Expt. No. 1 Page No. 1

Expt. Name PROGRAM TO ADD A ROBOT TO SIMULATION WORLD, ADD A PYTHON SCRIPT BEHAVIOUR TO ROBOT COMPONENT, PYTHON SCRIPT INITIALIZATION, ADD STATEMENTS, ADD MOTION STATEMENTS USING PYTHON API Date :

The fundamental challenge of all robotics is this : It is impossible to ever know the true state of the environment. Robot control software can only guess the state of the real world based on measurements returned by its sensors. It can only attempt to change the state of the real world through the generation of control signals.

The Programmable Robot Simulator :

The simulator is built is written in Python and very cleverly dubbed robot simulator. You can find v1.0.0 on GitHub. It does not have a lot of bells and whistles but it is built to do one thing, provide an accurate simulation of a mobile robot and give an aspiring roboticist a simple framework for practicing robot software programming.

API :

In robot simulator, the separation between the robot and the physical world is embodied by the file, which defines the entire API for interacting with the file.

The Simulator :

As you would use a real robot in the real world without paying too much attention to the laws of physics involved, you can ignore how the robot is simulated and just skip directly to how the controller software is programmed, since it will be almost the same between the real world and a simulation.

Expt. No. _____

Page No. 2

Expt. Name _____

Date : _____

The file is a Python class that represents the simulated world, with robots and obstacles inside.

RESULT:

Thus the program to add a robot to simulation world, add a Python script behaviour to robot component, python script initization, add statements, add motion statements using python API was studied.

Expt. No. 2 Page No. 3

Expt. Name PROGRAM TO ADD AND EDIT ROUTINES, STATEMENTS AND POSITIONS IN A ROBOT PROGRAM USING PYTHON API Date :

AIM:

To implement program to add and edit routines, statements and positions in a robot program using Python API.

SOFTWARE USED:

Robodk

PROCEDURE:

- Open robodk
- Create a new project.
- Search for the following components in Robodk.
- Make the following connections.
- Run the following code.

PROGRAM:

```
from vcScript import *

comp = getComponent()
robotExecutor = comp.FindBehavioursByType("rRobotExecutor")[0]
robotProgram = robotExecutor.Program
mainRoutine = robotProgram.MainRoutine

mainRoutine.clear()
for subroutine in robotProgram.Routines:
    robotprogram.deleteRoutine(subroutine)

statement = mainRoutine.addStatement(VC_STATEMENT_PRINT)
statement.Message = "Hello World"
```

```
statement = mainRoutine.addStatement (VC_Statement_DELAY, 0)
statement.Delay = 2.0

statement = mainRoutine.addStatement (VC_STATEMENT_DELAY)
position = statement.Position[0]

app = getApplication()
cube = app.findComponent("Cube")
pos_matrix = cube.positionMatrix
cube_center = cube.BoundCenter
pos_matri.translateRel(cube_center.X, ceibe_center.Y, cube_center.Z*2)
pos_matrix.rotateRelY(180.0)
position.PositionInWorld = pos_matrix

subroutine = robotProgram.findRoutine("Example")
if subroutine:
    subroutine.clear()
if not subroutine:
    subroutine = robotProgram.addRoutine("Example")

statement = subroutine.addStatement(VC_STATEMENT_PTPMOTION)
position = statement.Position[0]
position.PositionInWorld = pos_matrix
```

RESULT:

Thus, the simulation of to add and edit routines, statements and positions in a robot program using Python API have been executed and verified successfully.

Expt. No. 3 Page No. 5

Expt. Name PROGRAM TO MANIPULATE THE JOINTS OF A ROBOT USING ITS CONTROLLER AND DOF OBJECTS OF NODES Date :

AIM:

To manipulate the joints of a robot using its controller and DOF objects of nodes.

SOFTWARE:

Robodk

PROCEDURE:

- Open robodk
- Create a new project
- Search for the following components in robodk
- Make the following connections
- Run the following code.

PROGRAM:

```
from vc Script import *
comp = get Component ( )
robot Controller = comp. Find Behaviours By Type (VC_ROBOT CONTROLLER)
                                                    [0]

internal_joints = [ ]
for joint in robot Controller. Joints :
    if joint. External Controller == None :
        internal_joints. append (joint)

external_joints = [ ]
for joint in robot Controller. Joint :
    if joint. External Controller != None :
        external_joints. append (joint)
```

```
def getInternalJointsInRobot(rc):
    return list(filter(lamda n: n.ExternalController==None,
                                    rc.Joints))

def getExternalJointsInRobot(rc):
    return list(filter(lamda n: n.ExternalController!=None,
                                    rc.Joints))

def getAllNodesInComponent(comp, nodes=[], includeRootNode=
                                    false):
    if not nodes:
        nodes=[]

    if includeRootNode==True:
        nodes.append(comp)
    for node in comp.Children:
        if node.Component==comp.Component:
            nodes.append(node)
            if node.Children:
                getAllNodesInComponent(node, nodes)
    return nodes
```

RESULT:

Thus the simulation of manipulation of joints of robots using its controller and DOF objects of nodes has been executed and verified successfully.

Expt. No. 4 Page No. 7

Expt. Name PROGRAM TO TEACH A ROBOT TO PICK PARTS FROM A PALLET AND PLACE THEM ON A CONVEYOR USING PYTHON API. Date :

AIM:

To control and automate a robot using Python API

SOFTWARE USED:

RoboDK

PROCEDURE:

- Open RoboDK
- Create a new Project
- Search the following components in robodk
- Make the appropriate connections
- Run the code.

PROGRAM:

```
from robodk import *
from robolink import *
RL = Robolink()
Robot = RL.Item("Mitsubishi RV-8CRL")
Home = RL.Item("Home")
#V1 = RL.Item("V1")
#V2 = RL.Item("V2")
tool = RL.Item("gripper RobotiQ 85 opened")
App_A = RL.Item("App_A")
Pick_A = RL.Item("Pick_A")
App_B = RL.Item("App_B")
Pick_B = RL.Item("Pick_B")
robot.moveJ(home)
```

Expt. No. _____ Page No. 8

Expt. Name _____ Date : _____

```
while true:
    robot.moveJ(App_A)
    robot.moveL(Pick_A)
    tool.AttachClosest()
    robot.moveL(App_A)
    robot.moveJ(Home)
    robot.moveJ(App_B)
    robot.moveL(Pick_B)
    tool.DetachAll()
    robot.moveL(App_B)
    robot.moveL(Pick_B)
    tool.AttachClosest()
    robot.moveL(App_B)
    robot.moveJ(home)
    robot.moveJ(App_A)
    robot.moveL(Pick_A)
    tool.DetachAll()
    robot.moveL(App_A)
    robot.moveJ(home)
```

RESULT:

The robot has been successfully taught to pick part from a pallet and place them on a conveyor / stationary using Python API.

Expt. No. 5

Page No. 9

Expt. Name PROGRAM TO IMPLEMENT A ROBOT TO PICK MOVING PARTS AND THEN PLACE THEM ON OTHER MOVING PART USING PYTHON API.

Date :

AIM :

To teach a robot to pick parts from a moving conveyor and place them on another moving conveyor belt using Python API.

SOFTWARE REQUIRED :

Robodk

PROCEDURE :

- Open robodk
- Create a new project
- Search for the following components
- Make the following connections.
- Run the programme

CODE :

Input Belt -

```
From robodk import *
From robolink import *
RL = Robolink ()
Con = RL. Item ('Conveyor Belt in')
Home = RL. Item ('Home 1')
End = RL. Item ('End 1')
Con. moveJ (End)
```

Expt. No. _____ Page No. 10

Expt. Name _____ Date : _____

Output Belt -

```
from Robodk import *
from Robodk import *
RL = Robolink()
Con = RL.Item ('Conveyor Belt out')
Home = RL.Item ('Home 2')
End = RL.Item ('End2')
con.moveJ (End2)
```

Pick and Place -

```
from robodk import *
from robolink import *
RL = Robolink ()
robot = RL.Item ('UR5e')
tool = RL.Item ('Robotiq 2F-85 Gripper (mechanism)')
Home = RL.Item ('Home')
Bottle = RL.Item ('Bottle')
Con_out = RL.Item ('Conveyor Belt out')
Pick = RL.Item ('Pick')
Place = RL.Item ('Place')
robot.moveJ (Pick)
Bottle.setParent (tool)
robot.moveJ (home)
robot.moveJ (place)
Bottle.setParent (con_out)
robot.moveJ (home)
```

RESULT:

Robot has been successfully taught to move object from moving conveyor onto another moving conveyor using Python API.

Expt. No. 6 Page No. 11

Expt. Name PROGRAM TO CONTROL AND AUTOMATE THE ROBOT USING PYTHON API. Date :

AIM:

To control and automate a robot using Python API

SOFTWARE USED:

Robodk

PROCEDURE:

- Open RoboDK
- Create a new project
- Search the following components
- Make the appropriate connections
- Run the code in Robodk

PROGRAM:

```
from robodk import *
from robolink import *
RL = Robolink()
Robot = RL.Item("Mitsubishi RV-8CRL")
Home = RL.Item("Home")
X1 = RL.Item('X1')
X2 = RL.Item('X2')
Y1 = RL.Item('Y1')
Y2 = RL.Item('Y2')
Z1 = RL.Item('Z1')
Z2 = RL.Item('Z2')
```

Expt. No. _____ Date : _____

Expt. Name _____

```
while True:
    Robot.MoveJ(X1)
    Robot.MoveJ(X2)
    Robot.MoveJ(Home)
    Robot.MoveJ(Y1)
    Robot.MoveJ(Y2)
    Robot.MoveJ(Home)
    Robot.moveJ(Z1)
    Robot.moveJ(Z2)
    Robot.moveJ(home)
```

RESULT: The process of controlling and automating a robot has been successfully simulated using python API

Expt. No. 7 Page No. 13

Expt. Name PROGRAM TO CHECK IF A GIVEN SEQUENCE OF MOVES FOR A ROBOT IS CIRCULAR OR NOT Date :

AIM:

To write a program to check if a given sequence of moves for a robot is circular or not using python API.

SOFTWARE:

Robodk

PROCEDURE:

- Open robodk
- Create a new project
- Search for the following components in robodk
- Make the appropriate connections
- Run the code.

PROGRAM:

```
from robodk import *
from robolink import *
RL = Robolink()
robot = RL.Item("Mitsubishi RV-8CRL")
tool = RL.Item("RobotiQ 2F-85 gripper (closed)")
home = RL.Item("Home")

A = RL.Item('A')
B = RL.Item('B')
C = RL.Item('C')
D = RL.Item('D')
robot.MoveJ(home)
```

Expt. No. _____ Page No. 14

Expt. Name _____ Date : _____

```
while True:
    robot.MoveJ(A)
    robot.MoveJ(B)
    robot.MoveJ(C)
    robot.MoveJ(D)
    robot.MoveJ(A)
    robot.MoveJ(home)
```

RESULT:

The program to check if robot's movements are circular or not has been successfully executed.

Expt. No. 8 Page No. 15

Expt. Name PROGRAM FOR WALKING ROBOT SIMULATION IN PYTHON Date :

AIM:

To write a program for walking robot with following conditions -

- -2 to turn left 90 deg.
- -1 to turn right 90 deg.
- Any value from 1 to 9 to move forward n units.
- There are some grid squares that are obstacles.

SOFTWARE USED:

Robodk

PROCEDURE:

- Open robodk
- Create a new project
- Search for the components
- Make the appropriate connections
- Run the code.

PROGRAM:

```
class solution:
    def_position_offset = [(0,1), (1,0), (0,-1), (-1,0)]
    obstacles = set(map(tuple.object))
    x,y, direction, move_direction = 0,0,0,0
    for command in commands:
        if command == -2: direction = (direction - 1) %4
        elif command == -1: direction = (direction + 1) %4
        else:
            x_off, y_off = position_offset [direction]
```

Expt. No. _____ Page No. 16

Expt. Name _____ Date : _____

```
while command:
    if (x+x_off, y+y_off) not in obstacles:
        x+ = x_off
        y+ = y_off
    command- = 1
    move_direction = max(move_direction, x**2 + y**2)
return move_direction
ob = solution()
print(ob.robotSim([4,-1,4,-2,4], [[2,4]]))
```

RESULT:

Thus the simulation of a robot walking has been successfully executed.

Expt. No. 9 Page No. 17

Expt. Name PROGRAM TO CHECIC ROBOT CAN REACH TARGET BY KEEP MOVING ON VISITED SPOTS IN PYTHON Date :

AIM:

To write a program to check if robot can reach the target by moving on visited spots alone using Python API.

SOFTWARE:

Robodk

PROCEDURE:

- Open Robodk
- Create a new project
- Search the following components in robodk
- Make the appropriate connections
- Run the code.

PROGRAM:

```
Class Solution:
    def solve(self, moves, coord):
        ny = nx = 0
        l = {(0,0)}
        for k in moves:
            if k == 'N':
                while (nx, ny) in l:
                    ny += 1
            elif k == 'S':
                while (nx, ny) in l:
                    ny -= 1
            elif k == 'E':
                while (nx, ny) in l:
                    nx += 1
```

```
        else :
            while (nx, ny) in l :
                nx -= 1
            l.add ((nx, ny))
        return coord [0] == nx and coord [1] == ny
ob = solution ()
moves = ['N', 'N', 'E', 'N', 'W', 'S']
coord = [0, 1]
print (ob. solve (moves, coord))
```

Input

```
['N', 'N', 'E', 'N', 'W', 'S'], [0, -1]
```

RESULT:

Simulation to check if robot movement can reach target by moving through visited points has been successfully executed in Python API.

Expt. No. 10 Page No. 19

Expt. Name PROGRAM TO CREATE A ROBOT CLASS THAT CONTAINS METHODS ABOUT MOVEMENTS AND SAVING THE MOVEMENT INSTRUCTIONS ON A TEXT FILE. A ROBOT OBJECT CAN LOAD THE TEXT FILE AND EXECUTE THE MOVEMENTS, A ROBOT OBJECT FINALLY GIVES THE UPDATED POSITION AND DISTANCE TRAVELLED Date :

AIM :

To implement program to create a robot class that contains methods about movements.

SOFTWARE :

Robodk

PROCEDURE :

1) Open Robodk
2) Create a new project
3) Search for the following components in Robodk
4) Make the following connections
5) Run the following code.

PROGRAM :

```
def findPosition(move):
    l = len(move)
    countUp, countDown = 0,0
    countLeft, countRight = 0,0
    for i in range(l):
        if(move[i]=='U'):
            countUp+=1
        elif(move[i]=='D'):
            countDown+=1
        elif(move[i]=='L'):
            countLeft+=1
```

```
        elif (move [i] == 'R'):
            countRight += 1
    print ("Final position : (", (countRight - countLeft), ", ",
                                  (countUp - countDown), ")")

    print ("Distance Travelled : ", l, "units")
if __name__ == '__main__':
    move = "UDDLLRUUUDUURUDDUULLDRRRR"
    finalPosition (move)
```

OUTPUT :

Final Position : (2,3)

Distance Travelled : 25 units

RESULT:

The robot movement has been successfully saved and simulated.