UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

ARTIFICIAL INTELLIGENCE ON A DRONE

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Albert Borras Pons

In partial fulfilment

of the requirements for the degree in

TELECOMMUNICATIONS ENGINEERING

and Beihang University of Beijing, China

Advisors: Miguel Jesus Garcia Hernandez

and Zhang Baochang

Barcelona, June 2019

Abstract

The main aim of this thesis is to equip a drone with artificial intelligence so it can interact autonomously with its environment based on the image from its own camera, in a way that enables it to follow a person or to be controlled with human gestures.

In order to do so a trained neural network will be used to detect objects from the drone's image like people, faces and hands. Such object detection must guarantee a minimum frame rate so the drone can react in time.

Even though all the decision making based on the image detection and its output in form of drone movement must be editable in a single script, there must be a whole digital infrastructure behind it that enables the coordination of all of its parts.

Resum

El principal objectiu d'aquesta tesis és el de dotar a un dron amb intel·ligència artificial per a que sigui capaç d'interactuar autònomament amb el seu entorn basant-se amb la imatge de la seva pròpia càmera, de manera que el permeti seguir a una persona o ser controlat amb gestos humans.

Per a tal finalitat una xarxa neuronal entrenada s'utilitzarà per a detectar objectes de la imatge del dron com a persones, cares i mans. Tal detecció d'objectes ha de garantir un mínim de frames per segon per a que el dron pugui reaccionar a temps.

Tot i que tota la presa de decisions basada en la detecció de la imatge i les respostes en forma de moviment del dron han de ser editables en un sol script, ha d'haver-hi tota una infraestructura digital al darrera que permeti la coordinació de totes les seves parts.

Resumen

El principal objetivo de esta tesis es la de dotar a un dron con inteligencia artificial para que sea capaz de interactuar autónomamente con su entorno basándose en la imagen de su propia cámara, de manera que le permita seguir a una persona o ser controlado con gestos humanos.

Para tal finalidad una red neuronal entrenada se utilizará para detectar objetos de la imagen del dron como a personas, caras y manos. Tal detección de objetos ha de garantizar un mínimo de frames por segundo para que el dron pueda reaccionar a tiempo.

Aunque toda la toma de decisiones basada en la detección de la imagen y las respuestas en forma de movimiento del drone han de ser editables des de un solo script, tiene que haber toda una infraestructura digital detrás que permita la coordinación de todas sus partes.

Acknowledgements

First of all, I would like to recognize all the support given by my two advisors. As I have done my thesis in Beijing, China, as an international exchange program, I have had the luck to have an advisor in my home university UPC and another one in Beihang University. Their names are Miguel Garcia and Zhang Baochang respectively. They both have guided my development and trajectory through all the thesis, which I gladly appreciate.

A special thanks to Li'an Zhuo, a Chinese student who far from being part of my project or not even my friend at the beginning, he did not hesitate on helping me with his knowledge about the Ubuntu operating system. A help that came very handy when too many parts had to be combined and I didn't know where to start.

Thanks to Ilyas M'gharfaoui as well, an Italian student who helped me with the Python compilation process and with the drone simulator setup, two parts of the project that left me stuck for many days.

Finally, a huge thanks to Pol Martorell and Marco Marano, two Catalan students that did the same exchange program as me here in China. They both did not hesitate to help me with the endless drone and camera tests in which more than one person was required.

Revision history and approval record

Revision	Date	Purpose
0	05/15/2019	Document creation
1	05/20/2019	Document revision
2	05/22/2019	Document revision and pictures adding
3	05/26/2019	Pictures editing and elimination of annexes (already digitally included)

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Albert Borras Pons	albertborraspons@gmail.com
Miguel Garcia	miguel.j.garcia@upc.edu
Zhang Baochang	bczhang@buaa.edu.cn

Written by:		Reviewed and approved by:	
Date	05/26/2019	Date	05/27/2019
Name	Albert Borras Pons	Name	Miguel Garcia
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables:

1. Introduction

1.1. Objectives

The main objectives of this project is to give artificial intelligence to a drone so it can track a person and also be controlled with human gestures. For this purpose, minor independent objectives have been defined:

-Be able to connect to the drone via wi-fi with the computer and gather information from it, as its image, as well as send information to it like instructions about how to move.

-Use a trained YOLO neural network to detect objects on a video stream, it is mandatory that such video stream come from the drone or from the computer's camera

-Build a workspace in which both previous objectives can be accomplished at once

-Set up a simulation environment to test with a virtual drone so no harm may be caused if a wrong algorithm is tested on the real drone

-Create a custom package with a custom script that handles the object detection information, processes it and makes a decision about how the drone should move and send it to the actual or simulated drone

1.2. Requirements

There are a few requirements to be accomplished to consider this project a success:

-No private or paid services may be used, only open source and public

-The detection accuracy should be above 70% per second

-At least 20 frames per second must be processed as for the image detection

-The computational cost should be handled by any modern computer with a dedicated GPU (an NVIDIA GTX 1070 in this case for example)

1.3. Methods and Procedures

This project is an ongoing project in Beijing's Beihang University, which means that some procedures about how to build a workspace and how to install certain drivers have been used. Nevertheless, the application and final algorithm used has been written from scratch.

Some open libraries, algorithms, files and devices have been used:

- "Parrot Bebop 2" drone, a Parrot brand drone opened in terms of software to be easily programmed

- "ROS", it brings more direct ways to deal with a robot like a drone

- "Bebop Autonomy", a library that makes the use of a Bebop Parrot drone with ROS commands far more simple

- "YOLO" and "Darknet", an open source algorithm based on a neural network that, given certain weights for different objects, is very effective in detecting such objects on an image

- “YOLO and Darknet Weights”, on internet there are many weights for YOLO that come from thousands of images specifically trained for detecting objects like a person, different weights have been download according to the application

- “darknet_ros”, a library made to use all the YOLO potential but made compatible to be used with ROS

-“CUDA” and “cuDNN”, NVIDIA GPU based accelerators for image detection

- “Catkin”, it helps with the workspace creation and management

- “Rospy”, it makes the combination of python scripting and ROS commands easier

- “Sphinx”, a Parrot simulation tool for simulating real Parrot drones

All of these will be further explained later on.

1.4. Work Plan

1.4.1. Work Packages Carried Out

WP ref: WP1
Project: General Research On Image Recognition applied on Drones
Major constituent: Information Research
Short description: Have a look on current working technologies that have achieved similar applications to see its viability and strategy.

Table 1 [Work Package 1]

WP ref: WP2
Project: General Research on Image Recognition Techniques
Major constituent: Information Research
Short description: Decide which image recognition technic suits the best for our purpose.
Internal task T1: Summary of different techniques with their pros and cons. Internal task T2: Chose which one of them is the appropriate one.

Table 2 [Work Package 2]

WP ref: WP3
Project: General Research on Robot Programming
Major constituent: Information Research
Short description: Decide which robot programming language suits the best for our purpose.
Internal task T1: Summary of different ways with their pros and cons. Internal task T2: Chose which one of them is the appropriate one.

Table 3 [Work Package 3]

WP ref: WP4
Project: Aircraft Selection for the Project
Major constituent: Information Research
Short description: Decide which unmanned aerial vehicle suits the best for our purpose.
Internal task T1: Summary of different models with their pros and cons. Internal task T2: Chose which one of them is the appropriate one.

Table 4 [Work Package 4]

WP ref: WP5
Project: Receiving Aircraft Image in Real Time to the Computer
Major constituent: Software and Connectivity Prototype
Short description: Find a way to connect to the drone with no need of its factory software directly to the computer and receive its camera image in real time.
Internal task T1: Connect successfully to the drone without using its brand software. Internal task T2: Display the image in a terminal window able to process their frames.

Table 5 [Work Package 5]

WP ref: WP6
Project: Sending Commands to the Aircraft from the Computer
Major constituent: Software and Connectivity Prototype
Short description: Find a way to send moving commands to the drone.
Internal task T1: Take off. Internal task T2: Move horizontally. Internal task T2: Land.

Table 6 [Work Package 6]

WP ref: WP7
Project: Detecting People and Objects in Real Time from any Video Source
Major constituent: Software and Image Detection
Short description: Apply image recognition on the video stream received from the drone to be able to detect which region of the image is occupied by a person in real time.
Internal task T1: Detect people and objects in an image. Internal task T2: Detect people and objects at least every 10 frames of a video stream. Internal task T3: Display the same video stream but squaring the detected objects in a terminal.

Table 7 [Work Package 7]

WP ref: WP8
Project: Combining Previous Applications in a Single Autonomous Routine
Major constituent: Software and Workspace Programming
Short description: Find a way to unify the information got from the image recognition and send commands to the drone at the same time in the same routine.
Internal task T1: Create a single workspace that gives all the previous achieved results. Internal task T2: Create a script in the workspace that combines information of the image recognition and the drone communication. Internal task T3: Create a script able to give commands to the drone based on image recognition inputs.

Table 8 [Work Package 8]

WP ref: WP9
Project: Setting Up a Simulation Tool Compatible with All the Previous Work Spaces
Major constituent: Software and Simulation
Short description: Before testing the potential algorithm, we must dispose of a simulation tool that allows us to test it with no harm. Once the application works correctly in the simulation tool, we will be ready to test it in the real world, however not before disposing of a security routine that will allow us to land the drone immediately.
Internal task T1: Setting up the simulation tool itself. Internal task T2: Connecting to the virtual drone with a virtual network in order to test everything as in a real world situation. Internal task T3: Develop a subroutine that lands the drone immediately.

Table 9 [Work Package 9]

WP ref: WP10
Project: Creating Different Drone Automatic Behaviors based on Person Detection
Major constituent: Major Software Programming
Short description: Develop the final goal of the project which is to give autonomous flight to the drone depending on the people appearing in its sight.
Internal task T1: Rotate on its axis to keep the person horizontally centered in the image. Internal task T2: Move left and right to keep the person horizontally centered in the image. Internal task T3: Move back and forward to keep the person at the same distance from the drone. Internal task T4: Follow a person autonomously combining previous tasks.

Table 10 [Work Package 10]

1.4.2. Milestones

WP#	Task#	Short title	Days
1	1	AI & UAV Research	2
2	1	Image Recognition Techniques Summary	2
	2	Best Suitable Image Recognition Technique	1
3	1	Robot Programming Languages Summary	2
	2	Best Suitable Robot Programming Language	1
4	1	UAV Models Summary	2
	2	Best Suitable UAV Model	1
5	1	Root Connection with the Drone	3
	2	Drone Image Handling	4
6	1	Take Off Drone Command	1
	2	Move Horizontally Drone Commands	1
	3	Land Drone Command	1
7	1	People & Objects Detection in an Image	4
	2	People & Objects Detection in a Video Stream	3
	3	Square People & Objects on Real Time Video	3
8	1	Workspace with All Tools Combined	6
	2	Script with All Functions Combined	4
	3	Script with Image Detection Input & Drone Commands Output	4
9	1	Setting Up the Simulation Tool Itself	2
	2	Virtually Connect to the Virtual Drone	1
	3	Creating a Subroutine that Lands the Drone Immediately	1
10	1	Drone Rotation for Person Horizontal Centering	3
	2	Drone Left & Right Movement for Person Horizontal Centering	3
	3	Drone Back & Forward Movement for Constant Distance	3
	4	Drone Tracking Combining Previous Tasks	3

Table 11 [Milestones]

1.4.3. Gantt Diagram

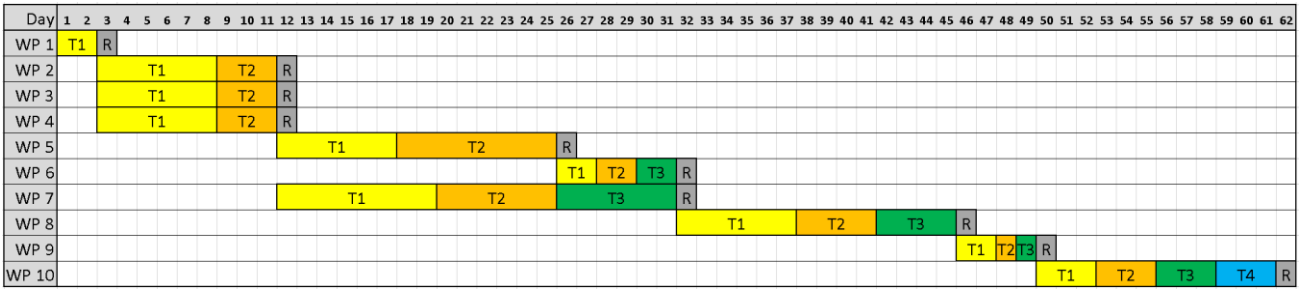


Table 12 [Gantt Diagram]

*R stands for Revision

1.5. Incidences

Unifying all the different applications (communicating with the drone, identify objects based on an image with an artificial intelligence and making automatic decisions based on the image recognition simultaneously) has turned out to be harder than expected. The actual unifying happens in a python script that merges many different libraries to handle all the aforementioned which is in a separated package of the main workspace. Once up and running major changes can easily be done, so what was expected to be quicker has turned out slower, but in contrast the final application of creating different drone automatic behaviors based on person detection has turned out to be way more agile.

Plenty of different parts have been involved in this project in form of libraries, drivers, programs, etc. All of these have a lot of versions, and not necessary they all work with each other, which has led to multiple errors because of incompatibility between them. Once the correct versions for all of them have been identified, such errors appear no more. But it has been kind of a “trial and error” process, because you cannot be certain that an error comes from an incompatibility, and deleting and reinstalling different drivers and libraries to test different combinations is a very tedious process. It has been possible to find information online about certain versions of libraries and drivers that we used that are surely known to work with each other, but such information only includes two or three of them, not all. This is the major cause for the first incidence explained which has turned out to be very time consuming, far more than expected.

Many not expected outcomes may happen, and as long as we are talking of a drone that will follow people automatically, it is not secure to try it without guaranties in the real world. So not only has been necessary to create a subroutine from which be able to land the drone immediately in case of an incidence, but also a simulator tool has been needed in order to test the drone many times with no risk. Such simulation tool is not trivial to be linked with all the previous done work.

Even with the simulation tool the real person tracking cannot be tested, as the artificial intelligence is so accurate that it needs real people to detect them as a person and not a 3D simulated model. So testing for the first time the tracking algorithm with the real drone means to have a big space where test it securely. Which means moving far from any current outlet, and as long as we only dispose of one single battery of about 10 minutes, the testing time is limited.

These incidences lead to some work plan modifications. Between the last 2 work packages, a new one has been added (WP9) to handle the simulation tool properly. Also the “Combining Previous Applications in a Single Autonomous Routine” work package took longer than expected in the same way as the “Creating Different Drone Automatic Behaviors based on Person Detection” work package took shorter than expected.

2. State of the art of the technology used or applied in this thesis:

There are already companies or organisms that have developed drones with artificial intelligence.

A very obvious application is the military and the rescuing use of them, as they are not piloted by anybody and have a very high manoeuvrability, this make them perfect for compromised operations in which a person could get harmed easily. However, there would be still the need of somebody piloting them remotely and even this for certain operations would suppose a problem. Giving the drone artificial intelligence makes the drone autonomous and capable of self-deciding, in some cases this makes them even faster and more accurate than a person controlling them.

A different approach of this application is photography, just like what the DJI company did. DJI is a camera and drone producer that has provided its newer drones with AI, they are capable of tracking the user and even perform some actions around it like orbit or taking pictures and videos with no need of a controller.

Nevertheless, all these applications and products are owned by private or closed organisms, which means that its software and method is not public. On the other side, this thesis has the requirement to rely on open source and public tools, so anyone can implement it with no need to pay for a service or a software beyond the drone and computer used themselves.

For this last matter, many parts are involved in this project, which have been mentioned in the “Methods and Procedures” section of the introduction. Now they all will be discussed to understand how they work and why they are the indicated tools for this project.

2.1. Parrot Bebop 2

“Parrot” is a drone producer company and “Bebop 2” is a model of such drones that has been selected for this project for many reasons. It only weights 500g and can outstand about 20 minutes of flight time with a brand new battery. It handles 1080p video quality which is important for our image recognition purpose. It behaves good enough even with quite high winds and it is very portable. But what is most remarkable about this drone are 4 particular facts.

-No moving parts image stabilization: It has a very wide angle spherical camera so it can get virtually all the semi sphere space in front of it, however only a small fraction of it is needed for a standard image size. This means that if the drone is facing a bit to any other direction because of the wind or a manoeuvre, instead of presenting the image just in front of it, it will present the portion that it was originally looking at. It can even anticipate the movement, for example if turning right, the actual drone movement will have a small delay, but as the initial righter part of the drone is already captured with the spherical camera, the image will already present the righter part of the drone’s sight without the drone moving yet. All of these leads to a stabilised and very responsive image with no moving parts, everything is software handled and this makes the drone far cheaper than those with mechanical stabilization that need moving parts. It is indeed not as good as mechanical stabilized image but for our purpose is more than enough.

-Cost: As aforementioned, this drone has a very good prize quality relationship, which mainly comes from its simple design. It depends from which country is bought, but generally can be found for around 500$ (less than 500€).

-Programmability: Even the Parrot company itself encourages advanced parrot drone users to develop applications for their drones. Which not only means that they provide a lot of very useful information about this topic on their website but also means that they provide the ARDroneSDK3 (a software development kit specifically for their drones) to easily access and program its software.

-Simulation tool: A mere curiosity at first but a complete must later on. There are many factors that may alter the final behaviour of the drone, and not all of them are foreseen in time, which leads to unexpected errors. It is inadmissible that such errors may cause harm to the real drone or to its surroundings, and as long as we do not dispose of a secure facility to test the drone it is an important need to be able to simulate it to minimize the risk of the first try with the real drone. Not only for security though, also for agility in progress. Being able to test what has just been programmed a minute ago in the same computer is a tremendous shortcut to keep improving the algorithm without losing too much time on each change.

2.2. ROS

ROS stands for Robot Operating System and it provides all sort of tools to help software developers create robot applications. It is licensed under an open source, BSD license. By tools all the following are included: hardware abstraction, device drivers, libraries, visualizers, message-passing, package management and more. Not all of its functions are used in this project, but the main ones as device drivers, message-passing and package management have supposed a major part of the project.

2.3. Bebop Autonomy

“bebop_autonomy” is one of the aforementioned ROS drivers exclusively for Parrot Bebop 1 and 2 drones, it is based on the so commented Parrot’s official ARDroneSDK3 and it makes much easier the ROS and Bebop2 intercommunication.

2.4. YOLO and Darknet

YOLO stands for You Only Look Once, it is an algorithm based on a neural network designed to carry out object detection on an image or a video in real time. It is part of the so called “Darknet” project, which develops all sort of neural network based image detection techniques.

Other detection systems repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales and high scoring regions of the image are considered detections.

However, YOLO (Darknet) uses a completely different approach, it applies a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities. Hence the name, because on contrast to other systems, this one only applies the neural network on the full image one single time, as it only “looked” to it once.

Thereby YOLO is a state-of-the-art, real-time object detection system. It has turned out to be evenly faster, more accurate and less computer power demanding than previous state-of-the-art systems.

But what really outstands about YOLO that made it the perfect choice for this project is that not only is open source based, but it also handles the ability to change the weights very easily of the so commented neural network.

2.5. YOLO and Darknet Weights

A neural network of this kind is trained giving to it many sorted images so after a lot of them have been processed the neural network ends up with different values that define its attitude. Such values receive the name of weights.

There are many weights on internet for the YOLO neural network that are the result of a heavy training carried out by someone else based on many pictures. So as a user, anybody can download such weights and apply them to their YOLO neural network so it now detects whatever object or objects it was trained for, without the need to do the training that someone else already did.

The user can also train the network himself with his own images, this is useful if there is the need to detect something very unique like a single particular person among all the others. But for detecting common objects like generic people, hands and faces, it is useless to train the network again as there exist weights online that are the result of the training of thousands of different images, so they are much more accurate than those that any user could get with only a bunch of pictures.

As long as we only need to detect generic people, faces and hands, certain online weights work, by far, good enough as to use them. In our case we have used two different kind of weights: one to detect people and another one to detect faces and hands.

2.6. Darknet ROS

“darknet_ros” is a YOLO library, which is the same YOLO algorithm explained before but made compatible to be used with ROS.

2.7. CUDA

“CUDA” is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs). It allows to speed up computing applications by harnessing the power of GPUs. Basically it gives the GPU the ability to speed up more processes than just graphics themselves.

2.8. cuDNN

“cuDNN” stands for Cuda Deep Neural Network, it is a NVIDIA GPU-accelerated library for deep neural networks. It basically optimizes the CUDA previously commented to accelerate not general processes but only the training and image detection processes, in our case, carried out by YOLO (darknet_ros).

This enables us to acquire a much higher rate of frames processed per second than we would by only using the computer’s CPU. A high enough frame rate is important as we need to process the image input fast enough for the drone to react in time to its environment or to the user’s control.

2.9. Catkin

Catkin is the official build system of ROS, a build system is responsible for generating "targets" from raw source code that can be used by an end user. Such "targets" may be libraries, executable programs, generated scripts, etc. Basically, as we are mixing so many different resources in this project, we need a way to unify them in a single workspace in the form of different packages that can be executed easily once built, this is exactly where catkin comes in. It creates all the workspace based on the source packages allowing for better distribution, cross-compiling support and portability, as well as building multiple, dependent projects at the same time.

2.10. Rospy

Rospy is a pure Python client library for ROS. The rospy client API enables Python programmers to quickly interface with ROS topics, services, and parameters, so we can in the end wrap everything up in a single Python script. This is needed because this project has three different dependable parts that need to be processed in a single routine:

1.- Apply object detection on the image to know where the objects are.

2.- Make a decision depending on where such objects are.

3.- Send commands to the drone about how it should move depending on such decisions.

So our final Python script will have an input of the YOLO image object detection, in the same script a decision will be made upon such input, and finally there will be the output of ROS drone commands to send to the drone.

2.11. Sphinx

Sphinx is a simulator tool provided by Parrot, thought to cover the needs of Parrot engineers developing drone software. The main concept is to run a Parrot drone firmware on a PC emulating a real Parrot drone (in our case the Bebop2), in an isolated environment well separated from the host system. It is based on Gazebo to simulate the physical and visual surroundings of the drone, Gazebo is a generic robot 3D simulation tool.

3. Project development:

In this section all the actual functional steps taken during the whole project will be clearly explained.

3.1. System Setup and Installation

3.1.1. Installing Ubuntu

An Ubuntu OS (Operating System) is required, particularly the 16.04 version. A virtual machine may be used only if the computer has a lot of computational power, otherwise the image detection will struggle running on a virtual machine, so a normal installation of the OS is recommended. No matter if it's the only OS on the system or if it is a disk partition, on the second case make sure to leave enough free disk space to install all what we'll need, around 40GB is recommended (in this project everything was done on a 50GB disk partition and only turned out to be used 33GB of them). For the Ubuntu installation online guide please follow this link: https://tutorials.ubuntu.com/tutorial/tutorial-install-ubuntu-desktop-1604. However, many other online guides will work alright too.

3.1.2. Opening and Using a Terminal Window

From now on, many terminal commands will be presented. To open a terminal window in Ubuntu in which to be able to write the commands, press "CTRL + ALT + T", or open the file explorer, go inside "Home" and right click on a blank space of the folder to select "Open in Terminal".

It is useful to know that in order to copy or paste something inside the terminal, instead of using "CTRL + C" and "CTRL + V", it must be used "CTRL + SHIFT + C" and "CTRL + SHIFT + V" respectively. So use "CTRL + C" to copy text from this file and use "CTRL + SHIFT + V" to paste it on the terminal.

Also there will be many times in which we'll need to move between directories. If we for example have the following folder structure: "Home>bebop_ws>src>my_package". And we open a terminal window in the workspace directory (the "bebop_ws" directory), it should look like this:

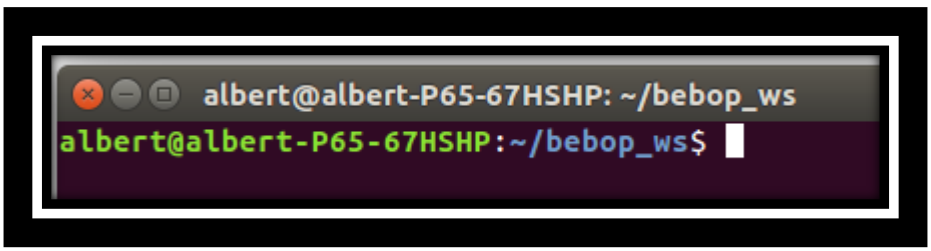


Figure 1 [bebop_ws]

The green letters ("albert@albert-P65-67HSHP") correspond to the name of the physical device, the hardware itself, in other words, the real computer that is being used. After the two dots ":" we can see the "~" sign, which represents the "Home" directory. And finally there is the "bebop_ws" that is the directory we are in. Each nested directory is separated by a "/".

Once in here we can write “cd” (which stands for “change directory”) and write the directory which we want to move to. However, it must be an existing path. For example, we could write “cd src” to move to the “src” directory, we could write “cd src/my_package” to move directly to the “my_package” directory, but we could not write directly “cd my_package” as in between “bebop_ws” and “my_package” directories there is the “src” one.

Also to move to the previous directory, we can use “cd ..”, which in this case would move us from the “bebop_ws” directory to the “Home” directory.

To know which directories are inside the current directory write “ls”, in this case after writing this we would get as the answer the “src” directory, as there is only one directory in “bebop_ws” in this example.

3.1.3. Opening and Editing a File

In many occasions we’ll need to edit certain files, such files must be opened with any text editor like “gedit”, the inbuilt Ubuntu text editor. There are cases in which the default application to open a file is not “gedit”, like when attempting to open for the first time a “.launch” file. In such cases just right click on the file and select “Open With” to select your text editor.

3.1.4. Installing ROS Kinetic

For a full guide about the installation of the ROS we’ll need (Kinetic), follow this link:

http://wiki.ros.org/kinetic/Installation/Ubuntu

Here we’ll summarize such guide writing down all the needed commands to run on the terminal in order to install it, for further information consult the link.

1.- Setup your computer to accept software from packages.ros.org:

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

2.- Set up your keys:

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-
key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

3.- Make sure your Debian package index is up-to-date:

```
sudo apt-get update
```

4.- Desktop-Full installation:

```
sudo apt-get install ros-kinetic-desktop-full
```

5.- To find available packages:

```
apt-cache search ros-kinetic
```

6.- Initialize rosdep:

```
sudo rosdep init

rosdep update
```

7.- Environment setup:

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc

source ~/.bashrc
```

If you just want to change the environment of your current shell, instead of the above you can type:

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

8.- Dependencies for building packages:

```
sudo apt install python-rosinstall python-rosinstall-generator python-ws
tool build-essential
```

With these 8 steps the ROS should be up and running in the Ubuntu OS.

3.1.5. Creating a Workspace with bebop_autonomy and darknet_ros included

Now it is time to create a workspace in which we'll develop all the work. We need this workspace to have installed the aforementioned "bebop_autonomy" ROS driver and the also already commented "darknet_ros" library. For this purpose, we will head to the online guide (note that in this case there will be minor but important changes from this online guide to take into account):

https://bebop-autonomy.readthedocs.io/en/latest/installation.html

Just like before, here we'll summarize all the terminal commands needed for the installation with the commented couple of variations from the online guide.

1.- There are 2 pre-requirements before installing it. The first one is to have ROS Kinetic installed, which we did before. The second one is to install the following packages:

```
sudo apt-get install build-essential python-rosdep python-catkin-tools
```

2.- Create and initialize the workspace (in this case we name it "bebop_ws" and we create it in the "Home" default folder of Ubuntu):

```
mkdir -p ~/bebop_ws/src $$ cd ~/bebop_ws

catkin init

git clone https://github.com/AutonomyLab/bebop_autonomy.git src/bebop_au
tonomy
```

3.- Update rosdep database and install dependencies:

```
rosdep update
rosdep install --from-paths src -i
```

4.- Before building the workspace as the website guide suggests, we first need to install the darknet_ros library (source: https://githubcom/leggedrobotics/darknet_ros)

```
cd bebop_ws/src
git clone --recursive git@github.com:leggedrobotics/darknet_ros.git
cd ..
```

5.- Now, back to the online guide, the workspace can be finally built:

```
catkin build
```

3.1.6. CUDA

Head to the download page:

https://developer.nvidia.com/cuda-downloads

And go to the “Legacy Releases” option to look for the desired version. Such version is the “CUDA Toolkit 8.0 GA2 [Feb 2017]”. Then select the following options.

-Operating System: “Linux”

-Architecture: “x86_64” (if running on a 64bits system, in this project a 32bits system has not been tested)

-Distribution: “Ubuntu”

-Version: “16.04”

-Installer Type: “runfile (local)”

Now we are ready to download the “Base Installer”.

It should look like this:

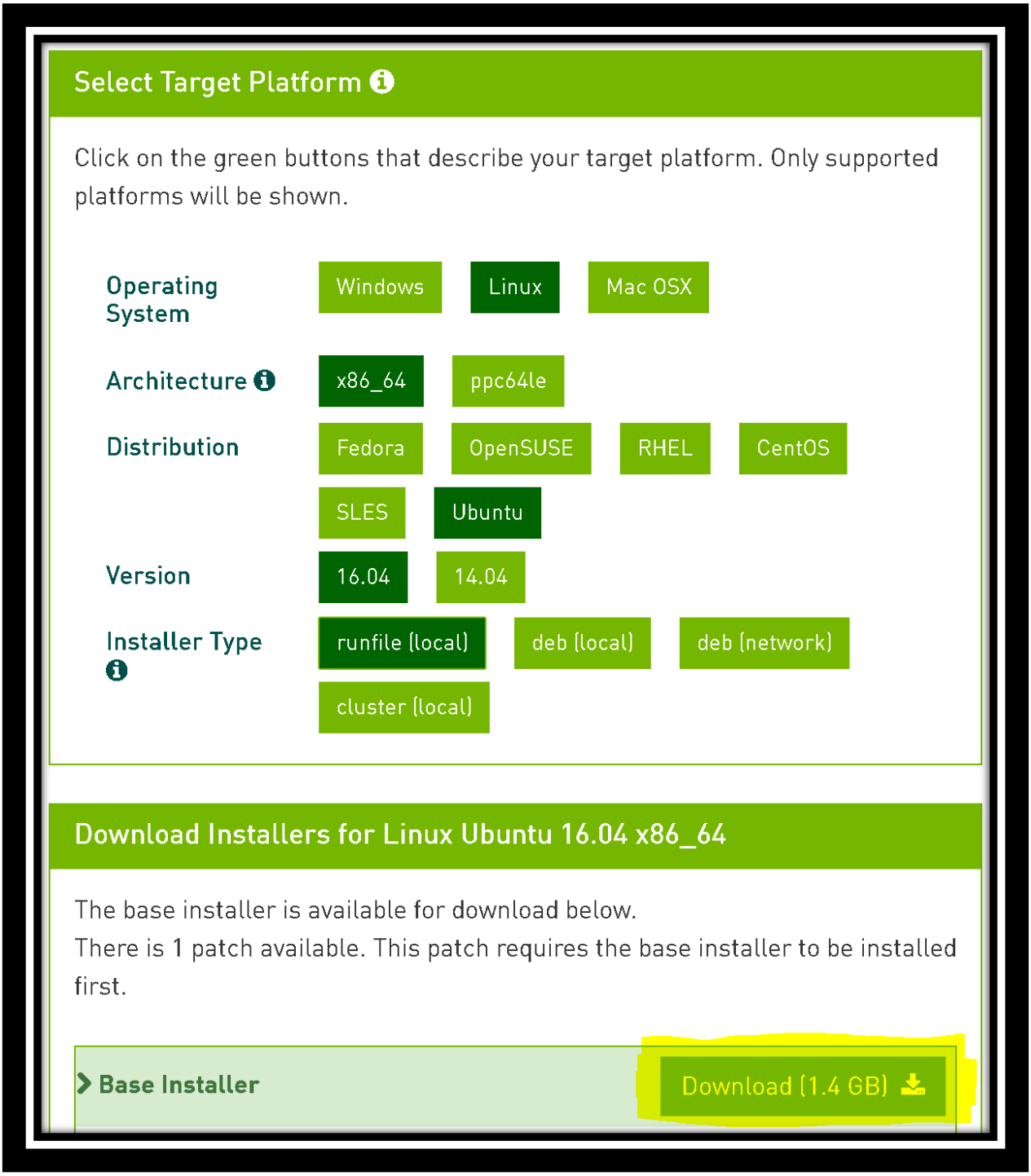


Figure 2 [NVIDIA CUDA Version]

Download and install it following the given instructions in the same download website.

As we are using many different libraries, the version of some of them are not trivial at all, as we have faced many problems because of using not compatible versions without knowing.

3.1.7. cuDNN

Head to the download page:

https://developer.nvidia.com/rdp/cudnn-download

A free membership is needed, so a mail confirmation will be required when signing up (just follow the instructions of the "sign up" option). Once in, check the "I Agree To the Terms of the cuDNN Software License Agreement" checkbox and press the "Archived cuDNN Releases" option at the end of the list.

Now you will be able to scroll down and select the desired version which is "cuDNN v5.1 [Jan 20, 2017], for CUDA 8.0":

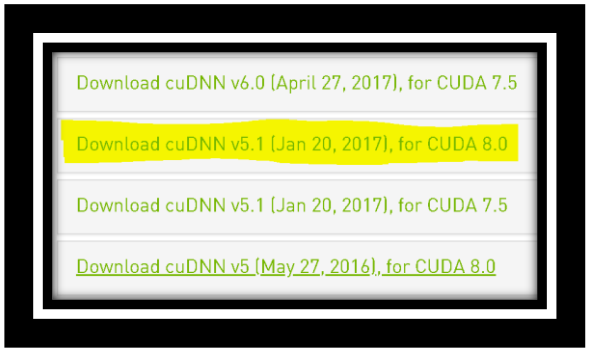


Figure 3 [NVIDIA cuDNN Version]

Choose the "cuDNN v5.1 Library for Linux" option:

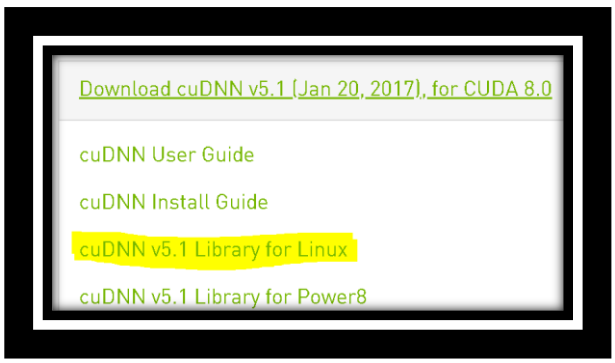


Figure 4 [NVIDIA cuDNN File Type]

Download and install it following the given instructions in the same download website.

3.1.8. Sphinx

A step by step guide can be found here:

https://developer.parrot.com/docs/sphinx/installation.html

No changes from the original guide apply in this case, so each step in the website is correct. However, just like before, a summary of the different terminal commands will be presented.

1.- Setup your computer to accept packages from Parrot's public server:

```
echo "deb http://plf.parrot.com/sphinx/binary `lsb_release -cs`/" | sudo tee /etc/apt/sources.list.d/sphinx.list > /dev/null

sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 508B1AE5
```

2.- Install the packages:

```
sudo apt-get update

sudo apt-get install parrot-sphinx
```

3.2. Configurating and Editing the Workspace

Now that everything has been set, we can start working with the actual workspace and making changes to it. None of the previous steps will be needed to be done again, as the drivers and libraries used are already installed.

From now on it will be presumed that the workspace is named "bebop_ws" and that it is located in the "Home" directory (just like showed in the previous guide). So any command used with a "bebop_ws" character string in it, must be changed to the workspace name if different.

Also, later on, it will be explained how to migrate an already existing workspace to a new one, to avoid the creation of the package re-downloading the previous libraries and drivers over and over again.

3.2.1. Changing the Image Source for the Detection

The "darknet_ros" takes an image input and opens a new window with the same image but with the detection applied to it. It basically shows the same video stream but with named boxes around the objects that it detects in real time.

Nevertheless, it must be specified which video stream take as an input. For such action we need to head to the workspace directory and navigate to:

```
/bebop_ws/src/darknet_ros/darknet_ros/config/
```

Open the "ros.yaml" file with a text editor and in the "camera_reading" section, edit the topic to **"/bebop/image_raw**" or to "**/usb_cam/image_raw**" in order to use the drone's camera or the PC's camera respectively.

Save and close the file.

3.2.2. Edit the Drone Type to be Used by bebop_autonomy

The “bebop_autonomy” is the previously commented ROS driver for the Parrot drones Bebop1 and Bebop2, so we need to specifiy which drone we’ll be using. In this case we used a Bebop2, so in order to change it we’ll head to the workspace directory and navigate to:

```
/bebop_ws/src/bebop_autonomy/bebop_driver/launch/
```

Open the “bebop_nodelet.launch” file and the “bebop_node.launch” file as well and perform the same change in both. Edit the drone type to “bebop1” or “bebop2” (in our case “bebop2”), it should look like this:

<arg name="drone_type" default="bebop1" />

Save and close the file.

3.2.3. Changing the darknet_ros Weights and Threshold

As mentioned before, one of the best aspects of the darknet_ros neural network is that it is fixed in terms of its layer and neuron structure, for different detections only the weights must be changed. This allows us to indicate the darknet_ros which file to take the weights from and use them right away with no training or restructuring. In order to do so we will do the following.

3.2.3.1. “.weights” File

Place the new “.weights” file, the “my.weights” for example, in this directory (such “.weights” files in our case have been downloaded from internet from repositories):

```
/bebop_ws/src/darknet_ros/darknet_ros/yolo_network_config/weights/
```

There can be multiple “.weights” files in this directory.

3.2.3.2. “.cfg” File

With the “.weights” downloaded file another “.cfg” file, the “my.cfg” for example, should come along with it, we need to place it here:

```
/bebop_ws/src/darknet_ros/darknet_ros/yolo_network_config/cfg/
```

Again, there can be, multiple “.cfg” files in here.

3.2.3.3. “.yaml” File

We also need to place a newly created “.yaml” file, the “my.yaml” for example, in the following path:

```
/bebop_ws/src/darknet_ros/darknet_ros/config/
```

Now there is no “.yaml” file along with the “.weights” file, so we need to create it. We will copy and paste the “yolov3.yaml” file (or any other will may work as well) that will be in this directory by default so we get a copy of it. Now let’s change the new file’s name to “my.yaml” for example, and finally we’ll open it to perform the following changes.

1.- Under “config_file”, we’ll set the “name” to “my.cfg” (or whatever name our new “.cfg” file has).

2.- Under “weight_file”, we’ll set the “name” to “my.weights” (or whatever name our new “.weights” file has).

3.- Under “threshold”, we’ll set the “value” to any number between 0 and 1. This is how sure the neural network needs to be about a detection to consider it a successful detection. For example, if we set a threshold of 0.3 and it detects a person with a certainty of 40%, it will turn out in a person detection. But if we set the threshold to 0.5, with the same 40%, it will not turn out in a person detection. If the threshold is too high, there will be times in which a miss detection occurs (when the system decided that something was not a person when it was indeed). If it is too low, a false detection may occur (when the system decided that something was a person when it was not indeed). So a good balance must be found. Surprisingly in our case, it does not make much of a difference with values between 0.2 and 0.7, so we set it at 0.3 in general.

4.- Under “detection_classes”, we’ll set the list of “names” to those objects we want to detect and that we are certain that such weights are for. For example, if the “.weights” file that we downloaded is for detecting the classes “tree”, “face” and “hand”, and we only want to detect “face” and “hand”, we’ll just write in the list this last two. But if we try to insert a class type for which the “.weights” file is not trained for, for example “car”, it will not work it out.

In this case, the “my.yaml” file should look something like this:

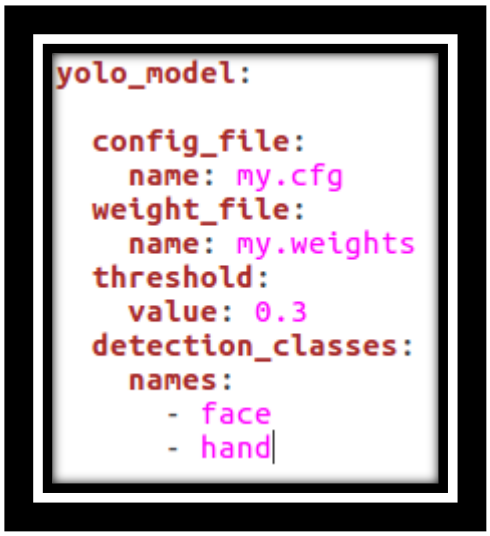


Figure 5 [".yaml" Weights File]

3.2.3.4. “.launch” File

Now we must create a “.launch” file as well, the “my.launch” for example, in the following path:

```
/bebop_ws/src/darknet_ros/darknet_ros/launch/
```

We will do the same as before, we'll take "yolo_v3.launch" file that is there by default (or any other may work as well) and we'll make a copy of it to edit it in the following way. Under the "<!-- Load parameters -->" section, we will change the second line so it has our "my.yaml" (the first line will remain with the "ros.yaml"). No more changes may be applied to this file, the edited part should look like this:

```
<!-- Load parameters -->
<rosparam command="load" ns="darknet_ros" file="$(find darknet_ros)/config/ros.yaml"/>
<rosparam command="load" ns="darknet_ros" file="$(find darknet_ros)/config/my.yaml"/>
```

Figure 6 [".launch" Weights File]

3.2.3.5. Rebuild Workspace

Finally, we must rebuild the workspace in the same way we did when building it for the first time, so we will open a terminal in the workspace directory and we'll run this command:

```
catkin build
```

3.2.4. Creating a Custom Package for our Executable Python Script

The main objective is to dispose of an executable python file integrated in the workspace so it can receive information from other packages and send new one to others as well. For this purpose, we'll follow the next steps.

3.2.4.1. Creating a New Package

To create a new package, we will open a terminal inside the workspace and we'll run this commands.

Change directory to the "src" folder:

```
cd bebop_ws/src
```

Create a package with the name of "my_package":

```
catkin_create_pkg my_package std_msgs ropsy roscpp
```

The "std_msgs" is to let it handle messaging, the "ropsy" is for Python compilation and the "roscpp" is for C++ compilation (not needed in our final application as we'll do a Python script but useful for potential C++ scripts).

Back to the workspace directory:

```
cd bebop_ws
```

Finally rebuild the workspace:

```
catkin build
```

3.2.4.2. Creating an Executable Python Script

To create a Python script inside our previously created package we'll open a terminal inside the workspace as well (we can continue using the same previous terminal window opened before to create the package).

Now we will change the directory to our package, which is inside the "src" folder:

```
cd bebop_ws/src/my_package
```

And we will create a new directory called "my_directory":

```
mkdir my_directory
```

Now, inside the new directory we just created, we can create a Python file (just right clicking and selecting "New Document > Empty Document" and changing its extension to ".py") or just paste an already existing or downloaded python script. Let's pretend that such new python file is named "my_python_file.py" for the following steps.

We still need to make it executable, so let's go inside this new directory:

```
cd my_directory
```

And run this command to make it executable:

```
chmod +x my_python_file.py
```

Head back to the workspace directory:

```
cd bebop_ws
```

Finally rebuild the workspace:

```
catkin build
```

3.2.5. Make the Workspace Compatible with the Sphinx Simulation

Our packages, libraries, files and connection expect to connect to a real Parrot Bebop2 drone, so even though the simulator pretends to be a real drone, many parameters should be changed, not only those of the simulator itself.

3.2.5.1. Get to Know the Current Wi-Fi Configuration

We need to know which network is being used as the simulator will use it to emulate the drone. For such purpose we'll open a terminal from anywhere (just typing "CTRL + ALT + T") and use the following command:

```
iwconfig
```

The output should look something like this:

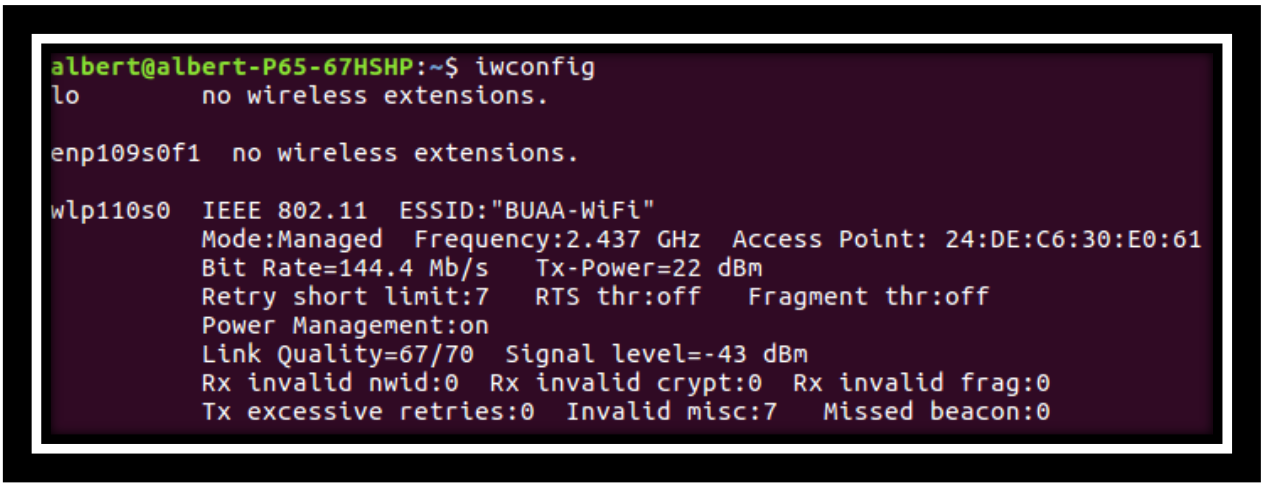


Figure 7 [In Use Wifi Network]

In here we will pay attention to the name associated to the “IEEE 802.11”, in this case would be “wlp110s0”. Many other times will be just “eth0”, and this is something that should be checked every time we use the simulator. From now one just keep in mind that the used network is, for this example, the “wlp110s0”.

3.2.5.2. “.drone” File

To simulate a particular kind of drone, the simulator uses a file with the “.drone” extension. There are many of these files and they are located here (it is outside our workspace directory):

```
/computer/opt/parrot-sphinx/usr/share/sphinx/drones/
```

Our file is the “bebop2.drone” one, as our drone model is the Bebop2, so we will copy and paste it in order to get a new “.drone” file, we will name it in this case “wlp110s0_bebop.drone”.

Now we have to make a small change to it, at the end of the document in the “<stolen_interface>” section. The first “eth0” should be changed to “wlp110s0” (as it is the name we got when we used the “iwconfig” command) so it looks like this:

<stolen_interface>wlp110s0:eth0:192.168.42.1/24</stolen_interface>

Save and close it.

Later on, when we see how to execute the simulator, there is a point in which we use the “.drone” file, is in such point that we will use our newly created “wlp110s0_bebop.drone” file.

At this point is very useful to do the same with a new file which we'll call "eth0_bebop.drone" which will be as the original one as it has the final line like "**<stolen_interface>eth0:eth0:192.168.42.1/24</stolen_interface>**". This is to avoid changing and editing files each time the network in use changes. In this way, if the next time we check our internet configuration turns out that "wlp110s0" is not in use but "eth0" is indeed, we'll just use the other "eth0_bebop.drone" file. Different computers and different networks may give more different names, it is ok to have a ".drone" file for each of them.

This change that we did to each ".drone" file is to specify to such virtual drone which network net to use.

3.2.5.3. ".launch" File

We finally need to create a ".launch" file that will be also used when executing the simulator. To get to the directory where these ".launch" files are, head inside your workspace directory to:

```
/bebop_ws/src/bebop_autonomy/bebop_driver/launch
```

Copy and paste the "bebop_node.launch" file to get a copy of it. Rename it, for example, to "bebop_node_simulation.launch" and open it.

Under "<launch>", on the second line, change the IP address to "10.202.0.1", so it looks like this:

<arg name="ip" default="10.202.0.1" />

Save and close it.

It is at this point, that just like before, it is useful to do the same with a new file that we'll name "bebop_node_normal.launch". This file will be like the original as such line we edited before will look like this: "**<arg name="ip" default="192.168.42.1" />**".

These changes that we did to each ".launch" file is because this simulator, in order to simulate the drone's connectivity, takes control over the computer's connectivity. So there is no way to connect to the virtual drone's wifi if the computer itself has no wifi available to connect with, as such wifi the computer would normally have, now it is owned by the virtual drone. To avoid the need of a new different computer with available wifi to connect to the virtual drone, what we have done with these changes is to connect with the same computer via a virtual Ethernet connection to the virtual drone, hence the new constant IP we introduced.

3.2.6. Migrating a Workspace

In many occasions, mainly while developing and testing, it is needed to test some changes on an already working workspace. So not to alter the already working workspace, a copy of it is needed. Then, in order to avoid repeating all the process we followed to create and build it, it can be copied or migrated. Now, how to do it will be explained.

First of all, we will create a new folder under the "Home" directory (where the other workspace is as well) and we will name it with the desired workspace name, "new_ws" for example.

After this, we will open the workspace folder that we want to copy and we will copy its “src” folder to paste it inside the new workspace folder. Make sure that inside the pasted “src” folder there is no “CMakeLists.txt” file, if such is the case, just delete it.

Finally, we will open a terminal window inside this new folder we have just created (inside the workspace as usual, not inside the “src” directory we have just pasted) and we’ll build the workspace normally running this command:

```
catkin build
```

3.3. Main Program Execution

Until now we have seen how to set up the computer drivers and libraries correctly and afterwards we have discussed how to edit and create files. Now it’s time to explain how to actually execute and run those processes and which protocol follow.

There are 2 different functional tested applications in this project, one that is used to control the drone with gestures (“Gesture Controls Application”) and another one used to let the drone track a person autonomously (“Person Tracking Application”). However, it is important to know how to use the simulator, so there will be 3 scenarios: “Gesture Controls Application” with the simulator and with the real drone (both using the computer’s camera), and the “Person Tracking Application” only with the real drone (using the drone’s camera).

One may wonder why not the “Person Tracking Application” with the simulator as well. And the reason is that as the artificial intelligence is so well trained, it needs actual real people to detect them as a person, so as long as we cannot introduce a real person (or at least a good enough 3D model of it) in the simulator, it makes no sense to try the person tracking algorithm in an environment where there is no person to track.

There are of course more combinations and more possible scenarios (which have been used to test different things), but as all the important commands and actions will appear in this executions, explaining them will not only explain how to actually execute this project main applications, but will also explain how to deal with all the processes and executable files.

3.3.1. Gesture Controls Application

This application allows you to control the drone with gestures using the computer’s camera, no matter if it is a real drone or a simulated one. However, the process to execute and run everything changes a little bit, this is why both options will be equally and separately explained.

3.3.1.1. Simulated Drone

1.- We’ll need a bunch of terminal windows and all of them sourced, so we’ll open them all right now. Open the workspace directory (in this example named “working_ws”) and open a terminal right clicking on a blank space and selecting “Open Terminal”. Do this seven times so we have seven terminal windows, and run the following command in all of them:

```
source /home/user_name/working_ws/devel/setup.bash
```

Where “user_name” your currently user name that is being used in Ubuntu and where “working_ws” the name of the opened workspace. To know the user name, you can go to the desktop and click the settings button on the right upper corner, under “Guest Session” you will see the current computer’s user:

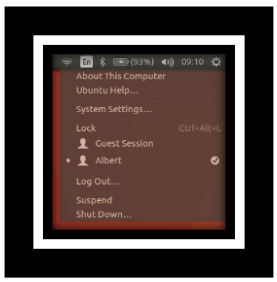


Figure 8 [Ubuntu User Name 1]

This command line that we just used is to source the terminal window itself, in other words, to let it know where it should look for the files and commands that we will input in it later on. As it must be done in every terminal window we use, it is way quicker to do it all at once at the same time.

From now on, each step will be done in one of these 7 terminal windows.

2.- In one of the 7 terminal windows we need to initialize the simulator. For this purpose, having an internet connection, we’ll run the following command (it may ask you for the computer’s password, it is completely normal, just write it down and press enter):

```
sudo systemctl start firmwared.service
```

To check that it is working properly write this:

```
fdc ping
```

If it answers with a “PONG” message it means that it is working alright. So we are ready to write the last command on this terminal window:

```
sudo firmwared
```

3.- In one of the non-already used 7 terminal windows we need to get to know the used network and actually launch the simulator. To know the currently in use network we’ll run the already seen command:

```
iwconfig
```

And pay attention to the network name associated to the “IEEE 802.11” network. Now depending on which name is associated (“eth0” or “wlp110s0” for example), we’ll run the following command:

```
sphinx --datalog /opt/parrot-sphinx/usr/share/sphinx/worlds/outdoor_1.wo
rld /opt/parrot-sphinx/usr/share/sphinx/drones/wlp110s0_bebop.drone
```

In here we are using one of the previously created “.drone” files, remember that we gave them names according to the network name. In this command we are running the “wlp110s0_bebop.drone” file for example, which is the one we edited to work with the

"wlp110s0" network name. If for any reason the network in use is any for which we did not create a ".drone" file, just go and create a file for it as explained before.

4.- In one of the non-already used 7 terminal windows we need to run the "roscore", the master ROS process that will allow the intercommunication between different ROS processes. Just run the following command:

```
roslaunch bebop_driver bebop_node_simulation.launch
```

Remember we created a "bebop_node_simulation.launch" file and a "bebop_node_normal.launch" file, in this case we are using the "simulation" one as it is for the simulator.

5.- In one of the non-already used 7 terminal windows we need to use the webcam to get an image stream, for this intention we'll run the following:

```
roslaunch usb_cam usb_cam-test0.launch
```

A new window should appear after a few seconds showing what our computer's camera is seeing.

6.- In one of the non-already used 7 terminal windows we need to initialize the darknet_ros (the YOLO image detection), just run this:

```
roslaunch darknet_ros my.launch
```

Notice that we are using the "my.launch", this is the edited ".launch" file we created. This one is made to detect faces and hands only. If you happen to have created the file with a different name, just use such name in here.

A new window should appear after a few seconds showing the same image shown by the computer's camera but with image detection applied on it, it should present a square on the detected hands and faces with the name of the detected object just like this:

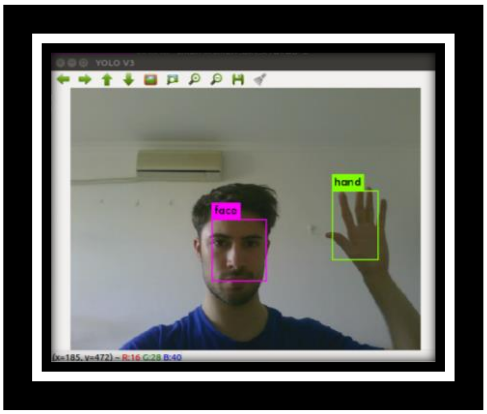


Figure 9 [Face and Hand Detection]

7.- In one of the non-already used 7 terminal windows we need to prepare a security command in case we need to land the drone immediately. What is recommended is to do the following. First take off the drone with this command:

```
rostopic pub --once bebop/takeoff std_msgs/Empty
```

Then land it again with this other one:

```
rostopic pub --once bebop/land std_msgs/Empty
```

If the drone actually took off and landed, we are sure that everything is working alright. This terminal window then, will only be in case of need to control the drone directly. For more movement commands to send to the drone from this same terminal window take a look to the “3.4 Drone Commands” topic.

8.- Finally on the last non-already used terminal window we’ll launch the python script, just running this line:

```
rosrun my_package my_better_gesture.py
```

Where “my_better_gesture.py” the name of the python script we want to launch, in this case it is the one to control the drone with gestures. If any problem occurs just press on this terminal and press “CTRL + C” to stop the process, the drone will not receive any instruction so it will just hover wherever it was, now you can freely use the previous explained terminal window to control it directly.

3.3.1.2. Real Drone

To carry out the “Gestures Control Application” with the real drone we’ll do exactly the same as we did with the simulated drone but instead of using 7 terminals we’ll just need 5. As we will not do the 2, 3 and 4 steps explained in the simulated drone explanation, and we will do this following two steps instead.

1.- Turn on the drone and connect to its wifi network from the computer, wait a few seconds until the computer recognizes the network and connect to it (no password will be required).

2.- Run this on a terminal window:

```
roslaunch bebop_driver bebop_node_normal.launch
```

Basically we did not use the first 2 terminals that were for the simulation environment and instead of using the “bebop_node_simulation.launch” file we have used the “bebop_node_normal.launch” file. However, the sourcing command for every terminal is the same (as we still are using the same workspace no matter if using a real or simulated drone):

```
source /home/user_name/bebop_ws/devel/setup.bash
```

The rest is exactly the same as before.

3.3.2. Person Tracking Application

This application is designed to give the drone the ability to track a person, such scenario would not exist in the simulator as there's no real people inside it to track. Hence the reason we only use it with the real drone. The procedure is very similar to the previous application.

1.- We'll need a bunch of terminal windows and all of them sourced, so we'll open them all right now. Open the workspace directory and open a terminal right clicking on a blank space and selecting "Open Terminal". Do this four times so we have four terminal windows, and run the following command in all of them:

```
source /home/user_name/better_track_ws/devel/setup.bash
```

Where "user_name" your currently user name that is being used in Ubuntu and where "better_track_ws" the name of the opened workspace. To know the user name, you can go to the desktop and click the settings button on the right upper corner, under "Guest Session" you will see the current computer's user:

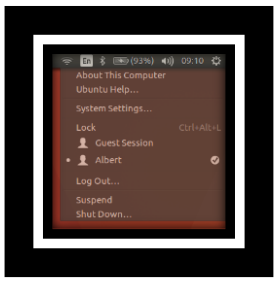


Figure 10 [Ubuntu User Name 2]

This command line that we just used is to source the terminal window itself, in other words, to let it know where it should look for the files and commands that we will input in it later on. As it must be done in every terminal window we use, it is way quicker to do it all at once at the same time.

From now on, each step will be done in one of these 4 terminal windows.

2.- Turn on the drone and connect to its wifi network from the computer, wait a few seconds until the computer recognizes the network and connect to it (no password will be required).

3.- In one of the non-already used 4 terminal windows we need to run the "roscore", the master ROS process that will allow the intercommunication between different ROS processes. Just run the following command:

```
roslaunch bebop_driver bebop_node_normal.launch
```

Remember we created a "bebop_node_simulation.launch" file and a "bebop_node_normal.launch" file, in this case we are using the "normal" one as it is for the real drone. If any other name has been used for this ".launch" file use it here, or even if no ".launch" file has been created the "bebop_node.launch" (the default one) one will probably work alright.

4.- In one of the non-already used 4 terminal windows we need to initialize the darknet_ros (the YOLO image detection), just run this:

```
roslaunch darknet_ros yolo_v3_tiny.launch
```

Notice that we are using the “yolo_v3_tiny.launch”, this is the default “.launch” file which detects many things (people is one of them).

A new window should appear after a few seconds showing the drone’s perspective with image detection applied on it, it should present a square on the detected objects with the name of them.

5.- In one of the non-already used 4 terminal windows we need to prepare a security command in case we need to land the drone immediately. What is recommended is to do the following. First take off the drone with this command:

```
rostopic pub --once bebop/takeoff std_msgs/Empty
```

Then land it again with this other one:

```
rostopic pub --once bebop/land std_msgs/Empty
```

If the drone actually took off and landed, we are sure that everything is working alright. This terminal window then, will only be in case of need to control the drone directly. For more commands to send to the drone from this same terminal window take a look to the “3.4 Drone Commands” topic.

6.- Finally on the last non-already used terminal window we’ll launch the python script, just running this line:

```
rosrun my_package my_better_track.py
```

Where “my_better_track.py” the name of the python script we want to launch, in this case it is the one to make the drone track a person. If any problem occurs just press on this terminal and press “CTRL + C” to stop the process, the drone will not receive any instruction so it will just hover wherever it was, now you can freely use the previously explained terminal window to control it directly.

Take into account that in order to make the drone track people with this script, it must be flying already, so the take off and land commands must be sent through the previous terminal window directly.

3.4. Drone Commands

As we have seen, we can send commands directly to the drone, in here we'll summarize which are these main commands.

Take off:

```
rostopic pub --once bebop/takeoff std_msgs/Empty
```

Land:

```
rostopic pub --once bebop/land std_msgs/Empty
```

Perform a movement:

```
rostopic pub -r 10 /bebop/cmd_vel geometry_msgs/Twist '{linear: {x: A, y
: B, z: C}, angular: {x: 0.0, y: 0.0, z: D}}'
```

Where A, B and C, a value between -1 and 1, to move in the X, Y and Z axis respectively. Where D a value between -1 and 1 as well, to turn on its Z axis (the rest of X and Y angular values are not used). 0 would be no movement, 1 the maximum velocity in one way and -1 the maximum velocity in the opposite way. Knowing this the following example commands can be understood.

Move forward:

```
rostopic pub -r 10 /bebop/cmd_vel geometry_msgs/Twist '{linear: {x: 1.0,
 y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}'
```

Move backwards:

```
rostopic pub -r 10 /bebop/cmd_vel geometry_msgs/Twist '{linear: {x: -1.0
, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}'
```

Move up:

```
rostopic pub -r 10 /bebop/cmd_vel geometry_msgs/Twist '{linear: {x: 0.0,
 y: 0.0, z: 1.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}'
```

Move down:

```
rostopic pub -r 10 /bebop/cmd_vel geometry_msgs/Twist '{linear: {x: 0.0,
 y: 0.0, z: -1.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}'
```

Move left (notice that left corresponds to positive Y linear values):

```
rostopic pub -r 10 /bebop/cmd_vel geometry_msgs/Twist '{linear: {x: 0.0,
 y: 1.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}'
```

Move right (notice that right corresponds to negative Y linear values):

```
rostopic pub -r 10 /bebop/cmd_vel geometry_msgs/Twist '{linear: {x: 0.0,
 y: -1.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}'
```

Turn counterclockwise (notice that counterclockwise corresponds to positive Z angular values):

```
rostopic pub -r 10 /bebop/cmd_vel geometry_msgs/Twist '{linear: {x: 0.0,
 y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.0}}'
```

Turn clockwise (notice that clockwise corresponds to negative Z angular values):

```
rostopic pub -r 10 /bebop/cmd_vel geometry_msgs/Twist '{linear: {x: 0.0,
 y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: -1.0}}'
```

All of these movement examples are to get to know the generic movement command, but it does not mean that one single value can be introduced at a time, all of them can be combined in one single command to accomplish different kind of movements. As a final example:

```
rostopic pub -r 10 /bebop/cmd_vel geometry_msgs/Twist '{linear: {x: 0.5
y: -0.5, z: 0.01}, angular: {x: 0.0, y: 0.0, z: 0.0}}'
```

This would make the drone move diagonally (forward and to the right at the same time) at half its maximum velocity at the same time it moves up very slowly.

3.5. Main Program Explanation

Until now we have seen how to set up the computer drivers and libraries correctly, afterwards we have discussed how to edit and create files and finally we have explained how to actually execute and run those processes. In this section though, we will discuss the concept behind the 2 Python scripts, in other words, what should they do once executed and why. Therefore, it doesn't matter if they are applied on a real or simulated drone.

3.5.1. Gesture Controls

The main objective of this script is to be able to control the drone standing in front of a camera (the computer's camera in this case) and doing gestures with the hands relative to the face. Before explaining the script itself, let's explain how it is physically controlled with gestures through this table (assuming a centered face):

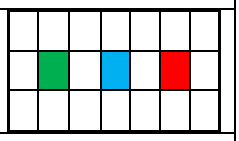
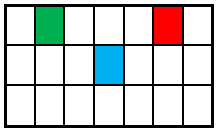
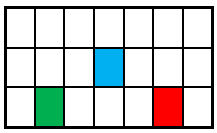
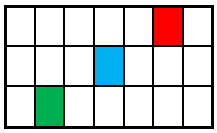
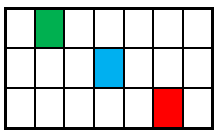
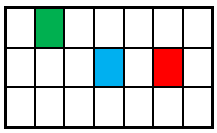
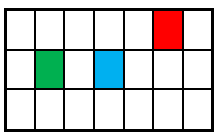
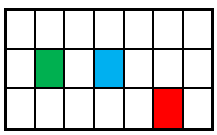
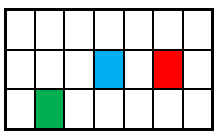
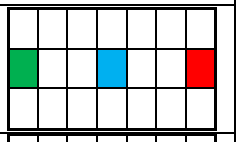
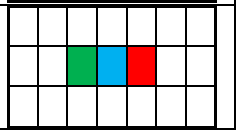
Vertical Hand Position		Horizontal Hand Position		Drone's Movement		
Left Hand	**Right Hand**	**Left Hand**	**Right Hand**	**Name**	**Symbol**	**Mirrored Position (left hand, face, right hand)**
Centered	Centered	Centered	Centered	Idle	●	
Up	Up	Centered	Centered	Forward	↑	
Down	Down	Centered	Centered	Backward	↓	
Down	Up	Centered	Centered	Left	←	
Up	Down	Centered	Centered	Right	→	
Up	Centered	Centered	Centered	Diagonally Forward and Right	↗	
Centered	Up	Centered	Centered	Diagonally Forward and Left	↖	
Centered	Down	Centered	Centered	Diagonally Bacward and Right	↘	
Down	Centered	Centered	Centered	Diagonally Backward and Left	↙	
Centered	Centered	Far from Face	Far from Face	Up / Take Off	↑	
Centered	Centered	Close to Face	Close to Face	Down / Land	↓	

Table 13 [Gestures Commands]

The script iterates all the detected objects and if they happen to be a face or a hand, it gets its information. Specifically, it gets the coordinates of the center of one face and two hands detected, in fact it will not send any command to the drone unless two hands and one face are detected. So no more than two hands and one face should be in front of the camera. These are the names of the variables:

Coordinates Values	X coordinate	Y coordinate
Face	f_x	f_y
Right Hand	rh_x	rh_y
Left Hand	lh_x	lh_y

Table 14 [Face and Hands Variables Names]

Then it compares the face's Y coordinate with the hands' Y coordinates, and gets its difference, so we end up with two values: rh_dy (f_y - rh_y) and lh_dy (f_y - lh_y), which are the difference in Y values between the face and the right hand and the difference in Y values between the face and the left hand respectively. As the top's image is Y=0 and the bottoms' image is Y=480, the maximum value that rh_dy and lh_dy variables may take, supposing a centered face, is -240 (hand down) or 240 (hand up).

rh_dy = f_y - rh_y

lh_dy = f_y - lh_y

We want the drone to move if such rh_dy and lh_dy values are greater than 40 (in absolute terms, less than -40 would mean movement as well). So there is a "safe" zone between -40 and 40 in which the drone will not move. Also we will consider that a difference in Y of 200 is the maximum, so if the difference is greater than 200 it will be the same as if it was 200. Basically what we want to accomplish is that if both hands have a face Y difference under 40, the drone will not move, and if it is 200 or greater the drone will move at its full velocity. Therefore between 40 and 200 we need all the different steps between not moving and moving at full speed.

Note that when we use the "dy" notation we are referring both to "rh_dy" and "lh_dy". In the same way that when we use "dy_norm" we refer both to "rh_dy_norm" and "lh_dy_norm".

For this last objective we will normalize the rh_dy and lh_dy values, obtaining rh_dy_norm and lh_dy_norm respectively in this way:

Absolute rh_dy and lh_dy values	Absolute rh_dy_norm and lh_dy_norm values
dy=<40	dy_norm=0
40<dy<200	0<dy_norm<1
200=<dy	dy_norm=1

Table 15 [Gestures Application Y Normalization Values]

The normalization between 40 and 200 values is done in this way for positive values:

dy_norm = (dy - 40)/160

And in this way for negative values:

dy_norm = (dy + 40)/160

This gives us a linear transformation from the [40,200] domain to the [0,1] domain and from the [-200,-40] domain to the [-1,0] domain.

We will use this Y normalized values later on, but first we need to get the X ones. All the Y values will be used for horizontal movement only, whereas the X ones will be used only for vertical movement. As the different positions we may manage to do with our face and hands are not endless and we want to make the drone easily controllable, this script will only allow you to move the drone vertically when it is not moving horizontally. In other words, the script will only calculate the normalized values of the X coordinate difference between the face and hands when both rh_dy_norm and rh_dy_norm values are 0. Because if the drone is moving horizontally (that's when the Y normalized values are not 0) it will not be able to move vertically, and as long as the X normalized values only are used for vertical movement, it makes no sense to calculate them when no vertical movement is allowed.

So in case dy_norm are 0, we will calculate rh_dx, lh_dx, rh_dx_norm and lh_dx_norm in a very similar way.

First of all, the rh_dx and lh_dx values are calculated like this:

rh_dx = abs(f_x - rh_x)

lh_dx = abs(f_x - lh_x)

Note that in this case both operations are absolute, as our target now is that the drone moves up when both hands are away from the face horizontally and it moves down when both hands are close to the face. No matter if the subtraction gives us a negative value.

Then the values are normalized.

rh_dx and lh_dx values	rh_dx_norm and lh_dx_norm values
dx<=125	dx_norm=-1
125<dx<175	-1<dx_norm<0
175<=dx<=225	dx_norm=0
225<dx<275	0<dx_norm<1
275<=dx	dx_norm=1

Table 15 [Gestures Application X Normalization Values]

Between 175 and 225 there is a "safe zone" in which the drone will not move, so the user has to clearly move the hands away from or closer to his face in order to make the drone move.

The normalization between 125 and 175 values is done in this way:

dx_norm = (dx - 175)/100

And between 225 and 275 values is done in this way:

dx_norm = (dx - 225)/50

This gives us a linear transformation from the [125,175] domain to the [-1,0] domain and from the [225,275] domain to the [0,1] domain.

The script also keeps track of two possible drone states: “flying” and “landed”.

If the current state is “landed”, the “take off” command will be sent if both rh_dy_norm and lh_dy_norm are equal to 1 (and then the drone state will switch to “flying”):

```
rostopic pub --once bebop/takeoff std_msgs/Empty
```

If the drone state is “flying”, it will send to the drone the movement command. In case of horizontal movement:

```
rostopic pub -r 10 /bebop/cmd_vel geometry_msgs/Twist '{linear: {x: A, y
: B, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}'
```

Where A = 0.5*rh_dy_norm + 0.5*lh_dy_norm

Where B = rh_dy_norm - lh_dy_norm

These two calculations are used to achieve the horizontal movement explained before.

In case of vertical movement:

```
rostopic pub -r 10 /bebop/cmd_vel geometry_msgs/Twist '{linear: {x: 0.0,
 y: 0.0, z: C}, angular: {x: 0.0, y: 0.0, z: 0.0}}'
```

Where C = (rh_dx_norm + rh_dx_norm) / 2

It uses the average position of both hands as a unique input, because both hands are mirrored used (both close to the face or both far from the face). In fact, if one is close to the face and the other one is far from the face, the drone will not do anything, as the information is contradictory. They would even cancel out with the average.

Finally, if we keep moving the drone down, it will eventually land automatically.

3.5.2. Person Tracking

The main objective of this script is to give the drone the ability to track a person at a constant height (the drone must be already flying when the script is executed and it will keep its height at the moment of such execution).

Basically there are two parts: the centering one, in which the drone turns on its own axis to keep the person centered, and the moving one, in which the drone moves forward or backward to keep the person at a constant distance. Also there is a constant checking of all the people appearing in front of the drone to make sure it is tracking the same one all the time even though there are more than one suitable candidate to track.

Let's explain first of all this checking process, as no tracking is considered good if the drone randomly changes the person who tracks every now and then. When the script is executed, the drone must be flying already (as the take off and land commands must be sent manually). So when the script starts the drone does not know who is the good candidate to follow. What does the script is to choose the most centered person in the image to be such candidate, as normally when we initialize this script, the person to track is already in front of the drone. To achieve this, the algorithm iterates every person detected and checks its X coordinate position. After comparing them all it will get as the candidate to follow the closest to the 340 X position, as the image is 680 wide.

Once it has decided who is the good candidate, it will compare every person detected on each frame again, but instead of looking for the one closest to the 340 X position, it will look for the one closest to the X position where the tracked person was in the previous frame. Obviously if the tracked person moves away from the drone's sight, it will track a new person. And if there is more than one new person, it will follow the one who is closest to the last X position where the previous tracked person was.

Now that we know how the drone knows who to keep track of, let's see how it moves.

The drone will only perform two movements: turn around its own axis and move forward and backward. With these two movements it achieves the centering and moving parts we mentioned before.

When the person is not centered in the image, the drone gets the X position of the person (specifically the X position of its center) and compares it with the target X position which is the center of the screen (340). Such difference value is called "dif_x":

dif_x = target_x - p_x

Then it normalizes this value:

dif_x_norm = dif_x/target_x

And divides it by 3 in order to make the drone turn slowly:

drone_twist = dif_x_norm/3

Finally, this command is sent to the drone:

```
rostopic pub -r 10 /bebop/cmd_vel geometry_msgs/Twist '{linear: {x: 0.0,
 y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: drone_twist}}'
```

Such process will be repeated until the person is centered, then it will move forward or backward to keep the person at the same distance. To do so, the script measures how wide is the detected box of the person (such width is called "p_width"):

p_width = box.xmax-box.xmin

Then compares it with the desired width, that is target_width = 180, which is the width of a detected person about 1 meter far away:

dif_width = target_width - p_width

Finally, we will normalize the dif_width value. If it has a positive value, which means that the target width is bigger than the width of the person so the drone needs to move forward in order to see the person width bigger, it will be normalized in this way:

dif_width_norm = dif_width/165

The 165 value is because it has been tested that the smallest width that the system can detect while successfully detecting a person is 15, and the difference between 180 (the target width) and 15 is 165. So it will always find a dif_width between 0 and 165. This gives us a linear transformation from the [0,165] domain to the [0,1] domain.

If dif_width has a negative value instead, which means that the target width is smaller than the width of the person so the drone needs to move backward in order to see the person width smaller, it will be normalized in this way:

dif_width_norm = dif_width/250

In the same way it was checked that the smallest width detected was 15, the biggest one is 430. As the difference between the target width (180) and 430 is 250, we use this value to normalize it. This gives us a linear transformation from the [0,180] domain to the [0,1] domain. As the subtraction of “dif_width = target_width - p_width” already gives us a negative value in this case, the normalized value turns out to be [-1,0].

Finally, the normalized value is divided by 3 so the drone does not move too fast:

drone_move = dif_width_norm/3;

With the drone_move value, it will send the following movement command to the drone:

```
rostopic pub -r 10 /bebop/cmd_vel geometry_msgs/Twist '{linear: {x: dron
e_move, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}'
```

Here is important to know that if the person opens his arms wide, the detected box of it will increase in width. This is useful because if the user wants the drone to move away quickly or just to stop while it is coming towards him, he just needs to open the arms as the drone will detect it as the user was closer to the drone.

Whenever the person is not centered anymore the moving routine will stop and it will start rotating again.

4. Results

Both applications have turned out to be a success, they both make the drone behave in the desired way and they both accomplish the requirements.

4.1. Working Workspace

The success of this project mainly resides in the combination of many parts in a same workspace, not only in finding and using the correct ones but also to make them compatible with each other. In fact, it is where more problems have appeared during the development. However, once all the compatible versions have been spotted and installed, and the procedure for each of them has been done following the same protocol, everything turned out to work alright.

Nevertheless, there has been one drawback. When installing all the packages sometimes some of their parameters depend on the computer's architecture or version, so migrating entire workspaces from one computer to another, in some cases, has turned out to be a bit tedious. In a way in which sometimes it is faster to reinstall the packages from the new computer.

However, some key files like the Python scripts can be easily moved from one computer to another (as long as both have a properly working workspace). Also migrating and copying workspaces in a same computer brings no troubles.

4.1.1. Working Environment Example

This would be the working environment while developing the applications. There are many windows, each one for a process, which are: the roscore, the USB camera terminal process, the USB camera video stream, the image recognition terminal process, the image recognition video stream, the firmware of the simulator, the simulator terminal process, the simulator window itself, a terminal to send direct commands to the virtual drone, a file explorer, a txt file with commands to copy and paste them quickly to the terminals, a gedit text editor to develop the python script and a terminal for the script execution.

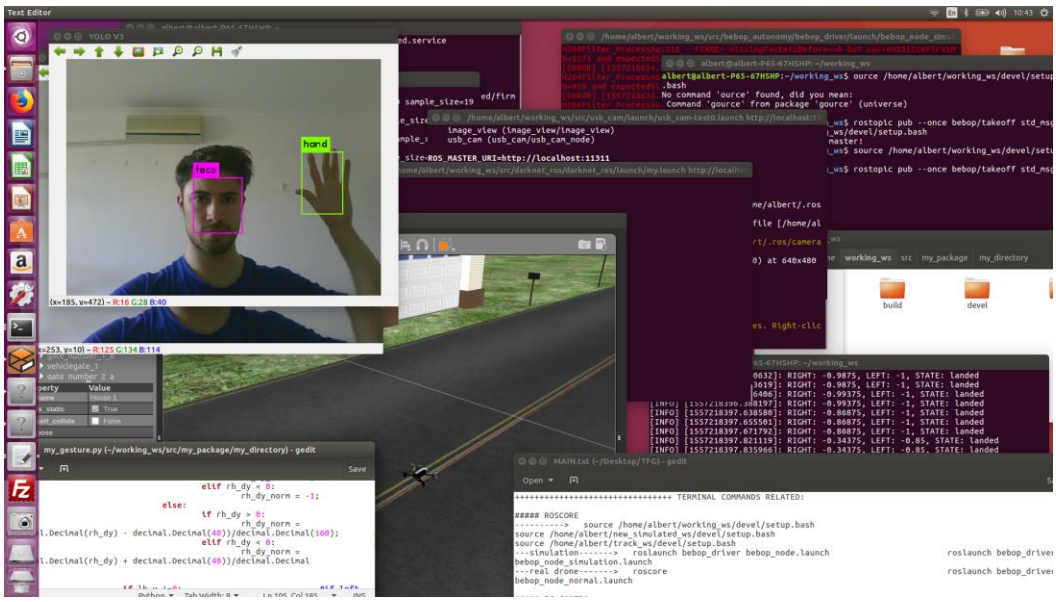


Figure 11 [Digital Working Environment]

4.2. Gesture Controls Application

This has turned out to be the most rewarding application, as it is not only very funny to use but very precise as well. It is not trivial though to control the drone with gestures, is natural enough but it needs an open space when used for the first time to practice securely.

The fact that the drone cannot move vertically and horizontally at the same time has been a human decision, not a result of a lack of performance or of good enough scripting. It has been tested indeed, and it works perfectly, just combining the commented gestures you can achieve vertical and horizontal movements at the same time. However, the controlling happens to be a bit too difficult for a standard user that is not familiar with this technology, so unless proper training is done, it has been decided that it should not be an "out of the box" option.

Another intentional limitation is the inability to make the drone turn on its own axis, as it would be far too many possible inputs for the user to take into account, unless combining it with other possible input controls as some kind of button hold by the hands. But as the main aim of this application was to fully control the drone only with gestures, that has been discarded.

One important drawback is that the face must be very centered. The script makes the decisions based on the relative position of the face and hands, so the gestures still work alright with a not centered face. However, if the face is not centered, it means that there will be more distance between the face and one screen's edge than between the face and the opposite screen's edge. This will leave the user with less distance to move the hands in one direction and this will turn out in less maneuverability, as he will not be able to achieve the maximum distance between the face and one of the hands. Nevertheless, as generally the used camera is from a laptop and normally the user is not in a fixed position, it is as easy as to move the setup a bit to leave everything centered.

What has turned out to be very useful and well performing, is the fact that the drone will hover if no two hands and one face is detected. So if the user wants to change the place with another user, or just does not know how to control it and moves away, the drone will immediately stop and hover wherever it is.

One final good aspect of this application, is to combine the computer's camera with the drone's camera. The first one is to control the drone as already commented, but the second one is to see what the drone sees in real time, as it has been seen that is far more natural to control the drone just looking at its perspective rather than looking at it from wherever the user may be.

4.2.1. Face and Hands Detection Example

This is how faces and hands are detected in this application.

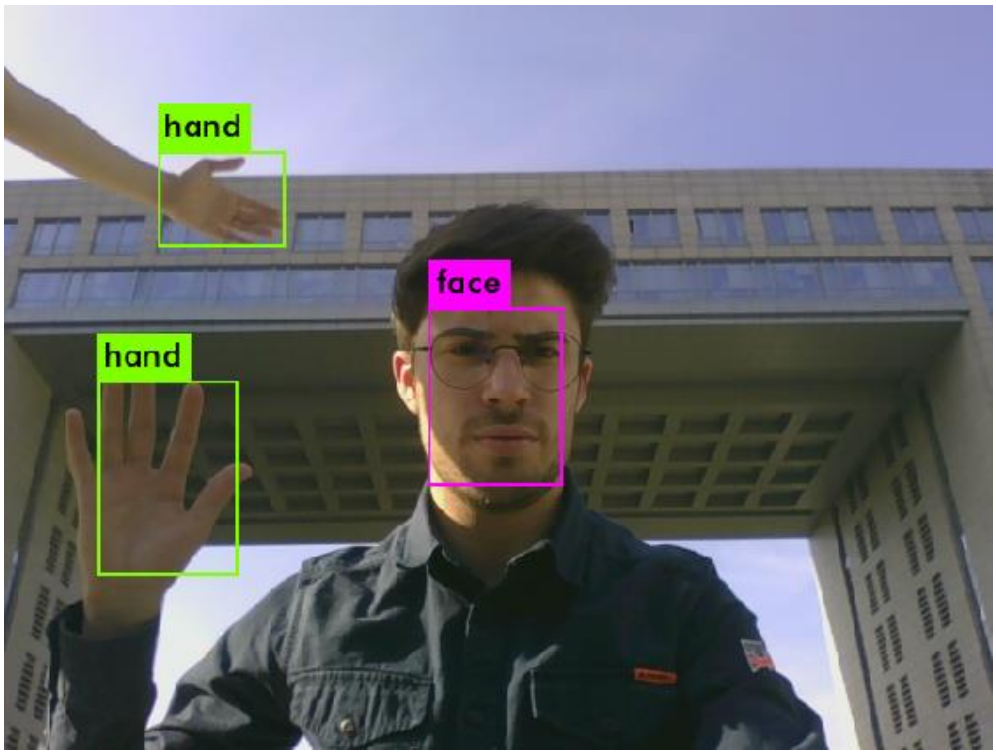


Figure 12 [Face and Hands Detection]

4.2.2. Actual Gesture Controls Application Example

A group of images will be presented about the performing of this application, they are snippets of an actual video. The drone has been marked with a red circle for its easy visual location.

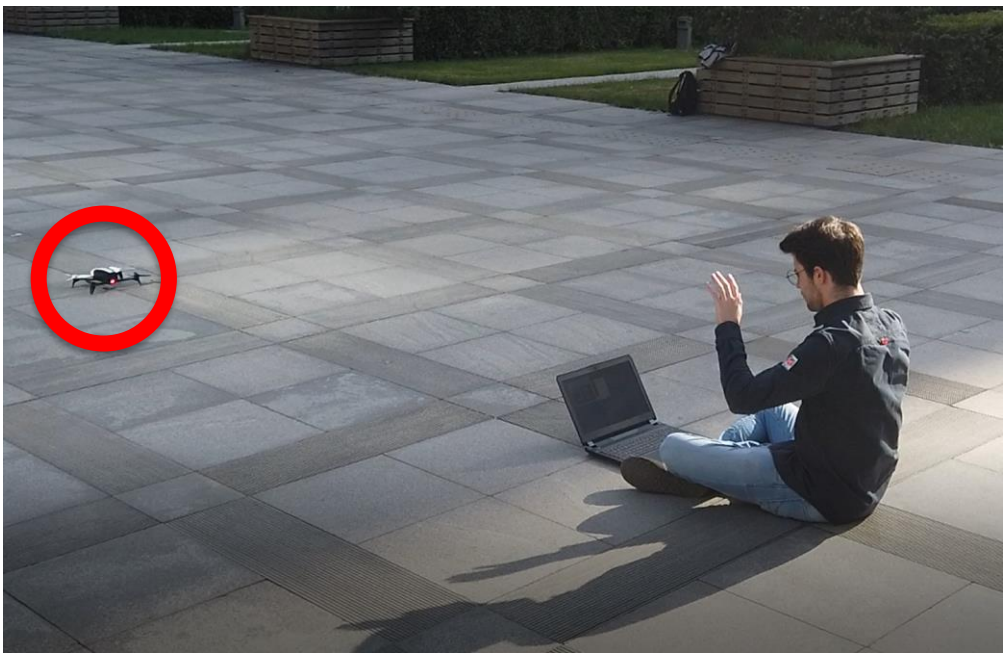


Figure 13 [Taking Off 1]

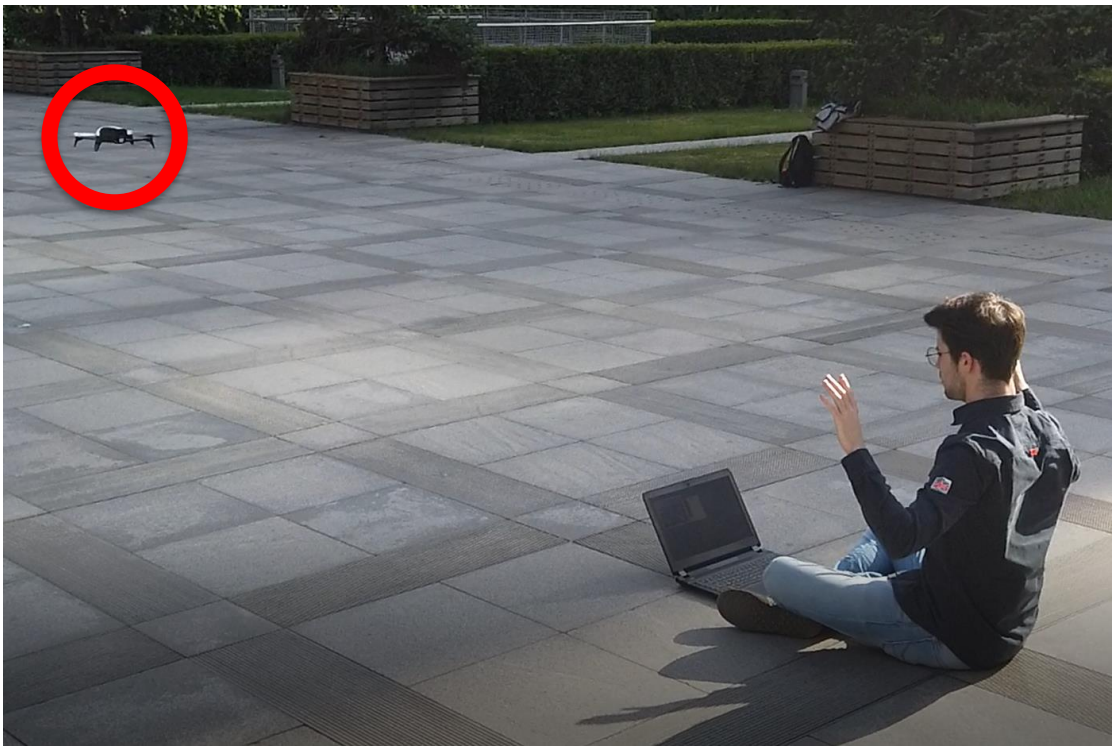


Figure 14 [Taking Off 2]

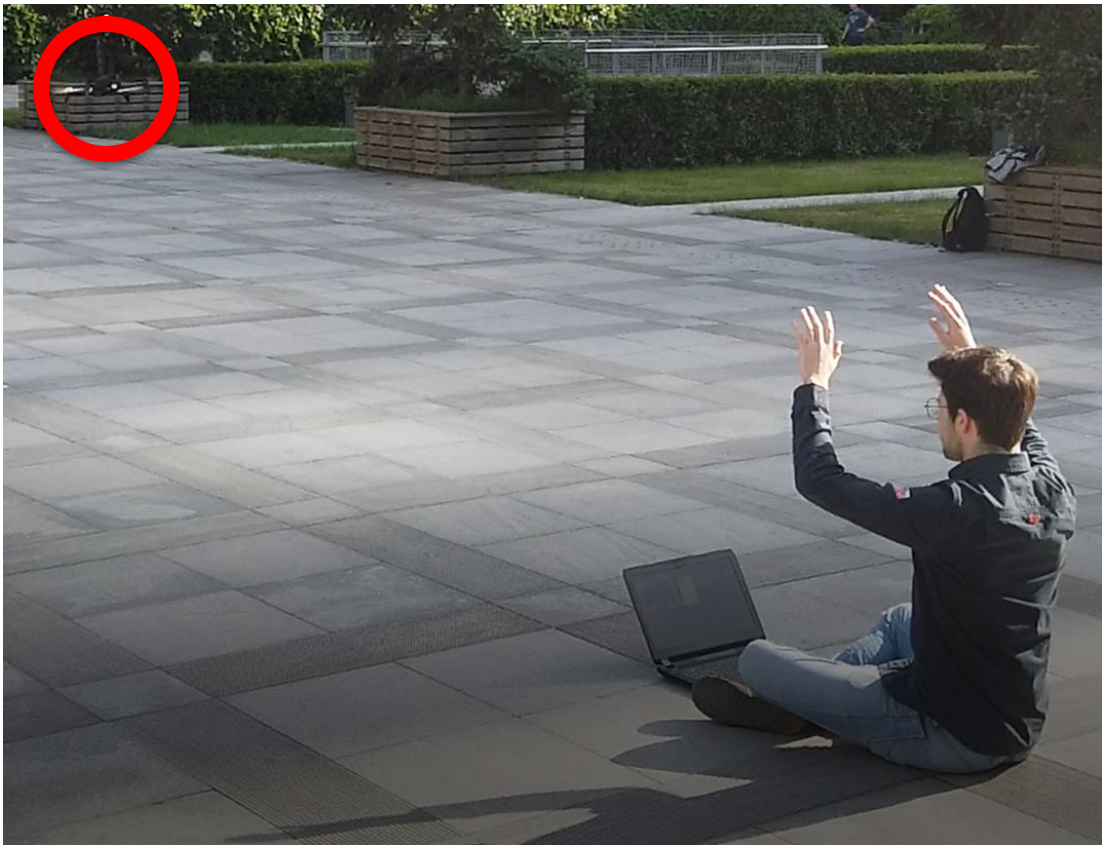


Figure 15 [Moving Forward]

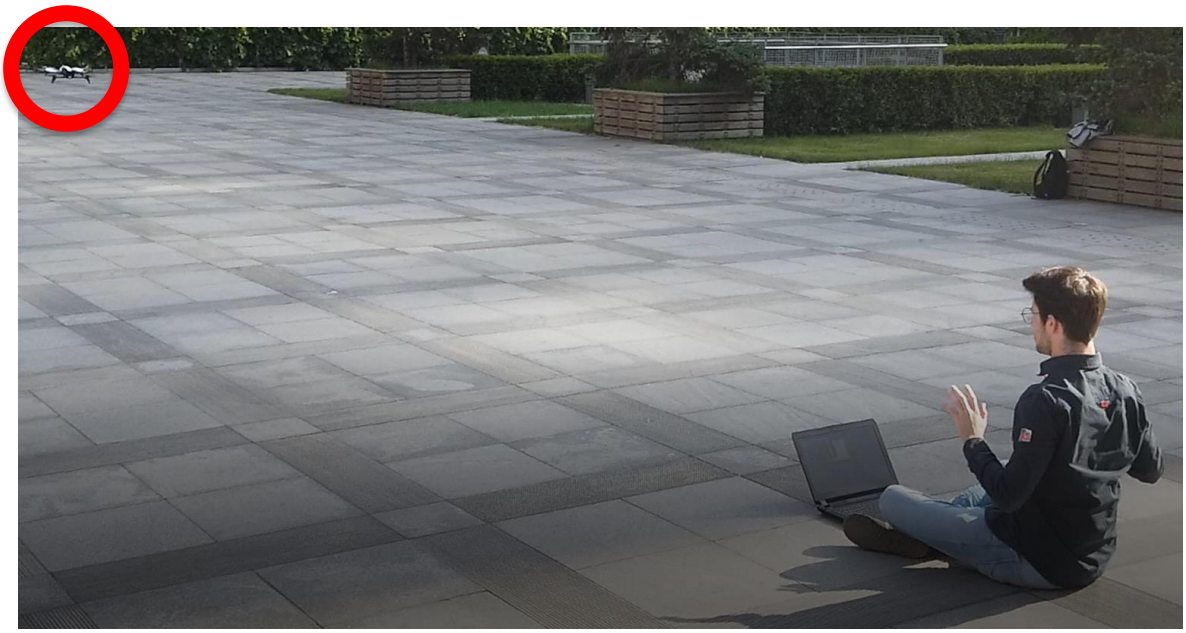


Figure 16 [Moving Backward]

Figure 17 [Moving Left]

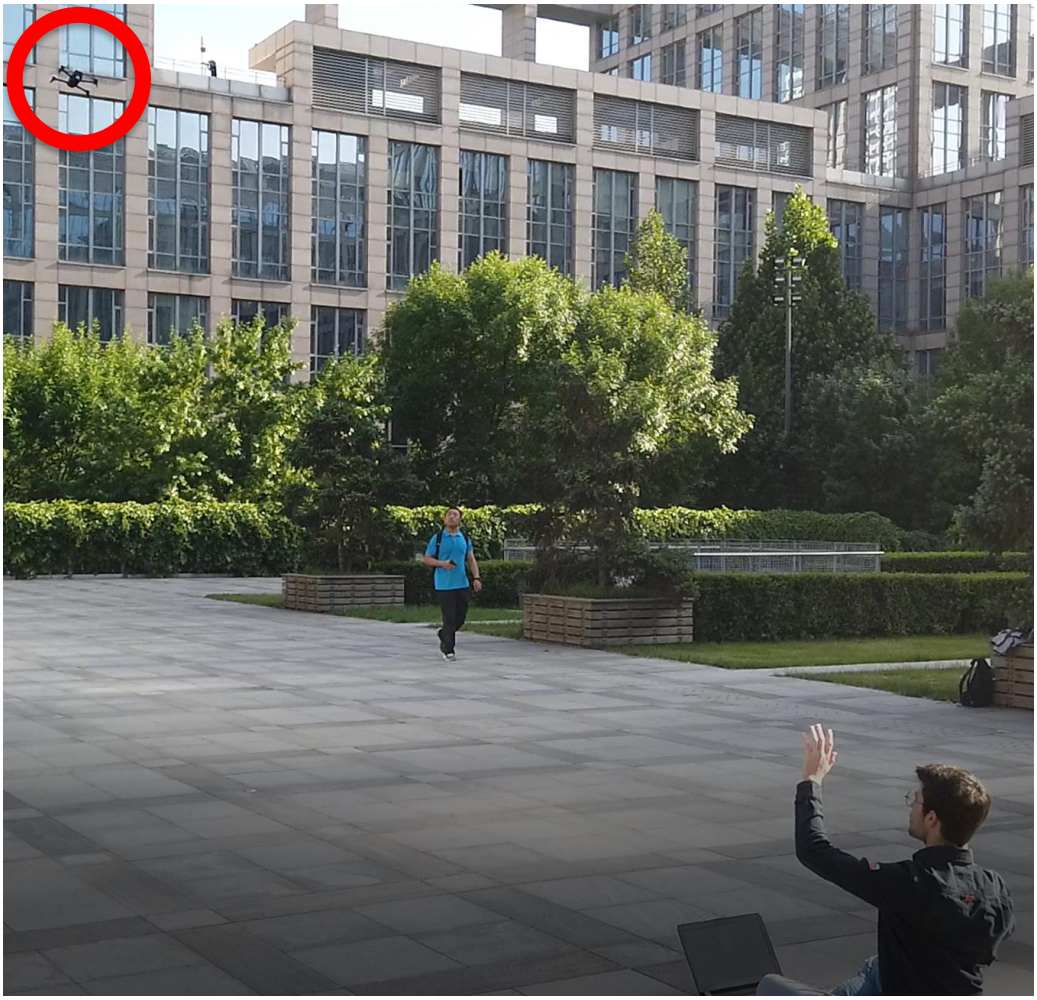


Figure 18 [Moving Right]

Figure 19 [Landing]

4.3. Person Tracking Application

This application was actually the main target of the whole project, and even though it felt too ambitious at the beginning, it finally came out alright. It does what it is intended to do: follow a person autonomously without losing track of it when more people appear in the image.

However, there are some drawbacks. Sometimes the drone feels a bit too slow and the target must move not too fast if it is too close to the drone to avoid leaving the drone's perspective. The reason for this are two combined aspects: the first one is that in order to keep the drone's attitude simple, its movement has been made very modular (one kind of movement at a time without combining them), and the second one is security itself, it has been decided that it is better to have a slow easily predictable drone rather than a drone that moves too fast and may lead to unexpected behaviors. It does not mean that it is impossible to make a faster and evenly secure drone, but for the scope and length of this project the already achieved behavior has been considered successful.

Another drawback is that the algorithm assumes an already flying drone, so you need to make it take off and land manually. In the same way the drone also assumes an already given height that it will maintain during all the tracking process. So the desired height must be manually adjusted before launching the tracking algorithm as well. To solve this problem there would be only two solutions: dispose of a remote controller to only make it take off and land (which was discarded as the main purpose of the whole project is to achieve an autonomous drone and it would only replace the already functional computer's controller, so it would leave us with the same situation), or to combine the gestures application with this one to be able to make it take off and land with gestures. However, this application was meant to achieve an autonomous tracking no matter the previous setup, so in case of combining such two applications would mean to develop a third one, which already fits in potential future development, not in the objectives of this project.

The last drawback is the lack of a PID (a calibration for the drone's movement intensity and velocity). Because if we made the drone turn faster than it does now to keep the person centered in the image faster, it would turn so fast that by the time it should stop, it would not stop in time. So it would get in a loop in which it would move from side to side always trying to keep the person centered. A quick solution has been to make it turn slower so this doesn't happen, but a proper better approach would be to slow down the turning as the person gets more centered. This leads us to the most important problematic scenario, as this matter also appears in the translation of the drone and not only in its turning: if the user happens to move way faster than intended and it runs in a straight line far from the drone, the drone will start moving towards it gaining more and more velocity, so by the time it should stop it will have no time to stop because of the gained momentum and this would end up in the drone moving past away from the tracked person (and loosing track of it) or, if flying at a low height, even crashing with it. This has been qualified as a bad usage by the user, as at no moment the tracking was intended to follow a fast moving target. However, it is not a trivial problem to solve, as the drone has no idea if the person will keep running or will stop all of a sudden, so this kind of adaptation would bring us to a completely new kind of application that clearly surpasses the initial scope of the project.

Despite the aforementioned problems, the most remarkable fact of this application, in case of a good usage, is its reliability. It has been tested in scenarios where more than 20

people were behind the tracked one at different distances and the drone did not lose track of the desired person at any moment. Also the responsiveness of opening the arms to make the drone stop or move away, has turned out to be very quick, which brings an extra feeling of security to the user. Even when running fast and in a straight line from the drone, which is the most problematic scenario that we commented before, simply by opening the arms while the drone is approaching to the user, solves the problem immediately as it stops immediately. It even feels like controlling the drone with the arms, which is what it is indeed.

4.3.1. Person Detection Example

These are examples of the person detection from the drone's perspective.

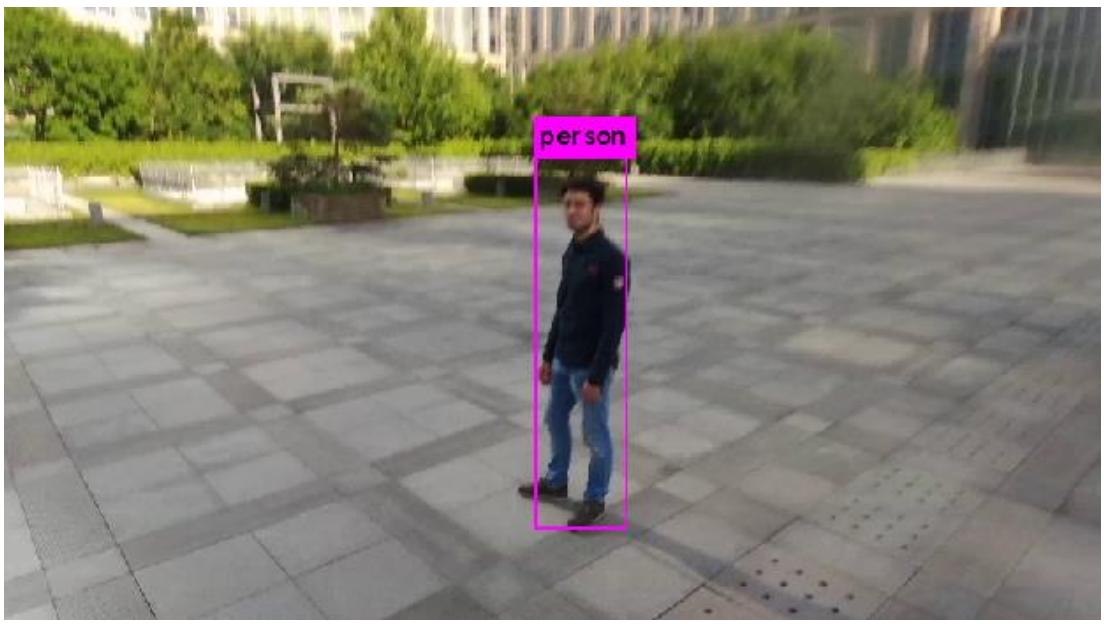


Figure 20 [Close Person Detection]

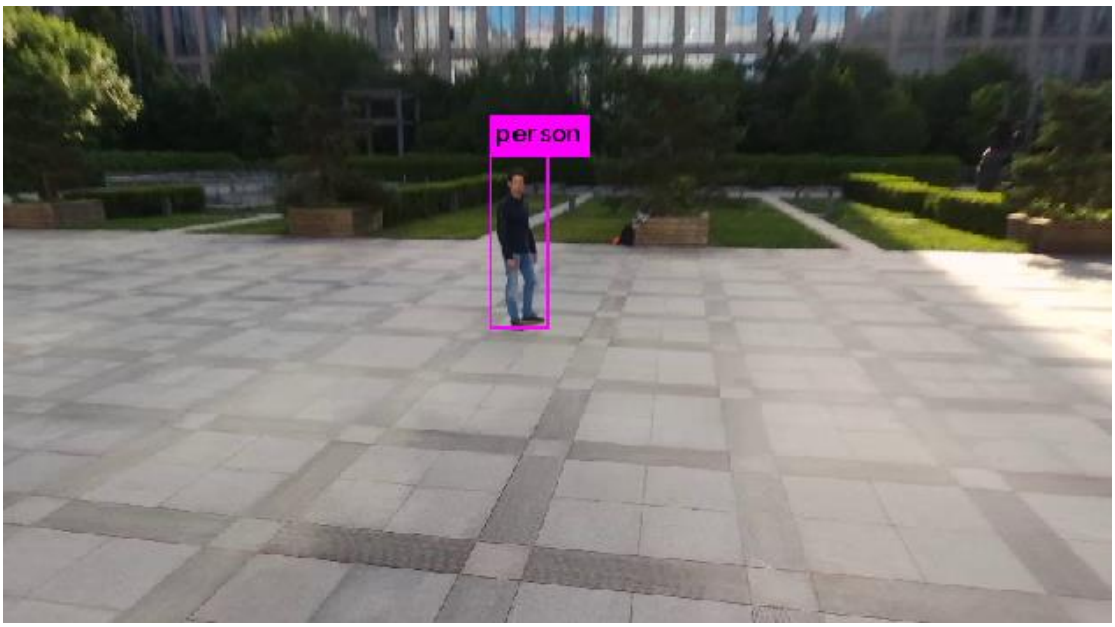


Figure 21 [Mid Distance Person Detection]

Figure 22 [Mid Far Person Detection]

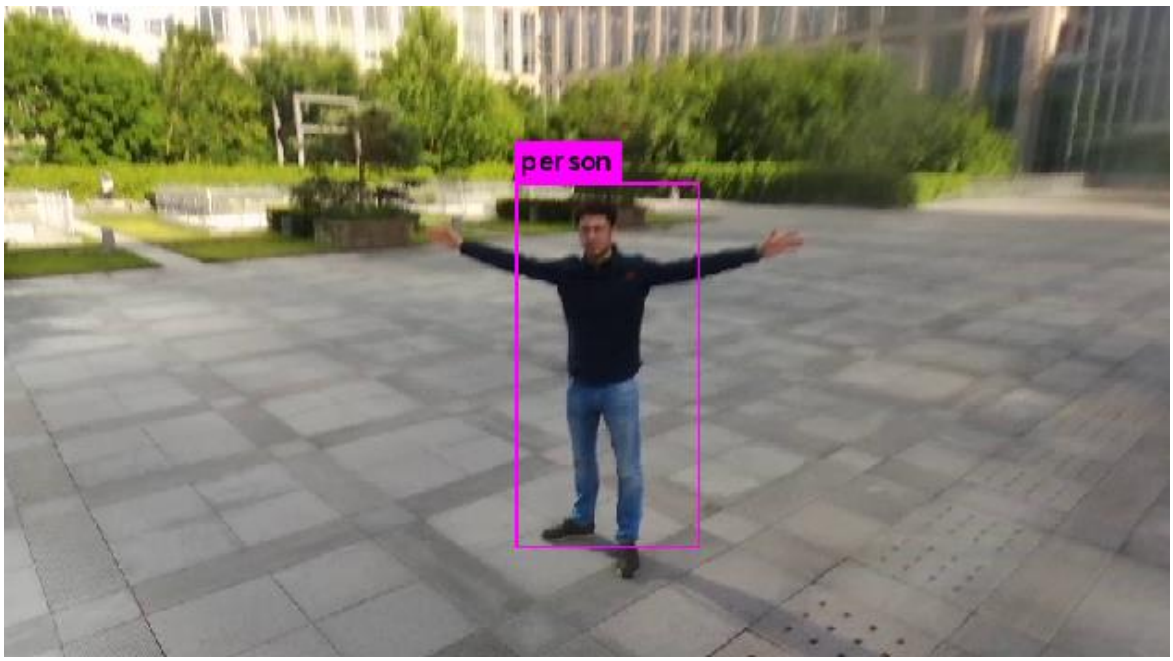


Figure 23 [Open Arms Person Detection]

4.3.2. Actual Person Tracking Application Example

These are snippets of a video of this application performance. Again, the drone has been marked with a red circle for its easy visual location.

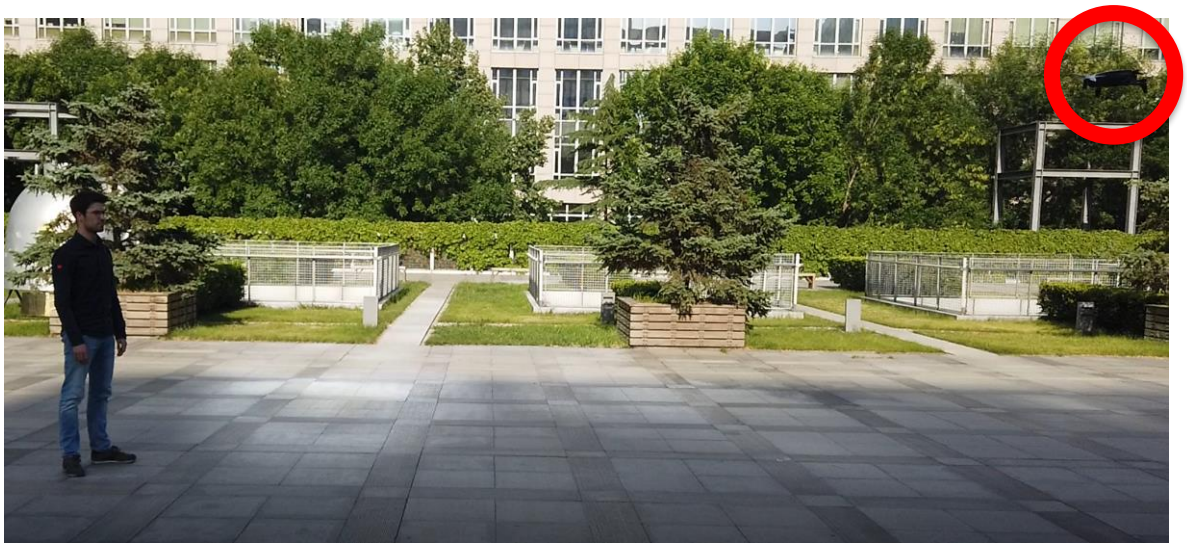


Figure 24 [Drone Centering Person]

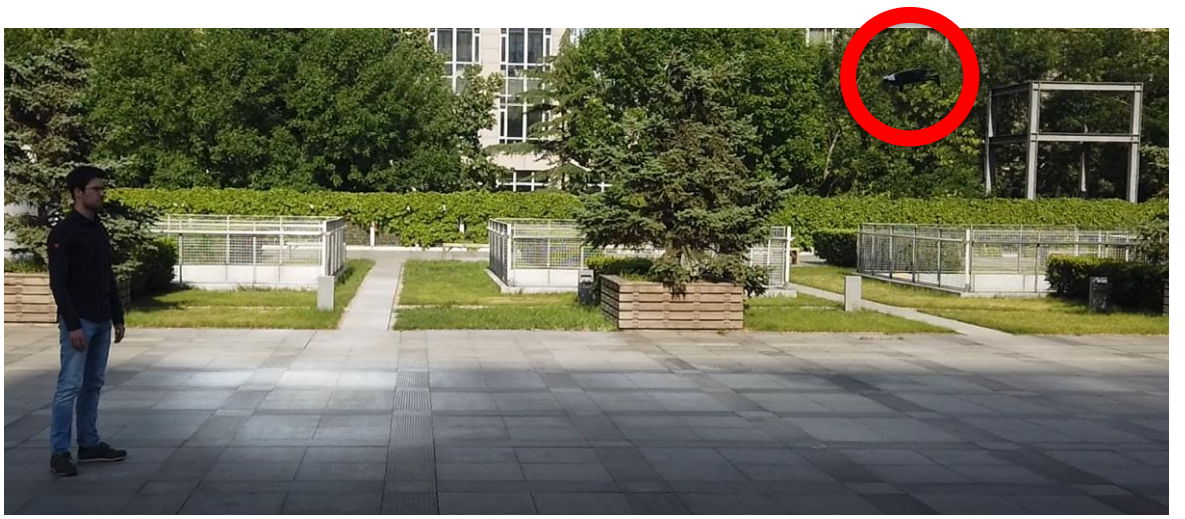


Figure 25 [Drone Moving Towards Person]

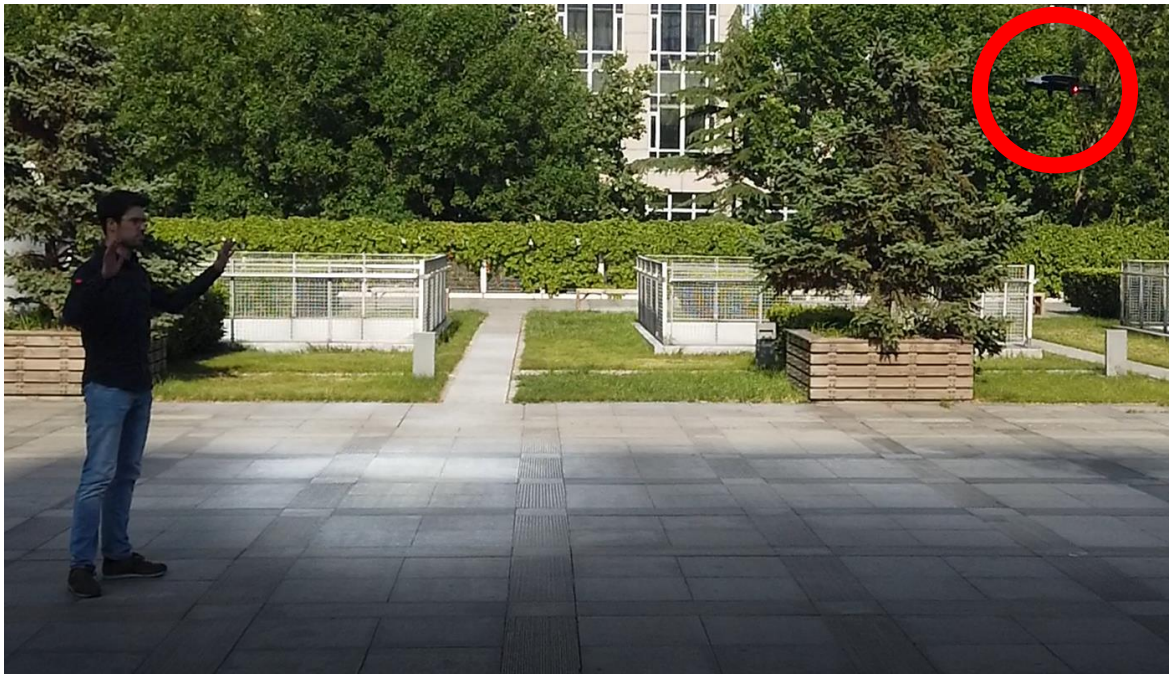


Figure 26 [Drone Moving Away because of Open Arms]

Figure 27 [Centering Person 1]

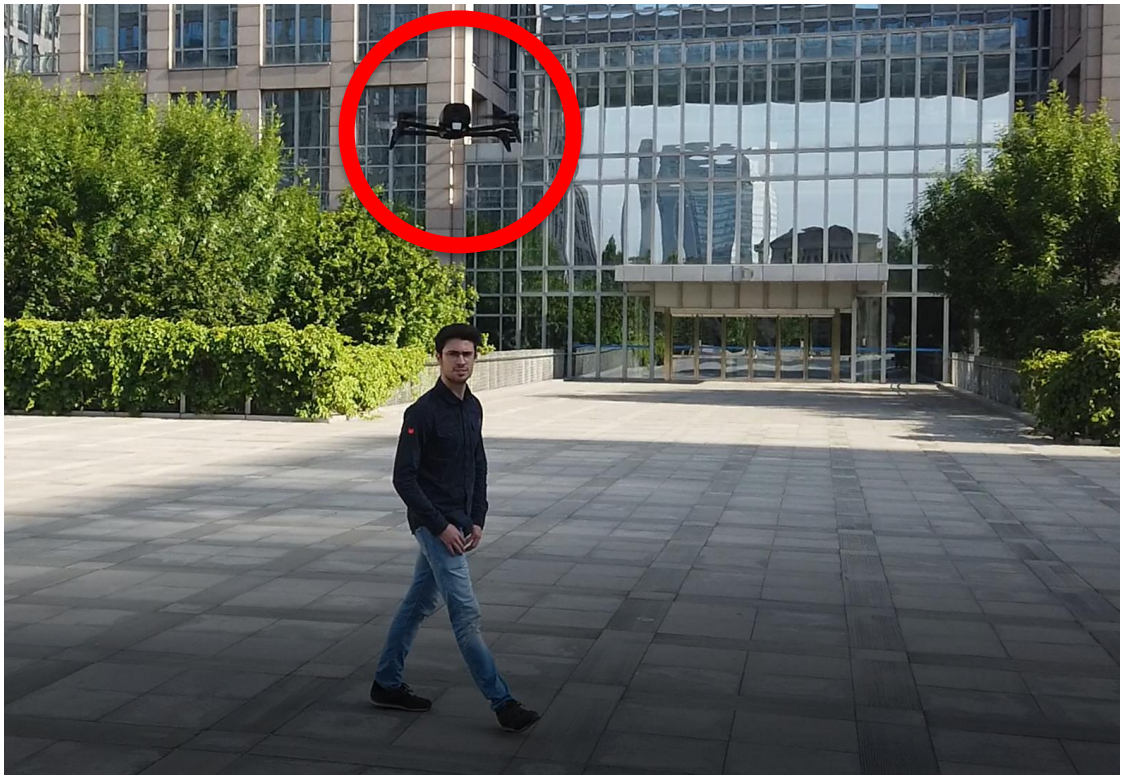


Figure 28 [Centering Person 2]

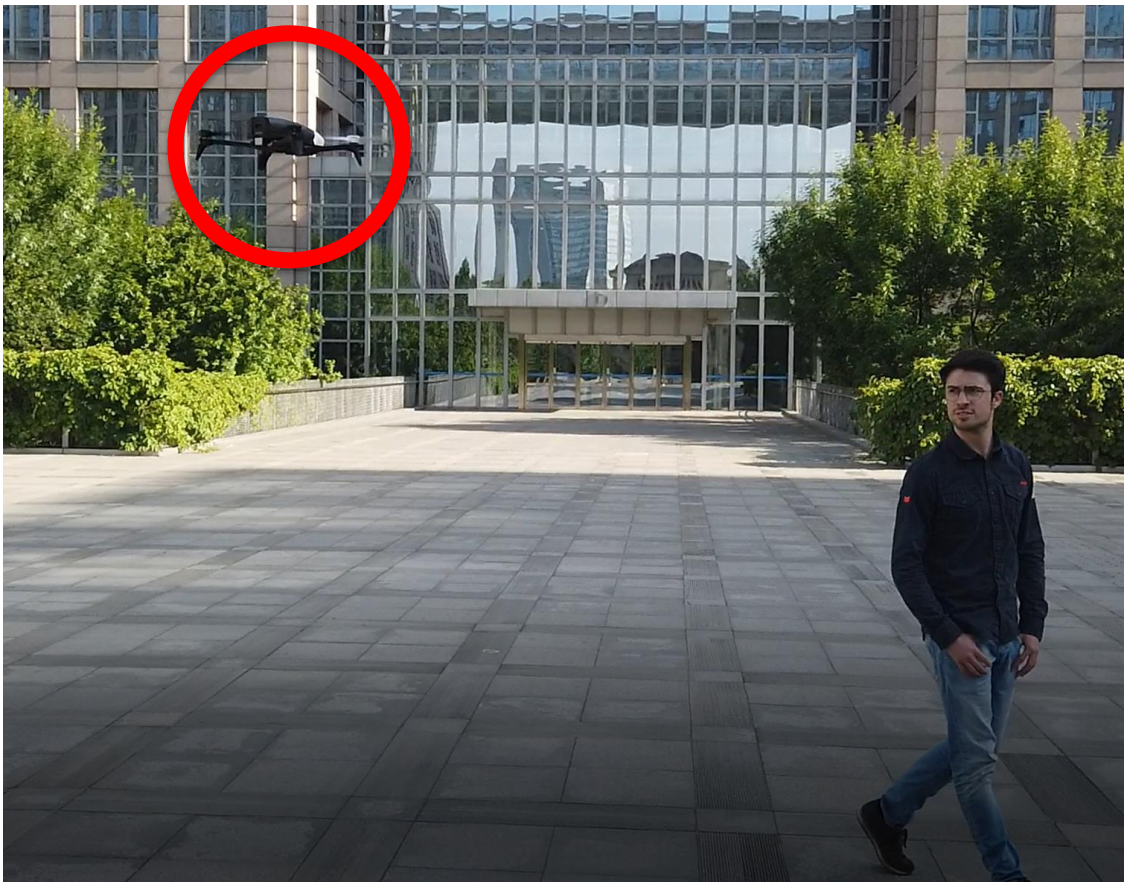


Figure 29 [Centering Person 3]

Now, images taken directly on the computer's screen (hence their low resolution and bad centering) will be presented to show the drone's perspective while following such person.

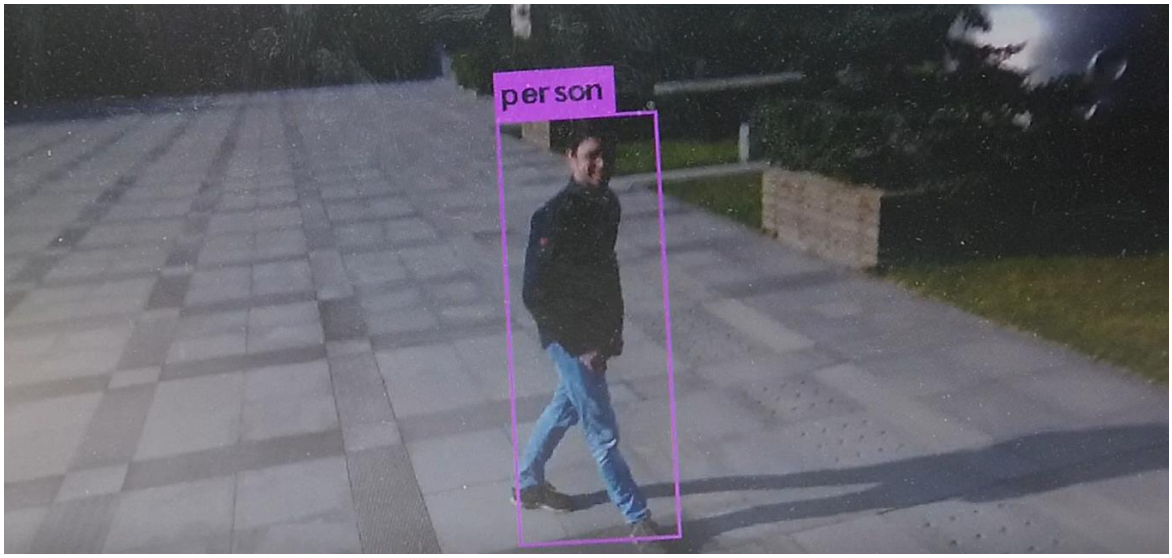


Figure 30 [Person Tracked Alone]

With the following 3 images we can see that the drone does not lose track of the followed person even if other people are by the sides of the image:

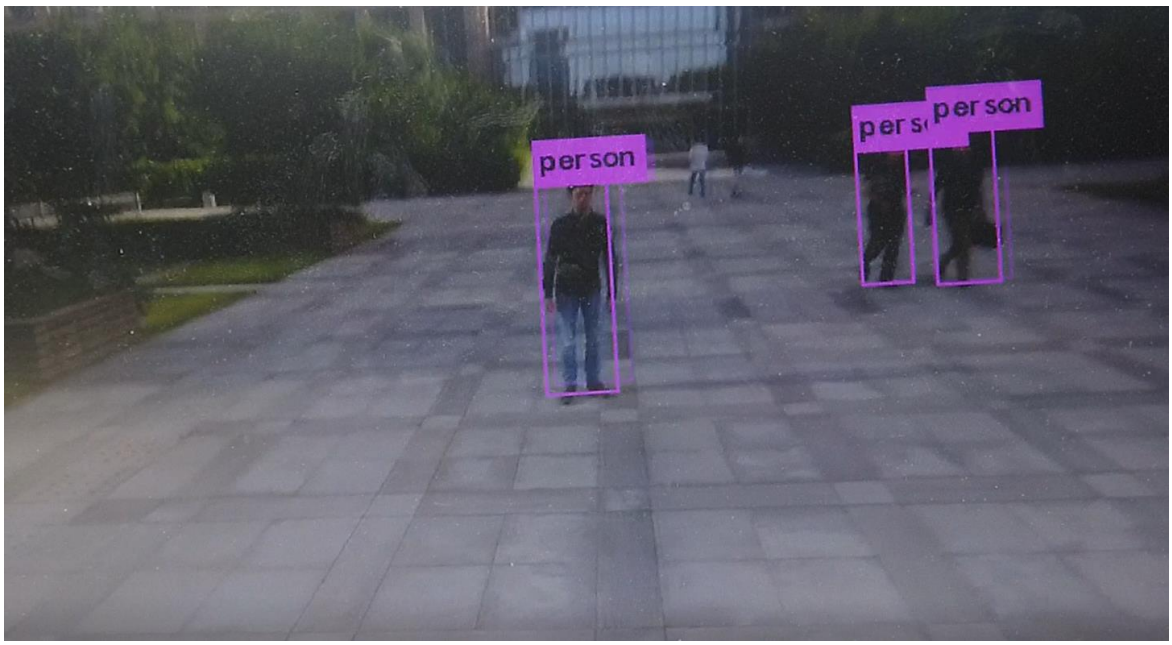


Figure 31 [Other People on a Side while Tracking 1]

Figure 32 [Other People on a Side while Tracking 2]

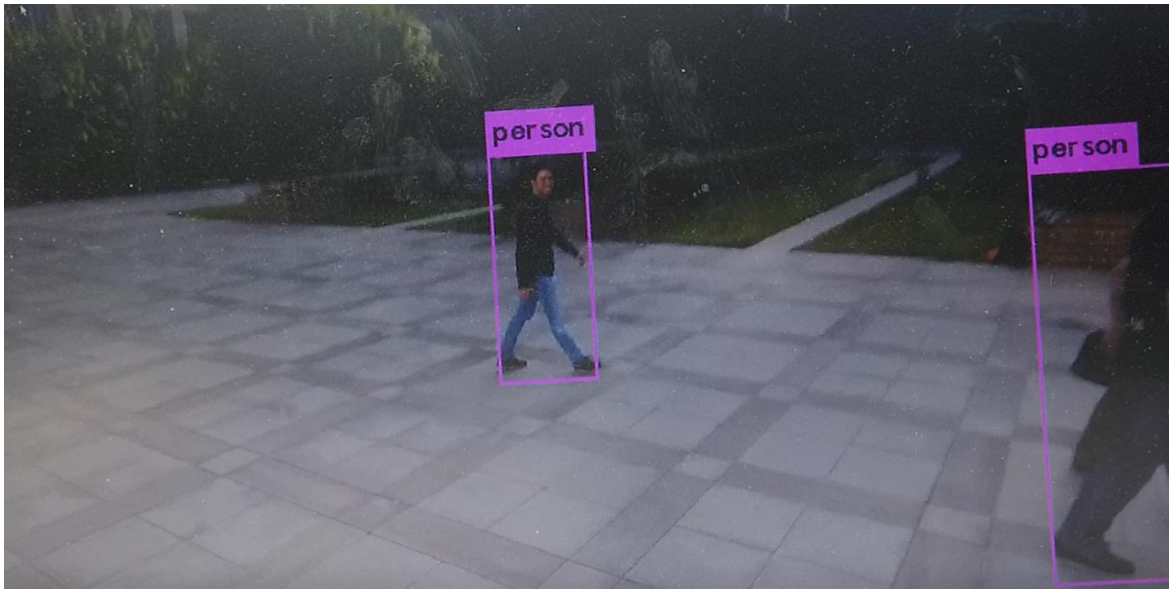


Figure 33 [Other People on a Side while Tracking 3]

With the following 3 images we can see that the drone does not lose track of the followed person even if other people are behind of the image:

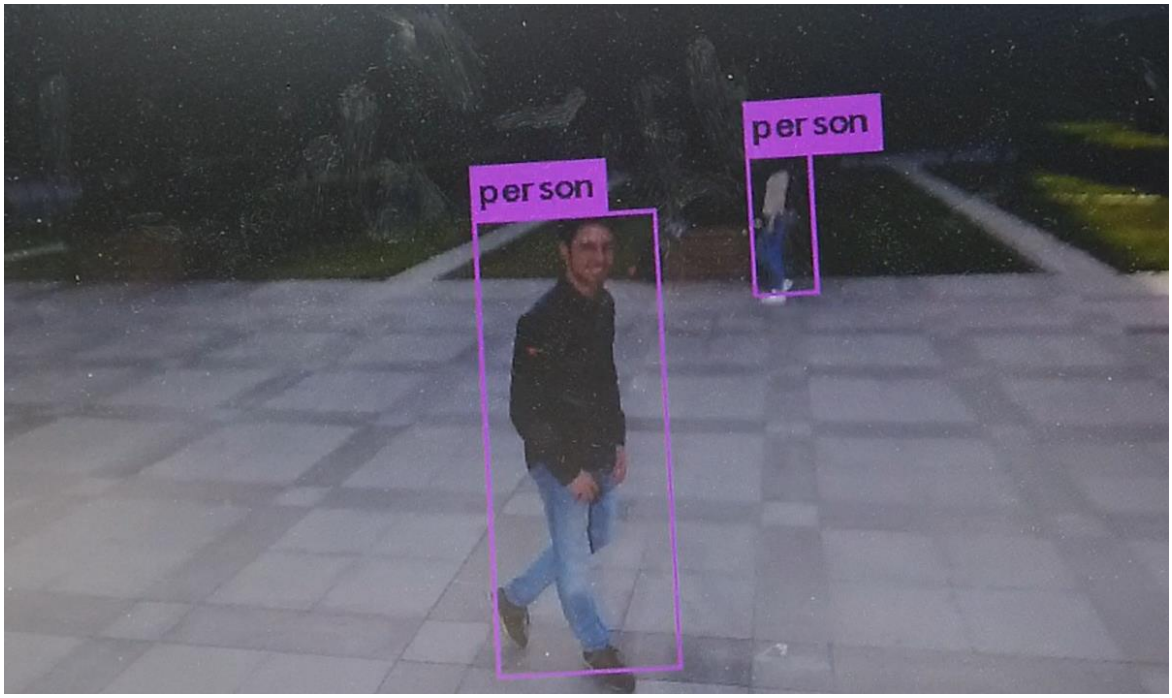


Figure 34 [Other People Behind while Tracking 1]

Figure 35 [Other People Behind while Tracking 2]

Figure 36 [Other People Behind while Tracking 3]

5. Budget

5.1. Product Aproximate Cost

As we have already seen, this project relies on open source based resources and no license required software has been used. The only non-zero cost elements are the drone itself, which is around 500$ depending on the country of purchase, and a powerful enough computer that can be easily found for 1000$. In this case such drone was provided by the university and the computer was already owned, but as a whole product approximate cost would only be the aforementioned 1500$. In fact, has been a major key element in the whole project development to keep such cost as it is.

5.2. Design Aproximate Cost

Supposing a junior engineer cost of 14€ per hour, and a total number of 450 hours invested in the project development. That is a design approximate cost of:

```
450h * 14 €/h = 6300€
```

5.3. Economic Viability

Even though this project relies on open source based resources it does not mean that such resources are free to use in a commercial way. In fact, there is no clear policy about this theme in many of them, they just declare that a partnership must be accorded to use their software commercially. So it highly depends on the final application and the potential negotiation with such entities. However, this does not make the project non-marketable, it simply has no direct way to calculate a real possible application cost without actually contacting the open source assets providers.

But surpassing the distance of a non-clear market budget, it clearly is a viable product. In fact, there are multiple emerging companies that are filling these new market possibilities that rely on this new kind of technologies. Not only many jobs that were used to be carried out by humans have turned out to be way cheaper and more accurate to be carried out by drones and artificial intelligences, but also there even are new possibilities that were unconceivable before without these technologies.

And as this project relies on some state-of-the-art technologies, if correctly developed for certain applications, it could easily compete with other already existing commercial applications or products.

6. Conclusions and Future Development

The key of this project is the use of many different parts well combined, it would way surpass the scope of a project of this kind to try to fully develop each of such parts from scratch, in fact some of those parts are current state-of-the-art technologies. So the true challenge has been to correctly combine them as many processes are running at once.

It is impressive how much fully working resources can be found online that rely on the open source idea. All of them may have different approaches when it comes to their actual commercial use, but for the kind of use that it has been given to them, they have come to be very handy.

Also it must be said that once everything is up and running, the improvement of the algorithms feels exponential. And even though the majority of the time has been invested in finding out how to get and combine different resources, if the project happens to be carried on, all of this time could be inverted in directly improve the drone's behaviour as the working environment has already been proved to be not only fully working but also to have a lot of potential.

One last thing to mention about this project is its accessibility, as there is only the need to own the drone (which is not that much expensive and it could even be easily found second handed) and a powerful enough computer (which nowadays many people already own). Or even the drone is dispensable in some applications because the simulator is free as well and it can emulate many real scenarios.

About the possibility to carry on the project, here a list of viable improvements will be presented:

PID Implementation	As seen before, the PID implementation would result very handy in tracking applications or autonomous movement performed by the drone based only on its environment through its view.
Track and Gestures Applications Combination	As already suggested, would be very interesting to combine the tracking algorithm with the gesture controlling. Obviously the possible inputs by the user should be less and the used camera for such gestures should be the drone's. Also there should be a checking to ensure that the face and hands gestures that are taken into account are those from the tracked user and none of the other that may appear in the surroundings.
Phone Application	It would be very handy to be able to perform all of these applications without the need to carry the laptop around, however, no regular phone has the needed computational power, so a running server on which perform all the processing and a good connectivity to send the data to it would be mandatory.
Tracking Improvement	With an already well implemented and fully functional PID, a better version of the already working tracking application could be developed. Letting the drone translate and turn at the same time for example, or even move up and won.

Tracking Adaptation for the Simulator	It has been commented the inability to use the simulator to test the tracking algorithm as it needs a real person to follow. However, a similar version could be implemented so it looks for something like a red cube, which can be easily introduced and moved in the simulator. So even the tracking algorithm could be tested without the need of a real drone.
Personal Tracking	The darknet_ros image detection can be trained with your own pictures, so it would be interesting to train it not to follow a generic person but to follow a particular person. In this way there would not be the need to predict who is the tracked person on each frame as it would not detect all the people as a possible candidate. Even if the tracked person moved away from the drone's perspective, the drone would not track a new candidate as it would be looking for that particular person.
Optional Personal Tracking	This would not be trivial at all, but a potential application would be to set a gesture (like rising both arms for example) to indicate the drone that it should take a few pictures. Then those pictures would be automatically cropped around the detected person box that has just made such gesture, and they would be automatically used to train the network and start following that particular person. Until here it would be a very similar way of the previously commented "Personal Tracking" improvement, but in this way, in the middle of the tracking, another person could rise both arms and the drone would switch to that target.
Movement Prediction	A final potential improvement is to give to the drone the ability to predict the tracked object movement. Maybe taking into account its current velocity and presuming a maximum and minimum acceleration. A clear example scenario could be the following: the tracked person moves from right to left so fast that it moves away from the drone's perspective, so the drone would keep turning in search of it instead of just stopping because it has lost the target. Another possible scenario would be that the tracked person is moving at a constant speed but at some point it goes behind a big object which is in between the drone and the user from one side, and after a couple of seconds it appears on the other side of it. So the drone could presume a constant velocity so it keeps moving even if no tracked person is in sight so it anticipates when the user will appear again from behind such object. Obviously there is no way to predict if the user will change the direction or velocity while it is behind such object.

Table 16 [Potential Improvements]

Bibliography

Drone website [1], developer drone website [2], ROS basics website [3], bebop_autonomy presentation website [4], darknet presentation website [5], CUDA nvidia general accelerator website [6], cuDNN nvidia image detection accelerator website [7], ROS and catkin combination website [8], ROS and Python combination website [9], Sphinx drone simulator website [10], Gazebo robot 3D simulation website [12], Ubuntu installation website [13], bebop_autonomy installation website [14], darknet_ros repository website [15], CUDA installation website [16], cuDNN installation website [17], Sphinx installation website [18]

[1] https://www.parrot.com/
[2] https://developer.parrot.com/
[3] http://www.ros.org/
[4] https://bebop-autonomy.readthedocs.io/
[5] https://pjreddie.com/darknet/yolo/
[6] https://developer.nvidia.com/cuda-zone
[7] https://developer.nvidia.com/cudnn
[8] http://wiki.ros.org/catkin
[9] http://wiki.ros.org/rospy
[10] https://developer.parrot.com/docs/sphinx/
[11] http://gazebosim.org/
[12] https://tutorials.ubuntu.com/tutorial/tutorial-install-ubuntu-desktop-1604
[13] http://wiki.ros.org/kinetic/Installation/Ubuntu
[14] https://bebop-autonomy.readthedocs.io/en/latest/installation.html
[15] https://githubcom/leggedrobotics/darknet_ros
[16] https://developer.nvidia.com/cuda-downloads
[17] https://developer.nvidia.com/rdp/cudnn-download
[18] https://developer.parrot.com/docs/sphinx/installation.html

Glossary

AI – Artificial Intelligence

ROS – Robot Operating System

YOLO – You Only Look Once

GPU – Graphics Processing Unit

CPU – Central Processing Unit

OS – Operating System

PID – Proportional Integral Derivative