BUMES Optimization and Vision

Vision Python Scripts Supporting Documentation

ME500 Advanced Manufacturing Final Project
Spring 2025
Wednesday AM Lab Group

Azamat Abdikarimov, Shrijit Banerjee, Aniruddha Dalal, Gaella Hawi, Kaleigh Hilal, Anirudh Sundaresan

Document Author: Gaella Hawi

Table of Content

Overview

This document contains descriptions of each .py file created for the vision system part of this project at Rosie's Station. For instructions on how these scripts are incorporated into BUMES, see document titled "ME500 Final Project WedsAM BUMES Guide." All .py files can be found within the shared folder.

opencv.py

This Python script captures live video from the webcam, defines a region of interest (ROI), and detects objects within that area using edge detection and contour analysis. For each detected object, it calculates the average color and classifies it as Blue, White, or Unknown, then displays the result on the video feed with visual indicators and RGB values. This script is designed to test and calibrate the camera and its output without actually running the robot. However, after using this script for calibrating, do not forget to change the values in the Vision_system.py.

```
import cv2
import numpy as np

def classify_color(bgr):
    b, g, r = bgr

    if b < 180 and g < 130 and r < 130:
        print("blue")
        return "Blue"
    elif r > 120 and g > 120 and b > 120:
        print("white")
        return "White"

    else:
        print("unknown")
        return "Unknown"


cap = cv2.VideoCapture(1, cv2.CAP_DSHOW)

x1, y1 = 140, 50
x2, y2 = 450, 360

while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame = cv2.resize(frame, (640, 480))
    roi = frame[y1:y2, x1:x2]

    # Convert to grayscale + blur
    gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)

    # Edge detection (no color masking)
    edges = cv2.Canny(blurred, 30, 100)

    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

```
    found = False

    for cnt in contours:
        area = cv2.contourArea(cnt)
        if area > 1000:
            approx = cv2.approxPolyDP(cnt, 0.02 * cv2.arcLength(cnt,
True), True)
            if len(approx) >= 4:
                x, y, w, h = cv2.boundingRect(approx)
                cx, cy = x + w // 2, y + h // 2

                # Extract region and get average color
                obj_roi = roi[y:y+h, x:x+w]
                b, g, r = cv2.mean(obj_roi)[:3]
                color_label = classify_color((b, g, r))
                print(color_label)

                # Format RGB values
                rgb_text = f"R:{int(r)} G:{int(g)} B:{int(b)}"

                # Draw stuff
                cv2.drawContours(roi, [approx], -1, (0, 255, 0), 2)
                cv2.circle(roi, (cx, cy), 5, (0, 0, 255), -1)

                # Display color + RGB
                cv2.putText(roi, f"{color_label} ({cx},{cy})", (x, y -
25),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 0),
2)
                cv2.putText(roi, rgb_text, (x, y - 5),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255,
255), 2)

                print(f"Detected {color_label} object at ({cx}, {cy}) with
RGB {rgb_text}")
                found = True

    if not found:
        print("No object detected")

    cv2.imshow("ROI", roi)
    cv2.imshow("Edges", edges)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

_mesProcess.py

This code is part of a Manufacturing Execution System (MES) that handles automated task scheduling and tracking in an assembly line using Python, MySQL, and MQTT communication. It defines all the functions that execute the BUMES process. This code is part of the original BUMES Python script and must not be deleted; students should exercise extreme caution when making any additions or modifications to the original Git repository. It is needed to create a new branch of the original repository and add additional functions and changes. The addition that is required to this code is the following:

```
def Vision_test(self, mqttTopic):
        processInfo = mqttTopic.split('/')
        task = processInfo[1]
        taskID = processInfo[0]

        print('Starting Vision_test')

        exec(open('_mesVision_test.py').read())

        update_task_complete_process_query = "UPDATE process_handler SET
task_complete = True, end_time = " + \
            str(time.time()) + " WHERE task_name = '" + task + "' AND
process_name = '" + self.processName + "' AND operation_name = '" +
self.operationName + "'"
        self.cursor.execute(update_task_complete_process_query)
        self.connection.commit()

        '''update_task_complete_process_query = "UPDATE process_handler
SET task_complete = True, end_time = " + \
            str(time.time()) + " WHERE id = " + str(taskID)
        self.cursor.execute(update_task_complete_process_query)
        self.connection.commit()'''

        print('Vision_test completed')
```

The Vision_test function is defined in this script and runs it by executing the_ mesVision_test.py script, and updates the task status to "complete" in the database. It also records the end time of the task and commits the changes to the database.

_mesVision_test.py

This code defines a function **runPythonScript** that takes a filename as an argument, constructs a command to execute the script using **python.exe**, and then runs that script using **os.system()**. In this case, the function is called with the file path 'C:\git\ADML\Vision_system.py', which executes the **Vision_system.py** Python script.

```
"""
Script executed with BUMES command: --- "functionalPrinting()" ---
"""

import os

def runPythonScript(filename):
    """
    Finds and executes the python script with given filename
    """

    command = 'python.exe "' + str(filename) + '"'
    print(command)
    os.system(command)


runPythonScript('C:\git\ADML\Vision_system.py')
```

Vision_system.py

This script runs the vision process to retrieve the finished cord organizer from the conveyor belt for inspection. It uses a camera to capture the image, detect contours, and classify the cord organizer's color as either "White" or "Blue." Depending on the detected color, the robot moves to specific positions (two different bins) to place the cord organizer, controlling the gripper to pick up and release the object at each stage. Poses that are defined are done manually. In order to get them, the robot has to be placed on the needed position and then the values x,y,z,rx,ry,rz have to be copy pasted from the Polyscope interface into the python (manually).

```
import rtde_control
import rtde_receive
import rtde_io
import time
import cv2
import numpy as np

z = 0.4

# Connect to robot
rtde_c = rtde_control.RTDEControlInterface("10.241.34.45")
rtde_r = rtde_receive.RTDEReceiveInterface("10.241.34.45")
rtde_io_ = rtde_io.RTDEIOInterface("10.241.34.45")

# Define poses
p_home = [0.10914, -0.48692, 0.03133+z, 0.0, 3.142, 0.0]
p_home_to_pallet =  [0.3219,0.38565,-0.20525+z,3.126,0.008,-0.008]
p_home_to_pallet_down =  [0.3219,0.38565,-0.25174+z,3.126,0.008,-0.008]
pallet_to_cam = [0.487,-0.0833,-0.07655+z,2.228,-2.203,-0.021]
p_home_cam = [0.8007,-0.0669,-0.355+z,2.198,-2.258,-0.005]
p_home_abvcam = [0.8007,-0.0669,-0.285+z,2.198,-2.258,-0.005]



away_from_cam =[0.59510,-0.09503,-0.19215+z,2.193,-2.145,-0.046]

pick_up =[0.8007,-0.0669,-0.3658+z,2.198,-2.258,-0.005]

white_position =[0.18706,-0.51690,-0.2807+z,2.249,2.204,0.037]

blue_position =[0.29910,-0.49650,-0.2807+z,2.242,2.148,-0.035]

def classify_color(bgr):
```

```
    b, g, r = bgr
    if b < 180 and g < 120 and r < 130:
        return "Blue"
    elif r > 120 and g > 120 and b > 120:
        return "White"
    else:
        return "Unknown"

def get_object_color():
    cap = cv2.VideoCapture(1, cv2.CAP_DSHOW)
    x1, y1 = 140, 50
    x2, y2 = 450, 360

    for _ in range(30):  # Let camera adjust
        ret, frame = cap.read()

    ret, frame = cap.read()
    cap.release()

    if not ret:
        return "Unknown"

    frame = cv2.resize(frame, (640, 480))
    roi = frame[y1:y2, x1:x2]

    gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    edges = cv2.Canny(blurred, 30, 100)

    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    for cnt in contours:
        area = cv2.contourArea(cnt)
        if area > 1000:
            x, y, w, h = cv2.boundingRect(cnt)
            obj_roi = roi[y:y+h, x:x+w]
            b, g, r = cv2.mean(obj_roi)[:3]
            return classify_color((b, g, r))

    return "Unknown"

rtde_io_.setStandardDigitalOut(0, True)
print("Gripper closed")
time.sleep(1)

# Move to p_home
```

```
q_home = rtde_c.getInverseKinematics(p_home)
rtde_c.moveJ(q_home, speed=0.3)
time.sleep(1)


q_next = rtde_c.getInverseKinematics(p_home_to_pallet)
rtde_c.moveJ(q_next, speed=0.3)
time.sleep(1)



rtde_io_.setStandardDigitalOut(0, False)
print("Gripper opened")
time.sleep(1)


q_next = rtde_c.getInverseKinematics(p_home_to_pallet_down)
rtde_c.moveJ(q_next, speed=0.3)
time.sleep(1)

rtde_io_.setStandardDigitalOut(0, True)
print("Gripper closed")
time.sleep(1)

q_next = rtde_c.getInverseKinematics(p_home_to_pallet)
rtde_c.moveJ(q_next, speed=0.3)
time.sleep(1)

q_next = rtde_c.getInverseKinematics(pallet_to_cam)
rtde_c.moveJ(q_next, speed=0.3)
time.sleep(1)




q_home = rtde_c.getInverseKinematics(p_home_cam)
rtde_c.moveJ(q_home, speed=0.3)
time.sleep(1)

rtde_io_.setStandardDigitalOut(0, False)
print("Gripper opened")
time.sleep(1)

q_home = rtde_c.getInverseKinematics(p_home_abvcam)
rtde_c.moveJ(q_home, speed=0.3)
time.sleep(1)
```

```
q_home = rtde_c.getInverseKinematics(away_from_cam)
rtde_c.moveJ(q_home, speed=0.3)
time.sleep(1)

color = get_object_color()
print(f"Detected color: {color}")

q_home = rtde_c.getInverseKinematics(p_home_abvcam)
rtde_c.moveJ(q_home, speed=0.3)
time.sleep(1)

q_home = rtde_c.getInverseKinematics(pick_up)
rtde_c.moveJ(q_home, speed=0.3)
time.sleep(1)

rtde_io_.setStandardDigitalOut(0, True)
print("Gripper closed")
time.sleep(1)

q_home = rtde_c.getInverseKinematics(p_home)
rtde_c.moveJ(q_home, speed=0.3)
time.sleep(1)

if color == "White":
    q_home = rtde_c.getInverseKinematics(white_position)
    rtde_c.moveJ(q_home, speed=0.3)
    time.sleep(1)
    rtde_io_.setStandardDigitalOut(0, False)
    print("Gripper opened")
    time.sleep(1)
elif color == "Blue":
    q_home = rtde_c.getInverseKinematics(blue_position)
    rtde_c.moveJ(q_home, speed=0.3)
    time.sleep(1)
    rtde_io_.setStandardDigitalOut(0, False)
    print("Gripper opened")
    time.sleep(1)
else:
    print("Unrecognized color.")

q_home = rtde_c.getInverseKinematics(p_home)
rtde_c.moveJ(q_home, speed=0.3)
time.sleep(1)
```

```
rtde_io_.setStandardDigitalOut(0, True)
print("Gripper closed")
time.sleep(1)




# Stop control script
rtde_c.stopScript()
```