**3. Sesión de Selección y movimiento manual y automática de objetos de interés**

**usando el manipulador y la cámara – Prerrequisito: Sesión 2.2.**

**Actividad guiada II:** Actividad guiada: Paquete en ROS usando los controladores de ROBOTIS

para mover automáticamente el efector final del manipulador a una posición

definida donde se encuentra un objeto de un color específico y agarrarlo.

# **1. ¿CÓMO FUNCIONA?**

**1.1 Posición de reconocimiento**

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 ].

**1.2 Reconocimiento de color (percepción)**

Para el reconocimiento del color, debemos saber que la cámara que utiliza el Open Manipulator X es una pi Camera V2.1 como esta:

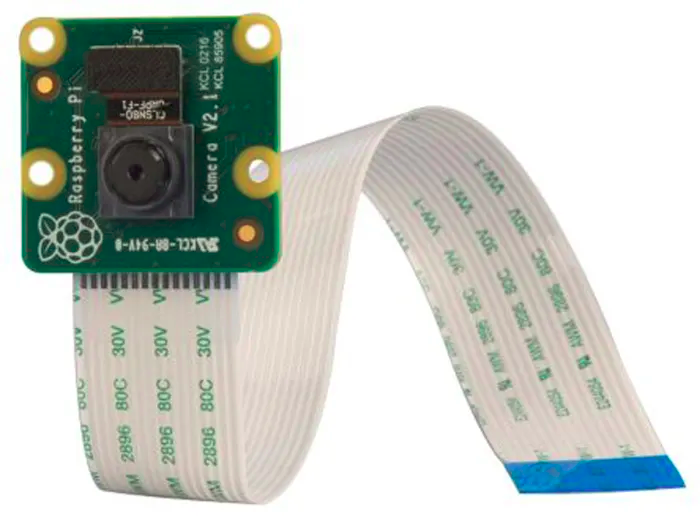


Imagen 1. Pi Camera V2.1

La información de la cámara se publica en el tópico /usb\_cam/image\_raw en formato BGR8 de ROS, esta información debe ser procesada usando CvBridge para cambiar el formato a numpy, es decir, un formato que pueda manipular OpenCV.

En el algoritmo de reconocimiento de objetos por color la imagen procesada se convierte en HSV (HUE, SATURATION, VALUE) para facilitar la detección de colores específicos. Es por esto que 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]

Esto facilita que en la imagen procesada con la máscara solo aparezcan los objetos que cumplen con esta característica. Posteriormente se encuentran los contornos de la imagen en la imagen binaria con cv2.findContours de OpenCV. Luego, con cv2.Moments se calculan los momentos espaciales del contorno, que son una serie de valores matemáticos que describen la forma y distribución del área encerrada, arrojando la ubicación X,Y del objeto en la imagen.

Si se define una resolución de 320x240 píxeles en la cámara del robot para una mejor velocidad de procesamiento, entonces cuando el objeto sea reconocido su centroide podría tomar valores en x de 0 hasta 319 y en Y de 0 hasta 239.

Posterior a esto, las coordenadas del centroide se publican en el tópico /objeto/Centroide.

**1.3 Agarre manual con Teleoperación por teclado**

Una vez identificado el objeto y determinadas sus coordenadas dentro del campo de visión (FOV) publicadas en el tópico /objeto/Centroide, el siguiente paso es teleoperar el robot de forma manual usando el teclado. este movimiento se puede realiza en el plano articular, es decir, el movimiento se daría articulación por articulación J1,J2,J3,J4. una letra del teclado aumentará o decrementará los grados de las Joint según corresponda Y en el plano cartesiano, es decir en X,Y,Z, cuya letra aumentará o decrementará la ubicación del efector final en el eje cartesiano.

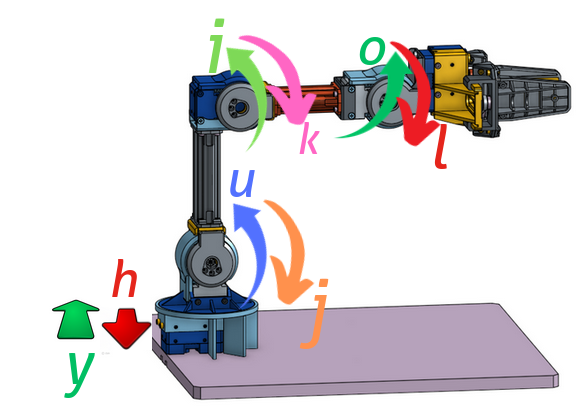


Imagen 2. TeleopKey en plano articular

# **2. PROCEDIMIENTO**

1. Para realizar nuestro paquete que reconozca objetos de un color específico ubicado en una posición definida para agarrarlo. Primero debemos crear nuestro paquete, lo nombramos tracking\_color , para ello usamos los siguientes comandos:

$ cd ~/catkin\_ws/src

$ catkin\_create\_pkg tracking\_color\_manual rospy std\_msgs

$ cd ~/catkin\_ws

$ catkin\_make



1. En nuestro paquete creamos los folders **src, scripts, launch** que contendrán los archivos necesarios. Para ello usa los siguientes comandos:

$ cd ~/catkin\_ws/src/tracking\_color\_manual

$ 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 ***posicion\_reconocimiento.py*** cuya finalidad será ubicar el robot Manipulador en la posición de reconocimiento. El script nombrado ***reconocimiento\_color.py*** será el encargado de realizar el reconocimiento del objeto del color específico. y finalmente el script **tracking\_color.py** el cual guiará el robot a las coordenadas en donde se ubique el objeto y lo agarrará.

$ cd ~/catkin\_ws/src/tracking\_color\_manual/scripts

$ nano posicion\_reconocimiento.py tracking\_color.py reconocimiento\_color.py

1. Ahora, agrega un código como este en el archivo posicion\_reconocimiento.py

#!/usr/bin/env python3

#SOLO LO LLEVA A LA POSICION DE RECONOCIMIENTO

import rospy

from std\_msgs.msg import Bool

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:

present\_kinematic\_position = None # Inicialización segura

def setTaskSpacePath(self, kinematics\_pose, path\_time):

"""

Envía una solicitud para mover el efector final en el espacio cartesiano.

:param kinematics\_pose: Lista con las coordenadas [x, y, z] de destino.

:param path\_time: Tiempo de ejecución.

:return: True si la planificación fue exitosa, False si falló.

"""

print("entra a la funcion de cartesiana")

try:

srv = SetKinematicsPoseRequest()

srv.end\_effector\_name = "gripper"

srv.kinematics\_pose.pose.position.x = kinematics\_pose[0]

srv.kinematics\_pose.pose.position.y = kinematics\_pose[1]

srv.kinematics\_pose.pose.position.z = kinematics\_pose[2]

print(f" si le llegan las coordenadas{kinematics\_pose}")

# Mantener la orientación actual del efector final

srv.kinematics\_pose.pose.orientation = self.kinematics\_pose.orientation

srv.path\_time = path\_time

response = self.goal\_task\_space\_path\_position\_only\_client(srv)

return response.is\_planned

except rospy.ServiceException as e:

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

return False

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

"""

Envía una solicitud para mover el manipulador en el espacio articular.

:param joint\_name: Lista con nombres de articulaciones.

:param joint\_angle: Lista con ángulos de destino.

:param path\_time: Tiempo de ejecución.

:return: True si el servicio fue exitoso, False si falló.

"""

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 kinematics\_pose\_callback(self, msg):

"""

Callback para recibir mensajes de KinematicsPose.

Extrae la posición y actualiza el estado.

"""

temp\_position = [msg.pose.position.x, msg.pose.position.y, msg.pose.position.z]

self.present\_kinematic\_position = temp\_position

self.kinematics\_pose = msg.pose

def getPresentKinematicsPose(self):

""" Retorna la posición cinemática actual. """

return self.present\_kinematic\_position

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):

"""

Controla la herramienta (gripper).

:param joint\_angle: Lista con la posición deseada del gripper.

:return: True si el servicio fue exitoso, False si falló.

"""

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):

"""

Función para manejar el clic del botón "Posición Inicial".

"""

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

path\_time = 2.0 # Tiempo de movimiento

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

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

joint\_angle=[0.01]

gripper\_closed = self.setToolControl(joint\_angle) #AQUI AGARRA EL OBJETO

if success ==True:

self.logrado = True

while True:

if not rospy.is\_shutdown():

self.publicador\_cumplimiento\_posicion\_reco.publish(self.logrado)

if not self.setToolControl(joint\_angle):

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

return

def pos\_reco(self):

kinematics\_pose = [0.23, 0.00, 0.0793]

print(f"{kinematics\_pose}")

path\_time=2.0

success1 = self.setTaskSpacePath(kinematics\_pose, path\_time)

def setActuatorState(self, actuator\_state):

"""

Activa o desactiva los actuadores del manipulador.

:param actuator\_state: True para habilitar, False para deshabilitar.

:return: True si fue exitoso, False si falló.

"""

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

self.process\_sim = None # Para el proceso de la simulación

self.process\_real = None # Para el proceso real

#Ejecución de movimiento del robot

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

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

self.kinematics\_pose\_subscriber = rospy.Subscriber('/gripper/kinematics\_pose', KinematicsPose, self.kinematics\_pose\_callback)

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

#Ejecución de movimiento del robot

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)

self.publicador\_cumplimiento\_posicion\_reco = rospy.Publisher('/cumplimiento\_posicion\_reco', Bool, queue\_size=10,latch=True)

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

"""

Callback para recibir mensajes de /joint\_states.

Extrae las posiciones de las articulaciones y actualiza el estado.

"""

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):

""" Retorna si el manipulador está en movimiento. """

return self.open\_manipulator\_is\_moving

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

""" Retorna si el actuador del manipulador está habilitado. """

return self.open\_manipulator\_actuator\_enabled

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

"""

Callback para recibir el estado del manipulador.

"""

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):

position = self.getPresentKinematicsPose()

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

print("No se hbilitaron los actuadores")

return

self.pos\_inicial()

def run(self):

""" Mantiene el nodo ROS en ejecución. """

rospy.spin()

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

try:

nodo = movimiento\_pos\_reconocimiento()

nodo.publicar\_mensaje()

except rospy.ROSInterruptException:

pass

1. Ahora, agrega un código como este en el archivo reconocimiento\_color.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 camera\_process:

global hsv\_inferior, hsv\_superior

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

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

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)

mask = cv2.inRange(hsv, hsv\_inferior, hsv\_superior)

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

for contour in contours:

cv2.drawContours(cv\_image, [contour], -1, (255, 255, 255), 13)

M = cv2.moments(contour)

cx,cy= None, None

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)

#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)

# Mostrar imagen procesada en cv\_imagen

imagen\_resized = cv2.resize(cv\_image, (600, 600)) # Redimensionar a 500x500 píxeles

cv2.imshow('Imagen con reconocimiento', imagen\_resized)

cv2.waitKey(1)

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

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

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

#Ejecución de movimiento del robot

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

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

self.publicador\_dimensiones = rospy.Publisher('/dimensiones\_imagen', Point, queue\_size=10)

try:

rospy.spin()

except KeyboardInterrupt:

print("Shutting down")

cv2.destroyAllWindows()

def run():

""" Mantiene el nodo ROS en ejecución. """

rospy.spin()

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

Process\_camera = camera\_process()

camera\_process.run()

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

$ cd ~/catkin\_ws/src/tracking\_color\_manual/launch

$ nano reco\_color\_manual.launch tracking\_pos\_final.launch

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

<launch>

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

<node name="nodo\_reconocimiento\_color" pkg="tracking\_color\_manual" type="reconocimiento\_color.py" output="screen"/>

</launch>

# 

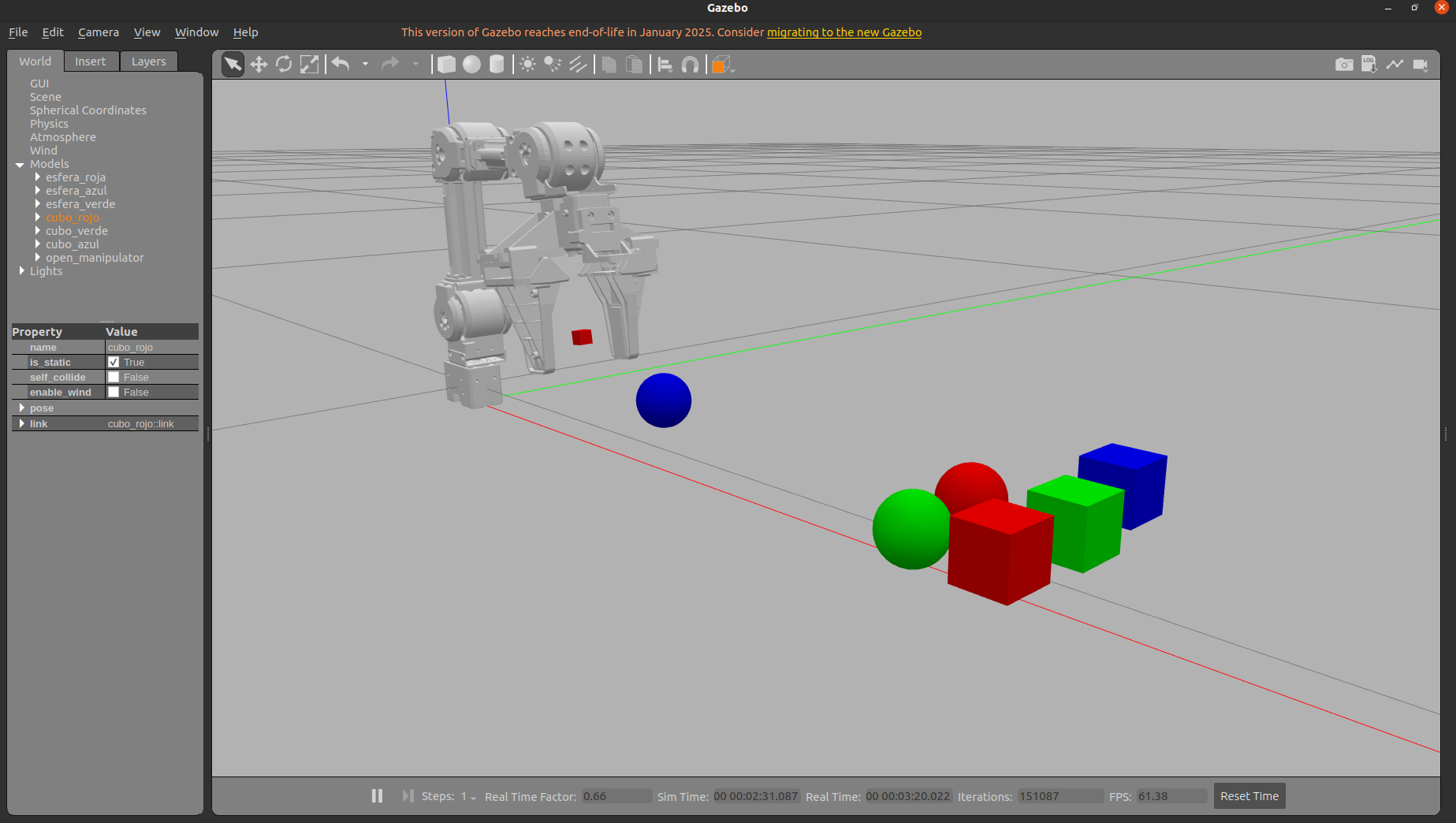
# **3. INICIALIZANDO EL ROBOT**

## **3.1 ROBOT EN 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

Verás lo siguiente:

¡LISTO, Ya está preparado el robot para atendernos!

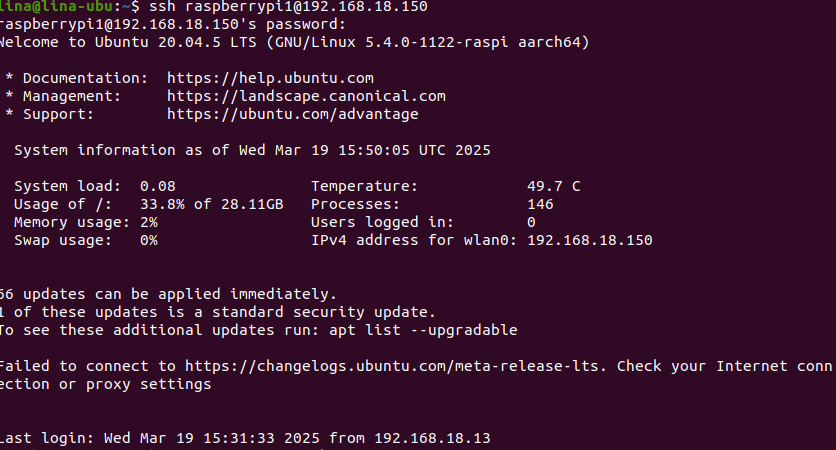
## **3.2 ROBOT REAL**

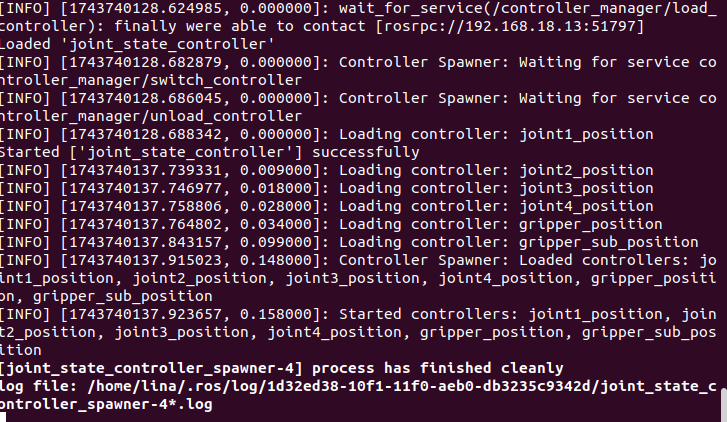
1. Si desea usar el robot real, primero debe conectarlo correctamente, asegúrese de conectar la potencia de la OpenCR y la alimentación de la RaspBerry.
2. Espere unos segundos mientras la raspberry se conecta a la red local. Luego intente conectarse vía ssh con el siguiente comando en la terminal de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

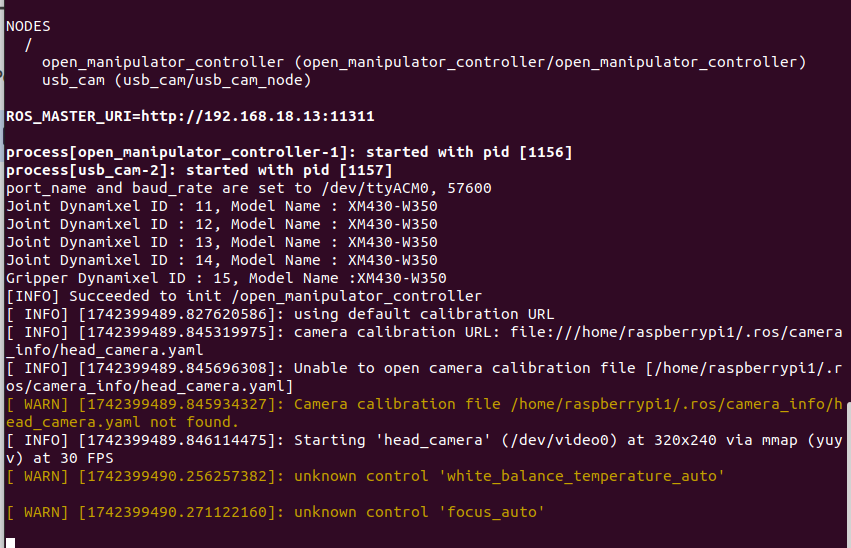
$ roscore



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:



¡LISTO, Ya está preparado el robot para atendernos!

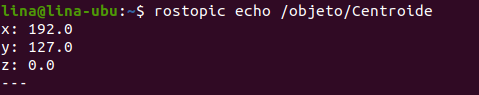
# **4. ¡A CORRER!**

Ahora, ya puedes lanzar el paquete creado. Primero lanzamos el archivo reco\_color\_manual.launch para llevar el robot a la posicion de reconocimiento y a ejecutarse el reconocimiento por el color indicado, que en este caso es el azul.

roslaunch tracking\_color\_manual reco\_color\_manual.launch

  
Podrás observar en el tópico la información de las coordenadas del objeto reconocido de color azul lanza este comando.

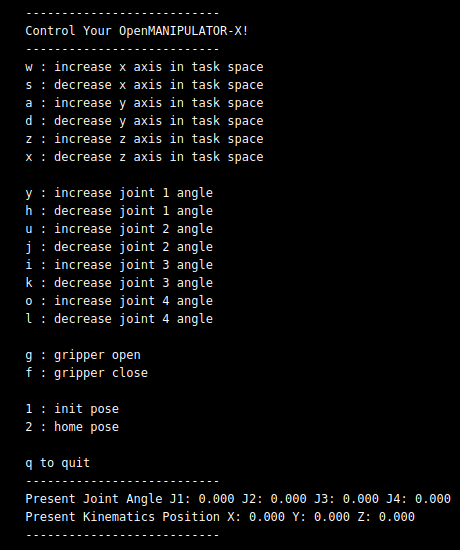
rostopic echo /objeto/Centroide



Como el objeto ya está reconocido, ya puedes empezar a teleoperador el robot para agarrarlo, para ello lanza el nodo de teleoperación por teclado siguiente:

roslaunch open\_manipulator\_teleop open\_manipulator\_teleop\_keyboard.launch

Verás esto en la terminal:



Cuando agarres el objeto verás algo como esto:

