Chapter 3: Coordinate based Pick and Place

3.A Objective

The objective of this experiment is to pick up an object from a given coordinate and place the said object at another coordinate.

3.B Outcome

By performing this experiment, the user will learn how to:-

- Move the robotic arm
- Save positions
- Open and close the end effector
- Develop basic Blockly code

3.C Steps

- While keeping the FREEMOTION button (highlighted in Fig. 3.1 in red) pressed, move the arm to a suitable position where it has to pick up an object as seen in Fig. 3.3. The FREEMOTION button cannot be used to open and close the end effector, for that, navigate to the “TOOL COMMAND” menu in Niryo Studio and use the “OPEN” and “CLOSE” buttons to open and close the end effector respectively as shown in Fig. 3.2.

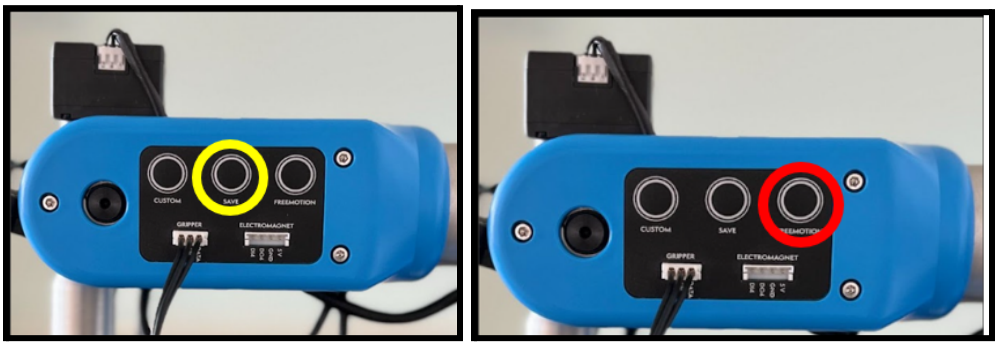


Fig. 3.1. Side panel of Niryo Ned2

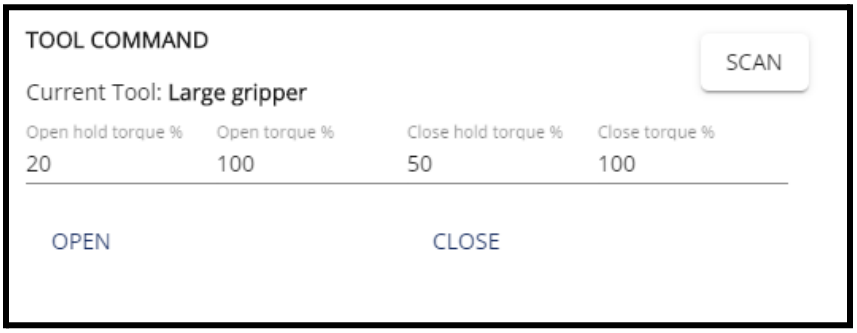


Fig. 3.2. Tool Command menu

- Set the Save mode in Niryo studio to "Pose" and then press the SAVE*(highlighted in Fig. 1.1. in yellow) button (on the side panel), to save the coordinates, which should then directly appear in the Niryo Studio Blockly code editor window (refer Fig. 2.11.).

Fig. 3.3. PICK Position

- Again, while keeping the FREEMOTION button pressed, move the arm to an intermediate position(seen in Fig. 3.4.) and save those coordinates.(This is to prevent the arm from colliding with the workspace while moving to the DROP/PLACE location).

Fig. 3.4. INTERMEDIATE Position

- While keeping the FREEMOTION button pressed, move the arm to its DROP/PLACE position (as shown in Fig. 3.5.) and save those coordinates.

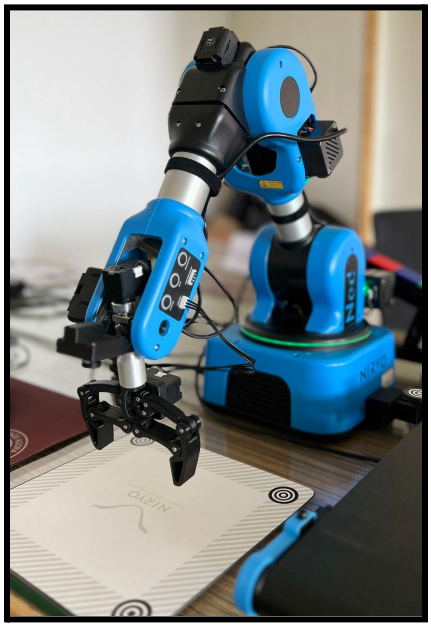


Fig. 3.5. DROP/PLACE Position

*The coordinates can also be directly saved through Niryo Studio (without having to press the SAVE button on the side panel of the arm) by clicking the “ADD POSITION BLOCK” button >> “POSE” >> “CURRENT POSITION” >> “ ADD BLOCK”. Refer Fig. 3.6.

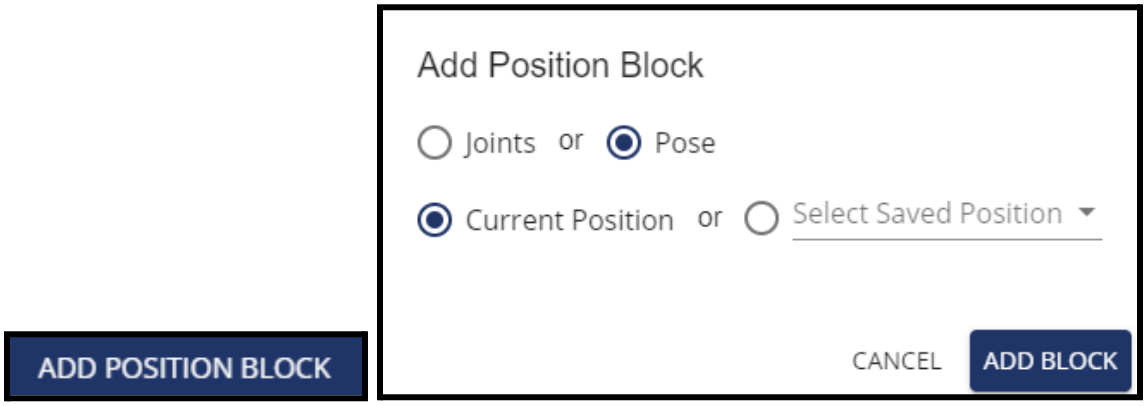


Fig. 3.6. Add Position Block

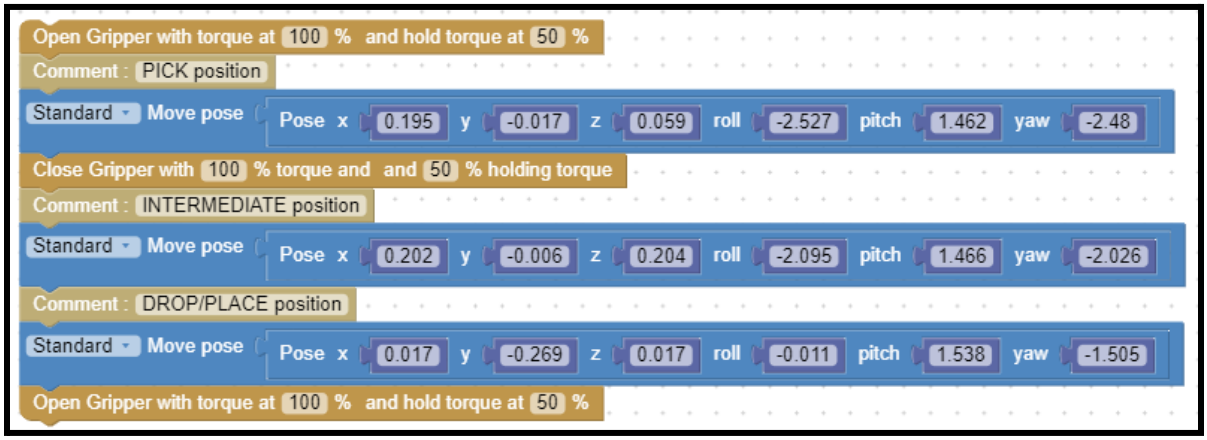


Fig. 3.7. Blockly Code for coordinate based pick and place

- As seen in Fig. 3.7., the gripper needs to be opened before the arm moves to its PICK position and needs to be closed after it picks up the given object.
- The gripper again needs to be opened after it reaches its DROP/PLACE position.
- The blocks for the opening and closing of the gripper can be dragged and dropped from the “Tools” sub-menu as seen in Fig. 3.8.

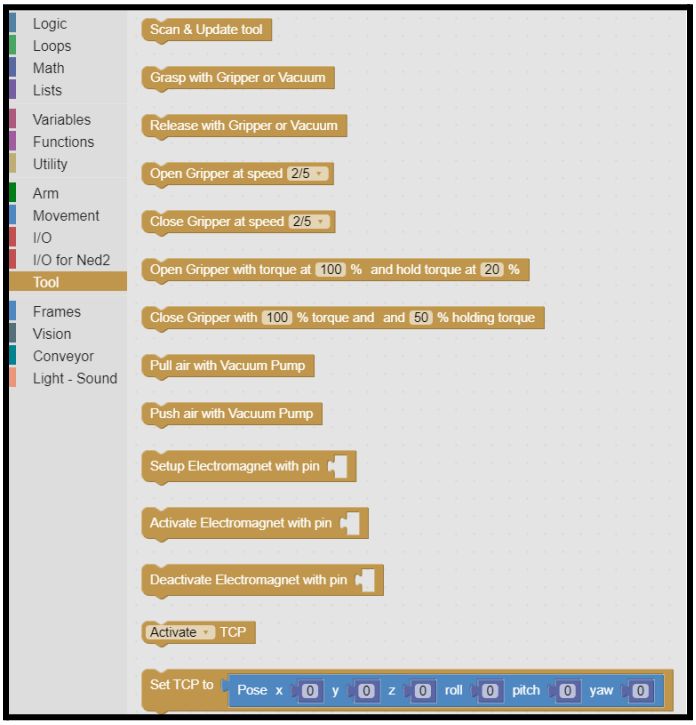


Fig. 3.8. Blocks of code

- Further, the holding torque of the gripper can also be adjusted depending on the weight of the object.

3.C.1 Notes

- End effector coordinates will change after every recalibration of the arm for a new workspace. So all the saved positions (observation, intermediate, place/drop positions) will need to be changed accordingly.
- The gripper needs to be chosen depending on the type/shape/size of the object being picked up.

Chapter 4: Pick and Place using Vision Set

4.A Objective

The objective of this experiment is to use the inbuilt image processing algorithm to detect a certain color (red/blue/green) and type of object (square/circle) within a given workspace using the camera, which will then be picked up from one position and placed at another position.

4.B Prerequisite

Section 2.E and Chapter 3

4.C Outcome

By performing this experiment, the user will learn how to:-

- Create a workspace
- Use the vision set
- Use the vision block in Blockly

4.D Steps

- Use the FREEMOTION button and move the arm to an observation position such that all 4 landmarks on the conveyor workspace are visible in the camera feed as shown in Fig. 4.1. and then save the position using the SAVE button. (The camera feed in Niryo Studio can be used for this, by selecting “Camera Streaming” and applying the “Landmarks” filter. If all the landmarks are clearly visible they will light up as green on the camera stream and light up red if only a few out of the 4 landmarks are visible).

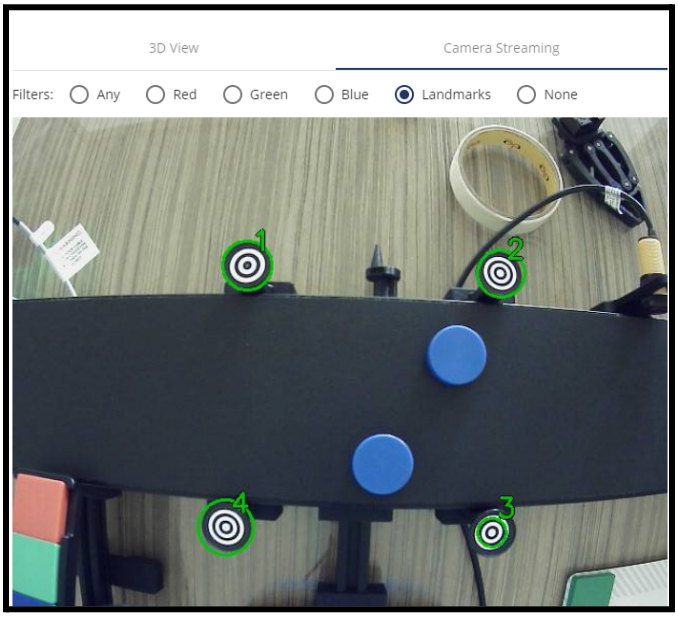


Fig. 4.1. Camera Stream in Niryo Studio

Fig. 4.2. Observation Position

- Make sure only the landmarks on the conveyor are visible(as seen in Fig. 4.1 and Fig. 4.2.), because of the presence of any more landmarks within the camera stream as seen in Fig. 4.3. will hinder the image processing algorithm from running correctly and the arm won't be able to pick up the object.

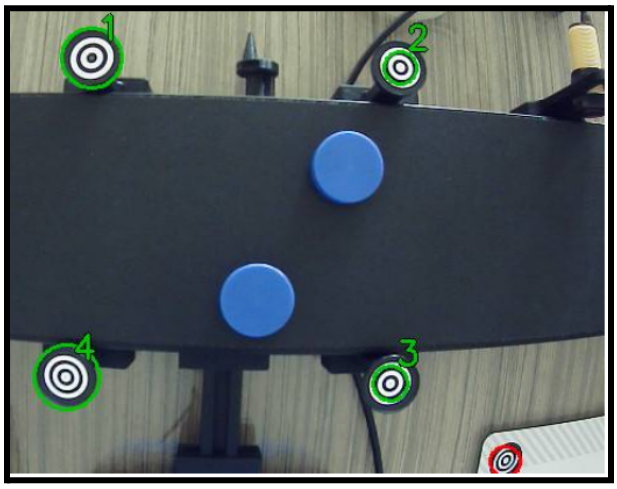


Fig. 4.3. Presence of another landmark other than the ones on the Conveyor

- Again using the FREEMOTION button move the arm to a DROP/PLACE position where the arm will drop the object it has picked up (as shown in Fig. 4.4.).

Fig. 4.4. PLACE/DROP Position

- The code block can be taken from the “Vision” sub-menu.
- A while loop can be added from the “Loops” sub-menu to keep the code running for multiple iterations.
- “Color" and “Shape” of objects can also be specified to pick up only a certain type of object as shown in Fig. 4.5.
- A workspace needs to be created with 4 landmarks (Refer Workspace Calibration for Vision Set) . In this case the “Conveyer_Workspace" is being used.

- A height offset needs to be given from the base of the robotic arm which can be determined using trial and error. For the conveyor workspace the offset was determined to be "-12" mm.
- If the arm detects an object of given shape and color from its observation position it will automatically pick up the given object and place it at the given DROP/PLACE location.

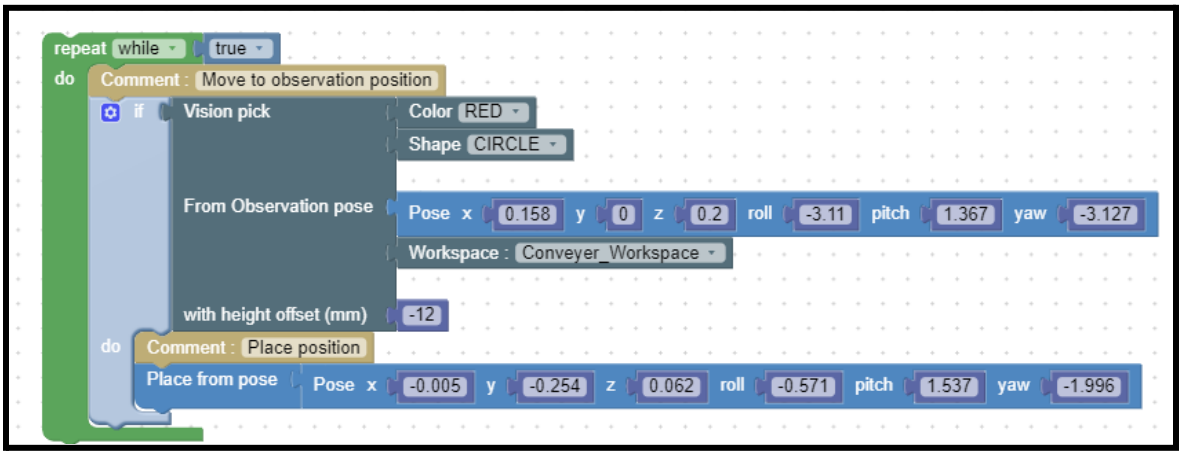


Fig. 4.5. Blockly Code for Vision based pick and place

Chapter 5: Pick and place using the conveyor belt and IR sensor

5.A. Objective

The objective of this experiment is to use the IR sensor to detect an object that is moving on the conveyor belt and stop the belt and accordingly move the robotic arm to the coordinates where the object is located, pick it up and place it at a desired location.

5.B. Prerequisite

Section 2.D and Chapter 3

5.C Outcome

By performing this experiment, the user will learn how to:-

- Control the conveyor belt
- Read the status of the IR sensor
- Use the Conveyor block in Blockly

Note:- Make sure the objects placed on the conveyor belt are centered properly like in Fig. 5.1.

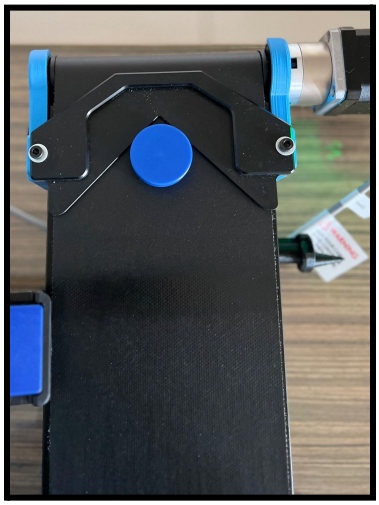


Fig. 5.1. Object placed at beginning of conveyor belt

5.D Steps

- Determine a home/starting position for the robotic arm and save it as seen in Fig. 5.2. Either the “Move joints” (saves joint angle in radians) or the “Move pose” (uses cartesian coordinate system to determine position (x,y,z) and orientation (roll,pitch,yaw)) block code can be used to save a particular position.

Fig. 5.2. Home/Starting position

- Block code to move the conveyor belt can be found under the Conveyor sub-menu in Niryo Studio. Up to 2 conveyor belts can be connected to Ned2. In this case set “Control Conveyor” to “1”.
- The speed and direction of the conveyor belt can also be set according to the use case.
- Use the “Get” block from the “I/O for Ned2” sub-menu to set the GPIO pin to which the IR sensor is connected to. In this case the pin is “DI5”. The conveyor needs to be stopped when the Digital Input from the IR sensor is low, i.e “false”.
- Determine the intermediate positions while picking up and placing objects and also the PICK and PLACE position and save it.
- The PICK position will also change depending on the speed of the conveyor belt since the object won't stop at the same position shown in Fig. 5.3.

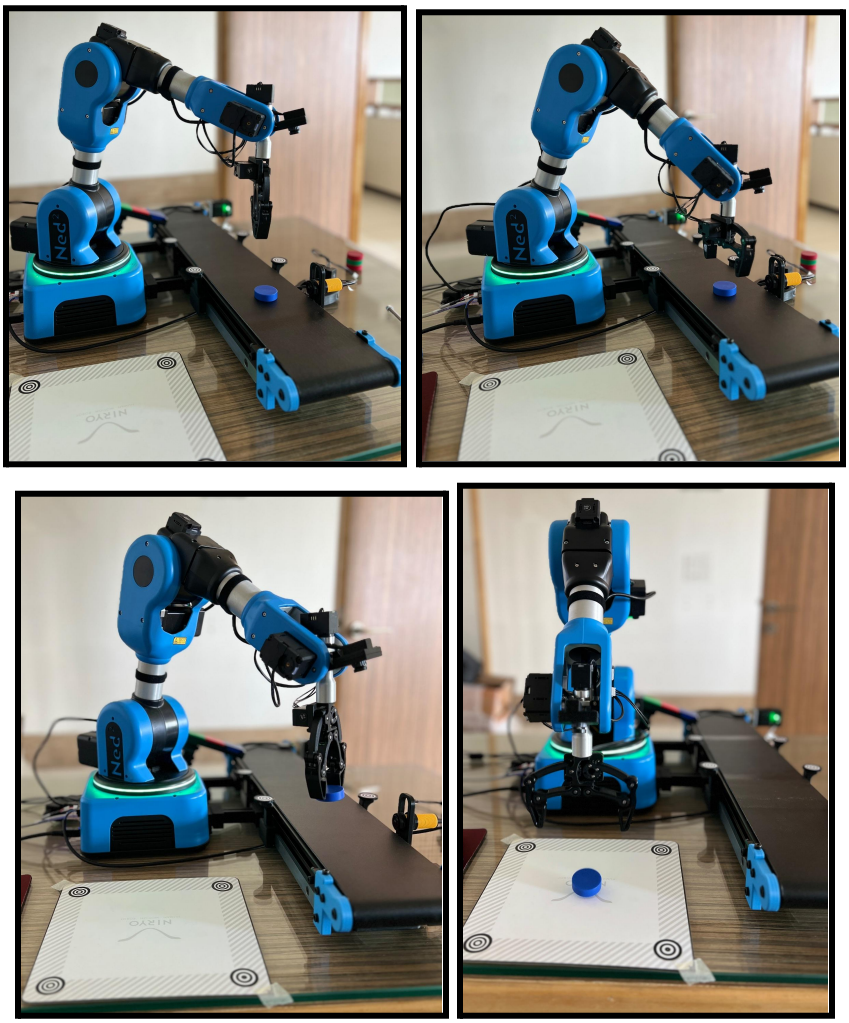


Fig. 5.3. Intermediate and final positions for picking and placing of the object

- After the arm picks up and drops an object at a given position, the conveyor belt needs to be restarted and should keep running till another object passes the IR sensor. Refer Fig. 5.4. A loop from the “Loops” sub-menu can be used for this.
- The “Wait for” block under the Utility sub-menu can be used to give delays.

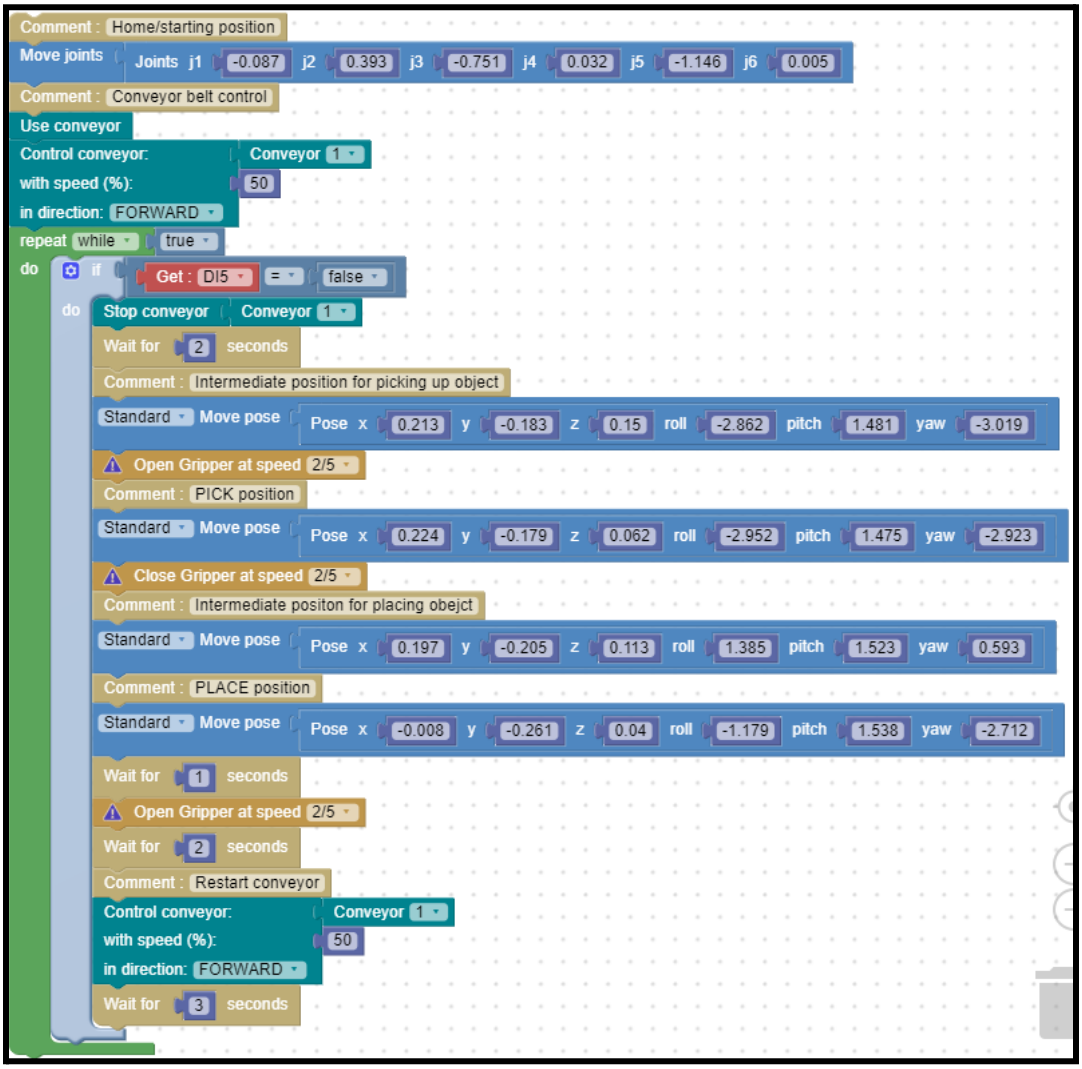


Fig. 5.4. Code for pick and place using the conveyor belt and IR sensor

Chapter 6: Pick and place using the conveyor belt and vision set

6.A Objective

The objective of this experiment is to move the robotic arm to an observation position where all 4 landmarks on the conveyor workspace are visible in the camera stream.The conveyor belt runs till an object of particular color and shape is detected within the workspace. The conveyor belt stops and the robotic arm picks up the particular object and places it at a given position.

6.B Prerequisite

Chapter 3,4,5

6.C Outcome

By performing this experiment, the user will learn how to:-

- Use the conveyor belt with the vision set
- Read the status of the IR sensor
- Use the Conveyor block in Blockly

6.D Steps

- Set Conveyor block with conveyor number, speed and direction.
- Use the FREEMOTION button and move the arm to a suitable observation position like the one in Fig. 6.1. where all the 4 landmarks which can be seen in Fig. 6.2. in the conveyor workspace are clearly visible, and then save the position.

Fig. 6.1. Observation Position

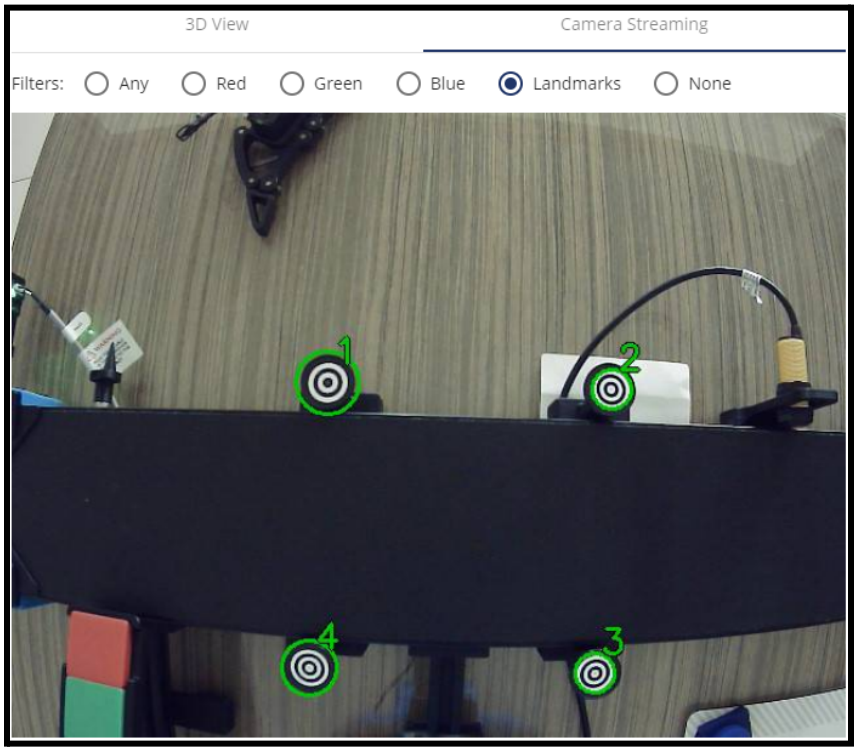


Fig. 6.2. Landmarks clearly visible from the observation position

- Use the “Is object detected” block from the “Vision” sub-menu to detect an object of particular color and shape or any object within the given workspace. In this case the workspace is the “Conveyer_Workspace”.
- Use the “Vision pick” block from the “Vision” sub-menu to pick up the given object from the workspace and give a suitable height offset (in mm) . In this case the height offset is “-12” as shown in Fig. 6.3.

Fig. 6.3. Picking up the object

- Finally determine a PLACE/DROP position for the object and save it as shown in Fig. 6.4.

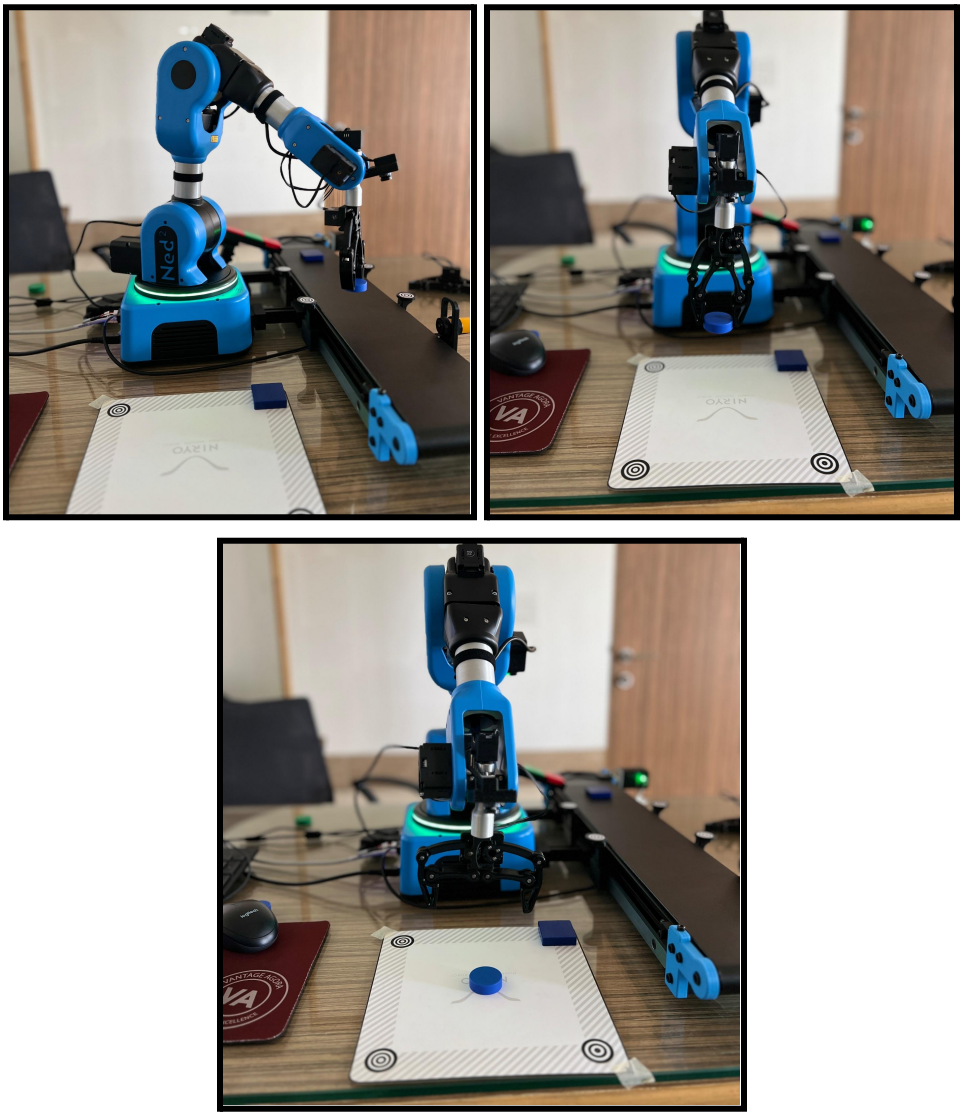


Fig. 6.4. Placing of object at a given location

- A loop can be used for multiple iterations of pick up and drop as shown in Fig. 6.5.

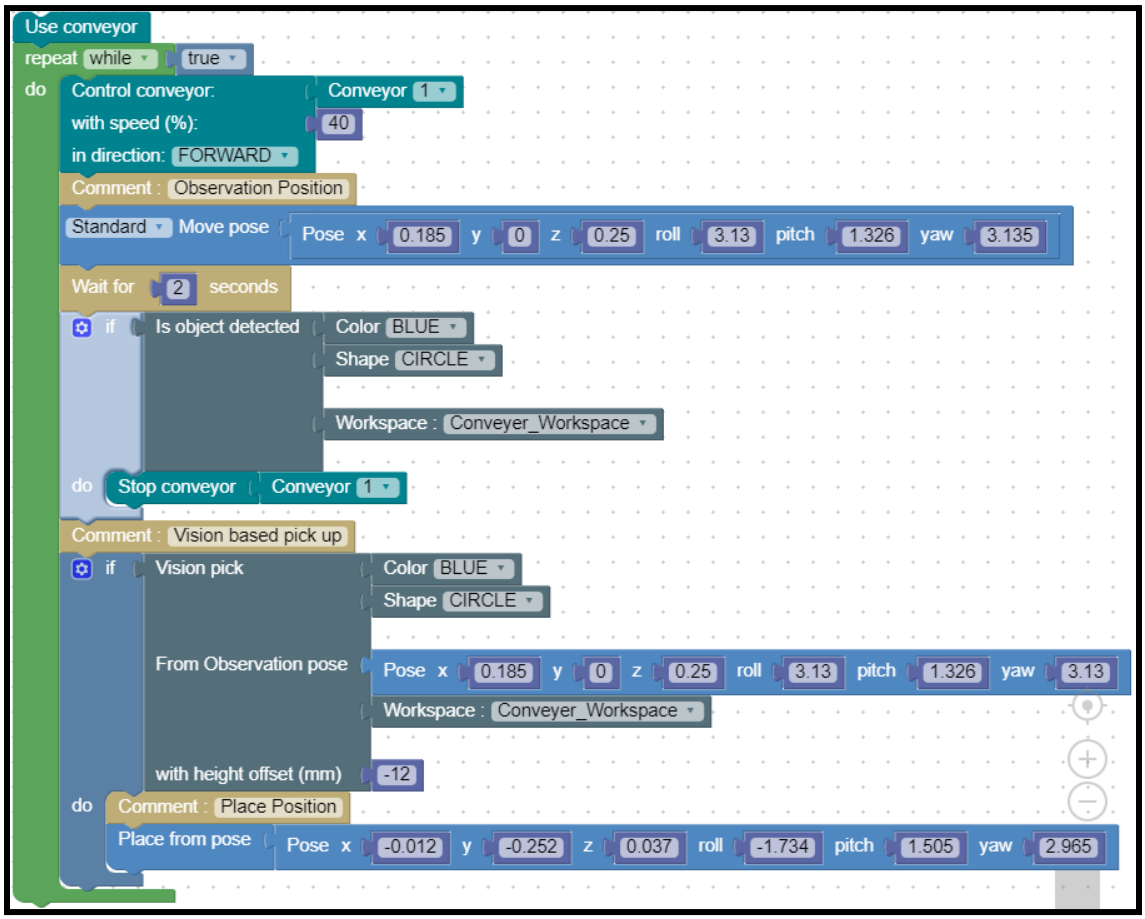


Fig.6.5. Code for pick and place using the vision set and conveyor belt

Chapter 7: Pick and place using the gravity feeder

7.A Objective

The objective of this experiment is to pick up square objects from the gravity feeder and place them at a given position using python.

7.B Outcome

By performing this experiment, the user will learn how to:-

- Use pyniryo2 library to control the arm

7.C Steps

- Python programming has been used in this case
- Import all public names from the module pyniryo2
- Establish connection with the robot by specifying the IP address
- Calibrate the arm and then equip the tool/end effector
- Determine the pick position as shown in Fig. 7.1. and use Niryo Studio to get the x,y,z coordinates as well as Roll,Pitch and Yaw and feed those coordinates into the "pick_from_pose" function in the order x,y,z,roll,pitch,yaw.

Fig. 7.1. Pick Position

- Determine an intermediate position similar to the one in Fig. 7.2. for the arm after it picks up the object so as to prevent it from colliding with the gravity feeder after picking up the object or dropping the object. Use the “move_pose” function for this.

Fig. 7.2. Intermediate Position

- Determine a DROP/PLACE position and use the “place_from_pose” function to place the object at those particular coordinates(as shown in Fig. 7.3.).

Fig. 7.3. DROP/PLACE Position

- Use a loop for multiple iterations. In this case a while loop has been used and there will be a total of 6 iterations as shown in Fig. 7.4.
- After the execution comes out of the loop the arm will return to its home position and will activate learning mode. (This is done using the “go_to_sleep” function)
- The connection to the arm can now be disabled using the “end” function.

```
from pyniryo2 import * #importing pyniryo2 module into the current workspace
robot = NiryoRobot("169.254.200.200") # this is the IP address for ethernet connection
i=0
robot.arm.calibrate_auto() # calibrating the arm
robot.tool.update_tool() # equiping the tool/end effector
robot.tool.release_with_tool() #opening the tool/end effector
while i<6:
    robot.pick_place.pick_from_pose([0.160,0.167,0.083,3.040,1.229,3.080]) #pick position (x,y,z,roll,pitch,yaw), arm will automatically close the gripper
    robot.arm.move_pose([0.186,0.155,0.145,2.757,1.166,3.079]) #intermediate position (x,y,z,roll,pitch,yaw)
    robot.pick_place.place_from_pose([0.000,-0.263,0.018,2.503,1.443,0.987]) #place position (x,y,z,roll,pitch,yaw),arm will automatically open the gripper
    i+=1


robot.arm.go_to_sleep() #arm returns to home position
robot.end() #connection is disabled
```

Fig. 7.4. Python code for pick and place with gravity feeder

Chapter 8: Sorting based on color/shape in python

8.A Objective

The objective of this experiment is to sort objects of three different colors (red, blue and green) placed within a given workspace into three different piles.

8.B Prerequisite

Chapter 7, Section 2.D and 2.E

8.C Outcome

By performing this experiment, the user will learn how to:-

- Use vision set using python
- Detect color and shape of object using python

8.D Steps

- Use “PoseObject” (a python object to store 6 coordinates (x,y,z,roll,pitch,yaw) in one instance) to store an observation position from which, all 4 landmarks, similar to the position in Fig. 8.1. of any workspace are clearly visible.

Fig. 8.1. Observation Position

- Also determine and save PLACE/DROP positions for three differently colored objects (red,blue and green)
- Then create an intermediate position(as shown in Fig. 8.2.) for the robotic arm between the pick and place position so as to prevent the arm from colliding with any other object in its workspace/environment.

```
observation_pose = PoseObject(x=0.203,y=-0.003,z=0.318,roll=-2.921,pitch=1.282,yaw=-2.950)
place_pose_blue=PoseObject(x=0.043,y=-0.168,z=0.125,roll=2.341,pitch=1.496,yaw=0.854)
place_pose_red=PoseObject(x=0.041,y=-0.334,z=0.137,roll=2.128,pitch=1.534,yaw=0.634)
place_pose_green=PoseObject(x=-0.117,y=-0.325,z=0.142,roll=-1.772,pitch=1.507,yaw=2.967)
intermediate_pose = PoseObject(x=0.207,y=0.003,z=0.267,roll=2.491,pitch=1.529,yaw=2.488)
```

Fig. 8.2. Intermediate Position

- Create a dictionary so as to keep count of the number of objects of each color being picked up

```
count_dict={
    ObjectColor.BLUE: 0,
    ObjectColor.RED: 0,
    ObjectColor.GREEN: 0}
```

- Create variables of type integer to keep count of the failures as well as the maximum number of objects that can be picked up. (“max_count”, “failure_count”, “count”)
- First move the arm to the given observation position using the “move_pose” function.
- Perform a vision based pick of an object using the “vision_pick” function which takes the workspace name and height offset as its parameters.

```
robot.arm.move_pose(observation_pose) #move to observation position
obj_found, shape, color = robot.vision.vision_pick(workspace_name,height_offset=-0.012) # perfrom vision based pick up
```

- If an object is not found within the given workspace, wait for 12 sec and continue if an object is detected(as shown in Fig. 8.3.), otherwise break out of the while loop and enter the robotic arm into sleep mode and end the connection.

```
if not obj_found:
    robot.wait(12) #waits for 12 seconds if no object is detected
    if not obj_found:
        print("Failed to detect object") #if no object is founnd after 12 seconds the executuin will break out the loop
        break
    continue
```

Fig. 8.3. Waiting for object

- If an object is detected, reset the failure count to zero and use if-elif statements to place the objects of different colors into different piles using the “move_pose” and “place_from_pose” functions(as shown in Fig. 8.4.).

```
if color==ObjectColor.BLUE :
    robot.arm.move_pose(intermediate_pose) #moving to intermediate position
    robot.pick_place.place_from_pose(place_pose_blue) # placing the object at a pre defined location

elif color == ObjectColor.RED:
    robot.arm.move_pose(intermediate_pose)
    robot.pick_place.place_from_pose(place_pose_red)
else:
    robot.arm.move_pose(intermediate_pose)
    robot.pick_place.place_from_pose(place_pose_green)
```

- Increment the count of the particular color in the dictionary as well as the overall count.

```
count_dict[color] += 1 #incrementing the count of objects in the dictionary of a particular color
count+=1
```

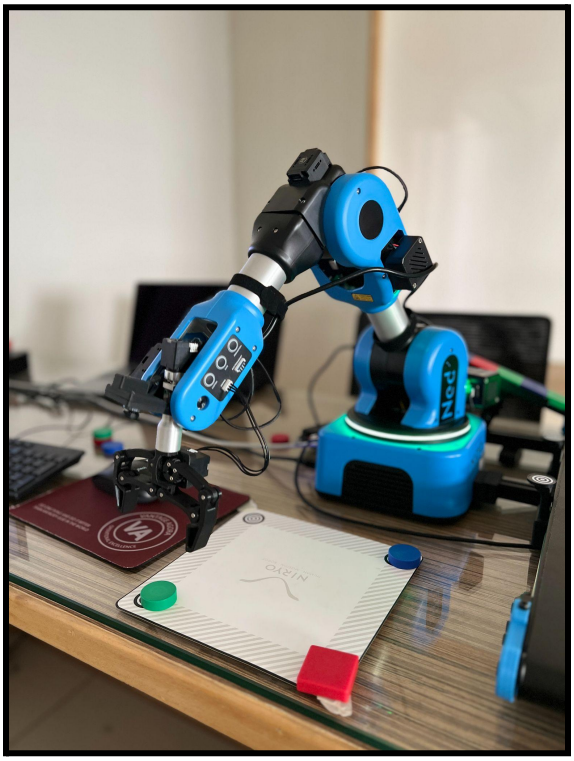


Fig. 8.4. Objects sorted

- After the execution comes out of the loop, close the end effector, put the robotic arm into sleep mode and end the connection(as shown in Fig. 8.5.).

```
robot.tool.grasp_with_tool() #close the gripper/end effector
robot.arm.go_to_sleep() #arm enters sleep mode
robot.end() # connection to the arm is closed
```

Fig. 8.5. Sleep mode

8.D.1 NOTE

The steps to be followed to segregate objects based on their shape (square and circle) are almost the same. The only changes to be made to the code are as follows:-

- Use “ObjectShape” in the dictionary instead of ObjectColor

```
count_dict={
    ObjectShape.SQUARE: 0,
    ObjectShape.CIRCLE: 0}
```

- In the if-else statements, check for equality of shape instead of color.

```
if shape==ObjectShape.SQUARE :
    robot.arm.move_pose(intermediate_pose)
    robot.pick_place.place_from_pose(place_pose_square)
else:
    robot.arm.move_pose(intermediate_pose)
    robot.pick_place.place_from_pose(place_pose_circle)
```

```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18:58:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Niryo\Desktop\Niryo Ned2\Code for Documentation\Color_sorting.py
{<ObjectColor.BLUE: 'BLUE'>: 0, <ObjectColor.RED: 'RED'>: 1, <ObjectColor.GREEN: 'GREEN'>: 0}
{<ObjectColor.BLUE: 'BLUE'>: 1, <ObjectColor.RED: 'RED'>: 1, <ObjectColor.GREEN: 'GREEN'>: 0}
{<ObjectColor.BLUE: 'BLUE'>: 1, <ObjectColor.RED: 'RED'>: 1, <ObjectColor.GREEN: 'GREEN'>: 1}
Failed to detect object
```

Shell displaying the number of objects of each color that is picked up

```python
from pyniryo2 import *
workspace_name="Conveyor_Workspace"
robot=NiryoRobot("169.254.200.200") #Establishing connection with the robotic arm


observation_pose = PoseObject(x=0.154,y=-0.031,z=0.181,roll=-2.676,pitch=1.366,yaw=-2.695)
place_pose_blue=PoseObject(x=0.048,y=-0.166,z=0.018,roll=3.139,pitch=1.561,yaw=1.546)
place_pose_red=PoseObject(x=0.045,y=-0.339,z=0.009,roll=-0.396,pitch=1.434,yaw=-1.937)
place_pose_green=PoseObject(x=-0.122,y=-0.331,z=0.014,roll=-0.714,pitch=1.436,yaw=-2.370)
intermediate_pose = PoseObject(x=0.214,y=-0.015,z=0.149,roll=-2.242,pitch=1.469,yaw=-2.400)


robot.arm.calibrate_auto() # Automatic calibration of the robotic arm
robot.tool.update_tool() # Equiping the given tool/end effector

max_count=10
count=0


count_dict={
    ObjectColor.BLUE: 0,
    ObjectColor.RED: 0,
    ObjectColor.GREEN: 0} # dictionary to keep count of the number of objects of differerent colors being picked up

while count<max_count:
    robot.arm.move_pose(observation_pose) #move to observation position
    obj_found, shape, color = robot.vision.vision_pick(workspace_name,height_offset=-0.012) # perfrom vision based pick up

    if not obj_found:
        robot.wait(12) #waits for 12 seconds if no object is detected
        if not obj_found:
            print("Failed to detect object") #if no object is founnd after 12 seconds the executuin will break out the loop
            break
        continue

    if color==ObjectColor.BLUE :
        robot.arm.move_pose(intermediate_pose) #moving to intermediate position
        robot.pick_place.place_from_pose(place_pose_blue) # placing the object at a pre defined location

    elif color == ObjectColor.RED:
        robot.arm.move_pose(intermediate_pose)
        robot.pick_place.place_from_pose(place_pose_red)
    else:
        robot.arm.move_pose(intermediate_pose)
        robot.pick_place.place_from_pose(place_pose_green)


    count_dict[color] += 1 #incrementing the count of objects in the dictionary of a particular color
    count+=1
    print(count_dict)



robot.wait(0.2)
robot.tool.grasp_with_tool() #close the gripper/end effector
robot.arm.go_to_sleep() #arm enters sleep mode
robot.end() # connection to the arm is closed
```

Code for color sorting

```
from pyniryo2 import *
workspace_name="Conveyor_Workspace"
robot=NiryoRobot("169.254.200.200")#Establishing connection with the robotic arm

observation_pose = PoseObject(x=0.154,y=-0.031,z=0.181,roll=-2.676,pitch=1.366,yaw=-2.695)
place_pose_square=PoseObject(x=0.045,y=-0.339,z=0.009,roll=-0.396,pitch=1.434,yaw=-1.937)
place_pose_circle=PoseObject(x=-0.122,y=-0.331,z=0.014,roll=-0.714,pitch=1.436,yaw=-2.370)
intermediate_pose = PoseObject(x=0.214,y=-0.015,z=0.149,roll=-2.242,pitch=1.469,yaw=-2.400)

robot.arm.calibrate_auto() # Automatic calibration of the robotic arm
robot.tool.update_tool() # Equiping the given tool/end effector

max_count=10
count=0



count_dict={
    ObjectShape.SQUARE: 0,
    ObjectShape.CIRCLE: 0} # dictionary to keep count of the number of objects of differerent shapes being picked up


while count<max_count:
    robot.arm.move_pose(observation_pose)#move to observation position
    obj_found, shape, color = robot.vision.vision_pick(workspace_name,height_offset=-0.012)# perfrom vision based pick up


    if not obj_found:
        robot.wait(12) #waits for 12 seconds if no object is detected
        if not obj_found:
            print("Failed to detect object") #if no object is founnd after 12 seconds the executuin will break out the loop
            break
        continue


    if shape==ObjectShape.SQUARE :
        robot.arm.move_pose(intermediate_pose) #moving to intermediate position
        robot.pick_place.place_from_pose(place_pose_square) # placing the object at a pre defined location
    else:
        robot.arm.move_pose(intermediate_pose)
        robot.pick_place.place_from_pose(place_pose_circle)


    count_dict[shape] += 1 #incrementing the count of objects in the dictionary of a particular shape
    count+=1


    print(count_dict)

robot.wait(0.2)
robot.tool.grasp_with_tool()#close the gripper/end effector
robot.arm.go_to_sleep()#arm enters sleep mode
robot.end()# connection to the arm is closed
```

Code for object sorting

8.E Errors one may expect

Error:- The robotic arm makes a beeping sound and does not move into its observation position.
Cause:- Incorrect coordinates given for the observation position.
Solution:- Use the FREEMOTION button to move the arm into a correct observation position and update that position in the code

Error:- The robotic arm moves into a different position other than its observation position.
Cause:- Coordinates may have changed after calibration of the robotic arm.
Solution:- Set new poses (observation, pick, intermediate and place).

Chapter 9: Segregation using the Vision Set, Conveyor belt, Vacuum pump and sound

9.A Objective

The objective of this experiment is to perform a vision based pick up using the vacuum pump on only a particular object (red circle in this case) moving on the conveyor belt.

9.B Prerequisite

Chapter 7,8 and Section 2.D and 2.E

9.C Outcome

By performing this experiment, the user will learn how to:-

- Vision Based sorting
- Sound interfacing with Python
- Vacuum Pump interfacing

9.D Note

To connect the Vacuum pump, attach the connector to the back panel interface of the robotic arm into the slot labeled “VACUUM PUMP” and insert the vacuum pump end effector into the given end effector slot on the arm.

9.E Steps

- Import all public names(functions, objects etc.) from the “pyniryo2” module into the given file.
- Define the workspace within which the image processing must work.(In this case it's the Conveyor_Workspace, a new workspace can be created using Niryo Studio and 4 landmarks)
- Establish connection to the robotic arm with the given IP address.
- Set the volume of the inbuilt speakers of the robotic arm to be used later on using the “set_volume” function.
- Automatically calibrate the arm using the “calibrate_auto” function.

- Equip the end effector (in this case it's the Vacuum pump)

```
from pyniryo2 import *
workspace_name="Conveyor_Workspace"
robot=NiryoRobot("169.254.200.200") #Establishing connection with the robotic arm
robot.sound.set_volume(100)#setting volume of the inbuilt speaker
robot.arm.calibrate_auto() # Automatic calibration of the robotic arm
robot.tool.update_tool() # Equiping the given tool/end effector
```

- Set Observation,Intermediate and DROP/PLACE(as shown in Fig. 9.1, 9.2, 9.3) positions using the “PoseObject”(python object to store 6 coordinates (x,y,z,roll,pitch,yaw) in one instance) object.

Fig. 9.1. Observation Pose

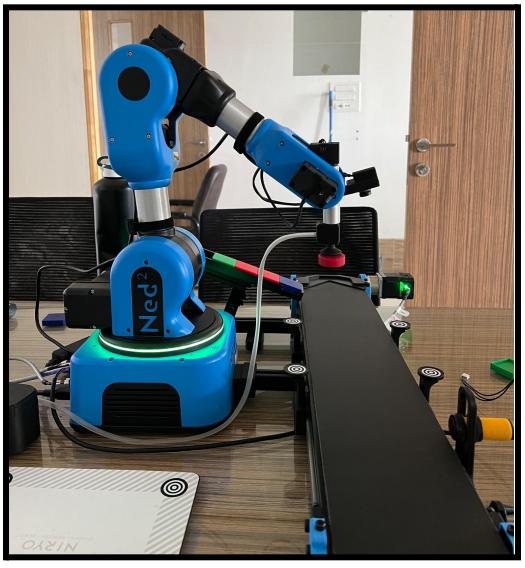


Fig. 9.2. Intermediate Pose

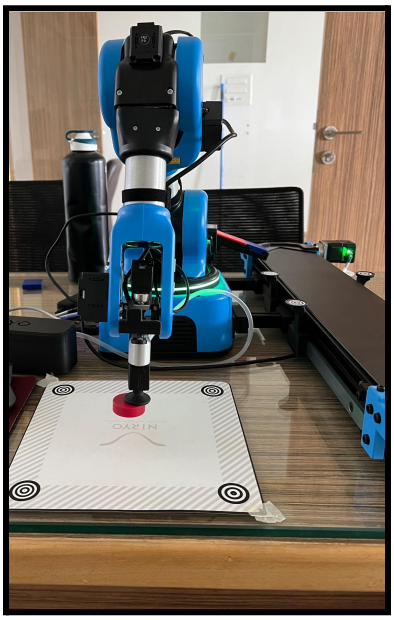


Fig 9.3. Place Pose

```
observation_pose = PoseObject(x=0.159,y=0.003,z=0.252,roll=2.953,pitch=1.420,yaw=2.938)
place_pose_red=PoseObject(x=-0.039,y=-0.258,z=0.032,roll=-1.003,pitch=1.512,yaw=-2.490)
intermediate_pose = PoseObject(x=0.210,y=-0.004,z=0.179,roll=1.565,pitch=1.488,yaw=1.537)
```

- Create variables to keep count of the red circles being picked up and failures.
- Create variables for shape and color of object to be picked up. In this case it's a red circle and get the conveyor id (through set_conveyor() function) and store it in the conveyor_id variable.

```
red_circle_count=0 #count for red circles picked up
failure_count=0
shape_expected = ObjectShape.CIRCLE
color_expected = ObjectColor.RED
conveyor_id = robot.conveyor.set_conveyor()
```

- Use a while true and first move the robotic arm to its observation pose using the move_pose() function.
- Further use the run_conveyor function to set the direction(forward/reverse) and speed (in terms of percentage) of the conveyor.
- Use the detect_object() function to only detect red circles within the given workspace by giving it the workspace name, shape and color as arguments.

```
while True:

    robot.arm.move_pose(observation_pose) #move to observation position
    robot.conveyor.run_conveyor(conveyor_id,speed=50, direction=ConveyorDirection.FORWARD)#running the conveyor

    obj_detect,obj_pos, shape, color = robot.vision.detect_object(workspace_name,shape=shape_expected,
                                        color=color_expected)#detecting red circles within the given Conveyor Workspace
```

- If no object of the given type(red,circle) is detected within the workspace, wait for 1 sec, if still no object is detected increment the failure_count variable by one. If the failure count reaches 25 break out of the while loop otherwise continue to the next iteration, also reset the failure count if an object is detected.

```
if not obj_detect:
    robot.wait(1)
    if not obj_detect:
        failure_count+=1
        if failure_count==25:
            print("Failed to detect object")
            break
    continue


if obj_detect:
    failure_count=0
```

- Stop the conveyor after the correct type of object is detected by using the stop_conveyor() function and giving the conveyor_id as an argument.
- Perform vision based pick up using the vacuum pump using the vision_pick() function and giving the workspace name, height offset, shape and color as arguments.
- Play the sound that a red object has been picked up using the inbuilt speakers, while moving the arm to its intermediate then the place position.

```
robot.conveyor.stop_conveyor(conveyor_id)

obj_found, shape, color = robot.vision.vision_pick(workspace_name,height_offset=-0.012,shape=shape_expected,
                                    color=color_expected)# perform vision based pick up



robot.sound.play("red.wav") #playing an audio file through the inbuilt speaker
robot.arm.move_pose(intermediate_pose)#moving to intermediate position
robot.pick_place.place_from_pose(place_pose_red)# placing the object at a pre defined location
```

NOTE:- The sound needs to be saved into a .wav file and imported into Niryo Studio. The sound can also directly be played using the gTTS python module.
The program will automatically activate the vacuum pump while using the vision_pick() function and will automatically deactivate the pump while using the place_from_pose() function.

- Increment the count of red circles and print it

```
    red_circle_count+= 1 #incrementing count of red circles
    print("Number of Red Objects picked up:",red_circle_count)
```

- Outside of the loop, play the sound stating that the arm is returning to its home position,while the program stops and disconnects the conveyor belt and deactivates the vacuum pump and moves the arm.
- Finally terminate the connection to the robotic arm to end the program.

```
robot.sound.play("return_home.wav")
robot.wait(0.2)
robot.conveyor.stop_conveyor(conveyor_id)#stopping the conveyor
robot.conveyor.unset_conveyor(conveyor_id)#disconnecting the conveyor
robot.tool.grasp_with_tool()#switching off the vacuum pump
robot.arm.move_to_home_pose()#moving the arm to a home position
robot.end()#terminating the connection to the robotic arm
```

```
from pyniryo2 import *
workspace_name="Conveyor_Workspace"
robot=NiryoRobot("169.254.200.200") #Establishing connection with the robotic arm
robot.sound.set_volume(100)#setting volume of the inbuilt speaker
robot.arm.calibrate_auto() # Automatic calibration of the robotic arm
robot.tool.update_tool() # Equiping the given tool/end effector

observation_pose = PoseObject(x=0.159,y=0.003,z=0.252,roll=2.953,pitch=1.420,yaw=2.938)
place_pose_red=PoseObject(x=-0.039,y=-0.258,z=0.032,roll=-1.003,pitch=1.512,yaw=-2.490)
intermediate_pose = PoseObject(x=0.210,y=-0.004,z=0.179,roll=1.565,pitch=1.488,yaw=1.537)


red_circle_count=0 #count for red circles picked up
failure_count=0
shape_expected = ObjectShape.CIRCLE
color_expected = ObjectColor.RED
conveyor_id = robot.conveyor.set_conveyor()


while True:

    robot.arm.move_pose(observation_pose) #move to observation position
    robot.conveyor.run_conveyor(conveyor_id,speed=50, direction=ConveyorDirection.FORWARD)#running the conveyor

    obj_detect,obj_pos, shape, color = robot.vision.detect_object(workspace_name,shape=shape_expected,
                                        color=color_expected)#detecting red circles within the given Conveyor Workspace

    if not obj_detect:
        robot.wait(1)
        if not obj_detect:
            failure_count+=1
            if failure_count==25:
                print("Failed to detect object")
                break
        continue


    if obj_detect:
        failure_count=0

    robot.conveyor.stop_conveyor(conveyor_id)

    obj_found, shape, color = robot.vision.vision_pick(workspace_name,height_offset=-0.012,shape=shape_expected,
                                        color=color_expected)# perform vision based pick up



    robot.sound.play("red.wav") #playing an audio file through the inbuilt speaker
    robot.arm.move_pose(intermediate_pose)#moving to intermediate position
    robot.pick_place.place_from_pose(place_pose_red)# placing the object at a pre defined location
    red_circle_count+= 1 #incrementing count of red circles
    print("Number of Red Objects picked up:",red_circle_count)

robot.sound.play("return_home.wav")
robot.wait(0.2)
robot.conveyor.stop_conveyor(conveyor_id)#stopping the conveyor
robot.conveyor.unset_conveyor(conveyor_id)#disconnecting the conveyor
robot.tool.grasp_with_tool()#switching off the vacuum pump
robot.arm.move_to_home_pose()#moving the arm to a home position
robot.end()#terminating the connection to the robotic arm
```

Full code for red circular object segregation

Chapter 10:Magnetic Object Segregation using the Conveyor belt,IR Sensor and Electromagnet

10.A Objective

The objective of this experiment is picking up metallic objects using the electromagnet moving on the conveyor belt.

10.B Prerequisite

Chapter 7, 8 and Section 2.D

10.C Outcome

By performing this experiment, the user will learn how to:-

- Interface the Electromagnet with the robotic arm

10.D Note

To connect the electromagnet, insert the electromagnet end effector into the robotic arm into the end effector slot on the arm and connect the 4 pin connector to the electromagnet connector as shown in Fig 10.1 and 10.2

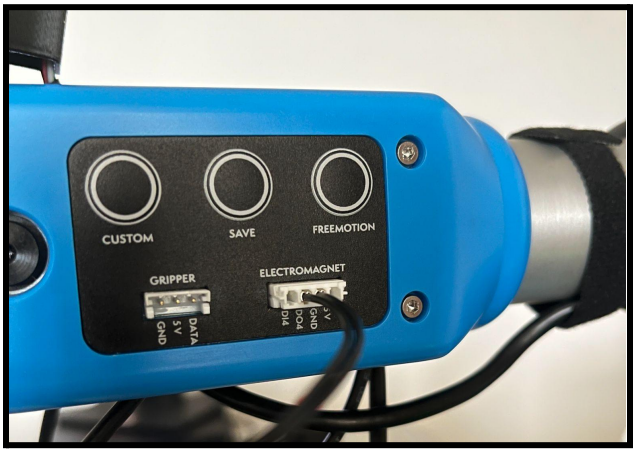


Fig. 10.1. Electromagnet connections end effector

Fig. 10.2. Electromagnet

10.E Steps

- Import all public names(functions, objects etc.) from the “pyniryo2” module into the given file.
- Establish connection to the robotic arm with the given IP address and set the robot calibration to auto.
- Setup the electromagnet to pin DO4 with command “robot.tool.setup_electromagnet(PinID.DO4)”.
- Determine the pick position(as shown in Fig. 10.3.) and use Niryo Studio to get the x,y,z coordinates as well as Roll,Pitch and Yaw and feed those coordinates into the “pick_from_pose” function in the order x,y,z,roll,pitch,yaw.

Fig. 10.3. Pick Position

- Determine an intermediate position(as shown in Fig. 10.4.) for the arm after it picks up the object so as to prevent it from colliding with the gravity feeder after picking up the object or dropping the object. Use the “move_pose” function for this.

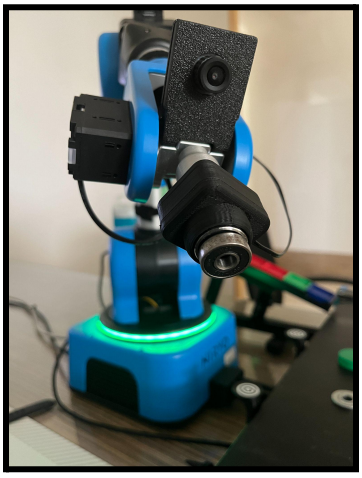


Fig. 10.4. Intermediate position

- Determine a DROP/PLACE position and use the “place_from_pose” function to place the object at those particular coordinates(as shown in Fig. 10.5.).

```
conveyor_id = robot.conveyor.set_conveyor()
pick=PoseObject(x=0.230,y=-0.185,z= 0.072,roll = -2.084,pitch = 1.466,yaw = -2.078)
inter=PoseObject(x=0.224,y = -0.171,z = 0.200,roll = -1.309,pitch =1.411,yaw = -1.362)
place=PoseObject(x = -0.013,y = -0.257,z = 0.024,roll = -1.863,pitch = 1.494,yaw=-3.089)
```

Fig. 10.5. Place position

- Read the input from the IR sensor at pin DI5 using “robot.io.digital_read(PinID.DI5)”
- In a loop run the conveyor and check if the IR sensor sends a False value.

```
    robot.conveyor.run_conveyor(conveyor_id,speed=50, direction=ConveyorDirection.FORWARD)
    if(robot.io.digital_read(PinID.DI5)==False):
```

- If the IR sensor value is False move the arm to pick position and energize the electromagnet. “robot.tool.activate_electromagnet(PinID.DO4)”
- Move to the PLACE position via the intermediate position and de-energize the electromagnet. robot.tool.deactivate_electromagnet(PinID.DO4)
- Use a loop for multiple iterations. In this case a while loop has been used and there will be a total of 10 iterations.
- After the execution comes out of the loop the arm will return to its home position.(This can be done via the move_to_home() function)
- The connection to the arm can now be disabled using the “end” function

```
from pyniryo2 import *
workspace_name="Conveyor_Workspace"
robot=NiryoRobot("169.254.200.200") #Establishing connection with the robotic arm

robot.arm.calibrate_auto() # Automatic calibration of the robotic arm
robot.tool.setup_electromagnet(PinID.DO4) # Equiping the given tool/end effector
conveyor_id = robot.conveyor.set_conveyor()
pick=PoseObject(x=0.230,y=-0.185,z= 0.072,roll = -2.084,pitch = 1.466,yaw = -2.078)
inter=PoseObject(x=0.224,y = -0.171,z = 0.200,roll = -1.309,pitch =1.411,yaw = -1.362)
place=PoseObject(x = -0.013,y = -0.257,z = 0.024,roll = -1.863,pitch = 1.494,yaw=-3.089)
k=0
robot.io.digital_read(PinID.DI5)
while(k<10):
    robot.conveyor.run_conveyor(conveyor_id,speed=50, direction=ConveyorDirection.FORWARD)
    if(robot.io.digital_read(PinID.DI5)==False):
        robot.conveyor.stop_conveyor(conveyor_id)
        robot.arm.move_pose(pick)
        robot.tool.activate_electromagnet(PinID.DO4)
        robot.arm.move_pose(inter)
        robot.arm.move_pose(place)
        robot.tool.deactivate_electromagnet(PinID.DO4)
        robot.wait(0.2)
        robot.arm.move_pose(inter)
        k+=1
    robot.conveyor.run_conveyor(conveyor_id,speed=50, direction=ConveyorDirection.FORWARD)
    robot.wait(0.7)
robot.conveyor.stop_conveyor(conveyor_id)#stopping the conveyor
robot.conveyor.unset_conveyor(conveyor_id)#disconnecting the conveyor
robot.tool.deactivate_electromagnet(PinID.DO4)#de-energizing the electromagnet
robot.arm.move_to_home_pose()#moving the arm to a home position
robot.end()#terminating the connection to the robotic arm
```

Full code for electromagnet pick and place

Chapter 11: 2D Shape Drawing

11.A Objective

The objective of this experiment is to draw 2D shapes using the robotic arm. The code is menu driven and will allow the user to choose from 5 different shapes which will be drawn by executing the precoded trajectories.

11.B Outcomes

By performing this experiment, the user will learn how to:-

- Learn about trajectories of the arm in python
- Write a menu driven program in python

11.C Steps

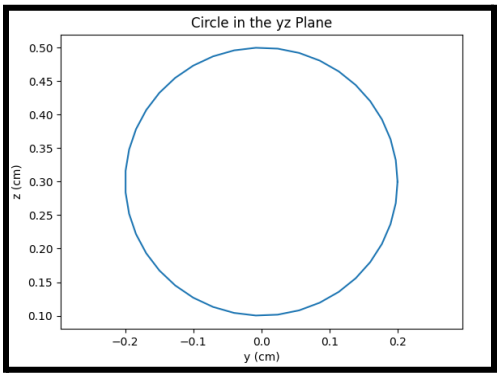
- Import all public names(functions, objects etc.) from the “pyniryo2” module into the given file.
- Establish connection to the robotic arm with the given IP address and set the robot calibration to auto.
- List out all the available options. In this case it has been made into a separate function as shown in Fig. 11.1.

```
def display_menu():
    print("Welcome to Shape Drawing Program!")
    print("1. Circle")
    print("2. Square")
    print("3. Rectangle")
    print("4. Hexagon")
    print("5. Triangle")
    print("6. Exit")
```

Fig. 11.1. Starting display menu

- For different shapes, design a trajectory based on the x,y,z and the quaternion values for roll pitch and yaw.

Note:- For a circle the radius is given and the points are calculated from the cos and sin values.




```python
def option1():
    print("Option 1 selected.")
    radius =0.2 # Radius of the circle in cm
    theta = np.linspace(0, 2*np.pi,40)  # Generate 40 points around the circle
    y = radius * np.cos(theta)  # Calculate y-coordinates
    z = radius * np.sin(theta)  # Calculate z-coordinates
    znew=[]
    for x in z:
        znew.append(x+0.3)
    plt.plot(y,znew)
    plt.xlabel('y (cm)')
    plt.ylabel('z (cm)')
    plt.axis('equal')  # Set equal scaling for x and y axes
    plt.title('Circle in the yz Plane')
    plt.show()
    for val1,val2 in zip(y,z):
        trajectory.append([0.3,val1,val2+0.3, 0., 0., 0., 1.])
    robot.trajectories.execute_trajectory_from_poses(trajectory)
```

For shapes with straight lines the 2 points are given as points for the trajectory.

- The trajectories are executed by the function robot.trajectories.execute_trajectory_from_poses(trajectory).
- Trajectories can also be saved by long pressing the save button on the button panel on the arm and moving the arm manually in the desired trajectory(as demonstrated in Fig. 11.2.).

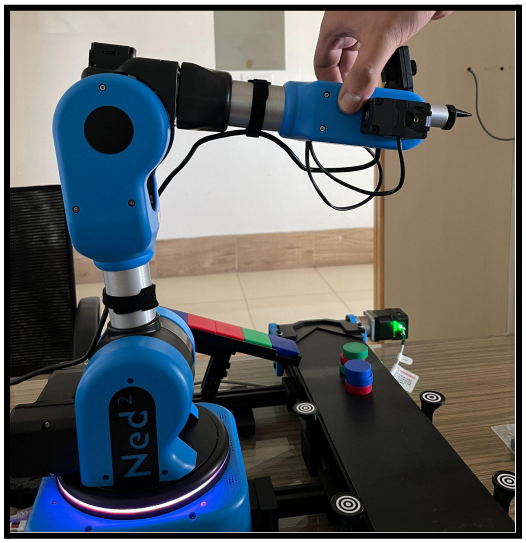


Fig. 11.2 Trajectory saving through save button

- In a While True loop get the input choices and call each function depending on the choice input.

```
while True:
    display_menu()
    choice = input("Enter your choice: ")

    if choice == '1':
        option1()
    elif choice == '2':
        option2()
    elif choice == '3':
        option3()
    elif choice == '4':
        option4()
    elif choice == '5':
        option5()
    elif choice == '6':
        print("Thank you for using the program. Exiting...")
        break
    else:
        print("Invalid choice. Please try again.")
```

- If exit is chosen break out of the loop

```
from pyniryo2 import *
import matplotlib.pyplot as plt
import numpy as np

robot=NiryoRobot("169.254.200.200")
robot.arm.calibrate_auto() # Automatic calibration of the robotic arm
robot.tool.update_tool()


def display_menu():
    print("Welcome to Shape Drawing Program!")
    print("1. Circle")
    print("2. Square")
    print("3. Rectangle")
    print("4. Triangle")
    print("5. Hexagon")
    print("6. Exit")


def option1():
    print("Option 1 selected.")
    trajectory=[]
    radius =0.065 # Radius of the circle in cm
    theta = np.linspace(0, 2*np.pi,60)  # Generate 40 points around the circle
    y = radius * np.cos(theta)  # Calculate y-coordinates
    z = radius * np.sin(theta)  # Calculate z-coordinates
    znew=[]
    for x in z:
        znew.append(x+0.3)
    plt.plot(y,znew)
    plt.xlabel('y (cm)')
    plt.ylabel('z (cm)')
    plt.axis('equal')  # Set equal scaling for x and y axes
    plt.title('Circle in the yz Plane')
    plt.show()
    for val1,val2 in zip(y,z):
        trajectory.append([0.3,val1,val2+0.3, 0., 0., 0., 1.])
    robot.trajectories.execute_trajectory_from_poses(trajectory)


def option2():
    print("Option 2 selected.")
    trajectory = [[0.3, 0.1, 0.3, 0., 0., 0., 1.],
                  [0.3, -0.1, 0.3, 0., 0., 0., 1.],
                  [0.3, -0.1, 0.5, 0., 0., 0., 1.],
                  [0.3, 0.1, 0.5, 0., 0., 0., 1.],
                  [0.3, 0.1, 0.3, 0., 0., 0., 1.]]
    robot.trajectories.execute_trajectory_from_poses(trajectory)
```

```
def option3():
    print("Option 3 selected.")
    trajectory = [[0.3, 0.1, 0.3, 0., 0., 0., 1.],
                  [0.3, -0.1, 0.3, 0., 0., 0., 1.],
                  [0.3, -0.1, 0.4, 0., 0., 0., 1.],
                  [0.3, 0.1, 0.4, 0., 0., 0., 1.],
                  [0.3, 0.1, 0.3, 0., 0., 0., 1.]]
    robot.trajectories.execute_trajectory_from_poses(trajectory)


def option4():
    print("Option 4 selected.")
    trajectory = [[0.3, 0.1, 0.3, 0., 0., 0., 1.],
                  [0.3, -0.1, 0.3, 0., 0., 0., 1.],
                  [0.3, -0.1, 0.4, 0., 0., 0., 1.],
                  [0.3, 0.1, 0.3, 0., 0., 0., 1.]]
    robot.trajectories.execute_trajectory_from_poses(trajectory)


def option5():
    print("Option 5 selected.")
    trajectory = [[0.3, 0.1, 0.3, 0., 0., 0., 1.],
                    [0.3, 0., 0.2, 0., 0., 0., 1.],
                  [0.3, -0.1, 0.3, 0., 0., 0., 1.],
                  [0.3, -0.1, 0.4, 0., 0., 0., 1.],
                    [0.3, 0.0, 0.5, 0., 0., 0., 1.],
                    [0.3, 0.1, 0.4, 0., 0., 0., 1.],
                  [0.3, 0.1, 0.3, 0., 0., 0., 1.]]
    robot.trajectories.execute_trajectory_from_poses(trajectory)


while True:
    display_menu()
    choice = input("Enter your choice: ")

    if choice == '1':
        option1()
    elif choice == '2':
        option2()
    elif choice == '3':
        option3()
    elif choice == '4':
        option4()
    elif choice == '5':
        option5()
    elif choice == '6':
        print("Thank you for using the program. Exiting...")
        break
    else:
        print("Invalid choice. Please try again.")

robot.arm.move_to_home_pose()#moving the arm to a home position
robot.end()#terminating the connection to the robotic arm
```

Full code for shape drawing

11.E Errors one may expect

Error:- The robotic arm makes a beeping sound and does not move in the desired trajectory.
Cause:- Coordinates given are outside the maximum reach of the arm
Solution:- Check if the robot can move to the coordinates that are being assigned by feeding the values in one at a time and observing if the robot moves.

Error:- The robotic arm stops at a particular point while forming a circle and readjusts the joints.
Cause:- The roll pitch and yaw at the particular point cannot be matched as joint 3 can move only 180 degrees and not 360 degrees.
Solution:- The maximum radius value for a smooth circle is 0.065m. Anything more than this value causes the arm to readjust itself at certain points and tilting the end effector at that point

Chapter 12: Joint Control through Keyboard

12.A Objective

The objective of this experiment is to control the robotic arm through a keyboard. The keypresses jog the joints similar to a joystick .

12.B Outcome

By performing this experiment, the user will learn how to:-

- Keyboard input through python
- Jogging joints of the robotic arm

12.C Steps

Import all public names(functions, objects etc.) from the “pyniryo2” module into the given file.

- Establish connection to the robotic arm with the given IP address and set the robot calibration to auto.
- Set the arm jog controls to true to enable the jogging of joints.
- Keeping in mind the max and min angles in radians of each joint which can be determined through Niryo Studio(displayed in Fig. 12.1.) assign keys using the function “keyboard.is_pressed()” to jog each joint back and forth by giving increments or decrements to the function robot.arm.jog_joints().

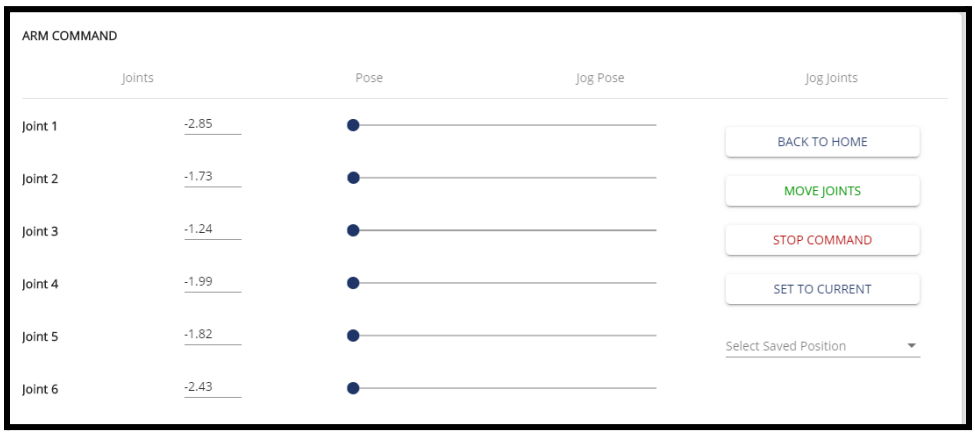


Fig. 12.1. Joints met as 0.1 and -0.1 radians respectively.

```
joint_angles = robot.arm.get_joints()
if keyboard.is_pressed('a') and joint_angles[0]<=2.7:
    robot.arm.jog_joints([0.1, 0.0, 0.0, 0.0, 0.0,0.0])
elif keyboard.is_pressed('d')and joint_angles[0]>=-2.7:
    robot.arm.jog_joints([-0.1, 0.0, 0.0, 0.0, 0.0,0.0])
elif keyboard.is_pressed('w') and joint_angles[1]<=0.4:
    robot.arm.jog_joints([0.0, 0.1, 0.0, 0.0, 0.0,0.0])
elif keyboard.is_pressed('s')and joint_angles[1]>=-1.6:
    robot.arm.jog_joints([0.0, -0.1, 0.0, 0.0, 0.0,0.0])
elif keyboard.is_pressed('i')and joint_angles[2]<=1.37:
    robot.arm.jog_joints([0.0, 0.0, 0.1, 0.0, 0.0,0.0])
elif keyboard.is_pressed('k')and joint_angles[2]>=-1.14:
    robot.arm.jog_joints([0.0,0.0,-0.1, 0.0, 0.0,0.0])
elif keyboard.is_pressed('j')and joint_angles[3]<=1.89:
    robot.arm.jog_joints([0.0, 0.0, 0.0, 0.1, 0.0,0.0])
elif keyboard.is_pressed('l')and joint_angles[3]>=-1.89:
    robot.arm.jog_joints([0.0,0.0,0.0,-0.1, 0.0,0.0])
elif keyboard.is_pressed('up_arrow')and joint_angles[4]<=1.72:
    robot.arm.jog_joints([0.0, 0.0, 0.0, 0.0, 0.1,0.0])
elif keyboard.is_pressed('down_arrow')and joint_angles[4]>=-1.72:
    robot.arm.jog_joints([0.0,0.0,0.0,0.0,-0.1,0.0])
elif keyboard.is_pressed('right_arrow')and joint_angles[5]<=2.33:
    robot.arm.jog_joints([0.0, 0.0, 0.0, 0.0, 0.0,0.1])
elif keyboard.is_pressed('left_arrow')and joint_angles[4]>=-2.33:
    robot.arm.jog_joints([0.0,0.0,0.0,0.0,0.0,-0.1])
elif keyboard.is_pressed('q'):
    break
```

Else if ladder for keybindings

- If ‘q’ is pressed the execution breaks out of the loop and the arm returns to its home position and the connection to the arm is disestablished.

```
from pyniryo2 import *
import keyboard
robot=NiryoRobot("169.254.200.200")
robot.arm.calibrate_auto() # Automatic calibration of the robotic arm
robot.tool.update_tool()
robot.arm.set_jog_control(True)

while True:

    joint_angles = robot.arm.get_joints()
    if keyboard.is_pressed('a') and joint_angles[0]<=2.7:
        robot.arm.jog_joints([0.1, 0.0, 0.0, 0.0, 0.0,0.0])
    elif keyboard.is_pressed('d')and joint_angles[0]>=-2.7:
        robot.arm.jog_joints([-0.1, 0.0, 0.0, 0.0, 0.0,0.0])
    elif keyboard.is_pressed('w') and joint_angles[1]<=0.4:
        robot.arm.jog_joints([0.0, 0.1, 0.0, 0.0, 0.0,0.0])
    elif keyboard.is_pressed('s')and joint_angles[1]>=-1.6:
        robot.arm.jog_joints([0.0, -0.1, 0.0, 0.0, 0.0,0.0])
    elif keyboard.is_pressed('i')and joint_angles[2]<=1.37:
        robot.arm.jog_joints([0.0, 0.0, 0.1, 0.0, 0.0,0.0])
    elif keyboard.is_pressed('k')and joint_angles[2]>=-1.14:
        robot.arm.jog_joints([0.0,0.0,-0.1, 0.0, 0.0,0.0])
    elif keyboard.is_pressed('j')and joint_angles[3]<=1.89:
        robot.arm.jog_joints([0.0, 0.0, 0.0, 0.1, 0.0,0.0])
    elif keyboard.is_pressed('l')and joint_angles[3]>=-1.89:
        robot.arm.jog_joints([0.0,0.0,0.0,-0.1, 0.0,0.0])
    elif keyboard.is_pressed('up_arrow')and joint_angles[4]<=1.72:
        robot.arm.jog_joints([0.0, 0.0, 0.0, 0.0, 0.1,0.0])
    elif keyboard.is_pressed('down_arrow')and joint_angles[4]>=-1.72:
        robot.arm.jog_joints([0.0,0.0,0.0,0.0,-0.1,0.0])
    elif keyboard.is_pressed('right_arrow')and joint_angles[5]<=2.33:
        robot.arm.jog_joints([0.0, 0.0, 0.0, 0.0, 0.0,0.1])
    elif keyboard.is_pressed('left_arrow')and joint_angles[4]>=-2.33:
        robot.arm.jog_joints([0.0,0.0,0.0,0.0,0.0,-0.1])
    elif keyboard.is_pressed('q'):
        break

robot.arm.move_to_home_pose()#moving the arm to a home position
robot.end()#terminating the connection to the robotic arm
```

Full code for keyboard control