POLITECNICO DI TORINO

CORSO DI LAUREA IN MECHATRONIC ENGINEERING

PROVA FINALE

IMPLEMENTATION OF AN IMAGE-BASED VISUAL SERVOING SYSTEM ON A PARROT BEBOP 2 UAV

CANDIDATO

M'GHARFAOUI ILYAS

RELATORE

CHIABERGE MARCELLO

OTTOBRE 2019

Contents

Abstract

The main objective of this proposed thesis is to design and implement two different systems on a Parrot Bebop 2 UAV.

The first system is a Human-UAV interaction (HUI) system to take off, fly and land the UAV by using the camera of the ground station machine to detect and identify the operator's gesture and send different commands to the drone based on the operator's gesture.
A dataset containing different hands postures have been used to train the detection framework to detect the face and hands of the operator.
Then, an algorithm is used to interpret the gestures obtained from the detection results, in which each interpreted gesture is equivalent to a flying command.

The second system is an Image-Based Visual Servoing IBVS controlling system for sending commands to the UAV in order to track and follow a detected object, in this case a person, by using the monocular camera of the drone.
This requires an algorithm that is able to use the detected object geometry and location in the image plane to send commands to the UAV in order to keep the target within a fixed distance and almost in the centre of its Field of View FoV.
To do so, a PID controller have been used to calculate the velocity (horizontal, lateral, vertical and angular) to send to the drone.
A dataset containing different pedestrians have been used to train the detection framework.
To support the detection framework, a tracking framework have been implemented to identify and assign a unique ID to each detected person.
In this way, the drone is able to continuously follow the same person even when in the image plane there are more people detected.

The system components used (deep neural network detector, tracker framework, HUI and IBVS) are built as nodes under ROS environment.
Both systems are verified to work off-board with a ground station machine with the Parrot Bebop 2 drone.

In the chapter 1, a description of the Robot Operating System and of the packages used for this project is given.

In chapter 2, an introduction to Object Detection and Deep Learning is given with details about Convolutional Neural Network and YOLO object detection system.

In chapter 3, a description of the steps taken to implement the Human-UAV in-

teraction system is given. Furthermore, the reasoning behind the interpretation of the gestures by the developed algorithm to send commands is explained.

In chapter 4, an introduction to object tracking and a description of the SORT and deep SORT trackers is given.

In chapter 5, a description of the steps taken to implement the IBVS system is given. Furthermore, the reasoning behind the design of the PID controller and of the object follower is explained.

In chapter 6, a discussion and analysis about the results obtained from both HUI and IBVS systems is given.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 UAV

An unmanned aerial vehicle UAV is an aircraft without a human pilot aboard. The necessity to start using UAVs originated in the military field, where the missions were too dangerous for humans to be executed without incurring in serious injuries or death.
The use of UAVs has rapidly expanded from the military field to the commercial field (with applications like aerial photography, surveillance, scientific research, agriculture, etc) thanks to the advances in computing technologies that allowed the miniaturization of sensors, computers and communication devices.

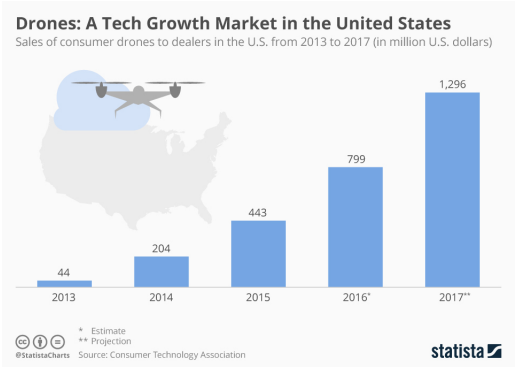


Figure 1.1: USA drone market growth. Source: Consumer Technology Association

Nowadays, drones are dexterous and can be piloted with a remote controller. Despite this, flying a drone with a controller is not straightforward: it can happen to lose control of the drone leading to a crash.
For this reason, a lot of effort has been put into developing autonomous flight software to make the drone fly by itself without the need of remote control. For example, an autonomous drone can follow an object of interest and avoid obstacles in the way. There are already different companies, both private and public, that have developed artificial intelligence software to allow drones to process in real-time what they see and to identify objects and respond to them instantly.

The most popular layout in circulation for small size UAVs is the quadcopter model.

1.1.1 Quadcopter

A quadcopter is a multirotor vehicle that is lifted and propelled by four rotors, of which 2 are clockwise and the other two are counterclockwise. The two couples of rotors have opposite direction in order to balance the torques exerted upon the body of the quadcopter.

Figure 1.2: Quadcopter rotors direction of rotation. From [29]

Each rotor produces both a thrust and torque about its center of rotation. It also produces a drag force opposite to the vehicle's direction of flight.

For this work, a quadcopter Parrot Bebop 2 has been used. The reason for this choice is that it is a low-cost, light weight drone with a stabilized camera for quality footage and an SDK available for developers.

Parrot Bebop 2

The Parrot Bebop 2 is a drone developed by Parrot with 180 degree fisheye lens capable of shooting 1080p HD videos (which is important for the image recognition purpose) and with a built-in GPS.
The wide field of view obtained by the fisheye lens allows to virtually capture all the semi sphere space in front of it and to tilt and pan the camera angle electronically as if it were on a mechanical gimbal thanks to the real-time digital image stabilization. This characteristic makes the drone cheaper than those with mechanical stabilization that need moving parts. Even if the electronic image stabilization is not as good as the mechanical stabilization, it is more than enough for our application.

The following table contains the technical specifications of the drone used during this thesis:

Table 1.1: Technical specifications. From [26]

Weight	380g (400g with hull)
Motors	4 Brushless rotating cage motors (7500 rpm)
Horizontal speed	max 16 m/s
Vertical speed	max 6 m/s
Processor	Dual core processor with quad-core GPU
Storage	8 GB flash storage system
Battery life	2700 mAh - 25 minutes flying
Sensors	GPS, IMU
WiFi	Bi-band MIMO with 2.4 and 5 GHz dual dipole antennas
Output power	Up to 21 dBm
Signal range	Up to 300 m
Video resolution	full HD 1080p with 3-axis digital stabilizer

Figure 1.3: Parrot Bebop 2 drone.

The main components of the quadcopter that are utilized in this thesis are the software of the flight controller and the camera.
The flight controller is the brain of the quadcopter and it uses the data coming from the sensors (camera, GPS, etc) to calculate the velocity at which each motor should be spinning.
For this drone, the company Parrot has made available for developers an SDK (set of software development tools that allows to create applications for a certain hardware platform) called ARDroneSDK3 to connect, pilot and receive video stream from the drone.
This SDK is available as a ROS driver and it will be used for sending commands to the drone.
The camera is used to send the video stream to the ground station, then the video stream will be processed to obtain the detected object thanks to a deep neural network.
The detected objects will be used to decide which commands to send to the drone via the SDK.

1.2 Robot Operating System ROS

1.2.1 Operating system and ROS version

Ubuntu is a free and open-source Linux distribution based on Debian, which is an operating system made from a software collection based upon the Linux kernel and a package management system.
The Ubuntu release used is Ubuntu 16.04 LTS ("Xenial Xerus").
The reason for using Ubuntu Linux-based operating system is that ROS has only experimental Windows support while Ubuntu has full support for ROS.
In addition, Ubuntu is less GUI oriented and most of the work done on this project is executed using the Linux terminal.
The ROS distribution used is ROS Kinetic Kame.

Figure 1.4: Ubuntu Linux-based system terminal

1.2.2 ROS

Robot Operating System ROS is a flexible software framework for robot software development containing tools, libraries for building, writing and running code across multiple computers.
Some of the key advantages of ROS are its modularity, its inter-platform operability and a vibrant community of user-contributed packages that add value on top of the core ROS system. It is licensed under an open source BSD license.

ROS is an open-source meta-operating system (it runs on top of an operating system like Ubuntu) in which users select the configuration of tools and libraries to use in their applications that run on the core of ROS.
The core of ROS is simply the underlying general structure within which applications run and communicate between each other.
Users can create new libraries and make them available to the rest of the ROS community.

Figure 1.5: ROS ecosystem

To synthetize, ROS is the underlying plumbing that runs under the nodes and messages passing.

In ROS, the code is organized in the following way:

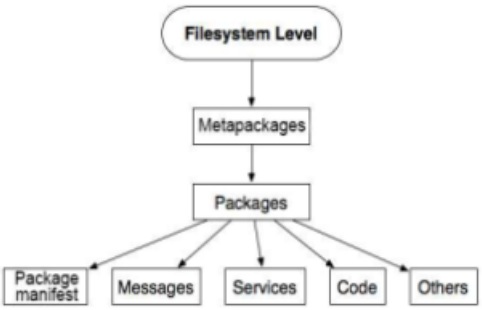


Figure 1.6: Code bundling organization

- **Metapackage:** is a set of packages that have common purpose
- **Package:** is a collection of files, including both executables and supporting files, that serve a specific purpose.
- **Node:** is an instance of a code executable file. Every code/node is always part of a package.

The runtime structure in ROS is the following:

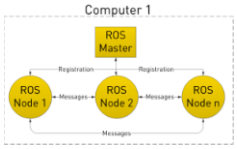


Figure 1.7: Runtime structure

- **Master:** it provides name registration and lookup (name server) for node-to-node connections and message communication. Without the master, nodes would not be able to find each other and exchange messages.
- **Node:** process that performs computation. A ROS node is written with the use of a ROS client library, such as roscpp (C++ implementation of ROS) or rospy (Python implementation of ROS).
  There are 2 types of nodes:
  - Subscriber node: it receives data from the other nodes running on ROS.
  - Publisher node: it sends data to the other nodes running on ROS
- **Message:** is a data structure used by the nodes to communicate with each other.

ROS processes are represented as nodes in a graph structure, each node is able to communicate with another nodes.

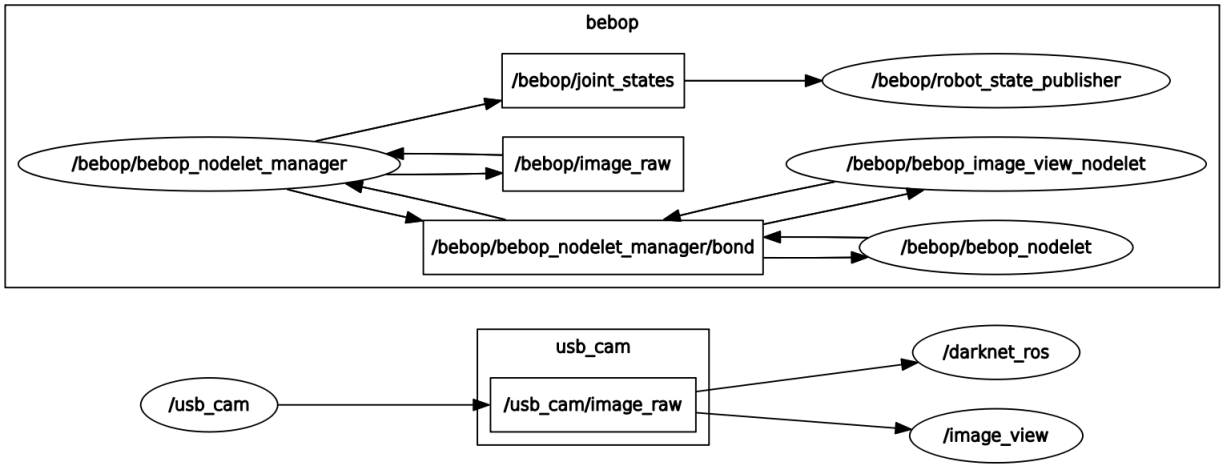


Figure 1.8: Graph structure of some of the ROS packages used

There are different ways for the nodes to communicate between each other:

- **Topics:** it allows a continuous exchange of data. To send a message to a topic, a node must publish to that topic. To receive a message from a topic, a node must subscribe to that topic. The type of message that can be sent via a topic varies and can be user-defined. The message can be composed of sensor data, state information and so on.
- **Services:** it consists of request from a node to another and response/result from the node that receives the request.
- **Actions:** it is similar to a service but with a continuous feedback from the node that receives the request.
- **Parameters:** it is a database shared between nodes. It contains data that does not change frequently.

Before setting up a node-to-node communication, a master node must always be run first.
Every node that wants to communicate with other nodes in ROS must register its node name, topic name, message type, URI address and port with the ROS master. The messages that the nodes want to send between each other do not pass through the master, rather the master is responsible for setting up a peer-to-peer communication between nodes that have registered themselves with the master.

This decentralized architecture is advantageous because a robot consists of a set of networked computers, and it may need to communicate with off-board computers for heavy computation. For example, in this work the video stream of the drone is sent to the ground station laptop where the computation for detecting objects and sending commands is done.

To begin a project in ROS, it is necessary to create a **catkin workspace**, which is a folder where it is possible to modify, build and install catkin packages. The

advantage is that it is possible to build multiple, interdependent packages together all at once.

Catkin is the official build system of ROS responsible for generating targets (libraries, executable programs, generated scripts or anything else that is not static code) from source code that is organized into packages.

A catkin workspace contains different spaces that serve different roles:

- **Source space:** it contains the source code of the catkin packages
- **Build space:** it is where CMake (software tool for managing build process) is invoked to build the catkin packages in the source space
- **Development (Dev) space:** it is where built targets are placed prior to being installed

1.2.3 Packages used for this project

- **bebop_autonomy:** it is a ROS driver for Parrot Bebop drone, based on Parrot's official ARDroneSDK3, developed in Autonomy Lab of Simon Fraser University.
- **darknet_ros:** is an open source neural network framework that runs on CPU and GPU.
- **teleop_twist_keyboard:** generic keyboard for twist robots. This package has been used to use the keyboard of the ground station to control the drone.
- **usb_cam:** to receive video stream from the integrated camera in the ground station laptop. This package has been used for the HUI system design to use YOLOv3 detection framework on the ground station laptop.
- **sort_track:** It is a ROS package that I developed to implement sort and deep sort tracker on ROS. It is available at https://github.com/ilyas95/sort-deepsort-yolov3-ROS

1.2.4 Parrot-Sphinx

Parrot-Sphinx is a simulation tool that allows to run a Parrot drone firmware on a PC, in an isolated environment well separated from the host system. It uses Gazebo to simulate the physical and visual surroundings of the drone.
Parrot-Sphinx can create on the host system real WiFi access points attached to the simulated drone. The simulated drone will use the host system WiFi as if it was using its own WiFi chip to connect to the ROS environment via bebop_autonomy ROS package.
The simulation tool has been a key component in the development of the HUI and IBVS systems because it allowed to simulate the drone instead of using the real one for the algorithm development, especially because there were no secure facilities available to test the drone.

By simulating the drone behaviour, unexpected errors in the drone software development will not cause damage to the drone or to its surroundings.
Moreover, apart from security reasons, being able to use a simulation tool instead of testing every algorithm modification on the field allowed to save time and to improve the algorithm faster.

1.2.5 Gazebo

Gazebo is a robot simulation tool for rapid algorithm testing, regression testing, and so on. It has a robust physics engine, high-quality graphics, and convenient graphical interface.

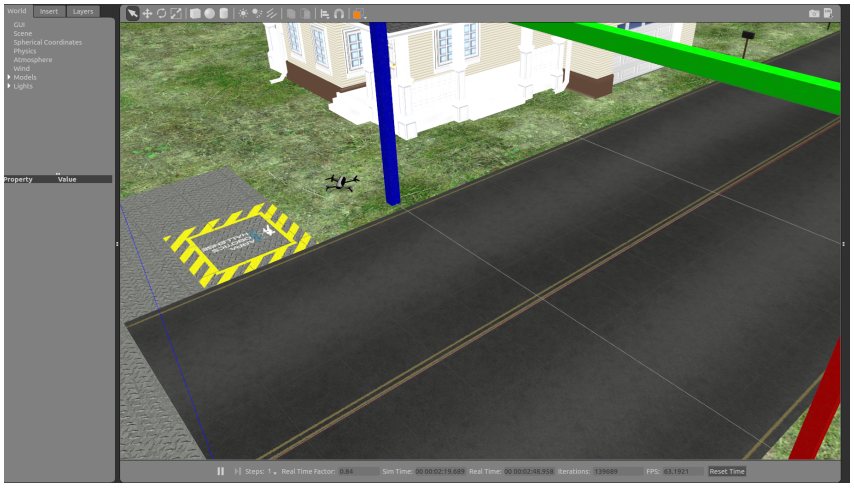


Figure 1.9: Parrot Bebop 2 drone in Gazebo environment

Gazebo can be used to simulate robots in complex indoor and outdoor environments. Its main components are the following:

- **World description file:** it contains all the elements in a simulation (robots, lights, sensors, ect). It is formatted using SDF (Simulation Description Format) and it has a .world extension
- **Model file:** it contains the components (links, joints, plugins) necessary to generate a robot model on Gazebo. A number of robot models is provided in an online model database
- **Gazebo Server gzserver:** it parses a world description file given on the command line, and then simulates the world using a physics and sensor engine.
- **Graphical Client gzclient:** it connects to a running Gazebo server

1.2.6 NVIDIA CUDA and cuDNN

"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 by giving the GPU the ability to speed up more processes than just graphics themselves.

"cuDNN" (CUDA Deep Neural Network) is a NVIDIA GPU-accelerated library for deep neural networks.

CUDA and cuDNN have been used with the deep neural network framework YOLO on ROS in order to acquire a higher rate of frames processed per second as we need to process the input image fast enough to achieve real time performance.
To process the detection of an image, the laptop used requires 0.95s with CPU and 0.013s with GPU: this means that using GPU is 73 times faster.

Chapter 2

Object detection and deep learning

2.1 Object Detection

Object detection is a computer vision technique used to detect objects of a certain class (such as humans, animals, cars, etc.) in digital images and videos. This technique is based on image classification (given an image I obtain as output a single class) and its aim is to localize exactly where the object is located in the image.

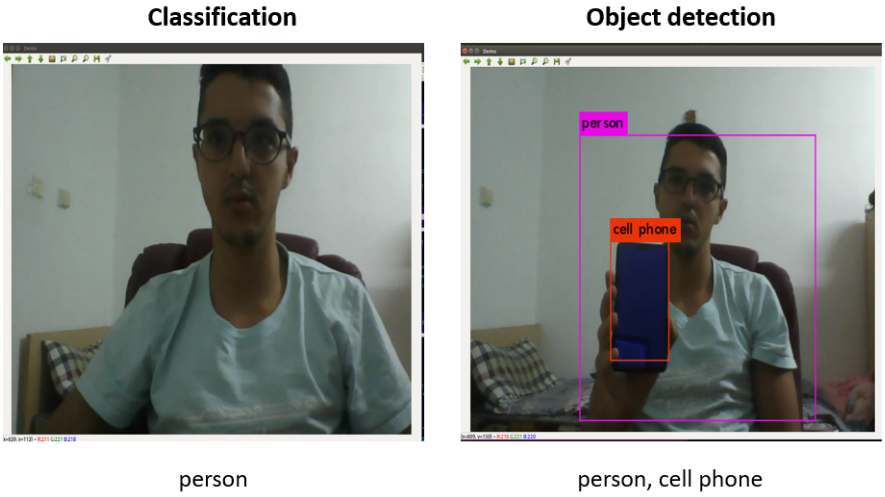


Figure 2.1: Difference between Classification and Object detection.

Given an input image, the output of the object detection task is:

- List of bounding boxes for each detected object
- Class label associated with the bounding box
- Probability/Confidence score associated with each bounding box and class label.

There are 2 general object detection methods used:

- Machine Learning approach

- Deep Learning approach: deep learning techniques are able to do end-to-end object detection without defining features. Deep learning techniques are typically based on convolutional neural networks CNN.

In this project deep learning approach will be used, especially the YOLO detection system.

2.2 Artificial intelligence

Artificial intelligence is a term used to describe any technique that enables computers to imitate human intelligence. Two subcategories of artificial intelligence are machine learning and deep learning.

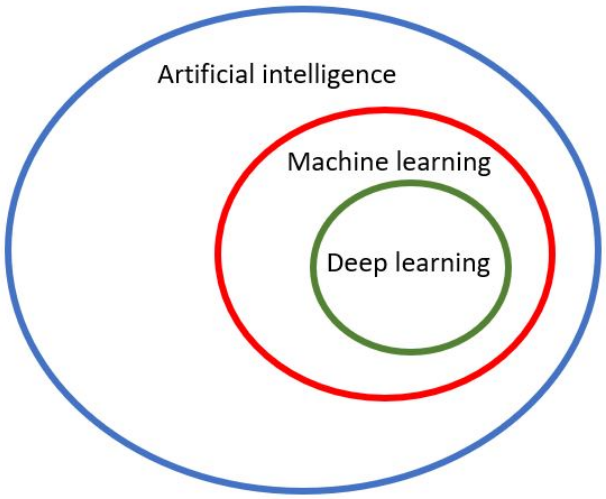


Figure 2.2: Artificial intelligence and its subcategories.

Machine learning ML is the scientific study of algorithms and statistical models that computer systems use to effectively perform a specific task without using explicit instructions, relying on patterns and inference instead.
Deep learning is a subset of machine learning that uses artificial neural networks.
The key difference between machine learning and deep learning is in the way in which the feature extraction is done.

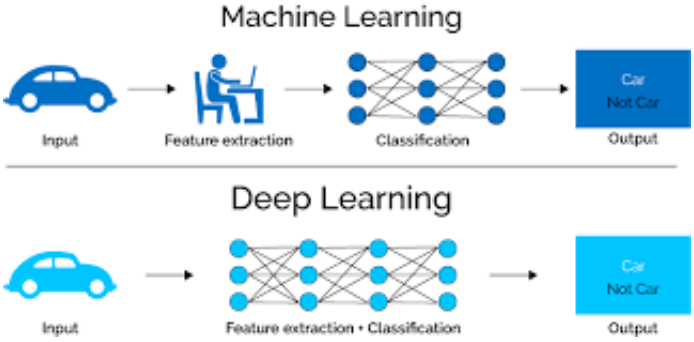


Figure 2.3: Feature extraction difference between machine learning and deep learning. Source: https://codeutsava.in/blog/40

In machine learning the feature extraction is done by human while in deep learning it is done by the deep learning network itself.

2.3 Deep learning

Deep learning became the widely used machine learning technique in academia and industry to solve a large number of problems like computer vision, natural language processing, pattern recognition, etc..
Deep learning is a subset of machine learning that uses multi-layered artificial neural networks to deliver state-of-the-art accuracy in tasks such as object detection.
The tasks are broken down in consecutive layers and each layer builds up on the output of the previous layer. Together the consecutive layers constitute an artificial neural network that imitate the structure and functioning of the human brain cells.
In the human nervous system each neuron has its own learnable weights and biases.
Each neuron is connected with each other to pass information between them.

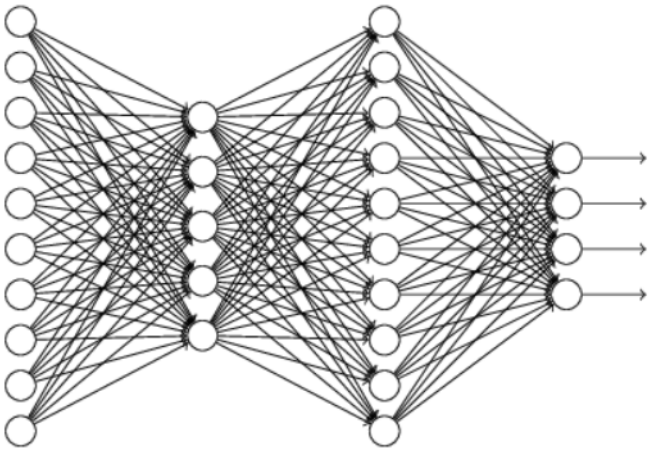


Figure 2.4: Deep network architecture with multiple layers. From [18]

An artificial neural network ANN consists of at least three different layers: input layer, hidden layers and output layer. Each layer accepts the information from the previous one and passes it to the next one.
A deep neural network DNN is an artificial neural network with multiple layers between the input and the output layer.

To introduce the mathematics behind a deep neural network, a simple artificial neural network called multilayer perceptron MLP will be used. [11]

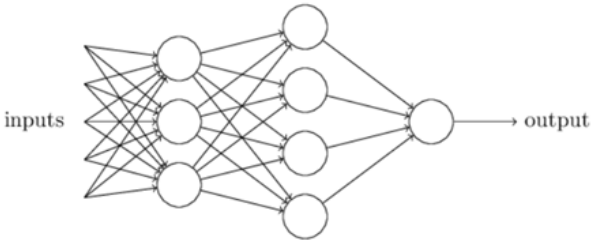


Figure 2.5: Multilayer perceptron neural network scheme. From [18]

In an artificial neural network, each neuron is a function in which the inputs are all the neuron outputs of the previous layer.
An artificial neuron calculates the weighted sum of its inputs and adds a bias term which is used to change the activation threshold by making the neuron inactive for a longer time.

$$Y = \Sigma(weight * input) + bias = (w_1 a_1 + w_2 a_2 + ... + w_n a_n) + bias = [-\infty, +\infty] \quad (2.1)$$

The output of a neuron can be any number in the range $[-\infty, +\infty]$ but we want to obtain an output in the range [0, 1] to activate the neuron (make it "fire") when it is greater than a threshold because with a range $[-\infty, +\infty]$ the neuron doesn't know the bounds of the value.
To limit the neuron output range, we insert the output of each neuron in an activation function. In this way, the neuron output with the activation function will be in the range [0,1].

An activation function is helpful because it also introduces a non-linearity into the output of the neuron.
There are different activation functions: step function, linear function, sigmoid function, ReLu function and so on.
As we want to introduce non-linearity in our network, a suitable activation function is the sigmoid function which is defined by:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (2.2)$$

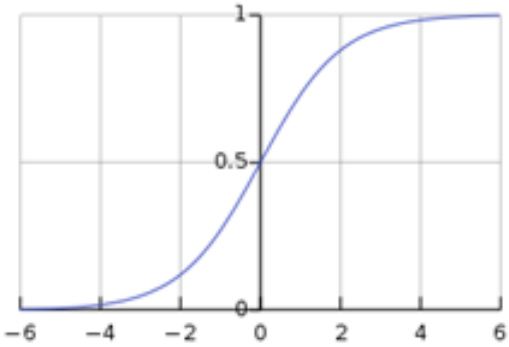


Figure 2.6: Sigmoid function graph.

With the activation function, the output of the neuron becomes:

$$Y = \sigma[\Sigma(weight * input) + bias] = [-1, +1] \tag{2.3}$$

With the activation function, small changes in the weights and bias will cause only small changes in their output.
To synthetize, an activation can be defined as:

$$a^{(1)} = \sigma(Wa^{(0)} + b) \tag{2.4}$$

An activation in one layer determines the activation in the successive layers. In this formula (1) represents the current layer while (0) represents the previous layer.

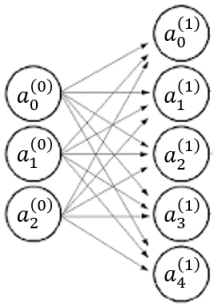


Figure 2.7: 2 layers of an artificial neural network

We can write an equation with the activations from a whole layer as:

$$\sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & ... & w_{0,n} \\ w_{1,0} & w_{1,1} & ... & w_{1,n} \\ ... & ... & ... & ... \\ w_{k,0} & w_{k,1} & ... & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ ... \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ ... \\ b_n \end{bmatrix}\right) \tag{2.5}$$

For an artificial neural network learning means finding the right weight and bias of each neuron.
Learning is done by training the neural network with a suitable dataset. After the training phase, there is a testing phase in which we feed labelled data to test the accuracy of the neural network.
At the beginning of the training, weights and bias are initialized randomly.
To evaluate the performance and to tune the parameters of the neural network, we use a cost function.
A cost function is a measure of the performance of the neural network with respect to the given training samples and the expected output. A neural network training is an optimization problem in which we seek to minimize a cost function.

There are different cost functions that can be used for training a neural network and they all must satisfy the following two properties:

- The cost function C must be written as an average over cost functions C_x (x is the individual training sample) in order to compute the gradient for the

gradient descent algorithm.

$$C = \frac{1}{n} \sum_{x} C_x \tag{2.6}$$

- The cost function must not be dependent upon any activation layer besides the output layer.

One of the most used cost functions is the quadratic cost also known as mean squared error.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 \tag{2.7}$$

where Y_i is the output of the activation layer and $\hat{Y}_i$ is the desired output.

The output of a cost function is a scalar number because it evaluates the neural network as a whole. The output is a small value when the network performs well and it is a large number when it doesn't performs well.

The algorithm used for minimizing the cost function is gradient descent which attempts to find a local or global minimum of a function by learning the direction/gradient that the model should take to reduce errors.

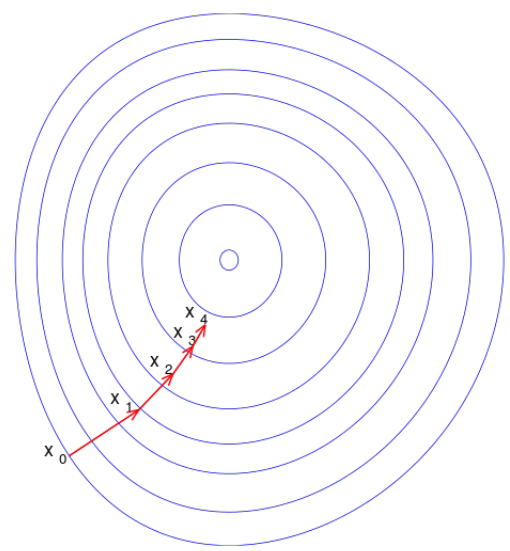


Figure 2.8: Illustration of the gradient descent algorithm. From [28]

Deep learning networks can be categorized according to their structure "architecture" and learning method.

2.3.1 CNN Convolutional Neural Network

It is a widely used deep learning method, especially for computer vision applications like object detection.
An application function such as the Softmax function is applied to classify an object with probabilistic values ranging from 0 to 1.

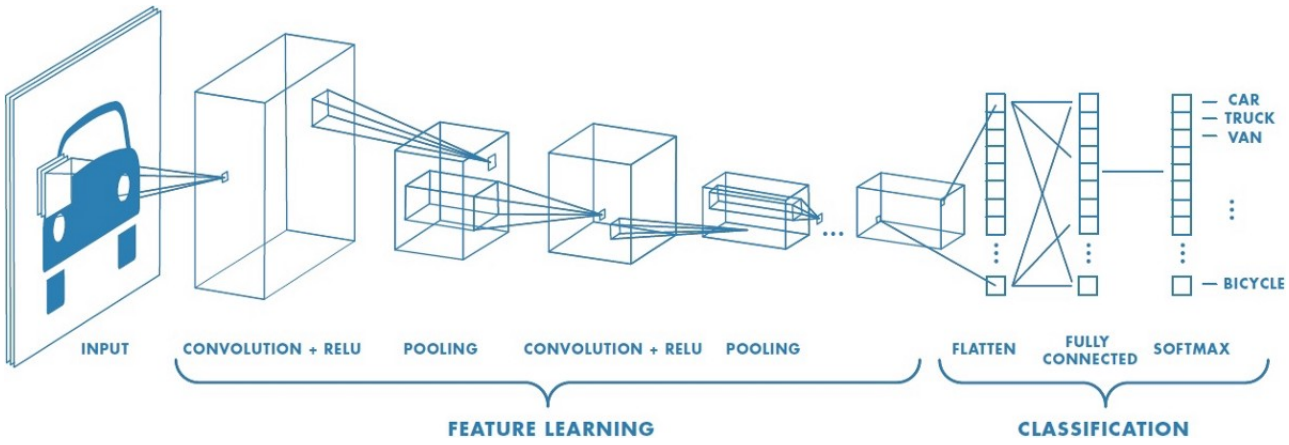


Figure 2.9: Convolutional neural network architecture. From [27]

Unlike standard neural networks, in the CNN architecture the input is a 4-D tensor which is a matrix of numbers with additional dimensions. An image represented by RGB (Red-Green-Blue) encoding produces three layers matrices.
A convolutional neural network is composed of:

- Convolutional layer: it is the first layer of the CNN and it has the function to extract features from an input image by doing a convolution operation between a set of kernels (filters) and the input image.
  The output of the convolution operation is called "Feature Map".
  Different types of filters produce different independent feature maps and typically a convolution layer includes a set of different filters in order to create different feature maps.
  As the real-world data is not linear, we need to introduce non-linearity in the CNN layer input. To do so, we use ReLU which stands for Rectified Linear Unit for a non-linear operation.
  ReLU is one of the most used non-linear functions due to its performance.

- Pooling layer: it is used to reduce the number of parameters of each feature map in order to reduce the computation in the network while retaining the important information. The pooling layer operates on each feature map independently.
  There are different types of spatial pooling: max pooling (most common approach used), average pooling and sum pooling.

- Fully connected layer: the feature map matrices are converted as vectors and then we combine the vectors together to create a model.

The last step is to use an activation function (such as softmax or sigmoid) to classify the output of the convolutional neural network into labels/classes.
We will focus on applying CNN architecture for detection task since it is considered the most effective method in real life computer vision applications.

2.4 Fast Deep Detection Framework YOLO

2.4.1 YOLO You only look once

YOLO is a state-of-the-art, real time fast deep detection framework ([1],[2],[3] have been used as reference).
Instead of repurposing classifiers to perform detection as other detection systems, YOLO applies a single neural network that divides the image into regions and predicts bounding boxes and probabilities directly from the images for each region in one evaluation. These bounding boxes are weighted by the predicted probabilities.

The YOLO detection system workflow is the following:

- Input image is resized to 448 x 448 pixels
- Run single convolutional network on the resized image
- Threshold the resulting detection based on the model's confidence

The key advantage of YOLO is the fact that it unifies separate components of object detection into a single neural network that is able to reason globally about the full image and its objects.
The neural network uses features from the entire image to predict each bounding box and it also predicts all bounding boxes for an image simultaneously.
To localize the target, YOLO models the detection as a regression problem.
After resizing the input image to 448 x 448 resolution, the detection system divides the input image into an S x S grid and each grid cell predicts B bounding boxes, confidence scores for those boxes and C class probabilities.
Each grid cell predicts only one object regardless of the number of bounding boxes B. This limits how close detected objects can be.

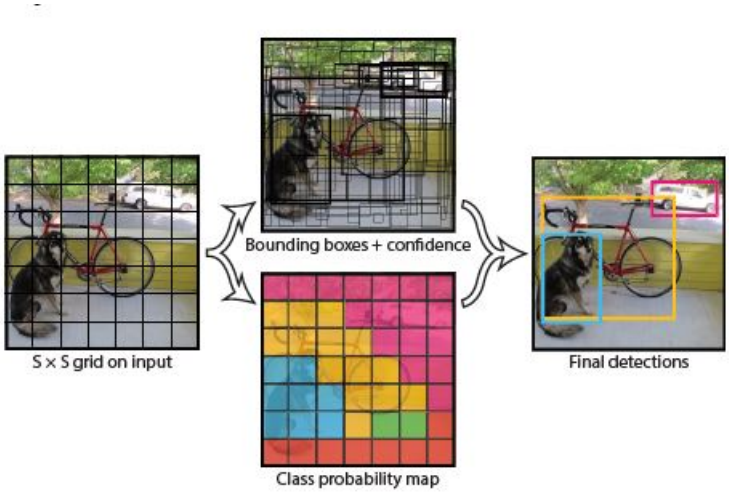


Figure 2.10: YOLO detection system model. From [1]

The output of the detection system consists of bounding boxes representing the detected objects. Each bounding box has 5 different predictions: x, y, w, h and a box confidence score:

- The (x,y) coordinates represent the bounding box center relative to the boundary or the containing grid cell (thus are between 0 and 1).

- The (w,h) coordinates represent the width and height of the bounding box, normalized by the image width and height (thus are between 0 and 1).

- The confidence represents the "objectness" score of the detection, which means the presence of an object (1 if there is an object and 0 if there is no detected object) multiplied by the IoU Intersection over Union between the predicted box and the ground truth box.

The architectural design of YOLO neural network comprises two stages (feature extraction and detection stage) and has 24 convolutional layers for the feature extraction followed by 2 fully connected layers for the detection stage in which 1 x 1 convolutional layers are alternated with 3 x 3 convolutional layers to reduce the depth of the features map from preceding layers.

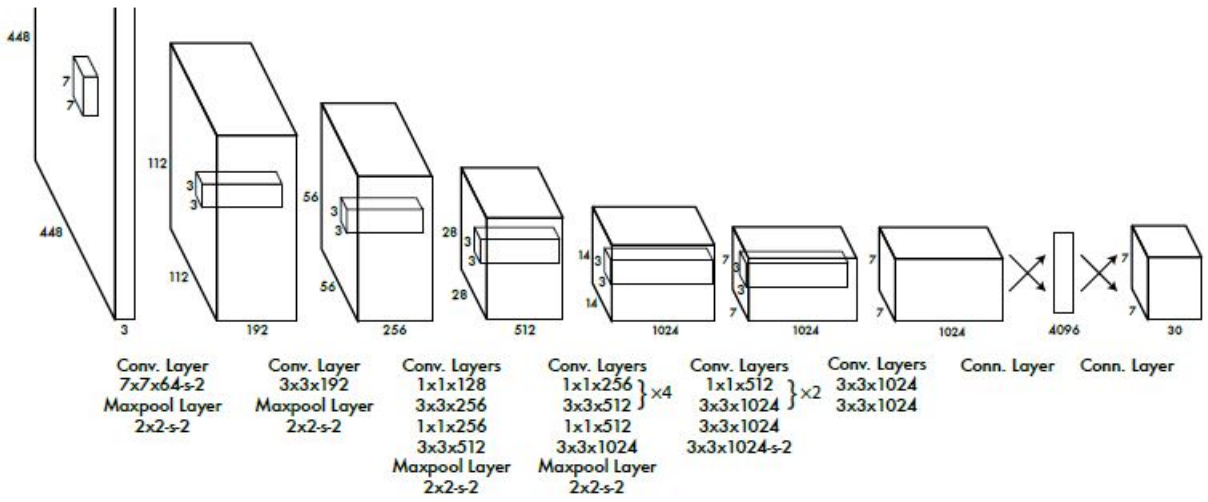


Figure 2.11: YOLO detection system architecture. From [1]

As detection requires rigid visual information, the input resolution of the network is increased to 448 x 448.
The final layer of the network predicts the class probabilities and the bounding box coordinates.
The final output of the neural network is a 7 x 7 x 30 tensor of predictions, given as (S, S, B x 5 + C).

The reasons for using YOLO deep learning detection system are the following:

- It is fast: this means it is good for real-time processing.

- Predictions are made from one single network.

- It accesses to the whole image in predicting boundaries.

2.4.2 YOLOv3

YOLOv3 is the third version of YOLO and it is the version used in this project.
For the feature extractor, YOLOv3 uses a new CNN architecture.
A new 53-layer Darknet-53 is used instead of the YOLOv2 Darknet-19. It is composed of 3 x 3 and 1 x 1 convolutional layers with some shortcut connections.
Darknet is an open source neural network framework written in C and CUDA.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2.12: Darknet-53. From [3]

YOLOv3 is better and more accurate than its previous versions but it is not faster due to the increase of complexity of the underlying architecture Darknet-53.
The total number of convolutional layers underlying YOLOv3 architecture is 106: in addition to the 53 original layers of Darknet trained on Imagenet, 53 more layers have been added for the task of detection.
The most salient feature of the new version is that it detects objects at different layers: the detection is done by applying 1 x 1 kernels on feature maps of three different sizes at three different places in the network.
For an input image of the same size, YOLOv3 predicts 10 times the number of boxes predicted by the previous version but it is slower due to the tradeoff to increase its accuracy.

Chapter 3

Human-UAV interaction for drone controlling

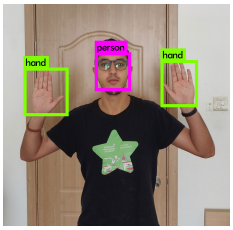
3.1 Human-UAV interaction

Humans have a natural desire to use their body for communication purposes (body launguage). For this reason, a Human-UAV interaction HUI system based on deep detection framework has been designed using [10] as reference.

This HUI system consists of an intuitive real-time system for controlling a parrot bebop 2 drone using a pre-designed and trained set of human gesture dataset. The real object detection system on which the dataset has been trained is YOLOv3.

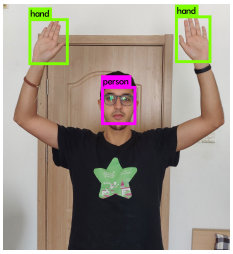
The dataset includes images focusing on human hands and faces with a resolution of approximately 1200x1000. The dataset includes different positions of the hands and face of the operator in order to later be able to send different commands to the drone according to the particular position of the face and hands of the drone operator.

The different commands that have been implemented with the HUI system are the following:

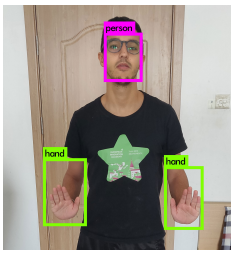


(a) Take off command

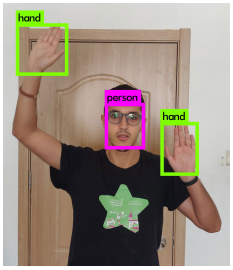
(b) Landing command



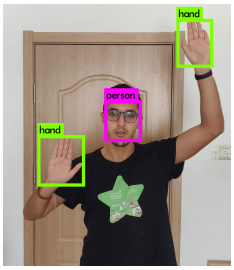
(c) Fly up command



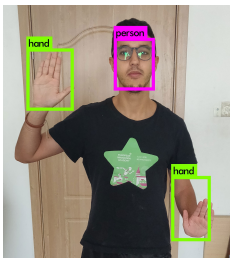
(d) Fly down command



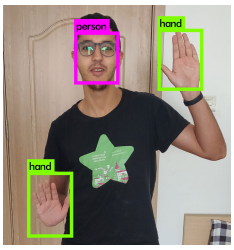
(e) Fly forward command



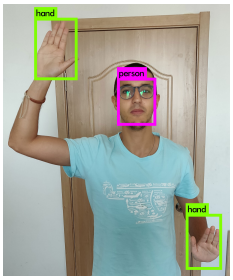
(f) Fly backward command



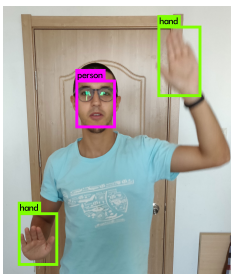
(g) Fly right command



(h) Fly left command



(i) Clockwise rotation around Z axis command



(j) Anticlockwise rotation around Z axis command

Figure 3.1: All the implemented gesture flight commands

When no command is sent, the drone will automatically keep hovering.

The real object detection system will detect the operator gesture (right and left hands and head) in each frame and it will publish three bounding boxes.

3.2 Drone controlling algorithm design

A python script interfacing with ROS has been written to use the detected objects for sending flight commands depending on the operator hands and face position.
In the script, a Python client library for ROS called rospy has been used.
The rospy client API enables Python programmers to quickly interface with ROS Topics, Services and Parameters.
While developing the algorithm, instead of using the real drone, a simulation tool called Sphinx has been used to run the drone firmware on a PC and Gazebo has been used to simulate the physical and visual surroundings of the drone.
Instead of using the drone camera, the ground station laptop camera has been used for detecting the gestures of the operator.
The reason for this choice is that the laptop camera is static as the ground station will not move. Instead, the drone will move based on the operator gestures so the drone camera can lose sight of the operator.

The algorithm subscribes to the topic /darknet_ros/bounding_boxes in order to receive from YOLO detection system the messages with bounding box coordinates, classes and probabilities of each detected object.
For each detected object, the algorithm will calculate its center coordinates (C_x, C_y) and then it will save them only if the class is "face" or "hand".
In particular for the class "hand", the algorithm will save the bounding box centre coordinates only if there is already a saved bounding box with the class "hand". The reason is because we want to send commands only if in the image plane there are 2 hands and one face.

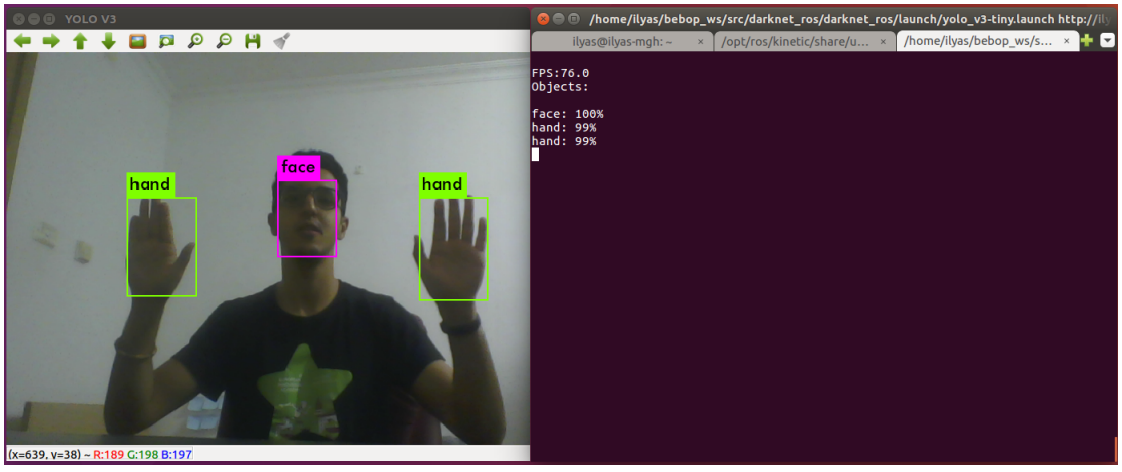


Figure 3.2: Detection of hands and face using darknet_ros package

Table 3.1: List of commands

HANDS AND FACE POSTURE	R_{dx}	R_{sx}	**COMMANDS**
All aligned at face level	0.9:1.1	0.9:1.1	Take off
Two hands aligned beneath far from the face	0:0.6	0:0.6	Landing
Two hands aligned above the face	1.6:2.4	1.6:2.4	Fly up
Two hands aligned beneath the face	0.6:0.85	0.6:0.85	Fly down
Left hand aligned with the face and right hand above	>1.6	0.8:1.2	Fly forward
Right hand aligned with the face and left hand above	0.8:1.2	>1.6	Fly backward
Right hand aligned with the face and left hand beneath	0.8:1.2	<0.7	Fly left
Left hand aligned with the face and right hand beneath	<0.7	0.8:1.2	Fly right
Left hand beneath and right hand above	>1.6	<0.7	Clockwise rot
Left hand above and right hand beneath	<0.7	>1.6	Anti-clockwise rot

The algorithm takes the hand which has lower horizontal location (x-values) as a "Right hand" and the other one as a "Left hand". Then we use the ratios between vertical centers (y-values) of the three bounding boxes:

$$R_{dx} = f_c/h_{dx} \tag{3.1}$$

$$R_{sx} = f_c/h_{sx} \tag{3.2}$$

where f_c is the vertical coordinate of face centre, h_{dx} is the vertical coordinate of the right hand and h_{sx} is the vertical coordinate of the left hand.

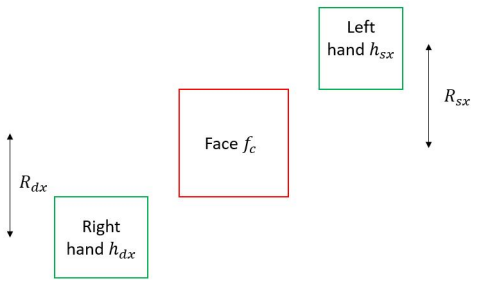


Figure 3.3: Ratio between hands and face vertical position

The advantage of using the ratios of the hands posture with respect to the operator's face is that it provides scale invariance: this means that it does not matter where the operator will stand in front of the ground station camera.

Based on the ratios R_{dx} and R_{sx}, the algorithm sends the right flight commands to the drone by publishing a ROS message of type stdmsgs/Empty to the topic /bebop/takeoff and /bebop/land respectively for taking off and landing. To move the drone, the algorithm publishes ROS message of type geometry_msgs/Twist to the topic bebop/cmd_vel.

Twist is a message type part of geometry_msgs that expresses the velocity into its linear and angular parts.

The algorithm is initialized with a state "landed" and after sending the command take off the state is changed to "flying". The take off command can be sent only if

the state is “landed”.
The flying and landing commands can be sent only if the state is “flying”.

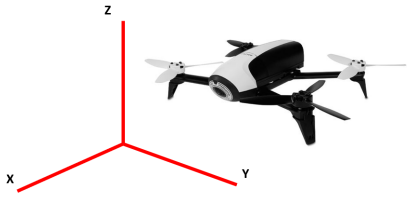


Figure 3.4: Drone reference system

Bebop autonomy package allows to send velocity commands with range [-1,1] m/s, and in order to not make the drone movement too rapid a velocity of 0.3 has been chosen through trial and error.

Table 3.2: Possible movement implemented with gesture

Movement type	**Command type**
Forward / Backward movement	twist.linear.x = ± 0.3
Up / Down movement	twist.linear.z = ± 0.3
Left / Right movement	twist.linear.y = ± 0.3
Left / Right rotation around Z axis	twist.angular.z = ± 0.3

The parrot bebop 2 drone is controlled through its driver/SDK within the ROS framework by means of Wi-Fi.
The ground station is a laptop with processor Intel Core i7-7700HQ @2.4GHz and equipped with NVIDIA GPU GeForce GTX 1050. The drone uses an ARM Dual core.

The architecture of the HCI system is the following:

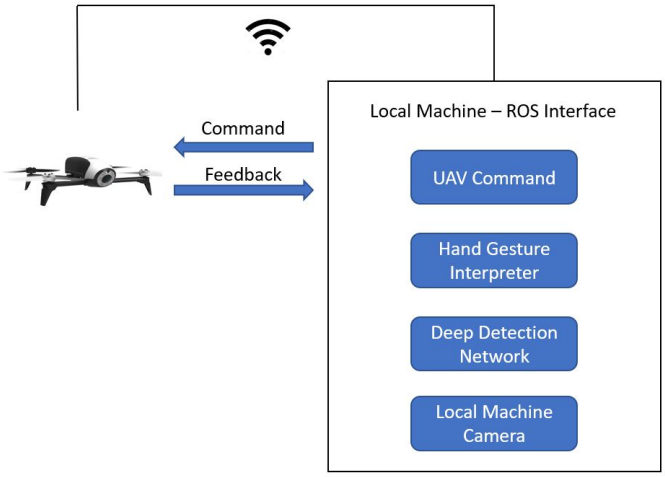


Figure 3.5: Gesture interpreter architecture

3.3 Human Gesture Database

To develop the Human-UAV interaction HUI, a new dataset with approximately 6000 images focusing on human's hands and faces have been created with the background focusing on both indoor and outdoor environments.
Each image of the dataset has been manually annotated to later train YOLOv3 to detect hands and face.
Data annotation consists in creating a text file for each image in the dataset with the object number and object coordinates.
Each text file has the following structure (YOLOv3 format):
[category number] [object center in X] [object center in Y] [object width in X] [object width in Y]
Before starting the training, we divide the dataset in training set and test set and prepare the YOLOv3 configuration files needed for the neural network to know how and what to train. The percentage of images used for the test set is 10%.
We will prepare 3 files

- **.data**: it says how many classes we are training, the location of the training and validation set files, the file containing the names of the categories and the location of the backup file in which the yolo weights will be stored.
- **.names**: it say the category names.
- **.cfg**: it says the chosen yolo architecture.

As the ground station laptop is equipped with NVIDIA GPU GeForce GTX 1050 the **tiny-yolo.cfg** architecture, which is a small model for constrained environments, has been chosen to obtain real time performance.

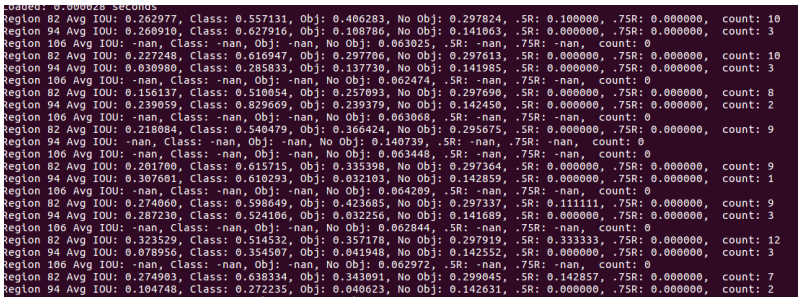


Figure 3.6: Server terminal during dataset training

The training has been done in the cloud because the laptop graphic card is not powerful enough to complete the training in a suitable amount of time.
The result of the training is a file containing the final neural network weights that will be used by the neural network detection system to detect custom objects.

Chapter 4

Object tracking

Up until now, a real time object detection system has been used to scan and search for a particular object (in our case hands and face) but relying only on object detection is not a good idea because, compared to object tracking, object detection has many drawbacks.
Object tracking allows to locate an object in successive frames of a scene and it has the following advantages:

- it is faster than detection: the reason is because the tracking algorithm has a lot of information about the detected object that has to track. For example, it knows the appearance of the detected object, the location in the previous frames, the direction and the speed of its motion and so on. This means that in the next frames, all this information can be used to locate accurately the object.
- works as a backup in case the detection fails: when the object detected is occluded, most of the time the detector will fail to detect it.
- it preserves the identity of each detected object: with detection it is not possible to be able to attach an identity to the detected objects and maintain it. On the contrary, this is possible with object tracking.

It is common for tracking algorithms to accumulate errors and as a result the bounding box of the tracked object starts to slowly drift away from the object it is tracking. For this reason, in an efficiently designed system, the detector is run on every n^{th} frame while the tracker is employed in the frames between the detector.

4.1 Tracker implementation

For the IBVS system to work in real time we need to use a tracking system that is fast in processing the received frames in real time.
The most popular and simplest algorithm for doing so is SORT.
The following papers have been used to implement this tracker: [5], [6]

4.1.1 SORT tracker

SORT (Simple Online and Realtime Tracking) is a simple tracking algorithm that can track multiple objects in real time by using Kalman Filter [7] to handle the

motion prediction for each tracked object and Hungarian assignment algorithm [8] for the data association problem.
It focuses on frame-to-frame prediction and association.
The SORT tracker uses an off-the-shelf object detection system to obtain the detected object bounding boxes and confidence scores in each frame: for this reason, detection quality is a key factor influencing the tracking performance.
The output of the tracker are the bounding box of the tracked object and its unique ID in the following frames.
The reason behind SORT tracker methodology of relying heavily on detection is that nowadays, thanks to deep learning, object detection algorithms have considerable improved to the point that a simple tracking algorithm is adequate to achieve an accuracy comparable to state-of-the-art trackers that are heavier in terms of computational cost.
Instead, due to its tracking method simplicity, SORT is over 20x faster making it fit for real-time applications.

The drawbacks of SORT tracker are mainly three:

- It depends heavily on detection system.
- It has a high number of identity switches due to missed detections by the detector framework that cause the tracker to reinitialize the same object with a new ID when the same object is detected again.
- It cannot handle short-term and long-term occlusion.

Estimation Model and Kalman Filter

The state of each detected object is modelled as:

$$x = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]^T \tag{4.1}$$

where u and v represent the centre of the target (respectively horizontal and vertical location) while s and r are respectively the area and the aspect ratio.
The aspect ratio of the image is the ratio between width and height.

A Kalman filter is an algorithm that combines information about the state of a

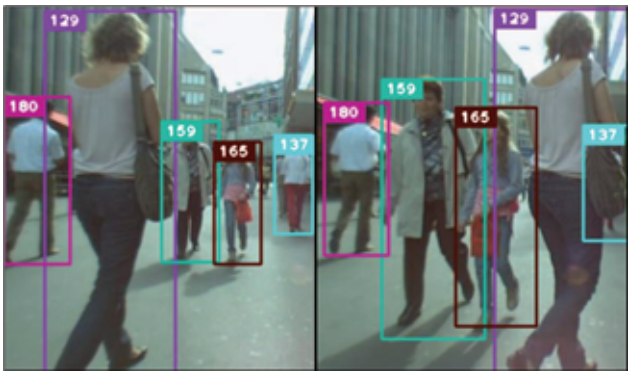


Figure 4.1: Output of SORT tracker in a common tracking situation. Image from [6]

dynamic system with measurements related to the system acquired by sensors in order to predict the behaviour of the system in the future.
A dynamical system can be represented in a state-space mathematical model.

$$\begin{aligned} x_t &= A_t x_{t-1} + B_t u_t + w_t \\ y_t &= C_t p_t + v_t \end{aligned} \tag{4.2}$$

where x_t is the state vector, y_t is the output vector, w_t is the process noise and v_t is the measurement noise.

The Kalman filter is an iterative algorithm that includes two steps: prediction and measurement update.
The prediction phase consists of estimating the system state in the next step based on the previous state estimate and the process noise.

$$\begin{aligned} \hat{x}_t^- &= A_t \hat{x}_{t-1}^- + B_t \overline{u}_t \\ \Sigma_t^- &= A_t \Sigma_{t-1} A_t^T + Q_t \end{aligned} \tag{4.3}$$

where $\hat{x}_t^-$ is the predicted (a priori) state of the system and Σ_t^- is the predicted (a priori) covariance estimate.

The measurement update phase consists of updating the estimated state using information coming from the sensors.

$$\begin{aligned} K_t &= \Sigma_t^- C_t^T (C_t \Sigma_t^- C_t^T + Q_t)^{-1} \\ \hat{x}_t &= \hat{x}_t^- + K_t(\overline{z}_t - C_t \hat{x}_t^-) \\ \Sigma_t &= (I - K_t C_t)\Sigma_t^- \end{aligned} \tag{4.4}$$

K_t is the Kalman Gain and it computes how much the estimation should be corrected given a measurement. It is used to update the a priori estimated state and to update the a priori covariance estimate.

$(\overline{z}_t - C_t \hat{x}_t^-)$ is the Innovation term: it adjusts the a priori estimate in order to obtain an a posteriori estimate. It is the difference between the measured quantity $\overline{z}_t$ and the quantity that we expect to measure from our a priori estimate $C_t \hat{x}_t^-$.

After assigning a new detection to an existing tracked object, the bounding box is used to update the state of the detected object where the velocity components are solved by using the Kalman filter framework.
If there is no new detection, the state is simply predicted without correction.

Data association and Hungarian algorithm

According to the Kalman filter model, the state of each detected object is updated thus also the bounding box of each detected object is updated in the current frame. The new updated bounding boxes are assigned to the existing objects.
This assignment is based on a similarity function and it is solved using the Hungarian algorithm.

The Hungarian algorithm solves an assignment problem in polynomial time by finding an optimal solution from a finite set of possible solutions.
The algorithm consists of four steps of which the first two are executed once while the other two are repeated until an optimal assignment is found. The input of the algorithm is an nxn cost matrix.

The steps are the following:

1. For each row, subtract the lowest element from each element in that row.

2. For each column, subtract the lowest element from each element in that column.

3. Cover all zeros in the resulting matrix after the first 2 steps using a minimum number of horizontal and vertical lines. If the number of lines required is equal to the dimension of the cost matrix, an optimal assignment exists among the zeros and the algorithm stops.

4. Create additional zeros in order to make the number of lines that cover the zeros equal to the dimension of the cost matrix. The creation of additional zeros is done by finding the smallest element that is not covered by a line in the previous step and then subtract it from all not covered elements and add it twice to all elements that are covered.

The Hungarian algorithm for the linear assignment uses the intersection-over-union IOU distance (bounding box overlap) as association metric.
The IOU is simply an evaluation metric of the performance of the predicted bounding boxes with respect to the detected bounding boxes.
The assignment cost matrix is computed as the intersection-over-union IOU distance between the detections performed by the detection system and all the predicted bounding boxes performed by the Kalman filter.
In order to avoid overlap between detection and predicted bounding boxes, a minimum intersection-over-union IOU distance is imposed.

4.1.2 DEEP SORT tracker

In order to reduce the ID switches and improve the occlusion handling, an improved version of SORT with appearance information integrated will be used in this work.
The high number of ID switches is due to the only association metric used (bounding box overlap) which is accurate only when the state estimation uncertainty is low.
To overcome this issue, "SORT tracker with deep association metric" combines motion and appearance information as association metric.
In addition, it applies a convolutional neural network CNN trained to discriminate persons: in this way the tracker robustness is increased while keeping the system simple and applicable to real time scenarios.

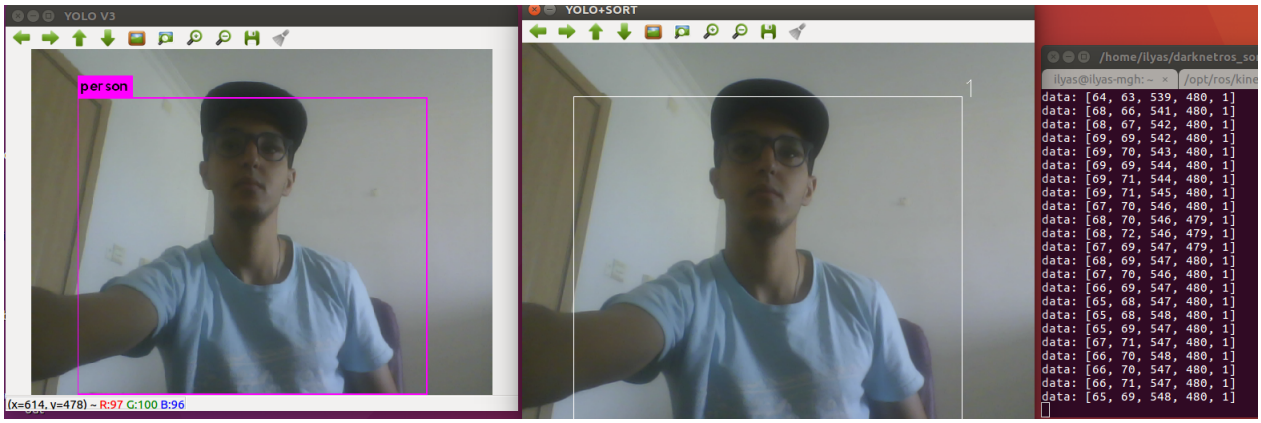


Figure 4.2: Left windows shows output of YOLOv3-tiny while right window shows output of deep SORT tracker with ID assigned. Both implemented on ROS

As SORT tracker with deep association metric is more robust and has less ID switches it will be used for the implementation of the IBVS system.

Chapter 5

Drone object following and control

5.1 Image-Based Visual Servoing IBVS system

The main idea is to implement on the Parrot Bebop 2 drone an algorithm that is able to track a detected object (in this case a person) and follow it by keeping some distance ([9] has been used as reference).
To implement a real-time tracking and follower the processing of the drone camera images must be fast. For this reason, the resolution of the drone camera is reduced to 856 x 480 pixels to help process quickly the images so that the control commands will act properly when they are sent to the drone.
For the object detection, the same detection system YOLOv3 will be used.

The drone is equipped with a single monocular camera so it is not possible to extract depth information from it. To estimate the detected object distance we assume that the object is rigid and that a change in the object size is due to a change of its distance from the camera.
An ideal bounding box centered in the image plane will be used as a reference for the control algorithm.

The goal of the algorithm is to minimize the error between the bounding box of the detected object and the ideal bounding box.

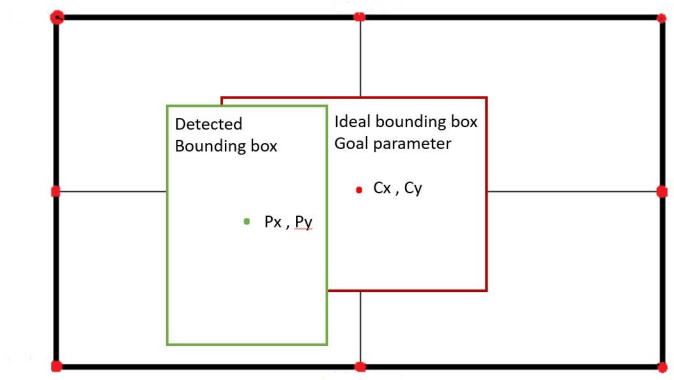


Figure 5.1: Detected bounding box and goal parameter bounding box: we want to minimize the error between them.

The algorithm will send commands to the drone until the goal parameters are reached. A PID controller is used to achieve the goal parameters ([11] has been used as reference).
The goal parameters are the centre of the image (428,240) and an ideal bounding box area of 30% the pixel space. The ideal area has been selected through trial and error.
Assuming that the object is rigid and its size doesn't change, any change in size is due to the change of the object distance to the camera.
For example, a bigger bounding box means that the object is close while a small bounding box means that the object is far.

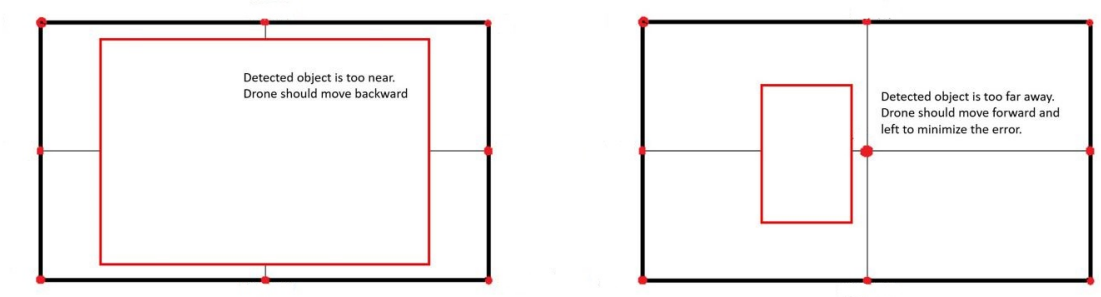


Figure 5.2: In the left detected object is too near while in the right it is too far away.

To be able to follow the detected object properly, at least 2 different PID controllers are necessary: for the X axis (forward/backward movement) and the Y axis (lateral movement left/right).

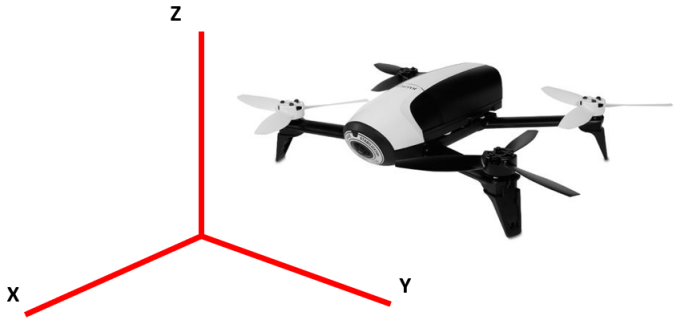


Figure 5.3: Coordinate system with respect to the drone. X forward horizontal movement, Y left lateral movement, Z vertical movement

To make the drone movements complete, another PID controller for the Z axis (vertical movement) has been used: in this way, the drone can regulate its height in order to have the target centered.
In addition to the 3 linear movements (X, Y and Z), an additional rotation around the Z axis based on the output of the Y axis PID controller has been added to make the drone more versatile.
The position error is calculated as:

$$e_{px}(t) = Area_{id} - Area_{ref} \tag{5.1}$$

where $Area_{ref}$ is the area of the tracked person while $Area_{id}$ is 30% of the image plane area (856x480x0.3).

$$e_{py}(t) = Cy_{centre} - Cy \tag{5.2}$$

where Cy_{centre} is the horizontal center of the image plane (428px) while Cy is the horizontal center of the person bounding box.

$$e_{pz}(t) = Cz_{centre} - Cz \tag{5.3}$$

where Cz_{centre} is the vertical center of the image plane (240px) while Cz is the vertical center of the person bounding box.

The output of the PID controllers is the drone velocity but the input error is in pixel: for this reason we have to convert from position error (pixels) to velocity values.
This is done by normalizing the error.
For the lateral and vertical error, 428 px will have the value of 1 and -428 px will have the values of -1.

$$V_y(t) = \frac{e_{py}(t)}{428} \tag{5.4}$$

$$V_z(t) = \frac{e_{pz}(t)}{428} \tag{5.5}$$

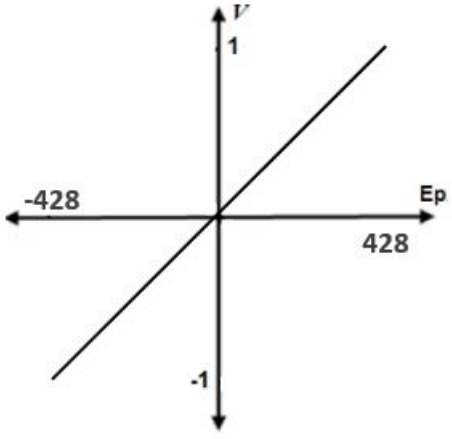


Figure 5.4: Normalization of the lateral and vertical position error

The same normalization have been done for the depth position error, the only difference is that we have used the ideal area for the normalization.

$$V_x(t) = \frac{e_{px}(t)}{Area_{ideal}} \tag{5.6}$$

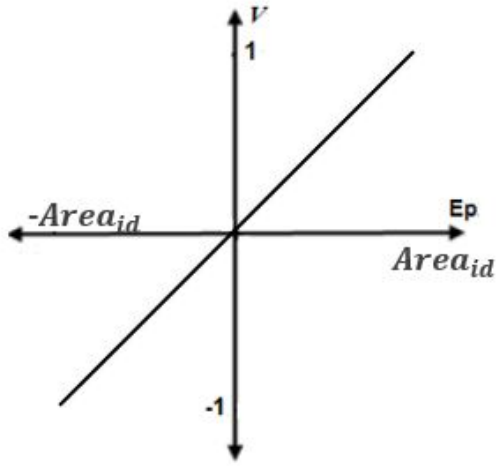


Figure 5.5: Normalization of the depth position error

This table summarizes the data used for the design of the 3 PID controllers:

Table 5.1: PID commands

PID	**position error**	**velocity error**	**COMMAND**
X axis	$e_{px}(t)$	$V_x(t)$	Forward/Backward
Y axis	$e_{py}(t)$	$V_y(t)$	Left/Right + Rotation around Z
Z axis	$e_{pz}(t)$	$V_z(t)$	Up/Down

The bebop_autonomy ROS package allows to send velocity within [-1, 1] range but for security reasons we limit the values that the drone can send to a lower velocity value.
After converting the position error values in velocity, a speed limiter has been added to limit the input values to the PID controller to the range [-1,1].
In addition, to limit the velocity commands to the drone, the outputs of the PID controllers have been further decreased by dividing for a constant.

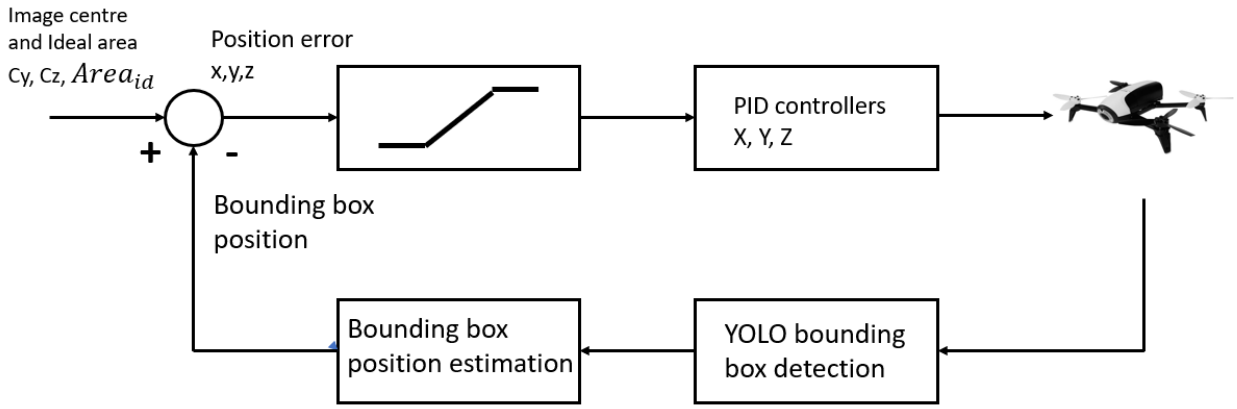


Figure 5.6: Block diagram of the Image-Based Visual Servoing system

The algorithm sends the outputs of the PID controllers to the drone by publishing a ROS message of type geometry_msgs/Twist to the topic bebop/cmd_vel.

Table 5.2: PID outputs and drone movements

PID axis	Command type	Controller action
PID axis X	twist.linear.x	Forward / Backward movement
PID axis Y	twist.linear.y	Left / Right movement
PID axis Y	twist.angular.z	Left / Right rotation around Z
PID axis Z	twist.linear.z =	Up / Down movement

Twist.msg is a ROS message that expresses the velocity in the free space broken into its linear and angular parts.
In our case we have:

When in the image plane of the drone camera there are more people, the algorithm will choose the most centered person bounding box as the person to follow and then it will save its bounding box center coordinates in the current frame.
In the next frame the algorithm will compare the stored bounding box center of the person in the previous frame with all the person bounding boxes in the new current frame and it will choose as the person to follow the one that has the smallest distance with respect to the person followed in the previous frame.

By using SORT tracker with YOLO detection framework, a unique ID can be assigned to each person in the image plane and with a modification to the already implemented algorithm the drone follows the most centered person and saves its unique ID in order to compare it with the other IDs in the next frames.
When an ID switch occurs and the drone loses the tracked person ID, the algorithm will choose the most centered person and it will save its new ID in order to compare it in the next frames.

The architecture of the IBVS system is the following:

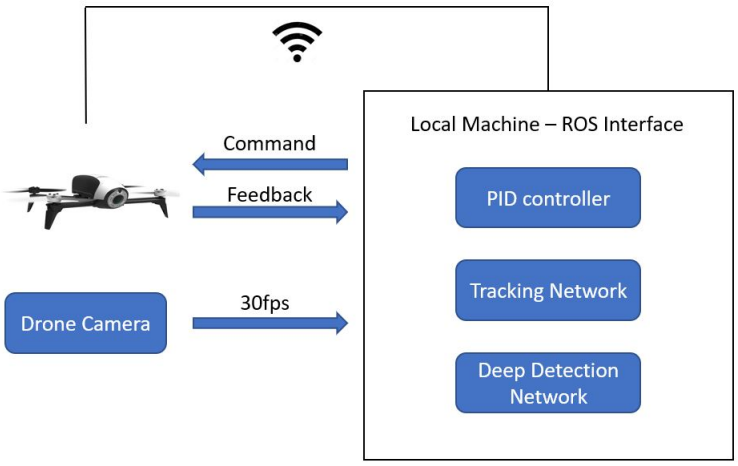


Figure 5.7: Architecture of the IBVS system

5.2 PID controller

A proportional-integral-derivative controller is a control loop feedback mechanism that calculates an error value and applies a correction based on proportional, integral and derivative terms.

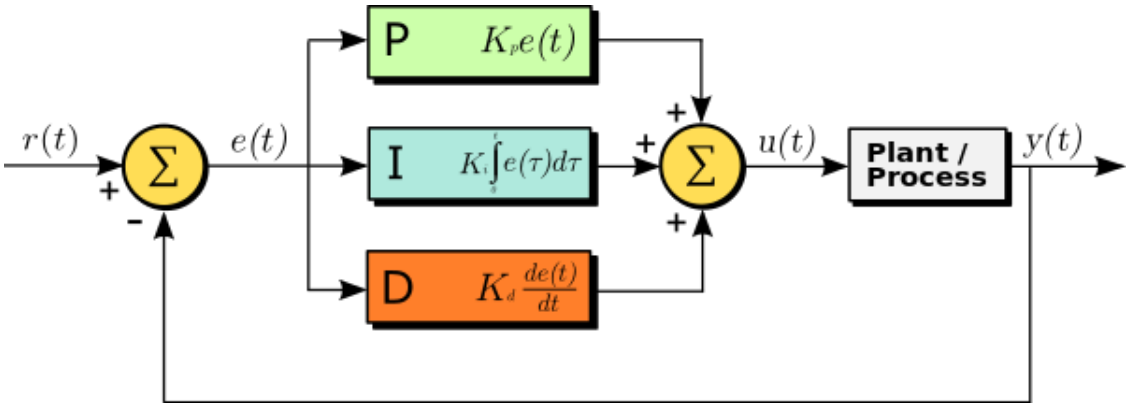


Figure 5.8: Block diagram of a PID controller. From [25]

The error is calculated as the difference between a desired setpoint and a measured process variable. In our case, the measured process variable is the detected bounding box position.

$$e(t) = r(t) - y(t) \tag{5.7}$$

The proportional term P is directly dependent on its input error e(t) multiplied by the proportional gain constant Kp. This term will move the system toward the desired setpoint with an intensity dependent on the value of the error and of the constant Kp.

$$P = K_p \; e(t) \tag{5.8}$$

The integral term I is the sum of all past errors integrated over time and multiplied by the integral gain constant Ki. If the error is not zero, the integral term will continue to grow over time until the error is eliminated.

$$I = K_i \int_0^t e(t)\,\mathrm{d}t \tag{5.9}$$

The derivative term D is the magnitude of the error change (current error value compared to the previous error value) multiplied by the derivative gain constant Kd. This term is an estimate of the future trend of the error based on its current rate of change. The more rapid the change, the greater will be the derivative term.

$$D = K_d \frac{\mathrm{d}e(t)}{\mathrm{d}t} \tag{5.10}$$

The overall control function can be expressed mathematically as:

$$u(t) = K_p \; e(t) + K_i \int_0^t e(t)\,\mathrm{d}t + K_d \frac{\mathrm{d}e(t)}{\mathrm{d}t} \tag{5.11}$$

The PIDs for the IBVS system are the following:

Table 5.3: PID gain values

PID	K_p **proportional**	K_i **integral**	K_d **derivative**
X axis	0.5	0.05	1.5
Y axis	0.4	0.3	0.8
Z axis	0.4	0.3	0.8

- For forward / backward movement (X axis) :

$$Vout_x = K_{px}\ V_x(t) + K_{ix} \int_0^t V_x(t)\,\mathrm{d}t + K_{dx}\,\frac{\mathrm{d}V_x(t)}{\mathrm{d}t} \tag{5.12}$$

- For up / down movement (Z axis) :

$$Vout_z = K_{pz}\ V_z(t) + K_{iz} \int_0^t V_z(t)\,\mathrm{d}t + K_{dz}\,\frac{\mathrm{d}V_z(t)}{\mathrm{d}t} \tag{5.13}$$

- For left / right movement (Y axis) :

$$Vout_y = K_{py}\ V_y(t) + K_{iy} \int_0^t V_y(t)\,\mathrm{d}t + K_{dy}\,\frac{\mathrm{d}V_y(t)}{\mathrm{d}t} \tag{5.14}$$

This is the complete controller scheme of the IBVS system:

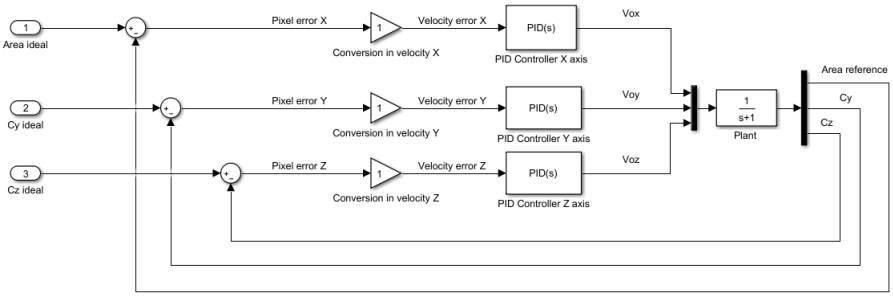


Figure 5.9: Complete block diagram of all implemented PIDs

To produce the optimal controller output we have to tune the three parameters K_p, K_i and K_d so each part has an appropriate impact on the output.

5.2.1 PID tuning

There are different strategies that can be used to tune the parameters of a PID controller. A simple and fast approach is the Ziegler-Nichols method.

In this method, all the gains K_p, K_i, K_d are set initially to zero and then K_p is incrementally increased until it reaches a value at which the output of the controller has stable and consistent oscillations.
This value is called ultimate gain Ku and it is used to set the values of the PID gains.

Chapter 6

Results

6.1 HUI system

The HUI system is responsible for interpreting the operator gestures and for sending different flight commands to the drone based on the gestures.

The position of the operator from the ground station laptop, which is positioned on the ground, is approximately 1 meter.

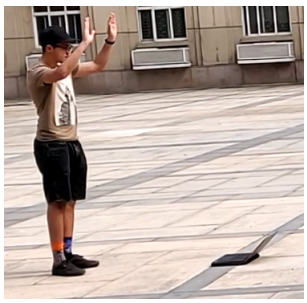


Figure 6.1: Distance between operator and laptop

This distance has been chosen by doing experiments with the different distances between the operator and the ground station in which the accuracy of the deep neural network detector in detecting correctly the 2 hands and the face has been evaluated.

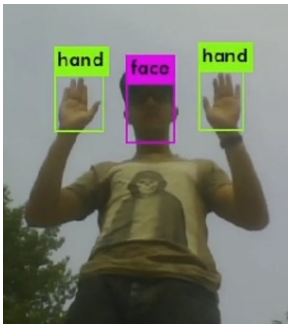


Figure 6.2: Detection of hands and face of the drone operator

The following table relates the detector accuracy with the distance between drone and operator:

Table 6.1: Detector accuracy after training on hands and face dataset

Distance (m)	**Right hand accuracy**	**Face accuracy**	**Left hand accuracy**
0.3 m	99%	99%	99%
0.6 m	99%	99%	99%
1 m	98%	99%	97%
1.3 m	94%	92%	92%
1.6 m	89%	90%	90%
2 m	83%	85%	81%

A total of 10 different gestures representing 10 different flight commands has been implemented on the HUI system. Other gestures could have been added to broaden the drone capabilities: for example a gesture for taking an on-board snapshot or for starting a video recording.

After the HUI system is activated, the first command to be sent is take off which consists of hands all aligned at face level. After taking off, the drone will hover at approximately 1 meter from the ground until the next command is received.
When there is no input command from the operator, the drone will keep hovering until the next command is received.

The command to take off is the following:
rostopic pub –once bebop/takeoff std_msgs/Empty

The landing command which consists of hands aligned beneath the face can be sent at any drone altitude and it will land safely the drone.
The command for landing is the following:
rostopic pub –once bebop/land std_msgs/Empty

Both the landing and take off commands don't require a velocity input as the velocity for ascending and descending during take off and landing is handled by the drone firmware. For the remaining commands a velocity input is required.

The velocity of the commands has been chosen by doing experiments and considering the reaction time of a person in order to not have drone movements that are too rapid: the final velocity chosen is 0.3m/s for each command.

The distance between the drone and the ground station does not have strict limits as the WiFi range of the connection can reach more than 100 meters. The range depends on the power of the WiFi generated by the device and this power depends on several parameters (device type, physical environment, selected channel and so on).
For the tests done with the HUI system, less than 100 meters between the drone and the ground station are satisfactory.

The accuracy of the neural network detector has been evaluated under different

lighting conditions and environments (i.e indoor and outdoor): the result is that the neural network is robust under different lighting conditions and environments.

Overall, considering the accuracy of the neural network detector trained on hands and face dataset and the stable drone response to the commands sent by the operator via gestures, the HUI system is suitable for its application.

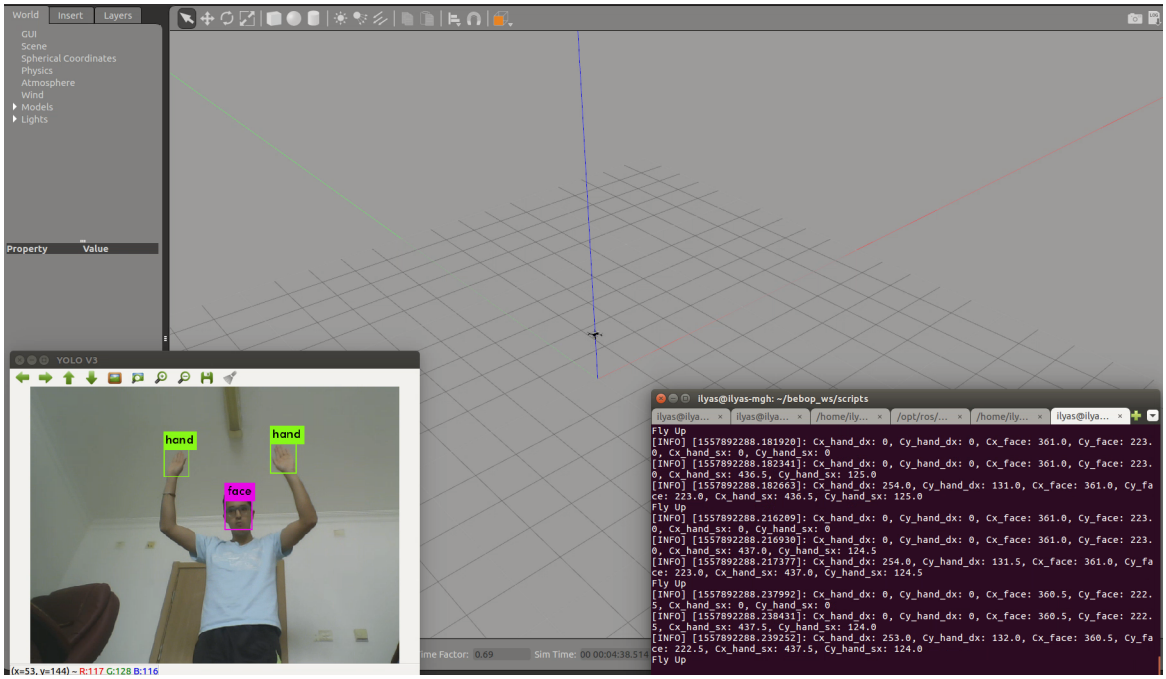


Figure 6.3: Simulated HUI system in action.

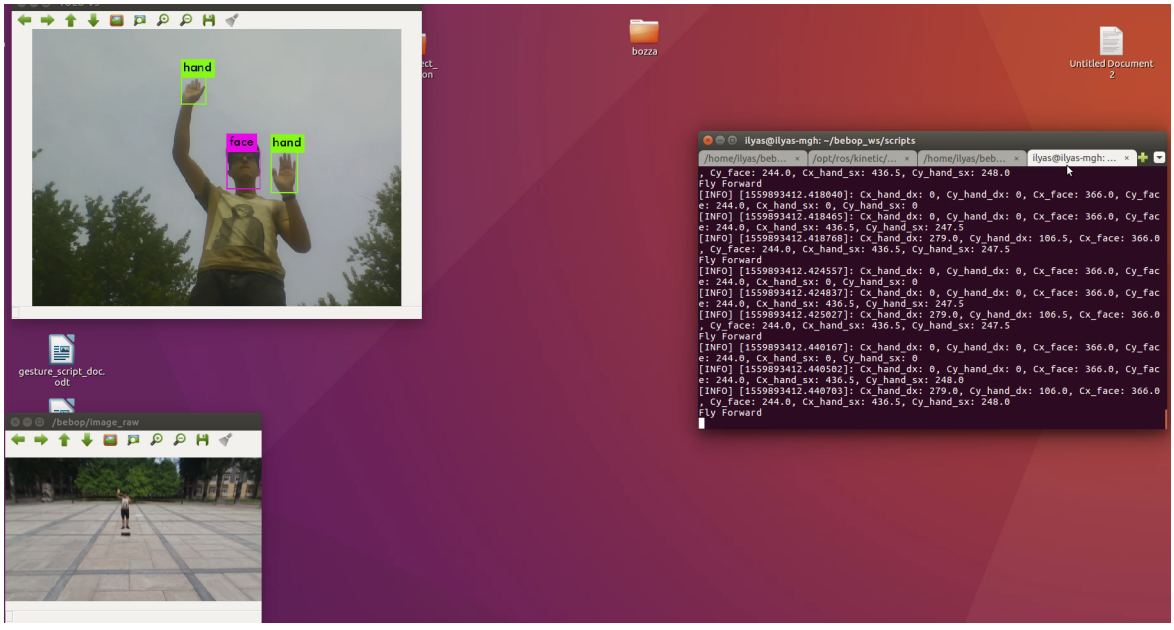


Figure 6.4: Real HUI system in action.

Figure 6.5: HUI system in action.

6.2 IBVS system

The IBVS system is responsible for storing the bounding box of the most centred person in the field of view and to follow the same stored person even when there are more people in the field of view.

To estimate the person distance from the drone we assume that detected objects are rigid, this means that a change in the object size is due to a change of its distance from the drone.

The IBVS system uses a PID controller to send velocity commands to the drone based on its distance from the detected person in order to keep the same predefined distance between drone and detected person that is based on the ideal bounding box area.

The ideal area has been selected by doing experiments with different bounding box areas in which the distance between drone and person has been measured.

The following table shows the results of the experiment:

Table 6.2: Ideal bounding box selection

Ideal BB area %	**Area pixel equivalent**	**Drone and target distance**
10%	41088	2m
20%	82176	1.5m
30%	123264	1m
40%	164350	0.6m
50%	205440	0.2m

Considering a trade-off between safety of the person and close distance between drone and person, an ideal area of 30% has been chosen.

In the IBVS system, the camera utilized and its resolution is different from the HUI system and also the detected object is different: for this reason the accuracy of the detection has been evaluated with respect to the distance between drone and target.

The reason for not wanting a distant distance between drone and person is that the resolution of the drone camera is not very high in order to help process quickly the images received by the neural network detector system. For this reason, when the drone is distant from the person the neural network detector system starts to fail to detect some people.

The following table shows the correlation between distance and detection accuracy measured with the IBVS system:

Table 6.3: Distance and detection accuracy correlation

Drone and target distance	**Area pixel equivalent**	**Detection accuracy**
1m	123264	99%
2m	106573	95%
3m	95678	89%
4m	84738	83%
5m	71829	77%
6m	62938	69%
over 7m	49384	less than 60%

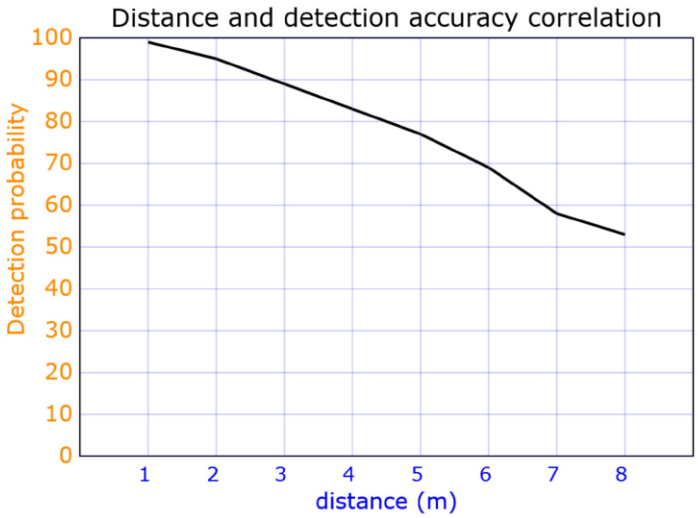


Figure 6.6: Distance and detection accuracy correlation graph

If the distance between drone and target is greater than 7m, the drone will stop following the target as it is not able to detect it due to the far distance.

To choose the ideal bounding box area, also the detection accuracy variation with the distance has been taken into consideration.

The computation and image processing is done on the ground station laptop which is positioned on the ground similar to the HUI system. The difference is that both the drone and the person will be moving away from the ground station. For this reason the movements had been limited to a distance of 200 meters from the ground station in order to keep a good WiFi signal between the drone and the laptop.

Figure 6.7: IBVS system: processing on laptop and commands sent to drone

The IBVS system has 3 PID controllers, one for each movement type (forward/backward, left/right, up/down).

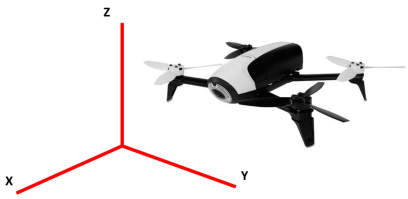


Figure 6.8: Drone orientation and axis

In addition to the ideal bounding box area as a goal parameter for the forward/backward movement PID controller, the goal parameter for the other 2 PID controllers is the centre of the image (428,240): 428 pixel for the left/right movement PID controller and 240 pixel for the up/down movement PID controller.

The Y axis PID controller, in addition to the linear velocity, also sends an angular velocity around Z axis.

The velocity sent by the X axis PID controller depends on the detected person distance from the drone (i.e bounding box area) while the velocity sent by the Y axis and Z axis PID controllers depend respectively on horizontal center Cy and vertical center Cz of the person bounding box.

The following tables represent the velocity values compared with the respective goal parameters:

Table 6.4: X axis PID controller table

Detection area $pixel^2$	**Velocity m/s**	**Controller action**
79746	0.61	Forward
97483	0.41	Forward
109374	0.24	Forward
123264	0	
135463	-0.21	Backward
144344	-0.38	Backward
154647	-0.49	Backward

A bounding box area bigger than the ideal area (30%) means that the person is close to the drone, so the drone has to move backward. If the bounding box area is smaller than the ideal area, it means that the person is far from the drone, thus the drone has to move forward.
When the drone is approaching the person, one method to stop or move away the drone is to open the arms to rapidly increase the bouding box area. In this way, the drone will move backward because it assumes that the person is very near.

Table 6.5: Y axis PID controller table

Cy	**Linear velocity m/s**	**Angular velocity rad/s**	**Controller action**
213	0.51	0.16	Left
326	0.24	0.07	Left
428	0	0	
530	-0.24	-0.07	Right
643	-0.51	-0.16	Right

If Cy is greater than 428, the detected person is on the right compared to the goal parameter so the drone will move to the right to realign the target with the center of the image. If it is minor than 428, the detected person is on the left so the drone will move to the left.

In addition to have a lateral linear movement, the drone also has an angular movement that allows it to turn in the direction of the followed person. The drawback is that the angular velocity is not high and it could happen that the target movement away from the field of view is rapid. In this case, the drone loses its detected person and it will hover until it detects a new person to follow. A solution for this problem could be to implement an angular velocity when there is no detection until the drone

Table 6.6: Z axis PID controller table

Cz	Velocity m/s	Controller action
82	-0.16	Down
156	-0.07	Down
240	0	
324	0.07	Up
398	0.16	Up

finds a new person to follow. If Cz is greater than 240, the detected person is on the upper side compared to the goal parameter so the drone will move up to realign the target with the center of the image. If it is minor than 240, the detected person is on the lower side so the drone will move down.

When the drone takes off, it hovers at approximately 1 meter from the ground and then when the IBVS system is activated it will adjust its height to have the bounding box of the person centered in the field of view.

The 3 PID controllers operate at the same time, for example if the detected person is distant and on the right side compared to the goal parameters the drone will move forward, right and rotate clockwise to align itself with the detected person.

The accuracy of the neural network detector has been evaluated under different lighting conditions and the result of the evaluation is that the neural network is robust enough to detect the target under different conditions.

Using only the neural network detector system the drone is not able to follow the same detected person when new people enter the drone field of view because the detector system does not identify and differentiate every single detection in the field of view. This means that there is no way for the detector system to identity and follow the same person.

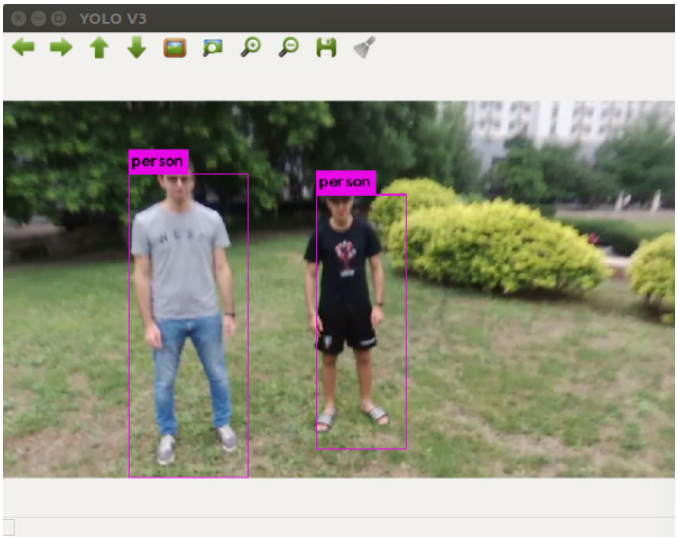


Figure 6.9: Detection of more than one person from drone camera

The solution is to add a tracking system to identify and track each detected object in each image frame.

Two different solutions have been implemented to solve this issue:

- **Deep SORT tracking system**

  With the tracking system, the IBVS system registers the ID of the most centred bounding box and it follows it until it goes out of the field view. In case of tracked object out of the field of view, the IBVS system will start tracking the new most centred person.
  The following table shows the tracking accuracy using the deep SORT tracker:

Table 6.7: deep SORT tracking accuracy

Drone and target distance	**Detection accuracy**	**Tracking accuracy**
1m	99	99%
2m	95	95%
3m	89	88%
4m	83	84%
5m	77	77%
6m	69	67%
over 7m	less than 60	less than 60%

  The tracking accuracy is similar to the detection accuracy because the tracker relies heavily on the detector system.

  The drawback of SORT is that it has a high number of identity switches due to missed detections by YOLO detector.

  The missed detections by the deep neural network detector cause the tracker to reinitialize the same object with a new ID when the same object is detected again. For example if an identity switch happens to a tracked object that is still in the field of view but is not the most centred object, after the tracker reinitializes the object with a new ID it will select the most centred person that is not the tracker object in the previous frame as its old ID is not present in the field of view anymore due to the identity switch.

- **Tracking system implemented directly in the IBVS system algorithm.**

  In the first frame, the algorithm will choose the most centred person and then in the next frames it will compare the stored bounding box of the most centred person with all the bounding boxes present in the field of view of the new frame (including the same detected person, which now is in a slightly different position) and it will choose as the person to follow the one that has

the smallest distance with respect to the stored bounding box in the previous frame.

This solution requires less computational power because the tracking system is integrated with the IBVS algorithm, while the SORT tracking system requires a separate system that is running in parallel with the neural network detector system.
As only the detector system is running with the IBVS algorithm, the processing is faster than with the SORT system.

A drawback of the IBVS system is that the algorithm does not handle the take off and land commands, this means that the operator has to manually send the take off command to the drone and then activate the tracking algorithm. When the operator wants to stop the algorithm and land the drone, he has to manually stop the algorithm and then send the landing command from the laptop.

Considering the accuracy of the neural network detector trained on pedestrians and the reliability of the tracking system in following the same person, the IBVS system is suitable for its application as it is able to track a detected target with precision.

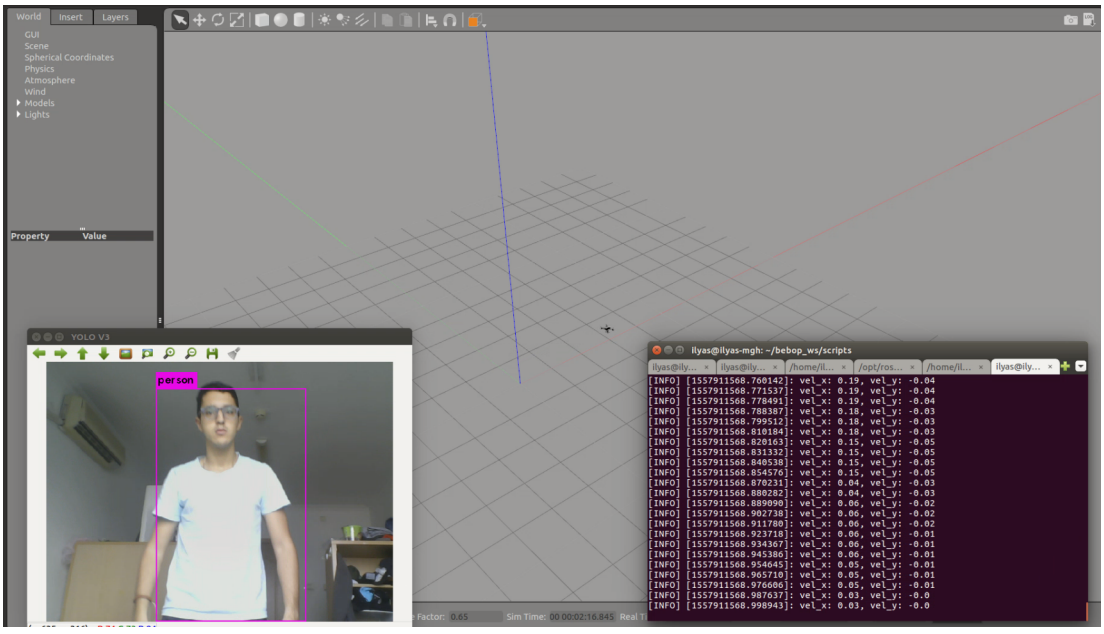


Figure 6.10: Simulated IBVS system in action.

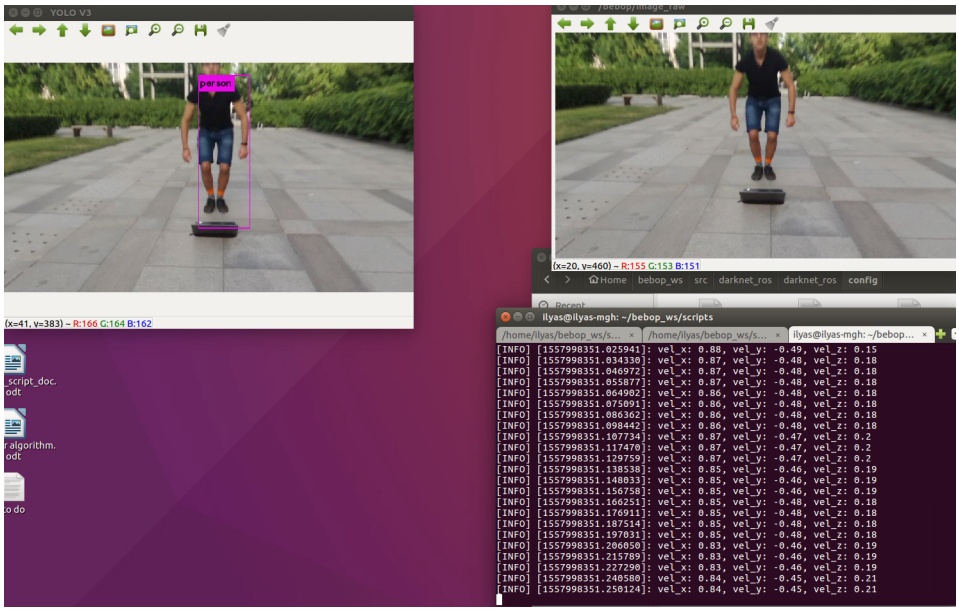


Figure 6.11: Real IBVS system in action.

Figure 6.12: IBVS tracking system in action.

Conclusions

This study is devoted to the design and implementation of a Human-UAV interaction system and an Image-Based Visual Servoing system based on deep learning.

In the HUI system, thanks to the personalized dataset used in the deep learning detection framework, a high gesture detection accuracy has been achieved.
As a result of the high accuracy and of the deep learning model deployed in the ground station equipped with GPU, the system is able to function in real-time.

In the IBVS system, thanks to the combination of detection framework, tracking framework and PID control methodology, the system is able to function in real-time and to follow the same person when the image plane has more people thanks to the assignment of unique IDs.
As the IBVS system relies on the ground station and the drone is connected to the ground station via WiFi, when the drone following the tracked person is too far away from the ground station it will lose the connection with the ground station and the ground station will not be able to further send the command to the drone to keep following the detected person.

For this reason, a further development of this thesis work could be to implement the IBVS components (i.e deep learning detector, tracking system, controlling algorithm) in an embedded system like the NVIDIA Jetson TX2 module or Jetson Nano module attached to the drone frame: that is adding edge computing capabilities to the drone in order to analyse the data coming from the camera close to its source to improve response times and save bandwidth.
In this way, there is no need for a ground station laptop and the computation can be done "on the spot" , the drone is not limited in movement due to WiFi range between drone and ground station and the whole performance and user experience is improved by reducing the latency.

Another further improvement would be to train a dataset able to detect persons, hands and faces in order to combine the HUI system with the IBVS system.
In this way, the problem of the IBVS system not handling take off and landing of the drone is solved.

Acknowledgements

This thesis has been developed at Beihang University at the School of Automation Science and Electrical Engineering during the academic year 2018/2019 under the supervision of prof. Baochang Zhang.

I would like to express my gratitude to Prof. Baochang Zhang for his guidance throughout the work and a special thanks to Ali Maher for his advices and suggestions throughout the work.

I would like to express my gratitude to prof. Marcello Chiaberge for his help in writing this work.

Bibliography

Scientific Papers

[1] J. S. D. R. G. A. F. Redmon, "(YOLO) You Only Look Once," in CVPR proceedings, 2016.

[2] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, 2017.

[3] J. Redmon and A. Farhadi, "YOLOv3," in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[4] R. Bartak, A. Hrasko, and D. Obdrzalek, "A controller for autonomous landing of AR.Drone," in 26th Chinese Control and Decision Conference, CCDC 2014, 2014.

[5] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in Proceedings - International Conference on Image Processing, ICIP, 2016.

[6] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in Proceedings - International Conference on Image Processing, ICIP, 2018.

[7] M. R E Kalman (Reserach Institute for Advanced Study, Baltimore, "A New Approach to Linear Filtering and Prediciton Problems," J. Basic Eng., 1960.

[8] H. W. Kuhn, "The Hungarian method for the assignment problem," in 50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art, 2010.

[9] A. Maher, H. Taha, and B. Zhang, "Realtime multi-aircraft tracking in aerial scene with deep orientation network," in Journal of Real-Time Image Processing, 2018.

[10] A. Maher, C. Li, H. Hu, and B. Zhang, "Realtime Human-UAV Interaction Using Deep Learning," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2017.

[11] R. Bartak, A. Hrasko, and D. Obdrzalek, "A controller for autonomous landing of AR.Drone," in 26th Chinese Control and Decision Conference, CCDC 2014, 2014.

Websites

[12] *ROS Wiki.*
http://wiki.ros.org/Documentation
[13] *Parrot-Sphinx Simulation Tool.*
https://developer.parrot.com/docs/sphinx/whatissphinx.html
[14] *YOLO.*
https://pjreddie.com/yolo/
[15] *bebop_autonomy Documentation.*
https://bebop-autonomy.readthedocs.io/en/latest/
[16] *ROS website.*
http://www.ros.org/
[17] *YOLOv3 custom detection training.*
https://medium.com/@manivannan_data/how-to-train-yolov3-to-detect-custom-objects-ccbcafeb13d2
[18] *Neural Networks and Deep Learning.*
http://neuralnetworksanddeeplearning.com/
[19] *Gazebo.*
http://gazebosim.org
[20] *Parrot drone website.*
https://www.parrot.com
[21] *Parrot for developers.*
https://developer.parrot.com/
[22] *NVIDIA CUDA .*
https://developer.nvidia.com/cuda-zone
[23] *NVIDIA cuDNN .*
https://developer.nvidia.com/cudnn
[24] *darknet_ros repository .*
https://githubcom/leggedrobotics/darknet_ros
[25] *PID controller .*
https://en.wikipedia.org/wiki/PID_controller
[26] *Parrot website .*
https://www.parrot.com/us/drones/parrot-bebop-2
[27] *Convolutional Neural Network .*
https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html
[28] *Gradient descent .*
https://en.wikipedia.org/wiki/Gradient_descent
[29] *Quadcopter .*
https://en.wikipedia.org/wiki/Quadcopter