

# Tutorial for developing applications for the drone challenge

[Introduction](#)

[How to write programs using pymavlink](#)

[pymavlink](#)

[Developing an application](#)

[Utility functions](#)

[Commonly used functionality](#)

[Given a requirement, how to choose the right API?](#)

[Using SITL with Sample applications](#)

[Sample application 1](#)

[Steps for running the sample application](#)

[Sample application 2](#)

[Steps for running the sample application](#)

[Saving sensor data](#)

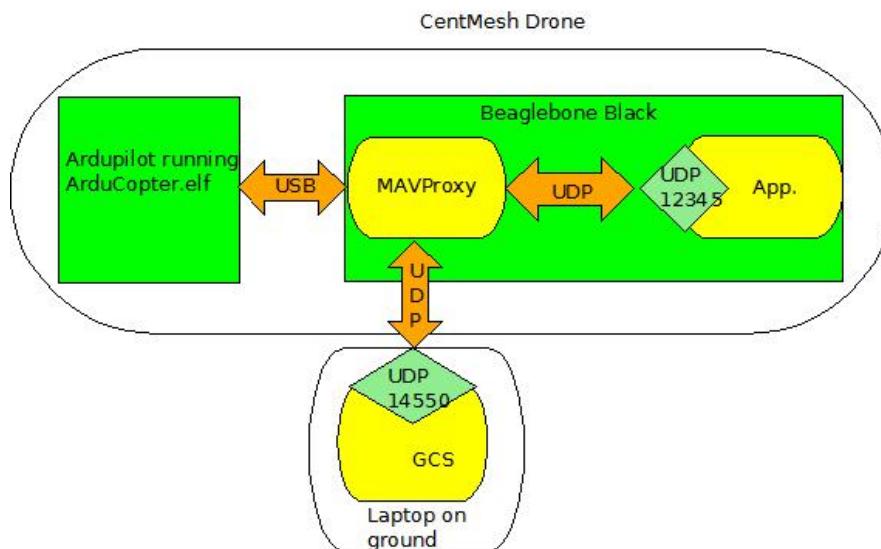
[Working of the sensing platform:](#)

[Running the sensing platform:](#)

Objective: The objective of this tutorial is to help develop and verify sample applications using the programmable platform offered by CentMesh drones.

## Introduction

As given in the following figure, the following components are involved in any application that runs on the CM drones.



The detailed description of each of these components is given in the documentation:

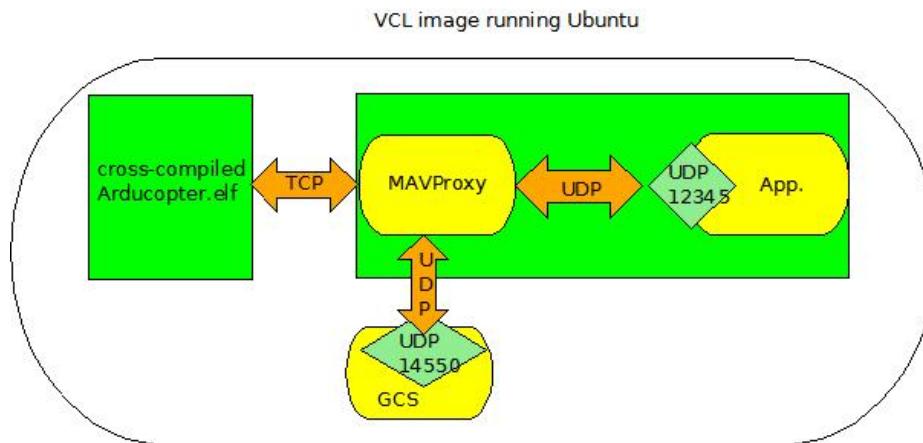
[APM](#)

[Beaglebone black](#)

[GCS](#)

[MAVProxy](#)

Using SITL (documentation [here](#)), the entire setup can be run in a single Virtual Machine or VCL image. The representation (simplified) is displayed by the following figure:



## How to write programs using pymavlink

### pymavlink

Python bindings for MAVLink are provided by [pymavlink](#). Using the interfaces provided, both the cases are possible:

- MAVLink messages can be read from the UAV
- MAVLink Messages can be sent to the UAV.

We are using pymavlink to communicate with the MAV and this tutorial describes its usage. You are welcome to use any other resources (like [this](#)) that you may find useful.

### Developing an application

1) Import 'mavlink\_apm.py': This module can 'decode' and 'encode' (pack and unpack) MAVLink messages. It needs to be included in your python program using 'import':

```
import mavlink_apm
```

In the VCL image, the module is already present under \$HOME/sample\_prog/lib. If you need to generate the same in a VM, please follow the steps mentioned [here](#)

2) 'mavlink\_apm.py' depends on other modules in the same source tree. So, to make sure that the dependencies are resolved, the 'generator' directory (generated by the python code generator that takes xml as its input) and all of its sub-directories are present in same directory where the sample program resides and the following is to be included:

```
sys.path.insert(0, os.path.join(os.path.dirname(os.path.realpath(__file__)), '..'))
```

3) Import the following commonly used python libraries:

```
import re, sys, os, socket, select, time
```

4) The application needs to create a UDP socket for reading messages from MAVProxy.  
Sample code:

```
HOST = ""  
mavproxy_port = 12345  
# Create a server socket for MAVProxy  
mavproxy_sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)  
mavproxy_sock.setblocking(0)  
mavproxy_sock.bind((HOST, mavproxy_port))
```

5) Create a MAVLink instance, which will perform 'encode'/'decode' operations on the socket created above

```
mav_obj = mavlink_apm.MAVLink(mavproxy_sock)
```

6) To read messages from the UAV, the decode() method is used. Sample code:

```
# Call to receive data over UDP socket. 1024 is the buffer size  
data_from_mavproxy,address_of_mavproxy = mavproxy_sock.recvfrom(1024)  
decoded_message = mav_obj.decode(data_from_mavproxy)
```

7) To access individual fields, there are other useful methods. For example, get\_msId() returns the message ID sent by the UAV (the message ID is used to distinguish the different possible messages between the UAV and the sample application). Sample code:

```
print("Got a message with id %u, fields: %s, component: %d, System ID: %d" %  
(decoded_message.get_msId(), decoded_message.get_fieldnames(),  
decoded_message.get_srcComponent(), decoded_message.get_srcSystem()))  
# Prints the entire decode message  
print(decoded_message)
```

*Sample output for a heartbeat message:*

```
Got a message with id 0, fields: ['type', 'autopilot', 'base_mode', 'custom_mode',  
'system_status', 'mavlink_version'], component: 1, System ID: 1  
# the decoded message  
HEARTBEAT {type : 2, autopilot : 3, base_mode : 89, custom_mode : 3,  
system_status : 4, mavlink_version : 3}
```

- 8) To send commands to the UAV, there are variations of encode() method. For example, to set a waypoint the UAV has to traverse to, the following method is used:

```
mission_item_encode()
```

## Utility functions

The current mode of UAV is present in the ‘heartbeat’ message generated periodically by the UAV. We expect ‘reading the current UAV mode’ and setting the current UAV mode to be common operations. So, these operations are provided in a module called FTL\_util.py (can be imported by using *import FTL\_util*)

To use this, an instance needs to be created:

```
FTL_util_obj = FTL_util.FTL_util()
```

The current mode of the UAV can be obtained by passing the heartbeat\_message as an argument to get\_mav\_mode() function. Sample code that returns the mode as a string:

```
mode = get_mav_mode (heartbeat_message)
```

The following sample code sets the UAV in AUTO mode:

```
FTL_util_obj = FTL_util.FTL_util()  
FTL_util_obj.set_mav_mode(FTL_util_obj.auto_mode,mav_obj,  
                           mavproxy_sock, address_of_mavproxy)
```

More utility functions may be added as needed.

## Commonly used functionality

Each MAVLink message has an ID that signifies the type of message. For example, the ‘heartbeat’ message that is sent by the UAV has a message ID of zero. This is given by **MAVLINK\_MSG\_ID\_HEARTBEAT** in `mavlink_apm.py`. Similarly, a message with GPS information has the message ID **MAVLINK\_MSG\_ID\_GPS\_RAW\_INT**. The following code checks if a message received is a heartbeat or it contains GPS information:

```
if msg_id == mavlink_apm.MAVLINK_MSG_ID_GPS_RAW_INT:  
    print 'This is a message with GPS information!'  
elif msg_id == mavlink_apm.MAVLINK_MSG_ID_HEARTBEAT:  
    print 'This is a heartbeat message'
```

Refer page 4 of this [document](#) for description of other commonly used message IDs.

Command sent to the UAV for performing actions (i.e. setting mode, setting a way-point etc.) have a command ID. For example, to arm/disarm the UAV, there is `MAV_CMD_COMPONENT_ARM_DISARM`. It can be accessed by using `mavlink_apm.MAV_CMD_COMPONENT_ARM_DISARM`. Similarly to set a way-point, `MAV_CMD_NAV_WAYPOINT` is used and it can be accessed by `mavlink_apm.MAV_CMD_NAV_WAYPOINT`.

### Given a requirement, how to choose the right API?

Refer `mavlink_apm.py`. Browse through the methods under ‘mavlink’ class (line 3008) to find the one that best suits your requirement. If you are still not sure, refer [mavproxy.py](#) and [mavutil.py](#). `mavproxy.py` makes use of the same library for sending messages. For example, entering ‘arm throttle’ in MAVProxy terminal will ARM the UAV. Checking `mavproxy.py`, we see that the `cmd_arm()` function handles this command, which in turn invokes `arducopter_arm()` in `mavutil.py`. This is a wrapper function which finally calls `command_long_send()` which is defined in `mavlink_apm.py` and that is what we need to use in our sample applications.

## Using SITL with Sample applications

This section describes how to validate a sample application using SITL.

### Sample application 1

This sample application performs the following operations:

- 1) Sets the copter to AUTO mode
- 2) Listens on UDP port 12345 for messages from MAVProxy
- 2) Reads the GPS data that is generated by SITL and forwarded by MAVProxy. Saves this raw data to a file every 5 seconds.

Code for the sample application is available in `/home/droneusr/sample_prog/program_1`

## Steps for running the sample application

Step 1: Make a reservation for 'APM\_Copter\_3DRobotics\_SITL\_Image' using

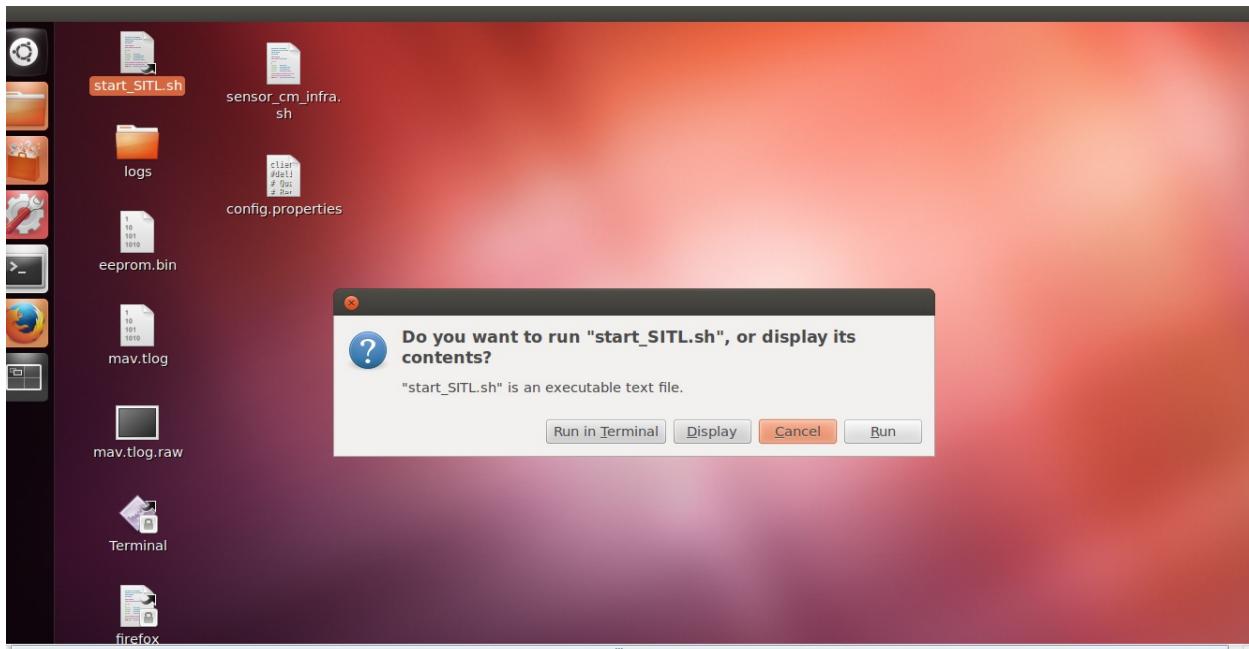
<https://vcl.ncsu.edu/>

Step 2: Use the Windows Remote Desktop Connection to this computer using the following credentials:

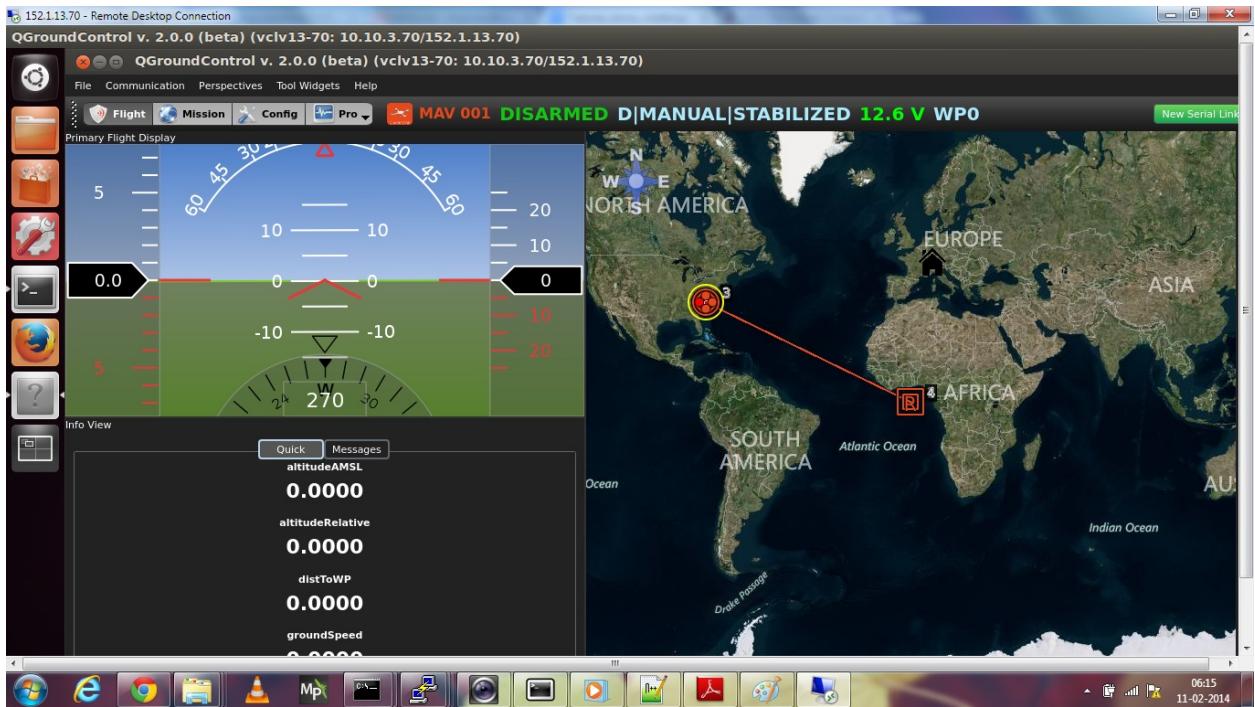
Username: droneusr

Password: drone123

Step 3: Running SITL: Click start\_SITL.sh. Select 'Run' when prompted:



This should open 4 tabs one of which runs the qgroundcontrol (the Ground Control Station that is bundled with this image):



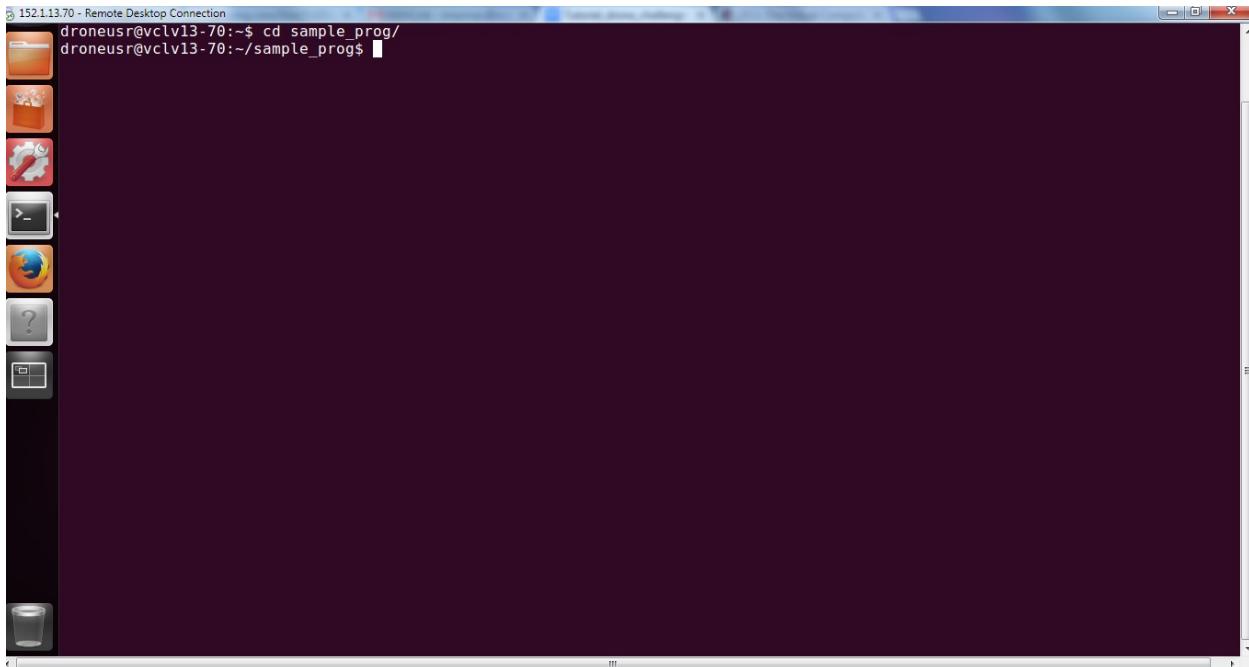
Zoom in using the side-bar on the right so that you have get a decent view of the copter:



NOTE: This has to be performed manually the first-time you run SITL after reserving an image. For subsequent runs, you can click on 'Last Pos' and you will be taken to the last position that you were looking at.

Step 4: Running sample app 1:

Open a new terminal and enter the 'sample\_prog' directory:



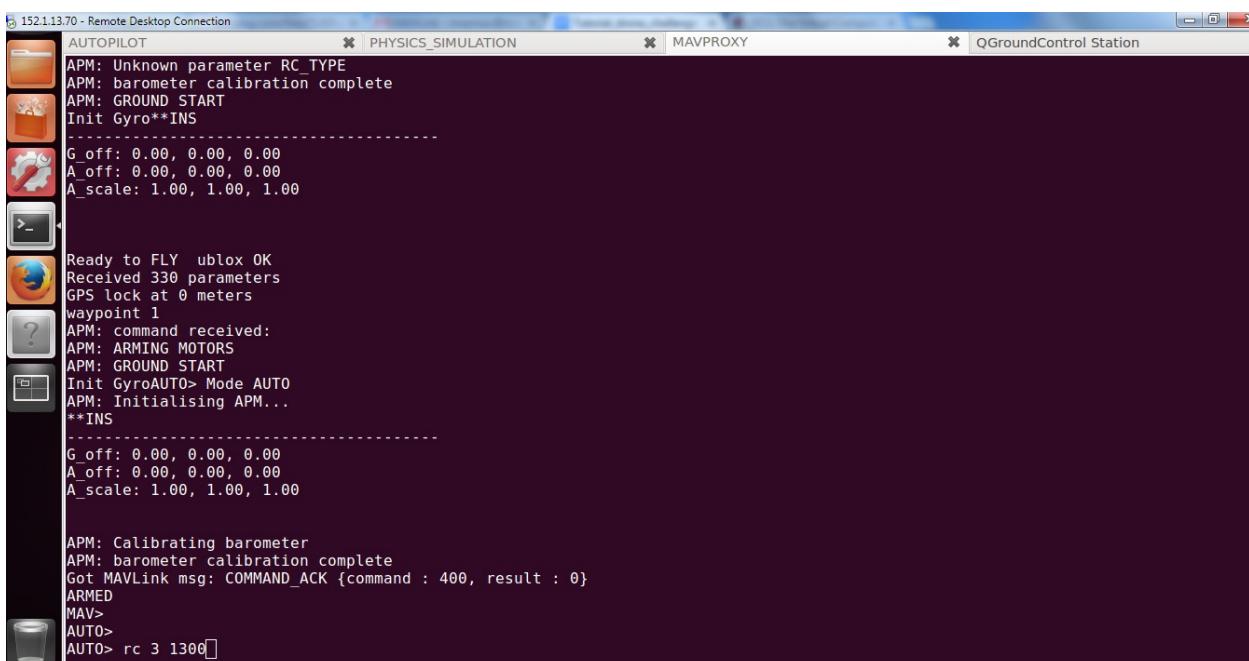
Enter program\_1 directory and start the app using the following syntax:

./uav\_auto\_mode.py <MAVProxy port to connect to> <name of the file>

In our example, the MAVProxy port is always 12345. We specify a file named gps\_data to which the readings are saved periodically (every 5 seconds).

Step 5: Provide throttle by entering the following command on MAVProxy terminal:

AUTO>rc 3 1300



Step 6: Observe the file 'gps\_data' getting populated with GPS readings every 5 seconds (run tail -f gps\_data in program\_1 directory):

```

droneusr@vcv13-70: ~/sample_prog/program_1
GPS_RAW_INT {time_usec : 535758000, fix_type : 3, lat : 357713121, lon : -786743912, alt : 584090, eph : 0, epv : 65535, vel : 0, cog : 0, satellites_visible : 10}
GPS_RAW_INT {time_usec : 541153000, fix_type : 3, lat : 357713121, lon : -786743912, alt : 584090, eph : 0, epv : 65535, vel : 0, cog : 0, satellites_visible : 10}
GPS_RAW_INT {time_usec : 546373000, fix_type : 3, lat : 357713121, lon : -786743912, alt : 584090, eph : 0, epv : 65535, vel : 0, cog : 0, satellites_visible : 10}
GPS_RAW_INT {time_usec : 551377000, fix_type : 3, lat : 357713121, lon : -786743912, alt : 584090, eph : 0, epv : 65535, vel : 0, cog : 0, satellites_visible : 10}
GPS_RAW_INT {time_usec : 556795000, fix_type : 3, lat : 357713121, lon : -786743912, alt : 584090, eph : 0, epv : 65535, vel : 0, cog : 0, satellites_visible : 10}
GPS_RAW_INT {time_usec : 561995000, fix_type : 3, lat : 357713121, lon : -786743912, alt : 584090, eph : 0, epv : 65535, vel : 0, cog : 0, satellites_visible : 10}
GPS_RAW_INT {time_usec : 567015000, fix_type : 3, lat : 357712967, lon : -786744114, alt : 586870, eph : 0, epv : 65535, vel : 185, cog : 22643, satellites_visible : 10}
GPS_RAW_INT {time_usec : 572255000, fix_type : 3, lat : 357711903, lon : -786745497, alt : 586920, eph : 0, epv : 65535, vel : 423, cog : 22666, satellites_visible : 10}
GPS_RAW_INT {time_usec : 577475000, fix_type : 3, lat : 357711043, lon : -786746630, alt : 586920, eph : 0, epv : 65535, vel : 36, cog : 22886, satellites_visible : 10}
GPS_RAW_INT {time_usec : 582697000, fix_type : 3, lat : 357711017, lon : -786746515, alt : 586900, eph : 0, epv : 65535, vel : 134, cog : 10158, satellites_visible : 10}
GPS_RAW_INT {time_usec : 587922000, fix_type : 3, lat : 357710705, lon : -786744906, alt : 586920, eph : 0, epv : 65535, vel : 304, cog : 10516, satellites_visible : 10}
GPS_RAW_INT {time_usec : 593143000, fix_type : 3, lat : 357710531, lon : -786744176, alt : 587030, eph : 0, epv : 65535, vel : 2, cog : 9138, satellites_visible : 10}
GPS_RAW_INT {time_usec : 598364000, fix_type : 3, lat : 357710551, lon : -786744189, alt : 593020, eph : 0, epv : 65535, vel : 3, cog : 28741, satellites_visible : 10}
GPS_RAW_INT {time_usec : 603565000, fix_type : 3, lat : 357710909, lon : -786744186, alt : 598160, eph : 0, epv : 65535, vel : 251, cog : 352, satellites_visible : 10}
GPS_RAW_INT {time_usec : 608786000, fix_type : 3, lat : 357712547, lon : -786744010, alt : 598230, eph : 0, epv : 65535, vel : 325, cog : 635, satellites_visible : 10}
GPS_RAW_INT {time_usec : 614010000, fix_type : 3, lat : 357713158, lon : -786743924, alt : 598380, eph : 0, epv : 65535, vel : 4, cog : 27966, satellites_visible : 10}
GPS_RAW_INT {time_usec : 619231000, fix_type : 3, lat : 357713096, lon : -786743959, alt : 594320, eph : 0, epv : 65535, vel : 8, cog : 20707, satellites_visible : 10}

```

Step 7: Stop the sample application (by giving a Ctrl-C) after the UAV has completed traversing all the waypoints:



Following [video](#) demonstrates the working of this sample application (this uses Mission planner as the ground control station).

## Sample application 2

This sample application performs the following operations:

- 1) Listens on UDP port 12345 for messages from MAVProxy
  - 2) Reads the file that has been generated by the first sample application, converts each reading to a WayPoint and guides the UAV to that WayPoint. This process is repeated for each reading in the file every 5 seconds. Once all the entries in the file are read, the UAV is set to 'Return to Launch (RTL)' mode.

Code for the sample application is available in /home/droneusr/sample\_prog/program\_2

## Steps for running the sample application

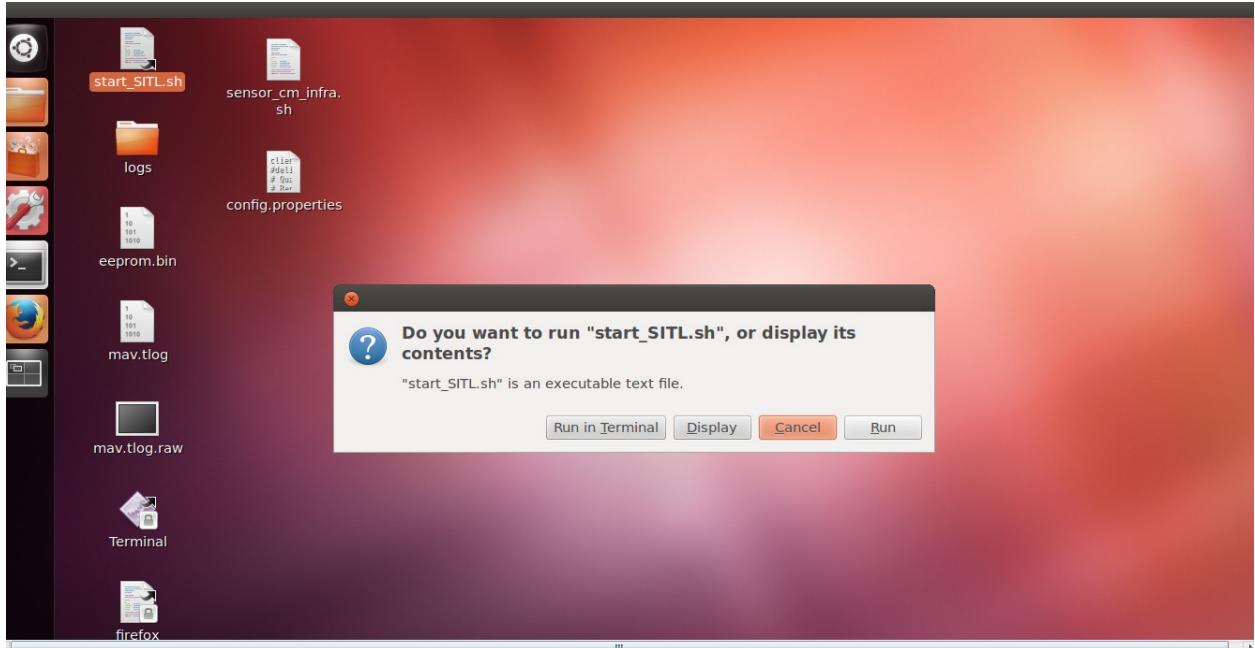
Step 1: Make a reservation for 'APM\_Copter\_3DRobotics\_SITL\_Image' using <https://vcl.ncsu.edu/>

Step 2: Use the Windows Remote Desktop Connection to this computer using the following credentials:

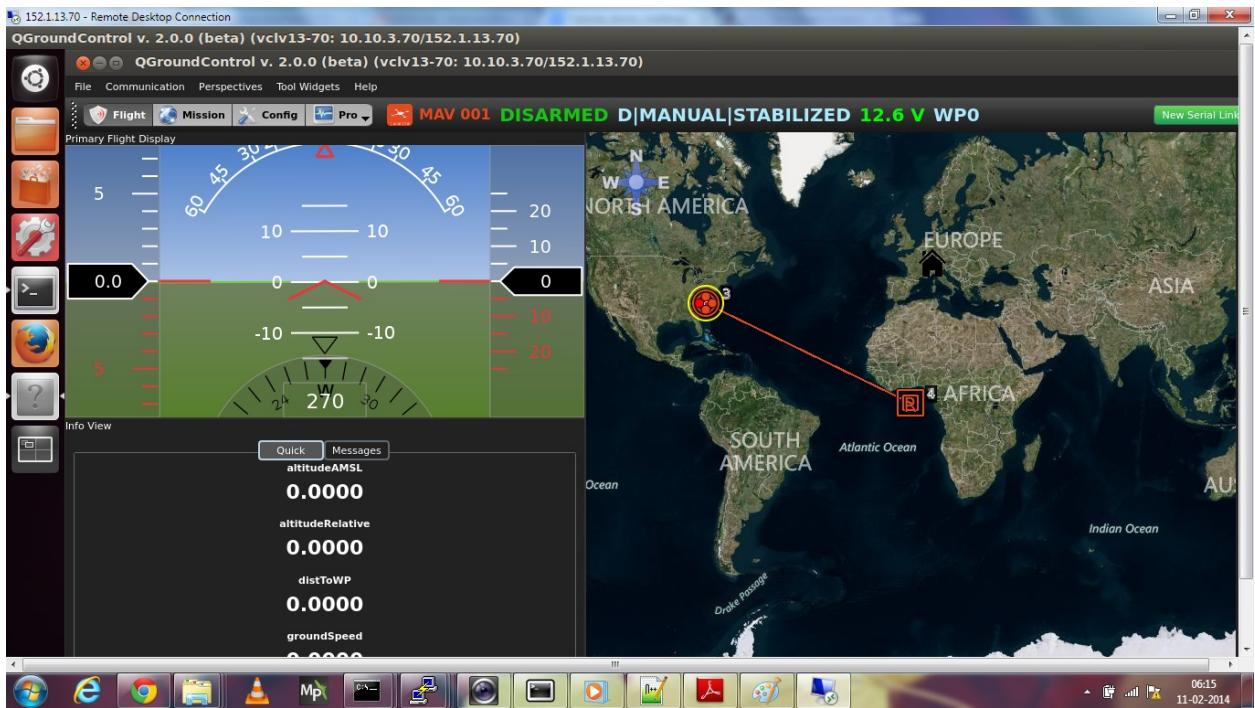
Username: droneusr

Password: drone123

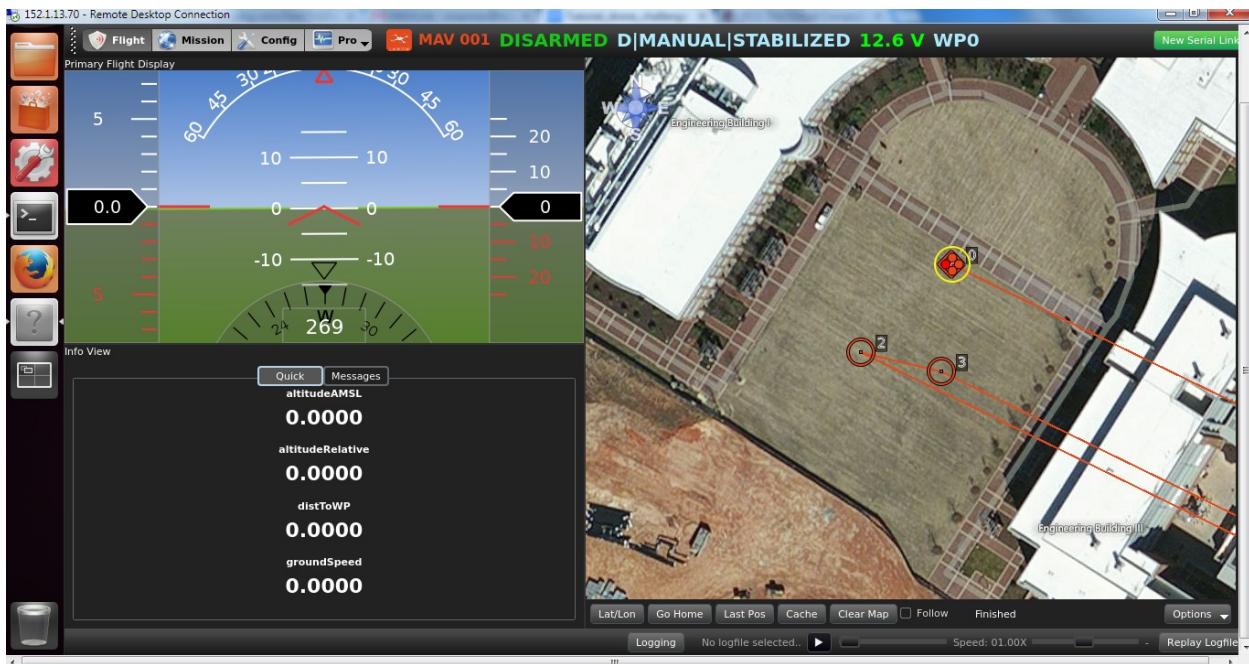
Step 3: Running SITL: Click start SITL.sh. Select ‘Run’ when prompted:



This should open 4 tabs one of which runs the qgroundcontrol (the Ground Control Station that is bundled with this image):



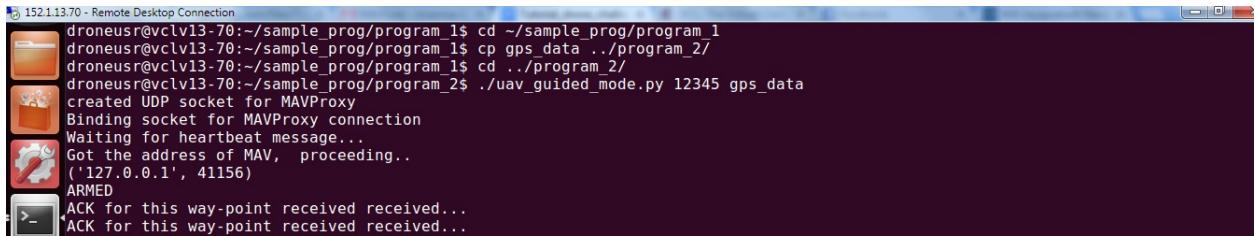
Zoom in using the side-bar on the right so that you have get a decent view of the copter:



NOTE: This has to be performed manually the first-time you run SITL after reserving an image. For subsequent runs, you can click on 'Last Pos' and you will be taken to the last position that you were looking at

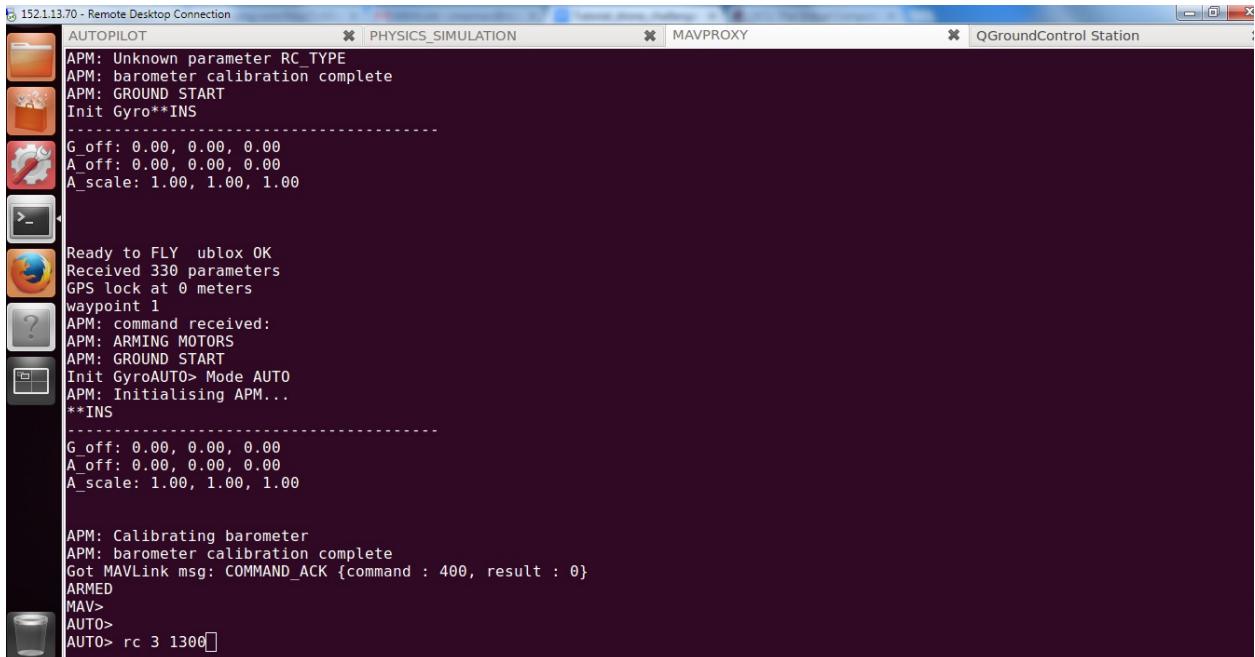
Step 4: Running sample application 2: We make use of the file that was generated by sample application 1. Copy the file to ‘program\_2’ directory and enter ‘program\_2’ directory:

#### Step 6: Start the sample application:



```
152.1.13.70 - Remote Desktop Connection
droneusr@vclvl3-70:~/sample_prog/program_1$ cd ~/sample_prog/program_1
droneusr@vclvl3-70:~/sample_prog/program_1$ cp gps_data .. /program_2/
droneusr@vclvl3-70:~/sample_prog/program_1$ cd .. /program_2/
droneusr@vclvl3-70:~/sample_prog/program_2$ ./uav_guided_mode.py 12345 gps_data
created UDP socket for MAVProxy
Binding socket for MAVProxy connection
Waiting for heartbeat message...
Got the address of MAV, proceeding..
('127.0.0.1', 41156)
ARMED
ACK for this way-point received received...
ACK for this way-point received received...
```

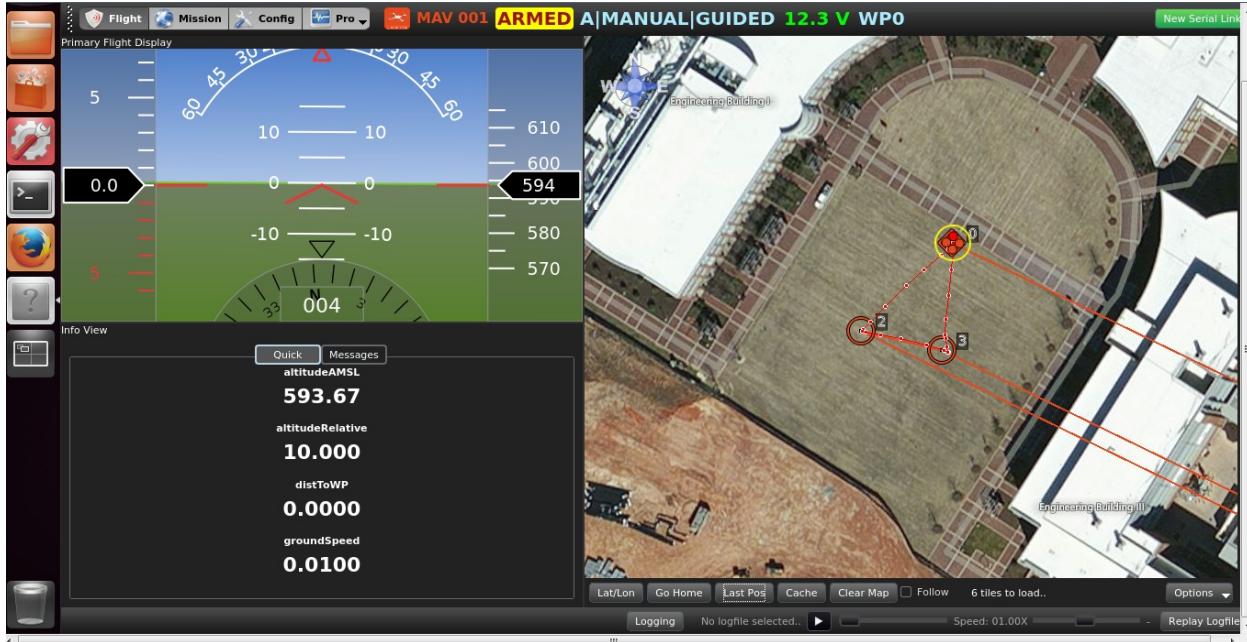
Step 7: Provide throttle by entering the following command on MAVProxy terminal:  
AUTO>rc 3 1300



```
152.1.13.70 - Remote Desktop Connection
AUTOPILOT PHYSICS_SIMULATION MAVPROXY QGroundControl Station
APM: Unknown parameter RC_TYPE
APM: barometer calibration complete
APM: GROUND START
Init Gyro**INS
-----
G_off: 0.00, 0.00, 0.00
A_off: 0.00, 0.00, 0.00
A_scale: 1.00, 1.00, 1.00
-----
Ready to FLY ublox OK
Received 330 parameters
GPS lock at 0 meters
waypoint 1
APM: command received:
APM: ARMING MOTORS
APM: GROUND START
Init GyroAUTO> Mode AUTO
APM: Initialising APM...
**INS
-----
G_off: 0.00, 0.00, 0.00
A_off: 0.00, 0.00, 0.00
A_scale: 1.00, 1.00, 1.00

APM: Calibrating barometer
APM: barometer calibration complete
Got MAVLink msg: COMMAND_ACK {command : 400, result : 0}
ARMED
MAV>
AUTO>
AUTO> rc 3 1300
```

Step 8: The UAV should follow a similar path as the UAV that flew in AUTO mode (sample application 1).



After all the way-points are traversed, the UAV is set to RTL mode.

```

AUTOPILOT          PHYSICS_SIMULATION      MAVPROXY           QGroundControl Station
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
height 10
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
height 0
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
Got MAVLink msg: MISSION_ACK {target_system : 0, target_component : 0, type : 0}
RTL> Mode RTL
height 10
height 0
RTL>

```

Following [video](#) demonstrates the working of this sample application.

## Saving sensor data

Any Sensor data that is read by applications running on the UAV can be saved to periodically to a repository. The VCL image has a basic client and server combination that achieves this functionality.

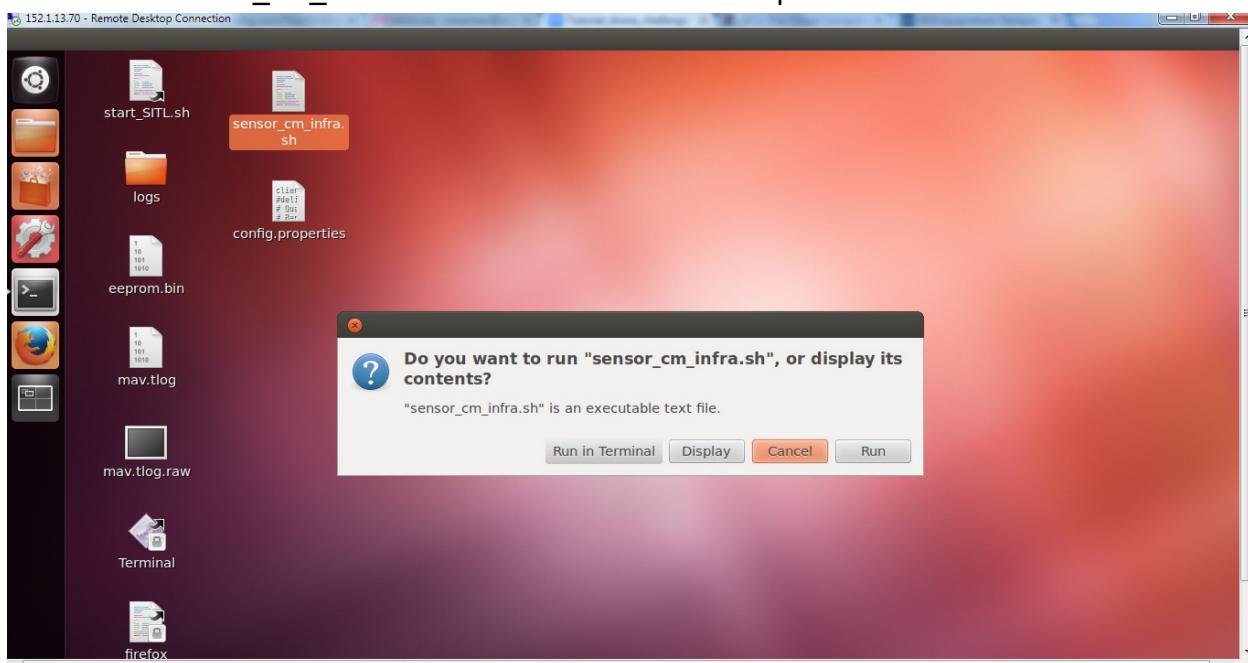
Sensing application (client): This reads sensor data periodically and reports the readings to a server. The type of sensor, the frequency of reading etc. are specified in a file ‘config.properties’. Server: The server receives readings for various data from different clients and saves this to a repository.

## Working of the sensing platform:

The VCL image has a client-server combination that works with sample application 1. We use GPS readings as sensor data. To recall, the sample application keeps saving the GPS readings to a file every 5 seconds. In addition to that, it also writes the current GPS reading to a file “/tmp/gps\_data”. So each time the UAV receives a message with GPS information, the older reading is replaced by the current one in this file. The sensing application reads this file periodically and sends it to the server. Whenever the sensing application reads the file, it is guaranteed to get the latest GPS reading (as that is written continuously).

## Running the sensing platform:

Run the file `sensor_cm_infra.sh` in `/home/droneusr/Desktop/`:



It spawns a new terminal with two tabs - one tab has the client running and the other has server running:

```

SENSING APPLICATION (CLIENT) SERVER
Packet [preamble=CENTMESH, actionCode=HEARTBEAT, id=00-50-56-18-20-44, sequenceNumber=-1, data=[Quantity, GPS reading, Representation, String, Max Sensing Freq, 50000]]
=====> Client Measurement Time 1392123340233
Performing Sensor Reading
GPS_RAW_INT {time_usec : 697170000, fix_type : 3, lat : 357713123, lon : -786743910, alt : 584090, eph : 0, epv : 65535, vel : 0, cog : 0, satellites_visible : 10}
=====> Client Reporting Time 1392123340234
===
Broadcast method called for CLIENT_BROADCAST_MEASUREMENT packet
Broadcast Report
Following information being sent from Client to Server:
Packet [preamble=CENTMESH, actionCode=REPORT_SENSED_DATA, id=00-50-56-18-20-44, sequenceNumber=589304, data=[Quantity, GPS reading, Representation, raw, Readin
84090, eph : 0, epv : 65535, velet from Client to Server:
Representation, raw, Reading, GnCode=REPORT_SENSED_DATA, id=00-50-56-18-20-44, sequenceNumber=589304, data=[Quantity, GPS reading, Representation, raw, Reading, GPS_RAW_INT {time_usec : 697170000, fix_type : 3, lat : 357713123, lon : -786743910, alt : 584090, eph : 0, epv : 65535, vel : 0, cog : 0, satellites_visible : 10}, Timestamp, 2014/02/11 07:55:35, Quantity, GPS reading, Representation, raw, Reading, GPS_RAW_INT {time_usec : 697170000, fix_type : 3, lat : 357713123, lon : -786743910, alt : 584090, eph : 0, epv : 65535, vel : 0, cog : 0, satellites_visible : 10}, Timestamp, 2014/02/11 07:55:40]]
20000
Received a Packet
Client Operation begins
+++++> [B@21d0a4f
Packet Type is ACKNOWLEDGE_SENSED_DATA
Packet [preamble=CENTMESH, actionCode=ACKNOWLEDGE_SENSED_DATA, id=00-50-56-18-20-44, sequenceNumber=589304, data=null]
Log Report Acknowledgement function
Sequence Number: 589304 matched
Removed measurement: ClientMeasurement@6791d8c1
Removed measurement: ClientMeasurement@182d9c06
21000
Socket timed out
No socket is open!
22000

```

From the tab in which the client process is running, it can be seen that this is accessing the gps reading in /tmp/gps\_data.

**NOTE:**

- If you run this without the sample application, the same data will be read by the client again and again (as there is no process refreshing the file).
- If you run this without running the sample application at least once, each time the client tries to read the file, there will be an exception (as the file does not exist).

Following snapshot shows the tab in which server process runs. Whenever some data is received from the client, it is printed by the server:

```
SENSING APPLICATION (CLIENT) | SERVER
SensorRecord [timeStamp=2014/02/11 08:00:56-18-20-44]
ime_usec : 697170000, fix_type : 3, lat:55, sensorQuantity=GPS reading, sensorRepresentation=raw, sensorReading=GPS_RAW_INT {t
tellites visible : 10}, deviceName=00-t : 357713123, lon : -786743910, alt : 584090, eph : 0, epv : 65535, vel : 0, cog : 0, sa
tellitesVisible : 10}, deviceName=00-50-56-18-20-44]
SensorRecord [timeStamp=2014/02/11 08:02:00, sensorQuantity=GPS reading, sensorRepresentation=raw, sensorReading=GPS_RAW_INT {t
ime_usec : 697170000, fix_type : 3, lat : 357713123, lon : -786743910, alt : 584090, eph : 0, epv : 65535, vel : 0, cog : 0, sa
tellitesVisible : 10}, deviceName=00-50-56-18-20-44]
SensorRecord [timeStamp=2014/02/11 08:02:05, sensorQuantity=GPS reading, sensorRepresentation=raw, sensorReading=GPS_RAW_INT {t
ime_usec : 697170000, fix_type : 3, lat : 357713123, lon : -786743910, alt : 584090, eph : 0, epv : 65535, vel : 0, cog : 0, sa
tellitesVisible : 10}, deviceName=00-50-56-18-20-44]
SensorRecord [timeStamp=2014/02/11 08:02:10, sensorQuantity=GPS reading, sensorRepresentation=raw, sensorReading=GPS_RAW_INT {t
ime_usec : 697170000, fix_type : 3, lat : 357713123, lon : -786743910, alt : 584090, eph : 0, epv : 65535, vel : 0, cog : 0, sa
tellitesVisible : 10}, deviceName=00-50-56-18-20-44]
SensorRecord [timeStamp=2014/02/11 08:02:15, sensorQuantity=GPS reading, sensorRepresentation=raw, sensorReading=GPS_RAW_INT {t
ime_usec : 697170000, fix_type : 3, lat : 357713123, lon : -786743910, alt : 584090, eph : 0, epv : 65535, vel : 0, cog : 0, sa
tellitesVisible : 10}, deviceName=00-50-56-18-20-44]
SensorRecord [timeStamp=2014/02/11 08:02:20, sensorQuantity=GPS reading, sensorRepresentation=raw, sensorReading=GPS_RAW_INT {t
ime_usec : 697170000, fix_type : 3, lat : 357713123, lon : -786743910, alt : 584090, eph : 0, epv : 65535, vel : 0, cog : 0, sa
tellitesVisible : 10}, deviceName=00-50-56-18-20-44]
SensorRecord [timeStamp=2014/02/11 08:02:25, sensorQuantity=GPS reading, sensorRepresentation=raw, sensorReading=GPS_RAW_INT {t
ime_usec : 697170000, fix_type : 3, lat : 357713123, lon : -786743910, alt : 584090, eph : 0, epv : 65535, vel : 0, cog : 0, sa
tellitesVisible : 10}, deviceName=00-50-56-18-20-44]
SensorRecord [timeStamp=2014/02/11 08:02:30, sensorQuantity=GPS reading, sensorRepresentation=raw, sensorReading=GPS_RAW_INT {t
ime_usec : 697170000, fix_type : 3, lat : 357713123, lon : -786743910, alt : 584090, eph : 0, epv : 65535, vel : 0, cog : 0, sa
tellitesVisible : 10}, deviceName=00-50-56-18-20-44]
/10.10.3.70
4000
java.net DatagramPacket@626f50a8
Following information being to sent from Server to Client:
Packet [preamble=CENTMESH, actionCode=ACKNOWLEDGE_SENSED_DATA, id=00-50-56-18-20-44, sequenceNumber=514554, data=null]
Packet has been properly processed.
432000
433000
```