ÉCOLE D'INGÉNIEURS
DE L'AIR, DE L'ESPACE ET DE LA MOBILITÉ DURABLE

DRONES AND VISUAL SERVOING
Following an Aruco marker with a UAV

AU515 - Lab 3

Teachers : Jonatan Alvarez, Jamy Chahal and Johvany Gustave

CONTEXT AND OBJECTIVE OF THE LABS

The objective of these two labs is to implement a controller that sends the adequate commands to a UAV so that it follows a moving target it is able to perceive with its embedded camera.

The labs will be divided as follow:

1. Discover a Linux environment
2. Understand the basics of Robot Operating System (ROS2)
3. Implement the strategy in a simulated environment using ROS2
4. Test the implemented strategy on a real UAV

SUBMISSION

You need to form **groups of four students**. By the end of the labs, you will have to submit on Moodle:

- A PDF report containing the answers to all the questions. You can also add pictures from the web shell, Gazebo or your code.
- The modified codes **aruco_focus.py** and **aruco_lab.py**, commented
- (Optional) A video from Gazebo of the stabilized UAV.

The final grade will take into account the quality of:

- The proposed solution
- The proposed improvements
- The implementation

DESCRIPTION OF THE UAV

During the labs, you will manipulate a Tello drone, a UAV with four motors (quadrotor) as described in Figure 1.
This UAV embeds a camera and a vision-based positioning system. Thanks to its positioning system and its advanced flight controller, the Tello drone can fly in indoor environments.

Figure 1: Tello drone EDU. Picture taken in our Lab.

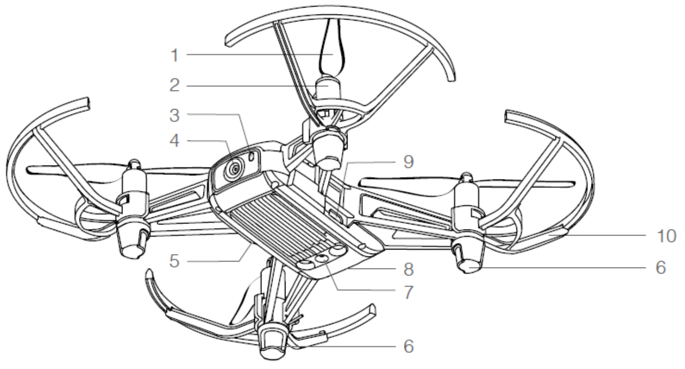


Figure 2: Tello drone - components [Official documentation]

Figure 2 describes the different components of the Tello drone:

1. Propellers
2. Motors
3. System state LED
4. Camera
5. Power button
6. Antennas
7. Vison-based positioning system
8. Battery
9. Micro USB port for charging the battery
10. Protection for the propellers

The different steps of the labs are described in the following sections.

1 INTRODUCTION TO LINUX

During these two labs, you will work on a Linux environment that will be accessed using one of the following options:

- Ros Development Studio (RDS): an online tool that allows you to work on a Linux environment without installing nothing on your local machine.
- Windows Subsystem for Linux (WSL) : create a Linux partition in Windows environment.
  The computer's resources are shared. The installation instructions are available in Appendix 5.
- Virtual Machine (WSL) : create a Linux partition within Windows. The computer's resources are split. The installation instructions are available in Appendix 5.
- Dual-Boot or native Ubuntu : create a Linux partition outside Windows (the installation is not covered).

1.1 Terminal - web shell

A terminal is a process that emulates a console in a graphical interface. Terminals allow you to run commands. In ROS Development Studio, terminals are substituted by **web shells**. **During the labs, each instruction you will have to enter in the terminal will be written manually, not copied-pasted from the PDF, otherwise you might face some copying issues.**

1.2 mkdir

This command makes a directory in your present working directory (pwd). For example, the following command will create the directory `ros2_ws` in your pwd (with the subfolder src).

```
mkdir -p ~/ros2_ws/src
```

1.3 ls

This command is used to list down all the directories and files inside the pwd.
You can also enter the path of a specific directory you want to list, as follow:

```
ls  ~/ros2_ws/src
```

1.4 cd

This command allows you to change directory. For instance, the following command will change the pwd to ros2_ws:

```
cd  ~/ros2_ws
```

The following is used to quickly move to the home directory:

```
cd
```

1.5 source

To make it simple, source is a command used to locate your project and/or define global variables, functions... For example, the following command allows you to indicate where your computer should load a set of instructions to "enable" ROS2 galactic distribution. This command will be used later (do not run it for now)

```
source /opt/ros/galactic/setup.bash
```

1.6 sudo

This key-word, which stands for *superuser do*, allows you to run commands/programs that require the security privileges of another user, by default the superuser.
For instance, you need to have the sudo privileges to install or remove packages on Ubuntu.
Most of the time, your password will be required to execute the commands. However, it is not asked on ROS Development Studio.

1.7 apt

Advanced Packaging Tool (apt) is used to install, remove or update packages (and even more). As explained in the previous section, you need the superuser privileges.
For instance, the following command will look for packages updates and then install the package `htop`:

```
sudo apt update && sudo apt install htop
```

You can now run the command:

```
htop
```

It allows you to see details about the Ubuntu system such as the processes run by the CPU, the memory utilization, etc...

2 ROS2 - BASIC COMMANDS

2.1 Introduction

Robot Operating System (ROS) is a set of software libraries and tools that help the programmer to build robotic applications. In order to implement robot behaviors, ROS mainly uses (1) nodes, (2) topics, (3) messages and (4) services.

1. nodes: A node is an executable that uses ROS to communicate with other nodes.
2. topics: Nodes can publish messages to a topic as well as subscribe to a topic to receive messages.
3. messages: ROS data type used when subscribing or publishing to a topic or while using a service.
4. services: A client / server relationship between two nodes.

In this lab, you will use ROS2, a distributed structure where each node can run independently, compared to ROS1, where each node is linked to a master.

2.2 Clone the project

Run your first ROS project, open a terminal and run the following instruction:

```
git clone https://github.com/KiloNovemberDelta/tello_ros_script
```

2.3 Install ROS2 for WSL or VM users

If you are using ROS Development Studio, go directly to section 2.4. For WSL and VM users, ROS2 is not natively installed on your machine. Use the following instruction, on your Linux environment, to install ROS2:

```
cd ~/tello_ros_script && ./install_galactic.sh
```

2.4 Install the project

```
cd ~/tello_ros_script && ./install.sh
```

2.5 Run the simulation

In the same terminal, enter this instruction:

```
cd ~/tello_ros_script && ./run_gazebo.sh
```

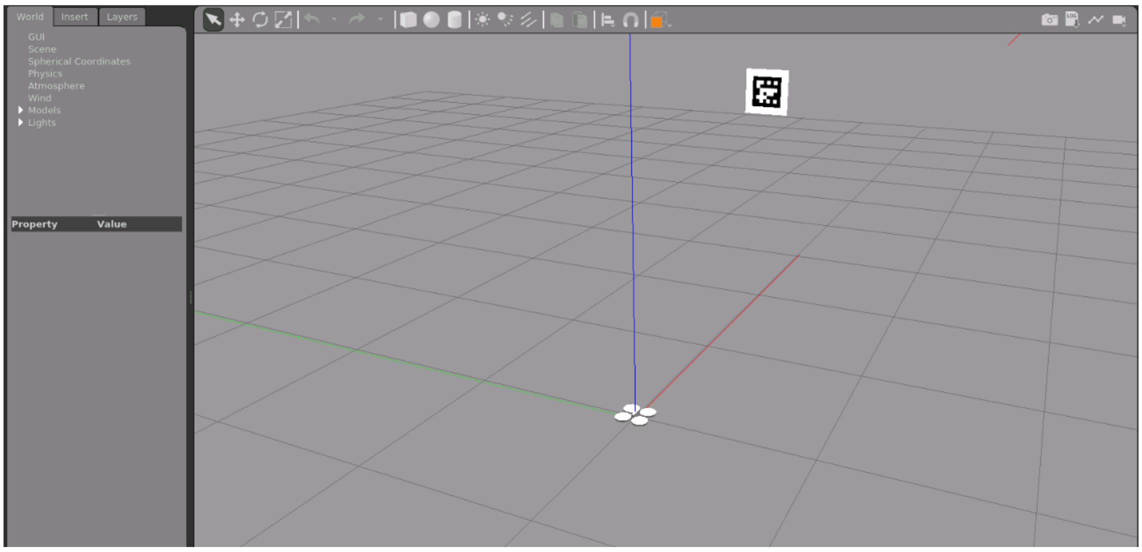


Figure 3: Simulated environment

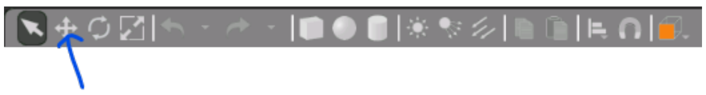


Figure 4: Move objects with Gazebo's tool

On ROS Development Studio, if you do not see anything, it is normal. **To visualize the Gazebo environment, you need to click on the button *Open Gazebo*.**
The simulated environment will be then displayed on Gazebo as shown in Figure 3.

2.6 Gazebo - Manipulation

In order to be familiar with the simulator (Gazebo), try to zoom in, zoom out, move and rotate the camera.
You can also move the UAV or the Aruco marker by selecting the arrow in the top left corner of Gazebo's interface and click on the object:

2.7 Topics - Image analysis

Open a second tab in your terminal. To be on the same workspace as the running simulation, you need to source it as follow:

```
source ~/tello_ros_ws/install/setup.bash
```

Whenever you open a new terminal, you need to repeat the previous instruction.

To list all the current topics on your ROS2 application, run the following instruction:

```
ros2 topic list
```

1) Identify which topic can provide the description of the drone's camera.

. . . .

2) Identify the type of message used in this topic by typing:

```
ros2 topic type [the camera information topic]
```

. . . .

3) Analyze the structure of the message used in the camera information topic with:

```
ros2 interface show [the camera information message_type]
```

. . . .

4) Which variables provide the resolution of the image?

. . . .

5) Which variable provides the intrinsic matrix of the camera (focal lengths fx, fy and principal point cx, cy)?

. . . .

6) Subscribe to the topic and identify the intrinsic matrix value as well as the resolution:

```
ros2 topic echo [the camera information topic]
```

. . . .

7) Identify which topic can provide the image flow from the UAV's camera

. . . .

8) Get the frequency of the received image (frames per second)

```
ros2 topic hz [the image topic]
```

. . . .

2.8 Topics - Command analysis

Now, you are familiar with the basic commands related to topics (info, echo, etc...).

1) Find the topic where velocity commands can be sent to the UAV

. . . .

2) Identify the message type expected by this topic

. . . .

3) What is the structure of this message? (how is it composed?)

. . . .

2.9 The service to takeoff and land

In order to ask the drone to takeoff or land, a service is used instead of topics.

To get the list of all the services available, type:

```
ros2 service list
```

1) The service used is */drone1/tello_action*. **Identify the message type with:**

```
ros2 service type /drone1/tello_action
```

. . . .

2) Identify the type of input expected by this service and the different output possible:

```
ros2 interface show [message_type]
```

. . . .

While running the simulation, execute the following instruction in the second terminal:

```
ros2 service call /drone1/tello_action tello_msgs/TelloAction "{cmd :
   'takeoff'}"
```

3) What is the response from the service? What does it mean? What happened in the simulation?

. . . .

4) Execute again the same instruction to takeoff. What is the new response from the service? What does it mean?

. . . .

After taking off, execute the following instruction to land:

```
ros2 service call /drone1/tello_action tello_msgs/TelloAction "{cmd :
    'land'}"
```

5) What is the response from the service? What does it mean? What happened in the simulation?

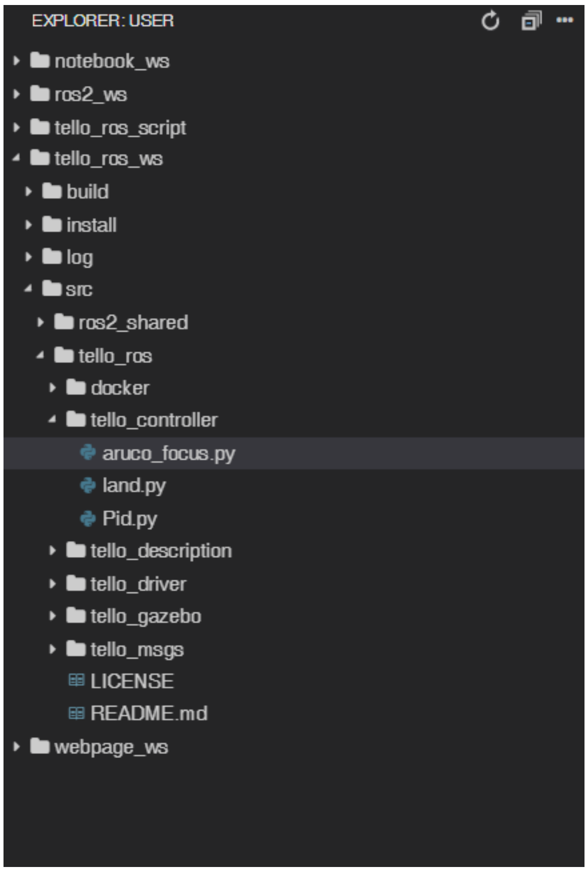
. . . .

3 SIMULATION - IMPLEMENTATION OF THE STRATEGY

Instead of using only ROS2 commands in the terminal, it is better to interact with the UAV directly with a Python script. In this section, you will implement a controller to move the drone.

3.1 Takeoff and land with a Python script

Open the `Code editor` and edit the file **aruco_focus.py**



```
EXPLORER: USER
▸ notebook_ws
▸ ros2_ws
▸ tello_ros_script
▾ tello_ros_ws
  ▸ build
  ▸ install
  ▸ log
  ▾ src
    ▸ ros2_shared
    ▾ tello_ros
      ▸ docker
      ▾ tello_controller
          aruco_focus.py
          land.py
          Pid.py
      ▸ tello_description
      ▸ tello_driver
      ▸ tello_gazebo
      ▸ tello_msgs
        LICENSE
        README.md
▸ webpage_ws
```

Figure 5: How to edit the file aruco_focus.py

1) Analyze the class ActionManager. What is its purpose?

2) Read the code in the `main` function. What is the sequential action process of the mission?

Open a new tab in your terminal an run:

```
cd  ~/tello_ros_ws/src/tello_ros/tello_controller
python3 aruco_focus.py
```

3) What happens to the UAV?

. . . .

To stop the code in the terminal, press `Ctrl + C`.

4) How does the UAV react? Identify the part of the code that executes this action.

. . . .

3.2 Move the UAV after taking off

The class `Controller` is used to move the UAV. The class subscribes to the image topic with the method `image_callback` and publishes the desired velocities with the method `cmd_vel_loop`. Modify only the attributes self.vx, self.vy, self.vz and self.v_yaw.

- A positive value of self.vx should move the drone forward, along its **X** axis.
- A positive value of self.vy should move the drone left, along its **Y** axis.
- A positive value of self.vz should move the drone up, along its **Z** axis.
- A positive value of self.v_yaw should rotate the drone clockwise, around its **Z** axis.

To have a little practice, add the following instruction at line 92, after the TODO instruction:

```
self.vy = 0.1
```

Run the python script. You should see the UAV moving left as long as the Aruco marker is visible by the UAV's camera. Otherwise, the UAV is not moving at all.

3.3 Get the relative pose of the UAV with the Aruco marker

Modify the code after the `TODO` to get the relative pose of the Aruco marker within the camera reference frame.

Use your code from Lab 2 and modify the right parameters. In Gazebo, the Aruco marker has a dimension of 0.2m x 0.2m.

To evaluate the transformation and the rotation, do not send the velocity commands. Instead, change directly the pose of the Aruco marker in the simulated environment.

If the obtained pose is coherent with the motion of the Aruco marker, you can move to the next part.

3.4 Stabilize on several axes

After getting the relative pose with the Aruco marker, you can stabilize the UAV along the different axes.
Perform the stabilization axis by axis (starting for example along the X axis, then the Y and finally the Z).
A PID library is already implemented and imported in the code. The PID class is located in the same folder.
To validate the stabilization, move the Aruco marker on Gazebo and check if the UAV follows the tag.
For each axis:

- the rising time should not exceed 2s per meter to fly
- the overshoot should not exceed 0.1m per meter to fly
- the settling time should be lower than 3s per meter to fly
- No steady-state error allowed

Be careful: The relative pose of the Aruco marker is expressed in the camera reference frame, not in the UAV's body frame.

Validate this step with your teacher before moving to the next section!

4 INTEGRATION OF THE APPROACH IN A REAL UAV

This final section consists of testing the implemented approach on a Tello drone using a computer running Ubuntu (locally).

Before testing your code with the real UAV, create a new file `aruco_lab.py` that is a copy of the original file `aruco_focus.py`. Then, adapt the code as follow:

- Set the right camera parameters mtx and dist as follows:
  - mtx: [879.064319, 0.0, 500.597355, 0.0, 876.381281, 367.439947, 0.0, 0.0, 1.0]
  - dist: [-0.068230, 0.075141, -0.005676, 0.003996, 0.0]
- Set the right marker size: 17cm.
- The qos profile of the node is different. Look for the variable `qos_policy` in the python file and update the parameters reliability and history as follow:
  - reliability=rclpy.qos.ReliabilityPolicy.RELIABLE
  - history=rclpy.qos.HistoryPolicy.KEEP_LAST

The following steps explain how to control the real UAV with your implemented approach:

1. Turn on the UAV by pressing its power button
2. Connect to the UAV's WIFI access point with your computer
3. Open a terminal and enter:

```
source ~/tello_ros_final/install/setup.bash
ros2 launch tello_driver ipsa_launch.py
```

   The previous command runs a driver that starts the communication with the Tello drone. Its camera flow should now be displayed in a window on the computer.

4. Open a second terminal and enter:

```
cd ~/tello_ros_final/src/tello_ros/tello_controller
python3 aruco_lab.py
```

5 BONUS: NAVIGATION BASED ON MARKER DETECTION

Once the previous steps have been completely validated by your teachers, you can work (not mandatory) on this new problem.
The objective is to implement a strategy that makes the simulated UAV follow a path defined by 4 Aruco markers whose identifiers differ from each other as shown in Figure 6.

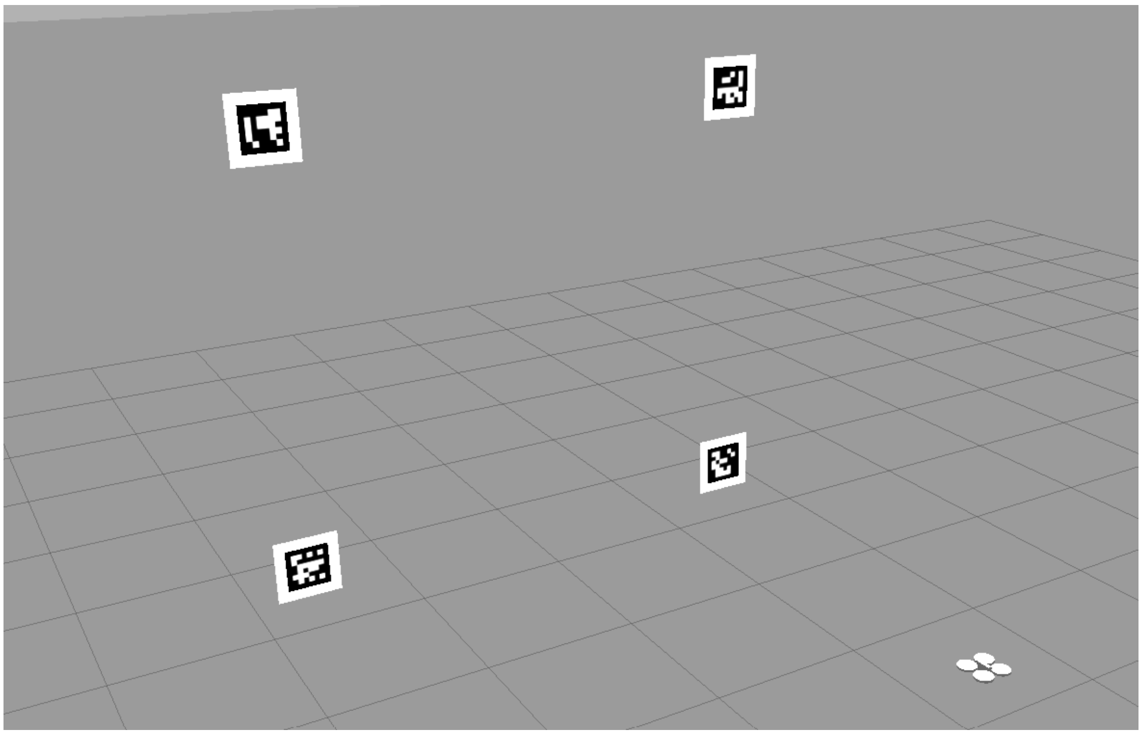


Figure 6: Simulated environment for the bonus

You will have to create a new python script `mission_2.py` that is a copy of the original file `aruco_focus.py`. Then adapt the code accordingly.

To run the new simulated environment, open a terminal and enter:

```
cd  ~/tello_ros_script && ./run_mission_2.sh
```

Appendices

Windows Subsystem Linux (WSL) installation

The WSL allows to use a Linux kernel within Windows. We use the WSL2. The installation instructions are available here.

1. Open a Powershell terminal, with administrator privileges. Then write within:
   ```
   wsl --install -d Ubuntu-20.04
   ```
   Follow the instructions.
2. Open an Ubuntu terminal (the application is called Ubuntu-20.04 on Windows). You are on Linux.
3. Try a GUI app, for example:
   ```
   sudo apt install gedit
   gedit
   ```

If there is no GUI (visual) application, then do the following:

1. Once upon, modify the .bashrc file with the nano command, as follows:
   ```
   nano ~/.bashrc
   ```
   Go at the end of the file, and copy past:
   ```
   export LIBGL_ALWAYS_INDIRECT=0
   export DISPLAY=$(awk '/nameserver / {print $2; exit}'
      /etc/resolv.conf 2>/dev/null):0 # in WSL 2
   ```
2. Install VcXsrv Windows X server from this website
3. Once executed:
   - Select "Multiple Windows, Display Number -1" (NEXT)
   - Select "Start no client" (NEXT)
   - Select "clipboard ; Primary Selection; Native opengl" (RUN)

Any graphical software should appear now, like firefox, gedit or Gazebo.

Virtual Machine (VM) installation

The virtual machine (VM) will create a Linux partition within Windows. However, it will split the resources (CPU, GPU, memory, hard disk etc.). Therefore, we recommend VM only if you have enough resources on your computer. Otherwise, we recommend to use online Ros Development Studio. To configure your virtual machine, follow the instructions from the file VM.pdf available on Moodle in the folder TP2_resources.