Universidad Zaragoza
1542

Anexos del Trabajo Fin de Grado

CONTROL DE UN PUENTE GRÚA DIDÁCTICO MEDIANTE VISUALIZACIÓN DE SU CARGA
CONTROL OF A DIDACTIC 3D-CRANE THROUGH VISUALIZATION OF ITS LOAD

Autor/es

Daniel Redondo Sanz

Director/es

José Ramón Beltrán Blázquez

Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza

2020

CONTENIDO

Índice de anexos

Lista de figuras del anexo

Lista de tablas del anexo

Anexo 1: Programas utilizados para la aplicación de visión y pruebas de parámetros

Programa para la grabación de vídeo

```
import numpy as np
import cv2
cap = cv2.VideoCapture(1)
while(True):
  ret, frame = cap.read()
  cv2.imshow('Imagen original',frame)
  k = cv2.waitKey(30) & 0xFF
  if k==27:
      break
cap.release()
cv2.destroyAllWindows()
```

Programa para la obtención de la imagen en escala de grises

```
import numpy as np
import cv2
cap = cv2.VideoCapture(1)
while(True):
  ret, frame = cap.read()
  gris = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
  cv2.imshow('imagen grises',median)
  k = cv2.waitKey(30) & 0xFF
  if k==27:
      break
cap.release()
cv2.destroyAllWindows()
```

Programa para la obtención de la imagen en escala de grises filtrada (tamaño de kernel 15)

```
import numpy as np
import cv2
cap = cv2.VideoCapture(1)
while(True):
  ret, frame = cap.read()
  gris = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
  median = cv2.medianBlur(gris,15)
  cv2.imshow('imagen grises',median)
  k = cv2.waitKey(30) & 0xFF
  if k==27:
      break
cap.release()
cv2.destroyAllWindows()
```

Imágenes obtenidas en gris y filtradas con distintos tamaños de kernel

Figura 1. Imagen en escala de grises sin filtrar.

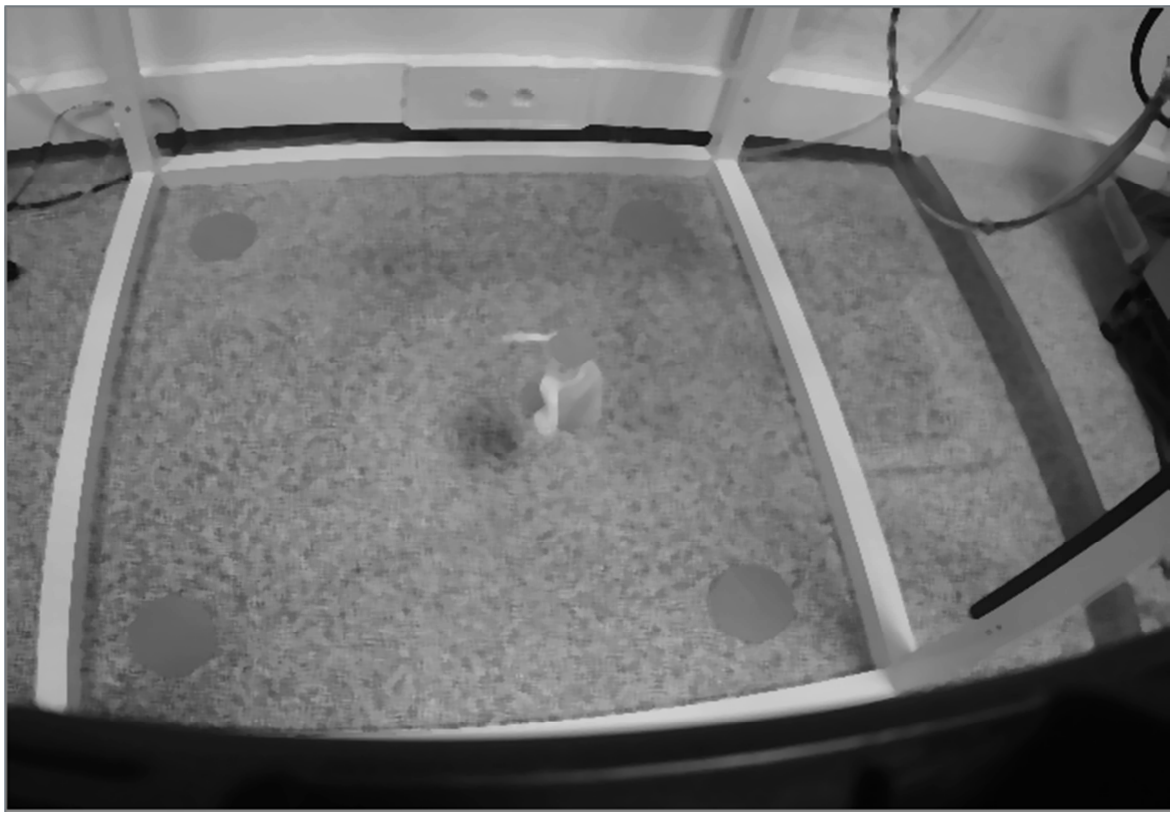

Figura 2. Imagen filtrada con tamaño de kernel 5.

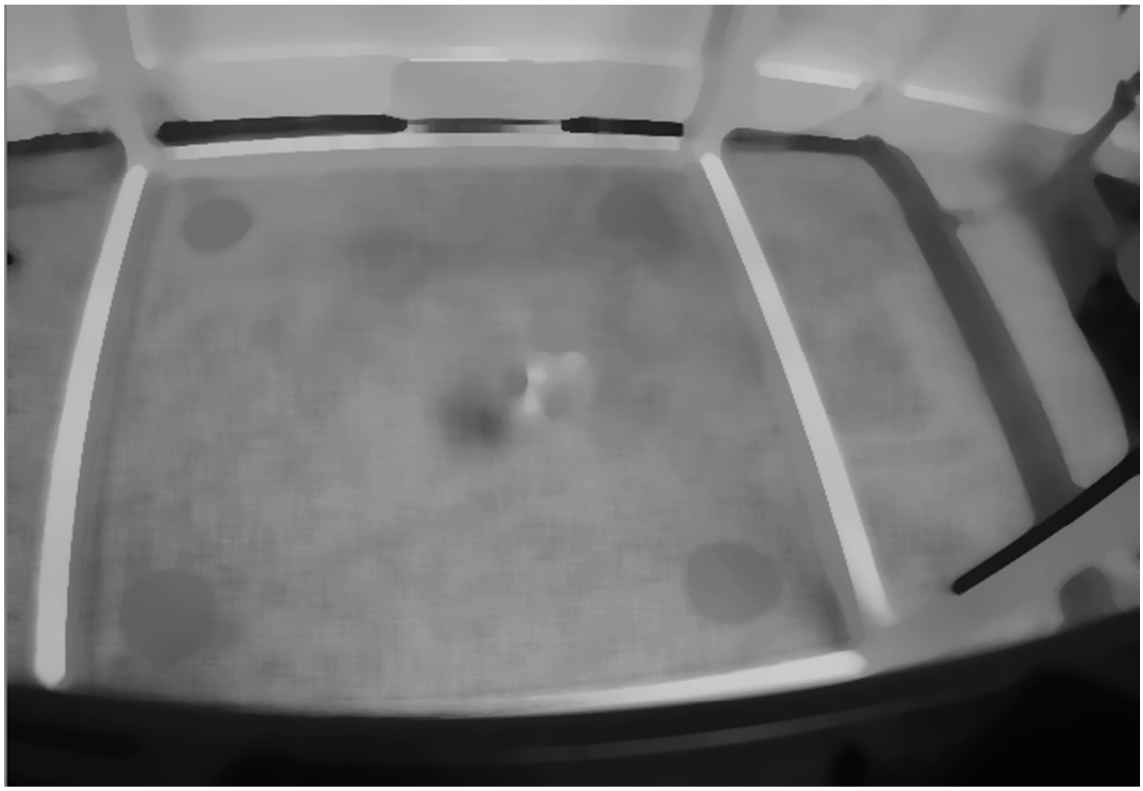

Figura 3. Imagen filtrada con tamaño de kernel 15.

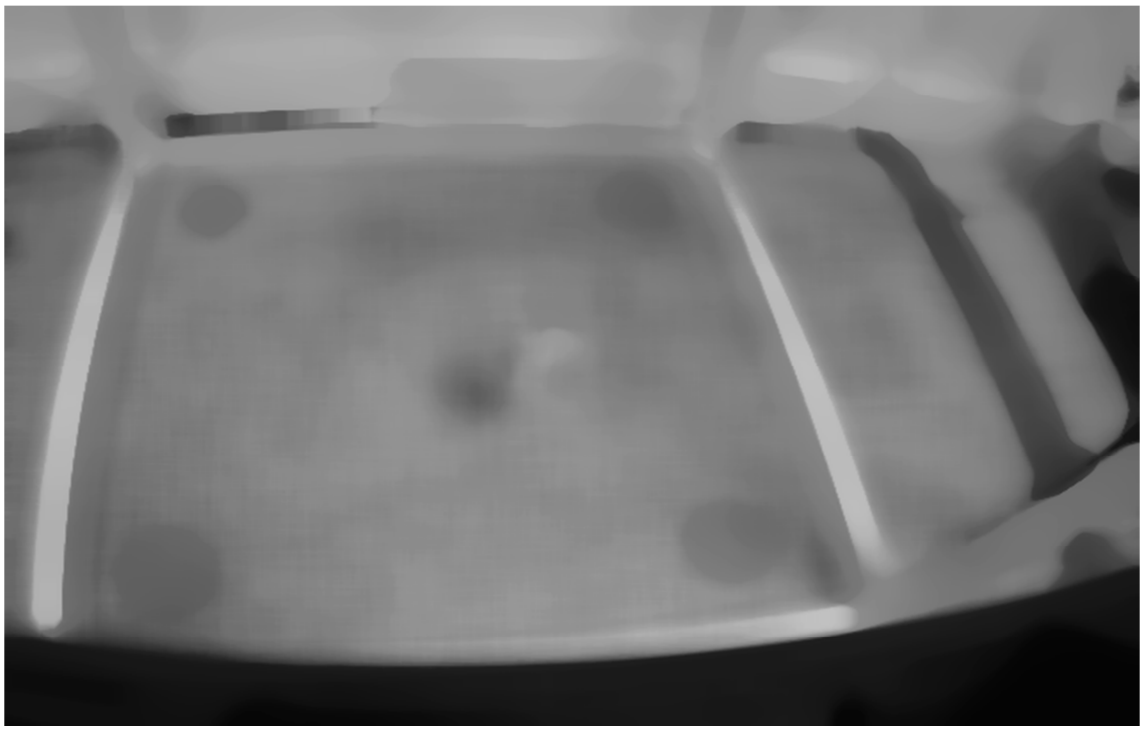

Figura 4. Imagen filtrada con tamaño de kernel 25.

Programa para la obtención de la imagen binaria por umbralización

```
import numpy as np
import cv2
cap = cv2.VideoCapture(1)
while(True):
  ret, frame = cap.read()
  gris = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
  median = cv2.medianBlur(gris,15)
  ret,thresh1 = cv2.threshold(median,l,255,cv2.THRESH_BINARY_INV)
  cv2.imshow('imagen binaria',thresh1)
  k = cv2.waitKey(30) & 0xFF
  if k==27:
      break
cap.release()
cv2.destroyAllWindows()
```

Imágenes binarias obtenidas por umbralización

Figura 5. Imagen binaria con k=5, l=90.

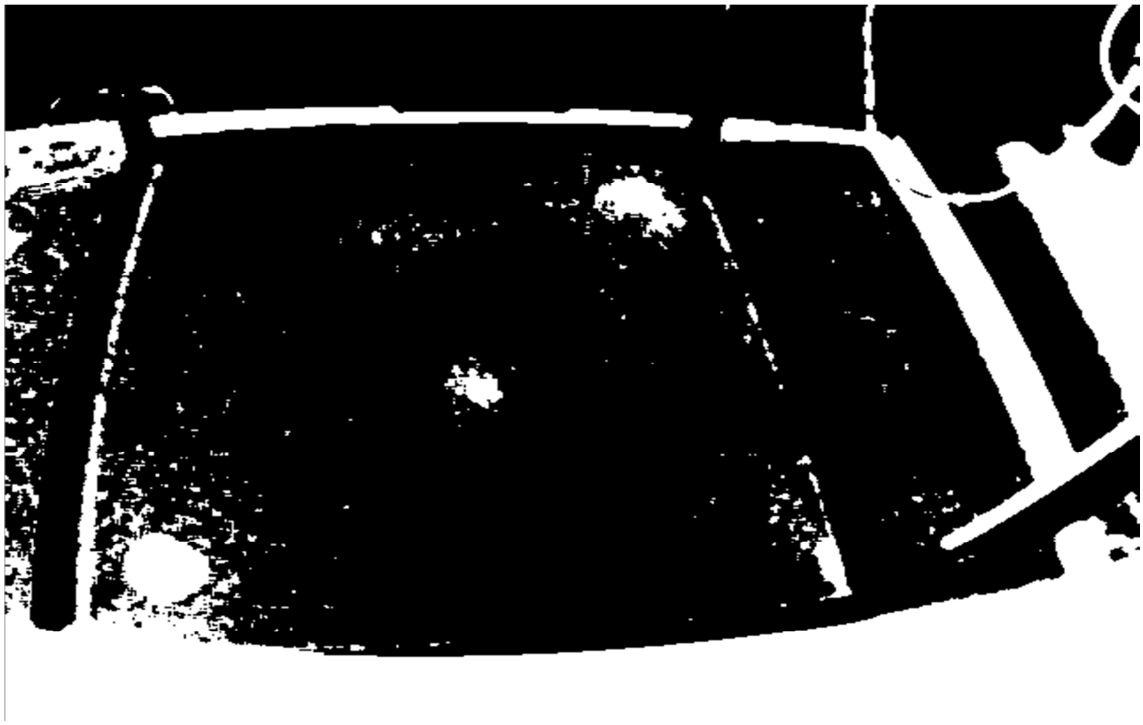

Figura 6. Imagen binaria con k=5, l=100.

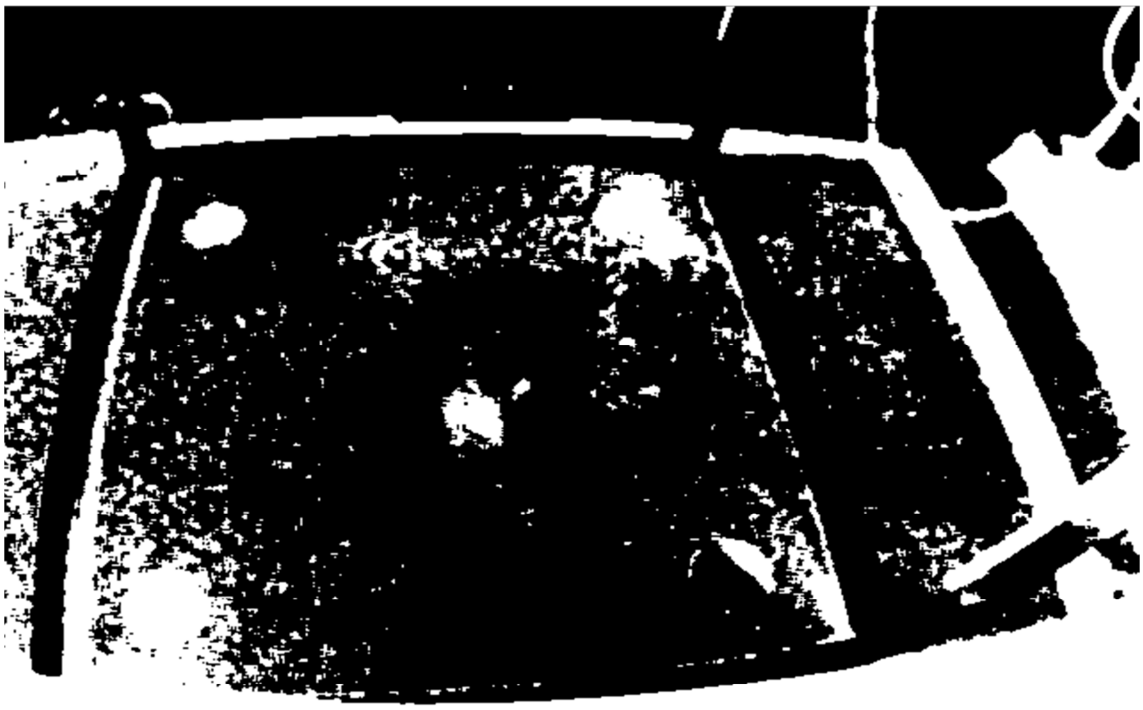

Figura 7. Imagen binaria con k=5, l=110.

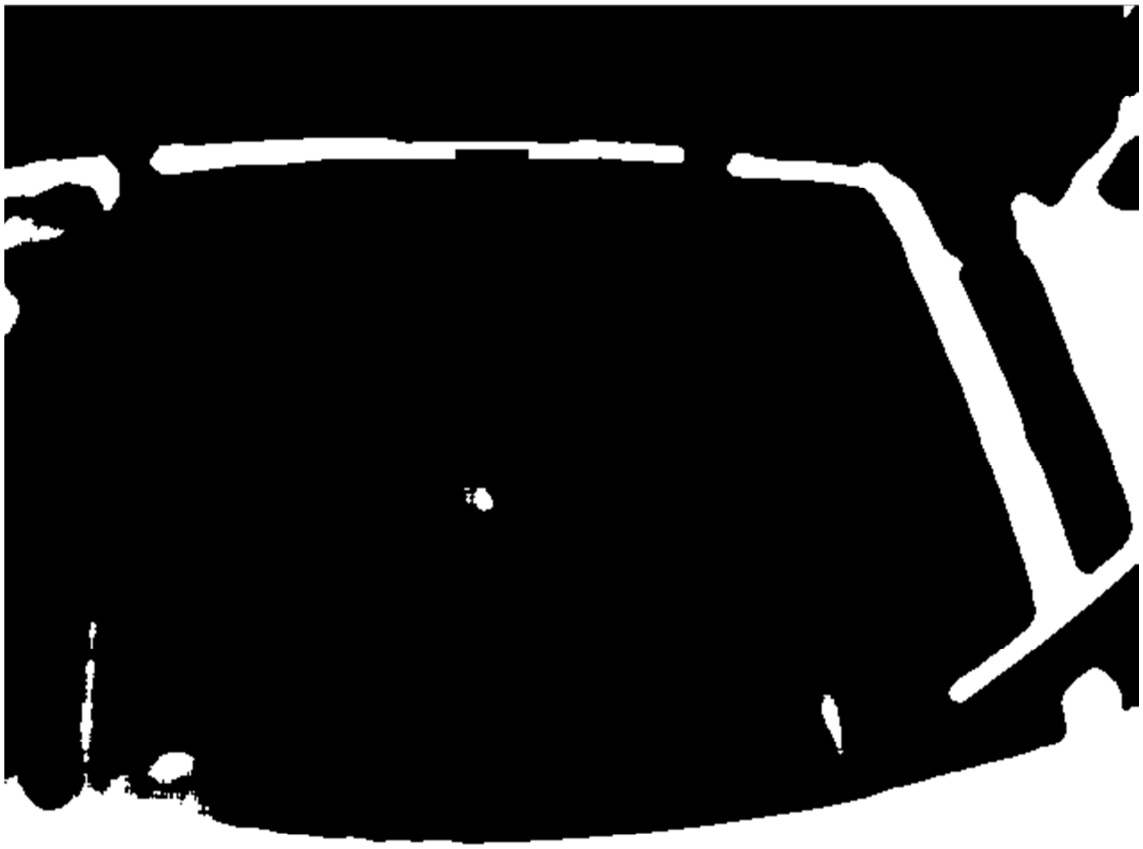

Figura 8. Imagen binaria con k=15, l=90.

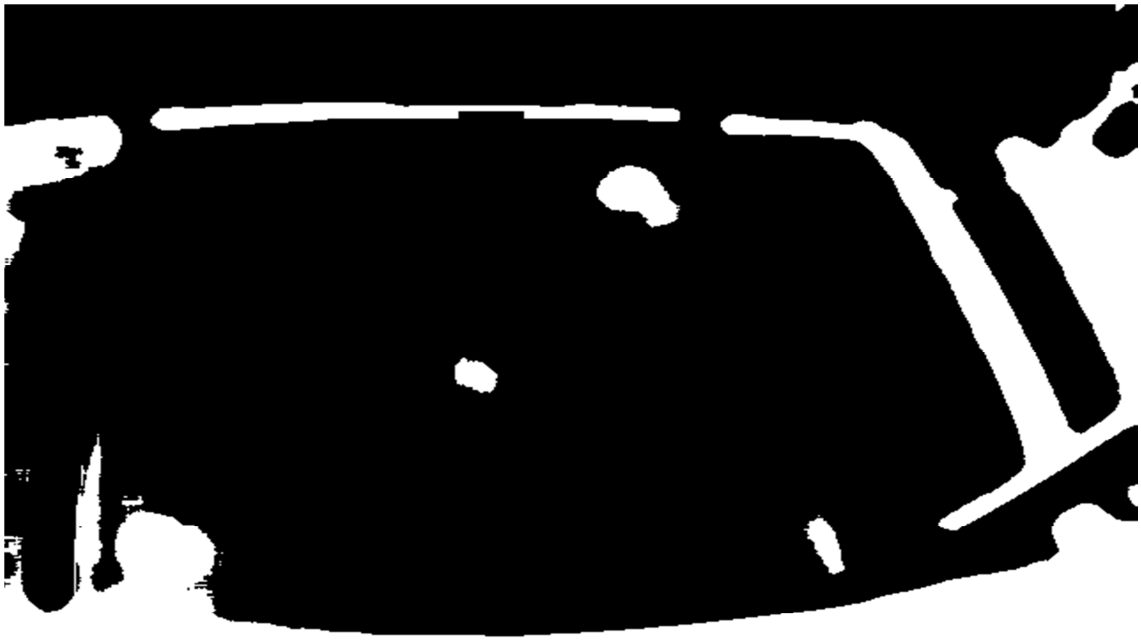

Figura 9. Imagen binaria con k=5, l=100.

Figura 10. Imagen binaria con k=15, l=110.

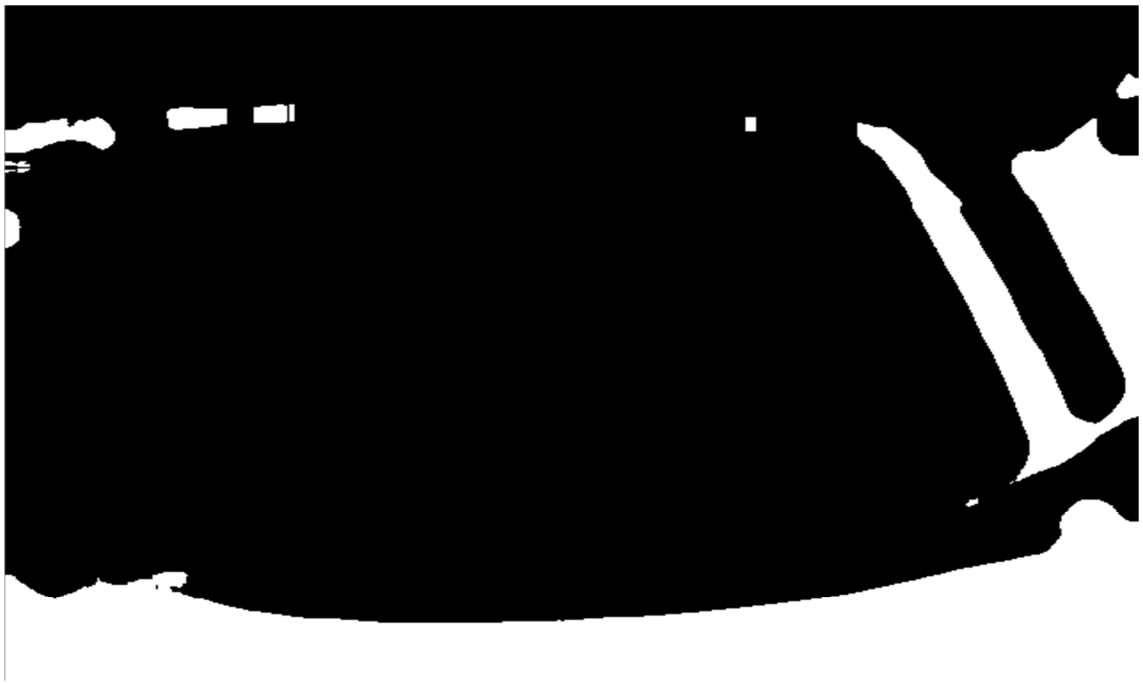

Figura 11. Imagen binaria con k=25, l=90.

Figura 12. Imagen binaria con k=25, l=100.

Figura 13. Imagen binaria con k=25, l=110.

Programa para la obtención de la imagen binaria por máscara de rojos

```
import numpy as np
import cv2
cap = cv2.VideoCapture(1)
while(True):

  ret, frame = cap.read()
  hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
  rojo_bajos1 = np.array([0,65,75], dtype=np.uint8)
  rojo_altos1 = np.array([12, 255, 255], dtype=np.uint8)
  rojo_bajos2 = np.array([240,65,75], dtype=np.uint8)
  rojo_altos2 = np.array([256, 255, 255], dtype=np.uint8)
  mascara_rojo1 = cv2.inRange(hsv, rojo_bajos1, rojo_altos1)
  mascara_rojo2 = cv2.inRange(hsv, rojo_bajos2, rojo_altos2)
  mask = cv2.add(mascara_rojo1, mascara_rojo2)
  cv2.imshow('imagen binaria máscara',mask)

  k = cv2.waitKey(30) & 0xFF
  if k==27:
      break

cap.release()
cv2.destroyAllWindows()
```

Programa para la obtención de la imagen anterior filtrada

```
import numpy as np
import cv2
cap = cv2.VideoCapture(1)
while(1):
    ret, frame = cap.read()
    gris = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    median = cv2.medianBlur(gris,5)
    ret,thresh1 = cv2.threshold(median,50,255,cv2.THRESH_BINARY_INV)
    kernel = np.ones((p,q),np.uint8)
    erosion = cv2.erode(thresh1,kernel,iterations = 1)
    dilatacion = cv2.dilate(erosion,kernel,iterations = 1)
    cv2.imshow('imagen tras filtrado de ruido',dilatación)
    if cv2.waitKey(1):
     break
cap.release()
cv2.destroyAllWindows()
```

Imégenes obtenidas por máscara de rojos y posterior filtrado

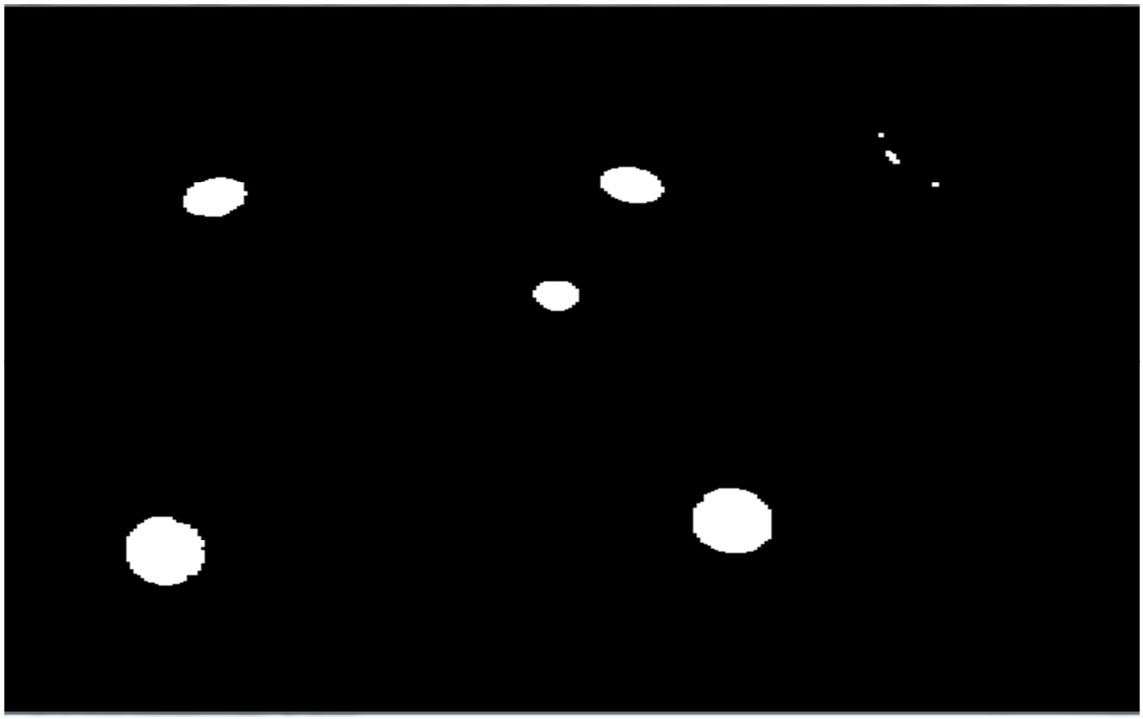

Figura 14. Imagen obtenida con p,q = 3,3.

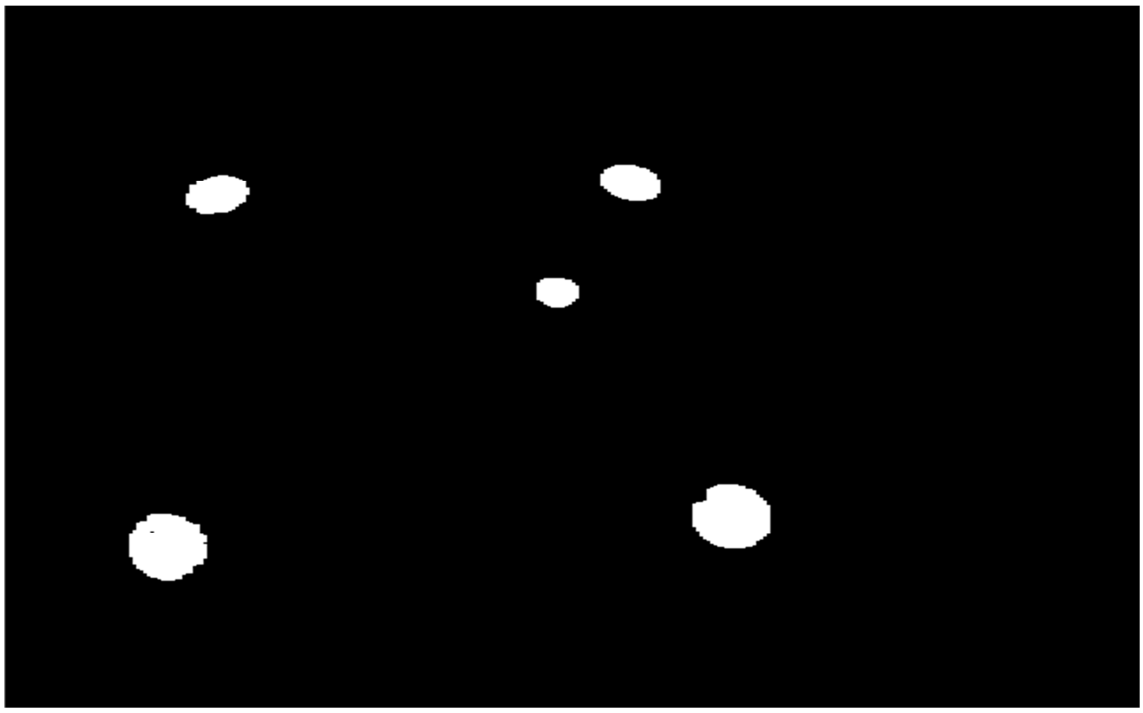

Figura 15. Imagen obtenida con p,q = 5,5.

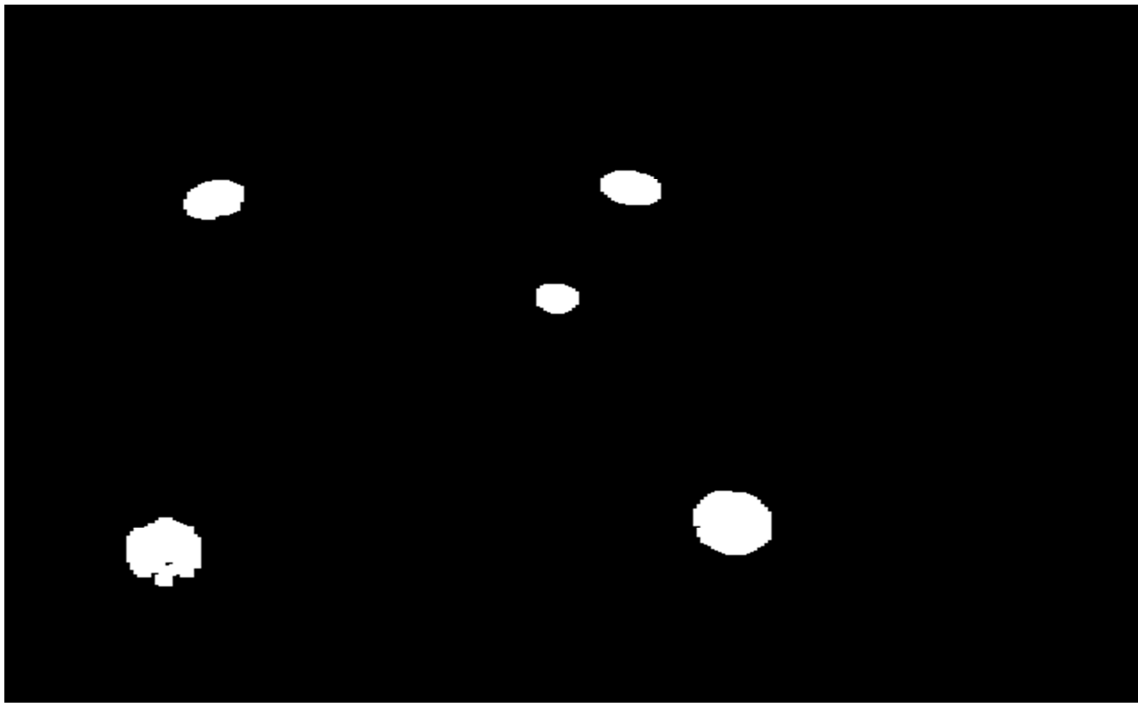

Figura 16. Imagen obtenida con p,q = 7,7.

Programa para la obtención de las coordenadas de los centros de las marcas de referencia

```
import numpy as np
import cv2
font = cv2.FONT_HERSHEY_SIMPLEX
cap = cv2.VideoCapture(1)
while(True):

  ret, frame = cap.read()
  hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
  rojo_bajos1 = np.array([0,65,75], dtype=np.uint8)
  rojo_altos1 = np.array([12, 255, 255], dtype=np.uint8)
  rojo_bajos2 = np.array([240,65,75], dtype=np.uint8)
  rojo_altos2 = np.array([256, 255, 255], dtype=np.uint8)
  mascara_rojo1 = cv2.inRange(hsv, rojo_bajos1, rojo_altos1)
  mascara_rojo2 = cv2.inRange(hsv, rojo_bajos2, rojo_altos2)
  mask = cv2.add(mascara_rojo1, mascara_rojo2)
  kernel = np.ones((11,11),np.uint8)
  erosion = cv2.erode(mask,kernel,iterations = 1)
  dilatacion = cv2.dilate(erosion,kernel,iterations = 1)
  cv2.imshow('imagen tras filtrado de ruido',dilatacion)
  contours,_ = cv2.findContours(dilatacion, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)
  cv2.drawContours(frame, contours, -1, (0,255,0), 2)
  a=0
  x=[0,0,0,0]
  y=[0,0,0,0]
  for i in contours:
    momentos = cv2.moments(i)
    area = cv2.contourArea(i)
    if area > 400:
     cx = int(momentos['m10']/momentos['m00'])
     cy = int(momentos['m01']/momentos['m00'])
     x[a] = cx
     y[a] = cy
     a=a+1
     cv2.circle(frame,(cx, cy), 3, (0,0,255), -1)
     cv2.putText(frame,"(x: " + str(cx) + ", y: " + str(cy) +
")",(cx+10,cy+10), font, 0.5,(255,255,255),1)
  print (x)
  print (y)
  x1=sorted(x)
  y1=sorted(y)
  print(x1)
  print(y1)
  cv2.imshow('Final', frame)
  k = cv2.waitKey(30) & 0xFF
  if k==27:
      break

cap.release()
cv2.destroyAllWindows()
```

Programa para la obtención de la imagen en planta rotada

```
import numpy as np
import cv2
font = cv2.FONT_HERSHEY_SIMPLEX
cap = cv2.VideoCapture(1)
while(True):

  ret, frame = cap.read()
  hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
  rojo_bajos1 = np.array([0,65,75], dtype=np.uint8)
  rojo_altos1 = np.array([12, 255, 255], dtype=np.uint8)
  rojo_bajos2 = np.array([240,65,75], dtype=np.uint8)
  rojo_altos2 = np.array([256, 255, 255], dtype=np.uint8)
  mascara_rojo1 = cv2.inRange(hsv, rojo_bajos1, rojo_altos1)
  mascara_rojo2 = cv2.inRange(hsv, rojo_bajos2, rojo_altos2)
  mask = cv2.add(mascara_rojo1, mascara_rojo2)
  kernel = np.ones((11,11),np.uint8)
  erosion = cv2.erode(mask,kernel,iterations = 1)
  dilatacion = cv2.dilate(erosion,kernel,iterations = 1)
  contours,_ = cv2.findContours(dilatacion, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)
  a=0
  x=[0,0,0,0]
  y=[0,0,0,0]
  for i in contours:
    momentos = cv2.moments(i)
    area = cv2.contourArea(i)
    if area > 400:
     cx = int(momentos['m10']/momentos['m00'])
     cy = int(momentos['m01']/momentos['m00'])
     x[a] = cx
     y[a] = cy
     a=a+1
  print (x)
  print (y)
  x1=sorted(x)
  y1=sorted(y)
  print(x1)
  print(y1)
rows,cols,ch = dilatacion.shape
pts1 = np.float32([[x2[1],y2[1]], [x2[4],y2[2]], [x2[2],y2[3]],
[x2[3],y2[4]]])
pts2 = np.float32([[0.0,0.0],[0.920,0.0],[0,0.920],[0.920,0.920]]
M = cv2.getPerspectiveTransform(pts1,pts2)
imgplanta = cv2.warpPerspective(dilatacion,M,(920,920))
rows,cols = imgplanta.shape
N = cv2.getRotationMatrix2D((cols/2,rows/2),180,1)
rotada = cv2.warpAffine(imgplanta,N,(cols,rows))
cv2.imshow('imagen rotada’,rotada)
k = cv2.waitKey(30) & 0xFF
if k==27:
      break

cap.release()
cv2.destroyAllWindows()
```

Programa para la obtención de las coordenadas del centro de la carga

```
import numpy as np
import cv2
font = cv2.FONT_HERSHEY_SIMPLEX
ox=0
oy=0
cap = cv2.VideoCapture(1)
while(True):

  ret, frame = cap.read()
  hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
  rojo_bajos1 = np.array([0,65,75], dtype=np.uint8)
  rojo_altos1 = np.array([12, 255, 255], dtype=np.uint8)
  rojo_bajos2 = np.array([240,65,75], dtype=np.uint8)
  rojo_altos2 = np.array([256, 255, 255], dtype=np.uint8)
  mascara_rojo1 = cv2.inRange(hsv, rojo_bajos1, rojo_altos1)
  mascara_rojo2 = cv2.inRange(hsv, rojo_bajos2, rojo_altos2)
  mask = cv2.add(mascara_rojo1, mascara_rojo2)
  kernel = np.ones((7,7),np.uint8)
  erosion = cv2.erode(mask,kernel,iterations = 1)
  dilatacion = cv2.dilate(erosion,kernel,iterations = 1)
  cv2.imshow('imagen tras filtrado de ruido',dilatacion)
  contours,_ = cv2.findContours(dilatacion, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)
  cv2.drawContours(frame, contours, -1, (0,255,0), 2)
  a=0
  x=[0,0,0,0]
  y=[0,0,0,0]
  for i in contours:
    momentos = cv2.moments(i)
    area = cv2.contourArea(i)
    if area > 400:
     cx = int(momentos['m10']/momentos['m00'])
     cy = int(momentos['m01']/momentos['m00'])
     x[a] = cx
     y[a] = cy
     a=a+1

  print (x)
  print (y)
  x2=sorted(x)
  y2=sorted(y)
  print(x2)
  print(y2)
  rows,cols,ch = frame.shape
  pts1 = np.float32([[x2[0],y2[1]], [x2[2],y2[0]], [x2[1],y2[3]],
[x2[3],y2[2]]])
  pts2 = np.float32([[0,0],[920,0],[0,920],[920,920]])
  M = cv2.getPerspectiveTransform(pts1,pts2)
  imgplanta = cv2.warpPerspective(dilatacion,M,(920,920))
  cv2.imshow('imagen en planta',imgplanta)
  rows,cols = imgplanta.shape
  N = cv2.getRotationMatrix2D((cols/2,rows/2),180,1)
  rotada = cv2.warpAffine(imgplanta,N,(cols,rows))
  contours2,_ = cv2.findContours(rotada, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)
```

```
  for i in contours2:
    momentos = cv2.moments(i)
    area = cv2.contourArea(i)
    if area > 5000:
     ox = int(momentos['m10']/momentos['m00'])
     oy = int(momentos['m01']/momentos['m00'])
    print(ox)
    print(oy)
  k = cv2.waitKey(30) & 0xFF
  if k==27:
      break

cap.release()
cv2.destroyAllWindows()
```

Anexo 2: Características del puente grúa y de sus componentes principales.

Características del puente grúa

En la tabla 1 se muestran los principales parámetros del puente grúa didáctico fabricado por INTECO.

Parámetro	**Valor**
Velocidad máxima v_{max} $[m/s]$	0,3
Aceleración máxima a_{max} $[m/s2]$	0,6
Longitud del espacio de trabajo en el eje X $[mm]$	920
Longitud del espacio de trabajo en el eje Y $[mm]$	920
Longitud de la cuerda l $[mm]$	820
Masa de la carga m_p $[kg]$	1
Masa del carro m_c $[kg]$	1,16
Masa del carril m_r $[kg]$	2,2

Tabla 1. Parámetros principales del puente grúa de INTECO.

Características de los motores RH158.24.75 DC

En la figura 17 se muestran las dimensiones del motor RH158.24.75 DC acoplado en cada eje del puente grúa. En la tabla 2 se pueden observar las características principales del motor mencionado.

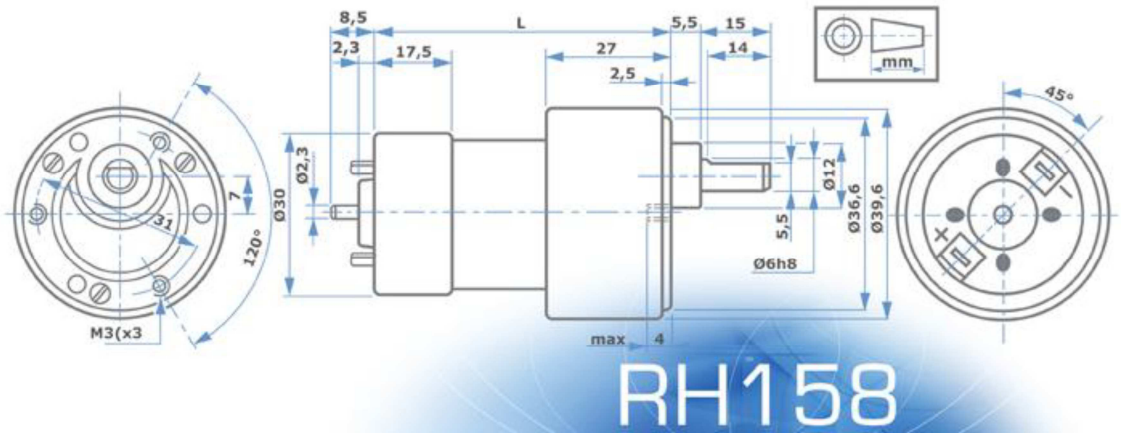

Figura 17. Imagen obtenida con p,q = 7,7.

Parámetro (A temperatura ambiente +20% - tolerancia +/-10%)	**Valor**
Tensión nominal [V]	24
L [mm]	66,5
Ratio de :1	76,84
Par nominal [Ncm]	50
Velocidad [m/s] (Sin carga)	81
Velocidad en funcionamiento nominal	55
Intensidad [mA] (Sin carga)	<70
Intensidad en funcionamiento nominal	340
Máxima fuerza en el eje radial [N]	50
Máxima fuerza en el eje axial [N]	10
Rango de temperatura [$°C$]	-20 / 60
Masa aproximada [g]	190

Tabla 2. Parámetros del motor RH158.24.75 DC .

Características del *encoder*

Estos interruptores de efecto Hall son sensores altamente estables a la temperatura y resistentes al estrés que se utilizan mejor en aplicaciones que proporcionan pendientes magnéticas pronunciadas y bajos niveles residuales de densidad de flujo magnético. Cada dispositivo incluye un regulador de voltaje, un generador de voltaje Hall cuadrático, un circuito de estabilidad de temperatura, un amplificador estabilizado por chopper de señal, un disparador Schmitt y un Mosfet de drenaje abierto en un solo chip de silicio.

El regulador a bordo permite la operación con voltajes de alimentación de 3,5 a 24V. El *mosfet* de salida puede hundirse hasta 20 mA con un *pull-up* de salida adecuado, se pueden usar directamente con circuitos lógicos bipolares o MOS. En la tabla 3 se muestran los parámetros del *encoder.*

Parámetro	**Valor**
Tensión suministrada [V]	28
Intensidad suministrada [mA]	50
Tensión de salida [V]	28
Intensidad de salida [mA]	50
Rango de temperatura de almacenamiento [$°C$]	-50 a 150
Máxima temperatura de unión [$°C$]	165

Tabla 3. Parámetros del *encoder* .

Anexo 3: Sistemas de control demo

En este anexo se muestran los distintos controladores que incluye el fabricante INTECO en las figuras 18 – 22.

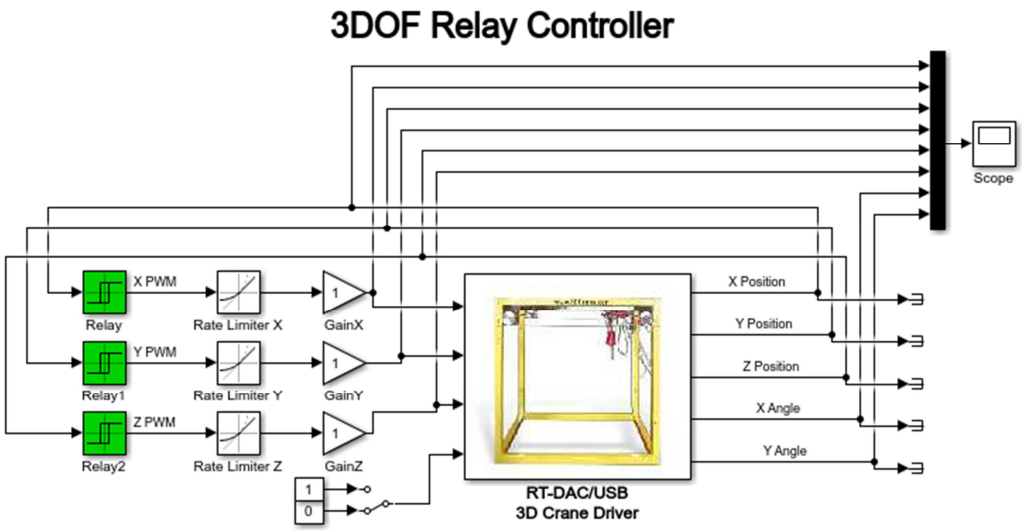

Figura 18. Controlador Relay.

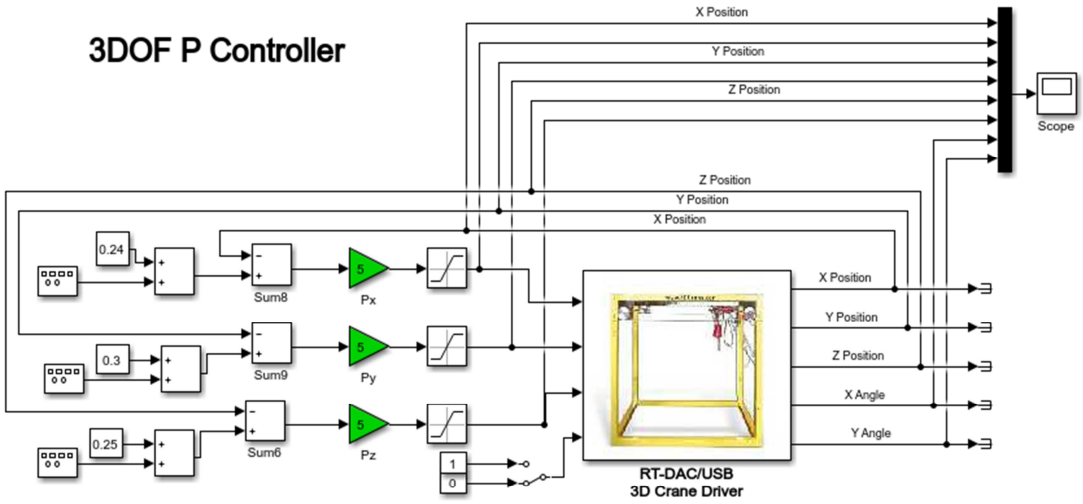

Figura 19. Controlador P XYZ.

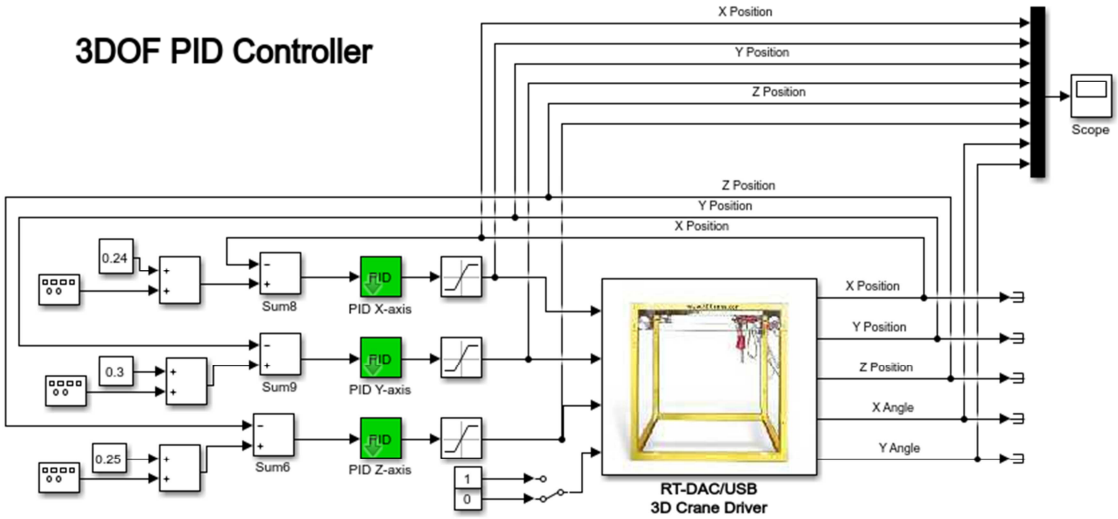

Figura 20. Controlador PID XYZ.

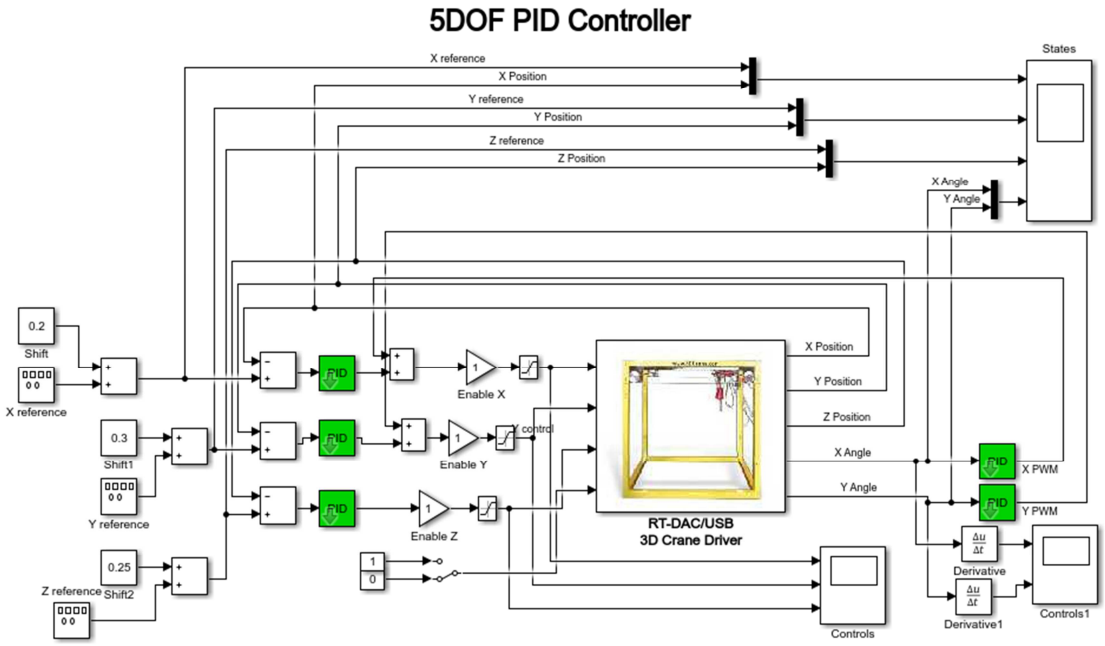

Figura 21. Controlador PID Angles.

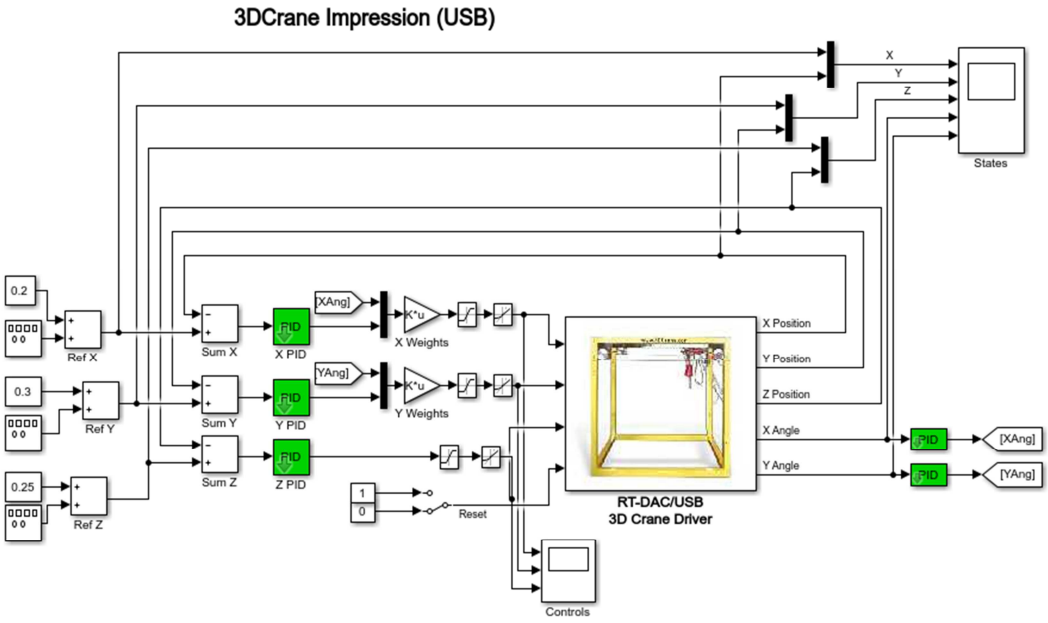

Figura 22. Controlador performance.

Anexo 4: Prueba de parámetros en los controladores del sistema de control

En este anexo se muestran en las tablas 4-15 y en las figuras 23-45 los valores de algunos de los parámetros que se han introducido en los controladores de nuestro sistema de control y los resultados obtenidos con cada combinación de parámetros.

	Proporcional	Derivativa	Integral	Figura
Eje x	3	0	0	
Eje y	3	0	0	
Ánguloα	0	0	0	
Ángulo β	0	0	0	

Tabla 4. Parámetros utilizados en la prueba 1 del ajuste de controladores PID.

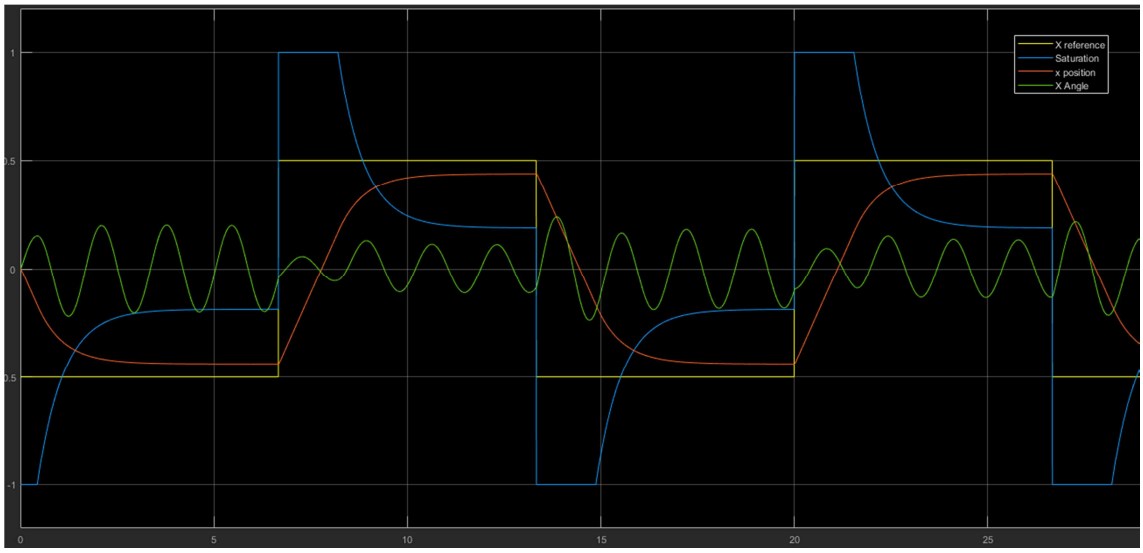

Figura 23. Resultados con los parámetros de la tabla 1 en el eje X.

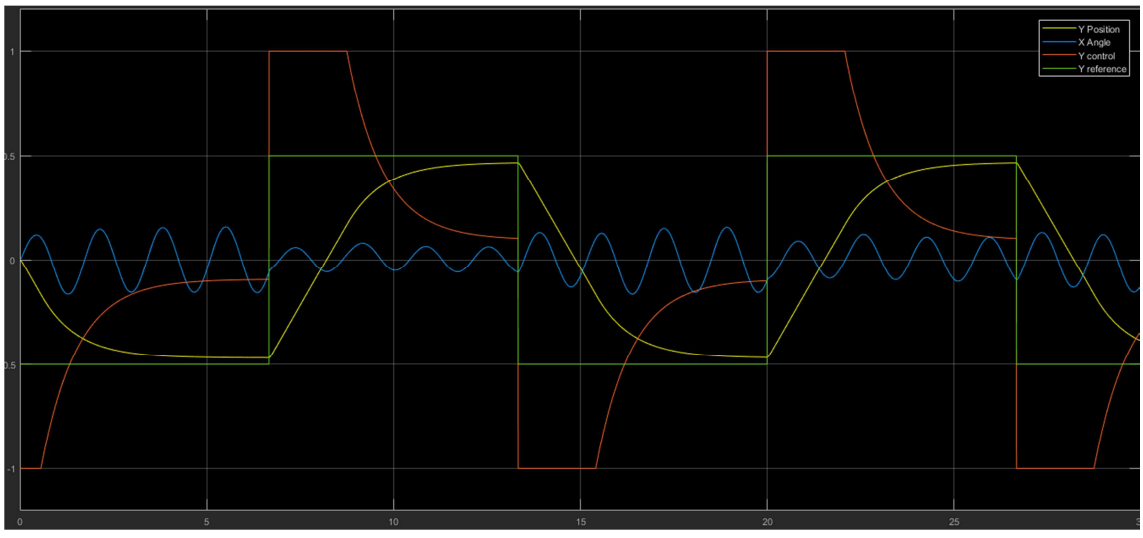

Figura 24. Resultados con los parámetros de la tabla 1 en el eje Y.

	Proporcional	Derivativa	Integral	Figura
Eje x	3	0	0	
Eje y	3	0	0	
Ánguloα	3	0	0	
Ángulo β	3	0	0	

Tabla 5. Parámetros utilizados en la prueba 2 del ajuste de controladores PID.

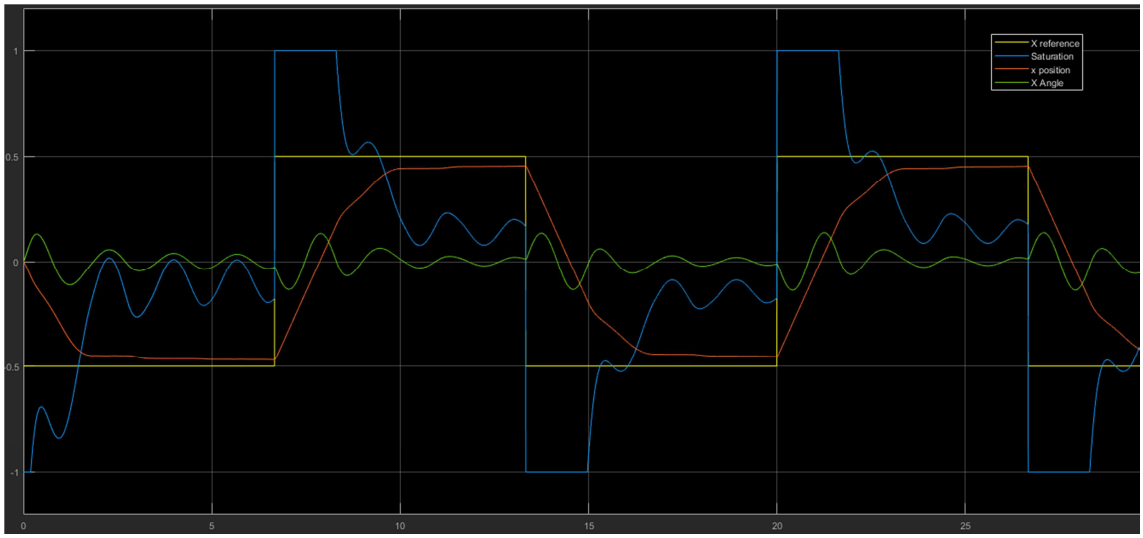

Figura 25. Resultados con los parámetros de la tabla 2 en el eje X.

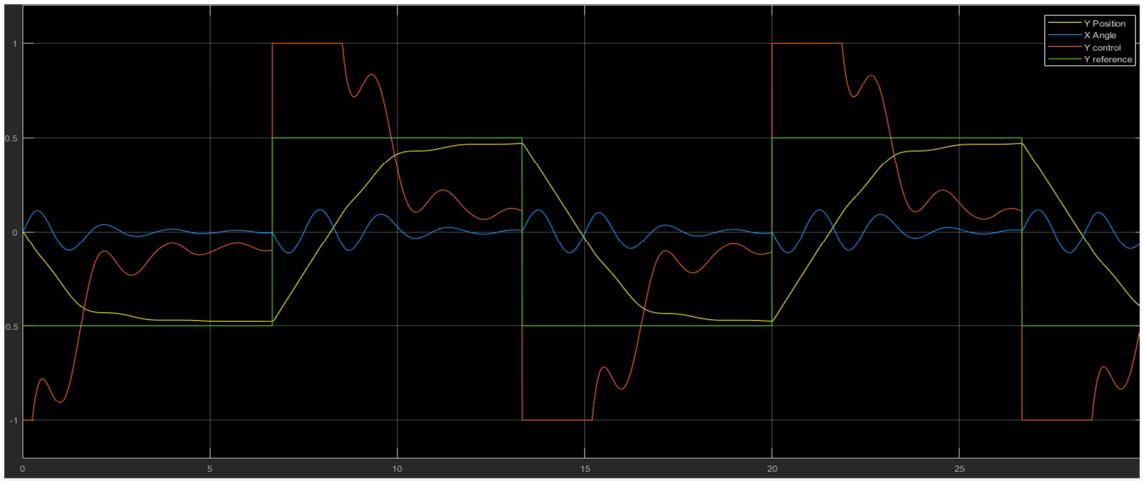

Figura 26. Resultados con los parámetros de la tabla 2 en el eje Y.

	Proporcional	Derivativa	Integral	Figura
Eje x	3	0	0	
Eje y	3	0	0	
Ánguloα	5	0	0	
Ángulo β	5	0	0	

Tabla 6. Parámetros utilizados en la prueba 3 del ajuste de controladores PID.

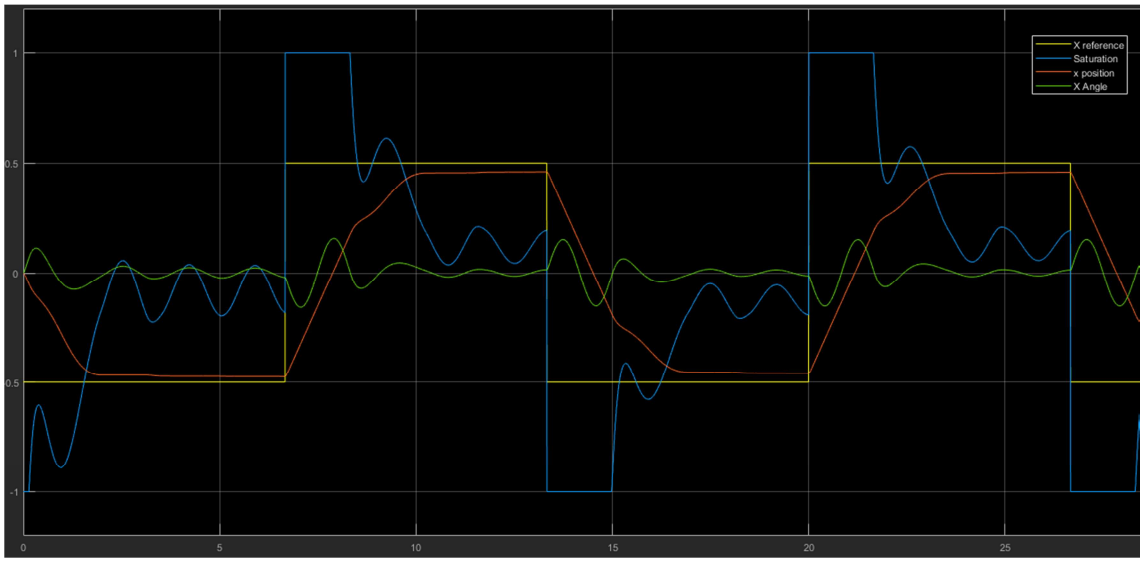

Figura 27. Resultados con los parámetros de la tabla 3 en el eje X.

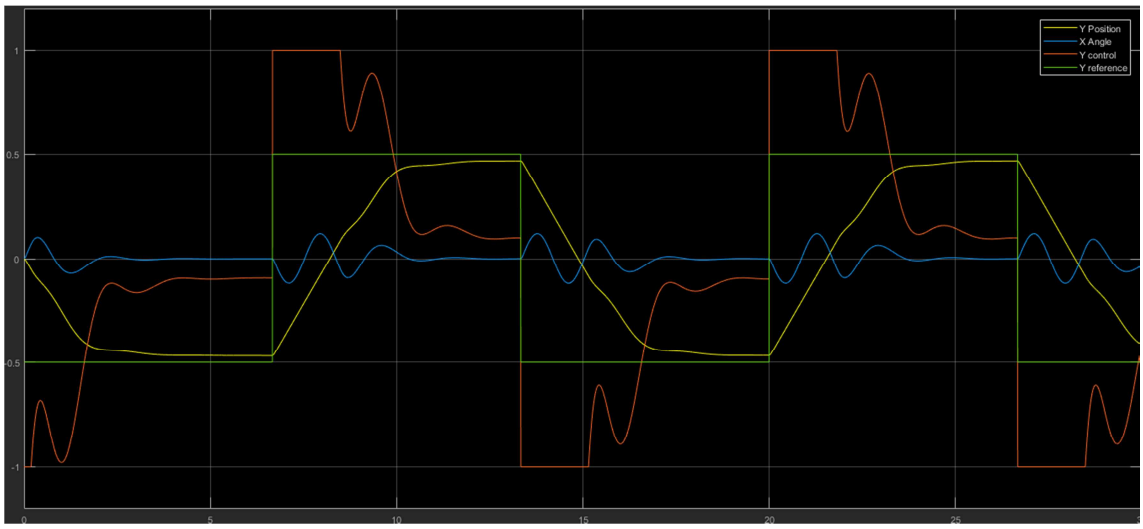

Figura 28. Resultados con los parámetros de la tabla 3 en el eje Y.

	Proporcional	Derivativa	Integral	Figura
Eje x	5	0	0	
Eje y	5	0	0	
Ánguloα	0	0	0	
Ángulo β	0	0	0	

Tabla 7. Parámetros utilizados en la prueba 4 del ajuste de controladores PID.

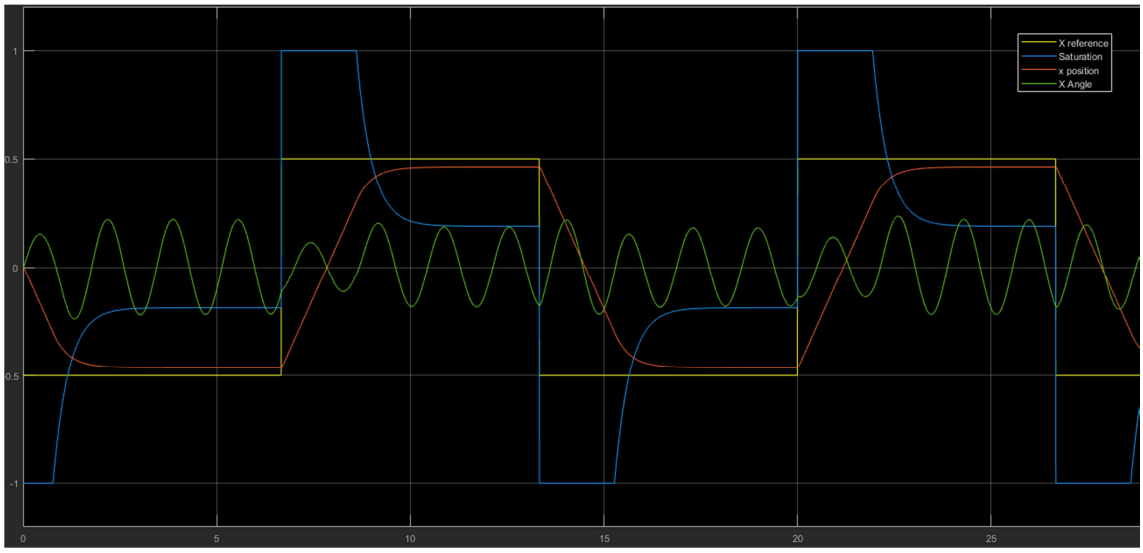

Figura 29. Resultados con los parámetros de la tabla 4 en el eje X.

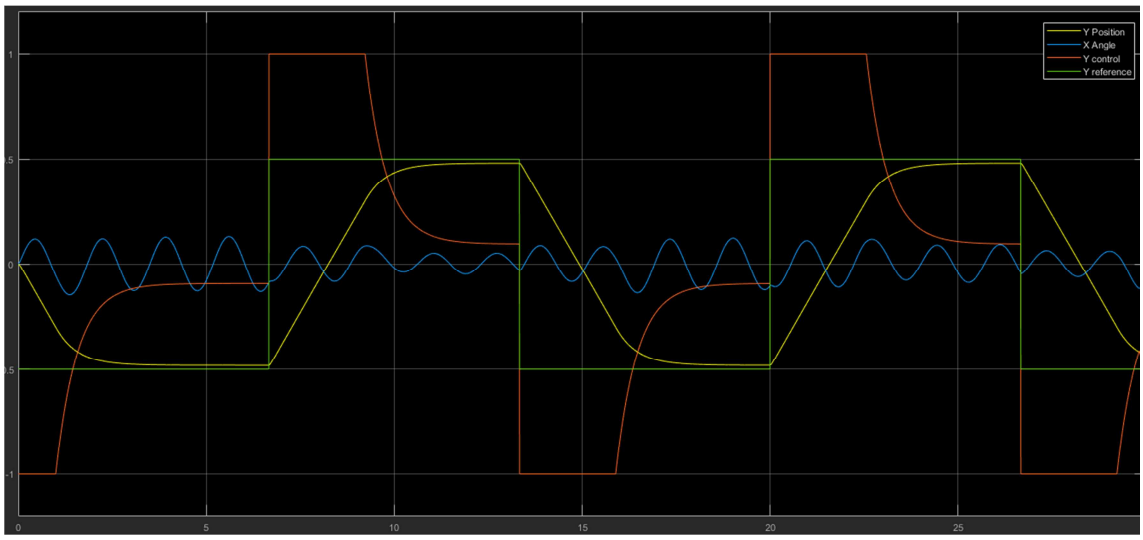

Figura 30. Resultados con los parámetros de la tabla 4 en el eje Y.

	Proporcional	Derivativa	Integral	Figura
Eje x	5	0	0	
Eje y	5	0	0	
Ánguloα	5	0	0	
Ángulo β	5	0	0	

Tabla 8. Parámetros utilizados en la prueba 5 del ajuste de controladores PID.

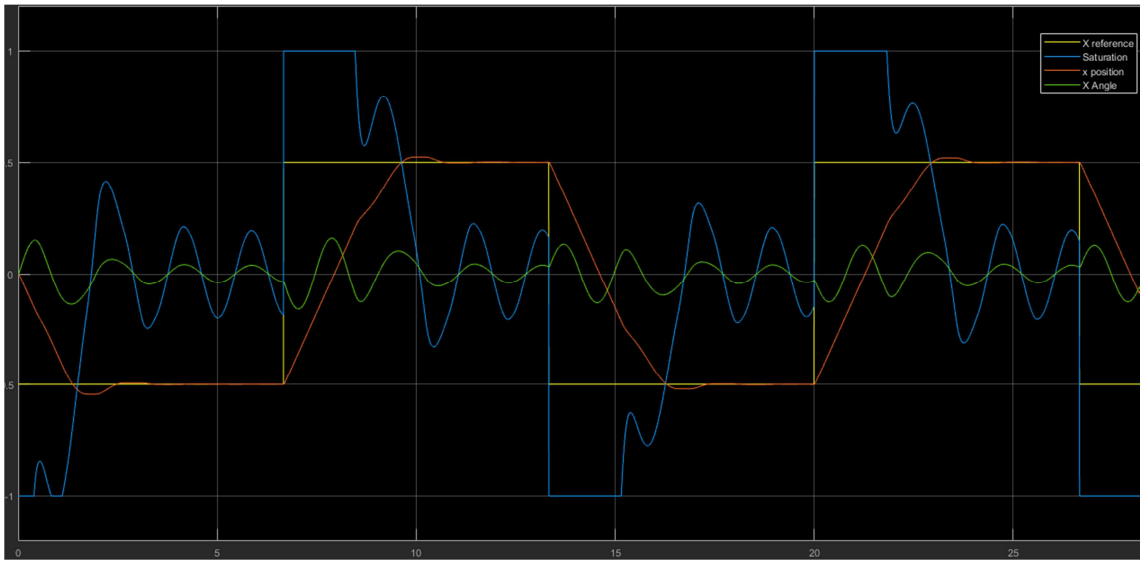

Figura 31. Resultados con los parámetros de la tabla 5 en el eje X.

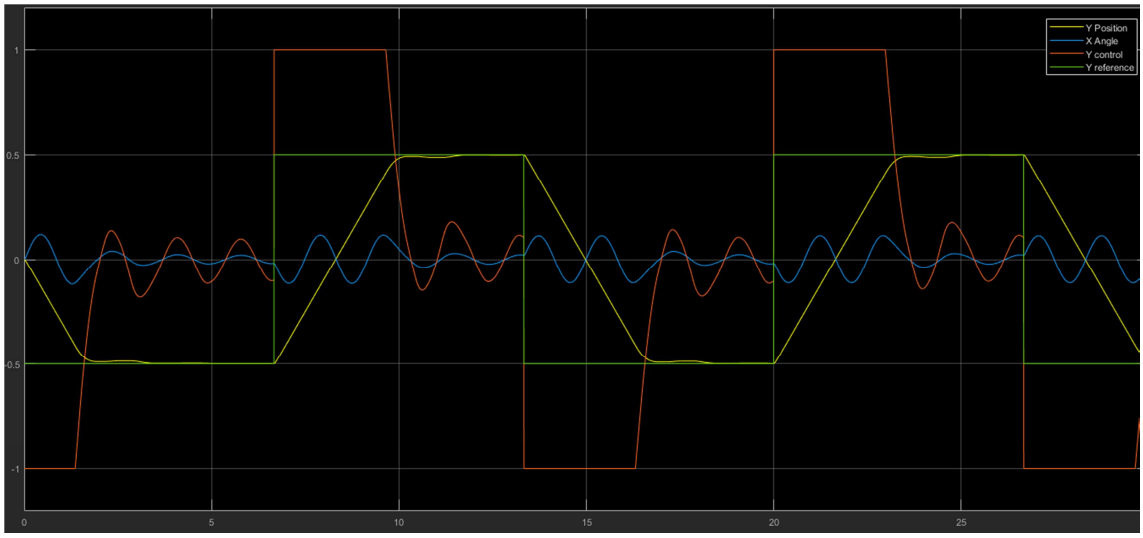

Figura 32. Resultados con los parámetros de la tabla 5 en el eje Y.

	Proporcional	Derivativa	Integral	Figura
Eje x	5	0	0	
Eje y	5	0	0	
Ánguloα	10	0	0	
Ángulo β	10	0	0	

Tabla 9. Parámetros utilizados en la prueba 6 del ajuste de controladores PID.

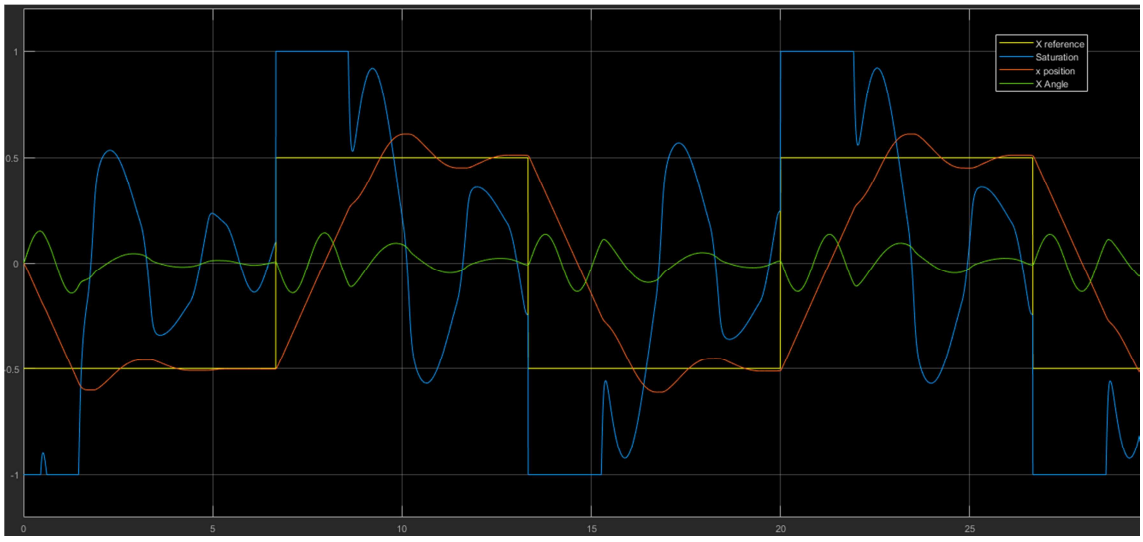

Figura 33. Resultados con los parámetros de la tabla 6 en el eje X.

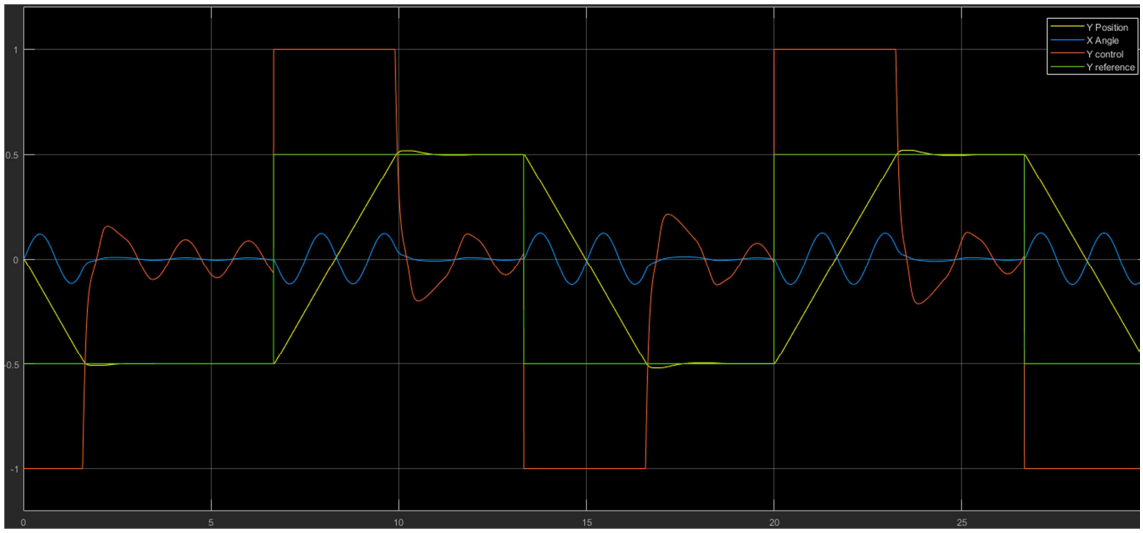

Figura 34. Resultados con los parámetros de la tabla 6 en el eje Y.

	Proporcional	Derivativa	Integral	Figura
Eje x	10	0	0	
Eje y	10	0	0	
Ánguloα	0	0	0	
Ángulo β	0	0	0	

Tabla 10. Parámetros utilizados en la prueba 7 del ajuste de controladores PID.

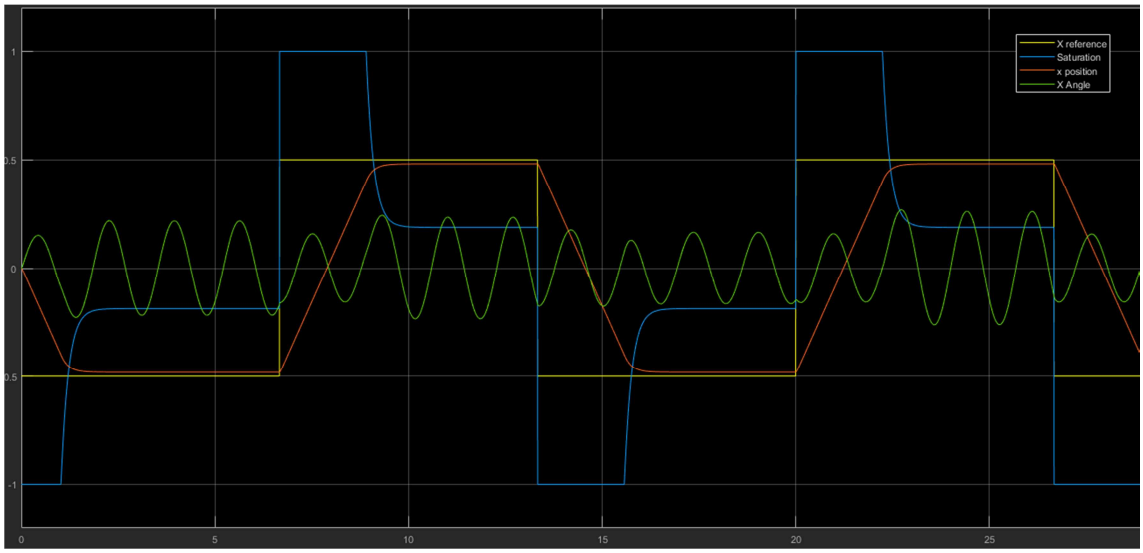

Figura 35. Resultados con los parámetros de la tabla 7 en el eje X.

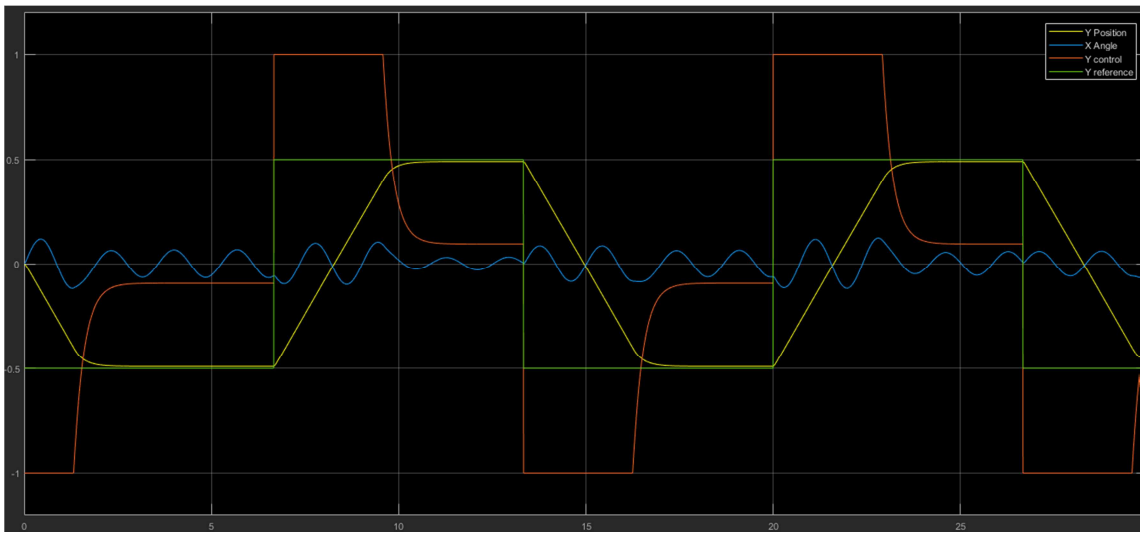

Figura 36. Resultados con los parámetros de la tabla 7 en el eje Y.

	Proporcional	Derivativa	Integral	Figura
Eje x	10	0	0	
Eje y	10	0	0	
Ánguloα	10	0	0	
Ángulo β	10	0	0	

Tabla 11. Parámetros utilizados en la prueba 8 del ajuste de controladores PID.

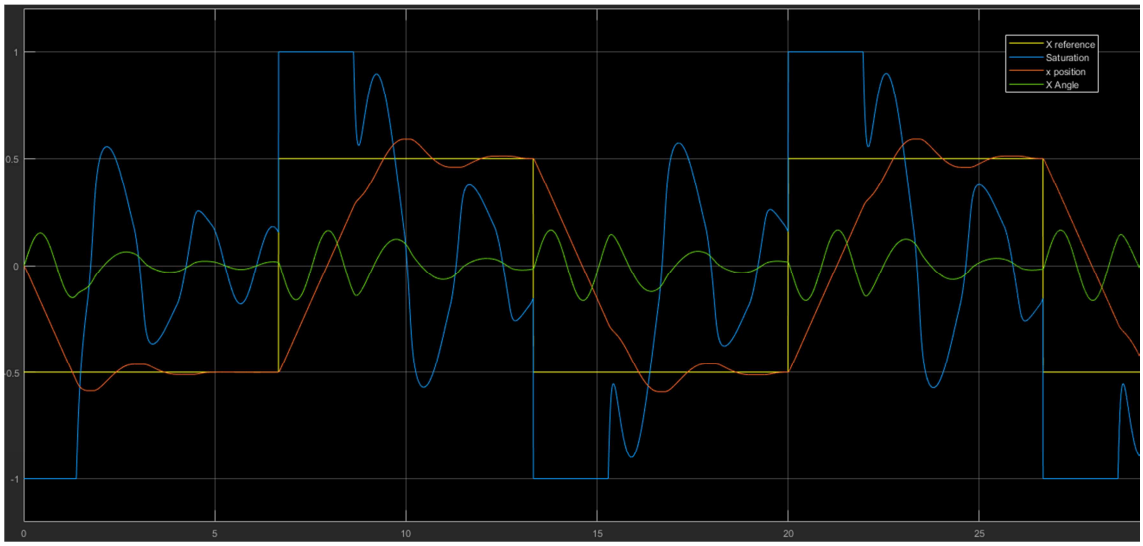

Figura 37. Resultados con los parámetros de la tabla 8 en el eje X.

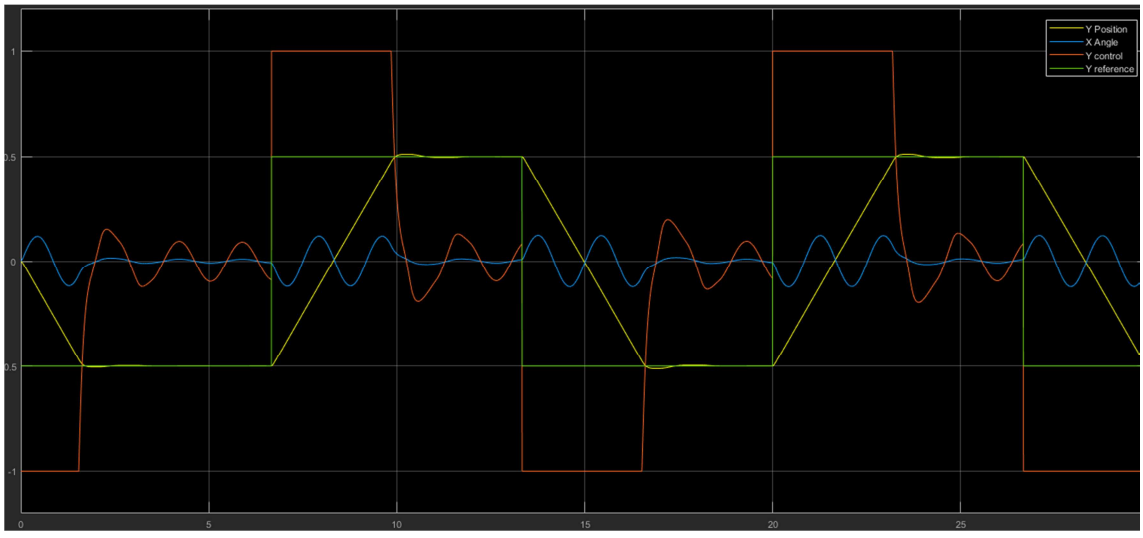

Figura 38. Resultados con los parámetros de la tabla 8 en el eje Y.

	Proporcional	Derivativa	Integral	Figura
Eje x	10	0	0	
Eje y	10	0	0	
Ánguloα	15	0	0	
Ángulo β	15	0	0	

Tabla 12. Parámetros utilizados en la prueba 9 del ajuste de controladores PID.

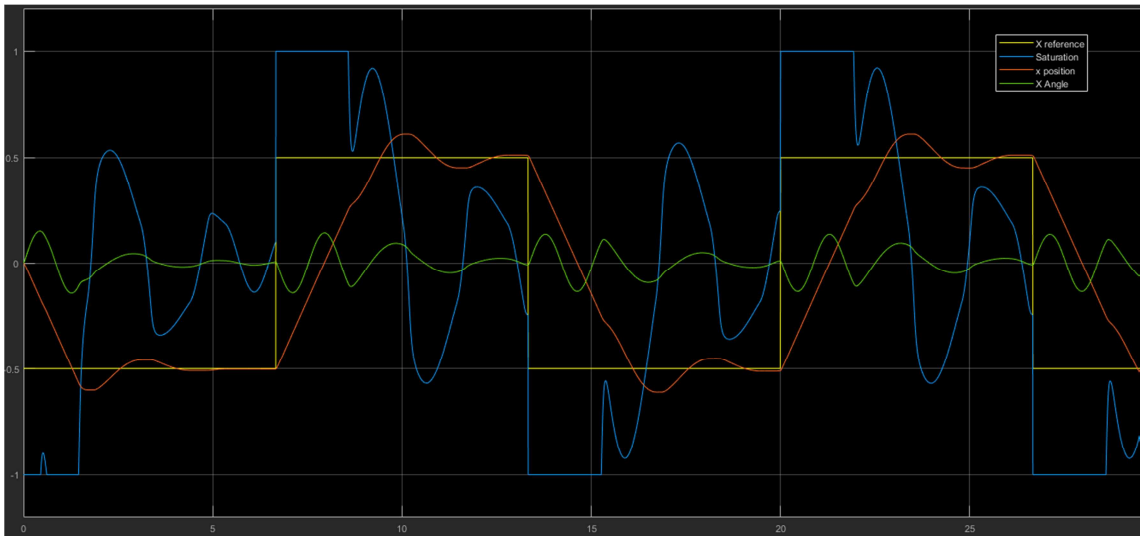

Figura 39. Resultados con los parámetros de la tabla 9 en el eje X.

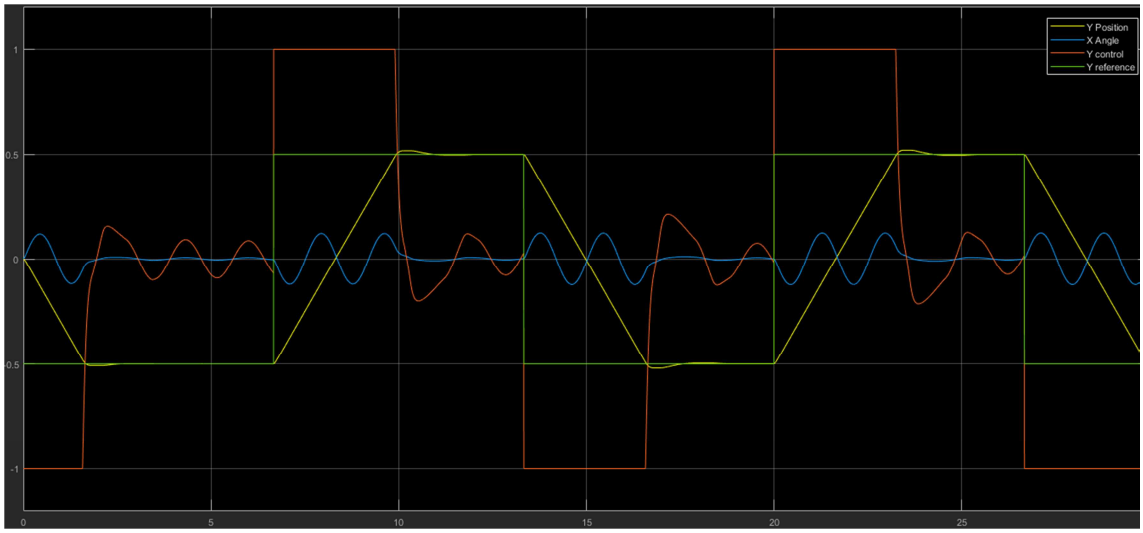

Figura 40. Resultados con los parámetros de la tabla 9 en el eje Y.

	Proporcional	Derivativa	Integral	Figura
Eje x	30	0	0	
Eje y	30	0	0	
Ánguloα	0	0	0	
Ángulo β	0	0	0	

Tabla 13. Parámetros utilizados en la prueba 10 del ajuste de controladores PID.

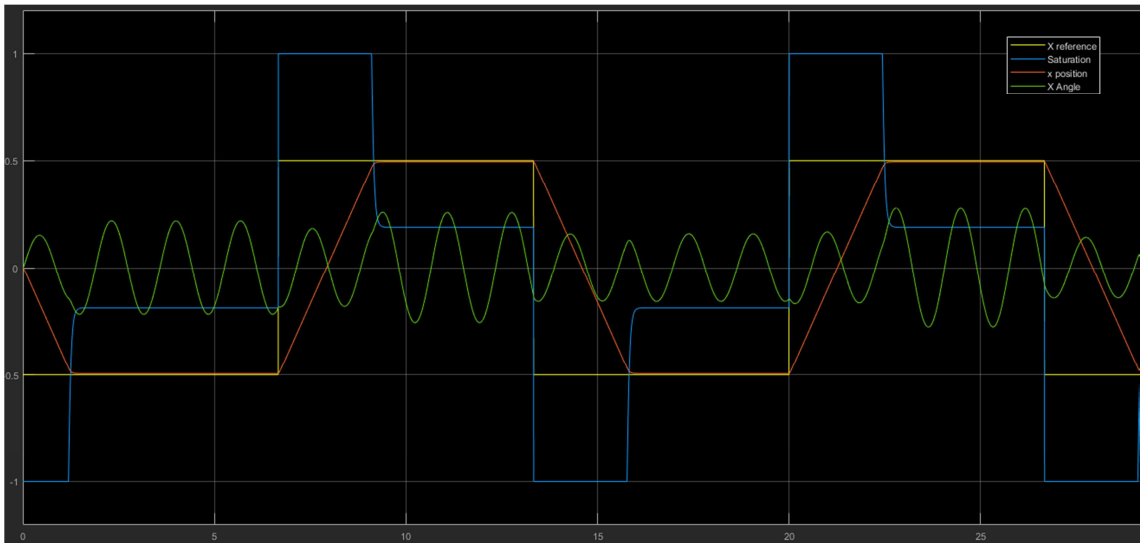

Figura 40. Resultados con los parámetros de la tabla 10 en el eje X.

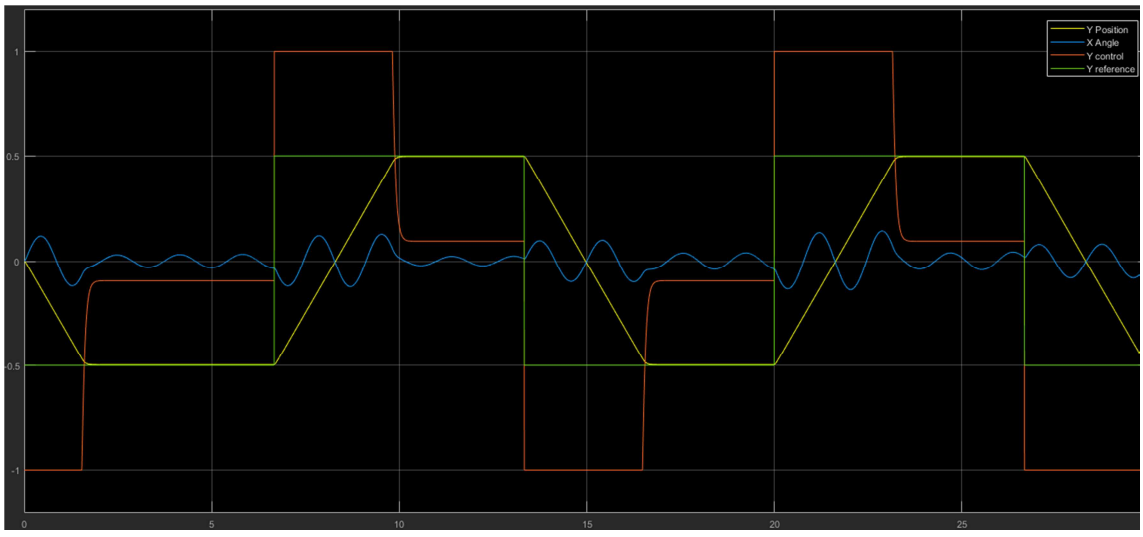

Figura 41. Resultados con los parámetros de la tabla 10 en el eje Y.

	Proporcional	Derivativa	Integral	Figura
Eje x	30	0	0	
Eje y	30	0	0	
Ánguloα	30	0	0	
Ángulo β	30	0	0	

Tabla 14. Parámetros utilizados en la prueba 11 del ajuste de controladores PID.

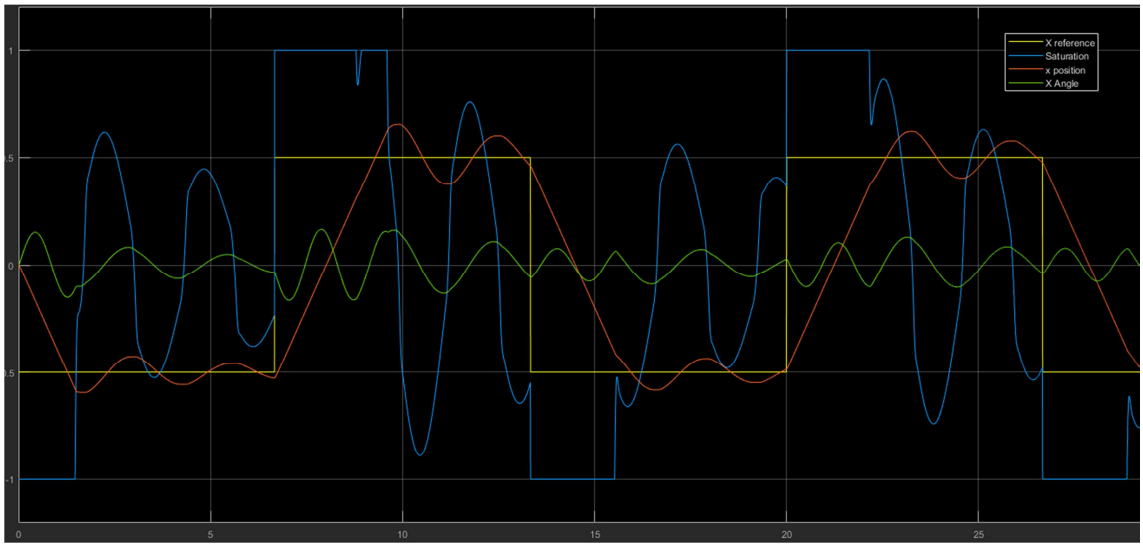

Figura 42. Resultados con los parámetros de la tabla 11 en el eje X.

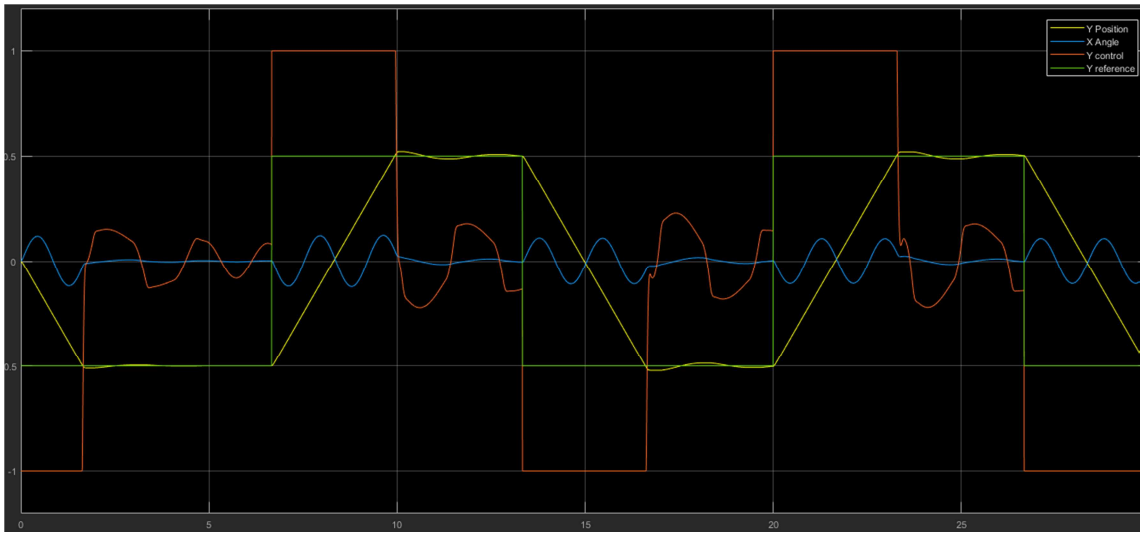

Figura 43. Resultados con los parámetros de la tabla 11 en el eje Y.

	Proporcional	Derivativa	Integral	Figura
Eje x	30	0	0	
Eje y	30	0	0	
Ánguloα	50	0	0	
Ángulo β	50	0	0	

Tabla 15. Parámetros utilizados en la prueba 12 del ajuste de controladores PID.

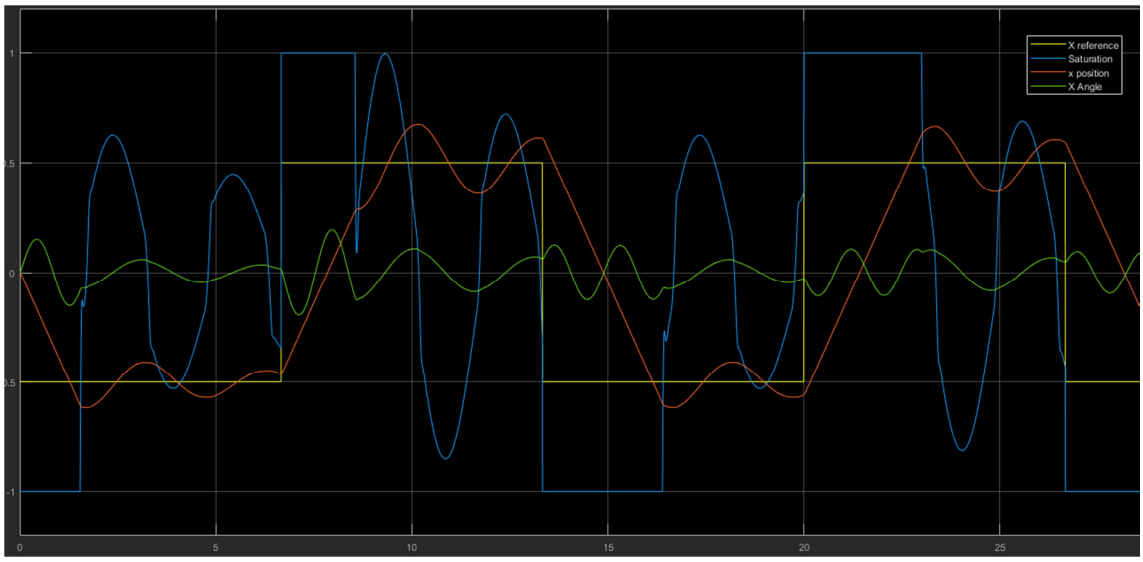

Figura 44. Resultados con los parámetros de la tabla 12 en el eje X.

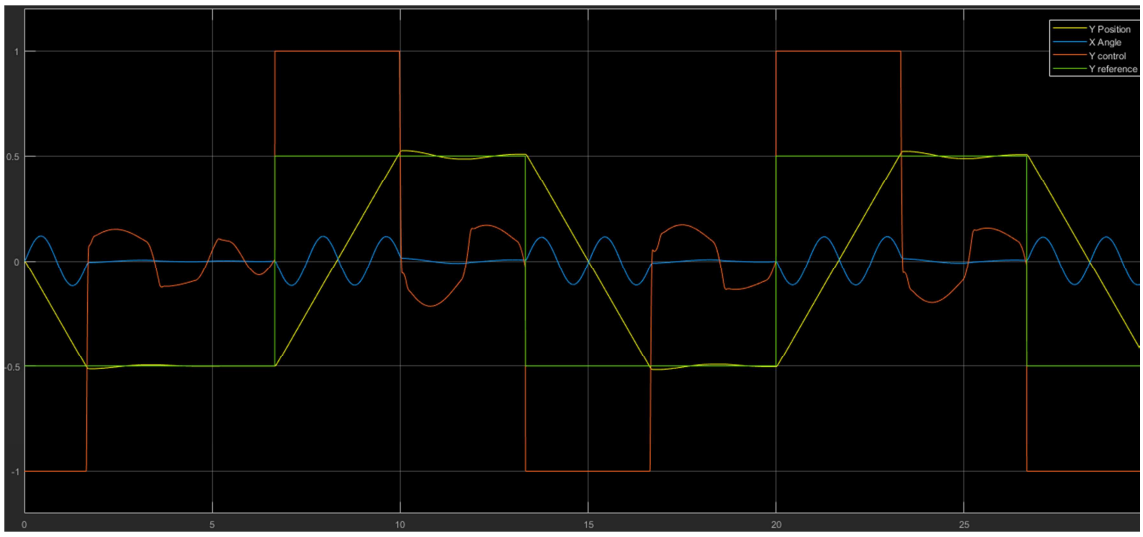

Figura 45. Resultados con los parámetros de la tabla 12 en el eje Y.

Anexo 5. Comunicación Spyder- Simulink.

Puerto serie virtual de Eltima

Virtual Serial Port Driver crea puertos de serie virtuales y los conecta en pares a través de cable de módem nulo virtual. Aplicaciones en ambos extremos del par podrán intercambiar datos, de tal manera, que todo lo escrito en el primer puerto aparecerá en el segundo y viceversa. El esquema de esta comunicación se puede ver en la figura 46.

Todos los puertos serie virtuales funcionan y se comportan exactamente igual que los reales, emulando todas sus configuraciones. Puede crear tantos pares de puerto virtual como desee, por lo que no habrá escasez de puertos serie y sin ningún hardware adicional en el escritorio.

Figura 46. Esquema puerto serie virtual.

Para comunicarse entre la aplicación de visión y la aplicación del puente grúa en Matlab/Simulink, es necesaria una comunicación entre las dos aplicaciones.

La idea del trabajo es realizar todo desde el mismo equipo, en este caso un ordenador, sin utilizar ningún dispositivo externo como pudiera ser un microcontrolador. Por ello, se utiliza un puerto serie virtual que crea pares de puertos. Un puerto recibe los datos de las coordenadas de la marca objetivo desde la aplicación de visión y los envía por el puerto restante comunicado con Matlab/Simulink.

Se crean dos pares de puertos, uno para el envío de la coordenada X de posición de la carga (COM1-COM2) y otro para el envío de la coordenada Y de posición de la carga (COM3-COM4). Se comprueba que los dos pares de puertos han sido creados correctamente, tal y como se observa en la figura 47.

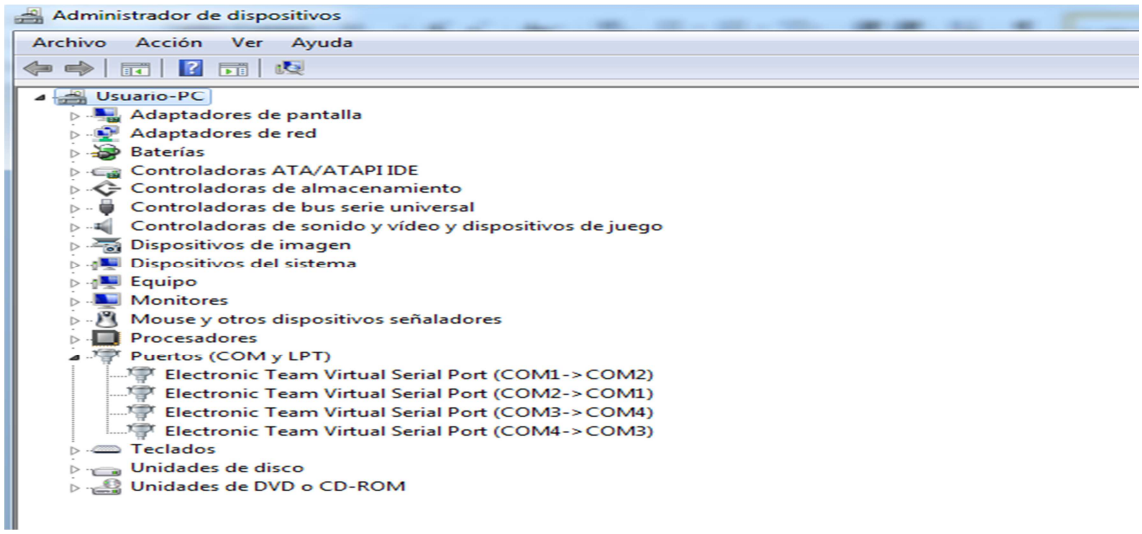

Figura 47. Puertos serie virtuales creados.

Se establece la conexión entre Spyder y el puerto serie virtual y se comprueba el envío correcto de números reales desde Spyder a la entrada del puerto serie. Para ello se crea un programa en el que se generan números reales entre cero y uno y se suman de forma acumulativa hasta que dicha suma sea superior a diez. En cada iteración: se conecta la aplicación con el puerto serie, se envía el resultado de la suma en forma de bytes, imprime por pantalla el valor enviado y el tipo de las variables tratadas, y se cierra el puerto serie. La aplicación de envío de datos se muestra a continuación.

```
import random
import serial
import time


sum = 0.0
while (sum <= 10.0):
  data = serial.Serial('COM2',9600)
  time.sleep(1)
  val = random.uniform(0, 1)
  sum = sum + val
  print(type(sum))
  cadena = str(sum)
  print(type(cadena))
  valor = str.encode(cadena) + bytes([13,10])
  print(type(valor))
  data.write(valor)
  print(sum)
  print(valor)
  data.close()
data.close()
```

Los resultados imprimidos por pantalla se pueden ver en la figura 48.

```
In [4]: runfile('C:/Users/Usuario/.spyder-py3/pruebafloatserie.py', wdir='C:/Users/
Usuario/.spyder-py3')
<class 'float'>
<class 'str'>
<class 'bytes'>
0.1383361269511707
<class 'float'>
<class 'str'>
<class 'bytes'>
1.1000399855582597
<class 'float'>
<class 'str'>
<class 'bytes'>
1.4174860724557155
<class 'float'>
<class 'str'>
<class 'bytes'>
1.934314611647018
<class 'float'>
<class 'str'>
<class 'bytes'>
2.27622445635876
```

Figura 48. Datos enviados desde Spyder a la entrada del puerto serie.

Estos datos son enviados desde la salida del puerto serie a Matlab/Simulink utilizando la siguiente aplicación en Matlab:

```
clc;clear all;close all;
s = serial('COM1');
set(s,'BaudRate',9600);
set(s,'DataBits',8);
set(s,'Parity','none');
set(s,'StopBits',1);
set(s,'FlowControl','none');
set(s,'TimeOut',0.5);
while 1
 fopen(s);
 s.ReadAsyncMode = 'continuous';
 readasync(s);
 out = fscanf(s);
 disp(out);
 fclose(s);
end
fclose(s);
delete(s);
clear s;
```

Los datos recibidos se muestran en la figura 49, y se verifica la funcionalidad de las aplicaciones realizadas para la obtención de datos.

Figura 49. Datos recibidos en Matlab por el puerto serie de Eltima.

Comunicación UDP

```
import random
import socket
import time
import struct

sum = 0.0
while (sum <= 30.0):
  sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
  server_address = ('localhost', 10000)
  time.sleep(0.5)
  val = random.uniform(0, 1)
  sum = sum + val
  packer = struct.Struct('f')
  packed_data = packer.pack(sum)

  print(sum)

  try:

      sent = sock.sendto(packed_data, server_address)

  finally:
      sock.close()
```

Anexo 6.Programa completo de Spyder

```
import numpy as np
import cv2
import struct
import socket
import time

font = cv2.FONT_HERSHEY_SIMPLEX
ox=0
oy=0
cap = cv2.VideoCapture(1)
while(True):

  ret, frame = cap.read()
  hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
  rojo_bajos1 = np.array([0,65,75], dtype=np.uint8)
  rojo_altos1 = np.array([12, 255, 255], dtype=np.uint8)
  rojo_bajos2 = np.array([240,65,75], dtype=np.uint8)
  rojo_altos2 = np.array([256, 255, 255], dtype=np.uint8)
  mascara_rojo1 = cv2.inRange(hsv, rojo_bajos1, rojo_altos1)
  mascara_rojo2 = cv2.inRange(hsv, rojo_bajos2, rojo_altos2)
  mask = cv2.add(mascara_rojo1, mascara_rojo2)
  kernel = np.ones((7,7),np.uint8)
  erosion = cv2.erode(mask,kernel,iterations = 1)
  dilatacion = cv2.dilate(erosion,kernel,iterations = 1)
  cv2.imshow('imagen tras filtrado de ruido',dilatacion)
  contours,_ = cv2.findContours(dilatacion, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)
  cv2.drawContours(frame, contours, -1, (0,255,0), 2)
  a=0
  x=[0,0,0,0]
  y=[0,0,0,0]
  for i in contours:
    momentos = cv2.moments(i)
    area = cv2.contourArea(i)
    if area > 400:
     cx = int(momentos['m10']/momentos['m00'])
     cy = int(momentos['m01']/momentos['m00'])
     x[a] = cx
     y[a] = cy
     a=a+1

  print (x)
  print (y)
  x2=sorted(x)
  y2=sorted(y)
  print(x2)
  print(y2)
  rows,cols,ch = frame.shape
  pts1 = np.float32([[x2[0],y2[1]], [x2[2],y2[0]], [x2[1],y2[3]],
[x2[3],y2[2]]])
  pts2 = np.float32([[0,0],[920,0],[0,920],[920,920]])
  M = cv2.getPerspectiveTransform(pts1,pts2)
  imgplanta = cv2.warpPerspective(dilatacion,M,(920,920))
  cv2.imshow('imagen en planta',imgplanta)
  rows,cols = imgplanta.shape
  N = cv2.getRotationMatrix2D((cols/2,rows/2),180,1)
```

```
  rotada = cv2.warpAffine(imgplanta,N,(cols,rows))
  contours2,_ = cv2.findContours(rotada, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)
  for i in contours2:
    momentos = cv2.moments(i)
    area = cv2.contourArea(i)
    if area > 5000:
     ox = int(momentos['m10']/momentos['m00'])
     oy = int(momentos['m01']/momentos['m00'])
    print(ox)
    print(oy)
  sock1 = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
  server_address1 = ('localhost', 10000)
  sock2 = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
  server_address2 = ('localhost', 8080)
  time.sleep(0.1)
  packer1 = struct.Struct('f')
  packed_data1 = packer1.pack(ox)
  packer2 = struct.Struct('f')
  packed_data2 = packer2.pack(oy)
  try:

      sent = sock1.sendto(packed_data1, server_address1)
      sent = sock2.sendto(packed_data2, server_address2)

  finally:
      sock1.close()
      sock2.close()
  k = cv2.waitKey(30) & 0xFF
  if k==27:
      break

cap.release()
cv2.destroyAllWindows()
```