Juan Carlos Escalante Diaz Del Campo
ID: 149498

Final Report

Modeling and simulation of trajectory generation by “Resistive Grid Algorithm”

Mechatronics Engineering

National Institute of Astrophysics Optics and Electronics

14th august 2017 – 04th December 2017

Objectives

➢ Principal: Application of “Resistive Grid” method for the generation and simulation of trajectories for Unmanned aerial vehicles.

➢ Secondary: Creation of .launch files in “Ubuntu” operating system, utilizing Robot Operating System. Programación en Python para trayectorias predeterminadas.

➢ Secondary: Implementation of simulation of UAV: quadrotor drone using Gazebo simulator and TUM_simulator.

➢ Secondary: Map conditioning for importing .collada for a .world file for Gazebo simulator.

➢ Secondary: Python programming for predetermined trajectories.

Goals

The goal is to generate trajectories with the “Resistive Grid algorithm” starting from a 2D map of 3D map utilizing Python programming language finding the best solution without collisions.

A second goal is to simulate a quadrotor drone’s movement with high level programming language utilizing feedback to achieve the most certain outcomes.

Technical Specifications

- Operative System Ubuntu 14.04.05 Trusty
    - This operative system was used because of the ROS dependencies, and the support that the page has for it, most of ROS's support is in this platform.

- ROS Indigo
    - This Robotic Operating System is used to manage, create and utilize different libraries, repositories or files, related, to robotics itself, in this case we choose this type of ROS because it supports Ubuntu's 14.04 version, and also has the most support online.
- AR. Drone Autonomy 2.0
    - Ardrone_autonomy is a ROS driver for Parrot AR-drone 1.0 and 2.0 quadrotor. This driver is based on the official AR-Drone SDK version 2.0.1. The package was developed in the "Autonomy" laboratory of Simon Fraser University by Mani Monajjemi and other contributors.
- Gazebo Simulator, TUM_simulator, TUM_ardrone
    - Gazebo is a robotic simulation environment, not only used for drones, it is also used for robotic arms, differential robots and bases.
    - TUM_simulator is a library or repository that allows the simulation of the Quadrotor Ardrone 2.0, based on the real specifications and situations in a real environment, simulating an IP address that is the one that the dummy drone is sending to the network adapter of the computer.

- TUM_ardrone is a library or repository that allows you to view the drone cameras for autonomous camera-based navigation.

➢ Parrot AR.Drone 2.0

Drone used for physical tests that was also used in the simulation tests.

Physical specifications:

Dimensions: Helmet: 52.5 x 51.5 cm | Without helmet: 45 x 29 cm

Weight: 380 g with the outer helmet cover. | 420 g with the helmet cover for interiors.

Running speed (cruise): 5 m / s, 18 km / h. Average flight autonomy: 12 minutes.

- Integrated computer system:

ARM9 RISC 32-bit microprocessor @ 468 MHz

128 MB DDR SDRAM Memory @ 200 MHz

Operating system with Linux kernel

Wi-Fi modem b / g

High speed USB connector

- Inertial guidance systems:

3-axis accelerometer built with MEMS technology.

3-axis gyroscope, 3-axis magnetometer and pressure sensor.

- Engines and energy:

or 4 brushless motors, running at 3,500 rpm with a power of 15 W

3-cell lithium-ion battery, capable of delivering 1000 mA / hour with a nominal voltage of 11.1 V, and which allows a flight autonomy of between 12 and 15 minutes.

Discharge capacity: 10C. Charging time: 90 minutes.

- Front, ventral and altimeter camera:

Front camera with CMOS sensor 93 ° amplitude, ventral with CMOS sensor with diagonal lens 64 ° of amplitude, two 40 khz ultrasounds to achieve great stability at great heights.

Project justification

The use of unmanned aircraft or, more technically called, autonomous unmanned vehicles (UAV) has increased in recent years. While the use of UAV used to be only military, today we can find UAVs for research, exploration, photography, patrol and even recreation.

Therefore, today it is more important than ever to learn to program the drones. Although they can be controlled in real time by the user, the implementation of routes and automatic trajectories, together with sensors and control systems, can help with the use of drones in various tasks such as patrol or rescue and aid missions, being totally autonomous.

The project presented in this paper was carried out for the reasons described above. With the ultimate goal of creating several routes and trajectories to subsequently create optimal routes through resistive networks.

Applicable regulations

In May 2016, the SCT and the general directorate of civil aviation published the requirements to operate a UAV. CO AV-23/10 R2 applies to any natural person who operates or intends to operate an unmanned aerial craft. The aviation authorities classified UAVs by their weight and use:

Weight	Category	Use
2 kg or less	Micro	Recreational(private)
		Noncommercial(private)
		Commercial
2kg to 25 kg	Small	Recreational (private)
		Noncommercial (private)
		Commercial
More than 25 kg	Big	Recreational (private)
		Noncommercial (private)
		Commercial

Some of the general requirements and limitations for all categories are:

- No pilot should throw or throw objects
- No pilot should operate the drone in restricted or restricted areas
- No pilot should operate the drone where it is not visible
- The pilot must maintain control of the flight path at all times
- The UAV must be operated in official hours

- The UAV must not be operated from moving vehicles
- The drone pilot must not operate more than one drone at a time
- Depending on the category and use, the requirements and limitations will change, the Parrot AR. DRONE 2.0 enters the micro category, all UAVs in this category do not require operating authorization, as long as the pilots respect the following requirements and limitations:
- The owner of the UAV must register all drones weighing more than 250 g on the website of the SCT / DGAC.
- The pilot must operate the buzz up to 122 meters.
- The drone must not be more than 450 meters horizontal distance from the pilot.
- It must be operated at a minimum distance of 9.2 km from an aerodrome.
- The pilot must operate the UAV at a maximum height of 100 meters.
- The UAV must be operated at least at a distance of 0.9 km from a heliport.
- The speed will depend on the maximum takeoff weight.

Feasibility analysis

A feasibility analysis helped compare the project with current technologies and current products, services and requirements and decide whether the objectives could be achieved or not.

The following aspects were considered:

Papers or Related Patents

Martin Lefebure has the patent WO 2010061099 A2, published on January 3, 2010, which refers to the control of a drone with a device for performing basic movement commands.

The invention relates to a device for piloting a drone, which includes a housing provided with a detector for tilting the housing and a touch screen, which shows a plurality of touch sensitive areas. The device includes a means controlled by a tactile

area forming an on / off button to alternately switch the pilot mode of unmanned aircraft between an activation mode of the drone autonomous stabilization system, in which the pilot orders sent to the hum are the result of the transformation of signals emitted by the touch-sensitive areas and a deactivation mode of the autonomous stabilization system of the drone, in which the steering commands sent to the hum are the result of the transformation of signals emitted by the accommodation inclination detector.

The control method in the elementary pilot functions are functions of the group comprising takeoff, earth, up, down, turn to the right, turn to the left, forward, backward, to the right, panning to the left.

Available Technologies

For the project various libraries and software were used to connect with the drone and fly it. As it was done before, we found several pages with help on the problems that we found in the development. Among the programs used, we find:

➔ Linux / Ubuntu

➔ ROS Kinetic

➔ AR Drone controllers

In addition, we must mention that we have reviewed several guides and libraries of third parties to help us solve certain problems that occurred during the development.

Current products and services

Nowadays, there are many UAVs in the market, from those used for photographs and recreations to the military. In general, recreational "drones" do not have options to configure them as autonomous Autonomous vehicles, because part of the experience offered to the user is to promote them.

However, those that are used for the military tasks of even scientific or amateur contests may include the option of installing drivers to allow for auto-routing or autonomous flying coding. The most used UA for this are the Parrot Drone AR series.

Improvements identified

The main improvement of the project gives the existing buzz is the ability to create autonomous trajectories and control using the computer, while receiving the current state of the engines and general navigation; While the AR Drone 2.0 usually only uses the user's cell phone for control.

Project Development

To begin, after the project was assigned, the scope of this summer practice and the objectives were discussed, the assigned project was: "Design and modeling of resistive networks for the planning of Robotic Trajectories free of collisions". The approach and resolution must all be in the ROS operating system. For the first week there was a PowerPoint presentation, which stated the activities that were carried out when taking the drone course.

Since the drone course is totally relevant to the subject and the preparation of the project, the presentation was made and presented, to see how advanced it could be in the use and management of ROS.

Later, a work proposal was made to carry out these practices, in which the simulation of a drone was proposed in an environment with obstacles, generate a trajectory, either vectorially or with resistive networks, to get from one point to another, or with a geometric figure.

For this project the Gazebo simulator was used, Ubuntu 14, Ubuntu 14 is used because it is the one with the most ROS support to carry out related activities, and ROS Indigo, which has the most support in the ROS page.

To do this, VMware was first downloaded, which is a software used for virtual machines, and a virtualized machine was used by Dr. Gerardo, who oversees the project; An image is shown below:

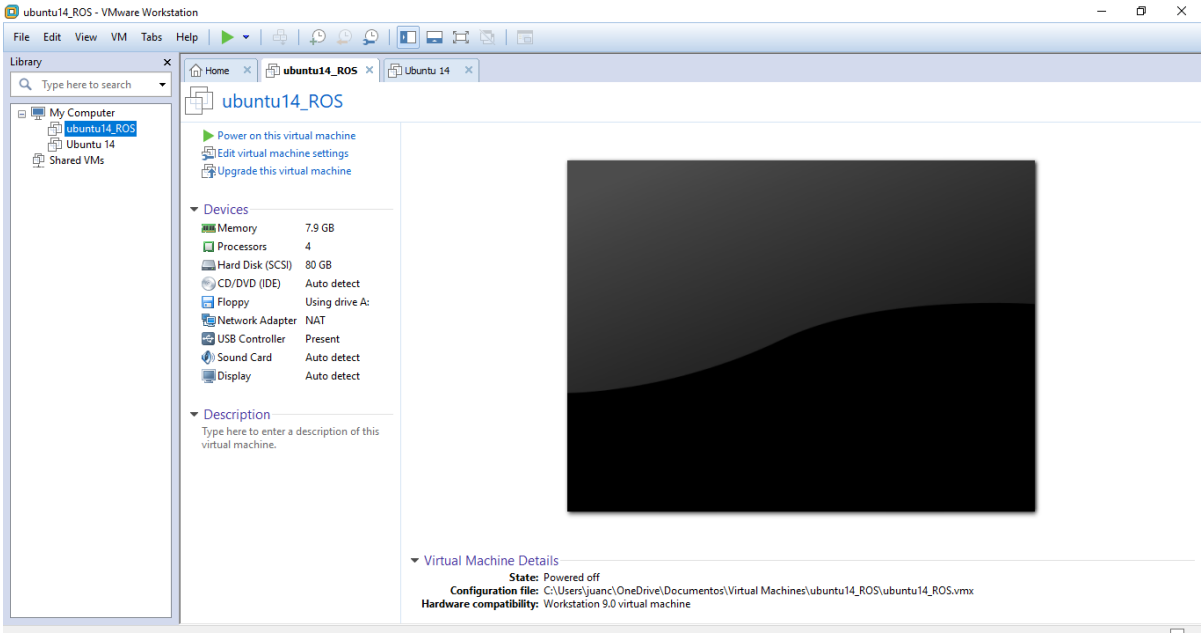


Image 1: Virtualized machine beginning

For optimal use of the virtual machine, the use of RAM memory was changed to approximately 8 gb, in addition to increasing the processors from 2 to 4.

When starting the virtual machine, you had to open a terminal or CMD to enter the following commands to do the whole simulation process of the AR.drone Parrot:

- Instalation of AR.Drone autonomy(utilizing binary installation by Debian):
- apt-get install ros-Indigo-ardrone-autonomy

- Installation of TUM_Simulator
  - Create a workspace for the TUM_simulator:
    - mkdir -p ~/tum_simulator_ws/src
    - cd ~/tum_simulator_ws/src
    - catkin_init_workspace

    - Download Dependencies:
        - git clone https://github.com/AutonomyLab/ardrone_autonomy.git
        - git clone https://github.com/occomco/tum_simulator.git
        - cd ..
        - rosdep install --from-paths src --ignore-src --rosdistro indigo -y
    - Build the simulator
        - Catkin make
    - Set the environment
        - source devel/setup.bash
- Instalation of Gazebo simulator (To get the map or environment simulation)
    - Installation of pre-built Debians:
        - sudo apt-get install ros-indigo-gazebo-ros-pkgs ros-indigo-gazebo-ros-control.
- Creation of catkin package for drone application:
        - cd ~/catkin_ws/src
        - catkin_create_pkg drone_application std_msgs rospy roscpp
        - Then, you create a .launch file in the text editor with the following:

```
<launch>
 <include file="$(find gazebo_ros)/launch/empty_world.launch">
   <arg name="world_name" value="$(find cvg_sim_gazebo)/worlds/ardrone_testworld.world"/>
 </include>
 <include file="$(find cvg_sim_gazebo)/launch/spawn_quadrotor.launch" >
   <arg name="model" value="$(find cvg_sim_gazebo)/urdf/quadrotor_sensors.urdf.xacro"/>
 </include>
</launch>
```

Executing the necessary nodes in parallel, to show the following:

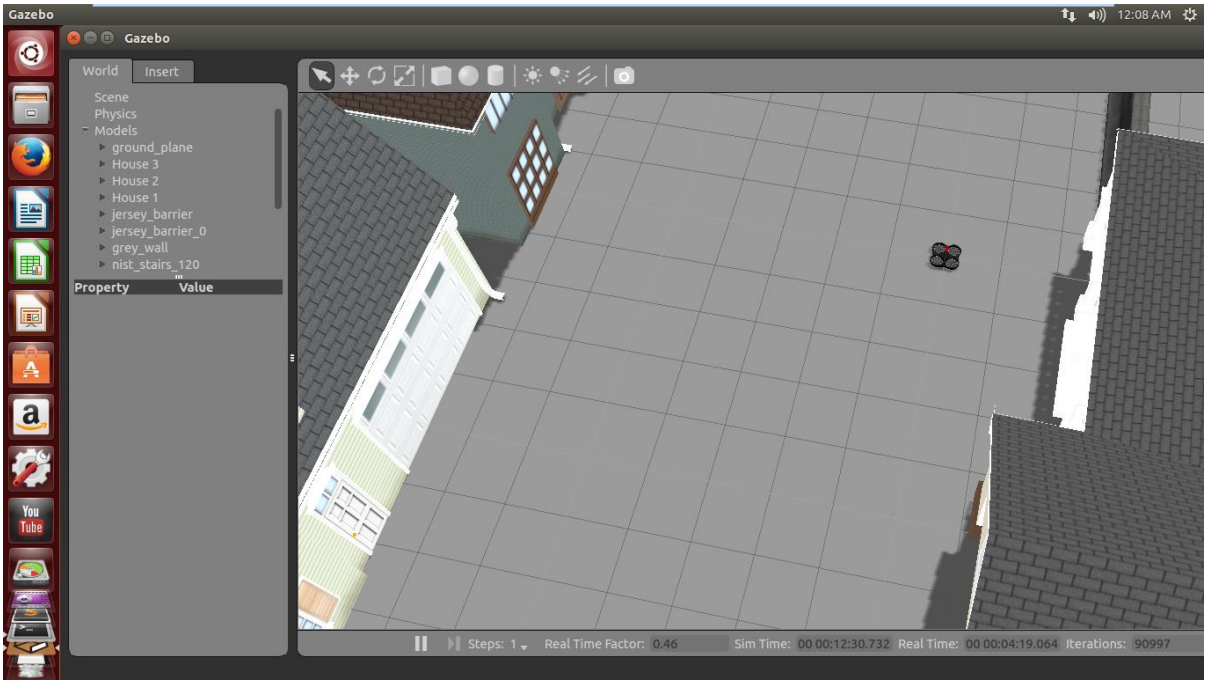


Image 2. Simulator with the quadrotor.

Then, when posting messages in cmd with the topics and appropriate message, the drone took off as shown below:

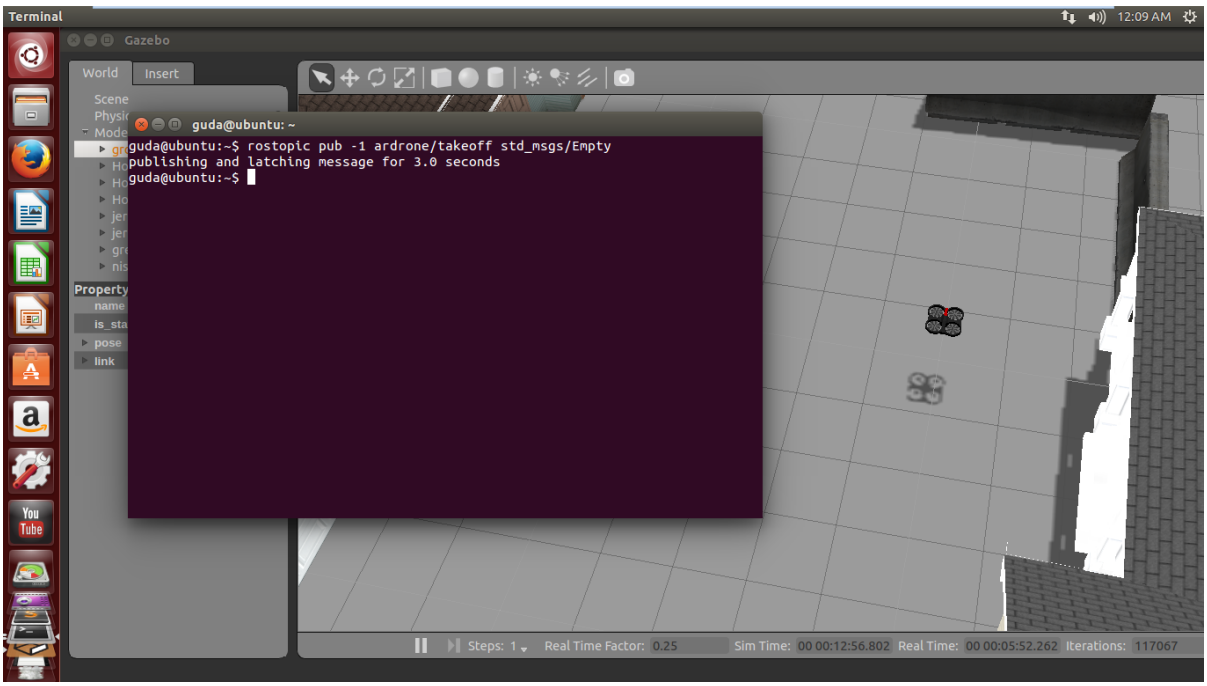


Image 3. Drone’s takeoff.

To subsequently make the landing:

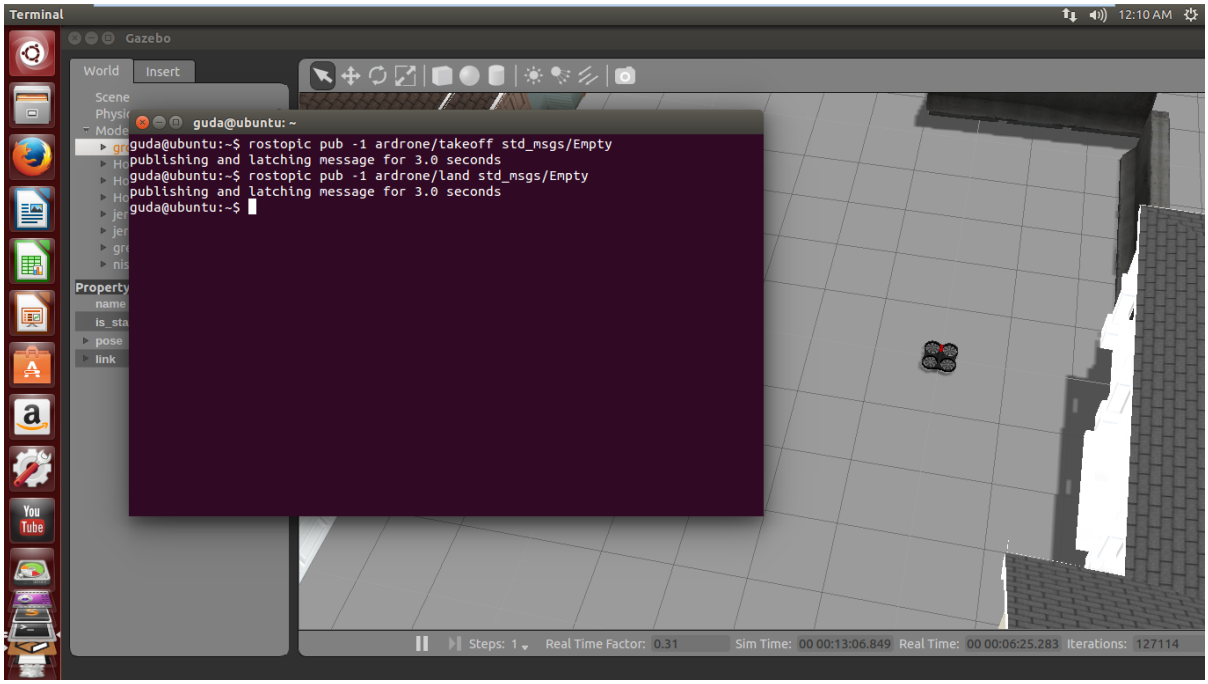


Image 4. Drone's landing.

Then, it shows how to make the drone advance by changing the pitch, through the publication of another message of the twist type in CMD, and it shows how the drone, when reaching the wall, collides with the environment, thus allowing the most closer to reality.

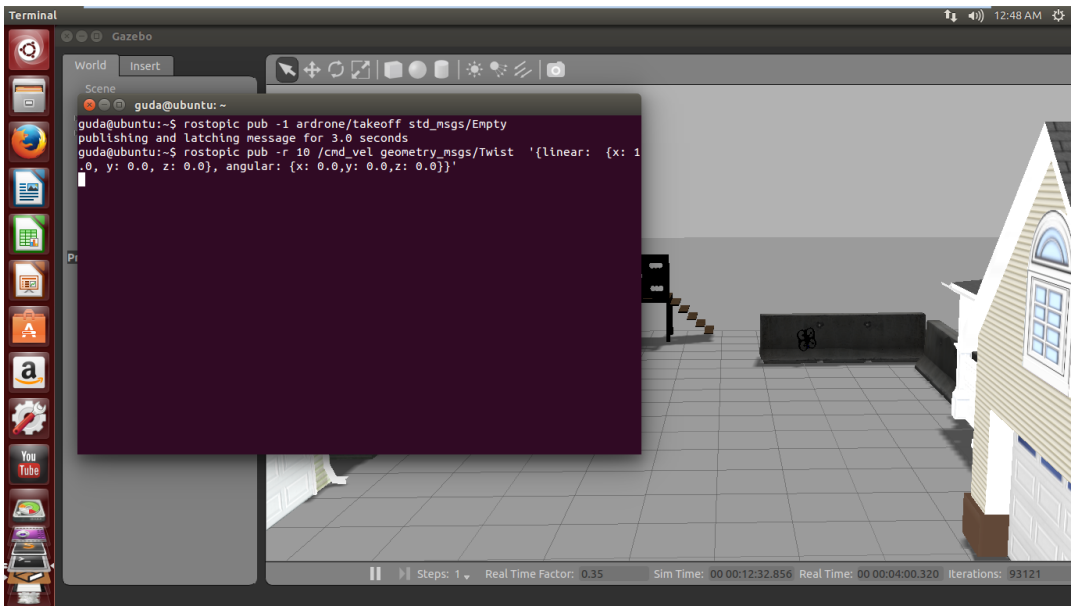


Image 5. Drone in collision.

Afterwards, utilizing a 3D map, we obtained the trajectory using the “resistive grid algorithm, here is the process:

1. We get the obstacles map as we see in next image.

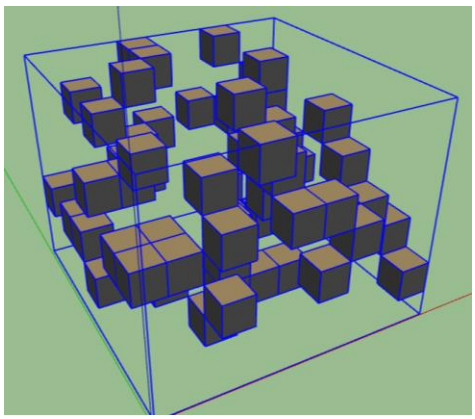


Image 6. Obstacles map.

2. Then, we obtain the trajectory in maple software in cartesian coordenates x, y and z. As we see in the next two images.

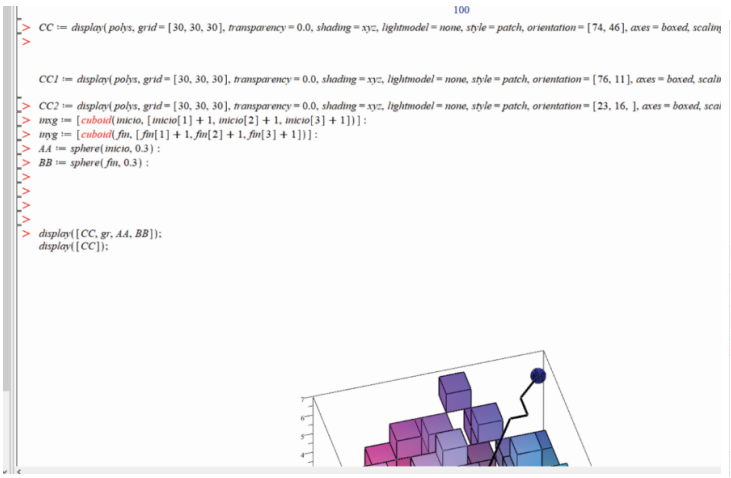


Image 7 and 8. Processing the trajectory in Maple software.

3. Then we implement the trajectory in Python programming language with feedback, so that we get accuracy in the simulation. The code as you can see is a bit long:

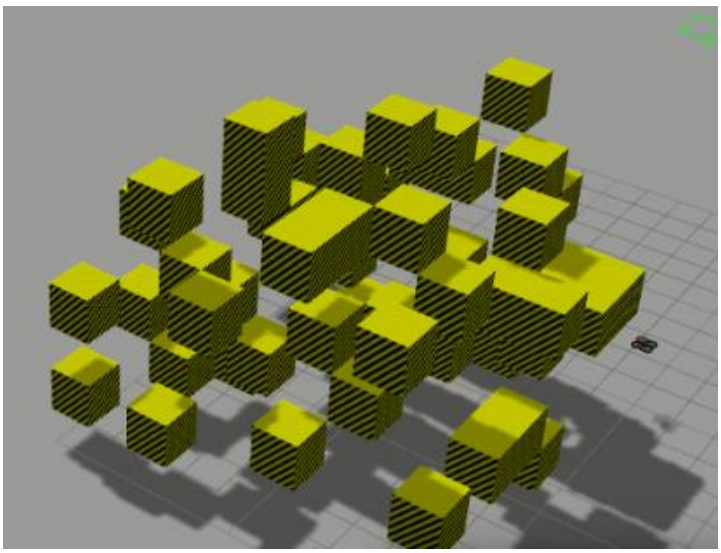
```
#!/usr/bin/env python

import rospy
import time
import math
from std_msgs.msg import String
from std_msgs.msg import Empty
from geometry_msgs.msg import Twist
from ardrone_autonomy.msg import Navdata
from nav_msgs.msg import Odometry
import numpy as np

def callback(navdata):
    global Yaw, poseZ
    Current_Time = 0.0
    poseZ = navdata.altd
    Yaw = navdata.rotZ
    if poseZ <= 40:
      poseZ = 0
    if (poseZ > 40) or (poseZ < 0):
      poseZ = (poseZ - 40)
    print(poseZ)
    return Yaw, poseZ

def Trayectoria():
      #Tamano de cola o queue size 10, permite que hayan 10 sentencias en cola
    pub = rospy.Publisher("cmd_vel", Twist, queue_size = 10)
    pub2 = rospy.Publisher("ardrone/takeoff", Empty, queue_size = 10 )
    pub3 = rospy.Publisher("ardrone/land", Empty, queue_size = 10 )

    #se asigna el atributo tipo Twist a command
    command = Twist()
    Current_Time = 0.0
    Current_Time1 = 0.0
    T = 0

    #Se declara el arreglo de puntos (x, y, z)
    #Coordenadas (1 2 6), (2 3 7), (3 4 6), (4 4 7), (5 4 7), (6 5 6), (6 6 5), (7 7 4), (8 8 3), (8 9 2), (9 9 1)
    Pts=[[0, 0, .85], [1, 1, 6], [2, 2, 6], [2, 3, 6], [3, 4, 7], [4, 5, 6], [5, 6, 5], [6, 7, 4], [7, 6, 3], [8, 6, 3], [9, 6,
3], [9, 6, 2],[10, 7, 1], [10, 8, 1], [9, 9, 1]]
    print(Pts)

    rate = rospy.Rate(10) # 10hz, standard

    while not rospy.is_shutdown():
      while (T < 1):
        #Utilizando funcion de rospy para el tiempo actual
       t0 = rospy.Time.now().to_sec()

      while (Current_Time <= 2):
      #Se da tiempo para el Takeoff
         pub2.publish(Empty())
         t1=rospy.Time.now().to_sec()
         Current_Time = t1-t0
      #Termina Takeoff

       ti = rospy.Time.now().to_sec()

      command.linear.x = 0;
      command.linear.y = 0;
```

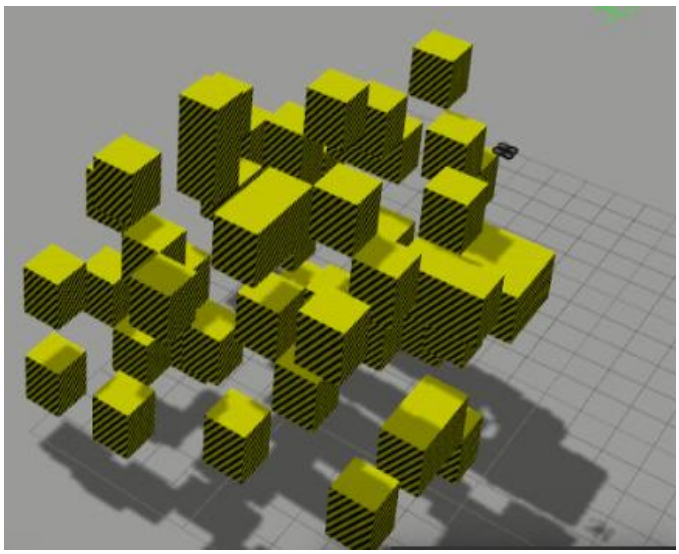
```
    command.linear.z = 0;

    while (Current_Time1 <= 1.5):
    #Se lanza el Hover
       pub.publish(command)

       #Da tiempo para hacer el hover
       tf=rospy.Time.now().to_sec()
       Current_Time1 = tf-ti
    #Termina Hover

    print(poseZ)
    print(Yaw)
    while((abs(Yaw)< 0.0 or abs(Yaw)> 0.03) and (poseZ < 835 or poseZ > 865)):
    #Mientras Yaw no sea igual a 0 no va a continuar
        while(Yaw > 0.01):
              command.angular.z = -.5
              pub.publish(command)
              print(Yaw)
        while(Yaw < 0.01):
              command.angular.z = .5
              print(Yaw)
              pub.publish(command)

    ti = rospy.Time.now().to_sec()

    command.linear.x = 0;
    command.linear.y = 0;
    command.linear.z = 0;
    command.angular.z = 0;

    while (Current_Time1 <= 1):
    #Se lanza el Hover
       pub.publish(command)

       #Da tiempo para hacer el hover
       tf=rospy.Time.now().to_sec()
       Current_Time1 = tf-ti
    #Termina Hover

    n = 17 #Numero de coodernadas
     for i in range(0, n-2):

       r = math.sqrt((Pts[i+1][0]-Pts[i][0])**2+(Pts[i+1][1]-Pts[i][1])**2+(Pts[i+1][2]-Pts[i][2])**2)#r =
root((Xi+1-xi)2+(yi+1-yi)2...)
       print("r", r, i)

       theta = math.atan2((Pts[i+1][1]-Pts[i][1]),(Pts[i+1][0]-Pts[i][0])) #theta = atan2((Yi+1-yi),(Xi+1-
Xi))
       print("theta", theta, i)

       #phi = atan2(root((Xi+1-xi)2+(yi+1-yi)2), (zi+1-zi))
       phi = math.atan2 (math.sqrt((Pts[i+1][0]-Pts[i][0])**2+(Pts[i+1][1]-Pts[i][1])**2), (Pts[i+1][2]-
Pts[i][2]))
       print("phi", phi, i)

       Vx = math.sin(phi) * math.cos(theta) #Velocidad en X para asignar al drone
       Vy = math.sin(phi) * math.sin(theta) #Velocidad en Y para asignar al drone
       Vz = math.cos(phi)                   #Velocidad en Z para asignar al drone
       print("Vx =", Vx, "Vy = ", Vy, "Vz = ", Vz)
```

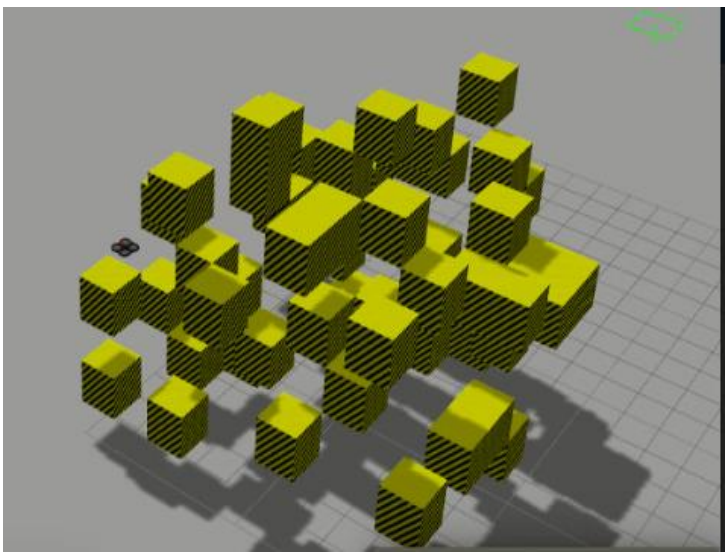
```
        Vt = ((Vx)**2+(Vy)**2+(Vz)**2)
        print("Vt=", Vt)

        Distance = r #Distancia es igual a rho

        #Se asignan las velocidades en el drone
        command.linear.x = Vx
        command.linear.y = Vy
        command.linear.z = Vz


        Current_Distance = 0
         Ti = rospy.Time.now().to_sec() #Reinicializa TiempoInicial para el siguiente while
        while((Current_Distance <= Distance) or ((poseZ/1000) != Pts[i+1][2])):

            #Publica la velocidad
            pub.publish(command)
            #Toma el tiempo actual
            Tf=rospy.Time.now().to_sec() #Tiempo Final
            #Calcula la distancia actual, teorica
            Current_Distance= Vt*(Tf-Ti)
                #print("Current Distance", Current_Distance)
                #print("Distance", Distance)
                #print("Dx", Vx*Current_Distance)
                #print("Dy", Vy*Current_Distance)
                #print("Dz", Vz*Current_Distance)


         #Hover
         command.linear.x = 0
         command.linear.y = 0
         command.linear.z = 0
        command.angular.z = 0

        Current_Time1 = 0.0

         ti = rospy.Time.now().to_sec()
        while (Current_Time1 <= 1.5):
          #Se lanza el hover
          pub.publish(command)

          #Da tiempo para hacer el hover
          tf=rospy.Time.now().to_sec()
          Current_Time1 = tf-ti
          #Termina Hover

      T = 1
    rate.sleep() #Hace que el loop se mantenga a 10 Hz




def move_forward():

      rospy.init_node('Trayectoria', anonymous=True)
      rospy.Subscriber("/ardrone/navdata", Navdata, callback)
       Trayectoria()
      rospy.spin()
```

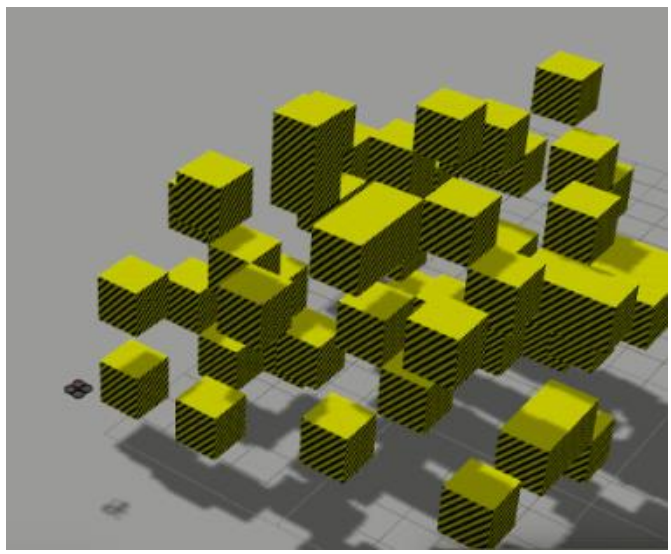
```
if __name__ == '__main__':

  try:
    move_forward()

  except rospy.ROSInterruptException:
    pass
```

4. And at last you get the following results with the implementation and simulation:









Achieved Results

In this project, the following results were obtained:

- Ubuntu, ROS Indigo and the drivers for the Parrot's AR Drone 2.0 were installed correctly.
- It was achieved that, in the simulation, a four-engine very close to reality was simulated, in addition to the simulation of the movement and creation of fictitious IP for the use of the Parrot 2.0.
- The implementation of the TUM simulator for communication between the virtual drone and ROS was achieved, in order to send CMD messages to the drone.
- The method of resistive networks was implemented to obtain an optimal trajectory from a 2D or 3D map.
- Paths generated by Python code were also made, which were subsequently modified for the use of the "resistive networks" method.

Conclusions

Doing internships at INAOE was a great professional experience in the field of mechatronics, because not only do you relate to people who can contribute much to your academic and professional training, but it allows you to grow as a person and do research related to your career, using what you previously learned in college.

The society has implemented drones for recreational or commercial use; But this has its limitations, the operative drones for this purpose have to follow rules to prevent accidents due to air traffic. Today, drones are used in many tasks, such as the ecological aspect, with monitoring and study of natural preserves with low disturbances on the environment or tree planting in arid areas. A problem with these devices over the environment is the disturbance by noise in natural and urban areas.

The use of drones is sustainable thanks to the development of new technologies that ensure a balance between costs, user experience and environmental impact. The UAV market is growing, with increasing use in recent years, and with that also came the development of several improvements that allow the use of drones in new tasks, making the research and marketing of these products future proof and very feasible.

In addition, it is important to mention that the UAV technology is also investing in the research and development of new materials, with the aim of having lighter but more resistant components that give more resistance and life to the drones, while allowing the use or in more dangerous or hostile conditions.

The manufacturability of drones is already dominated, with several companies and plants dedicated to the manufacture of these devices for military recreational purposes.

In conclusion, we learned to control a habitually recreational drone and turn it into a development tool, capable of performing complex tasks autonomously, only with the intervention of a user to give the path to follow.

References

- Geisendörfer, F. (2017). felixge/node-ar-drone. [Online] GitHub. Available in: https://github.com/felixge/node-ar-drone [Accesado 7 May 2017].
- Hamer, M. (2017). Up and flying with the AR.Drone and ROS: Getting started | Robohub. [Online] Robohub.org. Available in: http://robohub.org/up-and-flying-with-the-ar-drone-and-ros-getting-started/ [Accesado 4 Apr. 2017].
- Mexico, SCT, DGAC. (2016). Circular obligatoria que establece los requerimientos para operar un sistema de aeronave pilotada a distancia (pp. 1-47). [Online] Available in: http://www.sct.gob.mx/fileadmin/DireccionesGrales/DGAC-archivo/modulo3/co-av-23-10-r3.pdf [Accessed 12 May. 2017].
- Nodecopter.com. (n.d.). Hacker Guide. [Online] Available in: http://www.nodecopter.com/hack [Accessed 8 May 2017].
- Node.js Fundación (2017). Node.js. [Online] Nodejs.org. Available in: https://nodejs.org/es/ [Accessed 7 May 2017].
- Parrot (2010). Device for piloting a drone. WO2010061099 A2.
- Sublimetext.com. (2017). Sublime Text: The text editor you'll fall in love with. [online] Available at: https://www.sublimetext.com/ [Accessed 7 May 2017].
- Wiki.ros.org. (2017). Documentation - ROS Wiki. [online] Available at: http://wiki.ros.org/ [Accessed 20 Mar. 2017]