Lab Session 4

In this lab we will learn to use socket programming for Pi-to-Pi communication and control.

You should work out the following in a group of two.

By this lab session, you will learn:

- How to program a socket server on your Raspberry Pi;
- How to program a socket client on your Raspberry Pi;
- How to use socket to build a communication between two Pi's.

What you will need:

- The basic Raspberry Pi and the usual peripherals x2
- A Sense HAT x2

Part I: Create a server

Sockets provide the mechanism for two computers to communicate over TCP. A client program creates a socket on one end of the communication and attempts to connect that socket on the other end which is set up by a server. Python provides an easy-to-use **socket** module. Elect one of the Pi as the server and do the following:

1. Connect your Pi to the access point.
2. Create a file with the following codes:

```
import socket
PORT = 12345
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(('', PORT))
print('Server created.')
server.listen(1)
con, addr = server.accept()
print('Connected to %s:%s.' % (addr[0], addr[1]))
```

3. Run the file you have just created.
4. The above code snippet created a server on port **12345** and wait for the first client connection. **server.accept()** is a blocking function and thus the code will be blocked on this line until there is an incoming connection. Keep the code running and go on.

Part II: Connect to a server

The other Pi is the client. On this Pi, doing the following:

1. Create another file with the following codes:

```
import socket
SERVER = '<Server IP>'
PORT = 12345
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.connect((SERVER, PORT))
print('Server connected.')
server.close()
```

2. Run the code. You can see the server output some information on the connection.
3. Think why the server and client exit their own programs after the connection.

Part III: Say hello

1. Add the following lines to the end of your server code:

```
data = con.recv(1024)
print(data.decode())
con.send('Greetings from the server.'.encode())
```

2. Add the following lines just before `server.close()` in your client code:

```
server.send('Hello Server!'.encode())
reply = server.recv(1024)
print(reply.decode())
```

3. Run the server script first and then the client script.

Remarks:

1. **recv(1024)** means that the data length is equal to or shorter than 1024 bytes. This number can be changed depending on the application demand.
2. Strings transmitted over internet must be encoded into raw bytes. **encode()** and **decode()** do the work.

Part IV: Add some control

1. Add the following lines to the end of your server code:

```
while True:
   msg = con.recv(1024).decode().split(' ')
   if msg[0] == 'KILL':
       print('Shut down server.')
       con.close()
       server.close()
       break
   elif msg[0] == 'SHOW':
       print(' '.join(msg[1:]))
   else:
       print('Message: %s' % msg)
```

2. Add the following lines just before `server.close()` in your client code:

```
while True:
   msg = input('Input your message: ')
   server.send(msg.encode())
   if msg == 'KILL':
       break
```

3. Run the server script first then the client script. Input the following commands into your client code and see what happens:
    a. SHOW Test!
    b. RANDOM COMMAND
    c. KILL
4. Create a new message processing code block in your server code to respond the **ECHO** command from the client. **ECHO** allows the server to repeat to the client with content exactly the same as the incoming message.
5. Create a new message processing code block in your server code to response to the **EXIT** command from the client. **EXIT** attempts to close the connection but does not quit the server program. You also need to modify the client script accordingly.

Part V: Control the pixel on one Pi by another Pi

1. Add the following lines to the BEGINNING of your server code:

```
from sense_hat import SenseHat
sense = SenseHat()
sense.clear()
```

2. Add the following lines in an appropriate place of your server code to respond to the **MOVE** command. Be careful on the code indentation.

```
elif msg[0] == 'MOVE':
       sense.set_pixel(*map(int, msg[1:6]))
```

3. Modify your client code, as follows, to make it control a pixel on the SenseHATs of both the server and client. Pay attention to the IP address as well as the unfinished code in the joystick handling part.

```
from sense_hat import SenseHat
import pygame
from pygame.locals import *
import socket

sense = SenseHat()
sense.clear()
pygame.init()
pygame.display.set_mode((640, 480))
SERVER = '<Server IP>'
```

```
PORT = 12345
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.connect((SERVER, PORT))
print('Server connected.')
server.send('Hello Server!'.encode())
reply = server.recv(1024)
print(reply.decode())
x, y = 3, 3
while True:
   for event in pygame.event.get():
       if event.type == KEYDOWN and event.key == K_DOWN:
           server.send('MOVE {0} {1} 0 0 0 '.format(x,
y).encode())
           sense.set_pixel(x, y, 0, 0, 0)
           y = (y + 1) % 8
           server.send('MOVE {0} {1} 255 255 255 '.format(x,
y).encode())
           sense.set_pixel(x, y, 255, 255, 255)
       elif event.type == KEYDOWN and event.key == K_UP:
           # Add some lines here
       elif event.type == KEYDOWN and event.key == K_LEFT:
           # Add some lines here
       elif event.type == KEYDOWN and event.key == K_RIGHT:
           # Add some lines here
server.close()
```

4. Run server code first then the client code. Try move the joystick on your client Pi and see what happens.

Assignment

Demonstrate your implementations to the TA before you go.