**2. Sesión de Movimiento del manipulador usando una cámara– Prerrequisito: Sesión 2.1.**

**Actividad guiada II:** Paquete en ROS para la segmentación de objetos con

forma redonda usando OpenCV y los controladores de ROBOTIS.

# **¿CÓMO FUNCIONA?**

La segmentación de objetos por forma comienza convirtiendo la imagen original a escala de grises para mejorar la visibilidad de sus características geométricas. Luego, se aplica la función **Canny** de OpenCV para detectar los bordes, definiendo un rango de detección. Un valor mínimo más bajo aumenta la sensibilidad a líneas suaves, y este rango puede ajustarse según factores como la iluminación del ambiente.

Después, se encuentran los contornos y se simplifican con **approxPolyDP** para determinar el número de lados de cada objeto. En el caso del círculo, se considera que debe tener al menos **8 lados**. Cuando se cumple este criterio, se dibuja su contorno sí y solo si tiene forma redonda.

Posteriormente, cuando se detecta un objeto de forma redonda, se utiliza la función moments de OpenCV para calcular el centroide, es decir, las coordenadas x e y del objeto en la imagen. Considerando que las dimensiones de la imagen son de 320x240 píxeles, el valor de x oscilará entre 0 y 320, y el valor de y entre 0 y 240. Una vez obtenidas estas coordenadas, se dibujan en la imagen y se publican en el tópico **/objeto\_centroide** para su posterior uso o visualización.

# **PROCEDIMIENTO**

1. Para segmentar objetos de forma redonda, primero creamos un paquete en ROS usando el siguiente comando:

$ cd ~/catkin\_ws/src

$ catkin\_create\_pkg segmentacion\_forma rospy std\_msgs

$ cd ~/catkin\_ws

$ catkin\_make



1. En el paquete crea los folders **src, scripts, launch** que contendrán los archivos necesarios para ejecutar nuestra segmentación. Para ello usa los siguientes comandos:

$ cd ~/catkin\_ws/src/segmentacion\_forma

$ mkdir scripts launch

$ cd ~/catkin\_ws

$ catkin\_make

1. Con los folders necesarios ya creados, es hora de crear nuestros scripts. Uno de nuestros scripts nombrado ***segmentacion\_forma\_redonda.py*** cuya finalidad es realizar el reconocimiento de objetos de forma redonda. Y el script nombrado ***posicion\_reconocimiento.py*** que enviará nuestro robot a la posición de reconocimiento.

$ cd ~/src/segmentacion\_forma/scripts

$ nano segmentacion\_forma\_redonda.py posicion\_reconocimiento.py

1. Ahora, agrega un código como este en el archivo segmentacion\_forma\_redonda.py

#!/usr/bin/env python3

import rospy

from sensor\_msgs.msg import Image

from cv\_bridge import CvBridge, CvBridgeError

import cv2

import numpy as np

from geometry\_msgs.msg import Point # Tipo de mensaje para coordenadas

class reconocimiento\_forma(object):

def camera\_callback(self,data):

bridge = CvBridge()

try:

cv\_image = bridge.imgmsg\_to\_cv2(data, desired\_encoding="bgr8")

except CvBridgeError as e:

print(e)

return

cv\_image = cv2.resize(cv\_image, (320, 240))

hsv = cv2.cvtColor(cv\_image, cv2.COLOR\_BGR2HSV)

gray\_image = cv2.cvtColor(cv\_image, cv2.COLOR\_BGR2GRAY)

cv2.imshow('gray\_image', gray\_image)

canny\_min = 10

canny\_max=150

canny=cv2.Canny(gray\_image,canny\_min,canny\_max)

cv2.imshow('canny', canny)

# Encontrar y dibujar contornos

contours, \_ = cv2.findContours(canny, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

for contour in contours:

approx = cv2.approxPolyDP(contour, 0.02 \* cv2.arcLength(contour, True), True)

x, y, w, h = cv2.boundingRect(approx)

if len(approx) == 8:

M = cv2.moments(contour)

if M["m00"] != 0:

cx = int(M["m10"] / M["m00"])

cy = int(M["m01"] / M["m00"])

cv2.circle(cv\_image, (cx, cy), 5, (0, 255, 0), -1) # Dibujar centroide

cv2.drawContours(cv\_image, [contour], -1, (0, 255, 255), 9)

#publicar e topico de centroide

centroide\_msg = Point()

centroide\_msg.x = cx

centroide\_msg.y = cy

centroide\_msg.z = 0 # Si no hay profundidad, deja en 0

if not rospy.is\_shutdown():

self.publicador\_centroide.publish(centroide\_msg)

cv2.imshow("Shape Detection", cv\_image)

cv2.waitKey(1)

def \_\_init\_\_(self):

rospy.init\_node('segmentador\_objetos\_forma', anonymous=True)

rospy.Subscriber("/camera/color/image\_raw", Image, self.camera\_callback)

self.publicador\_centroide = rospy.Publisher('/objeto/Centroide', Point, queue\_size=10)

try:

rospy.spin()

except KeyboardInterrupt:

print("Shutting down")

cv2.destroyAllWindows()

if \_\_name\_\_ == '\_\_main\_\_':

node = reconocimiento\_forma()



1. Ahora en el archivo posicion\_reconocimiento.py agrega el siguiente código:

#!/usr/bin/env python3

import rospy

from geometry\_msgs.msg import Point # Tipo de mensaje para coordenadas

from sensor\_msgs.msg import JointState, Image

from open\_manipulator\_msgs.srv import (

SetActuatorState, SetActuatorStateRequest,

SetJointPosition, SetJointPositionRequest,

SetKinematicsPose, SetKinematicsPoseRequest)

from open\_manipulator\_msgs.msg import JointPosition, KinematicsPose, OpenManipulatorState

class movimiento\_pos\_reconocimiento:

def setJointSpacePath(self, joint\_name, joint\_angle, path\_time):

try:

srv = SetJointPositionRequest()

srv.joint\_position.joint\_name = joint\_name

srv.joint\_position.position = joint\_angle

srv.path\_time = path\_time

response = self.set\_joint\_position\_client(srv)

return response.is\_planned

except rospy.ServiceException as e:

rospy.logerr(f"Service call failed: {e}")

return False

#######################################################################################

def pos\_reco(self):

kinematics\_pose = [0.2300, 0.0000, 0.0793]

print(f"{kinematics\_pose}")

#print(f"{kinematics\_pose}")

#l gripper angle siempre abierto

path\_time=2.0

if not self.setTaskSpacePath(kinematics\_pose, path\_time):

print("[ERR!!] Failed to send joint angles")

return

print("Send task pose + gripper position")

def setToolControl(self, joint\_angle):

try:

srv = SetJointPositionRequest()

srv.joint\_position.joint\_name.append("gripper")

srv.joint\_position.position = joint\_angle

response = self.set\_tool\_control\_client(srv)

return response.is\_planned

except rospy.ServiceException as e:

rospy.logerr(f"Service call failed: {e}")

return False

def pos\_inicial(self):

joint\_angle = 0.01

joint\_name = ["joint1", "joint2", "joint3", "joint4"]

joint\_angle = [0.000, 0.000 , 0.000, 1.456] # Ángulos de la posición inicial

#joint\_angle = [0.000, 0.040, 0.410, 0.800 ] # Ángulos modificados para posición de reset

path\_time = 2.0 # Tiempo de movimiento

success = self.setJointSpacePath(joint\_name, joint\_angle, path\_time)

joint\_angle = [0.01] # Ángulo para cerrar gripper

if not self.setToolControl(joint\_angle):

print("[ERR!!] Failed to send service")

return

def setActuatorState(self, actuator\_state):

try:

srv = SetActuatorStateRequest()

srv.set\_actuator\_state = actuator\_state

response = self.set\_actuator\_state\_client(srv)

return response.is\_planned

except rospy.ServiceException as e:

rospy.logerr(f"Service call failed: {e}")

return False

def \_\_init\_\_(self):

self.kinematics\_pose = None

self.present\_kinematic\_position = [0.0, 0.0, 0.0] # Inicialización segura

self.open\_manipulator\_is\_moving = False

self.open\_manipulator\_actuator\_enabled = False

rospy.init\_node('nodo\_publicador\_pos\_reconocimiento', anonymous=True)

self.joint\_state\_subscriber = rospy.Subscriber('/joint\_states', JointState, self.joint\_states\_callback)

self.open\_manipulator\_states\_sub = rospy.Subscriber("/states", OpenManipulatorState, self.manipulator\_states\_callback)

self.set\_actuator\_state\_client = rospy.ServiceProxy('/set\_actuator\_state', SetActuatorState)

self.set\_joint\_position\_client = rospy.ServiceProxy('/goal\_joint\_space\_path', SetJointPosition)

self.set\_tool\_control\_client = rospy.ServiceProxy('/goal\_tool\_control', SetJointPosition)

self.goal\_task\_space\_path\_position\_only\_client = rospy.ServiceProxy('/goal\_task\_space\_path\_position\_only', SetKinematicsPose)

self.open\_manipulator\_states\_sub = rospy.Subscriber("/states", OpenManipulatorState, self.manipulator\_states\_callback)

def joint\_states\_callback(self, msg):

temp\_angle = [0.0] \* 5

for i in range(len(msg.name)):

if msg.name[i] == "joint1":

temp\_angle[0] = msg.position[i]

elif msg.name[i] == "joint2":

temp\_angle[1] = msg.position[i]

elif msg.name[i] == "joint3":

temp\_angle[2] = msg.position[i]

elif msg.name[i] == "joint4":

temp\_angle[3] = msg.position[i]

elif msg.name[i] == "gripper":

temp\_angle[4] = msg.position[i]

self.present\_joint\_angle = temp\_angle

def get\_open\_manipulator\_moving\_state(self):

return self.open\_manipulator\_is\_moving

def get\_open\_manipulator\_actuator\_state(self):

return self.open\_manipulator\_actuator\_enabled

def manipulator\_states\_callback(self, msg):

self.open\_manipulator\_is\_moving = msg.open\_manipulator\_moving\_state == msg.IS\_MOVING

self.open\_manipulator\_actuator\_enabled = msg.open\_manipulator\_actuator\_state == msg.ACTUATOR\_ENABLED

def publicar\_mensaje(self):

if not self.setActuatorState(True): # Llamada para habilitar los actuadores

print("No se hbilitaron los actuadores")

return

self.pos\_inicial()

def run(self):

rospy.spin()

if \_\_name\_\_ == "\_\_main\_\_":

try:

nodo = movimiento\_pos\_reconocimiento()

nodo.publicar\_mensaje()

except rospy.ROSInterruptException:

pass



1. No olvides convertir los scripts en archivos ejecutables:

$ cd ~/catkin\_ws/src/segmentacion\_forma/scripts

$ chmod +x segmentacion\_forma\_redonda.py posicion\_reconocimiento.py

1. Ahora, debes crear el archivo de lanzamiento **segmentacion\_forma.launch** en el folder launch.

$ cd ~/catkin\_ws/src/segmentacion\_forma/launch

$ nano segmentacion\_forma.launch

1. Ahora le agregas las siguientes líneas de código al archivo segmentacion\_forma.launch

<launch>

<node name="nodo\_segmentador\_forma" pkg="segmentacion\_forma" type="segmentacion\_forma\_redonda.py" output="screen"/>

<node name="nodo\_posicion\_reconocimiento" pkg="segmentacion\_forma" type="posicion\_reconocimiento.py" output="screen"/>

</launch>

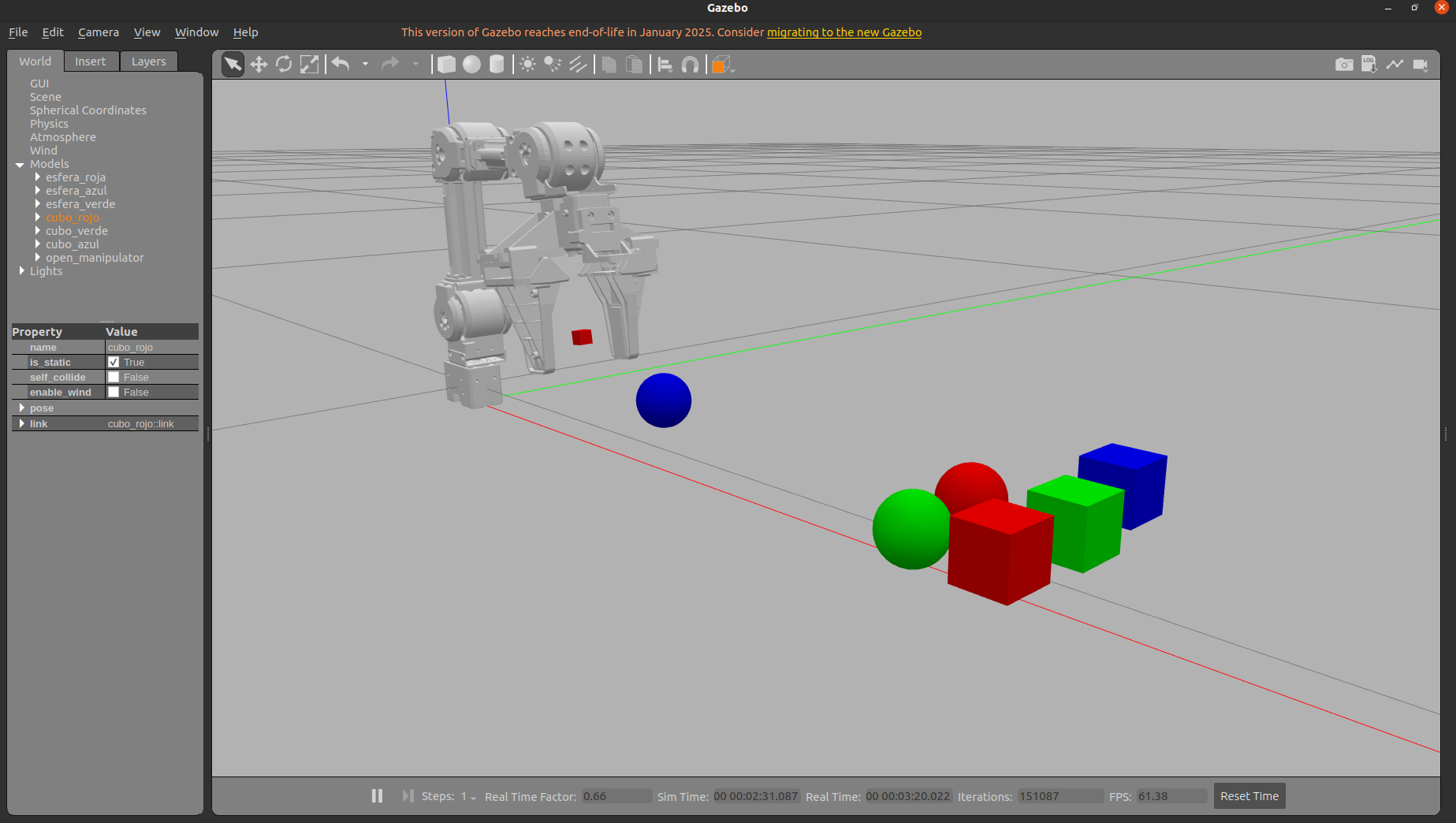
1. Ahora, ya es momento de ejecutar nuestro paquete en ROS.

# **EJECUCIÓN usando Gazebo**

1. Si deseas usar el robot Manipulator X en Gazebo haz los siguientes pasos:
2. Lanza el mundo en Gazebo con el siguiente comando:

roslaunch open\_manipulator\_gazebo open\_manipulator\_gazebo.launch

¡No olvides presionar Start en la simulación!



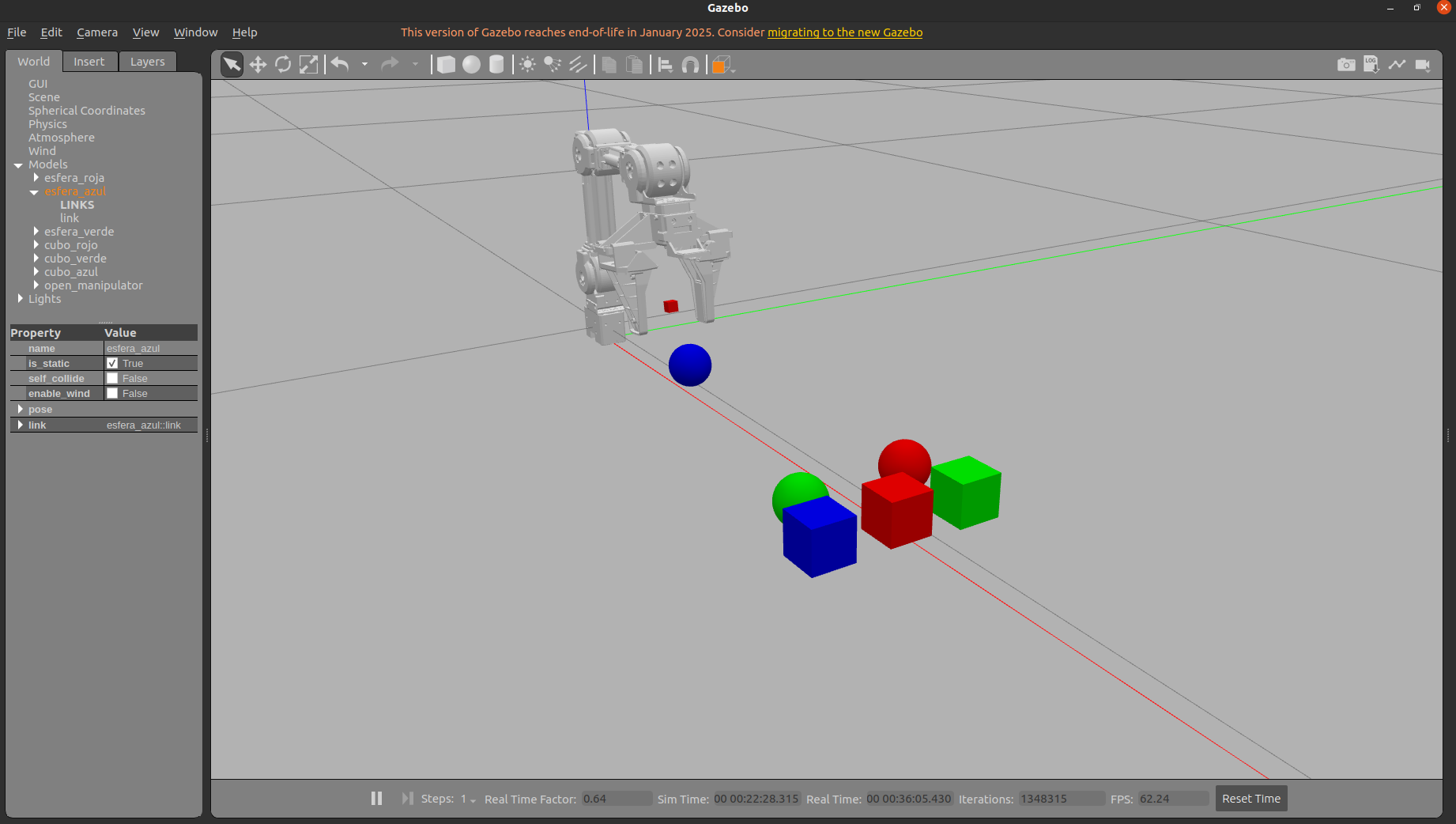
1. Ahora, lanza el controlador para poder activar el movimiento y la cámara del robot.

roslaunch open\_manipulator\_controller open\_manipulator\_controller.launch use\_platform:=false

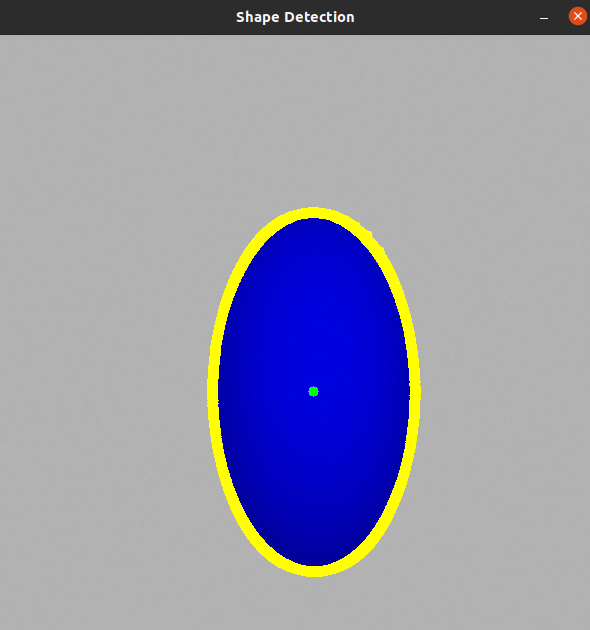
1. Ya está preparado el robot para atendernos. Ahora, ya puedes lanzar nuestro paquete segmentador de objetos de forma redonda:

roslaunch segmentacion\_forma segmentacion\_forma.launch

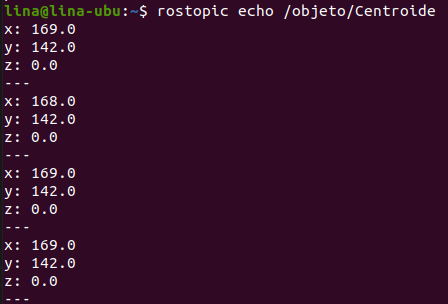
verás como el robot se desplaza a la “posición de reconocimiento”, luego, se abre la ventana de Opencv con la imagen de lo que captura la cámara del robot.



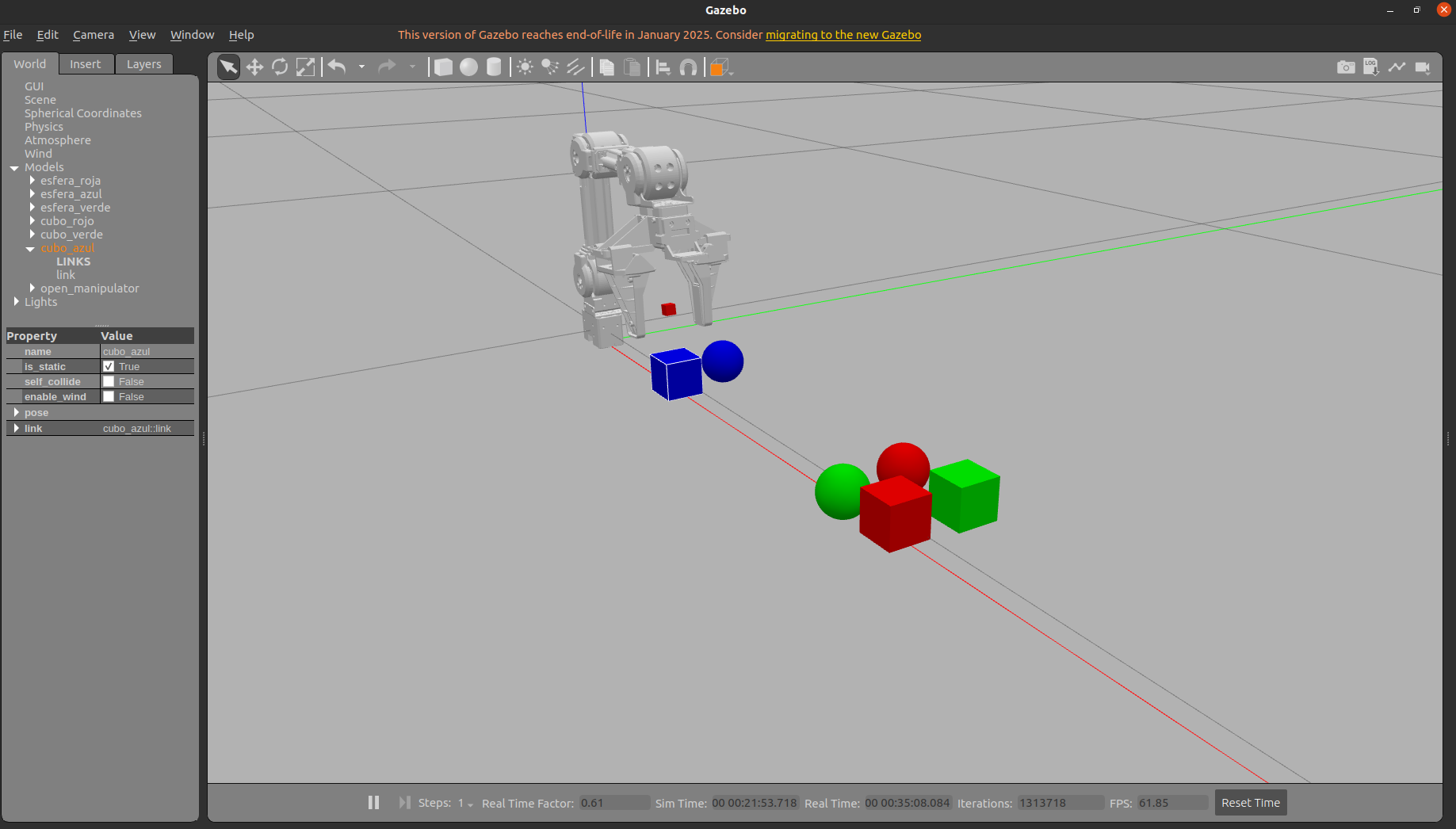
Verás que si el parámetro de cantidad de bordes es 8 reconoce perfectamente el círculo



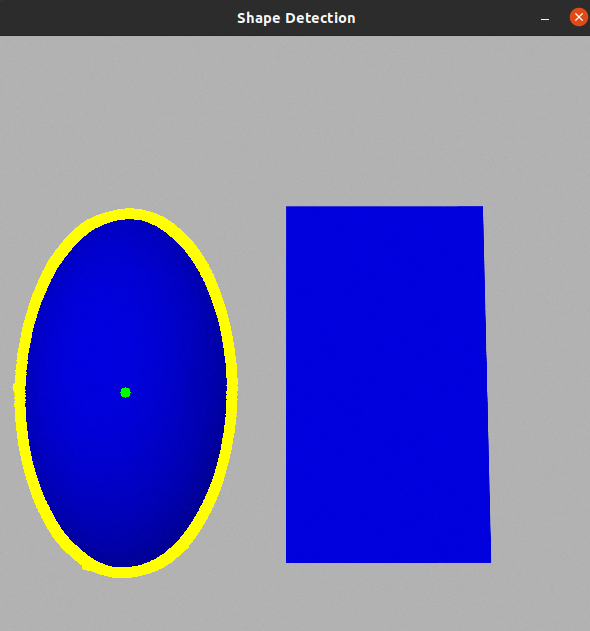
Del mismo modo, se observa en el tópico /objeto/centroide las coordenadas x,y de ubicación del objeto en la imagen.



Veamos sí el robot puede reconocer objetos de forma redonda aún cuando hay otro objeto del mismo color



Verás como se contornea el objeto con forma redonda, aún cuando hay otro objeto del mismo color.



Recuerda que puedes mover los objetos en el mundo en gazebo y serán reconocidos en tiempo real.

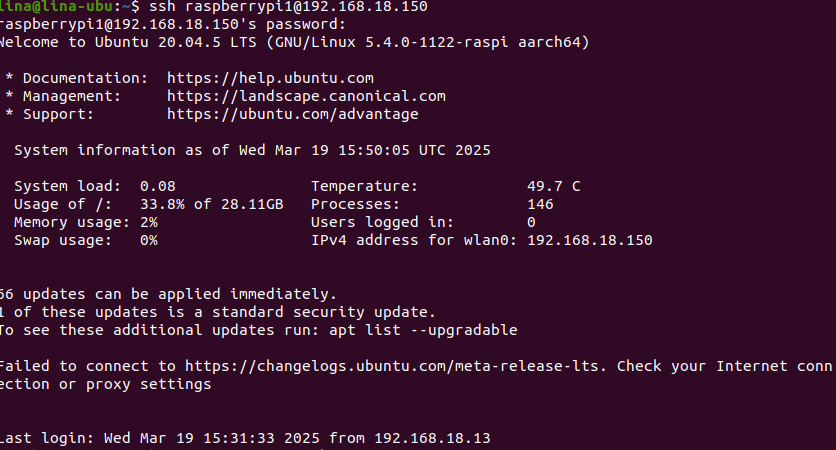
# **EJECUCIÓN usando robot real**

1. Si desea usar el robot real, primero debe conectarlo correctamente, asegúrese de conectar la potencia de la OpenCR y la alimentación de la RaspBerry.
2. Espere unos segundos mientras la raspberry se conecta a la red local. Luego intente conectarse vía ssh con el siguiente comando en la terminal de Ubuntu.

$ ssh raspberrypi1@192.168.18.150

Luego, ingrese la contraseña: **manipulatorx1**

Verás lo siguiente:

****

1. Ahora, ya estará conectado al servidor del robot, primero lanza el roscore para trabajar en un entorno de ROS compartido.

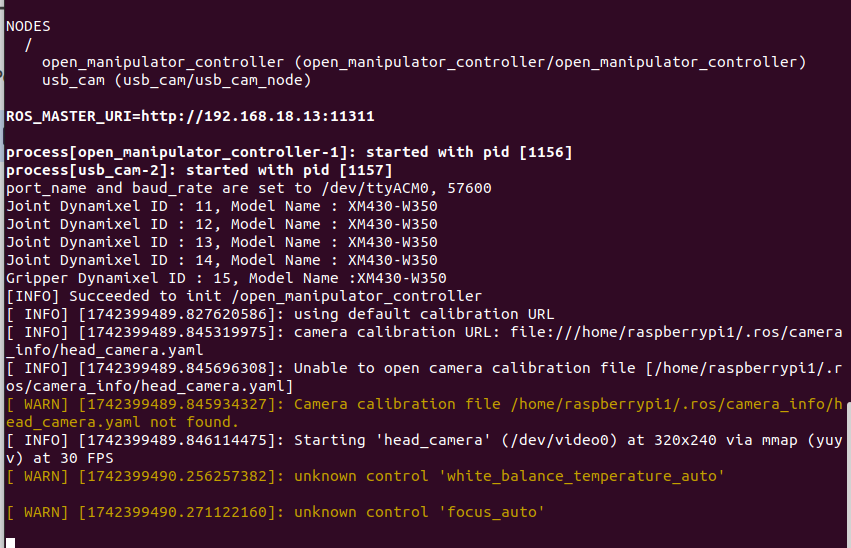
$ roscore



1. Por lo que ya es posible lanzar el controlador que activa los motores y la cámara, lanzando el siguiente comando:

$ roslaunch open\_manipulator\_controller open\_manipulator\_controller.launch usb\_port:=/dev/ttyACM0 baud\_rate:=57600

Verás lo siguiente:



1. Ya está preparado el robot para atendernos. Ahora, ya puedes lanzar el paquete de segmentación de objetos de forma redonda, con el siguiente comando:

roslaunch segmentacion\_forma segmentacion\_forma.launch



vemos, que el robot se dirige a su posición de reconocimiento.