Wood block volume measurement

1. Content Description

This lesson explains how to combine depth information and coordinate system transformation to calculate the volume of a wooden block.

This section requires entering commands in the terminal. The terminal you open depends on your motherboard type. This lesson uses the Raspberry Pi 5 as an example. For Raspberry Pi and Jetson-Nano boards, you need to open a terminal on the host computer and enter the command to enter the Docker container. Once inside the Docker container, enter the commands mentioned in this section in the terminal. For instructions on entering the Docker container from the host computer, refer to this product tutorial **[Configuration and Operation Guide]--[Enter the Docker (Jetson Nano and Raspberry Pi 5 users, see here)]**.

Simply open the terminal on the Orin motherboard and enter the commands mentioned in this section.

2. Program startup

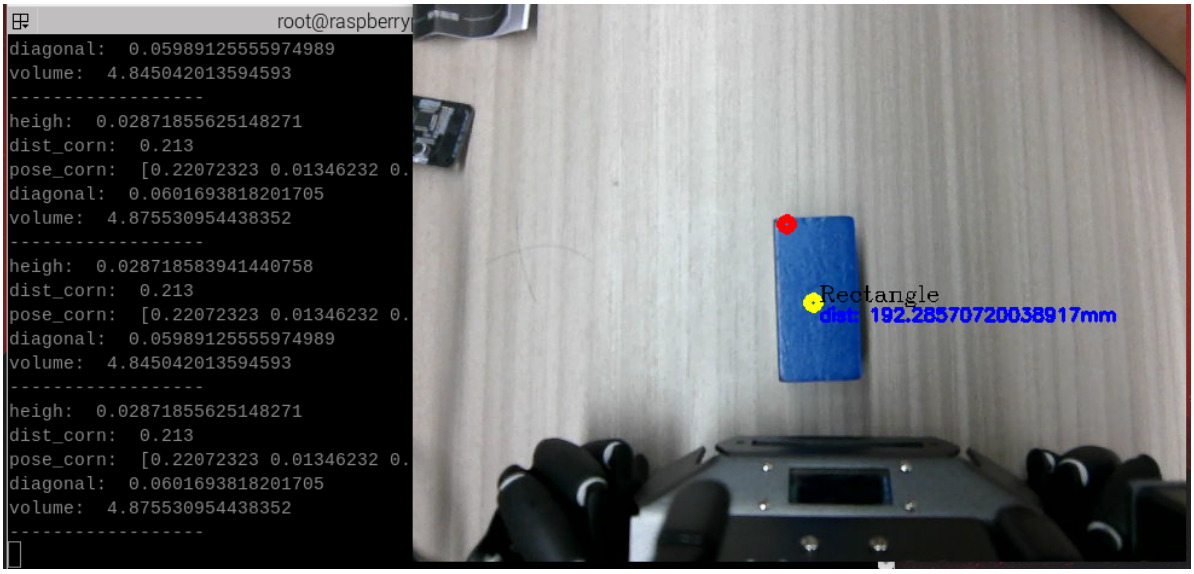
First, in the terminal, enter the following command to start the camera and robotic arm inverse solution program. The content of inverse solution will be introduced in this product course [9. Robotic Arm and 3D Space Gripping] - [Robotic Arm Inverse Solution]. Here you only need to understand that the inverse solution is to calculate the coordinate value.

```
ros2 launch M3Pro_demo camera_arm_kin.launch.py
```

After starting the camera and robotic arm inverse solution program, enter the following command to start the wood block volume measurement program. Enter the terminal,

```
ros2 run M3Pro_demo estimate_volume
```

After the startup is complete, we place a rectangular wooden block as close to the camera as possible (this is done to make the depth information as accurate as possible), then click on the image frame, and finally press the spacebar. The program will calculate the coordinates of the center point and the shape's vertices in the world coordinate system based on the recognized depth information and the posture of its own robotic arm. This value is generally represented by (x, y, z) in meters, indicating the distance from the base coordinate system base_link (0,0,0); then, combined with the recognized shape of the wooden block, the volume is calculated. As shown in the figure below, the recognized shape of the wooden block is a rectangular block, and the calculated volume is displayed on the terminal.



The dimensions of the cuboid are 2.8cm x 2.8cm x 5.8cm. The theoretical volume is 45.472 cubic centimeters. The calculated value is 4.87, which is within ±1 cubic centimeter. The error comes from the depth information and the virtual position of the servo.

3. Core code analysis

Program code path:

- Raspberry Pi 5 and Jetson-Nano board

  The program code is in the running docker. The path in docker is `/root/yahboomcar_ws/src/M3Pro_demo/M3Pro_demo/estimate_volume.py`

- Orin Motherboard

  The program code path is `/home/jetson/yahboomcar_ws/src/M3Pro_demo/M3Pro_demo/estimate_volume.py`

Import necessary library files

```
import cv2
import os
import numpy as np
from sensor_msgs.msg import Image, CameraInfo
import message_filters
from cv_bridge import CvBridge
import cv2 as cv
from arm_interface.srv import ArmKinemarics
from arm_interface.msg import AprilTagInfo,CurJoints
from arm_msgs.msg import ArmJoints
from std_msgs.msg import Float32,Bool,Int16
encoding = ['16UC1', '32FC1']
import time
import transforms3d as tfs
import tf_transformations as tf
import yaml
import math
from rclpy.node import Node
import rclpy
from message_filters import Subscriber,
TimeSynchronizer,ApproximateTimeSynchronizer
from sensor_msgs.msg import Image
```

Load the offset of the robot arm to compensate for the error caused by virtual position,

```
#The robotic arm offset parameter file address in the docker of pi and jetson-
nano motherboard: /root/yahboomcar_ws/src/arm_kin/param/offset_value.yaml
#Orin mainboard docker robot arm offset parameter file address:
/home/jetson/yahboomcar_ws/src/arm_kin/param/offset_value.yaml
offset_file = "/root/yahboomcar_ws/src/arm_kin/param/offset_value.yaml"
with open(offset_file, 'r') as file:
    offset_config = yaml.safe_load(file)
print(offset_config)
print("----------------------------")
print("x_offset: ",offset_config.get('x_offset'))
print("y_offset: ",offset_config.get('y_offset'))
print("z_offset: ",offset_config.get('z_offset'))
```

Subscribe to the depth and color image topics and synchronize the two topics. When the message times of the two topics are close, call the callback function to process the image message.

```
self.depth_image_sub = Subscriber(self, Image, '/camera/depth/image_raw')
self.rgb_image_sub = Subscriber(self, Image, '/camera/color/image_raw')
self.ts = ApproximateTimeSynchronizer([self.rgb_image_sub,
self.depth_image_sub], 1, 0.5)
self.ts.registerCallback(self.callback)
```

Create a publisher that publishes messages to control the six servos and a client that requests the inverse solution service.

```
self.pub_SixTargetAngle = self.create_publisher(ArmJoints, "arm6_joints", 10)
self.client = self.create_client(ArmKinemarics, 'get_kinemarics')
```

Control the robot arm to move to the recognized posture and calculate the current posture of the end of the robot arm.

```
self.pubSixArm(self.init_joints)
self.get_current_end_pos()
def get_current_end_pos(self):
    request = ArmKinemarics.Request()
    request.cur_joint1 = float(self.init_joints[0])
    request.cur_joint2 = float(self.init_joints[1])
    request.cur_joint3 = float(self.init_joints[2])
    request.cur_joint4 = float(self.init_joints[3])
    request.cur_joint5 = float(self.init_joints[4])
    request.kin_name = "fk"
    #Request to call the inverse solution fk service, that is, calculate the end
pose from the joint of each joint, and the incoming request is the joint value of
each servo of the current robot arm
    future = self.client.call_async(request)
    future.add_done_callback(self.get_fk_respone_callback)

def get_fk_respone_callback(self, future):
    try:
        response = future.result()
        #Get the end position of the robot arm, including xyz representing the
position and rpy representing the posture, and assign them to the self.CurEndPos
array
```

```
        #self.get_logger().info(f'Response received: {response.x}')
        self.CurEndPos[0] = response.x
        self.CurEndPos[1] = response.y
        self.CurEndPos[2] = response.z
        self.CurEndPos[3] = response.roll
        self.CurEndPos[4] = response.pitch
        self.CurEndPos[5] = response.yaw
        print("self.CurEndPose: ",self.CurEndPos)
    except Exception as e:
        self.get_logger().error(f'Service call failed: {e}')
```

In the image callback function, after a series of image processing, the shape of the wooden block is obtained. The content is as follows:

```
rgb_image = self.rgb_bridge.imgmsg_to_cv2(color_frame,'rgb8')
rgb_image = cv2.cvtColor(rgb_image, cv2.COLOR_RGB2BGR)
result_image = np.copy(rgb_image)
#depth_image
depth_image = self.depth_bridge.imgmsg_to_cv2(depth_frame, encoding[1])
frame = cv.resize(depth_image, (640, 480))
depth_to_color_image = cv2.applyColorMap(cv2.convertScaleAbs(depth_image,
alpha=1.0), cv2.COLORMAP_JET)
depth_image_info = frame.astype(np.float32)
#cv2.cvtColor converts the image from BGR to grayscale for subsequent image
processing
gray_image = cv2.cvtColor(depth_to_color_image, cv2.COLOR_BGR2GRAY)
#Create an all-zero array with exactly the same shape and data type as
gray_image
black_image = np.zeros_like(gray_image)
#Assign values to the array and copy the pixel values of the first 420 rows × the
first 640 columns of the grayscale image to the same position in black_image
black_image[0:420, 0:640] = gray_image[0:420, 0:640]
#Threshold black_image and set all pixels with values < 90 to 0 (pure black).
black_image[black_image < 90] = 0
cv2.circle(black_image, (320,240), 1, (255,255,255), 1)
#Gaussian filter to reduce image noise and details
gauss_image = cv2.GaussianBlur(black_image, (3, 3), 1)
#Convert the grayscale image to a binary image (black and white image)
_,threshold_img = cv2.threshold(gauss_image, 0, 255, cv2.THRESH_BINARY)
#Perform corrosion operation on the image to remove noise
erode_img = cv2.erode(threshold_img, np.ones((5, 5), np.uint8))
#Dilution operation on the image to connect the broken edges
dilate_img = cv2.dilate(erode_img, np.ones((5, 5), np.uint8))
#Find the contour according to the image and perform shape analysis. The returned
contours are a list of contour points.
contours, hierarchy = cv2.findContours(dilate_img, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
#Traverse each contour point list
for obj in contours:
    area = cv2.contourArea(obj)
    if area < 2000 :
        continue
    #Draw the contour function to visualize the detected object contours
    cv2.drawContours(depth_to_color_image, obj, -1, (255, 255, 0), 4)
    #Calculate the perimeter of the object
    perimeter = cv2.arcLength(obj, True)
    #Approximate the contour into a simpler polygon
```

```
    approx = cv2.approxPolyDP(obj, 0.035 * perimeter, True)
    self.corners = approx
    #print("self.corners: ",self.corners)
    cv2.drawContours(depth_to_color_image, approx, -1, (255, 0, 0), 4)
    # Calculate the number of edges
    CornerNum = len(approx)
    #print(CornerNum)
    x, y, w, h = cv2.boundingRect(approx)
    #4 sides indicate a rectangle, determine whether it is a rectangle or a
square
    if CornerNum == 4:
        side_lengths = []
        for i in range(4):
            p1 = approx[i]
            p2 = approx[(i + 1) % 4]
            side_lengths.append(np.linalg.norm(p1 - p2))  # Calculate the
distance between two points
        #Calculate the length of adjacent edges
        side_lengths = np.array(side_lengths)
#Judge the error of two adjacent sides. If the error is within the range, the
adjacent sides are considered equal, which is a square, otherwise it is a
rectangle
        if np.allclose(side_lengths[1], side_lengths[0], atol=50):  # Allow some
small errors
            objType = "Square"
        else:
            objType = "Rectangle"
    #According to actual needs, if the number of sides is greater than 5, it is
considered a circle, that is, the top of the cylinder
    elif CornerNum > 5:
        objType = "Cylinder"
    else:
        objType = "None"
```

Get the minimum enclosing rectangle of an outline of the shape and return the center point of a minimum area rectangle.

```
rect = cv2.minAreaRect(obj)
center = rect[0]
```

Calculate the pose of the center point, which represents the value of the point in the world coordinate system.

```
cx = int(center[0])
cy = int(center[1])
dist = depth_image_info[int(cy),int(cx)]/1000
pose = self.compute_heigh(cx,cy,dist)

def compute_heigh(self,x,y,z):
    #Conversion from plane to camera coordinate system
    camera_location = self.pixel_to_camera_depth((x,y),z)
    #print("camera_location: ",camera_location)
    #Conversion from camera coordinate system to end coordinate system
    PoseEndMat = np.matmul(self.EndToCamMat,
self.xyz_euler_to_mat(camera_location, (0, 0, 0)))
```

```
    #PoseEndMat = np.matmul(self.xyz_euler_to_mat(camera_location, (0, 0,
0)),self.EndToCamMat)
    #Get the end pose
    EndPointMat = self.get_end_point_mat()
    #Conversion from the end coordinate system to the world coordinate system
    WorldPose = np.matmul(EndPointMat, PoseEndMat)
    #Convert the 4*4 matrix representing the pose into translation and
orientation, where pose_T stores the xyz values, and the actual values also
include the offset of the robotic arm
    pose_T, pose_R = self.mat_to_xyz_euler(WorldPose)
    pose_T[0] = pose_T[0] + self.x_offset
    pose_T[1] = pose_T[1] + self.y_offset
    pose_T[2] = pose_T[2] + self.z_offset
    #print("pose_T: ",pose_T)
    #print("pose_R: ",pose_R)
    return pose_T
```

x value: the value of the point and base_link in the x-axis direction

y value: the value of the point and base_link in the y-axis direction

z value: The value of the point and base_link in the z-axis direction, which can be considered as the height.

If we can get the position of each point in the world coordinate system, we can solve the side length based on the identified shape. For example, for the simplest cube, we only need the z value to determine the volume. The side lengths of the cubes are the same, and the volume is the cube of the side length.

```
if objType == "Square":
    volume = pose[2]**3* 100000 # cubic centimeters
    print("volume: ",volume)
    print("------------------")
```

To calculate the volume of a cuboid, you need to first calculate the length of one side. The side length can be calculated by the distance between the two vertices. The distance calculation can be calculated by Euclid's theorem, using the coordinates of the two points to calculate the distance between the two points. Here, the distance between the vertex and the terminal is recommended, which is to get 1/2 of the diagonal length. The volume is calculated by the length of the diagonal and the height.

```
elif objType == "Rectangle":
    corn_x = self.corners[0][0][0] *1.01
    corn_y = self.corners[0][0][1] *1.01
    cv2.circle(rgb_image, (int(corn_x),int(corn_y)), 5, (0,0,255), 5)
    cv2.circle(rgb_image, (int(center[0]),int(center[1])), 5, (0,255,255), 5)
    dist_corn = depth_image_info[int(corn_y),int(corn_x)]/1000
    print("dist_corn: ",dist_corn)
    if dist_corn!=0:
        pose_corn = self.compute_heigh(corn_x,corn_y,dist_corn)
        print("pose_corn: ",pose_corn)
        diagonal = math.sqrt((pose[1] - pose_corn[1]) ** 2 + (pose[0] -
pose_corn[0])** 2) * 2 #diagonal meters
        print("diagonal: ",diagonal)
        volume = pose_corn[2]**2 * math.sqrt(diagonal**2 - pose_corn[2]**2)*
100000 # cubic centimeters
```

```
        print("volume: ",volume)
        print("-------------------")
```

The volume of a cylinder is calculated by calculating the distance between the vertex and the center point to obtain the radius of the circle at the top of the cylinder, and finally the volume is calculated using the formula for calculating the volume of the cylinder.

```
elif objType == "Cylinder":
    corn_x = self.corners[0][0][0] *1.01
    corn_y = self.corners[0][0][1] *1.01
    cv2.circle(rgb_image, (int(corn_x),int(corn_y)), 5, (0,0,255), 5)
    cv2.circle(rgb_image, (int(center[0]),int(center[1])), 5, (0,255,255), 5)
    dist_corn = depth_image_info[int(corn_y),int(corn_x)]/1000
    print("dist_corn: ",dist_corn)
    if dist_corn!=0:
        pose_corn = self.compute_heigh(corn_x,corn_y,dist_corn)
        print("pose_corn: ",pose_corn)
        radius = math.sqrt((pose[1] - pose_corn[1]) ** 2 + (pose[0] -
pose_corn[0])** 2)  #radius meters
        print("radius: ",radius)
        volume = pose_corn[2] * radius **2 *math.pi * 100000 #cubic centimeters
        print("volume: ",volume)
        print("-------------------")
```