Segmentación objetos con color azul

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

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

azul usando OpenCV.

# **¿CÓMO FUNCIONA?**

La segmentación de color se hace mediante la creación de máscaras binarias sobre la imagen. Estas máscaras requieren de un rango de color HSV cuya función de Opencv convierte los píxeles dentro del rango de color especificado se establecen en blanco (255 en escala de grises), y los que están fuera se establecen en negro (0).

El color rojo tiene un rango HSV mínimo: [0,50,50] y un HSV máximo: [10,255,255]

El color azul tiene un rango HSV mínimo: [100,50,50] y un HSV máximo: [140,255,255]

El color verde tiene un rango HSV mínimo: [40,50,50] y un HSV máximo: [80,255,255]

La cámara del robot tiene una resolución de 320x240 píxeles, optimizada para una mejor velocidad de procesamiento, y publica las imágenes en el tópico ***/usb\_cam/image\_raw.***

El robot debe ubicarse en la **posición de reconocimiento** para garantizar una perspectiva adecuada del objeto, asegurando que sea completamente visible y minimizando los efectos de sombras o variaciones en la iluminación del entorno. Esta posición de reconocimiento se encuentra en el plano espacial: [j1, j2, j3, j4] = [0.0, 0.0, 1.456, 0.0] y en el plano cartesiano: [x,y,z] = [17.2, 0.0, 7.1 ].

# **PROCEDIMIENTO**

1. Para segmentar objetos por color azul, primero creamos un paquete en ROS usando el siguiente comando:

$ cd ~/catkin\_ws/src

$ catkin\_create\_pkg segmentacion\_color 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\_color

$ 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\_color\_azul.py*** cuya finalidad es realizar el reconocimiento por color azul. Y el script nombrado ***posicion\_reconocimiento.py*** que enviará nuestro robot a la posición de reconocimiento.

$ cd ~/src/segmentacion\_color

$ nano segmentacion\_color\_azul.py posicion\_reconocimiento.py

1. Ahora, agrega un código como este en el archivo color\_azul.py

#!/usr/bin/env python

import rospy

from sensor\_msgs.msg import Image

from cv\_bridge import CvBridge, CvBridgeError

import cv2

import numpy as np

from std\_msgs.msg import Bool

# Variables globales para rangos de color en HSV

global hsv\_min, hsv\_max

hsv\_min = np.array([100, 50, 50]) # HSV mínimo

hsv\_max = np.array([140, 255, 255]) # HSV máximo

def camera\_callback(data):

global hsv\_min, hsv\_max

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

mask = cv2.inRange(hsv\_image, hsv\_min, hsv\_max)

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

for contour in contours:

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

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)

cv2.imshow('Original', cv\_image)

cv2.waitKey(1)

def main():

rospy.init\_node('nodo\_segmentador\_Color', anonymous=True)

rospy.Subscriber("/usb\_cam/image\_raw", Image, camera\_callback)

try:

rospy.spin()

except KeyboardInterrupt:

print("Shutting down")

cv2.destroyAllWindows()

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

main()

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\_color/scripts

$ chmod +x segmentacion\_color\_azul.py posicion\_reconocimiento.py

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

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

$ nano segmentacion\_color.launch

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

<launch>

<node name="nodo\_segmentador\_Color" pkg="segmentacion\_color" type="segmentacion\_color\_azul.py" output="screen"/>

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

</launch>

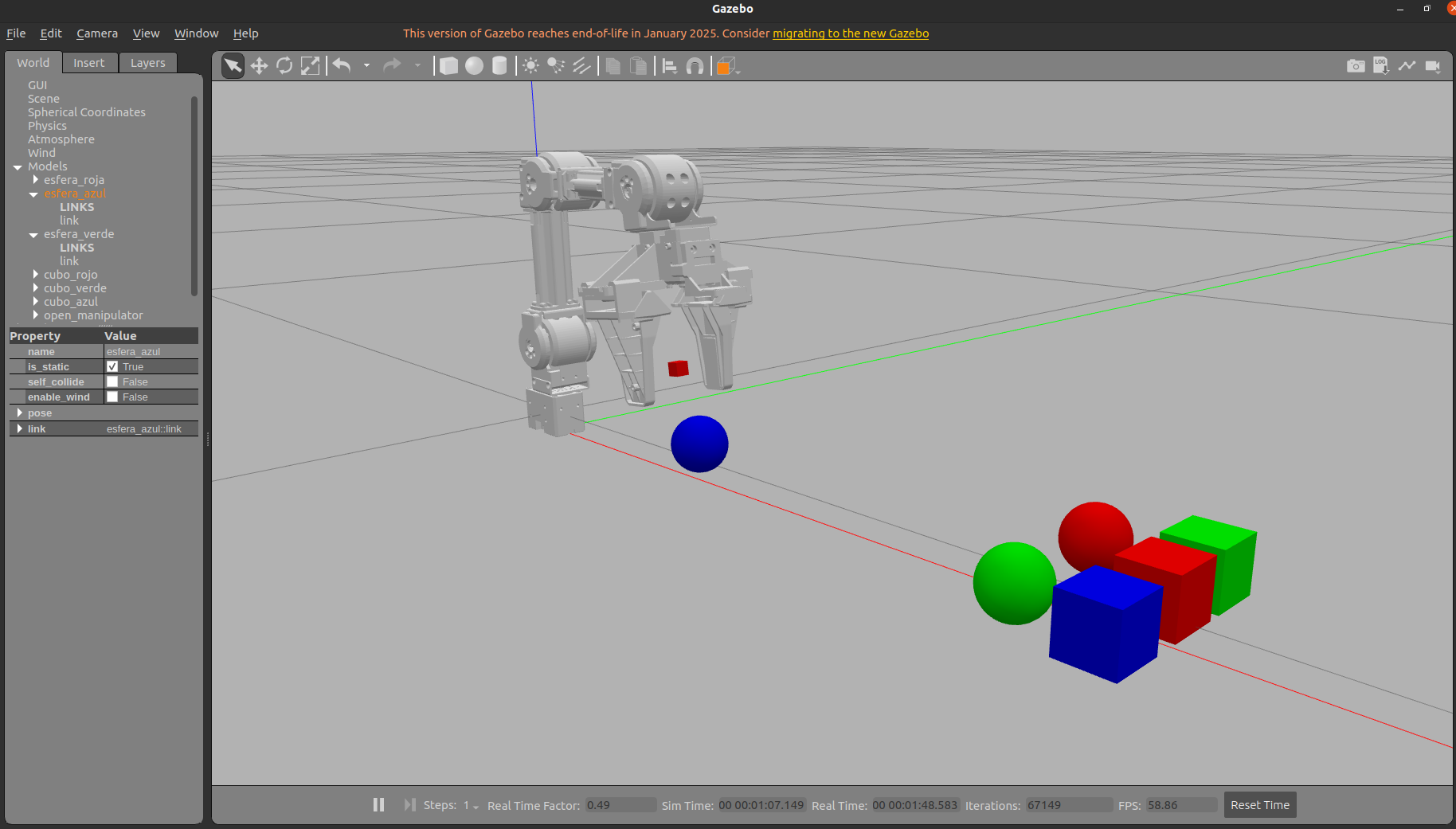
1. Ahora, ya es momento de ejecutar nuestro paquete en ROS.

# **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!



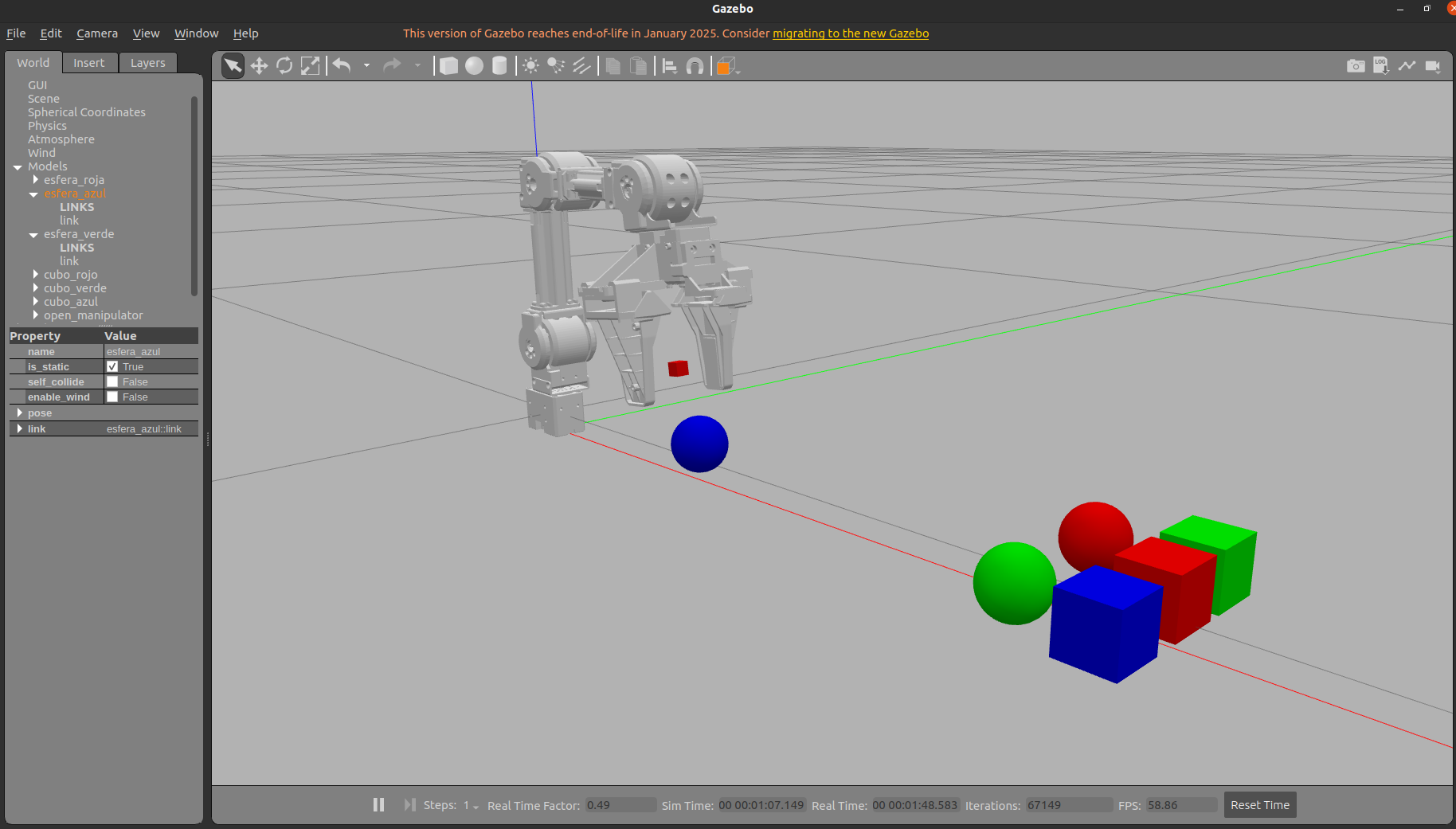
1. Ahora, lanza el controlador para poder activar el movimiento y la cámara del robot.

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

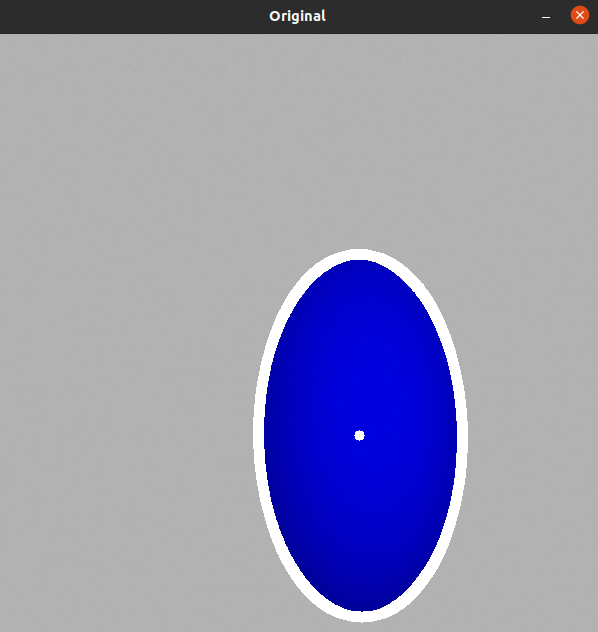
1. Ya está preparado el robot para atendernos. Ahora, para ejecutar nuestro código segmentador de objetos por color azul lanza el comando:

roslaunch segmentacion\_color segmentacion\_color.launch

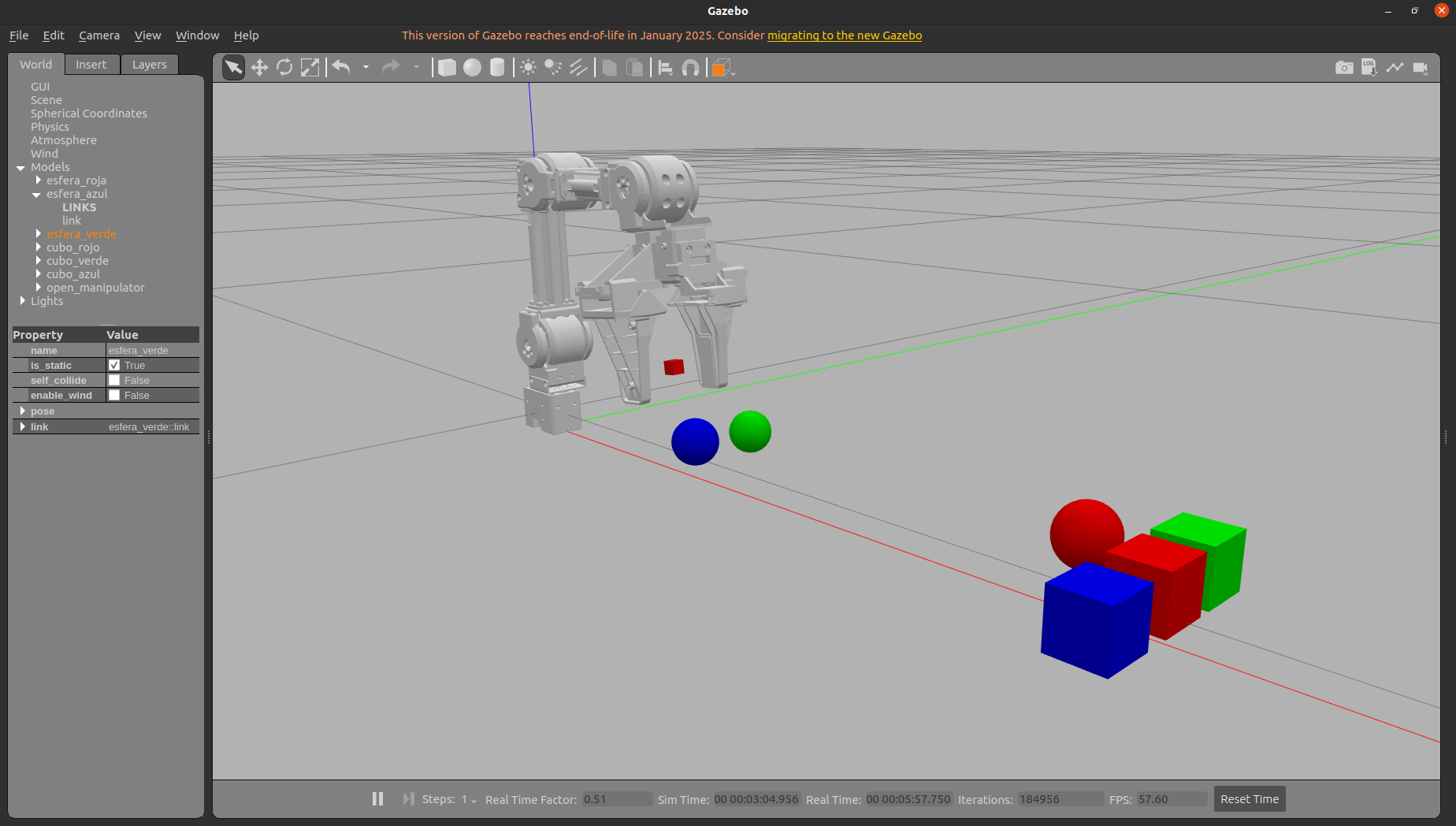
verás como el robot se desplaza a la “posición de reconocimiento”, luego, se abre la ventana de Opencv con la imagen de lo que captura la cámara del robot.



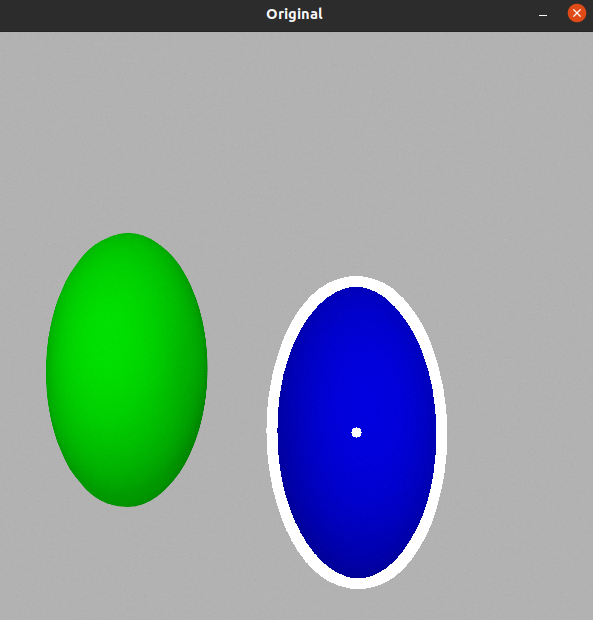
Verás que con un rango HSV correspondiente al color azul, sí reconoce la esfera azul.



Veamos sí el robot puede reconocer el color azul aun cuando hay otro objeto con la misma forma pero diferente color:



Verás como se contornea el color azul aún cuando hay un objeto con la misma forma pero de otro color.



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

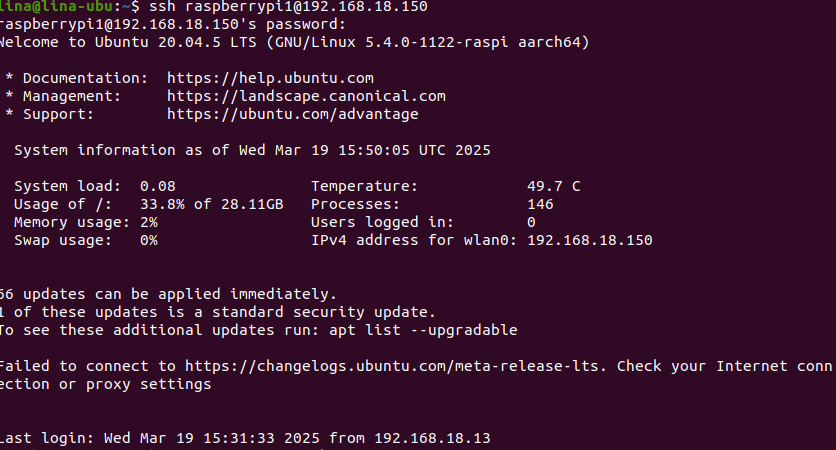
# **EJECUCIÓN usando robot real**

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.

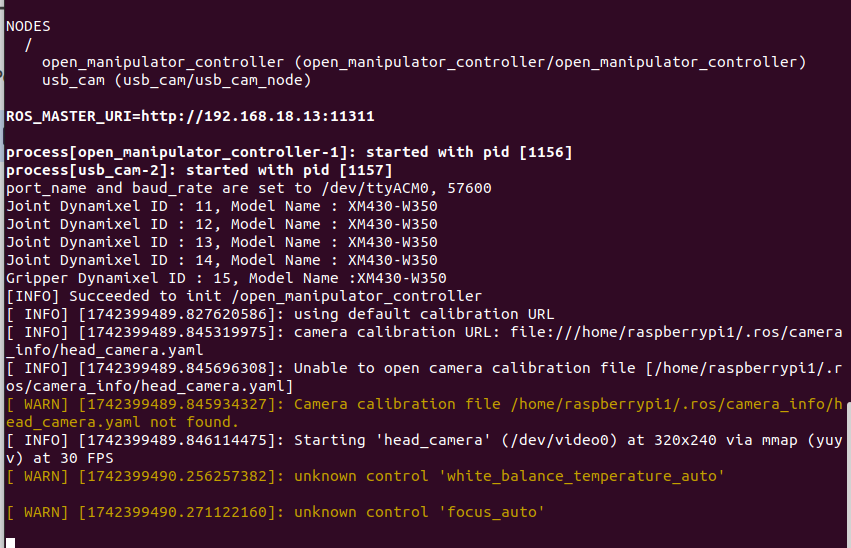
$ roscore



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:

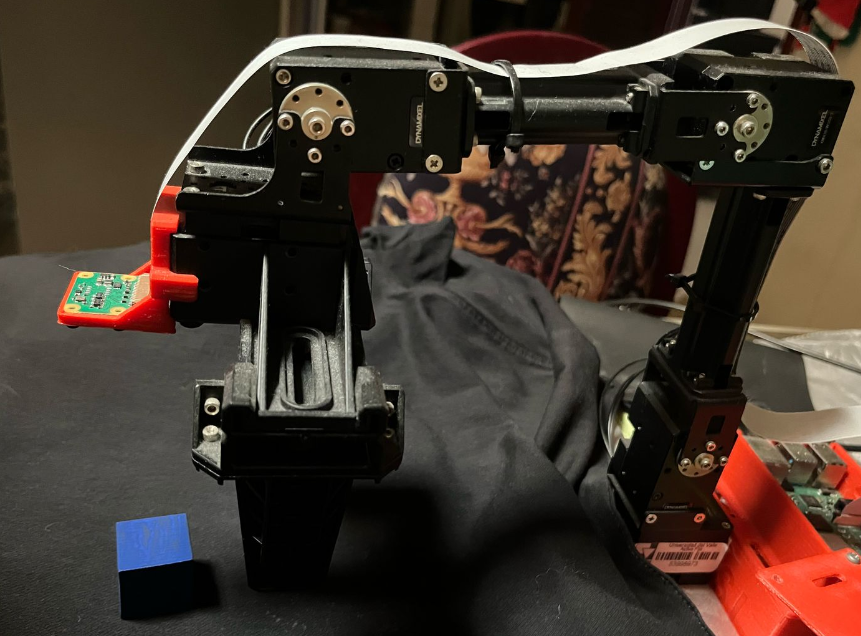


1. Ya está preparado el robot para atendernos. ya puedes lanzar el paquete de segmentación de color azul, con el siguiente comando:

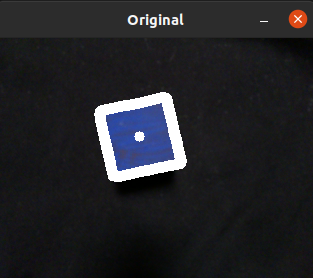
roslaunch segmentacion\_color segmentacion\_color.launch



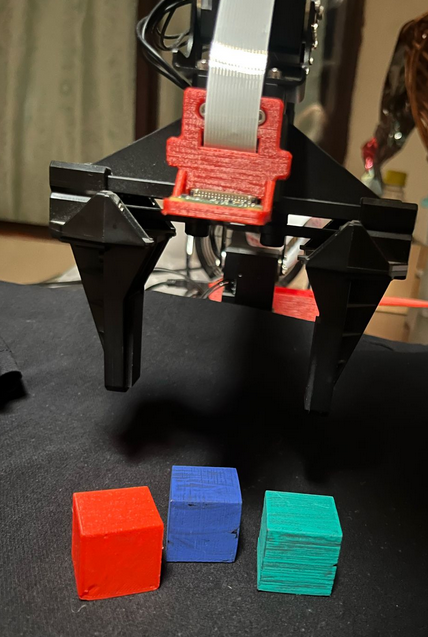
vemos, que el robot se dirige a su posición de reconocimiento.

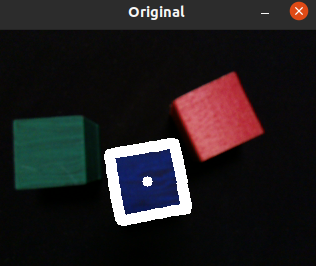


Y posteriormente la ventana de Opencv nos muestra que sí reconoce el objeto de color azul:

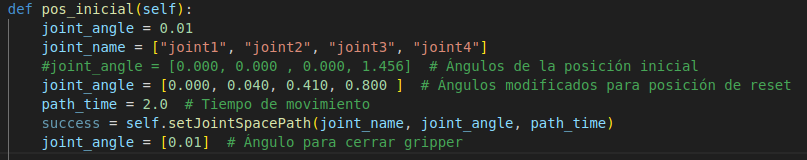


Puedes comprobarlo con tres objetos de la misma forma, pero diferente color. Observa como reconoce el color azul.





**NOTA IMPORTANTE:** Si se encuentra trabajando con el **ROBOT REAL** y desea ya no utilizar nuestro paquete de segmentación de color azul con CTRL+C, es necesario que envíe al robot a una posición segura, para evitar que el robot sufra daños por golpe al deshabilitar los actuadores, lo puede hacer descomentando la línea que corresponde a la posición de reset, y comentando la de posición inicial, así:



Guarda los cambios y ejecuta el script con el comando:

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

$./posicion\_reconocimiento

