DOBOT robotic arm manipulation using interface and image processing Alexander Alzate-Frank

Kevin
Castro-Susana Toro alexander.alzatez@udea.edu.co,
fkevin.castro@udea.edu.co, susana.toro@udea.edu.co University from Antioch.

Medellin

Abstract—In the development of this project, the DOBOT robotic arm is implemented to position 3 cubes of different colors, for which a GUI interface is created through Python 2.5 and the recognition of the colors of the cubes is carried out using digital image processing with Python 3.0. This report presents: the assembly carried out in the project, the programming logic used, a brief introduction about the configuration and manipulation framework of the DOBOT robotic arm and the results of the implementation.

INTRODUCTION

The DOBOT robotic arm is a device of great precision and ease of control, allowing it to be suitable for tasks such as: drawing, writing, object manipulation, among others.

In this project an interface is created using Python which indicates the position of three cubes in a specified space. Additionally, image processing is used to recognize the colors of the cubes and carry out the task successfully.

II. DEVELOPMENT

1. PYTHON-DOBOT COMMUNICATION

The DOBOT robotic arm has specific manufacturing frames for its initialization and movement types. Therefore, you can tell DOBOT what type of end effector you want to manipulate as a Gripper, sucker or laser. This configuration is presented in figure 1. Where the sucker is chosen for the development of this project.

```
def dobot_cmd_send_9():#configuracion de pump
    global cmd_str_10
    cmd_str_10 = [ 0 for i in range(10) ]
    cmd_str_10[0] = 9
    cmd_str_10[1] = 4
    cmd_str_10[2] = 0 #param1 0:suction;1:Gripper;2:laser
    dobot_cmd_send( cmd_str_10 )
```

Figure 1. DOBOT configuration to use the sipper as an end effector.

The function that sends the frame of coordinates and the behavior of the sucker (suck/drop) is dobot_cmd_send_3 where in position 6 of the array cmd_str_10[] indicates this action. The information on these frames is found in Annex 1.

```
#state 3
def dobot_cmd_send_3( x = 265, y = 0, z = -30 ,r = 0,ig=0):
    global cmd_str_10
    cmd_str_10 = [ 0 for i in range(10) ]
    cmd_str_10[0] = 3
    cmd_str_10[2] = x
    cmd_str_10[3] = y
    cmd_str_10[4] = z
    cmd_str_10[5] = r
    cmd_str_10[6] = ig
    cmd_str_10[7] = 1 # MOVL
    dobot_cmd_send( cmd_str_10 )
```

Figure 2. Configuration of coordinates and actuation of the sucker.

2. INTERFACE

Using the Tkinter Python library, a simple and easy-to-use interface was developed. The objective of this interface is to provide the program with a desired configuration of the positioning of three cubes of different colors. There are 5 positions for this where you can only choose 3 of these and each for a specific color.

Each possible position of the cube corresponds to a button in the programming, when the button is pressed it changes color and increases a counter that indicates what this color is. If the counter is equal to zero, the color of this button has not been changed, if it is equal to 1 corresponds to the color red, if it is equal to 2 it corresponds to the color green and if it is equal to 3 it corresponds to the color blue. Below is part of the code for the behavior of the buttons.

```
def clicked1():

    global num_count1 #Incrementando contador
    global num_count2
    global num_count3
    global num_count4
    global num_count5

    num_count1 += 1


    if(num_count1==1 and num_count1 != num_count2 and
!= num_count5):
        btn1.configure(bg="red")
    else:
        if(num_count1==1):
            num_count1+= 1
```

Figure 3. Counter increment for each color

Finally, the interface developed is the following

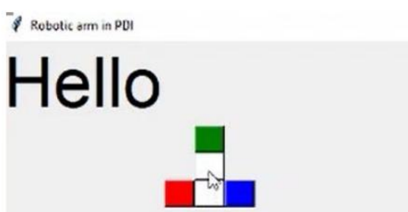

Figure 4. Interface developed Python-Tkinter.

3. DIGITAL PROCESSING OF IMAGE

For the identification of the order of the three cubes in a specific space (only three possible positions), the Python OpenCv library is used. Which allows each color to be identified according to a specific color space, that is, defining a threshold for the binarization of the image. Image processing is done with Python 3.0 in a different file that returns a .txt file which contains an array. This array has three positions whose values can be from 0 to 2, the first position indicates the location of the red cube, the second position indicates the location of the green cube, and the third position the location of the blue cube. In this way, if the array is [2,0,1], it is established that the order is thus first the green cube, next to it the blue cube and lastly the red cube.

The assembly is presented below.

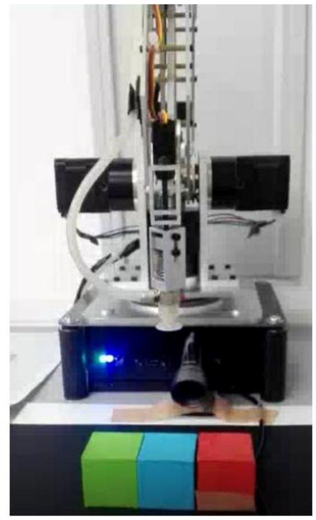

Figure 5. Assembly carried out of the project.

4. IMPLEMENTED SOLUTION METHOD

Before starting the interface, the image processing code (Annex 2) is executed to obtain the array of positions of the three cubes. Next, using the interface, the desired positions are indicated, with the value of the counters of each button a cycle of 5 states is started, which is in charge of carrying out this new positioning.

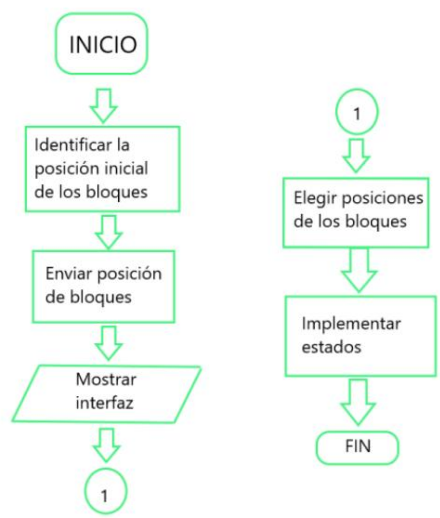

Figure 6. Implemented flowchart

5. IMPLEMENTATION OF STATES

For the positioning of the cubes, 5 states are used, each state makes use of the function mov_position_block(position), this function receives the variable position, whose value corresponds to the position of the array that contains the value of the button counter minus one, since that in the array of positions the range is from 0 to 2 (the values of the counters from 0 to 3) as shown below.

```
def Estado_1():
    global num_count3
    global arreglo_color
    posicion = 0
    if(num_count3!=0):
        posicion = arreglo_color[num_count3-1] + 1
        mov_posicion_bloque(posicion)
        mov_posicion_cuatro()
    Estado_2()
```

Figure 7. Definition of a state

The mov_posicion_bloque() function can make use of three functions, each of which is responsible for moving the corresponding cube and positioning it on the central axis. In the case of state 1, after performing the central positioning of the cube, it is brought to the position of button 4. The following figure shows a graphic example of the operation of the states.

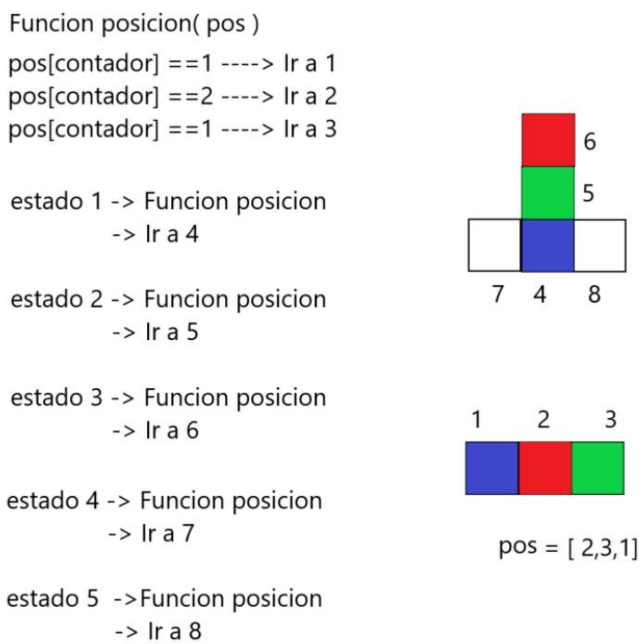

Figure 8. Example of state operation.

III. CONCLUSIONS

- Carrying out applications with the DOBOT robotic arm requires knowledge of the configuration frames and reference system to easily position an element in the desired location.

- The combination of Python tools such as digital image processing allows the application range of the robotic arm to increase. New applications can be considered as future lines of this project. An example of this is optimizing the processing so that figures are recognized and increasing the range of positions.

- A useful tool in the development of the project is Python's Tkinter, through which a user-friendly interface is established so that the user may or may not know programming and manipulate the robot to control the cubes in any of the preset positions.

IV. ANNEXES

1. Configuration parameters (Taken from Dobot Manual)

Mode name	header	Float1	Float2	Float3	Float4	Float5	Float6	Float7	Float8	Float9	Float10	tail
Mouse control(additive coordinate) mode	0XA5	1		additive value of Y axis	additive value of X axis	additive value of Z axis	Rotation angle	Suction cap ON/OFF				0X5A
Joint Jog		2	*Range:* 1-14						*Range:* 1-100 Percentage of the maximum speed			
Linear Jog		7										
Target moving mode		3		x coordinate	y coordinate	z coordinate	Rotation angle	Suction cap ON/OFF	0:Jump 1:MovL 2: MovJ	*Range: 90 to -90*	Pause time after the action (unit: s)	
		6		Joint1 angle	Joint2 angle	Joint3 angle						
Writing and laser mode		4	0: writing 1: laser	additive value of Y axis	additive value of X axis	additive value of Z axis		0: laser ON 0: laser OFF	initial speed	final speed *when Dobot reach its target point*	Maximum speed	
Config Dobot		9	0 Teach configuration	joint jog speed	joint jog acceleration	joint 4 speed	joint 4 acceleration	linear jog speed	linear jog acceleration			
			1 Playback configuration	max joint moving speed	max joint moving acceleration	max servo speed	max servo acceleration	max linear moving seed	max linear moving acceleration	default pause time (unit: s)	JUMP height	
			2 writing configuration	writing acceleration								
			3 manually set initial angle	joint2 angle	joint3 angle							
			4 end effector settings	0: suction cap 1: Gripper 2: Laser								
		10	0 playback speed adjustment	playback moving acceleration percentage	playback moving speed percentage	Teaching mode moving speed percentage						

** note1 : the empty cells should be filled with 0(float, four bytes).*

** note2: state= 5 & state= 8 is used for voice control and gesture control, not introduced here.*

2. Motion Mode Parameters (Taken from Dobot Manual)

Index	header	Float1	Float2	Float3	Float4	Float5
Name	header	state	reserved	X	Y	Z
Explanation	0xA5	3		x coordinate	y coordinate	z coordinate
		6		Joint1 angle	Joint2 angle	Joint3 angle
Index	**Float6**	**Float7**	**Float8**	**Float9**	**Float10**	**tail**
Name	RHead	isGrab	MovingMode	GripperValue	PauseTime	tail
Explanation state: 3 state: 6	Rotation angle	Suction cap ON/OFF	0:Jump, 1:MovL, 2: MovJ	*Range: 90 to -90*	Pause time after the action (unit: s)	0x5A

3. Code digital image processing

```
import cv2
from matplotlib import pyplot as plt
import numpy as np

def In_What_Order_Are_The_Colors():
    cap = cv2.VideoCapture(1)
    read, frame = cap.read()

    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame2 = cv2.cvtColor(frame, cv2.COLOR_RGB2HLS)


    Red=frame[:,:,0]
    Green=frame2[:,:,1]
    Blue=frame[:,:,2]

    #binarize
    LimColorAzul= 100;
    Blue[Blue<=LimColorBlue] = 0; Blue[Blue>0]= 255;

    LimColorRed= 200;
    Red[Red<=LimColorRed] = 0; Red[Red>0]= 255;

    GreenLimColor= 100;
    Green[Green<=LimColorGreen] = 255; Green[Green<255]= 0;

    plt.figure()
    plt.imshow(Red)
    plt.show()

    R=np.mean(np.where(Red==255)[1])
    G=np.mean(np.where(Green==255)[1])
    B=np.mean(np.where(Blue==255)[1])
    Output=[R,G,B]
    ordered_output=[0,0,0]
    sorted_out[0]=np.where(Output==np.min(Output))[0][0]
    sorted_out[1]=np.where(Output==np.median(Output))[0][0]
    sorted_out[2]=np.where(Output==np.max(Output))[0][0] return
    sorted_out
```

4. Graphic interface code and location of cubes

It is attached to the sent files.