**Installation**

**Ubuntu installation**

1. Download from <https://releases.ubuntu.com/18.04.5/> the Ubuntu version **18.04 LTS.** 
   1. Download VirtualBox from <https://www.virtualbox.org/> and install.
2. Launch VirtualBox and set the VM possibly with this **minimum** requirement:
   1. 5Gb of RAM
   2. 4 Processor
   3. 128 ;b of Video memory
   4. At least 40 Gb of storage memory with a dynamic allocation
3. Start the installation of Ubuntu.
   1. Select the .iso file of Ubuntu ([how to use VM](https://brb.nci.nih.gov/seqtools/installUbuntu.html))
   2. Fill all the field and choose a password
4. After the installation restart the system

**Component installation**

1. Download from Next Cloud the folder installation named “Installation”.
2. Open a new terminal (Ctrl+Alt+T) and navigate inside the folder in which you save the files.
   1. [here a brief guide how to navigate using prompt command](https://help.ubuntu.com/community/UsingTheTerminal)
3. Go inside the folder “File” and digit for each file:

**chmod +x [file\_name.sh]** {in this way you make the file executable}

1. To install Ubuntu dependency:

**./1\_system\_deps.sh**

1. To install ROS :

**sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb\_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'**

[this command does not return anything]

**sudo apt-key adv --keyserver 'hkp://[keyserver.ubuntu.com:80](http://keyserver.ubuntu.com/)' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654**

**sudo apt update**

**sudo apt install ros-melodic-desktop-full**

**echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc**

**source ~/.bashrc**

**sudo apt install python-catkin-tools**

**sudo apt-get install python-dev python-opencv python-wxgtk4.0 python-pip python-matplotlib python-lxml python-pygame**

**pip install PyYAML mavproxy --user**

**echo "export PATH=$PATH:$HOME/.local/bin" >> ~/.bashrc**

Link: <http://wiki.ros.org/noetic/Installation/Ubuntu>

1. To install Ardupilot:

**./4\_ardupilot.sh**

1. To install QGroundControl:

**./5\_qgroundcontrol.sh**

**NOTE:** use “***./ubuntu\_check.sh***” in order to verify the correct installation. No problem if

* python-argparse [FAIL]
* python-pyserial [FAIL]
* python-pyymal [FAIL]
* python-pyulog [FAIL]
* docker-ce [FAIL]
* ansible [FAIL]

**Import Ardupilot/Gazebo packages (DA CONTROLLARE)**

In order to do a SITL with MavProxy and Gazebo/ROS it is necessary to have the proper models of Gazebo with the plugin of ArduPilot. In a terminal, return in the Home folder ( >>cd ../.. ) and digit:

**cd** *[be sure to be in HOME path]*

Go into the tribe page of [GitLab](https://gitlab.com/users/sign_in?__cf_chl_jschl_tk__=35e93b2747a96df519494ed6af4f28c211f64a03-1607853151-0-AWOAijF1tH1SCJVJmJwejDrYs3DfLbF2DufD7kS0ypPFC3ufcki1wH4nvpjM2X75EP8aBi89SdJJ0t78pZmIFoKNhQxcSVyDqotWuKuuD_f3CRm_HzmxaytP6TG2Bd0WLPsbis-QCyHrxbsQJKW8ZnIvUv0Zw09mPxNckdfqthPGuX-WsWTLO6qecR8BkfaeFHszQ9-zO_CT6aQV_QaVm-iifwBn9R_4JFWqMd245L2ZbpAwaIXTvSC0P9YIU_aKFn8RkmrO0AJcNrLfkYh5bmVz3mrcoNqWrZxi1o_SxJHh9Bp5ZriGWRxKUmhR95ZE7IEoU7dQRbEwThB07_C4UKUWW1STqCmIG6wP2967p0gMbHOMneN7A5883QeHou3CNg), open “***DRAFT Polito / OAeMP / Leonardo\_20-21\_ws***” project, click on “**clone**” button, and cogitpy the http address to paste here:

**git clone *[paste\_the\_URL]* NOTE:** no differences between ssh or https

**NOTE**: Here we locally copy an entire ROS workspace, with a sufficient documentation. To use it is needed only to follow these steps:

**cd ~/leonardo\_20-21\_ws**

**catkin\_make**

**echo source ~/leonardo\_20-21\_ws/devel/setup.bash >> ~/.bashrc**

**source ~/.bashrc**

**Initialization SITL (Software-in-the-Loop)**

1. Setup: open a terminal and digit  
     
   **echo export PATH=$PATH:$HOME/ardupilot/Tools/autotest >> ~/.bashrc**

**echo export PATH=/usr/lib/ccache:$PATH >> ~/.bashrc**

2. Open a terminal and digit:

**cd ~/ardupilot/ArduCopter**

**sim\_vehicle.py –w**

**NOTE:** in case of sim\_vehicle.py command not found

1. In the terminal open the bashrc file: gedit ~/. Bashrc
2. Scroll all the file and delete the last two export command and substitute with:

export PATH=$PATH:$HOME/ardupilot/Tools/autotest

export PATH=/usr/lib/ccache:$PATH

1. Save and close the file
2. Try to run sim\_vehicle.py –w in the folder **ardupilot/ArduCopter**

3. Kill the application:

**Ctrl+c**

4. troubleshooting : if you get the following error messages  
  
SIM\_VEHICLE: Run MavProxy  
SIM\_VEHICLE: "mavproxy.py" "--map" "--console" "--out" "127.0.0.1:14550" "--out" "127.0.0.1:14551" "--master" "tcp:127.0.0.1:5760" "--sitl" "127.0.0.1:5501"  
RiTW: Starting ArduCopter : /home/draft/ardupilot/build/sitl/bin/arducopter -S --model + --speedup 1 --defaults /home/draft/ardupilot/Tools/autotest/default\_params/copter.parm -I0  
[Run MavProxy] An exception has occurred with command: 'mavproxy.py --map --console --out 127.0.0.1:14550 --out 127.0.0.1:14551 --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501'  
[Errno 2] No such file or directory  
SIM\_VEHICLE: Killing tasks

run these lines in the terminal:

**sudo pip2 uninstall mavproxy**

**sudo pip2 install -U mavproxy**

and try again step 2 and 3

**Simulation**

**Software in the Loop and simulation**

**NOTE**: This is the classical simulation of a iris drone model with a GPS flight mode, the model already has all the sensor components but are not used for the No GPS flight mode.

1. First basic simulation using MavProxy:

**cd ~/ardupilot/ArduCopter**

**sim\_vehicle.py -j4 --map --console**

1. In the same terminal using the following MAVLink command:
   1. To take off:

**mode guided**

**arm throttle**

**takeoff 40**

* 1. To perform a circular trajectory:

**rc 3 1500**

**mode circle**

**param set circle\_radius 2000**

* 1. To return at the starting point and execute a land manoeuvre:

**mode rtl**

**SITL with Gazebo (DA CONTROLLARE)**

1. Start the Gazebo ambient with:

**roslaunch iris\_gazebo iris\_empty\_world.launch**

1. Start MavProxy in another terminal/tab:

**cd ~/ardupilot/ArduCopter**

**sim\_vehicle.py -v ArduCopter -f gazebo-iris --map --console**

From now on you have 2 possibilities.

1. Use the MAVLink command as in point 2 of **Software in the Loop**
2. Use MAVROS.

**MAVROS**

**NOTE**: This is the classical simulation of a iris drone model with a GPS flight mode, the model already has all the sensor components but are not used for the No GPS flight mode.

1. Start MavROS in a new termianl

**roslaunch mavros apm.launch fcu\_url:=udp://127.0.0.1:14551@14555**

**NOTE:** wait until the model receives GPS signal

**Basic command:**

**Take off command**

Open a new tab (a new window in the terminal):

* 1. **rosrun mavros mavsys mode -c GUIDED** (this command changes the flight mode)
  2. **rosrun mavros mavsafety arm**
  3. Wait until the motor start otherwise -> **motor failed**
  4. **rosrun mavros mavcmd takeoffcur 0 0 1**

**NOTE**: cur = current coordinates

0 0 1 is the follow vector [pitch, yaw, altitude] so with this vector you can set the coordinates to make the drone taking off

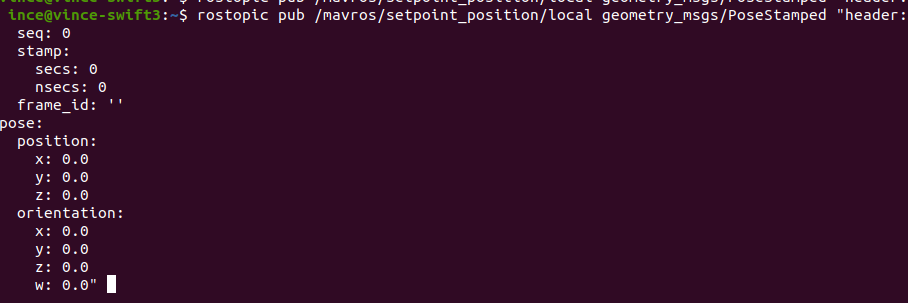
**Navigation command**

**Set point command**

With the following command you start a navigation using waypoints.

**rostopic pub /mavros/setpoint\_position/local** (“tab button” for autocompletion)

This is the topic message:

****

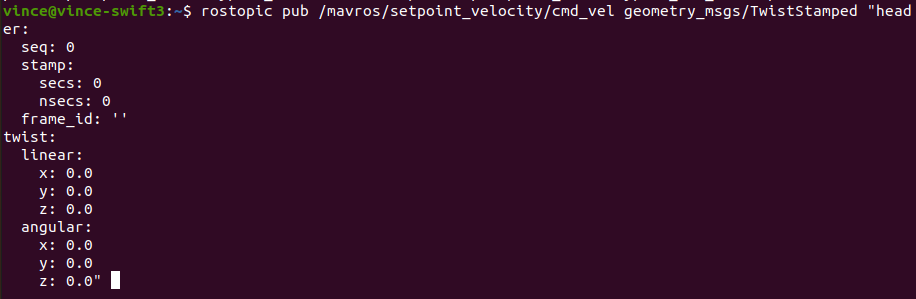
Here change the values of coordinates:

* [x,y,z] position a normal vector
* [x,y,z,w] orientazione using quaternion [link](https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles)

**Velocity command**

In this way you can command the drone using a velocity vector:

**rostopic pub /mavros/setpoint\_velocity/cmd\_vel** (“tab button” for autocompletion)

****

Here you can change the velocity (in case of [0 0 0] vectors the drone will travel at a default speed):

* [x,y,z] linear velocity
* [x,y,z] angular velocity

**Land command**

This command allow you to do the landing maneuver:

**rosrun mavros mavcmd landcur 0 0**

**SITL No GPS model (work in progress!!)**

In order to change the SITL in NO\_GPS flight mode:

Go in the follow folder:

**~/ardupilot/Tools/autotest/default\_params/gazebo-iris.parm**

And so change the follow parameters:

|  |  |  |
| --- | --- | --- |
| **Parameters** | **Description** | **Value** |
| FRAME\_CLASS 1 | Represent the drone class | 1 = quadcopter  4 = octacopter |
| FRAME\_TYPE 1 | Drone confiuration  E.g. X or H or + | 1 = X |
| EK2\_ENABLE 1  EK3\_ENABLE 0 | EKF (extendend kalman filter) parameters | 1 = activated  0 = deactivated |
| EK2\_GPS\_TYPE 3 | EKF input signal | 3 = no from GPS |
| AHRS\_EKF\_TYPE 2 | Attitude heading reference system  for EKF | 2 = odometry parameters from EKF2 |
| AHRS\_GPS\_USE 0 | Attitude heading reference system  for GPS | 0 = disabling GPS for AHRS |
| GPS\_TYPE 0 | GPS | 1 = activated  0 = disactivated |
| VISO\_TYPE 1 | Visual odometry (valid only for ArduPilot version greater than 4.1.0) | 0 = disactivated  1 = MAVLink  2 = Intel T265 |
| COMPASS\_USE 0 | Compass enable/disable |  |
| COMPASS\_USE 0 | Compass enable/disable |  |
| COMPASS\_USE2 0 | Compass enable/disable |  |
| COMPASS\_USE3 0 | Compass enable/disable |  |
| ARMING\_CHECK 0 | As the name says | 0 = disable  1 = enable all arming check |
| EK2\_GPS\_CHECK 0 |  | 0 = disable  2 = enable all checks |

**NOTE**: For the complete list of Copter parameters see: <https://ardupilot.org/copter/docs/parameters.html>

**Script development**

In order to create a packages in wcih store all the script and work with it:

**cd ~/*leonardo\_20-21\_ws*/src** : go inside the cloned workspace folder

**catkin\_create\_pkg** ***[package\_name]*** **rospy roscpp rospy roscpp std\_msgs geometry\_msgs mavros\_msgs gazebo\_msgs** [*dependencies*] : create a packages [*package\_name*] with the dependencies

**cd ~/*leonardo\_20-21\_ws***

**catkin\_make** : build the package

**cd ~/*leonardo\_20-21\_ws*/src/*[package\_name]***  for example   
 **cd ~/*leonardo\_20-21\_ws*/src/OAMP\_follow\_me**

**mkdir scripts** : create script folder inside the package created

**touch [script\_name].py** :create a new python script (empty)

**chmod +x [script].py** : make the script executable

You can also upload the scripts **waypoint\_sim\_v1.py, rotor\_mtx\_converter.py** inside the folder “catkin\_ws/src/OAMP\_follow\_me/scripts”.

**chmod +x waypoint\_sim\_v1.py**

**chmod +x rotor\_mtx\_converter.py**

to run:

**roscore**

**NOTE**: If there is not a ROS master yet, then you should run ‘roscore’ first.

And in a new terminal:

**rosrun *[pakcage\_name] [script]*** : execute the script

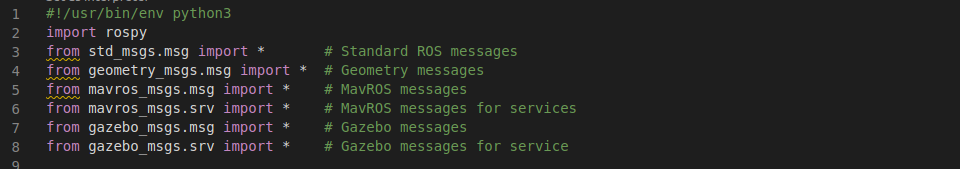
**NOTE**: the script waypoint\_sim\_v1 works using a file.txt. Be sure to change the path folder at line 73 and use the right formatting like in the *waypoints.txt.*

**Script and message guide**

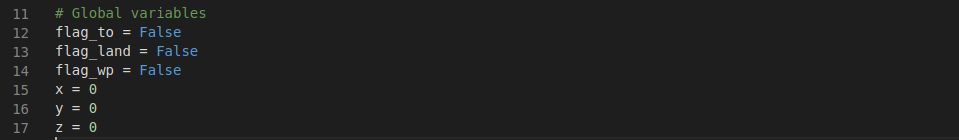
Example 1

Here a brief guide in order to create a sequence of instructions to be done by the drone. Definition of message and how to acquire the same information from the sensor or other programs.

Here the definition of the libraries/modules we need like rospy, geometry\_msg etc.. the “**import \***” allow us to import all the content.



This the declaration of the global variables = variables that you can use in all the script, in particular here we have the same flags to define the condition of a status and the starting initial condition of the pose.



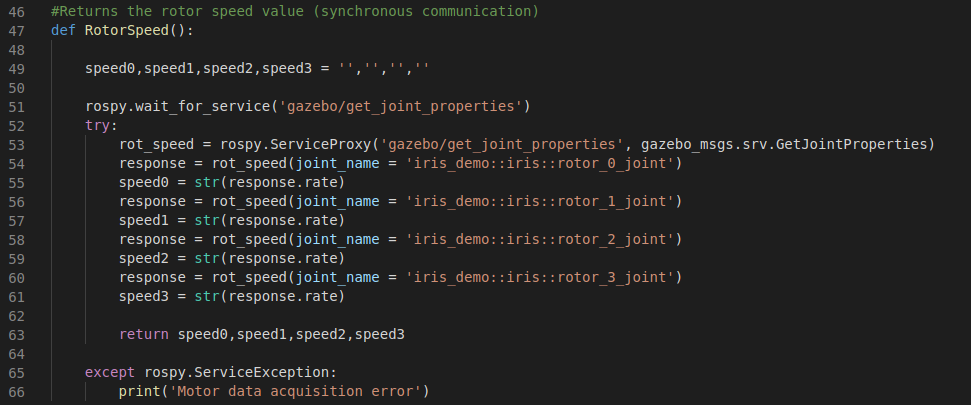
Here an example for the **acquisition** of the rotor velocity during a simulation. This allowed us to create a function rotor velocity -> battery consumption

With **rospy** we have the possibility to convert a file srv (service file) into a Python source code. A service is a “request and response message” so we ask for some information and receive it.

* rospy.wait\_for\_service(‘[*service\_name*]’) = a method used to block until we receive the response
* rot\_speed = rospy.ServiceProxy(‘[*service\_name*]’, [*service\_type*]) = definition of a service inside the variable rot\_speed

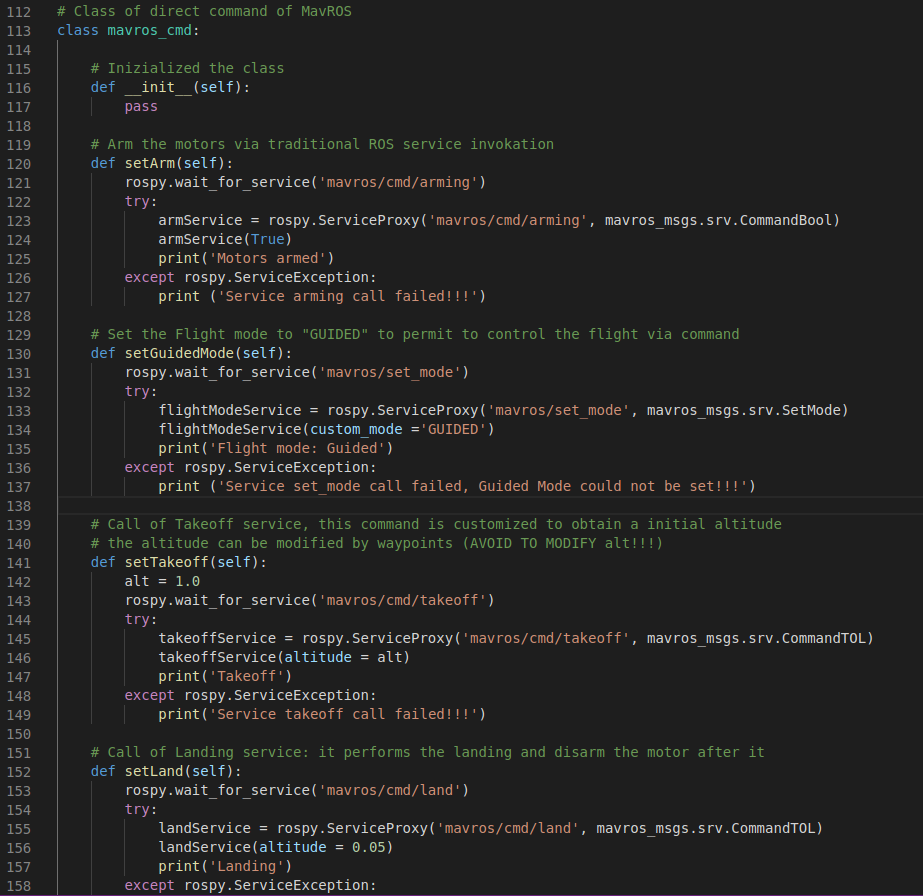
So every time we call rot\_speed we ask and acquire the information defined (in this case the joint properties).

* response = rot\_speed(joint\_name = ‘information\_name’)
* speed0 = str(response.rate) -> here the rotor velocity streamed at each rate (Hz)
* str = translate the file in a string file

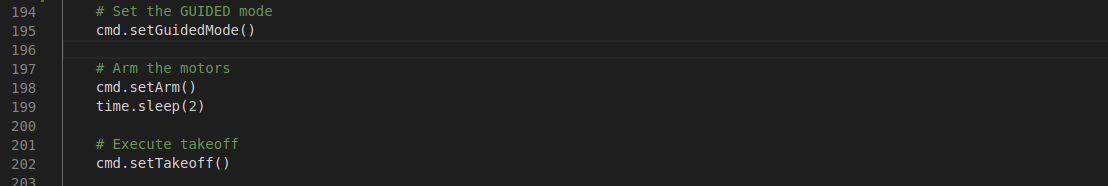
After that we define speed0,speed1, speed2, speed3 in order to save inside this variable the value of the rotor velocity

Here we declare some useful function inside the class mavros\_cmd:

* setArm
* setGuidedMode
* setTakeof
* setLand



Here we create the object of the classHere we call the function



Into the main we first initialize the ROS node with the “rospy.init\_node(“*node\_name*”).

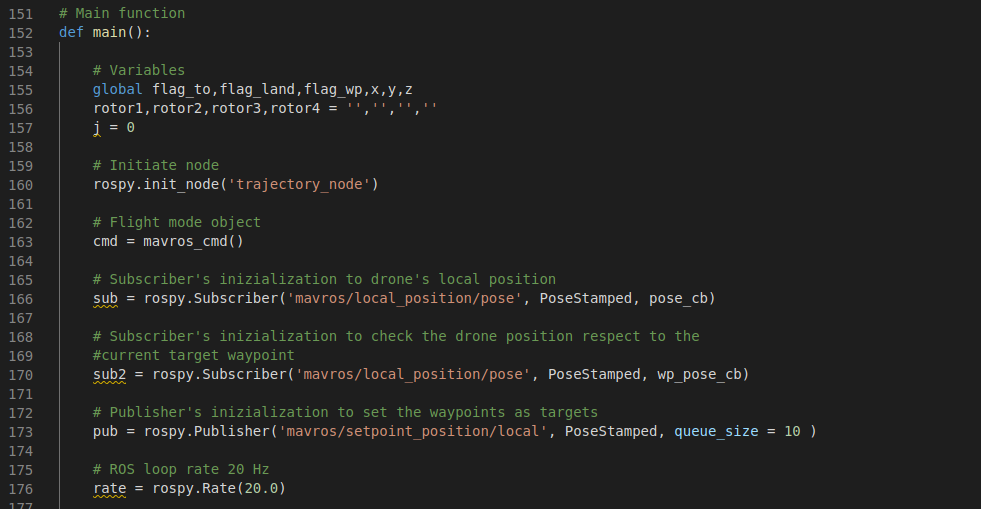
Then we initialize the publishers with “rospy.Publisher”, where we specify the topic over which we publicate and the message type.

* pub = rospy.Publisher(‘[*topic*]’,[message\_type])

Then we initialize the subscribers with “rospy.Subscriber”, where we specify the topic from which we get the data, the expected message type and the callback function which is the function that actually retrieves these data.

* sub = rospy.Subscriber(‘[*topic*],’[*message\_type*]’,’callback\_function’)
* callback\_function store the value inside a variable and permit us to elaborate data

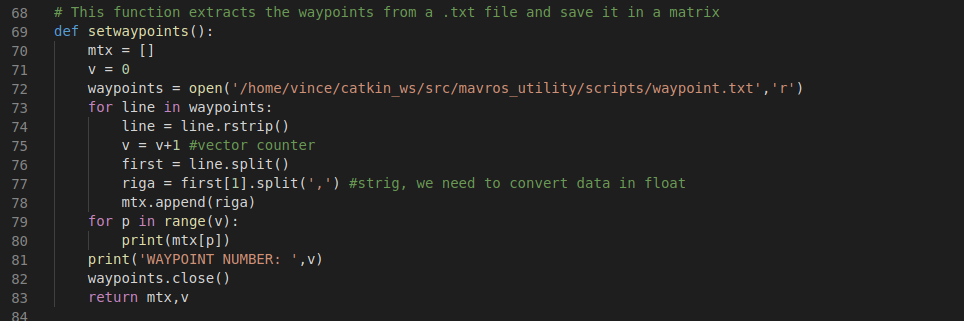
Finally we define the rate (frequency in Hz) at which we want to publish over the topic.



Here we manage some information inside a .txt file using python libraries. In this way we can use data from other scripts.

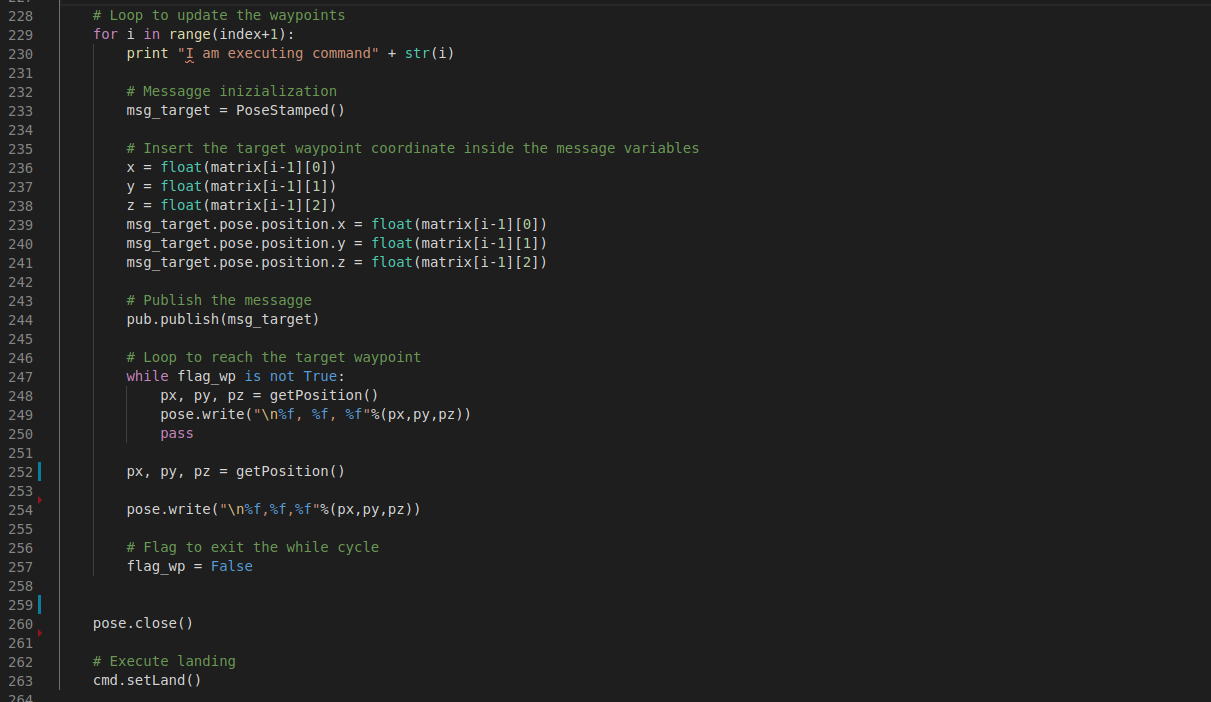
This is a simple for loop of a matrix, here we read and save the coordinates of the waypoints.

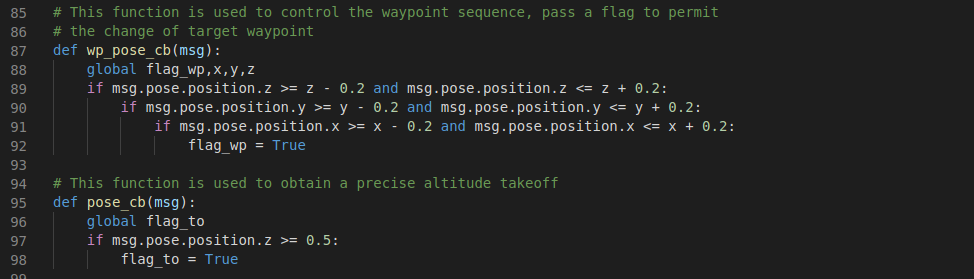
For the description of this few lines we recommend this link: [loop](https://www.w3schools.com/python/python_for_loops.asp), [how to open a file](https://www.w3schools.com/python/python_file_open.asp)



After the collection of the waypoints we send the coordinates to the drone using:

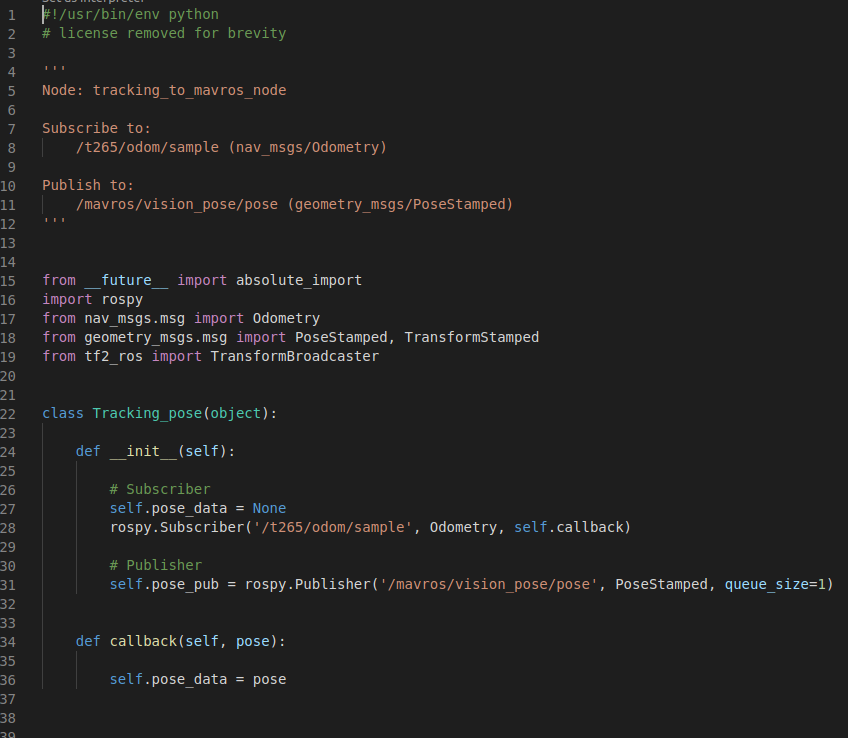
* msg\_target = PoseStamped() -> creation of the object
* msg\_target.pose.position.[*variable*] = float([*value*])
* pub.publish(msg\_target) -> publication
* <http://docs.ros.org/en/melodic/api/geometry_msgs/html/msg/PoseStamped.html>



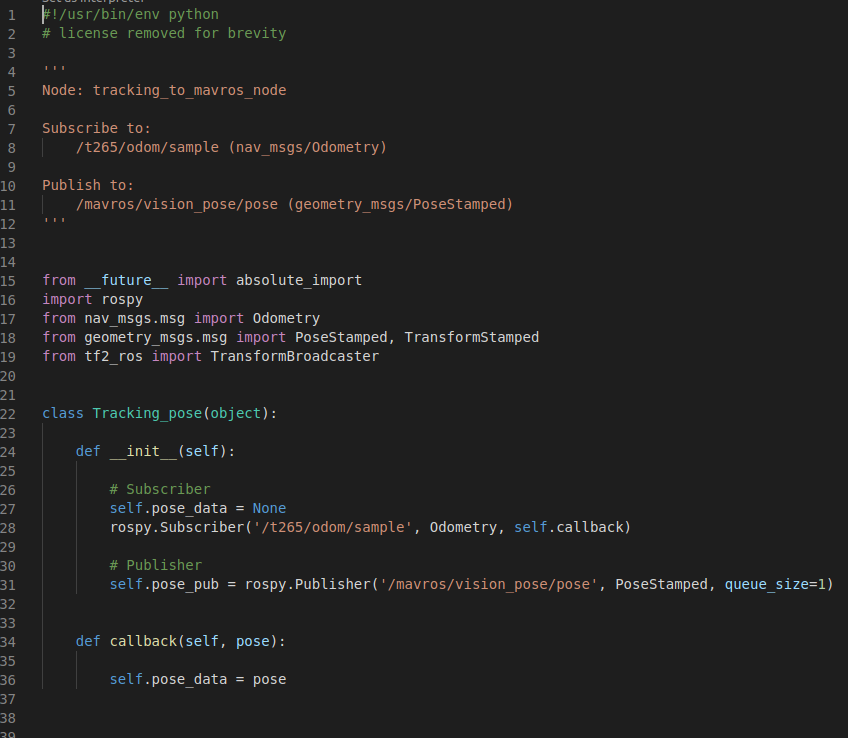
Here we compare the current position of the drone with the coordinates of the waypoint to be reached. The triple if condition allows us to create a region of acceptance. If flag\_wp is true then we reach the waypoint.

Example 2

Here the first thing is importing the needed module for the script.



Here we define a class for the tracking of the pose, in which we initialize the subscriber and the publisher in “\_\_init\_\_” and then we define the callback function to store the data of the subscriber.

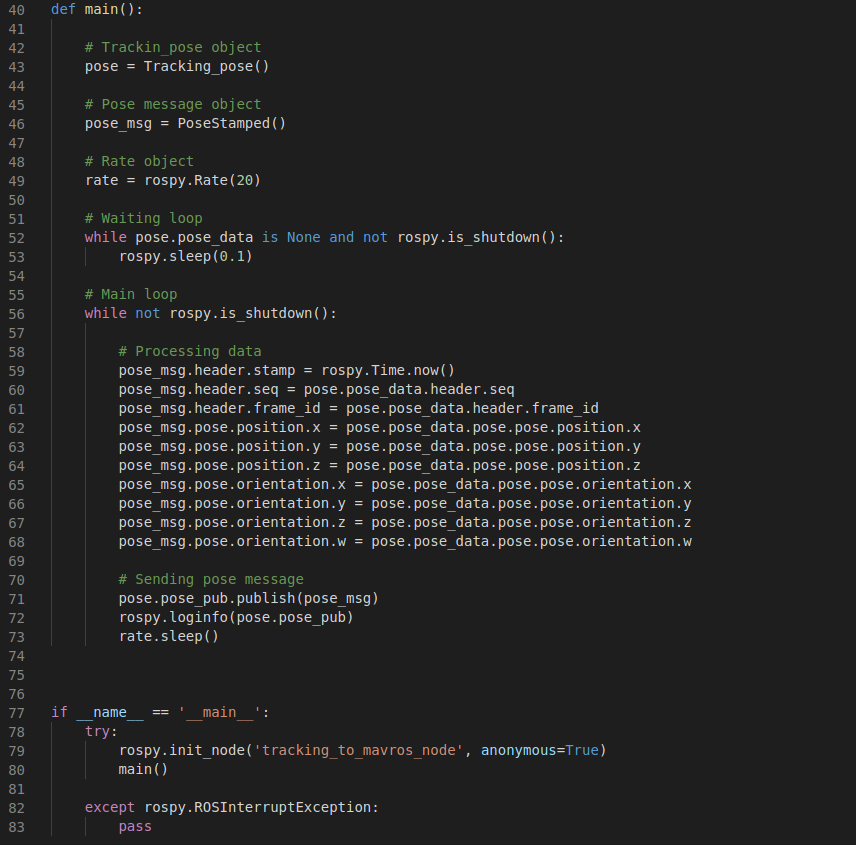


In the main function we perform all the data operations. First we create the objects we need to use and the rate at which we want to publish.

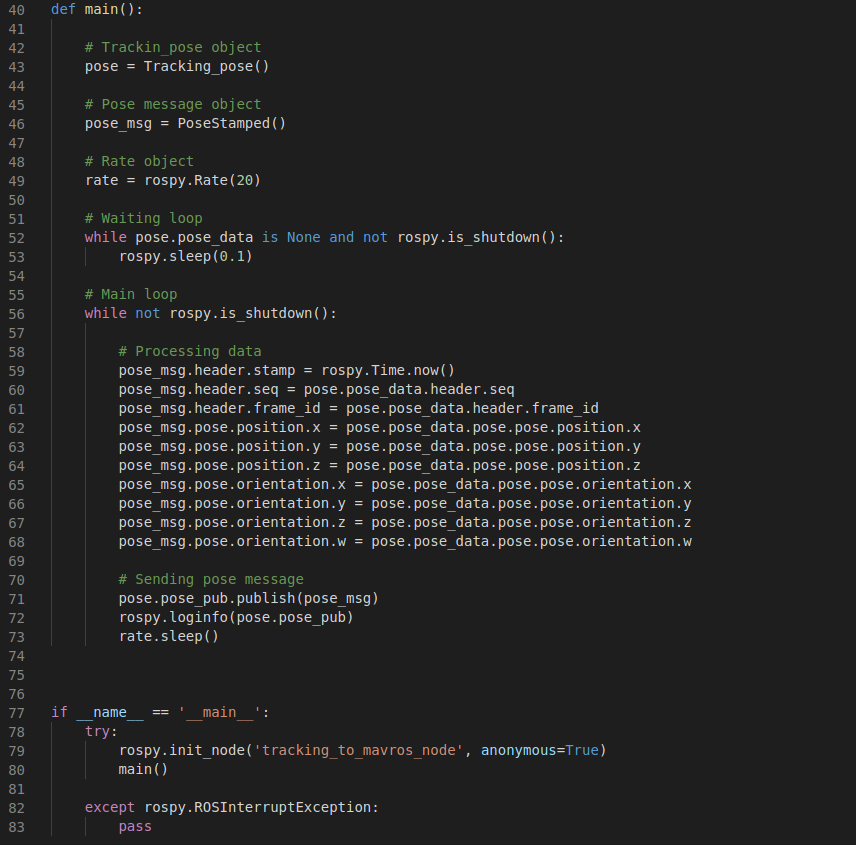
A first while loop is required in order to wait till the data from the subscriber are available.

The second while loop is the main loop that assigns dynamic values to the fields of the object.

Finally the “rate.sleep” is required to publish at the specified rate.



Here we run the script. First we initialize the script as a ROS node and execute the “main” function. The exception handler is required to catch possible interruptions in ROS execution.



**Webography**

* [**http://wiki.ros.org/mavros**](http://wiki.ros.org/mavros) **MAVROS**
* **<https://www.youtube.com/watch?v=Gxx3iWTuc_U&list=UUV86qjrnnmBH9tt3L5qGEwQ&ab_channel=ludusrusso> ROS video lessons (italian version)**
* **<http://wiki.ros.org/rospy/Overview> Rospy**
* **<https://www.w3schools.com/python/default.asp> Python**
  + [**for loop**](https://www.w3schools.com/python/python_for_loops.asp)
  + [**function**](https://www.w3schools.com/python/python_functions.asp)
  + [**class/object**](https://www.w3schools.com/python/python_classes.asp)
  + [**module**](https://www.w3schools.com/python/python_modules.asp)
  + [**read file**](https://www.w3schools.com/python/python_file_open.asp) **/** [**write file**](https://www.w3schools.com/python/python_file_write.asp)