Design and test of a multi-agent robotic system for inspection: coordination and integration of a drone and a robotic rover

By

Andrea Galeone

Supervisor

Alessandro Rizzo

Co-supervisor

Carlo Canali

---

degree of *Mechatronic engineering*

Department of CONTROL AND COMPUTER ENGINEERING (DAUIN)

School of Mechanical and Manufacturing Engineering (SMME)

Politecnico di Torino

Torino, Italy

December 2019

This thesis is dedicated to *my beloved parents and friends*

Abstract

The purpose of this project is the development of a multi-vehicle system capable of performing concurrently aerial and ground inspections. The possibility of having two vehicles inspecting from two different point of views, one from the ground the other from above, guarantees a more accurate inspection in places that are not easily accessible to humans, for example, limited spaces or too elevated areas. Another advantage is the dynamic inspection which, compared to fixed cameras, allows the elimination of the dark areas of the latter. The types of inspection for which this system has been designed are search and rescue and industrial inspection. During search and rescue operations, for example, it is important to minimize potential dangers that can threaten both the life of rescue teams and people to be rescued (for example in case of unstable soils or exposed electric cables). In industrial inspections, on the other hand, it can be useful as a surveillance system in large areas, for example, container areas, in large warehouses as product control or to inspect dangerous areas.

Figure 1: The multi-vehicle system

The two tasks that have been developed, during thesis work, namely: the autonomous flying capabilities of a drone and the development of a research rover able to be transported on by the drone through a gripping system. The development of a predefined path that the drone will follow for the inspection flight, allows the search and, in second time, the precision landing on a visual marker. Within this project, the visual marker is positioned above the rover so that the drone can pick up the rover and bring it back. DJI Matrice 100 quadcopter has been used as development platform, along with a single-board computer, which acquires, processes and exchanges data between drone and rover. The rover has been designed and integrated through a WiFi Development Board that receives data from the main computer and sends the commands to the motors and camera installed inside. To build and run codes across the multiple computers used in this project, ROS (Robot Operating System) has been used to provide tools and libraries.

The goal of this thesis is to design a system composed by an autonomous drone that interacts with a robotic rover, and perform a "proof-of-concept" of the system in laboratory environment.

This thesis represents the starting point of a longer term research plan: the work described in this thesis will therefore pave the way to future developments that are outside the scope of this project.

The work done to accomplish this thesis can be briefly listed as follow:

- An unmanned aereal vehicle capable of performing a predefined path has been programmed;
- A vision algorithm that identify a visual marker positioned on the drone has been implemented;
- A rover capable of being both autonomous or teleoperated has been designed, built and tested ;
- A grasping system, mounted on the drone, able to grab the rover has been designed, built and tested ;
- A communication system used to integrate the drone, the rover and a remote ground station has been implemented.

In the following, details about all the all above mentioned tasks will be given.

The system has been first tested in simulation through the official application of the drone and GAZEBO (an indoor and outdoor simulation environment). Later, experimental tests have been performed on each task of the project, and then concluded with a complete simulation of the entire mission.

Contents

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

UAV	Unmanned Aerial Vehicle
ROS	Robot operating system
M100	DJI Matrice100
tf	transfer function

Chapter 1

Introduction

1.1 Background

Since the invention of the first robot in 1739, an android capable of playing the flute, the development of robots has increased. They are often used to replace men, for example in an industrial context, to reduce production times and increase the quality of a product. Recently, thanks to the small size and greater computing power, it became possible to build robots capable of performing simple tasks and operating in different areas, like aerial and ground places. Furthermore, being able to use communication protocols which permits the exchange of big amount of data in real time has allowed the development of robots capable of communicating with each other. This results useful in order to have multi-agent systems that work together to improve yield.

In particular, these robots are widely used in the industrial field, to inspect the quality of the produced parts or for batch monitoring. They are also used in the search and rescue field.

Taking into account rover and drones (named UAV), they provide numerous benefits for inspection or disaster response. These kind of robots can be sent into places that are inaccessible or too dangerous for human workers, for example underground mining. This kind of enviroment poses numerous problems such as ground movement (fall of roof/sides), inundation, air blast, etc. In the industrial field, instead, they are useful in very high areas, such as cranes or container zone, or very restricted areas.

The possibility of making these vehicles autonomous decreases the possibility of human error during the piloting phase and allows workers to operate in safe areas.

1.2 Aim of this Thesis

The aim of this thesis is the development of a multi-agent system, consisting of a rover and a drone, able to carry out a research mission in a hostile or difficult to reach environment. The possibility of operating both in the air and on the ground guarantees a complete understanding of the site to be explored.

The scenario that has been considered during this work is a situation where the drone carry on the rover, deposit it on a point of interest, coordinate it from above and then pick it up and bring it back to the home point: the drone can explore an area and, if necessary land on a specific position to release a robotic rover used to inspect areas that are not accessible from a sky view. The rover can receive specific instruction from the drone (benefiting from information received from the aerial view) or from a remote human operator. Once the mission of the rover is accomplished, the drone is able to localize again the rover, pick it up and return it to the home point.

In particular, the drone used for this work is a DJI Matrice 100. It is controlled by an onboard PC, that controls the drone on a definite path, meanwhile a camera takes a sequence of pictures to find a visual marker that is placed upon the rover.

The defined path has been realized according to the characteristics of the camera. It guarantees a total coverage of the area to be explored, avoiding dead zones caused by the focal width of the latter.

During the rover research phase, it can be controlled by the operator through an APP, which it sends the movement signal to a Wi-Fi board that syncronyze the four motors of the robot and inspect the area with the camera connected to the robot.

Thanks to a vision algorithm and GPS installed on the UAV, the drone combine local position with marker position and it makes a precision landing on the rover when the inspection mission is completed.

During the flight phases, the use of the GPS and visual signal filtering system guarantees great precision both during the research phase and the landing phase.

The rover acquires the data of the drone camera and makes some movements to position itself under the UAV to increase the landing accuracy of the drone.

To complete the mission, the drone has the task of hooking the rover through a gripping

system and bringing it back to the operator.

The development of this project can be divided into three macro-topics:

1. Software development of the onboard PC to implement the autonomous flight of the DJI Matrice100;
2. Development of the mechanical architecture of a grab system able to grasp a rover;
3. Software, hardware and mechanical architecture development of a mini inspection robot.

The figure below shows the complete scheme of the principal components used during the project, in particular the connections between various components for data exchange have been highlighted.

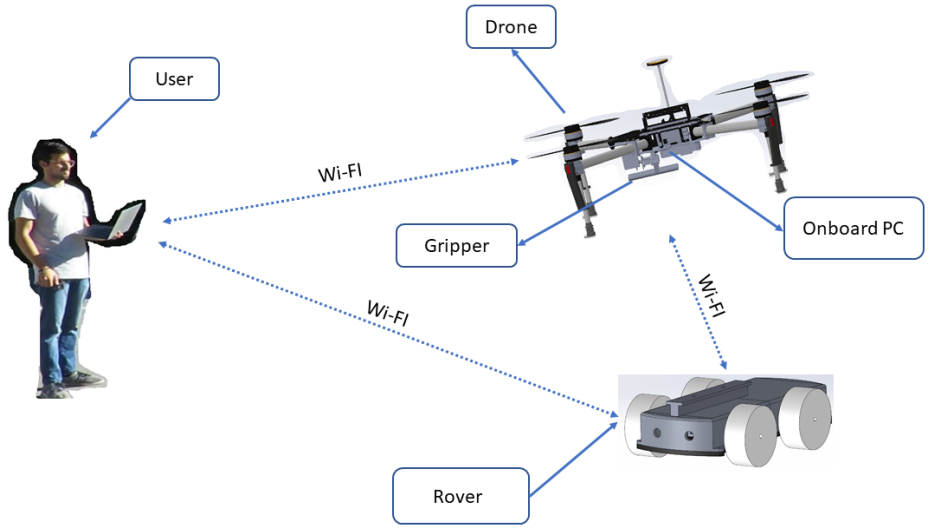

Figure 1.1: Project scheme

This thesis extensively describes the developed functions, implementation of the features mentioned above and the mechanical architectures of the components created.

Chapter 2

State of the art

In this chapter, a list of works related to the topic of this thesis is described. The goal of the thesis is the design of an autonomous drone that will have to recognize and transport an inspection rover. Four macro-topics will then be explored: autonomous drones, vision systems for object recognition, aerial vehicles with grasping systems and inspection rovers.

The main sites that have been used for this research are github, an American company that provides hosting for software development version control using Git, and Google Scholar, a freely accessible web search engine that indexes the complete text or metadata of the academic literature through a series of formats and publishing disciplines. In all three sections the Python [1], ROS [2] and OpenCV [3] sites were useful for the development of various features.

2.1 Unmanned Aerial Vehicles

An unmanned aerial vehicle (UAV), commonly known as drone, is an aircraft without a human pilot on board. UAVs are a component of an unmanned aircraft system (UAS); which include a UAV, a ground-based controller, and a system of communications between the two. The flight of UAVs may operate with various degrees of autonomy: either under remote control by a human operator or autonomously by onboard computers. The drones manufactured these days are becoming smarter by integrating open source technology, smart sensors and more flight time.

The article by Lian Pin Koh and Serge A. Wich, entitled "Dawn of Drone Ecology:

Low-Cost Autonomous Aerial Vehicles for Conservation"[4] is an example of UAVs used for inspections. The drone is able to fly pre-programmed missions autonomously for a total flight time of 25 minutes and over a distance of 15 km.

Figure 2.1: The prototype of the Conservation Drone used in test missions

The autopilot system of the Conservation Drone is based on the 'ArduPilot Mega' (APM), which has been developed by an online community (diydrones.com). The APM includes a computer processor, geographic positioning system (GPS), data logger, pressure and temperature sensor, airspeed sensor, triple-axis gyro, and accelerometer. By combining the APM with an open-source mission planner software (APM Planner), most remote control model airplanes could be converted to an autonomous drone.

A thesis useful for understanding the development of autonomous flight is "Design and Implementation of a Development Platform for Indoor Quadrotor Flight Control "[5], written by Jose Libardo Navia Vela. This thesis project follows up this interest aiming to build a real-time development platform for experimentation and testing of indoor flight controllers Embedded Robotics.

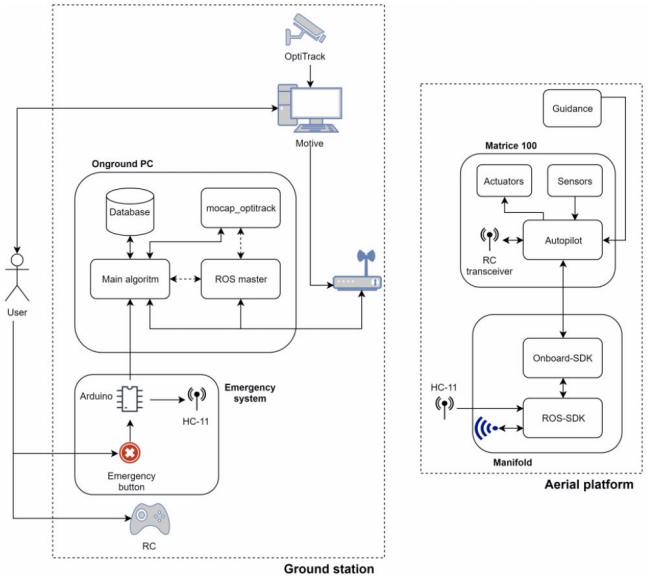

Figure 2.2: The prototype of the Conservation Drone used in test missions

The author of the thesis employed a DJI Matrice 100 quadrotor alongside its ROS C++ Software Development Kit (SDK). He designed and implemented a control topology for real-time position control of the UAV inside the laboratory by means of a Model Predictive Controller (MPC) and a Linear-Quadratic Regulator. The use of external sensor systems does not allow the use of the UAV in outdoor environments.

"Dynamic System Identification, and Control for a cost effective open-source VTOL MAV"[6], written by Inkyu Sa et al., describes dynamic system identification, and full control of a cost-effective vertical take-off and landing (VTOL) multi-rotor micro-aerial vehicle (DJI Matrice 100). The dynamics of the vehicle and autopilot controllers are identified using only a built-in IMU and utilized to design a model predictive controller (MPC). Experimental results for the control performance have been evaluated using a motion capture system while performing hover, step responses, and trajectory following tasks in the presence of external wind disturbances. The figure below illustrates the planned trajectory (red) and the vehicle position (blue) obtained from a motion capture device. The left column is without the wind and the right is in windy condition.

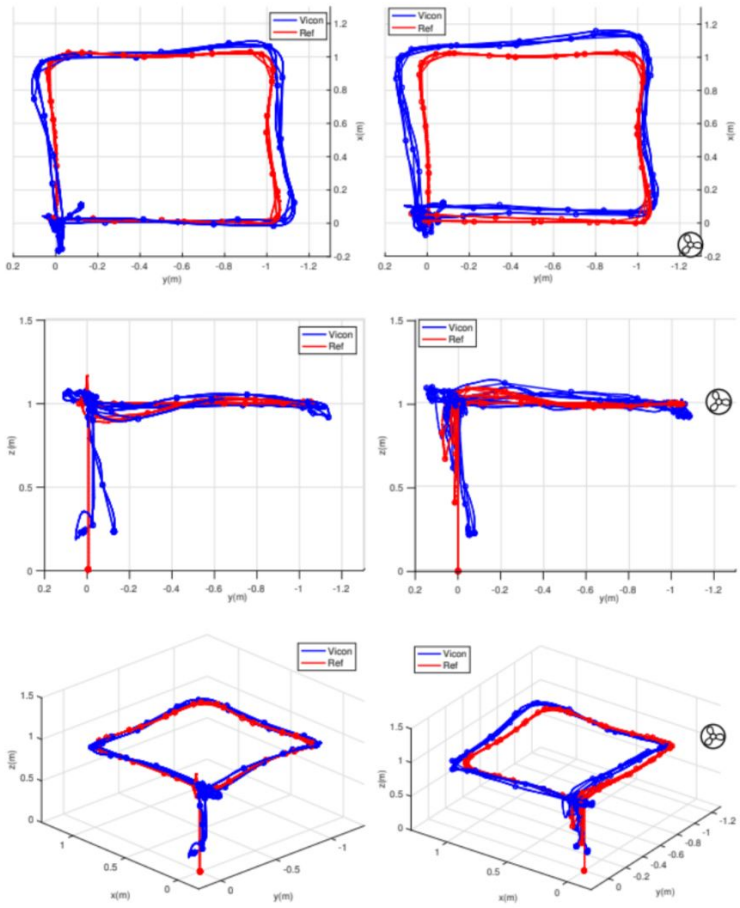

Figure 2.3: 3D pose while trajectory following without (left column) and with (right column) wind disturbances

2.2 Vision system

Machine vision is the technology and method used to provide image-based automatic inspection and analysis for such applications as automatic inspection, process control, and robot guidance.

Combining these three characteristics and adding the data coming from the sensors, it is possible to develop a system able to recognize the objects and obstacles and estimate the distances from them.

An example is given in the article "Vision-based autonomous navigation approach for unmanned aerial vehicle transmission-line inspection", written by Xiaolong Hui et al. [7]. This article presents an autonomous navigation approach based on an aerial vehicle (UAV) for power line inspection. He designed a perspective navigation model, which

enhances the capability of the perception of three-dimensional direction and improves the safety of intelligent inspection.

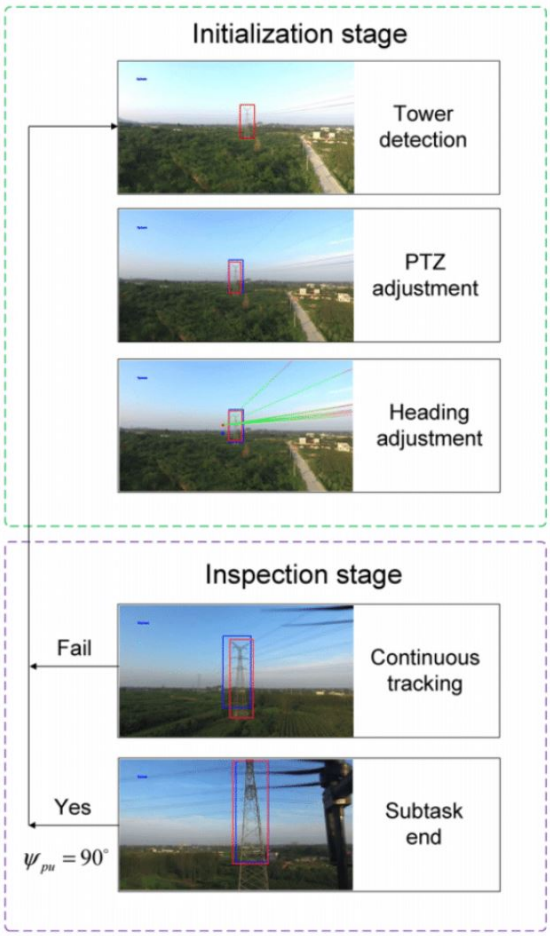

Figure 2.4: Flowchart of autonomous navigation between adjacent towers

A DJI Matrice100 drone has been used for the development of the project. As shown in the figure above, the camera mounted on the drone first identifies the target, makes corrections, orients it in space and then sends the data to the drone moving towards the target to carry out the inspection.

Using such complex vision systems, they need quite powerful and fast PCs for such types of processing. If there is no need to identify the object but only to understand its position and orientation, it is possible to use lighter vision algorithms such as visual markers.

The work done by Alvaro Fernandez Cobo (Approach for Autonomous Landing on Moving Platforms based on computer vision [8]), shows the development of a vision system for the recognition of a marker used as a static and dynamic landing base. The use of

these markers guarantees a very high detection precision, on the three coordinates and the three angles, but in the case in which the marker is partially covered the algorithm does not recognize the marker.

The paper written by Xiaoyue Zhao et al., "Detection, Tracking, and Geolocation of Moving Vehicle from UAV Using Monocular Camera "[9], talks about the development of a framework for moving vehicle detecting, tracking and geolocating based on a monocular camera, a GPS receiver and inertial measurement units (IMU) sensors. The project has been developed on a DJI M100 platform on which a monocular camera and a microcomputer Jetson TX1 are added.

The flowchart of the visual tracking and geolocation part is shown below. The initial pixel position of the target vehicle obtained by the vehicle detection algorithm, along with the aerial video and flight motion parameters, is input into both the visual tracking and geolocation algorithm. The pixel position and GPS coordinates of the vehicle will be output as reference data for the flight control algorithm. The position coordinates of the vehicle are sent to the ground control station, and the vehicle is displayed on the satellite map of the tablet.

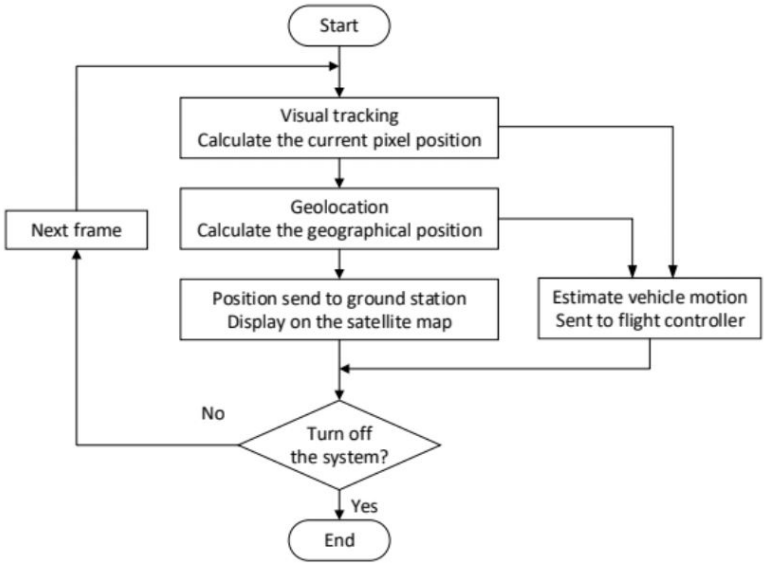

Figure 2.5: Flowchart of the visual tracking and geolocalization algorithm

The experimental results shows that this drone is capable of detecting, tracking and geolocating the interested moving vehicle with high precision. The framework demonstrates its capacity in automatic supervision on target vehicles in real-world experiments,

which suggests its potential applications in urban traffic, logistics, and security.

2.3 Aerial vehicles with grasping systems

UAVs have rapidly evolved into capable mobility platforms able to maneuver, navigate and survey proficiently. A natural progression is to advance beyond simple motion and observation to interaction with objects and the fixed environment. Of specific interest is grasping and retrieving objects while hovering, combining terrestrial robot manipulation capabilities with the range, speed and vertical workspace of flying vehicles. This could make possible novel applications for UAVs, such as search and retrieval in rough or inaccessible terrain or networked aerial logistical supply chains over large areas.

Interesting studies has been carried out by the Grab lab of the University of Yale, and in particular the article written by Paul EI Pounds et al., Grasping From the Air: Hovering Capture and Load Stability [10], where all the mechanical forces, that come into play when an object is picked up and lifted by a UAV, have been shown. It has also been described the types of stabilization used to avoid unfavorable moments of inertia for the drone flight.

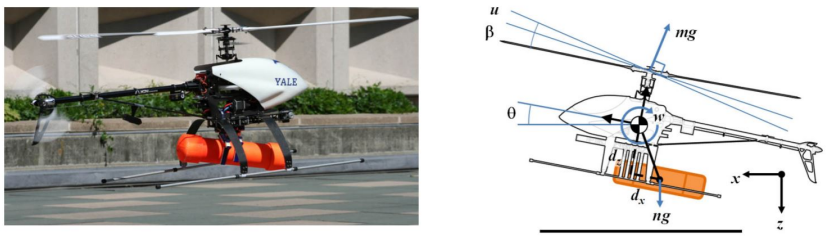

Figure 2.6: Planar aircraft dynamics free body diagram

Adding grasping and manipulation capabilities to unconstrained vehicles such as UAVs, AUVs, and small space craft so that they can deliver cargo, grasp and retrieve objects, perch on features in the environment, and even manipulating their environment is an ongoing area of research. However, these efforts have relied heavily on structuring the interaction task and have predominantly utilized existing gripper designs that were not specialized for the platform or task.

"Design Optimization of a Prismatic-Revolute-Revolute Joint Hand for Grasping from Unconstrained Vehicles"[11], written by Spencer B Backus and Aaron M. Dollar, presents

a parametric model of a novel underactuated hand design, composed of prismatic-revolute-revolute joint fingers. This kinematic configuration attempts to minimize disturbance forces to the body of the vehicle while achieving stable grasps on a wide range of objects under significant positional uncertainty. In particular, this paper investigates the impact of various design parameters, including the relative link lengths and force allocation across the three joints, on grasping performance and suggests optimal design parameters for a prototype hand.

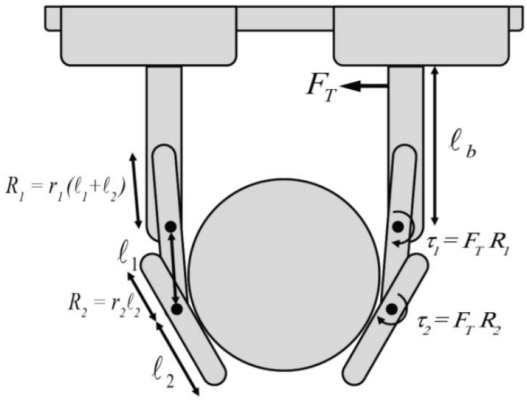

Figure 2.7: Diagram of the hand design grasping a circular object, labeled with important parameters.

2.4 Ground Inspection Robots

Various types of inspection robots were examined in depth. Taking as an example the publication of Love P. Kalra and Jason Gu, "An autonomous self contained wall climbing robot for non-destructive inspection of above-ground storage tanks" [12], they design a wall climbing robot (WCR) for the non-destructive inspection (NDT) of the above-ground storage tanks (ASTs) autonomously making the industrial inspection and maintenance tasks safer.

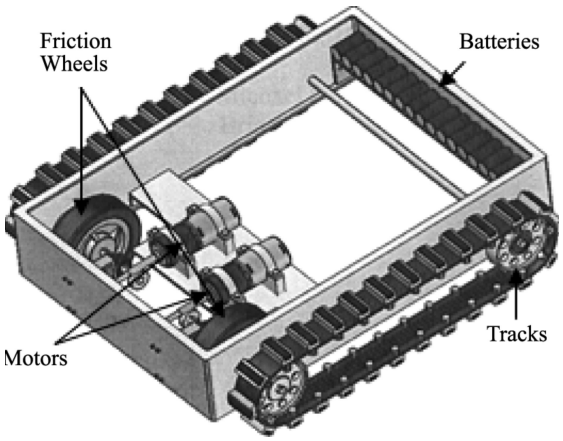

Figure 2.8: Mechanical structure of the inspection robot

2.5 Multi-agent systems

A multi-agent system (MAS) is a computerized system composed of multiple interacting intelligent agents. Multi-agent systems can solve problems that are difficult or impossible for an individual agent or a monolithic system to solve. Intelligence may include methodic, functional, procedural approaches, algorithmic search or reinforcement learning. Typically multi-agent systems research refers to software agents. However, the agents in a multi-agent system could equally well be robots, humans or human teams. A multi-agent system may contain combined human-agent teams.

"Task description, decomposition, and allocation in a distributed autonomous multi-agent robot system"[13] is an article written by T. C. Lueth and T. Laengle. In this paper a new intelligent control architecture for autonomous multi-robot systems has been presented. Furthermore, the paper deals with task description, task distribution, task allocation and coordination of the system components.

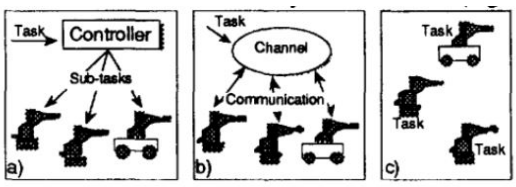

Figure 2.9: Execution view of multi-agent systems: a) centralized, b) distributed, c) decentralized

The main advantage of this control architecture is the distributed execution of tasks or

subtasks by components of the multi-robot system. The components are able to build teams dynamically thereby avoiding the bottle neck problem of the information flow in centralized controlled architectures.

To achieve to distributed organized control architectures, the detailed investigation of communication and cooperation beween components is imperative. The described intelligent control architecture is to replace the former control architecture of the autonomous robot KAMR.

The article we're going to read with is "A Cooperative Multi-Agent System and Its Real Time Application to Robot Soccer"[14], written by J.-H.Kim et al. The soccer robot system consists of multi agents, with highly coordinated operation and movements so as to fulfill specific objectives, even under adverse situation.

The coordination of the multi-agents is associated with a lot of supplementary work in advance. The associated issues are the position correction, prevention of communication congestion, local information sensing in addition to the need for imitating the human-like decision making. In the robot-based soccer robot system, each robot has many functions for autonomous behaviors.

All calculations are done locally in each of the robots. The host computer processes vision data on the position of the ball and robots and forward the same to the robots. Each of the robots decide their own behavior autonomously using the received vision data, its own sensor data and strategies. This can be considered as a distributed control system, where each robot has its own intelligence. It may be very hard to implement, the robot with established for velocity control, position control, obstacle avoidance, communication, decision making, etc. The host computer processes only vision data and can be considered as a kind of sensor.

The use of mechanical systems that perform simple jobs but in places that are not easily accessible to humans is therefore a reason for research that will be studied in depth in this project.

Chapter 3

Hardware, Software and Mechanical architecture

In this chapter the hardware and software structures used for the development of the project have been described. Technical and theoretical data of each developed task will be reported. The CAD drawings will be shown for the created components.

3.1 Hardware description

The project that will be described in this chapter is the development of a multi-agent system consisting of a rover and a drone capable of performing aerial and ground inspections.

Figure 3.1: Drone hardware component

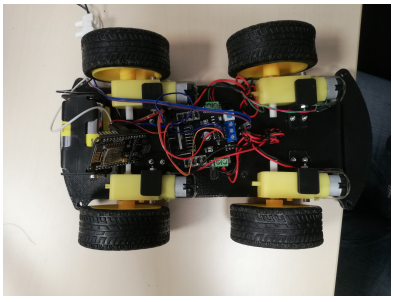

Figure 3.2: Rover hardware component

The pictures above show the hardware used in this project. In the following paragraphs the components will be described in detail.

3.1.1 DJI M100

The DJI M100, called "quadcopter for developers", is a stable, flexible and powerful flying platform intended to be suitable for a wide range of applications in the areas of research, business and recreation.

The center frame is designed to be expandable. That areas can be found in the upper and lower part of the central track structure where it is possible to connect the extra components. In the bottom part, the rasperry pi has been mounted in a 3D printed case.

M100 includes the Standard Propeller, airscrew that converts rotary motion from an engine or other power source into a swirling slipstream which pushes the propeller forwards or backwards. Another component is the electronic speed controller or ESC, that is an electronic circuit with the purpose to vary an electric motor's speed, its direction and possibly also to act as a dynamic brake. It converts DC battery power into 3-phase AC for driving brushless motors. On the drone there are brushless motors which, connected to the four ends of the drone, take care of the propulsion of the latter. The flight controller interprets input from receiver, GPS module, battery monitor, IMU and other onboard sensors. The flight controller is another component that regulates motor speeds, via ESCs, to provide steering, as well as triggering cameras or other payloads. It controls autopilot, waypoints, follow me, failsafe and many other autonomous functions. The flight controller is central to the whole functioning of the UAV. In front of the drone there is a camera mounted on a gimball.

Being mainly manufactured on carbon fiber, M100 minimum takeoff weight varies between 2355 g and 2431 g while the maximum takeoff weight is 3600 g. This model counts on C1 dedicated remote controller, which features dedicated buttons for photo and video capture, a gimbal control dial and integrated rechargeable battery. It also includes HDMI and USB ports and mobile/tablet holder, allowing to connect mobile devices and compatible screens. The maximum transmission distance provided by this remote controller is 5 km in a unobstructed and free of interference environment.

Figure 3.3: Matrice 100

Figure 3.4: Matrice100 controler

3.1.2 Raspberry

Raspberry Pi is an ARM based credit card sized SBC (Single Board Computer) created by Raspberry Pi Foundation. Raspberry Pi runs Debian based GNU/Linux operating system Raspbian and ports of many other OSes exist for this SBC. The Raspberry Pi 3 Model B+ is the final revision in the Raspberry Pi 3 range. In particular the Raspberry Pi 3B+ mounted on the UAV has:

Figure 3.5: Raspberry pi 3B+

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
- Extended 40-pin GPIO header
- Full-size HDMI
- 4 USB 2.0 ports

- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- 4-pole stereo output and composite video port
- Micro SD port for loading your operating system and storing data
- 5V/2.5A DC power input
- Power-over-Ethernet (PoE) support (requires separate PoE HAT)

3.1.3 PARALLAX servomotor

The Parallax Standard Servo is ideal for robotics and basic movement projects. It can hold any position over a 180-degree range and is easily interfaced with any Parallax microcontroller.

Key Features:

- Holds any position between 0 and 180 degrees
- Accepts four mounting screws
- High precision gear made of the POM (polyacetal) resin makes for smooth operation with no backlash
- Weighs only 144 g

Figure 3.6: PARALLAX servomotor

3.1.4 NodeMCU Amica

NodeMCU Amica is a ESP8266 Wifi Module based development board. It has got Micro USB slot that can be directly connected to the computer or other USB host devices. It has got 15X2 Header pins and a Micro USB slot, the headers can be mounted on breadboard and the micro USB slot is for connection to USB host device that may be a computer. It has got CP2102 USB to serial converter.

Figure 3.7: NodeMCU Amica

3.1.5 Arduino

The Arduino Uno is an open-source microcontroller board based on the Microchip ATmega328P microcontroller and developed by Arduino.cc. The board is equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards (shields) and other circuits.The board has 14 Digital pins, 6 Analog pins, and is programmable with the Arduino IDE (Integrated Development Environment) via a type B USB cable. It can be powered by the USB cable or by an external 9-volt battery, though it accepts voltages between 7 and 20 volts.

Figure 3.8: Arduino

3.1.6 DC motors

A DC motor is any of a class of rotary electrical machines that converts direct current electrical energy into mechanical energy. The most common types rely on the forces produced by magnetic fields. Nearly all types of DC motors have some internal mechanism, either electromechanical or electronic, to periodically change the direction of current flow in part of the motor. The motors and the wheels that the rover uses belong to the Runt Rover Kit.

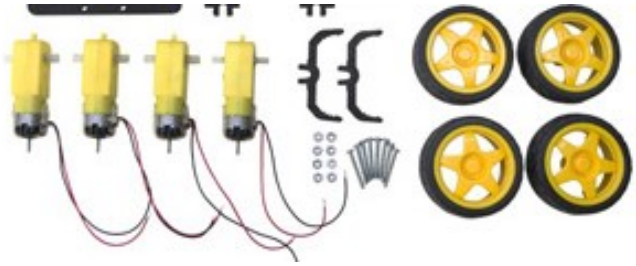

Figure 3.9: runt motors and wheels

3.2 Software description

A brief description of the software SDK used that allowed the development of the UAV and rover mission.

3.2.1 Onboard SDK

Onboard SDK is a protocol provided by DJI that allows users to connect their own On-board Embedded System (OES) to a supported DJI vehicle (Matrice 100 or Matrice 600) or flight controller (A3) using a common serial port (TTL UART). The Onboard

SDK enables deep interaction between the OES and a DJI flight controller, making it possible to get current state of the flight platform and to send commands to it. The structure consists of a series of packages that are an implementation of DJI Onboard SDK protocol and many other applications. It contains the messages, services and actions formats, the publishers and the subscribers required to run the Onboard SDK within the ROS framework. It also provides a python and C++ files that offering SDK functionality and a client node with functional examples.

The following command and actions are supported by the onboard SDK ROS:

- Activation
- Obtain/release flight Control
- Take off
- Go home
- Gimball Control
- Attidtude Control
- Photo Taking
- Start/stop Video
- Virtual Rc Control
- Broadcast frequency control
- Timestamp syncronization
- Hotpoint
- Waypoint Navigation
- Websocket Whith Baidu Map
- Mavlink and QgroundStation
- Arm/disarm Control
- Local Navigation

- Global Navigation
- Native Waypoint

The packages included within the ROS SDK and their structure are:

- dji_sdk_lib: The communication protocol implementation. It is a "catkinized" version of the available code on the official github repository.
- dji_sdk: This package is a group of packed APIs from dji_sdk_lib. All received data from the drone is published into ROS topics and all command sending API s have become ROS services. Other than the logic of processing topics and services, it also provides the above mentioned header file dji_drone.h in the include folder as a packed class of subscriber and service client. Developers are able to access all topics and services simply by this header file.
- dji_sdk_demo: A demo of how to use the dji_drone.h mentioned above.
- dji_sdk_dji2mav: An extended package of dji_sdk, which converts messages between DJI protocol and MAVLink protocol. It may be useful to developers who want do work on the QGroundControl.
- dji_sdk_web_groundstation: A ground station package using ROS Javascript API (ROS bridge suite) and Baidu Map API.
- dji_sdk_read_cam: A specified ROS package for Manifold. It converts the video stream from X3 into ROS sensor msgs/Image message format and publish it, but that function is not implemeted for M100.

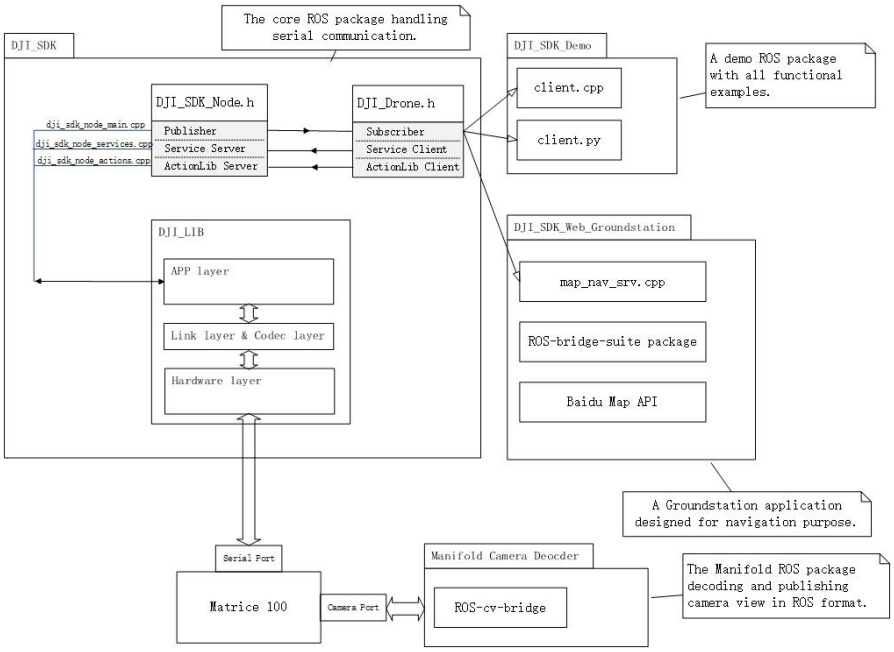

Figure 3.10: Onboard SDK ROS structure

3.2.2 ROS

ROS is an open-source collection of software frameworks for robot software development, providing operating system-like functionality. ROS provides standard operating system services such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Its architecture is based on Peer-to-Peer (P2P) communication between different nodes. These nodes can be defined as programs that perform different tasks and are running on one or more computers being part of a network.

The P2P communication is performed through messages that are posted, received and multiplexed by the different running nodes. The available documentation, the existent community and the convenient tools provided by ROS made of this OS the framework of choice. The ROS distribution that has been used is ROS Kinetic, released on May 2016. The Python ROS client libraries have been the implementation of choice to code the entire project, which is available online on github.

3.2.3 GAZEBO

Gazebo is an open-source 3D robotics simulator. Gazebo was a component in the Player Project from 2004 through 2011. Gazebo integrated the ODE physics engine, OpenGL rendering, and support code for sensor simulation and actuator control. In 2011, Gazebo became an independent project supported by Willow Garage. In 2012, Open Source Robotics Foundation (OSRF) became the steward of the Gazebo project. OSRF changed its name to Open Robotics in 2018. Gazebo can use multiple high-performance physics engines, such as ODE, Bullet, etc (the default is ODE). It provides realistic rendering of environments including high-quality lighting, shadows, and textures. It can model sensors that "see" the simulated environment, such as laser range finders, cameras (including wide-angle), Kinect style sensors, etc.

3.2.4 OpenCv

OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license. OpenCV's application areas include:

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object identification
- Segmentation and recognition
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)

- Motion tracking
- Augmented reality

OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. There are bindings in Python, Java and MATLAB/OCTAVE.

3.2.5 ArUco marker

ArUco is an OpenSource library for detecting squared fiducial markers in images. Additionally, if the camera is calibrated, you can estimate the pose of the camera with respect to the markers. The library is written in C++, but there are tools for using the library without programming. ArUco uses the class Marker, representing a marker observed in the image. Each marker is a vector of 4 points (representing the corners in the image), a unique id, its size (in meters), and the translation and rotation that relates the center of the marker and the camera location. It is prepared to detect markers of any of the Dictionaries allowed. By default, the MarkerDetector will look for squares and then it will analyze the binary code inside. For the code extracted, it will compare against all the markers in all the available dictionaries. Markers are comprised by an external black border and an inner region that encodes a binary pattern. The binary pattern is unique and identifies each marker. Depending on the dictionary, there are markers with more or fewer bits.

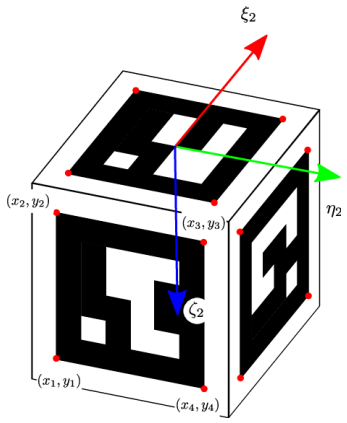

Figure 3.11: Aruco marker

3.2.6 DJI Flight simulator

The DJI Flight Simulator is a professional pilot training software for the enterprise. Adapting leading flight control technology to simulate the aircraft models and scenarios. The Flight simulator recreates the natural flight experience and provides enterprise users whith a complete training solution.

3.2.7 Kivy

Kivy is a free and open source Python library for developing mobile apps and other multitouch application software with a natural user interface (NUI). It is distributed under the terms of the MIT License, and can run on Android, iOS, GNU/Linux, OS X, and Windows. Kivy is the main framework developed by the Kivy organization,[2] alongside Python for Android, Kivy iOS and several other libraries meant to be used on all platforms.

3.3 UAV development

3.3.1 Hardware and Software initial setup

The first step of the project has been to control the drone with the computer external commands, in order to create a completely autonomous exploration mission that will be described in the next paragraph. For the on-board computer to work with DJI libraries, certain characteristics need to be fullfilled:

- An available TTL UART port.
- A small-form-factor PCs that can be powered from the aircraft's bus
- A communication system to choose from ROS, Linux, STM32 or QT in order to communicate to M100.

The choice is the Raspberry pi 3B+, with ubiquityRobots (ubuntu 16 LTS equivalent) and ROS kinetic installed inside, for the characteristics mentioned in the relevant paragraph. That operative system has been chosen because in order to use ROS as middleware, in particular, the kinetic version inasmuch as it is one of the version that supports the DJI-SDK Onboard library. It is necessary an operating system Linux-based and compatible with the Raspberry Pi. The environment that has been created is shown in the figure below.

Figure 3.12: Interface

In the Raspberry Pi through a UART cable to M100, in order to send simple commands that are listed in paragraph 3.2.1., two DJI official repositories have been used, DJI_SDK and DJI_SDK_demo. The first one allows to connect the on-board computer to the drone, using a serial port (TTL UART). The second one release adds support of Onboard-Payload SDK communication and time sync function. It also contains the Onboard SDK ROS wrapper and demos.

The initial ROS nodes created by the repositories is shows in the figure 3.8. It can be seen that the main node DJI_SDK handles serial communication between the drone and DJI_SDK_demo, where the program to control the drone has been created. The node DJI_SDK_Web_Groundstation is not useful to our purpose.

Both the ROS repositories have been written in C++, a cross-platformed language that can be used to create sophisticated high-performance applications, but we want to use a language easier than the other one, so thanks to [15] it can be understood how to fork DJI_SDK from the official github repository of DJI. To create the rosnode as previous configuration, we use catkin_make, is a convenience ROS tool for building code in a catkin workspace (The work environment where it can be used all the ROS features).

For the physical connection between raspberry and drone a UART cable, as previously mentioned, has been used; in particular, it has been connected to raspberry in the following ports: pin 8 with drone RX pin, Pin 10 with drone TX pin and pin 6 to ground.

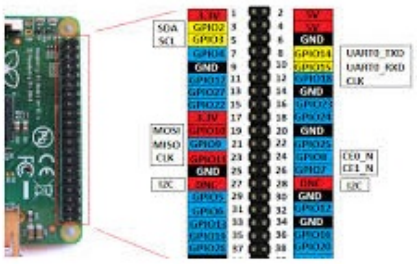

Figure 3.13: Raspberry pin

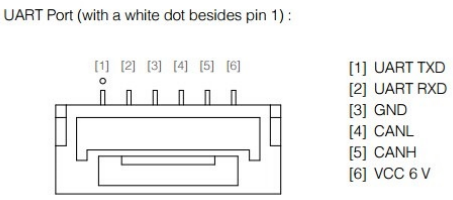

Figure 3.14: UART port

The figure below shows the complete configuration. In particular, the controller is always connected to the drone but deactivated (it can be reactivated by changing the flight mode from the controller and deactivate the Raspberry). The PC has been used only for simulations together with GAZEBO and to enable OSDK API. This allows communication between the onboard computer and the aircraft or flight controller.

3.3.2 Drone movement

The first flight test was made using the ROS node **dji_sdk_demo**, to verify the operation of the native functions and to understand if the output respects the given input. The tests were carried out using the functions **velocity_control**, **attitude_control**

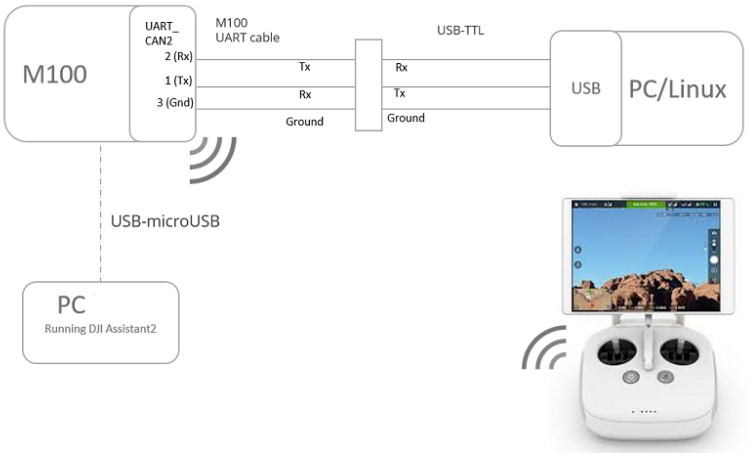

Figure 3.15: Interface

and **local_navigation** and making the drone perform small and long trajectories.

The first takes the speed as input and outputs a constant speed shift. The second takes a shift to perform a shift with variable speed. The last one has a GPS coordinate and performs a shift in that coordinate with respect to the local reference system. The first two must be executed within an iterative cycle, the third by a command. Based on the results obtained, which will be shown and explained in the next chapter, position errors were found in all three functions. These were depending on various factors, such as wind, GPS error, not sufficiently wide iterative cycles and slow stabilization due to high speed, as regards the second function.

To increase the precision of the position during inspection mission, it has been thought to create a control system that lets us know precisely the position of the drone. The GPS system and the vision system will also be used as auxiliary sensors to obtain position data within the research environment and with respect to the rover. The methods and tools used for this task will then be discussed in the following paragraph.

3.3.3 Trajectory planning

The trajectory planning is an algorithm that aims to find a sequence of valid configurations of the robot from the source to the destination. The trajectory chosen by the drone has been a wave path in order to guarantee the camera a complete view of the place where the target will be searched. The path that the drone will execute can be divided into cyclic paths, as shown in the figure below, which follow one another until

the drone identifies the rover.

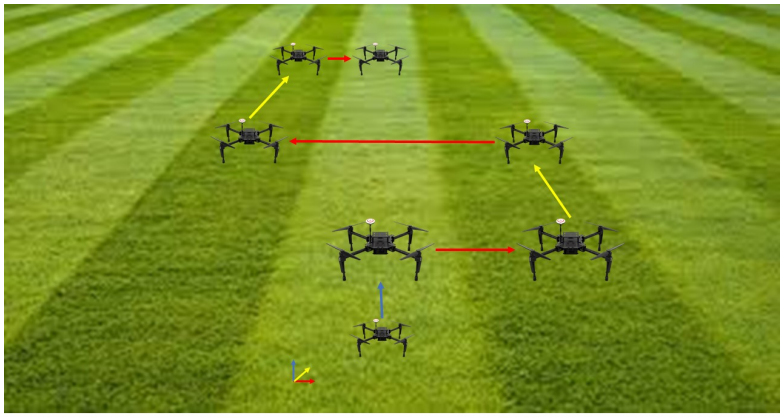

Figure 3.16: drone path

The path that the drone follows has been created by developing a planning point-to-point trajectory. Trajectory planning for point-to-point motion maps position as function of time between specified points. Velocity and acceleration along the trajectory can be computed by differentiating position with respect to time, and for a smooth path, velocity cannot have any discontinuities or the specified trajectory would require infinite acceleration.

As shown in Tables 3.1, the velocity profile, from t_0 to t_1, has a curvilinear path with a constant acceleration and an increase velocity. From t_1 to t_2, a straight profile whith constant velocity. In the end from t_2 to t_f a decrease velocity with constant negative acceleration.

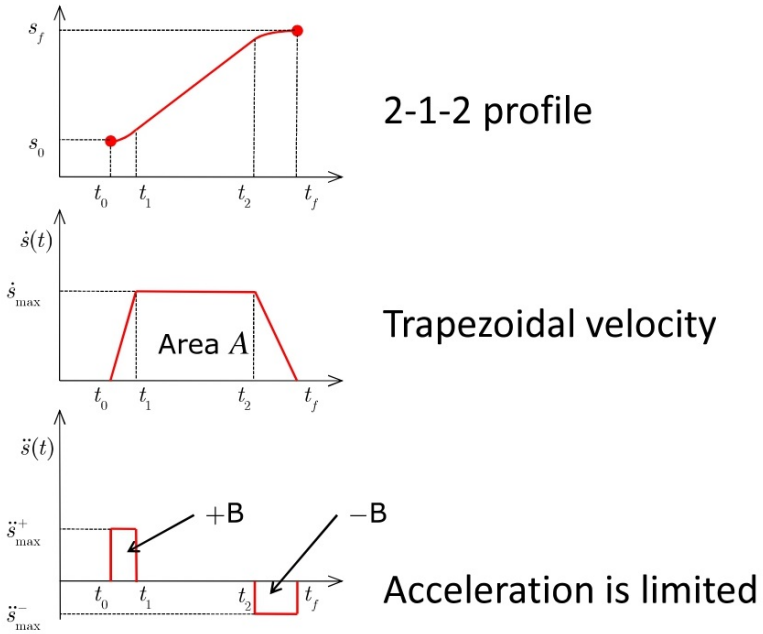

Table 3.1: Trajectory planning tables

In order to find the equation of the 2-1-2 profile, we define t_f,t_1 and t_2:

$$t_f = \frac{1}{\dot{s}_{\max}} + \frac{1}{2}\left(\frac{\dot{s}_{\max}}{\ddot{s}^{+}_{\max}} + \frac{\dot{s}_{\max}}{\ddot{s}^{-}_{\max}}\right) + t_0 \tag{3.3.1}$$

$$t_1 = \frac{\dot{s}_{\max}}{\ddot{s}^{+}_{\max}} + t_0 \tag{3.3.2}$$

$$t_2 = -\frac{\dot{s}_{\max}}{\ddot{s}^{-}_{\max}} + t_f \tag{3.3.3}$$

Since we have digital inputs, with a sample interval, it is needed to the use of a sampled data profile. So we assume that:

$$k_0 = t_0/T \quad k_1 = t_1/T \quad k_2 = t_2/T \quad k_f = t_f/T \tag{3.3.4}$$

Where k_0,k_1,k_2 and k_f are samples in time to,t_1,t_2 and tf. With N number of samples, three intervals are so defined:

$$\begin{aligned} \mathcal{K}_1 &\stackrel{\text{def}}{=} \{k : 0 \le k < k_1\} \\ \mathcal{K}_2 &\stackrel{\text{def}}{=} \{k : k_1 \le k < k_2\} \\ \mathcal{K}_3 &\stackrel{\text{def}}{=} \{k : k_2 \le k \le N-1\} \end{aligned} \tag{3.3.5}$$

The functions that represent the graphs in the table 3.1 are:

$$\ddot{s}_k = \begin{cases} \ddot{s}^+_{\max} & k \in \mathcal{K}_1 \\ 0 & k \in \mathcal{K}_2 \\ -\ddot{s}^-_{\max} & k \in \mathcal{K}_3 \end{cases} \tag{3.3.6}$$

$$\dot{s}_k = \begin{cases} \ddot{s}^+_{\max} kT + \dot{s}_0 & k \in \mathcal{K}_1 \\ s_{\max} & k \in \mathcal{K}_2 \\ \dot{s}_{\max} - \ddot{s}^-_{\max} (k - k_2) T & k \in \mathcal{K}_3 \end{cases} \tag{3.3.7}$$

$$s_k = \begin{cases} \frac{1}{2} \ddot{s}^+_{\max} k^2 T^2 + \dot{s}_0 kT + s_0 & k \in \mathcal{K}_1 \\ \dot{s}_{\max} (k - k_1) T + s_1 & k \in \mathcal{K}_2 \\ -\frac{1}{2} \ddot{s}^-_{\max} (k - k_2)^2 T^2 + \dot{s}_{\max} (k - k_2) T + s_2 & k \in \mathcal{K}_3 \end{cases} \tag{3.3.8}$$

To make the trajectory planning generic for any distance, is necessary to get the path percentage. So, it is needed to make the hypotesis of unit trajectory and zero initial and final velocity and to fix velocity, maximum positive and negative acceleration constant:

$$s_0 = 0, \dot{s}_0 = 0, s_f = 1, \dot{s}_f = 0, \ddot{s}^+_{\max}, \ddot{s}^-_{\max}, \dot{s}_{\max} \tag{3.3.9}$$

Finally, a value of s is obtained, which it will be multiplied for the difference between the initial position and the final position, to output the sampled shift. The software implementation is shows in Appendix A.

The block diagram, shown below,represents how the trajectory planning is implemented on the drone. It should be noted that an open loop system does not monitor or measure the condition of its output signal, as there is no feedback. Indeed, during flight tests, the trajectory planning does not guarantee sufficient accuracy and precision by itself.

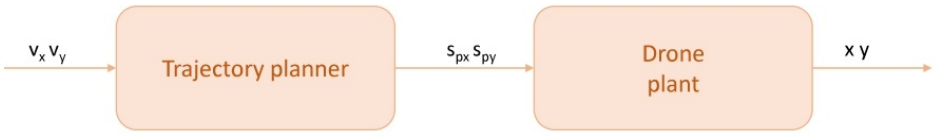

Figure 3.17: Open loop Drone system

3.3.4 Drone transfer function

To achieve better flight performance, it has been decided to implement a PID control. In order to choose the type of PID controller, the first stuff it has been to find the drone transfer function. The only present information about the drone is the input signal that is sent from the on-board PC and the output signal that is transformed into drone movements. This transfer function is therefore totally unknown, so it will take the name of black-box or box-Jenkins.

Before using estimation methods for the formulation of a drone transfer function, flight tests have been performed with the drone, where the input and output data have been acquired.

The acquired data has been taken by setting different speeds within the planning trajectory and tested on the different axes. The data obtained was merged to obtain a sufficient number of values , as shows in plot 3.2 .

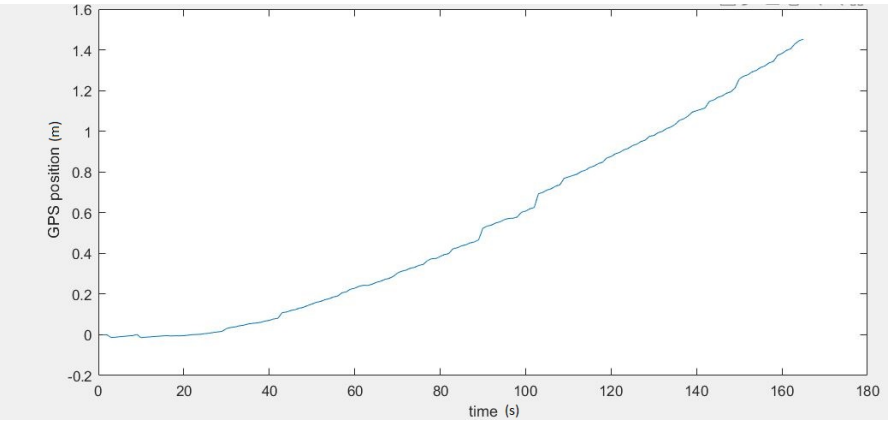

Table 3.2: Real output values whith trajectory planning

Different estimation methods have been used to study the drone transfer function.

The estimation problem refers to the empirical evaluation of an uncertain variable, like an unknown characteristic parameter or a remote signal, on the basis of observations and experimental measurements of the phenomenon under investigation.

The first estimation method that has been used is the least-square method, a standard approach in regression analysis to approximate the solution of overdetermined systems.

The "least squares" method is a form of mathematical regression analysis used to determine the line of best fit for a set of data, providing a visual demonstration of the

relationship between the data points. Each point of data represents the relationship between a known independent variable and an unknown dependent variable.

The transfer function will be calculated through the least square method:

$$Gp(z) = \frac{\theta_3 z + \theta_4}{z + \theta_1} \tag{3.3.10}$$

Where the Theta are the unknown coefficients of the transfer function.

Given the measurements of real value over a time interval, we find the real parameters Theta such that the following relationship holds:

$$y(t) = \varphi(t)^T \theta \tag{3.3.11}$$

Where the y is the output and the Phi is the input matrix and Theta the unknown parameters matrix.

The theta unknowns are then estimated using the following equation:

$$\hat{\boldsymbol{\theta}}_{\mathrm{LS}} = \left[\sum_{t=1}^{N} \varphi(t)\varphi(t)^T\right]^{-1} \left[\sum_{t=1}^{N} \varphi(t)y(t)\right] \tag{3.3.12}$$

Assuming that the plant tf has a second-order system, as describe in equation 3.3.10 and replacing it in the found Theta, it was obtained the following transfer function:

$$Gp(z) = \frac{0.041011z}{(z-1) \quad (z-0.0799)} \tag{3.3.13}$$

Inserting then an input to the transfer function and comparing the results with the real outputs, Table 3.3, it can be seen that the transfer function found does not respect the desired trend.

It is needed, therefore, an estimation method that better follows the curve's progression, carried out by experimental data , so let's consider three estimation methods: ARX, ARMAX and OE.

The first one, autoregressive exogenous model, has a type structure:

$$y(t) = \frac{B(z)}{A(z)}u(t) + \frac{1}{A(z)}e(t) \tag{3.3.14}$$

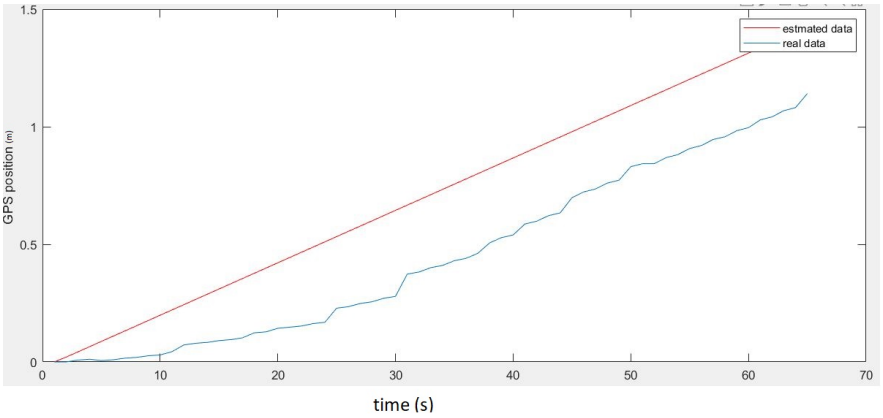

Table 3.3: Tf with Least Square method

where:

$$\begin{aligned} A(z) &= 1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_{n_a} z^{-n_a} \\ B(z) &= b_1 z^{-1} + b_2 z^{-2} + \cdots + b_{n_b} z^{-n_b} \end{aligned} \tag{3.3.15}$$

The second, autoregressive moving average exogenous model, has a type structure:

$$y(t) = \frac{B(z)}{A(z)} u(t) + \frac{C(z)}{A(z)} e(t) \tag{3.3.16}$$

with:

$$\begin{aligned} A(z) &= 1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_{n_a} z^{-n_a} \\ B(z) &= b_1 z^{-1} + b_2 z^{-2} + \cdots + b_{n_b} z^{-n_b} \\ C(z) &= 1 + c_1 z^{-1} + c_2 z^{-2} + \cdots + c_{n_c} z^{-n_c} \end{aligned} \tag{3.3.17}$$

The last one, output error model, the less reliable, has a type structure:

$$y(t) = w(t) + e(t) = \frac{B(z)}{F(z)} u(t) + e(t) \tag{3.3.18}$$

with:

$$\begin{aligned} F(z) &= 1 + f_1 z^{-1} + f_2 z^{-2} + \cdots + f_{n_f} z^{-n_f} \\ B(z) &= b_1 z^{-1} + b_2 z^{-2} + \cdots + b_{n_b} z^{-n_b} \end{aligned} \tag{3.3.19}$$

These three methods have been executed simultaneously with the help of the Matlab. The mean values has been removed from the input-output measurements, to obtain zero mean value sequences of length N. The estimation dataset has been used to identify

ARX,ARMAX and OE models of different orders and delay, and to look for models that guarantee satisfactory characteristics of whiteness of the residuals, in particular:

- ARX models of order na = nb and delay nk [1,2,3,4] require nk>2 to have a few (less than 5) values of the autocorrelation function of residual outside enough the 99% confidence limits;
- ARX models of order na = nb =nc and delay nk [1,2,3] require nk>1 to have a few (less than 5) values of the autocorrelation function of residual outside enough the 99% confidence limits;
- ARX models of order nf = nb and delay nk [1,2,3] do not guarantee satisfactory characteristic of whiteness of the residuals for any order of nf, since too many values of autocorrelation function of residuals are largery outside the 99% confidence limits

The validation dataset is then used to compare the identified models and to assess their model quality, by minimizing the Root Mean Square Error:

$$RMSE = \sqrt{\frac{1}{N - N_0} \sum_{t=N_0+1}^{N} [y(t) - \hat{y}(t)]^2} \tag{3.3.20}$$

Note that, in the case of ARX and ARMAX models, the predicted output provides better performance than the simulated output, since ti exploits more information. For this reason, the value of the RMSE using only the predicted output are here reported for all the models:

The choice of the best value of RMSE is given taking into consideration only the values that satisfied the residue analysis and choosing the RMSE with the lowest value.

For ARX models, the best trade-off between RMSE and model order n is given by ARX(3,3,1), for ARMAX model, insted, is given by ARMAX(2,2,2,2) and the OE. The transfer function obtained, ftrom ARX(3,3,1), is equal to:

$$Gp(z) = \frac{0.039679z}{(z - 1.008) \quad (z + 0.2475) \quad (z - 0.1345)} \tag{3.3.21}$$

It should be noted that although the plant was taken as a blank box, there is a very close pole from one, which induces an integrating characteristic.

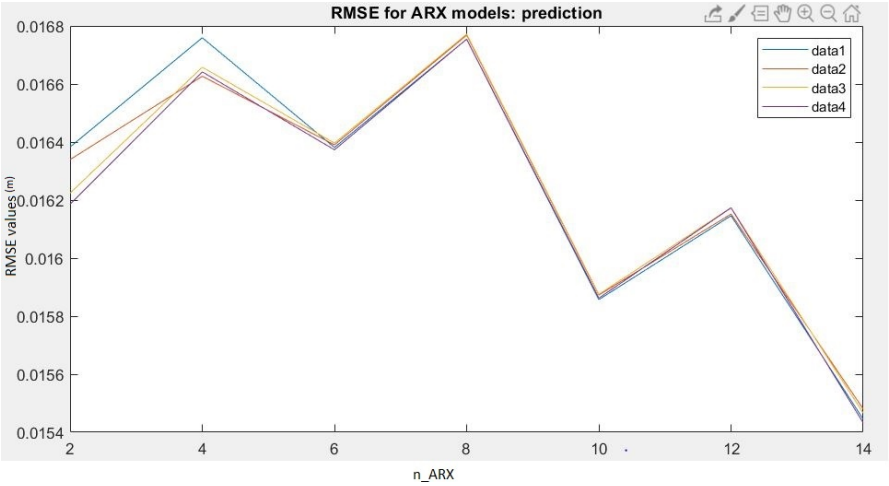

Table 3.4: RMSE for each n_ARX =na+nb

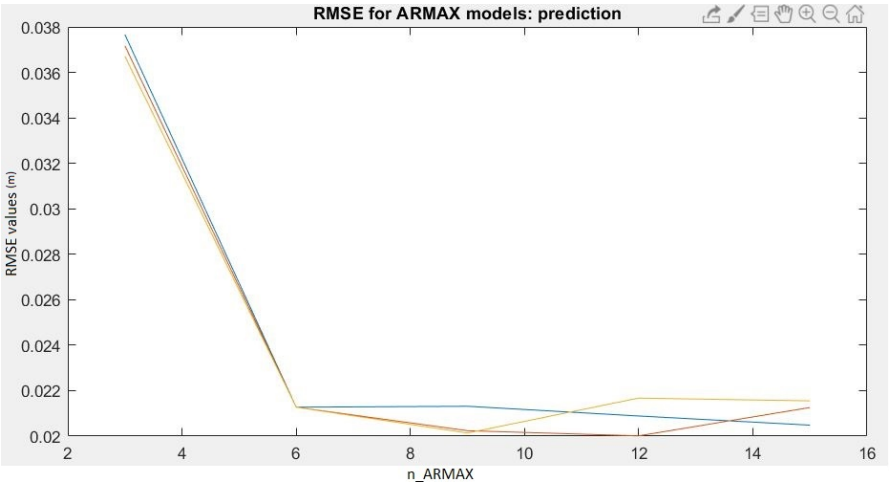

Table 3.5: RMSE for each n_ARMAX =na+nb+nc

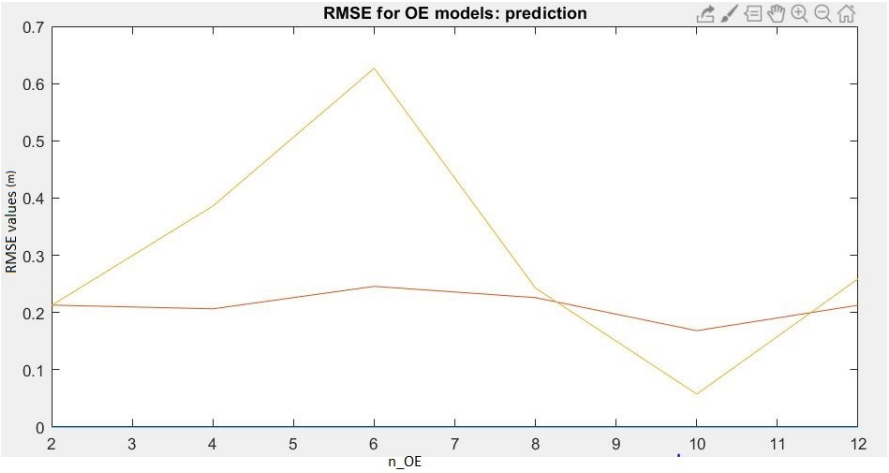

Table 3.6: RMSE for each n_OE =nf+nb

Inserting then an input to the transfer function and comparing the results with the real outputs, Table 3.7, shows how the transfer function follows the real output better.

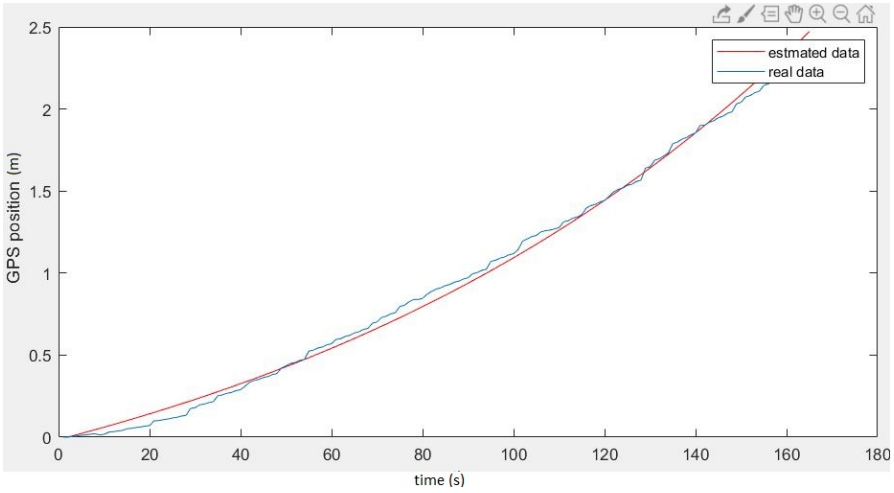

Table 3.7: Tf with ARX method

3.3.5 PD control

Now that the transfer function of the plant is know, it is possible to design the controller. Since the tf already has an integrative action, intrinsic to the velocity_control function, it was decided to create a PD control, with this structure:

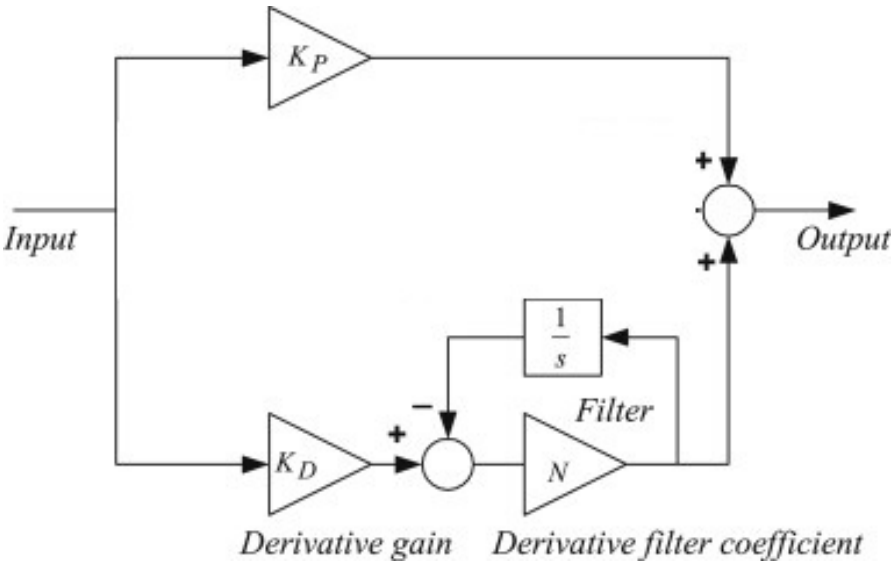

Figure 3.18: PD control

A typical structure of a PD control system is shown in Fig. 3.18, where it can be seen that in a PID controller, the error signal e(t) is used to generate the proportional and

derivative actions, with the resulting signals weighted and summed to form the control signal u(t) applied to the plant model. A mathematical description of the PD controller is:

$$P + D\frac{N}{1 + N \cdot T_s \frac{z}{z-1}} \tag{3.3.22}$$

where:

- Proportional compensation: the main function of the proportional compensator is to introduce a gain that is proportional to the error reading which is produced by comparing the system's output and input;
- Derivative compensation: in a unitary feedback system, the derivative compensator will introduce the derivative of the error signal multiplied by a gain KD, in other words, the slope of the error signal's waveform is what will introduced to the output. Its main purpose is that of improving the transient response of the overall closed-loop system.

A problem with the derivative term is that it amplifies higher frequency measurement or process noise that can cause large amounts of change in the output. So, in this case it can be implemented in order to filter the measurements with a low-pass filter to remove higher-frequency noise components. As low-pass filtering and derivative control can cancel each other out, the amount of filtering is limited.

The constants were obtained using Matlab, in particular the PID tuner tool. The settling time and the overshoot are set so as to have a fast response and a transient behavior as robust as possible, as shown in the figure below.

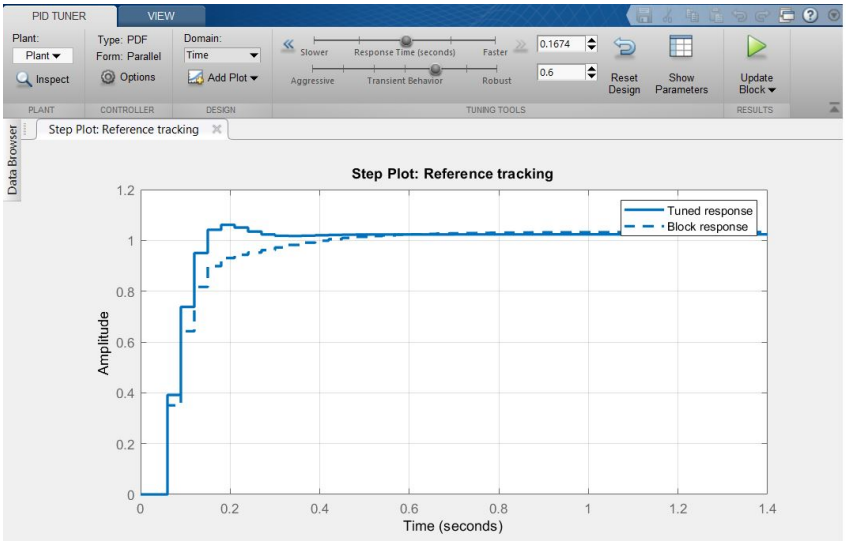

Figure 3.19: PID Tuner

Bringing the complete block system back to simulink, simulating the trajectory planning speed with a trapezoidal wave and then integrating it to get the abscissa profile, we will have a system like:

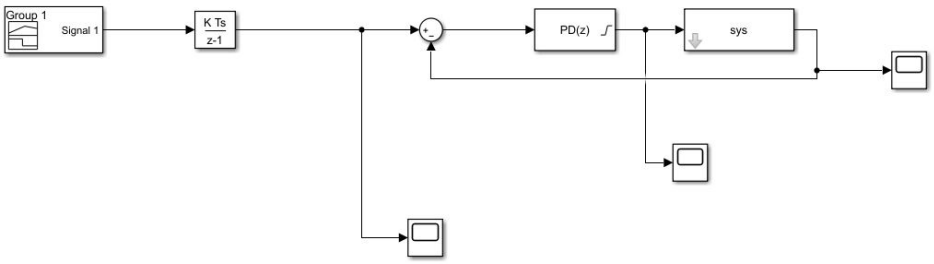

Figure 3.20: control block scheme

The block system shown above represents the drone transfer function with PID control implemented. The reference signal simulates the trajectory planning to move the drone one meter up along the x-axis (left figure). The figure on the right shows the output signal. The simulated data shows how the output signal follows the reference signal well. These results display that given an input signal to the drone it can be obtained a precise movement output.

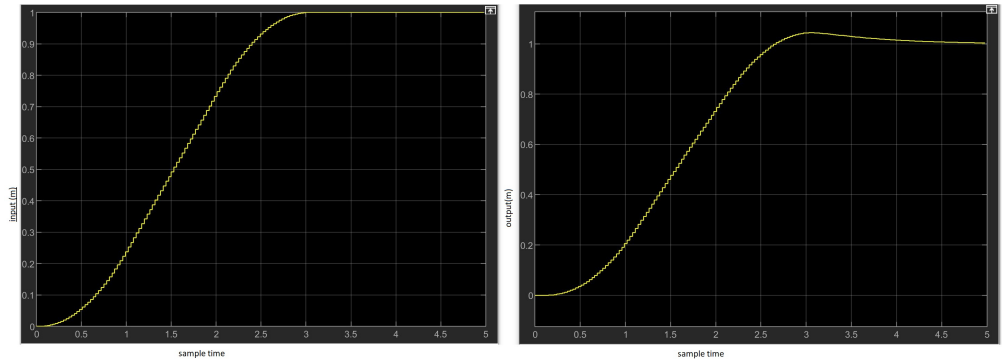

Figure 3.21: reference-output response

3.3.6 Target search and landing

For the search of the rover and the precision landing, a ROS node has been created. This works in sync with a Kalman filter, described in next paragraph, exchanging angle and position data of the target. The research objective is ArUco marker, a synthetic square marker composed by a wide black border and an inner binary matrix, that determines its identifier (id). Given an image where some ArUco markers are visible, the detection process has to return a list of detected markers. This includes the position of its four corners in the image (in their original order) and the id of the marker. When the camera identifies the marker, the Cartesian axis is also identified on the ArUco center. From the image below, it can be seen how the trench of the ArUco is unbalanced with respect to the one of the camera. So, it will be necessary to operate on the obtained results such that there are no errors in relative positions.

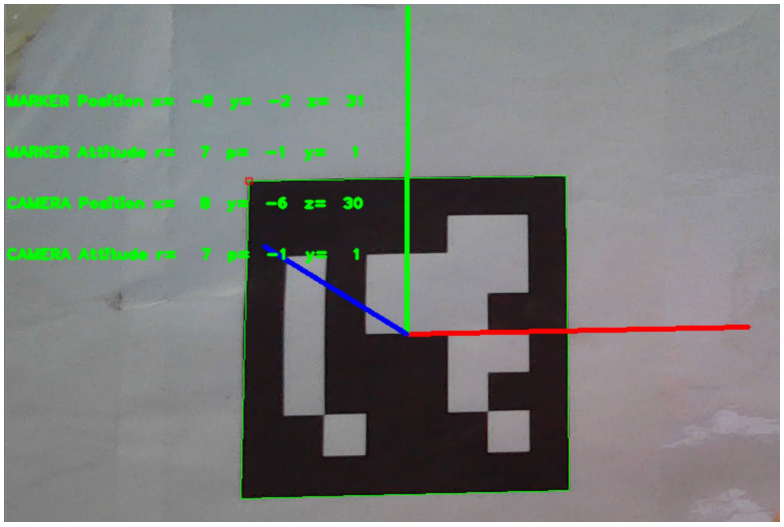

Figure 3.22: Example of ArUco detenction

To perform camera pose estimation, we need to know its calibration parameters. They are the camera matrix and distortion coefficients. That calibration is performed thanks to calibratecamera() function which takes ArUco images in different positions from a predefine folder. It creates the camera and the distortion matrices that are saved in a text file before every mission. During the research mission, the camera takes a frame every 1 second. The pictures are then rendered in black and white, therefore lighter and easier to process. When the marker is found the program estimates M100 position with respect to the marker. For the ArUco the Perspective-n-Point (PnP) problem is solved and the final estimation is averaged. The PnP problem aims to determine the position (T) and orientation (R) of a camera given its intrinsic parameters (K) and a set of n correspondences between 3D points and their 2D projections.

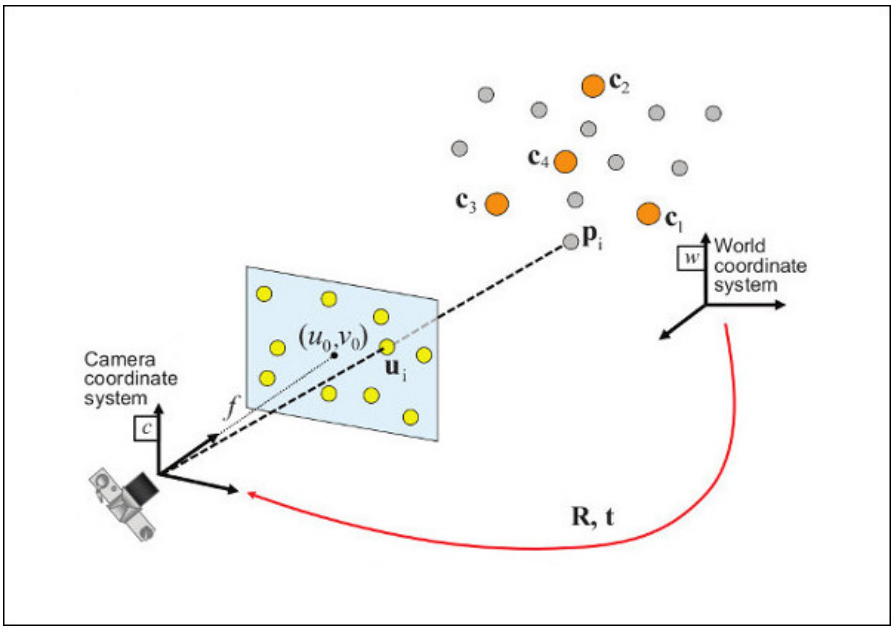

Figure 3.23: PnP problem

The PnP problem has been addressed by Opencv that implements the SOLVEPNPITERATIVE, based on Levenberg-Marquardt optimization, where the function finds such a pose that minimizes reprojection error, that is the sum of squared distances between the observed projections and the projected.

Once the marker has been found, the software identifies the roto-translation matrix of the marker that is saved as a backup location. The appendix B shows the code implementation.

After that, the landing phase begins. A virtual truncated cone will be created above the

ArUco marker. The surfaces of the virtual solid represent the movement boundaries of the drone, beyond which the marker will no longer be visible. During landing the drone will line up with the axis of the dino cylinder to reach the lower base of the virtual cone trunk described above.

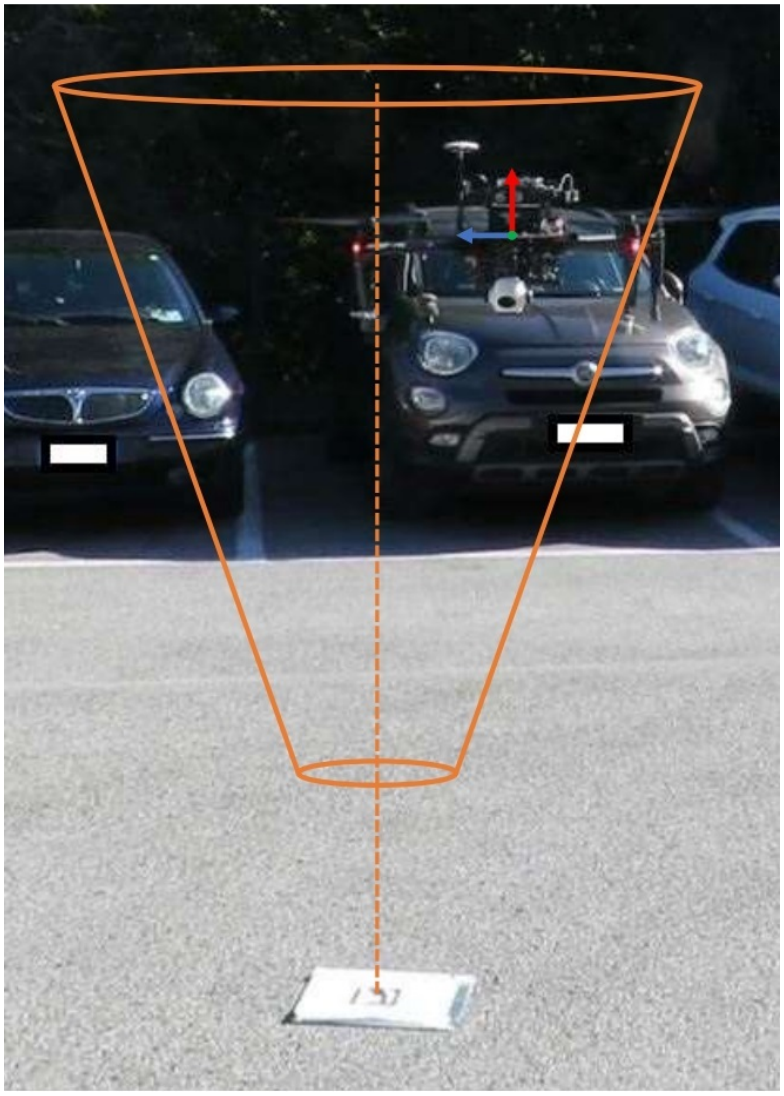

Figure 3.24: truncated cone bound

During the alignment of the axis of the drone with the axis of the marker, the drone will gradually descend until it reaches a height close to the marker. Now, the distance of the UAV to the origin of the ArUco will be calculated and sent via ROS to the rover, that will try to minimize the landing inaccuracy.

When the two axes are close enought, the drone aligns to the x-axis to ensure a secure rover grip after landing, through a yaw movement.

A Kalman filter has been designed to have a prediction of the position and to reduce the signal noise.

Kalman filtering, also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each timeframe.

Kalman filter has been inserted inside a ROS node that takes the input data of the ArUco marker and as output the predictions of the data filtered 10 times faster than the camera.

The landing platform state vector x(tk) consists of the six position coordinates and the six velocity components . Moreover, the measurement vector y contains the six position coordinates computed by the detection algorithm.

$$\mathbf{x}\left(t_k\right) = \begin{bmatrix} \hat{x}\left(t_k\right) \\ \hat{y}\left(t_k\right) \\ \hat{z}\left(t_k\right) \\ \hat{v}_x\left(t_k\right) \\ \hat{v}_y\left(t_k\right) \\ \hat{v}_z\left(t_k\right) \\ \alpha_x\left(t_k\right) \\ \beta_y\left(t_k\right) \\ \gamma_z\left(t_k\right) \\ \hat{\alpha}_x\left(t_k\right) \\ \hat{\beta}_y\left(t_k\right) \\ \hat{\gamma}_z\left(t_k\right) \end{bmatrix} \tag{3.3.23}$$

$$\mathbf{y}\left(t_k\right) = \begin{bmatrix} x\left(t_k\right) \\ y\left(t_k\right) \\ \alpha\left(t_k\right) \\ \beta\left(t_k\right) \\ \gamma\left(t_k\right) \end{bmatrix} \tag{3.3.24}$$

The state transition matrix corresponding to this system is:

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & Kt & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & Kt & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & Kt & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & Kt & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & Kt & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & Kt \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3.25}$$

The landing platform state prediction is performed between a measurement and the follow. As the measurement gets older, the uncertainty of the prediction grows, as shows in equation below, since landing platform's trend of motion is more likely to have changed. P is the covariance matrix and Q is the process noise covariance matrix.

$$\mathbf{P}\left(t_k\right) = \mathbf{F}\mathbf{P}\left(t_{k-1}\right)\mathbf{F}^{\mathbf{T}} + \mathbf{Q}\left(t_k\right) \tag{3.3.26}$$

Once a new measurement arrives, the new information is employed to update the state vector. The state vector x and covariance matrix is computed at the measurement time, then, x(tm) and P(tm) are employed to updatethe state vector with the new measurement.

$$\begin{aligned} \mathbf{x}\left(t_m\right) &= \mathbf{x}\left(t_m\right) + \mathbf{K}\left(t_m\right)\left(\mathbf{y}\left(t_m\right) - \mathbf{H}\mathbf{x}\left(t_m\right)\right) \\ \mathbf{P}\left(t_m\right) &= \mathbf{P}\left(t_m\right) - \mathbf{K}\left(t_m\right)\mathbf{H}\mathbf{K}\left(t_m\right) \end{aligned} \tag{3.3.27}$$

H is the transformation matrix that maps the predictions into the measurement domain, K is the Kalman gain matrix which is a function of the relative certainty of the measurements and predicted state. With a high gain, KF places more weight on the measurements, and thus follows them more closely. With a low gain, the filter follows the predictions more closely, smoothing out measurement noise but decreasing the

responsiveness.

$$\mathbf{K}(t_m) = \mathbf{P}(t_m)\,\mathbf{H}^T\left(\mathbf{H}\mathbf{P}^T(t_m)\,\mathbf{H}^T + \mathbf{R}\right)^{-1} \tag{3.3.28}$$

R is the covariance matrix of the measurement noise, whose terms can be estimated experimentally.

Notice that while the measurement provides only the three position coordinates, the state vector consists of the three coordinates of both position,velocity and acceleration. As the position coordinates are expressed on the body reference frame, assuming they remain the same is not plausible as long as M100 moves. Thus, it is necessary to estimate the landing platform relative velocity and acceleration from the computed position.

An example of result of the Kalman filter are shown in the table below.

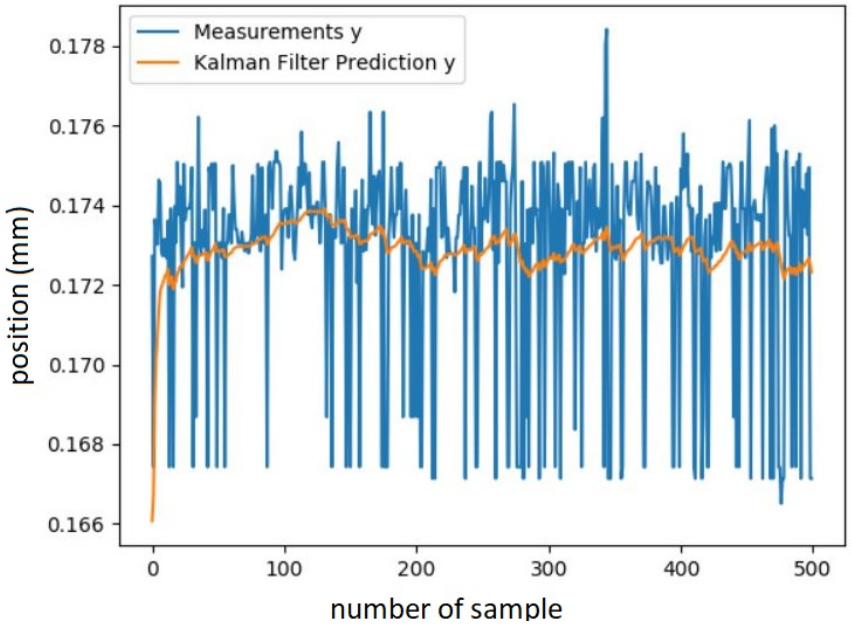

Table 3.8: Tf with Least Square method

The example shows an experiment carried out with a fixed camera and an ArUco marker positioned in front of it, on the graph we can see the measurement error with respect to filtered data. Thanks to the Kalman filter, the disturbances are damped, so when the data of the ArUco position is taken, the errors on the Cartesian triad will be reduced.

3.3.7 ROSnode final structure

A node is a process that performs computation. Nodes are combined together into a graph and communicate with one another using streaming topics, RPC services, and the Parameter Server. These nodes are meant to operate at a fine-grained scale. A robot control system usually comprises many nodes.

The ROSnode structure to be created is shown in the figure below;

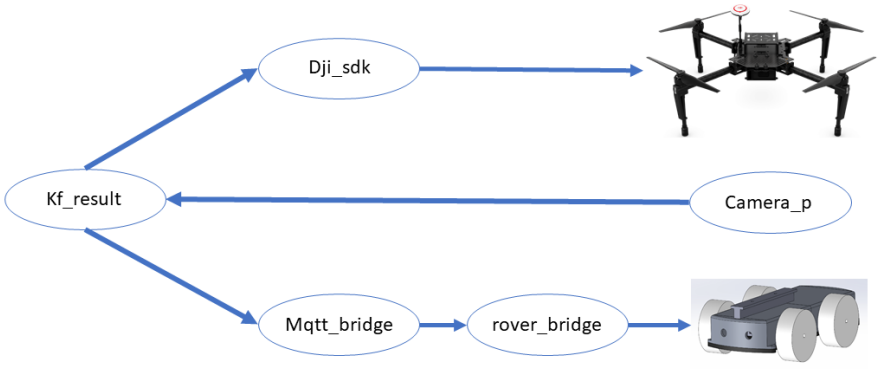

Table 3.9: ROS node Structure(to update)

Predifine DJI nodes have not been modified, but three extra nodes are added. Communications between the various ROS nodes starts through publish – subscribe (messaging pattern). Publish–subscribe is a sibling of the message queue paradigm, and is typically one part of a larger message-oriented middleware system. The senders of messages, called publishers, send the data, with a topic that specifies which kind of message has been sent, in a central broker where the receivers, called subscribers, call that topic to receive the data.

The first node, dji_sdk, is the one used to control the drone. From this node it is possible to access the information of the GPS, gyroscope, compass and other sensors integrated into the drone.

The second node, camera_p, is the identification node of the ArUco marker, which processes the frames and publishes a file with a flag. If positive, that means that the

marker has been identified, the node sends the 3D coordinates of the marker. This message will then be read by the main program which will set the mission based on the content of the message.

The third node, MQTT_bridge, is a bridge between MQTT and ROS. It creates a server where both protocols can receive and send messages through a topic. It is used for communication between the rover and the application and between the rover and the drone.

The fourth node, kf_result, is the kalman filter that receives the data from the camera node. It filters and predicts the data sent to the control node of the drone and of the rover (a node that comes into action only during landing).

The fifth node, rover_bridge, is the one that deals with the control of the rover in the landing phase. For this task an algorithm has been created which, communicating with the Kalman filter node, identifies the position of the rover with respect to the drone coordinates. It also computes the differences in the angles between the two reference systems. The centering mission is divided into two parts:

- Rotation of the rover so that the Marker's x-axis includes the origin of the reference axes of the drone.
- Movement of the rover until the origin of its reference axes coincides with the origin of the reference axes of the drone.

3.3.8 Drone specification

After different tests and after observing the drone behaviors in various environments, the following features can be listed:

- The drone has been tested in different places, indoor and outdoor, it has been noted that the software developed, using the GPS system, has greater precision in outdoor locations;
- The marker can be recognized by the software if at least 90% of the marker is visible;
- The maximum load the drone can withstand does not allow to add many features to the search rover;

- The drone is controlled through the 2.5Ghz band. The distance within which the drone receives data is 10 meters.

3.4 Rover development

3.4.1 Design Concept

For the construction of the inspection robot capable of being transported by the drone, it was necessary to design a solution that is compatible with the drone specifics (maximum payload, form suitable for the grab). At the same time, it has been developed small enough to be used for research missions in cramped places.

Due to the various constraints, it was therefore decided to create a robot for simple visual inspection. So, the drone will then be equipped with a camera and a LED to have good exposure of the image even in dark environments.

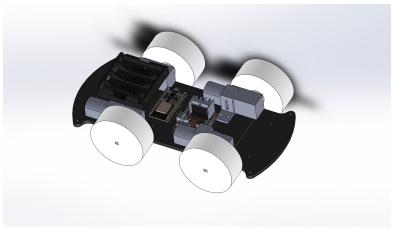

Figure 3.25: bottom of the rover

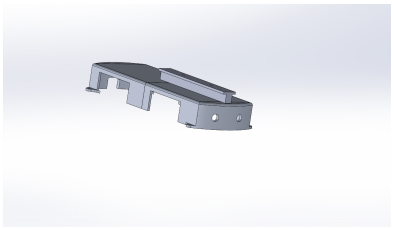

Figure 3.26: top of the rover

As can be seen in the figure above, the internal structure of the rover has been designed to minimize the width of the car.

The basic structure of the rover has been formed by a plate of ABS suitably shaped to have a light but spacious structure that can contain the internal components.

As far as the rover movement is concerned, a car has been designed with a rubber-wheeled four-wheel-drive structure, so that it can move forward even in the case of uneven terrain.

The motors have been positioned at the vehicle's outside and blocked by L-shaped structures of the same material as the base. Connected to the motor, we find an H bridge, a fundamental component for DC motor control. This component allows us to control the speed and direction of rotation of the motor, unlike a simple switch that would only allow us to work in ON/OFF mode. In the bottom of the drone base we find the battery compartment which is connected in parallel with the H bridge and the MCUNode Amica, in order to guarantee the correct voltage (6V) to the components.

The MCU Amica is the microcontroller that takes care of the movement of the rover.

It is equipped with a Wi-Fi module that is connected with the Raspberry Pi for the exchange of information.

On the upper part of the rover, made of ABS, there is a fundamental part of the car, a T-shaped part. Thanks to this piece, the drone can hook the rover.

In the front of the rover two slots have been made for the insertion of the inspection camera and the LED.

The rover can be driven wirelessly, through a dedicated app, where it will be possible to view the images of the camera and acquire a frame in case of need.

3.4.2 Movement

The main core of the rover is the MCUNode that sends a PWM signal to the H-bridge, an electronic circuit that switches the polarity of a voltage applied to a load.

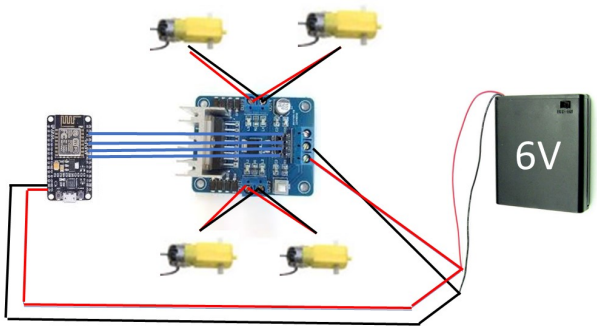

Figure 3.27: Rover scheme

The figure above shows the layout of the rover's internal connections. The red and black connections represent the power supply, the blue connections the PWM signals.

The information, that the PC use to move the rover, is exchanged through two different programs. The first is the inspection mission program, where you can control the rover and receive information from the camera. The second is the rover recovery program, where the Raspberry Pi sends the command data to the rover that will be positioned under the drone.

In particular, four types of movements are implemented:

- Forward, where the four motors turn clockwise;
- Backward, where the four motors turn counterclockwise;
- Left, where the right motors turn clockwise and the left motors turn counterclockwise;
- Right, where the left motors turn clockwise and the right motors turn counterclockwise.

The movement command for the inspection part, is given through a dedicated application, created in Python through the Kivy libraries.

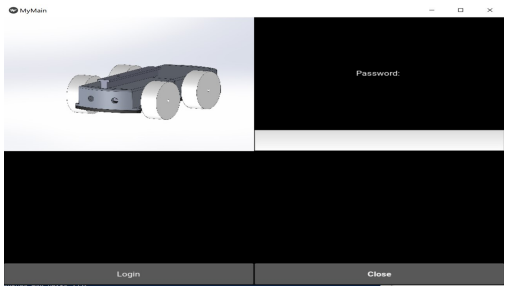

Figure 3.28: Login page

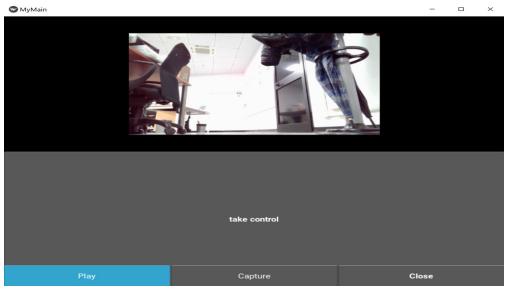

Figure 3.29: main page

It consists of two screens, the login page, left figure , where it is necessary to enter with a password. The second screen, right figure, is the command screen where there are four keys:

- Play: which activates the rover's camera
- Capture: takes a camera frame
- Take control: get control of the rover through the keyboard
- Close: close the application

The third button was designed in such a way that when one of the four arrows on the keyboard is pressed, it sends the movement signal. This is based on the arrow pressed and the respective command sent to the MCUNode via the MQTT protocol. To stop the movement of the motors a null message needs to be sent.

The connection between the application and the MCUNode is possible thanks to a ROS node started inside the Raspberry Pi. At the beginning of the mission, the latter creates

a local message broker to which it is possible to send movement messages through a specific topic. The ROSnode structure has been described in the previous paragraph.

When the drone's inspection mission is over, the movements of the rover pass to the onboard PC In this task the rover has the aim of improving the centering accuracy between the reference systems of the rover and the UAV.

As already briefly described in the previous paragraph, it is possible to divide the rover movement algorithm into two parts:

- In the first part, the rover aims to rotate until the x-axis of the visual marker coincides with the origin of the reference axes of the drone. The Raspberry Pi identifies the position of the rover with respect to that of the drone. In particular, using the visual marker identification algorithm and the Kalman filter explained above, the position of the marker and the yaw angle are identified to understand where and how the rover is oriented with respect to the drone position.

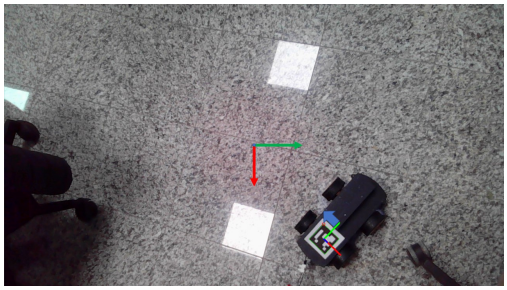

Figure 3.30: Rover before rotation

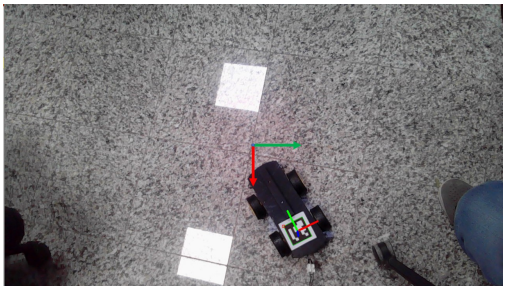

Figure 3.31: Rover after rotation

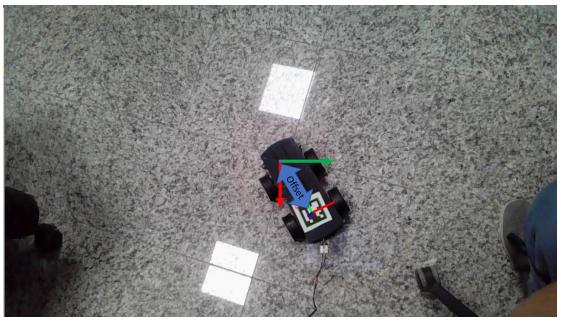

Figure 3.32: Rover translation

As can be seen in the figure above, the drone identifies the marker positioned on the rover. Then depending on the sign of the coordinates of the origin of the ArUco with respect to the coordinates of the camera, the drone estimates in which

position it is compared to the rover. The algorithm then calculates the current yaw angle and estimates the desired yaw angle when the origin lies on the x-axis of the visual marker. So, the rover makes a rotation on itself until the condition mentioned above is not reached.

- The second part of the task is the movement of the drone forward until its origin coincides with the origin of the rover. The ArUco and the camera aren't respectively centered in the origin of the rover and of the drone. So, an offset has been applied to compensate that distances.

3.4.3 Rover specification

The reasoning carried out on the drone can also be done on the rover, in fact, after a series of tests carried out it is possible to list a series of specifications.

- The rover guarantees a good grip on different types of terrains but with few steep gradients;
- The commands are sent through the 2.5 GHz band, the maximum limit that the rover can go far from the operator is about 10 meters.

3.5 Rover grab

Here, the design, the fabrication, and the evaluation of the grasping system has been presented. This system will be used to take the rover once the inspection mission is completed.

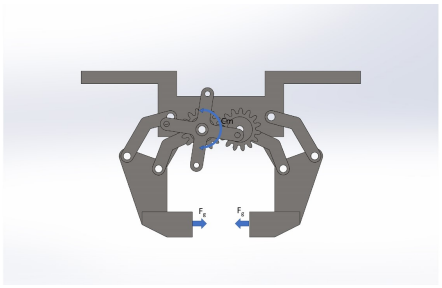

Figure 3.33: torque acting on the grab and resulting forces

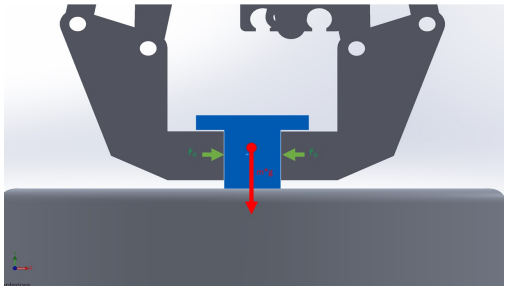

Figure 3.34: gripping force on the rover

The grip structure of the rover has been designed to lift the weight of the drone, around 700g. It does not obstruct the vision of the inspecting camera and it is positioned to reduce the inertia of the body transported. The conceived structure is formed by eight revolute joints and two degrees of freedom. There are eight arms that support and move the two pliers that will hook the rover. Two arms are connected to two gears that transmit a variable torque from the servomotor.The servomotor will give positive torque in the case of pliers opening, negative otherwise. The other arms support the movement and the weight of the rover once hooked.

That device can perform its task throught two modes:

- Force-fit where contact forces are applied and, because of friction, they hold the rover in position.
- Form-fit where the shape of the gripping device is designed in order to create a hook for the rover.

The system is translational and guarantees a perpendicular force to the contact surface. For the calculation of the normal force, a free body analisys of the component that will carry out the grasping has been made.

Taking into consideration the grab system once the rover has been hooked, and bearing in mind that, the rover after the engagement will exert a pressure of 5.444 N/cm^2, the

free-body system results (Figure 3.35):

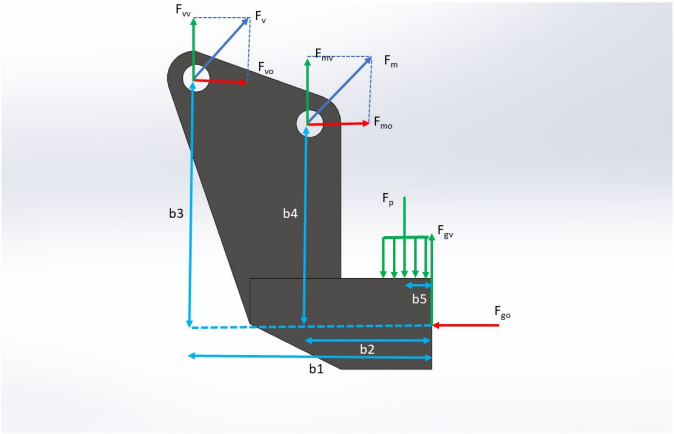

Figure 3.35: forces acting on the gripping body

The resulting equilibrium equations are:

$$\begin{cases} \mathrm{F}_{vv} + \mathrm{F}_{mv} = \mathrm{F}_{\mathrm{gv}} \\ \mathrm{F}_{vo} + \mathrm{F}_{mo} + \mathrm{F}_{\mathrm{go}} = \mathrm{F}_{\mathrm{p}} \\ \mathrm{F}_{vv} * b_1 + \mathrm{F}_{mv} b_2 + \mathrm{F}_{mo} b_4 + \mathrm{F}_{vo} b_3 = \mathrm{F}_{\mathrm{p}} b_5 \end{cases} \tag{3.5.1}$$

Carrying out the equations, it results that the applied horizontal force is equal to about 0.432N and the vertical one to 4.15 N per arm. It can therefore be stated that the grabbing system conceived is able to support the weight of the rover.

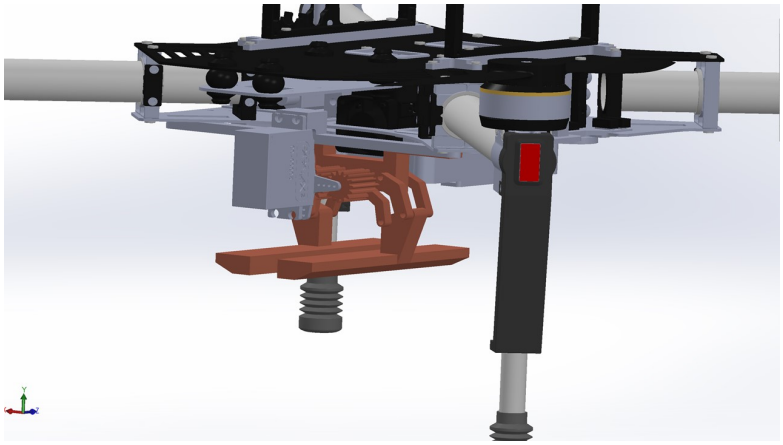

Figure 3.36: drone + Grab part

The figure shown above shows the rover grabbing system.

Chapter 4

Mission Demo

In the previous chapters the software implementation for the creation of an inspection mission has been shown and explained. The instruments used for the project were also presented.

In this chapter the different tests performed will be shown and the data acquired during the test missions will be commented. In particular we will analyze the data of the drone movement, during the rover search mission and the image acquisition and estimation data of the rover position.

4.1 Scenarios

The testing of a newly developed UAV application is always a risky task. This is why it is important to carry out the tests in a controlled environment. The presented tests have been performed in two different places, a dry river where the first parts of the development of the control program were tested. To reduce damage in the event of a fall, IIT parking has been choosen for the final stages. Other tests have been conducted in simulation. A fail-safe pilot has been present in every test, ready to take back control of M100 in case of any unexpected behavior.

Figure 4.1: River

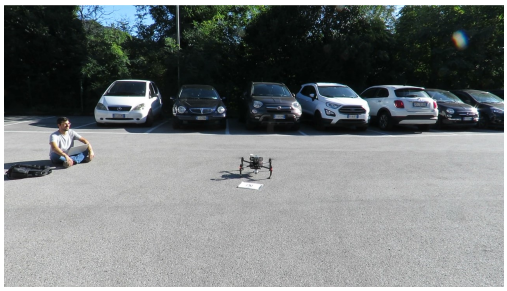

Figure 4.2: parking

Test simulations were performed through the DJI Assistant and Gazebo program. The first creates a simulation environment, for drone movement tests. The second takes the data from the simulated sensors of the DJI-Assistant and creates an environment where it is possible to add the visual marker for the landing test.

For the development of the Gazebo environment, the work done by caochao39 [16] has been taken as the starting point. This was the first environment to test the drone's movements. The cad file of the drone has been taken from GRABCad. The grab system was added later to this model. A camera has been inserted under the drone to simulate the camera connected to the raspberry. The drone control system of the previously mentioned project has been taken as basis.

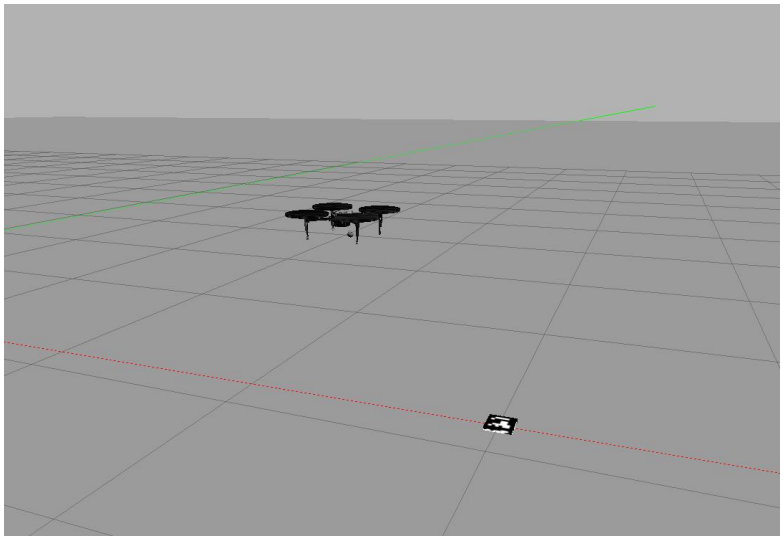

Figure 4.3: Gazebo Enviroment

4.2 Tests and results

For the tests, the mission has been divided into three parts: the first is the drone movement, the second is the identification of the ArUco marker and the third the precision landing.

The first part can be splitted into the following steps:

- The drone takes off, flies for 6 meters and lands;
- The drone takes off, completes the trajectory planning chosen for the search mission of the rover and lands..

To understand the usefulness of the developed control systems and to have a comparison with the tests carried out inside the gazebo, the graphs of the movement without controls, the movement with the PD controller and the movement with controller and planning trajectory will be shown. The tests are performed with a maximum speed of 0.2 m/s and acceleration of 0.1 m/s^2

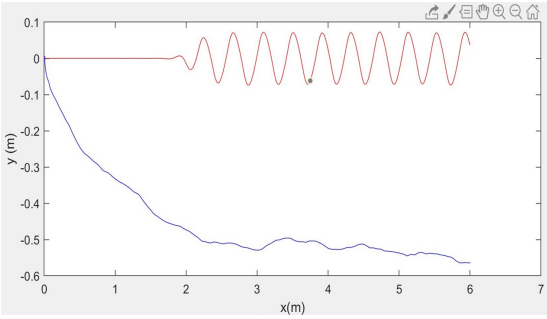

Figure 4.4: Movement without any controller

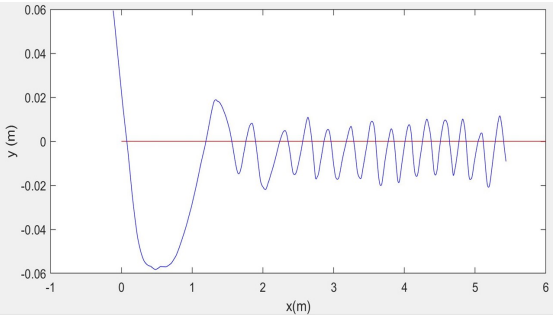

Figure 4.5: movement with PD

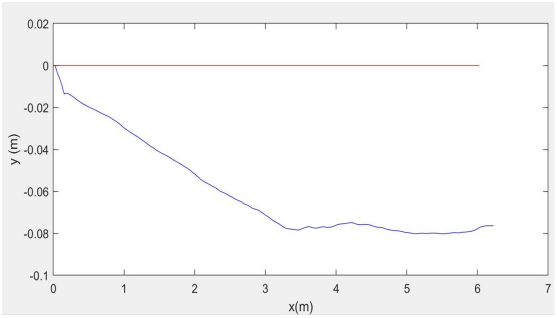

Figure 4.6: movement with PD and trajectory planning

In the graphs above, the data of the coordinates of the drone movement, which has executed a path of 6 meters along the x-axis, are reported. In particular, the red line

represents the coordinates of the drone in Gazebo, the blue line the drone during the physical test.

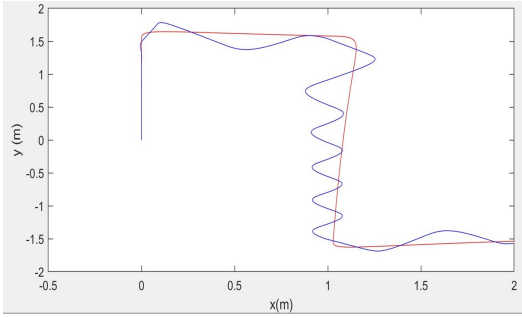

Figure 4.7: Mission without any controller

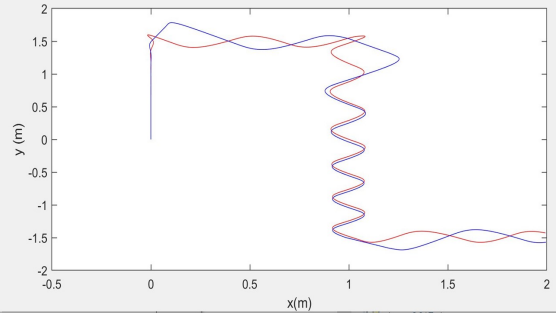

Figure 4.8: Mission with PD

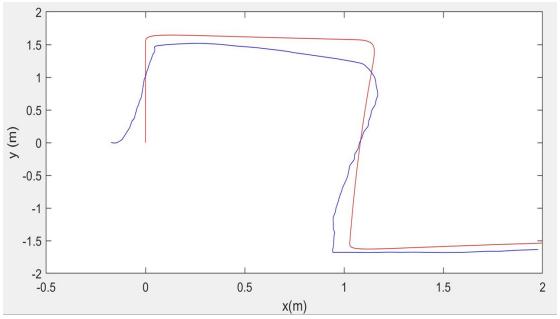

Figure 4.9: Mission with PD and trajectory planning

The graphs below show the drone's movements during the mission (black), in simulation (red) and during the tests (blue).

As it is shown in the graphs, we can see a partial improvement by switching to the PD control. The oscillation is due to a sampling not suitable for the controller. The sampling is then regulated by the trajectory planner which estimates the different positions of the drone to reach the desired position. In tests with complete control, we note a feature similar to the simulated one, with a slight deviation caused by wind speed higher than the velocity of the drone.

The second part refers to the identification of the marker. During the research phase, the camera identifies the marker, finds the reference axes and sends the data to the Kalman filter which estimates the data between one frame and another.

In this experiment, a stationary camera and a moving visual marker have been used.

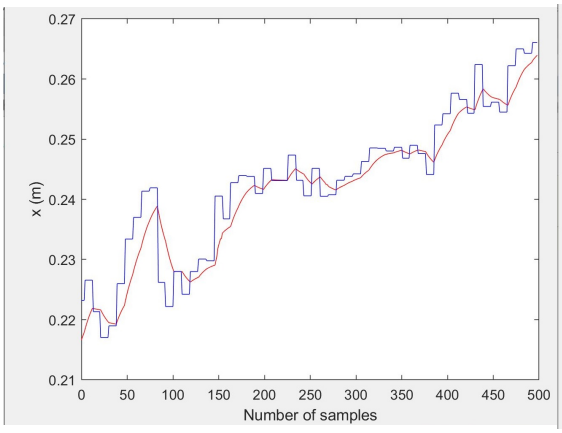

Figure 4.10: Marker identification and signal filtered in x axis

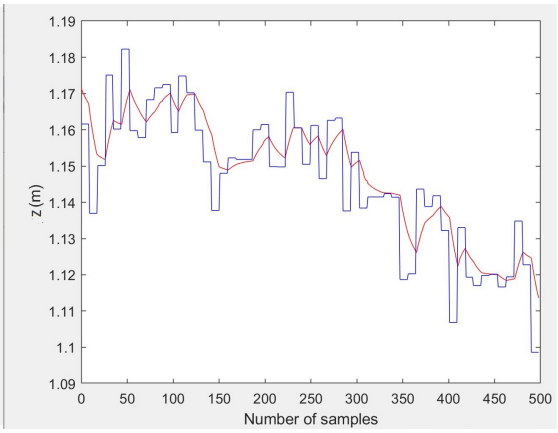

Figure 4.11: Marker identification and signal filtered on z axis

In the graphs above, the identification data of the marker (blue) and the predicted and filtered data from the Kalman (red) are shown. As can be seen on the graphs, the marker position signals are updated every second, but thanks to the implementation of the Kalman filter the data are predicted 10 times faster. The Kalman also reduces systematic errors caused by disturbances.

The third part of the test is the precision landing that can be split into the following steps:

- Placement of the Drone within the truncated cone of the landing: here the drone enters the solid figure mentioned above to position itself where the marker is included in the camera frame;
- The positioning of the UAV in the lower base of the truncated cone: the drone having centres itself with the marker, careful that the rover completed the centering, reducing the error of alignment of the z-axes;
- Return to the starting point: the drone clings to the rover and brings it back.

Chapter 5

Conclusion

As conclusion of this document, all the developed work and achieved objectives is briefly summarize in the next section. Finally, a series of potential improvements and worklines are outlined in the last section.

5.1 work done

This thesis work was undertaken for the purpose of design a multi-robots system, consisting of a drone and a rover capable of performing inspections. The state of the art has been made on macro-topics that included the individual components of the project. Projects were evaluated on the development of an UAV system in the field of inspections, on grasping systems connected on air vehicles, on inspection rovers and on the intelligence of multi-agents systems.

In this project several different technologies were involved. On one hand, the drone stabilization through trajectory planning and PD control, landing marker detection and positioning can be found. These tasks have been achieved by means of the search for the drone transfer function in order to implement the appropriate control system. Various estimation methods have been tested, the choice fell on the ARX model since it is the one that respects best system features. The use of the trajectory planning has allowed a greater precision on the displacement of the drone in the search phase of the rover.

The image processing, to determine the ArUco position in the 2D image coordinate system, was achieved using a PnP estimator. The coordinates are then processed by a filter that estimates the positions between one acquisition and another and reduces noise.

The Kalman filter data is finally combined with the drone position data to calculate the relative positions between the rover and the drone.

On the other hand, a rover designed for the display of hard-to-reach places and the creation of a multi-agent system to reduce landing errors.

The transport of the rover is made possible by a gripping system positioned under the UAV which, in the case of a completed mission, will retrieve the rover

5.2 Possible uses

The potential uses that this system can have, in this level of development, are outdoor visual inspection missions.

Thanks to the drone's ability to recover the rover, the multi-agent system can carry out missions in places that are not easy to reach by humans. This system can also be used to inspect industrial aircraft such as construction sites, container areas, and naval ports. The rover's small size allows it to fit into tight places.

The data of the various development phases of the project were then shown and commented on.

5.3 Possible future developments

The possible future work that can be done, mainly serves to increase the flight safety of the drone and the possibility of making it fly in environments with air obstacles, but also of being able to have a robot able to perform actions within the environment in which it is located.

Specifically the improvements can be about:

- The use of a DJI GUIDANCE SDK, in order to have proximity sensors and additional cameras for the analysis of the surrounding environment in order to avoid any obstacles;

Figure 5.1: GUIDANCE kit

- The use of the open CV SIFT library in order to be able to recognize the rover directly. It was not used in this project due to hardware limitations;

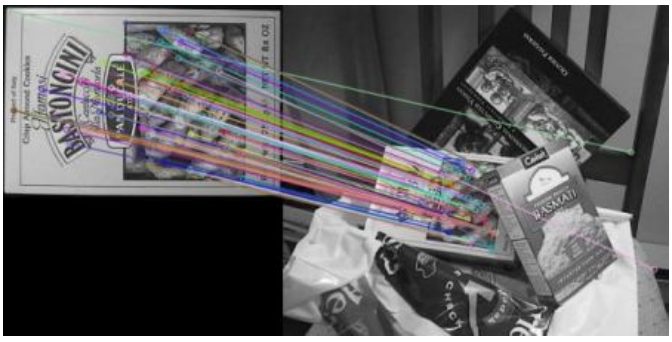

Figure 5.2: SIFT example

- Use of an octocopter, more stable compared to a quadcopter and with a greater payload. In this way it is possible to increase the rover's development;

Figure 5.3: eight propellers drone

- The possibility of inserting modules on the rover to be able to change the inspection based on environmental conditions.

References

[1] Python. https://www.python.org/.

[2] Robot operating system. https://www.ros.org/.

[3] Opencv library. https://docs.opencv.org.

[4] Serge A. Wich Lian Pin Koh. Dawn of drone ecology: Low-cost autonomous aerial vehicles for conservation. 5(2), june. 2012.

[5] Libardo Navia Vela. Design and implementation of a development platform for indoor quadrotor flight control. *master thesis*, 09 2018.

[6] Inkyu Sa, Mina Samir Kamel, Raghav Khanna, Marija Popovic, Juan Nieto, and Roland Siegwart. Dynamic system identification, and control for a cost effective open-source vtol mav. 01 2017. doi: 10.1007/978-3-319-67361-5_39.

[7] Xiaolong Hui, Jiang Bian, Xiaoguang Zhao, and Min Tan. Vision-based autonomous navigation approach for unmanned aerial vehicle transmission-line inspection. *International Journal of Advanced Robotic Systems*, 15(1):1729881417752821, 2018. doi: 10.1177/1729881417752821. URL https://doi.org/10.1177/1729881417752821.

[8] W. Kong, D. Zhou, D. Zhang, and J. Zhang. Vision-based autonomous landing system for unmanned aerial vehicle: A survey. pages 1–8, Sep. 2014. doi: 10.1109/MFI.2014.6997750.

[9] X. Zhao, F. Pu, Z. Wang, H. Chen, and Z. Xu. Detection, tracking, and geolocation of moving vehicle from uav using monocular camera. *IEEE Access*, 7:101160–101170, 2019. doi: 10.1109/ACCESS.2019.2929760.

[10] Pauline Pounds, Daniel Bersak, and Aaron Dollar. Grasping from the air: Hovering

capture and load stability. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2491 – 2498, 06 2011. doi: 10.1109/ICRA.2011.5980314.

[11] Spencer Backus and Aaron Dollar. Design optimization of a prismatic-revolute-revolute joint hand for grasping from unconstrained vehicles. page V05BT08A002, 08 2017. doi: 10.1115/DETC2017-67222.

[12] Love Kalra and Jason Gu. An autonomous self contained wall climbing robot for non-destructive inspection of above-ground storage tanks. *Industrial Robot: An International Journal*, 34:122–127, 03 2007. doi: 10.1108/01439910710727469.

[13] T. C. Lueth and T. Laengle. Task description, decomposition, and allocation in a distributed autonomous multi-agent robot system. 3:1516–1523 vol.3, Sep. 1994. doi: 10.1109/IROS.1994.407653.

[14] Jeong-Hoon Kim, Hyun-Sik Shim, H.-S Kim, M.-J Jung, I.-H Choi, and J.-O Kim. A cooperative multi-agent system and its real time application to robot soccer. pages 638 – 643 vol.1, 05 1997. doi: 10.1109/ROBOT.1997.620108.

[15] Shikhargupta. Robot operating system framework for autonomous landing of miniature uav in non-gps mode. https://github.com/Shikhargupta/vision-based-autonomous-UAV, 2016.

[16] caochao39. hku_m100_gazebo. https://github.com/caochao39/hku_m100_gazebor, 2016.

Appendices

Appendix A

Trajectory planning

```
from dji_sdk.dji_drone import DJIDrone
import math

class trajectory(object):
def __init__(self,Ts,v_max,a_max_p,a_max_n):
self.sample = 0
self.Ts = Ts
self.v_max = v_max
self.a_max_p = a_max_p
self.a_max_n = a_max_n



def plan(self,pi,pf):

result = []
tf = 1/self.v_max+0.5*(self.v_max/self.a_max_p+self.v_max/self.a_max_n)
t1 = self.v_max/self.a_max_p
t2 = tf - self.v_max/self.a_max_n
N = int(tf/self.Ts)
s1 = 0.5*self.a_max_p*(t1)**2
s2 = self.v_max*(t2-t1)+s1
k1 = int(int(t1/self.Ts))
k2 = int(int(t2/self.Ts))

if 1/self.v_max > 0.5*(self.v_max/self.a_max_p+self.v_max/self.a_max_n):
for k in range (0,N):
if k < k1:
acc = self.a_max_p
vel = acc * k * self.Ts
sp = 0.5 * acc * k**2 * self.Ts **2
```

```
if k >= k1 and k < k2:

acc = 0
vel = self.v_max
sp = self.v_max * (k - k1) * self.Ts + s1


if k >= k2:
acc = -self.a_max_n
vel = acc * ( k -k2 ) * self.Ts + self.v_max
sp = 0.5 * acc * ( k -k2 ) **2 * self.Ts **2+ self.v_max * self.Ts * ( k -k2 ) + s2
p = (pf-pi)*sp
result.append(p)
else:
sp_hat = math.sqrt(2*(self.a_max_p*self.a_max_p)/(self.a_max_p+self.a_max_n))
t_hat = sp_hat/self.a_max_p
k_hat = int(t_hat/self.Ts)
for k in range (0,N):
if k < k_hat:
acc = self.a_max_p
vel = acc * k * self.Ts
sp = 0.5 * acc * k**2 * self.Ts **2

if k ==k_hat:

sp = sp_hat


if k > k_hat:
acc = -self.a_max_n
vel = acc * ( k -k2 ) * self.Ts + self.v_max
sp = 0.5 * acc * ( k -k2 ) **2 * self.Ts **2+ self.v_max * self.Ts * ( k -k2 ) + s2
p = (pf-pi)*sp
result.append(p)
return result
```

Appendix B

Vision Algorithm

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String
from sensor_msgs.msg import Image
import pandas as pd
import picamera
import io
import numpy as np
import cv2
import cv2.aruco as aruco
import glob
from imutils import paths
import ast
import math
from cv_bridge import CvBridge, CvBridgeError
import json
import time
from picamera.array import PiRGBArray
from picamera import PiCamera



def isRotationMatrix(R):
    Rt = np.transpose(R)
    shouldBeIdentity = np.dot(Rt, R)
    I = np.identity(3, dtype=R.dtype)
    n = np.linalg.norm(I - shouldBeIdentity)
    return n < 1e-6


# Calculates rotation matrix to euler angles
```

```
# The result is the same as MATLAB except the order
# of the euler angles ( x and z are swapped ).
def rotationMatrixToEulerAngles(R):
    assert (isRotationMatrix(R))

    sy = math.sqrt(R[0, 0] * R[0, 0] + R[1, 0] * R[1, 0])

    singular = sy < 1e-6

    if not singular:
        x = math.atan2(R[2, 1], R[2, 2])
        y = math.atan2(-R[2, 0], sy)
        z = math.atan2(R[1, 0], R[0, 0])
    else:
        x = math.atan2(-R[1, 2], R[1, 1])
        y = math.atan2(-R[2, 0], sy)
        z = 0

    return np.array([x, y, z])


class Camera(object):
def __init__(self):
self.result = cv2.imread(
'/home/ubuntu/catkin_ws_py/src/vision-based-autonomous-UAV-master/src/dji_sdk_demo/script/cam.jp
rospy.init_node('camera_p', anonymous=True)
rospy.Subscriber('/dji_sdk/raspicam/image_raw', Image, self.callback)
rospy.Subscriber('/landflag', String, self.callback2)
self.fg = {}
self.bnn = time.time()
self.cnt = 0
Ts = 0.01
self.Rate = rospy.Rate(1/Ts) # 10hz
def callback(self,msg):
bridge = CvBridge()
self.result = bridge.imgmsg_to_cv2(msg, "bgr8")
def callback2(self,data):
self.fg = data.data


def Camera_pub(self):
self.ann = time.time()

pub = rospy.Publisher('camera_c', String, queue_size=1)


c = 0
```

```
rat = 0
while not rospy.is_shutdown():

stream = io.BytesIO()
count =0

dist_cam = None
id_to_find  = 1

marker_size  = 9.4#- [cm]
#marker_size  = 10#- [cm]

with picamera.PiCamera() as camera:
camera.iso = 100
camera.exposure_mode = 'auto'
camera.shutter_speed = 20000 #800
camera.resolution = (640,480)
camera.capture(stream, format='jpeg')
# Capture frame-by-frame
# Construct a numpy array from the stream
data = np.fromstring(stream.getvalue(), dtype=np.uint8)
# "Decode" the image from the array, preserving colour
frame = cv2.imdecode(data, 1)

#frame = self.result
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
#cv2.imwrite(os.path.join(path), gray)
# set dictionary size depending on the aruco marker selected
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
# detector parameters can be set here (List of detection parameters[3])
parameters = aruco.DetectorParameters_create()
parameters.adaptiveThreshConstant = 10
# lists of ids and the corners belonging to each id
corners, ids, rejectedImgPoints = aruco.detectMarkers(gray, aruco_dict, parameters=parameters)
#frame_markers = aruco.drawDetectedMarkers(frame.copy(), corners, ids)
# font for displaying text (below)
#font = cv2.FONT_HERSHEY_SIMPLEX

#calib = ast.literal_eval(calib)
mtx = np.asarray(
[[540.1502894219647, 0.0, 313.10312605545727],
 [0.0, 540.3089130651056, 245.5699095507961],
 [0.0, 0.0, 1.0]])
dist = np.asarray(
[[0.06696980246005554, 2.253624062354743, 0.0028214801062855363,
-0.013759554174996028, -10.866621622723056]])
'''
```

```
# check if the ids list is not empty
# if no check is added the code will crash
#Gazebo
mtx = np.asarray(
[[1.019099074177694320e+03,0.000000000000000000e+00,6.557727729771451095e+02],
[0.000000000000000000e+00,1.011927236550148677e+03,
3.816077913964442700e+02], [0.0, 0.0, 1.0]])
dist = np.asarray( [[2.57678460515330443 0e-01,-1.300640184051879311e+00,
-4.285777480424158084e-03,-2.507657388926626523e-03,
2.307018624520866812e+00]])
'''
flag = 0
dist_cam = None
print ids
if ids == None:
result =
str({"x":0,"y":0,"z":0,"count":0,"new_data":c,
"alpha":0,"beta":0,"gamma":0,"rate":rat})
pub.publish(result)

if ids is not None:
#print 'find it'
for i in range(0,len(ids)):
if ids[i]== 1:
a =i
flag = 1

if ids is not None and flag == 1:
c+=1
count = 1
ret = aruco.estimatePoseSingleMarkers(corners, marker_size, mtx, dist)
#— Unpack the output, get only the first
rvec, tvec = ret[0][a,0,:], ret[1][a,0,:]
result =
str({"x":tvec[0]/100,"y":-tvec[1]/100,"z":tvec[2]/100,
"count":count,"alpha":rvec[0],"beta":rvec[1],"gamma":rvec[2]})
pub.publish(result)

self.cnn = self.bnn-self.ann
print self.cnn
self.bnn=self.ann
rat+=1
self.Rate.sleep()

if __name__ == '__main__':
a = Camera()
try:
```

```
a.Camera_pub()
except rospy.ROSInterruptException:
pass
```