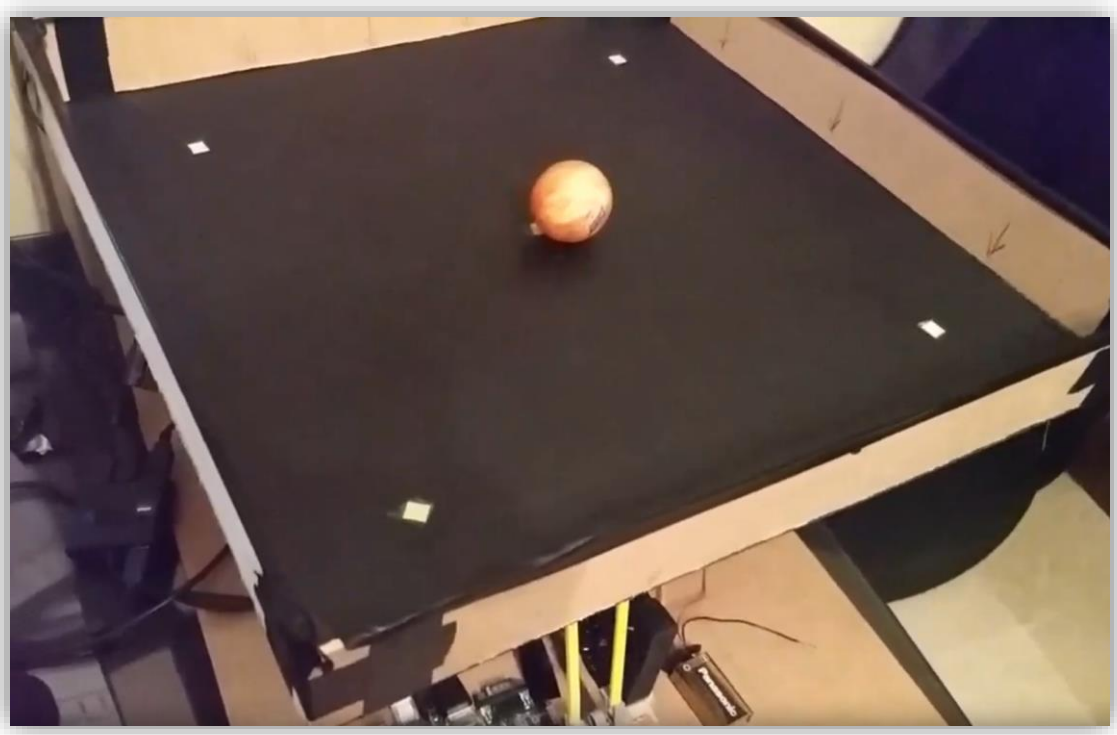


Figura 5-3. Prueba con el centro como referencia

Capturas del vídeo de la segunda prueba de llevar el objeto a las cuatro esquinas de la plataforma:

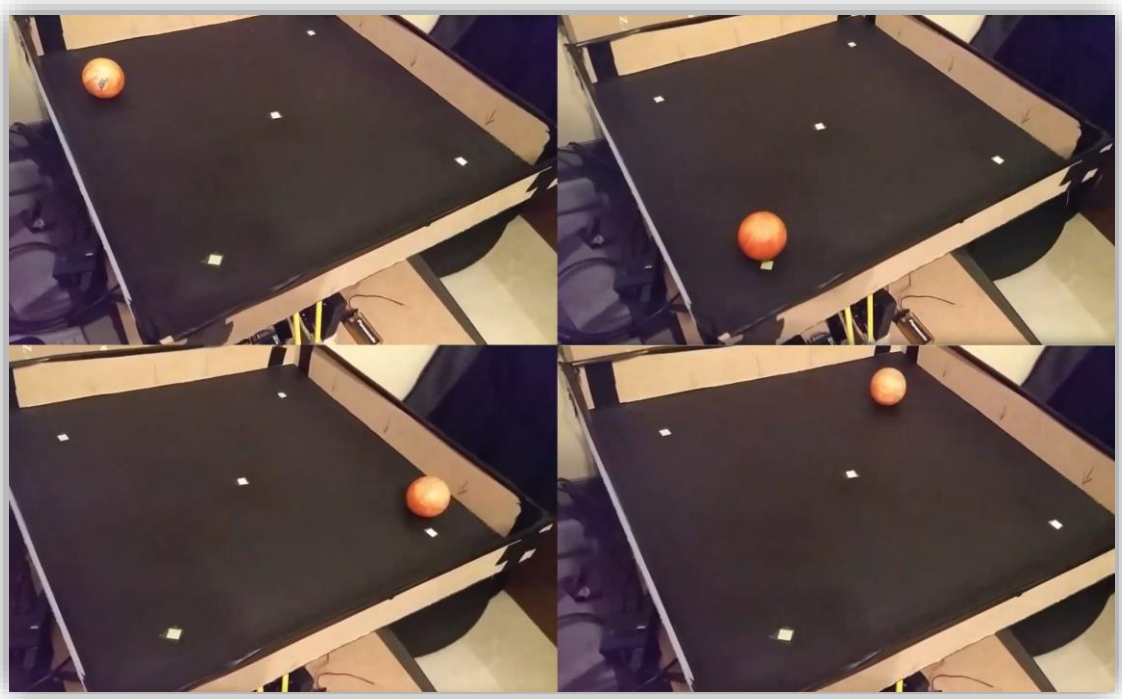


Figura 5-4. Prueba con las esquinas como referencias

Para la última prueba era difícil de representar con imágenes estáticas, así que se superpusieron varios frames del vídeo:

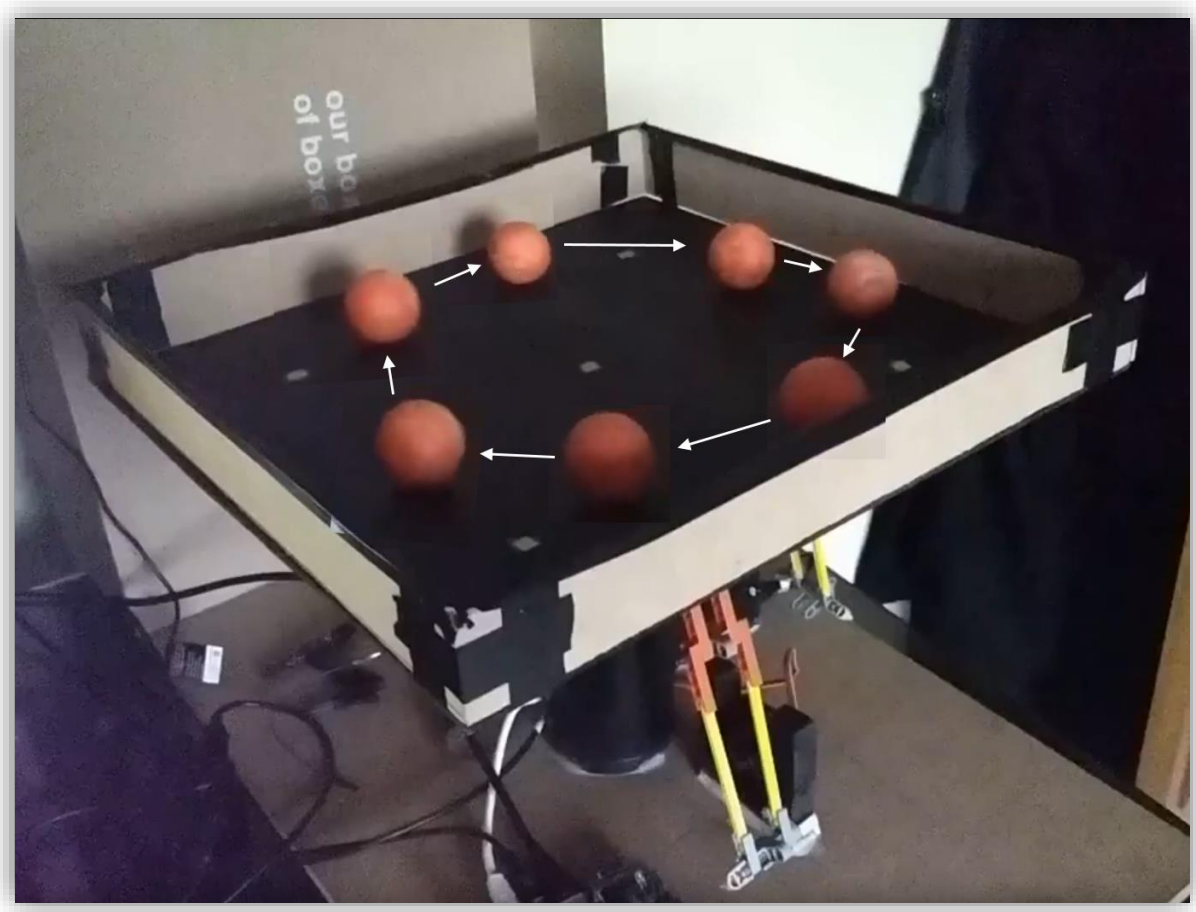


Figura 5-5. Prueba describiendo una circunferencia

6 PROBLEMAS EN EL DESARROLLO

6.1 Problemas en la construcción del mecanismo

Uno de los mayores problemas fue el encontrar piezas adecuadas para las bielas del mecanismo, ya que fue difícil encontrar soluciones adecuadas entre las opciones del mercado.

En primera instancia, se intentó usar bielas de componentes de juguetes RC, pero no dio buen resultado ya que eran componentes muy pequeños y el movimiento de la plataforma no era preciso ya que bailaba bastante.

6.2 Problemas con la Jetson TK1

El flasheo de la Jetson TK1 genero muchas dificultades, la principal es que la instalación es bastante confusa porque se puede hacer de muchas formas en función de la versión de L4T.

Otro problema es que el entorno para realizar dicha instalación es muy poco flexible y está lleno de incompatibilidades. Es necesario que el host con el que realizar el flasheo tenga una versión de Ubuntu especifica según la versión que vayas a instalar y, una vez más, no está del todo bien especificado, teniendo que instalar varias versiones de Ubuntu diferentes y realizar bastantes pruebas. Por si fuera poco, poner la tarjeta en el modo de flasheo no es fácil, y además se queda congelado bastantes veces a mitad del flasheo, teniendo que empezar la operación otra vez desde el principio.

Como se ha comentado al inicio, al haber múltiples formas de hacerlo no todas llegan a la misma conclusión. Esto es debido a que puedes instalar L4T con el pack de CUDA o sin él, y hay algunas versiones que están bastante abandonadas por parte de NVIDIA y dan constantes problemas, teniendo que flashear de nuevo la placa y empezar todo de cero.

La instalación de OpenCV tampoco es un reto sencillo, ya que es un proceso largo que también se puede quedar congelado y al haber tantas versiones y tantas formas de instalarlo puede llegar a ser algo confuso. Además, que el OpenCV que tenemos que instalar no es el que instalaríamos de manera habitual en un equipo con Ubuntu, lo cual hace el problema algo más difícil.

6.3 Problemas con la retroalimentación visual

El principal inconveniente en este ámbito fue la optimización del proceso de obtención de la imagen y tratamiento de ésta, que era con diferencia lo más costoso a nivel de rendimiento en cada ciclo. La idea en un principio era utilizar los núcleos CUDA con los que cuenta la placa Jetson TK1 para paralelizar todo el trabajo con las imágenes con su GPU, ya que es mucho más eficiente al trabajar con imágenes.

El problema es que la imagen no se puede capturar con la GPU, de manera que tenemos que en cada ciclo pasar los datos de la imagen de la CPU a la GPU, y luego devolver la imagen binarizada de la GPU a la CPU. Estos tiempos de transporte de datos son tan costosos que hacían que el proceso con la GPU en cómputo global fuese peor que con la CPU.

Otra cuestión es que no todas las cámaras del mercado son compatibles con nuestra placa Jetson TK1, así que eso redujo mucho las posibilidades. De esta manera, al ser nuestro sistema tan rápido, era muy crítico tener un tiempo de ciclo total lo más breve posible para tener un buen control de éste. De hecho, comprobamos que valores de ciclo de 15-20 fps hacían que el sistema fuese completamente imposible de controlar debido a que se volvía inestable. Nuestra cámara tenía un máximo de 30 fps y, por lo tanto, teníamos que conseguir que el cuello de botella fuese ese tiempo de captura y no nuestro algoritmo. Finalmente, tras reducir la resolución de la cámara e intentar hacer más optimo y eficiente el algoritmo, conseguimos alcanzar casi los 30 fps haciendo que el sistema se controlase de una forma correcta. Probablemente con una cámara que llegase a 60 fps, o incluso más, el control seria aún mejor y no habría dado tantos problemas en este apartado.

Por último, describir las dificultades experimentadas con la percepción del objeto, el cual es el método que elegimos para obtener una imagen binarizada de los pixeles de nuestro objeto a controlar. El problema radicaba en que era muy variable en función de la luz de la habitación. Dicho de otra forma, el funcionamiento dependía mucho de la intensidad y de cómo incidiese la luz en nuestro sistema. Esto se pudo solventar usando el rango de colores HSV en vez del RGB que hacían mucho más robusto la detección del objeto a los cambios de luz, incluso a los propios reflejos de luz de la plataforma.

6.4 Problemas con los servomotores y la alimentación eléctrica

La primera vez que se cerró el bucle y se vio cómo funcionaba el control del sistema, ocurrió que los servomotores apenas se movían. El problema era que los servomotores eran poco potentes y no tenían torque para mover el peso de la plataforma usada, por lo que se tuvo que comprar servomotores más potentes.

La sorpresa fue que, al probar el funcionamiento con los nuevos servomotores, esta vez sí que podían con el peso de la plataforma, pero se movían de forma errática y como dando unos escalones demasiado rápidos. E problema, pero era que ambos motores estaban conectados a la placa Arduino y no tenía suficiente tensión para alimentar ambos servomotores.

La solución fue usar un trasformador de 6V para alimentar en paralelo a ambos servomotores de forma externa, con esto ya sí que empezaron a funcionar los servomotores de la manera esperada.

7 CONCLUSIONES Y TRABAJO FUTURO

Una vez analizados todos los factores tenidos en cuenta en el presente proyecto, detallamos una serie de medidas que serían interesantes incorporar en futuros proyectos.

En primer lugar, una de las primeras mejoras que se realizaría a dicho proyecto sería la actualización del mecanismo de las bielas por unas más robustas que diesen algo más de precisión al control de la plataforma.

En segúndo lugar, sería muy relevante usar una cámara que tuviese mayor tasa de fps, puesto que quedó comprobado que, al ser un sistema tan rápido, con cada pocos fps que se ganaban en el ciclo del proceso el control mejoraba sustancialmente.

En tercer lugar, en el apartado del software sería interesante implementar alguna función de autocalibración, de manera que el programa calculase al inicio del proceso unos valores óptimos de umbrales para el rango HSV. De igual forma, sería interesante no sólo capturar la bola, sino también la plataforma, de manera que podríamos mover la cámara en pleno control y seguiría calculándose de forma dinámica el control de la bola en la plataforma.

Para finalizar, el principal trabajo que queda pendiente en este proyecto, que de hecho en un principio iba a realizarse, pero debido a los problemas surgidos y las complicaciones se desestimó, es el comparar el control clásico PID con el aprendizaje automático. Es decir, usando el método de control que mejor funcionase, construir una base de datos realizando un gran número de pruebas y luego generar un modelo con diferentes técnicas de aprendizaje automático y ver si mejora el control con respecto al control automático clásico. La implementación de estas técnicas es relativamente sencilla en este proyecto, ya que las herramientas más usadas en aprendizaje automático como *Scikit-learn*, *Tensorflow*, *PyTorch* son fáciles de usar en Ubuntu.

Este trabajo puede considerarse la fase inicial de investigación que propicie el desarrollo de futuros proyectos. De esta manera, las conclusiones de los resultados analizados pueden contribuir en la realización de proyectos relacionados a los que sería interesante implementarles estas mejoras y observar el comportamiento del sistema.

ANEXO A: CÓDIGO

Un buen código es su mejor documentación.

Steve McConnell

Código Arduino

```
// Inclusion de ficheros de cabeceras - dependencias

#include <Servo.h>

// Variable global para control del servomotor

Servo servoMotor_1;
Servo servoMotor_2;
int motor = 0;

// Metodo de configuracion inicial

void setup() {

  // Inicializacion de la comunicacion por el puerto serie

  Serial.begin(9600);

  // Asignacion del servomotor al pin 9

  servoMotor_1.attach(9);
  servoMotor_2.attach(10);
}
// Metodo de ejecucion continua
void loop() {
  byte angulo;
  // Detectamos si hemos recibido un dato (byte)
  // en el puerto serie
  if (Serial.available() > 0) {
    // Leemos el byte como valor de la variable angulo
    angulo = Serial.read();
    // Indicamos al servomotor que realice el movimiento
    // segun el valor de angulo leido

    if (motor == 0) {
      servoMotor_1.write(angulo);
      motor = 1;
    } else {
      servoMotor_2.write(angulo);
      motor = 0;
    }
  }
}
```

Código Jetson TK1

```
#include <stdio.h>
#include <string>
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <ctime>
#include <stdlib.h>
#include <ctime>
#include <unistd.h>
#include <stdint.h>
#include <fcntl.h>
#include <termios.h>
#include <errno.h>
#include <sys/ioctl.h>
#include <math.h>

#define DEV "/dev/ttyACM0"
#define PI 3.14159265
using namespace cv;
using namespace std;

int main() {

	int fd; // Descriptor de fichero para el puerto serie de comunicación

	struct termios toptions;

	/* Abrir el puerto serie de comunicación con Arduino */
	fd = open(DEV, O_RDWR | O_NOCTTY);

	if (fd < 0) {
		printf("Error abriendo el canal de comunicación en %s\n", DEV);
		exit(-1);
	}
	printf("Canal de comunicación en %s abierto correctamente: %i\n", DEV, fd);
```

```
/* Establecimiento de conexión con Arduino */
printf("Esperando a Arduino para arrancar\n");
usleep(3500000);
printf("Conectado correctamente con Arduino\n");

/* obtenemos la configuracion del puerto actual */
tcgetattr(fd, & toptions);
cfsetispeed( & toptions, B9600);
cfsetospeed( & toptions, B9600);
toptions.c_cflag &= ~PARENB;
toptions.c_cflag &= ~CSTOPB;
toptions.c_cflag &= ~CSIZE;
toptions.c_cflag |= CS8;
toptions.c_lflag |= ICANON;
/* confirmamos la configuracion del puerto serie */
tcsetattr(fd, TCSANOW, & toptions);

int number = 0;
int data[10];
int t = 90;

clock_t inicio_total, final_total;
double total_frames = 0;
double total_secs = 0;
double frame_clock, frame_secs;
double average_secs_by_frame;
double frames_per_sec;
double fps;
double MAX_frame;
int REF = 0;
double posicion_x, uk_x, last_uk_x = 0, angulo_x, referencia_x_d = 0.34;
double posicion_y, uk_y, last_uk_y = 0, angulo_y, referencia_y_d = 0.22;
int referencia_x = 34;
int referencia_y = 22;
int PF_x = 102;
int PF_y = 117;
double T = 1.0 / 25;
```

```
double q0_x = 0;
double q1_x = 0;
double q2_x = 0;
int Kp_x = 50;
int Ti_x = 2200;
int Td_x = 800;
int Kp_x_d = 65;
double Ti_x_d = 0.9;
double Td_x_d = 0.7;
double q0_y = 0;
double q1_y = 0;
double q2_y = 0;
int Kp_y = 50;
int Ti_y = 2200;
int Td_y = 800;
double Ti_y_d = 0.9;
double Td_y_d = 0.7;
int cont = 0;

double error_x[3];
error_x[0] = 0;
error_x[1] = 0;
error_x[2] = 0;

double error_y[3];
error_y[0] = 0;
error_y[1] = 0;
error_y[2] = 0;

Point center;

int i, j, fila, columna, valor;
double m00, m01, m10;

Mat frame, gray, binary, HSV;

namedWindow("Cam", CV_WINDOW_AUTOSIZE);
namedWindow("HSV", CV_WINDOW_AUTOSIZE);
```

```
namedWindow("Control", CV_WINDOW_AUTOSIZE);
namedWindow("Binary", WINDOW_AUTOSIZE);
namedWindow("REF", CV_WINDOW_AUTOSIZE);

// Valores inciales filtro HSV

int iLowH = 0;
int iHighH = 36;

int iLowS = 104;
int iHighS = 219;

int iLowV = 140;
int iHighV = 255;

// Barras para graduar el filtro del espacio HSV

createTrackbar("LowH", "HSV", & iLowH, 255);
createTrackbar("HighH", "HSV", & iHighH, 255);

createTrackbar("LowS", "HSV", & iLowS, 255);
createTrackbar("HighS", "HSV", & iHighS, 255);

createTrackbar("LowV", "HSV", & iLowV, 255);
createTrackbar("HighV", "HSV", & iHighV, 255);

createTrackbar("PF_x", "REF", & PF_x, 180);
createTrackbar("PF_y", "REF", & PF_y, 180);
//createTrackbar("Ref_x", "REF", &referencia_x, 100);
//createTrackbar("Ref_y", "REF", &referencia_y, 100);
createTrackbar("REF", "REF", & REF, 4);

createTrackbar("Kp_x", "Control", & Kp_x, 100);
createTrackbar("Kp_y", "Control", & Kp_y, 100);
createTrackbar("Ti_x", "Control", & Ti_x, 10000);
createTrackbar("Ti_y", "Control", & Ti_y, 10000);
createTrackbar("Kd_x", "Control", & Td_x, 1000);
createTrackbar("Td_y", "Control", & Td_y, 1000);
```

```
VideoCapture capture(0);
if (!capture.isOpened()) {
        cerr << "ERROR! No se pudo iniciar la camara\n";
        getchar();
        return -1;
}

// Establecemos la resolucion que queremos en nuestra camara

//capture.set(CV_CAP_PROP_FRAME_WIDTH, 100);
//capture.set(CV_CAP_PROP_FRAME_HEIGHT, 100);
//capture.set(CV_CAP_PROP_FPS, 60);

for (;;) {

        referencia_x_d = (double) referencia_x / 100;
        referencia_y_d = (double) referencia_y / 100;

        referencia_x_d = (double) referencia_x / 100;
        referencia_y_d = (double) referencia_y / 100;

        if (REF == 0) {
                referencia_x = 0.35;
                referencia_y = 0.20;
        }

        if (REF == 1) {
                referencia_x = 0.215;
                referencia_y = 0.1;
        }

        if (REF == 2) {
                referencia_x = 0.45;
                referencia_y = 0.1;
        }

        if (REF == 3) {
```

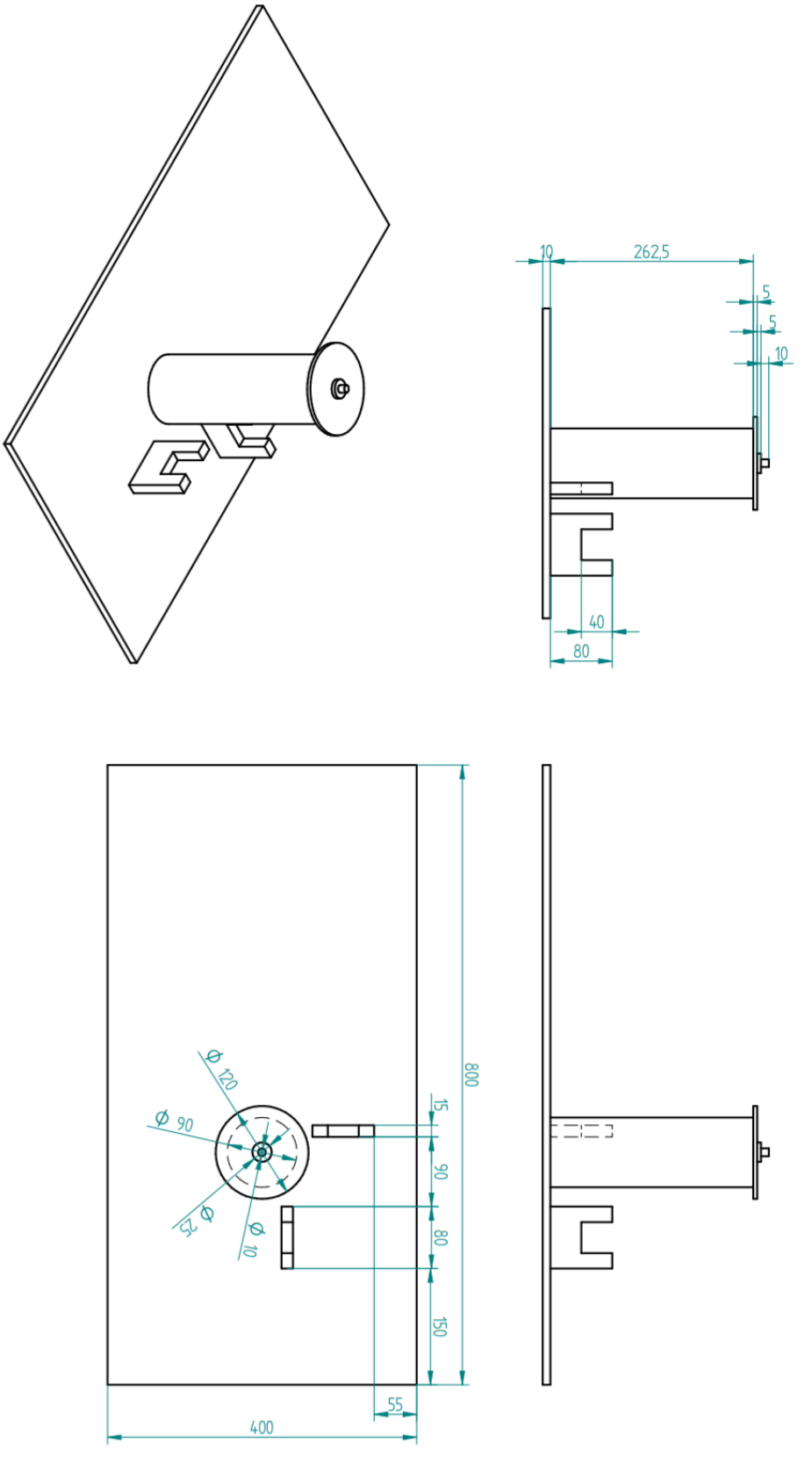
```
        referencia_x = 0.45;
        referencia_y = 0.32;
    }

    if (REF == 4) {
        referencia_x = 0.215;
        referencia_y = 0.32;
    }

    //referencia_x_d = 0.12 * cos(t*PI/180) + 0.35;
    //referencia_y_d = 0.1 * sin(t*PI/180) + 0.22;

    inicio_total = clock();

    capture >> frame;

    //¡¡¡SOLO EJECUTAR ESTA LINEA TARDA 150 ms!!!

    fila = frame.rows;
    columna = frame.cols;

    // Pasamos la imagen al espacio HSV

    cvtColor(frame, HSV, COLOR_BGR2HSV);

    //medianBlur(HSV, HSV, 5); // Suavizamos la imagen para conseguir un mejor resultado (OPCIONAL)

    // Aplicamos el filtro

    inRange(HSV, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH, iHighS, iHighV), binary);

    // CALCULAMOS LOS MOMENTOS GEOMETRICOS DE LA IMAGEN BINARIZADA

    m00 = 0;
    m01 = 0;
    m10 = 0;
```

```
for (j = 1; j < columna; j++)
        for (i = 1; i < fila; i++) {
                valor = (binary.at < char > (i, j));
                m00 += valor;
                m10 += i * valor;
                m01 += j * valor;
        }
if (m00 > 15000) {
        center.x = round((int)(m01 / m00));
        center.y = round((int)(m10 / m00));
        printf("LA BOLA ESTA EN LA PLATAFORMA\n");
} else {
        center.x = 400;
        center.y = 260;
        printf("LA BOLA NO ESTA EN LA PLATAFORMA\n");
}

//printf("m00= %f, m01= %f, m10= %f \n", m00, m01, m00);

// Calcular yk
posicion_x = (center.x * 0.2) / 230; // transformamos pixeles a m
posicion_y = (center.y * 0.2) / 230;

// Calcular ek
error_x[0] = posicion_x - referencia_x_d;
error_y[0] = posicion_y - referencia_y_d;

// Calcular uk

Ti_x_d = (double) Ti_x / 1000;
Ti_y_d = (double) Ti_y / 1000;
Td_x_d = (double) Td_x / 1000;
Td_y_d = (double) Td_y / 1000;

q0_x = Kp_x * ((1 + (T / (2 * Ti_x_d)) + (Td_x_d / T)));
q1_x = Kp_x * ((T / (2 * Ti_x_d)) - 1 - (2 * (Td_x_d / T)));
q2_x = Kp_x * (Td_x_d / T);
```

```
q0_y = Kp_y * ((1 + (T / (2 * Ti_y_d)) + (Td_y_d / T)));
q1_y = Kp_y * ((T / (2 * Ti_y_d)) - 1 - (2 * (Td_y_d / T)));
q2_y = Kp_y * (Td_y_d / T);

uk_x = q0_x * error_x[0] + q1_x * error_x[1] + q2_x * error_x[2] + last_uk_x;
uk_y = q0_y * error_y[0] + q1_y * error_y[1] + q2_y * error_y[2] + last_uk_y;

angulo_x = uk_x + PF_x;
angulo_y = uk_y + PF_y;

// Aplicar uk

if (angulo_x > (PF_x + 10))
      angulo_x = PF_x + 10;
if (angulo_x < (PF_x - 10))
      angulo_x = PF_x - 10;
if (angulo_y > (PF_y + 10))
      angulo_y = PF_y + 10;
if (angulo_y < (PF_y - 10))
      angulo_y = (PF_y - 10);

data[0] = angulo_y;
data[1] = '\0';
write(fd, data, 1);

data[0] = angulo_x;
data[1] = '\0';
write(fd, data, 1);

// Actualizar uk-1, ek-1, ek-1

last_uk_x = uk_x;
last_uk_y = uk_y;

error_x[2] = error_x[1];
error_x[1] = error_x[0];
error_y[2] = error_y[1];
error_y[1] = error_y[0];
```

```
        //imshow("Cam", frame);
        imshow("Binary", binary);

        final_total = clock();
        frame_clock = final_total - inicio_total;
        frame_secs = frame_clock / CLOCKS_PER_SEC;
        total_frames++;
        total_secs += frame_secs;
        average_secs_by_frame = total_secs / total_frames;
        frames_per_sec = 1.0 / average_secs_by_frame;
        fps = 1.0 / frame_secs;

        printf("%f\n", frame_clock / CLOCKS_PER_SEC);
        if (frame_clock > MAX_frame)
            MAX_frame = frame_clock;

        printf("%f\n\n", MAX_frame / CLOCKS_PER_SEC);

        printf("Posicion = %f %f\n", posicion_x, posicion_y);
        printf("T = %d \n", t);
        printf("Referencia = %f %f\n", referencia_x_d, referencia_y_d);
        printf("Error = [%f %f %f]\n", error_x[0], error_x[1], error_x[2]);
        printf("Error = [%f %f %f]\n", error_y[0], error_y[1], error_y[2]);
        printf("Angulo = %f %f\n\n", angulo_x, angulo_y);
        printf("uk = %f %f\n", uk_x, uk_y);
        printf("q0 = %f %f\n", q0_x, q0_y);
        printf("q1 = %f %f\n", q1_x, q1_y);
        printf("q2 = %f %f\n\n", q2_x, q2_y);

        if (waitKey(30) >= 0) break;

    }

    getchar();

    return 0;
```

ANEXO B: DISEÑOS



BIBLIOGRAFÍA

- Kumar, Joselin & Showme, N. & Aravind, M. & Akshay, R.. (2016). Design and control of ball on plate system. 9. 765-778.
- Daniel Rodríguez Ramírez y Teodoro Alamo Cantarero, *Ingenieria de Control. Tema 6: Diseño de controladores discretos*: https://es.scribd.com/document/259102111/Tema-6-Diseno-Controladores-Aproximaciones-8
- Documentación Transformada de Hough OpenCV: https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html