UNIVERSITY OF CYPRUS

Department of Computer Science

Applications of Robotic Arm

Ieronymos Maxoutis

21/06/2021 - 30/07/2021

1. Summary

During my internship at LInC I had the opportunity to work on a Dobot Magician robotic arm and a JeVois smart machine vision camera. Using the JeVois camera my task was to design a machine vision module that could be used to identify the colour and shape of some boxes. I developed the module based on red, green and blue boxes but it can be easily modified to detect accurately combinations of those colours such as yellow, cyan and magenta. The Dobot Magician arm together with a conveyor belt, would then be used to move and sort the boxes based on their colour. In order to do so, I attached a suction cup at the tip of the robotic arm that could be used to lift and move the boxes. Difficulties arose while developing the machine vision module since image noise and background caused many errors and difficulties. Mechanical limitations of the robotic arm were also an issue since it required calibration often. The programming language that I used to program the robotic arm and the JeVois camera was Python. The main libraries used were OpenCV, pySerial and threading.

2. Design

The main design idea was to pick one box at a time from a matrix consisting of 9 randomly arranged boxes as shown in figure 1 below. The box would then be placed on the conveyor belt start position (just above the matrix) and the conveyor belt would move it closer to the JeVois camera (upper left corner) so that its colour could be detected. The JeVois camera would then send a serial message to the edge device with information about the colour of the box, that could then be used by the robotic arm. This process is repeated until all 9 boxes are sorted based on their colour.

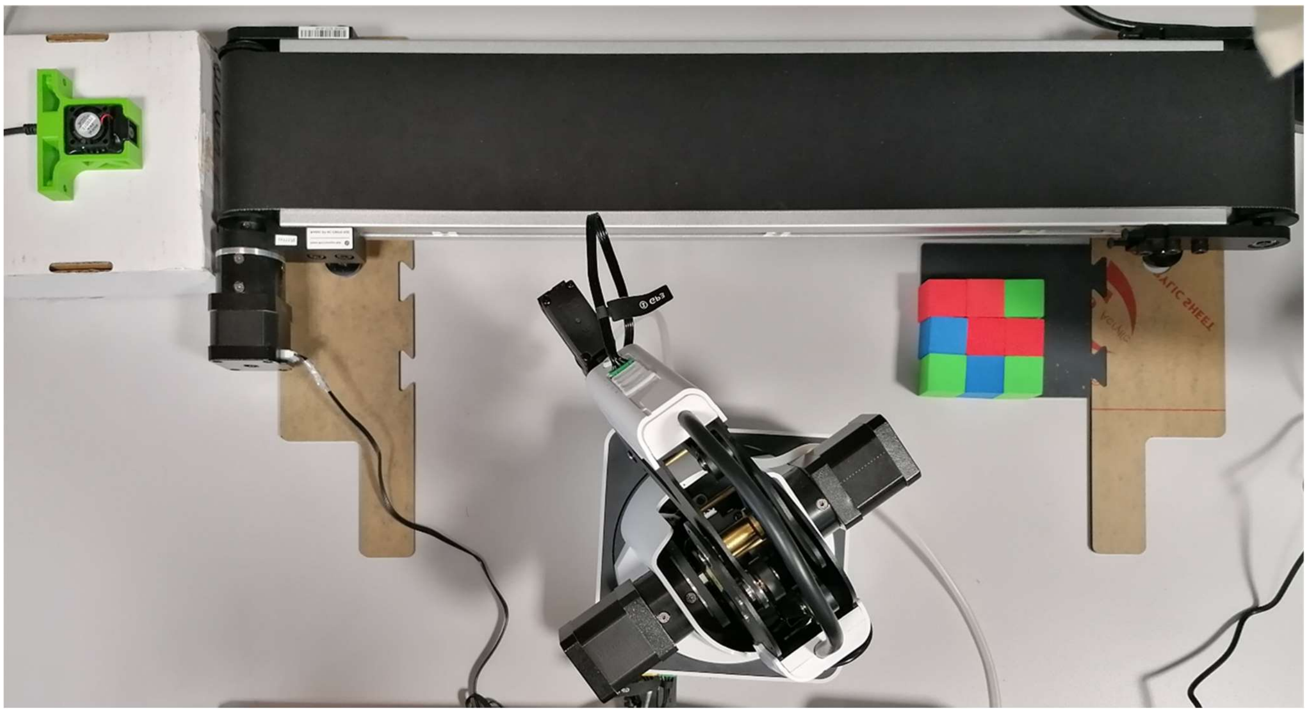


Figure 1 – Dobot and JeVois Camera setup

3.1 JeVois Camera

To control the camera with a python script I needed to install the JeVois Inventor app. This allowed me to create a machine vision module, using the OpenCV library, which was used to detect the colour of the boxes. Image noise and background were the main problems that caused difficulties with the video analysis. To remove most of the background the image was resized as shown by the yellow rectangle on figure 2. I then smoothed the resized image to remove most of the noise. The smoothed image was then used to identify the colour and the shape of the box (height and width). Edge detection was done using a binary mask of the image in order to get the dimensions of the box, as shown on figure 3. For colour detection, I extracted a small 5x5 pixel patch from the smoothed image, as shown by the white ring on figure 2. This was done to increase the accuracy of the colour detection. Overall, the edge detection algorithm was less accurate than the colour detection since it is more sensitive to noise. This is supported by the fact that the box is not reconstructed perfectly on figure 3. The JeVois camera then sends a serial message to the edge device with information about the colour detected which is then used by the Dobot.

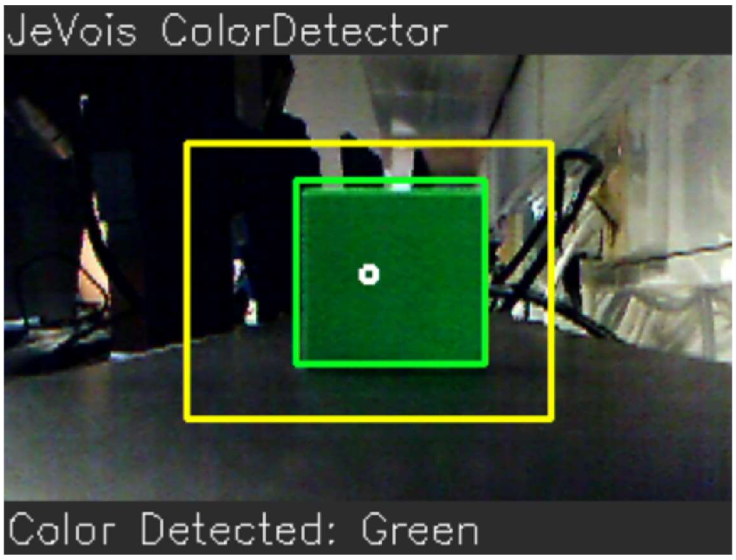


Figure 2 – Colour Detector Output

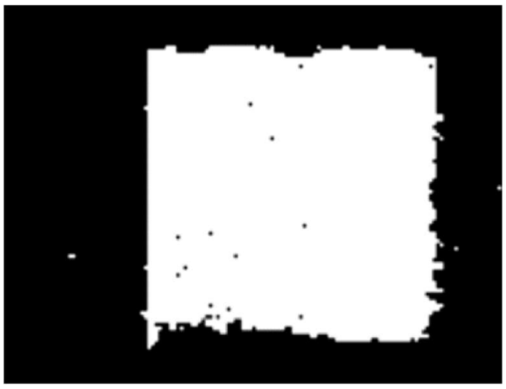


Figure 3 – Edge Detector Output

3.2 Dobot Magician Robotic Arm

The robotic arm was controlled using several functions. The *pick_and_place()* function was used to move a box from the matrix to the conveyor belt starting position. When called, this function updates the location of the next box. The *move_and_sort()* function is then used to move the arm and the conveyor belt concurrently, using the python threading module. When the conveyor belt stops the JeVois sends a serial message with information about the colour of that box. The colour is then passed to the *Sorting()* function, which separates the boxes based on their colour. The parameters required for the program are initialised by the INITIALIZE_PickAndPlace() and INITIALIZE_Sorting(). Documentation about the Dobot's commands can be found on Dobot Studio application or on the web. All the necessary methods to control the Dobot are found in the *DobotDllTypeX.py* file.

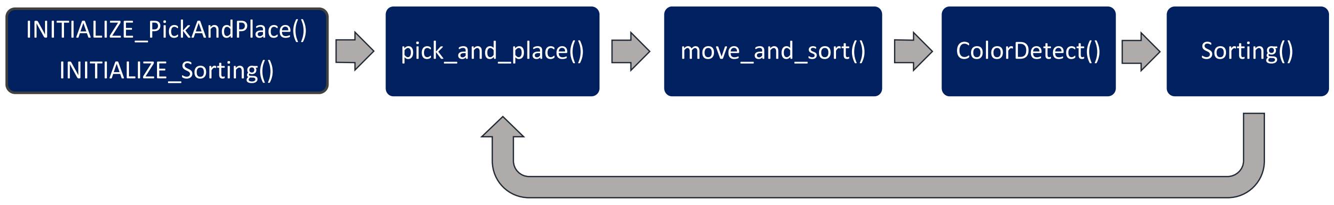


Figure 4 – Flowchart of the main program functions

4. Problems and Difficulties

JeVois camera

The JeVois camera is very sensitive to lighting changes, such as brightness and contrast, which create noise in the video. For this reason, make sure that the room is bright and set the camera parameters of the JeVois Inventor application manually, instead of automatically, as shown in the *initscript* configuration file. The colour detector module works best using these parameters and I would suggest to not changing them. Another problem with the JeVois camera is overheating. After approximately 1 hour of operation the fans do not cool the camera sufficiently and thus it becomes slow and inaccurate. I would recommend to not cover the camera and to disconnect it when not in use.

Dobot Arm

One problem with the robotic arm is the time delay between the execution of some python methods and the real-time response of the robotic arm. One such method is the *SetEndEffectorSuctionCupEx()*, which sets the suction cup on or off. When called in the python script, it is executed too quickly and as a result the Dobot arm sometimes does not lift the boxes, or it releases too soon while moving. This problem can be easily solved by adding a *sleep()* method before and after the Se*tEndEffectorSuctionCupEx()* method. Another problem with the Dobot arm is that it loses its calibration easily. This means that you have to re-calibrate the initialisation parameters all over again which is very time consuming. One way to avoid this is to perform the "homing" action once on the Dobot Studio application before connecting it with another python IDE. The *"home"* position acts as a reference point for both the cartesian and the joint coordinate system which are used by the Dobot to traverse in space. Regularly repeat the *"homing"* process to refresh the reference point and prevent the Dobot from losing its calibration.

4. Measurements and Results

Threading library was used by the *move_and_sort()* algorithm in order to decrease the total time needed by the program to sort all 9 boxes, which is indicated by the dark blue columns on the plot below. This time was also checked using a stopwatch which gave approximately the same result as the one calculated by the program. The light blue columns on the top are calculated by finding the time needed to run each function in the program separately and then subtracting it from the total time mentioned above. Moreover, increasing the arm's speed decreases the total execution time.

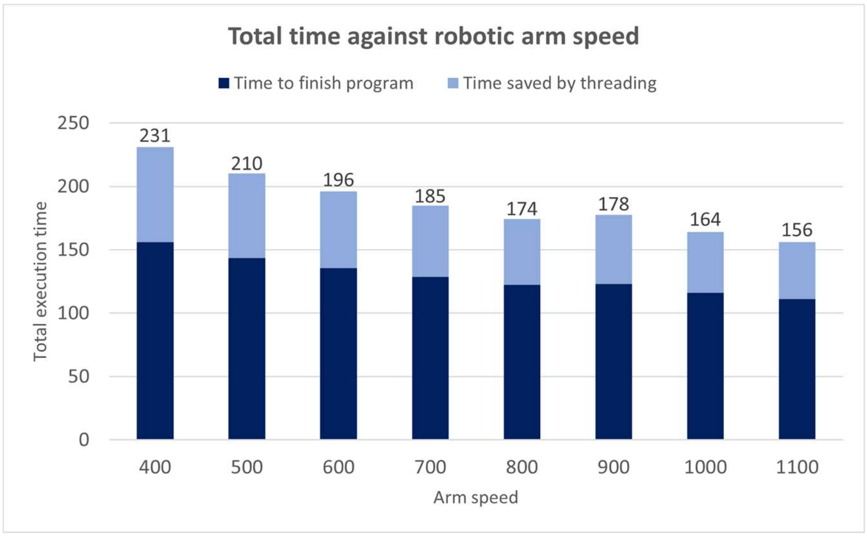
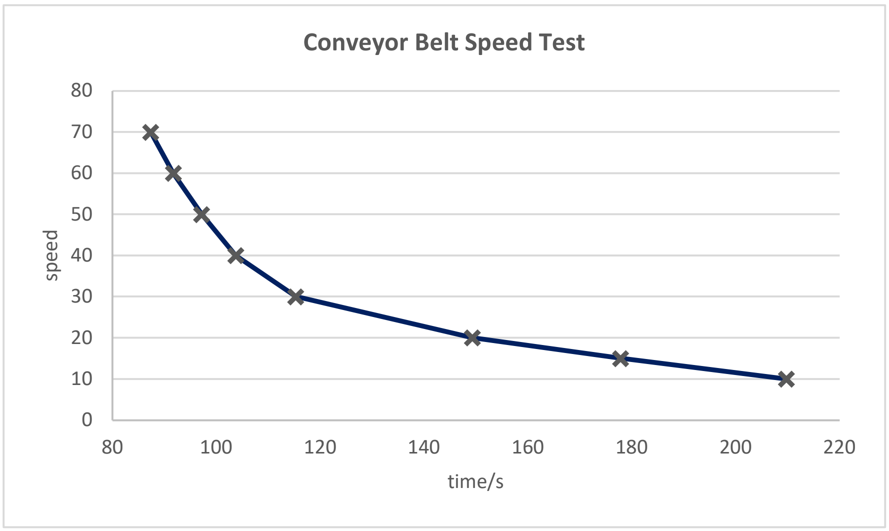


Figure 5 – Plot of program's execution time against arm speed

Besides changing the speed of the robotic arm, I have also tested the total time taken by the *move_and_sort()* function at different conveyor belt speeds. As expected, decreasing the belt's speed increases the execution time. The maximum safe speed that the belt can work is 70 mm/s. It is important to note that changing the speed of the belt didn't affect the accuracy of the JeVois camera. This is because the *ColorDetect()* function is designed to work only when the box is at rest in front of the camera.



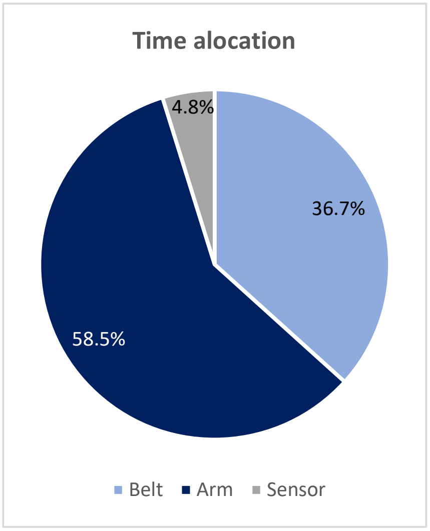


Figure 6 & 7 – Plots of belt's speed vs execution time and the program's time allocation

Appendix

Main algorithm

```
def main():
    global total_time_belt, total_time_arm, total_time

    CON_STR = {
        dTypeX.DobotConnect.DobotConnect_NoError: "DobotConnect_NoError",
        dTypeX.DobotConnect.DobotConnect_NotFound: "DobotConnect_NotFound",
        dTypeX.DobotConnect.DobotConnect_Occupied: "DobotConnect_Occupied"}

    api = dTypeX.load()
    state = dTypeX.ConnectDobot(api, 'COM3', 115200)
    print("Connect status:", CON_STR[state[0]])

    INITIALIZE_PickandPlace(api)
    INITIALIZE_Sorting()

    switch = True
    i = 0

    start = time.perf_counter()

    while switch:
        pick_and_place(api)
        total_time_belt += move_and_sort(api)
        i += 1
        if i == 9:
            print('Dobot Finished')
            print(f'Red count: {red_count}')
            print(f'Blue count: {blue_count}')
            print(f'Green count: {green_count}')
            print(f'Calibrations needed: {calibration}')
            switch = False
            dTypeX.SetPTPCmdEx(api, 2, HomeX, HomeY, HomeZ, 0, 1)
            dTypeX.DisconnectAll()

    finish = time.perf_counter()
    t = finish - start
    total = total_time_arm + total_time_belt + total_time_colour

    print(f'Actual time: {round(t, 2)} seconds at arm speed {arm_speed} and belt
speed {belt_speed}')
    print(f'Elapsed time: {round(total, 2)} seconds at arm speed {arm_speed} and
belt speed {belt_speed}')
    print(f'Time saved due to threading: {round(total-t, 2)} seconds ')
    print(f'Total time in move_and_sort: {round(exec_time_sort, 2)} seconds ')
    print(f'Total time to move belt: {round(total_time_belt, 2)} seconds
({round(100*(total_time_belt)/total, 2)}%)')
    print(f'Total time to move arm: {round(total_time_arm, 2)} seconds
({round(100*(total_time_arm)/total,2)}%)')
    print(f'Total time in colour sensor: {round(total_time_colour, 2)} seconds
({round(100 * (total_time_colour) / total,2)}%)')
    print(f'Total time delays: {round(time_slept, 2)}')
```

Colour Detector algorithm

```
def ColorDetect():
    '''Function used to read the serial message from the JeVois camera. A colour
buffer is filled based on the times each colour red, green or blue is detected.
The most frequent colour is then returned'''
    global total_time_colour

    serdev = 'COM5'  # serial device of JeVois 1

    start = time.perf_counter()
    dict = {'Red': 0, 'Green': 0, 'Blue': 0}

    with serial.Serial(serdev, 115200, timeout=1) as ser:
        while 1:
            # Read a whole line and strip any trailing line ending character:
            line = ser.readline().rstrip().decode()
            if line == 'Blue' or line == 'Green' or line == 'Red':
                dict[line] += 1  # add one to the value of the dictionary
associated with that colour

            if dict['Red'] == 40 or dict['Green'] == 40 or dict['Blue'] == 40:
                decision = max(dict.items(), key=operator.itemgetter(1))[0]
                all_values = dict.values()
                max_value = max(all_values)
                print(f'Colour detected: {decision}')
                print(f'Times detected: {max_value}')
                finish = time.perf_counter()
                total = finish - start
                total_time_colour += total
                return decision

            finish = time.perf_counter()
            timer = round(finish - start, 2)  # get the total time taken by the
function if no colour is detected

            if timer >= 6:  # protect against infinite loops
                print(f'Time elapsed in colour sensor: {timer} second(s)')
                print('No color detected...')
                return None
```

Pick and Place algorithm

```
def pick_and_place(api):
    global PickX, PickY, PickZ, StartX, StartY, StartZ, EndX, EndY, EndZ, HomeX,
HomeY, HomeZ, total_time_arm

    start = time.perf_counter()

    cubes = 9
    # pick first cube
    dTypeX.SetPTPCmdEx(api, 2, PickX, PickY, PickZ+10, 0, 1)
    dTypeX.SetPTPCmdEx(api, 2, PickX, PickY, PickZ, 0, 1)
    time.sleep(0.1)
    dTypeX.SetEndEffectorSuctionCupEx(api, 1, 1)
    time.sleep(0.1)
    dTypeX.SetPTPCmdEx(api, 2, PickX, PickY, StartZ+15, 0, 1)
    dTypeX.SetPTPCmdEx(api, 2, StartX, StartY, StartZ, 0, 1)  # start position of
block
    time.sleep(0.1)
    dTypeX.SetEndEffectorSuctionCupEx(api, 0, 1)
    time.sleep(0.1)
    dTypeX.SetPTPCmdEx(api, 2, StartX, StartY, StartZ+10, 0, 1)
    if cubes > 1:
        updateX(PickX, PickY)  # get the new values of the x-axis cube
        cubes -= 1

    finish = time.perf_counter()
    total = finish - start
    total_time_arm += total
```

Move and Sort algorithm

```
def move_and_sort(api):
    global HomeX, HomeY, HomeZ, RedX, RedY, RedZ, GreenX, GreenY, GreenZ, BlueX,
BlueY, BlueZ, \
        red_count, blue_count, green_count, threading_time, exec_time_sort,
time_slept

    start = time.perf_counter()

    t1 = Thread(target=move_belt, args=[api])
    t2 = Thread(target=move_arm, args=[api])

    t1.start()
    t2.start()

    if True:
        time.sleep(threading_time)  # threading time is the optimised time after
the first cube is sorted.
        threading_time = 0
        threading_time = sleep + 0.1  # get the optimised threading time to
minimise the time wasted
        time_slept += threading_time
        print('JeVois Starting...')
        Sorting(api, ColorDetect())

    t1.join()
    t2.join()

    finish = time.perf_counter()
    exec_time = finish - start
    exec_time_sort += exec_time
    print(f"--- Sorting {round(exec_time, 2)} second(s) ---")
```

Sorting algorithm

```python
def Sorting(api, decision):
    '''Sorts the boxes based on colour. If no colour is detected the Dobot
    returns to Home position and it is disconnected in order to re-calibrate'''

    global HomeX, HomeY, HomeZ, RedX, RedY, RedZ, GreenX, GreenY, GreenZ, BlueX,
BlueY, BlueZ, red_count, blue_count, green_count, calibration, total_time_arm,
total_time_colour

    start = time.perf_counter()

    dTypeX.SetPTPCmdEx(api, 2, EndX, EndY, EndZ, 0, 1)
    time.sleep(0.1)
    dTypeX.SetEndEffectorSuctionCupEx(api, 1, 1)
    time.sleep(0.1)
    dTypeX.SetPTPCmdEx(api, 2, EndX, EndY, EndZ+5, 0, 1)

    if decision == 'Red':
        dTypeX.SetPTPCmdEx(api, 2, RedX, RedY, EndZ + 5, 0, 1)
        dTypeX.SetPTPCmdEx(api, 2, RedX, RedY, RedZ, 0, 1)
        RedX -= 3
        RedY += 25
        red_count += 1

    elif decision == 'Green':
        dTypeX.SetPTPCmdEx(api, 2, GreenX, GreenY, EndZ + 5, 0, 1)
        dTypeX.SetPTPCmdEx(api, 2, GreenX, GreenY, GreenZ, 0, 1)
        GreenX -= 3
        GreenY += 25
        green_count += 1

    elif decision == 'Blue':
        dTypeX.SetPTPCmdEx(api, 2, BlueX, BlueY, EndZ + 5, 0, 1)
        dTypeX.SetPTPCmdEx(api, 2, BlueX, BlueY, BlueZ, 0, 1)
        BlueX -= 3
        BlueY += 25
        blue_count += 1

    elif decision == None:
        # if None of red, green or blue is detected
        calibration += 1
        total_time_colour += 6
        dTypeX.SetPTPCmdEx(api, 0, HomeX, HomeY, HomeZ, 0, 1)
        time.sleep(0.1)
        print('Calibrate the JeVois Camera')
        dTypeX.SetEndEffectorSuctionCupEx(api, 0, 1)
        dTypeX.DisconnectDobot(api)
        return

    time.sleep(0.1)
    dTypeX.SetEndEffectorSuctionCupEx(api, 0, 1)
    time.sleep(0.1)
    x, y, z = calibrate(api)
    dTypeX.SetPTPCmdEx(api, 2, x, y, z + 10, 0, 1)

    finish = time.perf_counter()
    total = finish - start
    total_time_arm += total
```