CODE for project

Contents

[1.Camera extrinsic matrix node 1](#_Toc181981874)

[2.Tag detect and positioning node 6](#_Toc181981875)

[Link to video: 14](#_Toc181981876)

[Contribution table: 14](#_Toc181981877)

# 1.Camera extrinsic matrix node

This node calibrates the camera matrix to the end factor of the dobot recording the tag positions and manually recording the dobot arm positions when placed on the tag.

#!/usr/bin/env python

import rospy

import cv2

import numpy as np

import json

from sensor\_msgs.msg import Image, JointState

from geometry\_msgs.msg import PoseStamped

from cv\_bridge import CvBridge

import os

class ArucoMarkerDetectionNode:

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

rospy.init\_node('extrinsic\_calibration\_node', anonymous=True)

# Create a CvBridge for image conversion

self.bridge = CvBridge()

# Camera matrix and distortion coefficients

self.camera\_matrix = np.array([

[599.22323485, 0.0, 339.16246969],

[0.0, 600.35006132, 258.15884256],

[0.0, 0.0, 1.0]

])

self.dist\_coeffs = np.array([ 0.05088455 ,-0.13392238 , 0.0078341, -0.00211491, 0.0]) # Assuming no distortion based on the message

# Define the ArUco dictionary and parameters

self.aruco\_dict = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT\_APRILTAG\_36h11)

self.parameters = cv2.aruco.DetectorParameters()

# Subscribers for camera image, joint states, and end-effector pose

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

rospy.Subscriber('/dobot\_magician/joint\_states', JointState, self.joint\_state\_callback)

rospy.Subscriber('/dobot\_magician/end\_effector\_poses', PoseStamped, self.pose\_callback)

# Store positions and joint states

self.marker\_positions = []

self.robot\_poses = []

self.joint\_states = None

self.current\_pose = None

# File path to save the extrinsic matrix

self.extrinsic\_file\_path = os.path.expanduser('~/dobot\_ws/src/image\_detection\_calib/extrinsic\_matrix.json')

# Try loading existing extrinsic matrix

self.extrinsic\_matrix = self.load\_extrinsic\_matrix()

def load\_extrinsic\_matrix(self):

"""Load the extrinsic matrix from a file, or initialize a default one."""

if os.path.exists(self.extrinsic\_file\_path):

with open(self.extrinsic\_file\_path, 'r') as f:

data = json.load(f)

return np.array(data["extrinsic\_matrix"])

else:

rospy.loginfo("No saved extrinsic matrix found. Using default.")

return np.eye(4)

def save\_extrinsic\_matrix(self):

"""Save the current extrinsic matrix to a file."""

data = {"extrinsic\_matrix": self.extrinsic\_matrix.tolist()}

with open(self.extrinsic\_file\_path, 'w') as f:

json.dump(data, f)

rospy.loginfo("Saved the updated extrinsic matrix to file.")

def image\_callback(self, image\_msg):

try:

# Convert the ROS Image message to OpenCV format

frame = self.bridge.imgmsg\_to\_cv2(image\_msg, 'bgr8')

# Detect ArUco markers

corners, ids, \_ = cv2.aruco.detectMarkers(frame, self.aruco\_dict, parameters=self.parameters)

if ids is not None:

# Estimate the pose of the detected markers

rvecs, tvecs, \_ = cv2.aruco.estimatePoseSingleMarkers(corners, 0.031, self.camera\_matrix, self.dist\_coeffs)

# Log the position of each detected marker

for i, marker\_id in enumerate(ids):

marker\_position\_camera = tvecs[i][0]

rospy.loginfo(f"Marker ID {marker\_id[0]} Position (Camera Frame): {marker\_position\_camera}")

self.marker\_positions.append(marker\_position\_camera)

else:

rospy.logwarn("No markers detected")

except Exception as e:

rospy.logerr(f"Error processing image: {e}")

def joint\_state\_callback(self, joint\_state\_msg):

self.joint\_states = joint\_state\_msg.position

def pose\_callback(self, pose\_msg):

self.current\_pose = pose\_msg

def record\_position(self):

input("Press Enter to record the current ArUco position and robot pose...")

if self.marker\_positions:

latest\_marker\_position = self.marker\_positions[-1]

if self.current\_pose and self.joint\_states:

rospy.loginfo(f"Recording ArUco Position: {latest\_marker\_position}")

rospy.loginfo(f"Recording Robot Pose: {self.current\_pose.pose.position.x}, {self.current\_pose.pose.position.y}, {self.current\_pose.pose.position.z}")

rospy.loginfo(f"Recording Joint States: {self.joint\_states}")

self.robot\_poses.append((latest\_marker\_position, self.current\_pose.pose.position, self.joint\_states))

self.estimate\_extrinsic\_matrix()

else:

rospy.logwarn("No robot pose or joint states available")

else:

rospy.logwarn("No ArUco marker positions detected")

def estimate\_extrinsic\_matrix(self):

if len(self.robot\_poses) < 3:

rospy.logwarn("Not enough points collected to estimate extrinsic matrix")

return

# Separate the collected data into camera and robot points

camera\_points = np.array([pose[0] for pose in self.robot\_poses], dtype=np.float64)

robot\_points = np.array([[pose[1].x, pose[1].y, pose[1].z] for pose in self.robot\_poses], dtype=np.float64)

# Estimate the transformation using cv2.estimateAffine3D

retval, R = cv2.estimateAffine3D(camera\_points, robot\_points)[:2]

if retval:

extrinsic\_matrix = np.eye(4)

extrinsic\_matrix[:3, :3] = R[:, :3] # Extract the 3x3 rotation part

extrinsic\_matrix[:3, 3] = R[:, 3] # Extract the translation part

rospy.loginfo(f"Updated Extrinsic Matrix:\n{extrinsic\_matrix}")

else:

rospy.logwarn("Failed to estimate extrinsic matrix")

def start(self):

while not rospy.is\_shutdown():

self.record\_position()

if rospy.is\_shutdown():

break

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

try:

node = ArucoMarkerDetectionNode()

node.start()

except rospy.ROSInterruptException:

pass

# 2.Tag detect and positioning node

This node converts the detected tag image and sends it to the DoBot controller so the arm can position itself.

#!/usr/bin/env python3

import rospy

from sensor\_msgs.msg import Image

from geometry\_msgs.msg import Pose

from sensor\_msgs.msg import JointState

from cv\_bridge import CvBridge

import cv2

import numpy as np

import random

# Load the dictionary for the AprilTag 36h11 family

aruco\_dict = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT\_APRILTAG\_36h11)

# Initialize the detector parameters

parameters = cv2.aruco.DetectorParameters()

camera\_matrix = np.array([

[599.22323485, 0.0, 339.16246969],

[0.0, 600.35006132, 258.15884256],

[0.0, 0.0, 1.0]

])

# intrinsic and extrinsic matrix from kalibr self calibrated

# Intrinsics: via topic

# [606.8311157226562, 0.0, 333.31500244140625],

# [0.0, 606.0000610351562, 246.64346313476562],

# [0.0, 0.0, 1.0]

# via 'kalibr\_calibrate\_cameras'

# [599.22323485 600.35006132 339.16246969 258.15884256]

# Distortion:

# [ 0.05088455 -0.13392238 0.0078341 -0.00211491]

dist\_coeffs = np.array([ 0.05088455 ,-0.13392238 , 0.0078341, -0.00211491, 0.0]) # Assuming no distortion based on the message

extrinsic\_matrix = np.array([

[-1.20040889, 15.74843836, 16.63322791, -3.28564496],

[-0.36502823, 2.96074512, -0.45302241, 0.17169582],

[-1.50155796, -21.47453118, -23.52317082, 4.91828151],

[0.0, 0.0, 0.0, 1.0]

])

# ([

# [0.7071, 0, -0.7071, 0.43], # Rotation + Translation in X

# [0, 1, 0, 0.23], # Rotation + Translation in Y

# [0.7071, 0, 0.7071, 0.26], # Rotation + Translation in Z (26 cm height)

# [0, 0, 0, 1]

# ])

def compute\_correction\_matrix(published\_positions, actual\_positions):

P = np.hstack((published\_positions, np.ones((published\_positions.shape[0], 1))))

A = np.hstack((actual\_positions, np.ones((actual\_positions.shape[0], 1))))

centroid\_P = np.mean(P[:, :3], axis=0)

centroid\_A = np.mean(A[:, :3], axis=0)

P\_centered = P[:, :3] - centroid\_P

A\_centered = A[:, :3] - centroid\_A

H = np.dot(P\_centered.T, A\_centered)

U, S, Vt = np.linalg.svd(H)

R = np.dot(Vt.T, U.T)

if np.linalg.det(R) < 0:

Vt[2, :] \*= -1

R = np.dot(Vt.T, U.T)

t = centroid\_A - np.dot(R, centroid\_P)

correction\_matrix = np.identity(4)

correction\_matrix[:3, :3] = R

correction\_matrix[:3, 3] = t

return correction\_matrix

class ImageDetectionAndControl:

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

# Initialize ROS node

rospy.init\_node('image\_detection\_and\_control', anonymous=True)

# Create a CvBridge object for image conversion

self.bridge = CvBridge()

# Publisher for goal position to Dobot's end effector pose

self.goal\_pub = rospy.Publisher('/dobot\_magician/target\_end\_effector\_pose', Pose, queue\_size=10)

# Subscribe to the RGB image topic from RealSense

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

rospy.Subscriber('/doobot\_magician/joint\_states', JointState, self.joint\_state\_callback)

self.joint\_angles = [0, 0, 0, 0]

rospy.loginfo("Image Detection and Control Node Initialized")

self.correction\_matrix = np.identity(4)

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

# Map the joint names to indices for better flexibility

joint\_names = ["shoulder\_pan\_joint", "shoulder\_lift\_joint", "elbow\_joint", "wrist\_joint"]

name\_to\_index = {name: i for i, name in enumerate(joint\_names)}

# Extract joint angles based on the joint names in the message

for joint in msg.name:

if joint in name\_to\_index:

idx = name\_to\_index[joint]

self.joint\_angles[idx] = msg.position[idx]

def image\_callback(self, data):

try:

# Convert the ROS Image message to OpenCV format

cv\_image = self.bridge.imgmsg\_to\_cv2(data, 'bgr8')

# Detect ArUco markers

corners, ids, rejected = cv2.aruco.detectMarkers(cv\_image, aruco\_dict, parameters=parameters)

if ids is not None:

# Draw the detected markers

cv2.aruco.drawDetectedMarkers(cv\_image, corners, ids)

rospy.loginfo(f"Detected ArUco IDs: {ids.flatten()}")

# Transform the first detected marker's first corner to Dobot coordinates

for i, marker\_id in enumerate(ids):

# Pose estimation for each marker

rvecs, tvecs, \_ = cv2.aruco.estimatePoseSingleMarkers(corners, 0.031, camera\_matrix, dist\_coeffs)

marker\_position\_camera = tvecs[i][0] # Extracting the translation vector

rospy.loginfo(f"Marker ID {marker\_id[0]} Position (Camera Frame): {marker\_position\_camera}")

# Transform to Dobot coordinates

dobot\_coords = self.transform\_to\_dobot(marker\_position\_camera)

rospy.loginfo(f"Dobot coordinates for marker {marker\_id[0]}: {dobot\_coords}")

# Send goal to Dobot

self.send\_goal(dobot\_coords)

else:

rospy.logwarn("No ArUco markers detected")

except Exception as e:

rospy.logerr(f"Error processing image: {e}")

def dh\_transformation(self, theta, d, a, alpha):

"""Return the DH transformation matrix."""

return np.array([

[np.cos(theta), -np.sin(theta) \* np.cos(alpha), np.sin(theta) \* np.sin(alpha), a \* np.cos(theta)],

[np.sin(theta), np.cos(theta) \* np.cos(alpha), -np.cos(theta) \* np.sin(alpha), a \* np.sin(theta)],

[0, np.sin(alpha), np.cos(alpha), d],

[0, 0, 0, 1]

])

def forward\_kinematics(self, theta1, theta2, theta3, theta4):

"""Calculate the forward kinematics for Dobot Magician."""

# Define the DH parameters (lengths in meters)

L1, L2, L3, L4 = 0.103, 0.135, 0.160, 0.015 # in meters

# Create the transformation matrices using DH parameters

T0\_1 = self.dh\_transformation(theta1, d=L1, a=0, alpha=np.pi/2)

T1\_2 = self.dh\_transformation(theta2, d=0, a=L2, alpha=0)

T2\_3 = self.dh\_transformation(theta3, d=0, a=L3, alpha=0)

T3\_4 = self.dh\_transformation(theta4, d=0, a=0, alpha=0)

# Compute the overall transformation from base to end-effector

T\_base\_ee = T0\_1 @ T1\_2 @ T2\_3 @ T3\_4

return T\_base\_ee

def transform\_to\_dobot(self, camera\_coordinates):

"""Transform the camera coordinates to Dobot end-effector coordinates."""

# Step 1: Convert camera coordinates to homogeneous

camera\_coordinates\_homogeneous = np.append(camera\_coordinates, 1)

# Step 2: Apply extrinsic transformation from camera to robot base

base\_coordinates\_homogeneous = np.dot(extrinsic\_matrix, camera\_coordinates\_homogeneous)

# Step 3: Calculate the forward kinematics to get the end-effector pose

theta1, theta2, theta3, theta4 = self.joint\_angles

T\_base\_ee = self.forward\_kinematics(theta1, theta2, theta3, theta4)

# Step 4: Transform base coordinates to end-effector frame

end\_effector\_coordinates\_homogeneous = np.dot(T\_base\_ee, base\_coordinates\_homogeneous)

# Step 5: Extract the 3D coordinates

dobot\_coordinates = end\_effector\_coordinates\_homogeneous[:3]

return dobot\_coordinates

def normalize\_quaternion(self,x, y, z, w):

norm = np.sqrt(x\*\*2 + y\*\*2 + z\*\*2 + w\*\*2)

return x / norm, y / norm, z / norm, w / norm

def send\_goal(self, target\_coords):

goal = Pose()

goal.position.x, goal.position.y, goal.position.z = target\_coords

goal.position.z = -0.05

# Adapt the goal positions within the min and max range

goal.position.x = max(0.16, min(0.24, goal.position.x)) # Clamp x to be within 0.16 and 0.24

goal.position.y = max(-0.1, min(0.13, goal.position.y)) # Clamp y to be within -0.1 and 0.13

qx, qy, qz, qw = self.normalize\_quaternion(0, 0, 0, 1)

goal.orientation.x, goal.orientation.y, goal.orientation.z, goal.orientation.w = qx, qy, qz, qw

# goal.position.x = random.uniform(0.16, 0.24) # Min is 0.16 and max is 0.24

# goal.position.y = random.uniform(-0.1, 0.13) # Min is -0.1 and max is 0.13

# Publish the goal to move the Dobot's end effector

self.goal\_pub.publish(goal)

rospy.loginfo("Published goal to Dobot")

rospy.sleep(3)

def start(self):

rospy.spin()

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

try:

node = ImageDetectionAndControl()

node.start()

except rospy.ROSInterruptException:

pass

# Link to video:

<https://studentutsedu-my.sharepoint.com/personal/nazia_m_nayna_student_uts_edu_au/_layouts/15/stream.aspx?id=%2Fpersonal%2Fnazia%5Fm%5Fnayna%5Fstudent%5Futs%5Fedu%5Fau%2FDocuments%2FAttachments%2FProject%20Demonstration%2Emov&ct=1731068802805&or=OWA%2DNT%2DMail&cid=27583280%2D6ce8%2D6f2b%2D7a31%2Dc97f3b2f4f2f&ga=1&referrer=StreamWebApp%2EWeb&referrerScenario=AddressBarCopied%2Eview%2E97337988%2Ddfff%2D4b52%2D8b3c%2D4acf8f4be3df>

# Contribution table:

Camera calibration – Adesh, Nazia

Dobot interfacing – Adesh

ArUco Tag detection - Adesh, Nazia

Visual Servoing - Adesh

Report – Adesh, Nazia

Code and ROS package development – Adesh