Controlling your robot with Python

October 9, 2018

1 Preparing the environment

Controlling a robot is essentially having a way to transmit a script to the device (robot). For that, we usually employ a client-server model. The robot is the server to which a client sends commands or a full program. We are going to setup a VREP server and a python client.

1.1 Basic Tools

- Download and install a scientific python distribution: Anaconda or Python (x,y), or any installation with `scipy,numpy,matplotlib` and other major libraries. Spyder is another useful tool as it is an IDE that helps debug your python programs. Spyder is included with the tools mentioned above.
- From that install, open Spyder (this is optional, but helpful). Alternatively, install `numpy,scipy and matplotlib` in your python environment and use your favorite IDE.
- Open your bubbleRob scene on VREP. We will work with it.

1.2 Files setup

The first thing we need is to set up the environment using your file explorer/finder.

- Decide on what will your working folder (directory). Create it if necessary. This is where your scripts will reside.
- Navigate to your V-REP installation folder
- Navigate into the `programming/remoteApiBindings/python/python`
- Copy all the `.py` files into your working folder
- Navigate into the `programming/remoteApiBindings/lib/lib/`
- On some V-REP installations you may have to navigate to your operating system's directory (Windows, Linux or Mac OS).
- Copy the `.dylib` or `.dll` file to your working directory.

1.3 Setting up a Server

Now, on VREP your scene needs to start a server so that the python scripts can connect to it. To start the server you have to add a threaded script to an object in your scene that will start the server.

- Open your bubbleRob scene
- Click on any of the cylinders you used as obstacles
- Click on Add→Associated Child Script*rightarrow*Threaded Script
- Open the script by double clicking on the "file" icon next to the cylinder
- Find the function `sysCall_threadmain()`
- Write the following as the only line in that method: `simExtRemoteApiStart(19999)`

Now, as soon as the scene is played, a server will be listening on port 19999. Before we start the server we need to disable the existing script

- Open the scripts window (click on the "page" on the left hand side icons)
- Select the script on bubbleRob
- Click on "Disable"
- Close the scripts window. You will see an "x" next to the script for bubbleRob

Now, start the scene. That will start the server.

1.4 Setting up the Client

Switch to Python (Spyder, Terminal or your favourite IDE). Open a new file and immediately save it to your working directory as `bubbleRob_control.py`

Write the following python code in the file.:

```
import vrep # access all the VREP elements
vrep.simxFinish(-1) # just in case, close all opened connections
clientID=vrep.simxStart('127.0.0.1',19999,True,True,5000,5) # start a
    connection
if clientID!=-1:
    print ("Connected to remote API server")
else:
    print("Not connected to remote API server")
    sys.exit("Could not connect")
```

And execute the program. You should see "Connected to remote API server" on the screen.

You have effectively established communication with the scene.

2 Controling the Robot

We will make the robot move by accessing its motors. For this we need a handle to control each individual motor.

The function `vrep.simxGetObjectHandle` will return an error code and a handle to an object. Add the following to the code to obtain a handle to both motors of the robot.

```
err_code,l_motor_handle = vrep.simxGetObjectHandle(clientID,"
    bubbleRob_leftMotor", vrep.simx_opmode_blocking)
err_code,r_motor_handle = vrep.simxGetObjectHandle(clientID,"
    bubbleRob_rightMotor", vrep.simx_opmode_blocking)
```

The parameters to the functions are specified in the Python remote API for VREP at http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsPython.htm. In this case, the `vrep.simxGetObjectHandle`, it needs a handle to the server `clientID`, the name of the object, and a mode (in this case `simx_opmode_blocking`)

Now that we have handles to the motors, we can make them go with the following code:

```
err_code = vrep.simxSetJointTargetVelocity(clientID,l_motor_handle,1.0,
    vrep.simx_opmode_streaming)
err_code = vrep.simxSetJointTargetVelocity(clientID,r_motor_handle,1.0,
    vrep.simx_opmode_streaming)
```

Switch to VREP and watch bubbleRob go. You can stop your simulation if bubbleRob goes out of the scene or gets trapped.

3 Sensing

Sensors work by detecting objects at a given point. They provide the information to check the distance and coordinates of the detected object with respect to the sensor.

Sensors need to be initialized (first detection) and then used over and over again. To initialize a sensor we call the `simxReadProximitySensor` function. Notice the `simx_opmode_streaming` at the end. This function returns several variables.

```
err_code,ps_handle = vrep.simxGetObjectHandle(clientID,"
    bubbleRob_sensingNose", vrep.simx_opmode_blocking)

err_code,detectionState,detectedPoint,detectedObjectHandle,
    detectedSurfaceNormalVector=vrep.simxReadProximitySensor(clientID,
    ps_handle,vrep.simx_opmode_streaming)
```

You can print them. The `detectedPoint,detectedObjectHandle` and `detectedSurfaceNormalVector` can be used to see if our robot has detected something.

For all subsequent sensing, use

```
err_code,detectionState,detectedPoint,detectedObjectHandle,
    detectedSurfaceNormalVector=vrep.simxReadProximitySensor(clientID,
    ps_handle,vrep.simx_opmode_buffer)
```

Lastly, an important piece of information with proximity sensors is: How far is the object I detect?

One of the returned values of `simxReadProximitySensor` is the detected point relative to the sensor's origin. The norm of that vector tells us the distance from the sensor to the object. To obtain the norm we need to import `numpy` and then apply the norm function like so:

```
  import numpy as np #do this at the top of the program.
  np.linalg.norm(detectedPoint)
```

3.1 Little program to detect stuff and react to it

```
import vrep
import numpy as np
import time
vrep.simxFinish(-1) # just in case, close all opened connections
clientID=vrep.simxStart('127.0.0.1',19999,True,True,5000,5)
print(clientID) # if 1, then we are connected.
if clientID!=-1:
    print ("Connected to remote API server")
else:
    print("Not connected to remote API server")
    sys.exit("Could not connect")

err_code,l_motor_handle = vrep.simxGetObjectHandle(clientID,"
    bubbleRob_leftMotor", vrep.simx_opmode_blocking)
err_code,r_motor_handle = vrep.simxGetObjectHandle(clientID,"
    bubbleRob_rightMotor", vrep.simx_opmode_blocking)

err_code,ps_handle = vrep.simxGetObjectHandle(clientID,"
    bubbleRob_sensingNose", vrep.simx_opmode_blocking)
err_code,detectionState,detectedPoint,detectedObjectHandle,
    detectedSurfaceNormalVector=vrep.simxReadProximitySensor(clientID,
    ps_handle,vrep.simx_opmode_streaming)

t = time.time() #record the initial time

while (time.time()-t)<10: #run for 20 seconds
    sensor_val = np.linalg.norm(detectedPoint)
    if sensor_val < 0.2 and sensor_val>0.01:
        l_steer = -1/sensor_val
    else:
        l_steer = 1.0
    err_code = vrep.simxSetJointTargetVelocity(clientID,l_motor_handle,
    l_steer,vrep.simx_opmode_streaming)
    err_code = vrep.simxSetJointTargetVelocity(clientID,r_motor_handle
    ,1.0,vrep.simx_opmode_streaming)
    time.sleep(0.2)
    err_code,detectionState,detectedPoint,detectedObjectHandle,
    detectedSurfaceNormalVector=vrep.simxReadProximitySensor(clientID,
    ps_handle,vrep.simx_opmode_buffer)
    print (sensor_val,detectedPoint)
vrep.simxStopSimulation(clientID,vrep.simx_opmode_oneshot)
print("Done")
```

4 Seeing With the Eyes of A Robot

For the last portion of this tutorial, we can see what the robot sees with its camera sensor.

First, we need a handle to the vision sensor. We will call it "camera"

```
err_code,camera = vrep.simxGetObjectHandle(clientID,"Vision_sensor",
    vrep.simx_opmode_blocking)
```

Then, as with the proximity sensor, we need to do a first read by calling `simxGetVisionSensorImage`. This function takes in the `clientID`, the handle to the vision sensor, a 0 or a 1 depending on whether the picture is grayscale or color, and a mode.

As with the proximity sensor, the only difference between the first read and the subsequent ones is the mode.

```
err_code,resolution,image = vrep.simxGetVisionSensorImage(clientID,
    camera,0,vrep.simx_opmode_streaming)
```

For the subsequent reads

```
err_code,resolution,image = vrep.simxGetVisionSensorImage(clientID,
    camera,0,vrep.simx_opmode_buffer)
```

The image returned is an array of values. Each pixel is represented by three values (RGB for color and grayscale for non-color).

Therefore, the next step is to try to convert it to something readable, say a three dimensional array where for every pixel (i, j) there's a 3-value array with information about its color.

To do this we need `numpy`. First, to convert the python array (image) into a numpy array (or matrix) which is generally a more efficient array for numpy to work with. Second, we tell numpy to group this array in columns, rows and triplets. That is, we tell `numpy` that our representation of an image is of $ixjx3$ where i and j are the resolution of the screen.

```
    img = np.array(image, dtype = np.uint8)
    img.resize([resolution[0],resolution[1],3])
```

To view the image we need to import `matplotlib.pyplot`. Then, we can issue `mlp.imageshow(img)`, But notice that the image is upside down. That is because the screen reads from bottom to top, but the rendering is done from bottom to top. We need to tell matplotlib that the origin of our image is at the bottom.

```
  import matplotlib.pyplot as mlp
  mlp.imshow(img,origin="lower")
```

5 Conclusion

Now you know how to connect to V-REP with python, move motors and read sensors and images.

6 API Reference

To understand the python API reference, please follow Coppelia Python's reference API. Here you will find all the function calls you can do with your robot. Experiment away!