Planning and Control of an autonomous UAV for large-scale photovoltaic plants inspection

Università
di Genova

Fabio Conti

DIBRIS - Department of Computer Science, Bioengineering, Robotics and System Engineering

University of Genova

Supervisors: A. Sgorbissa, C. T. Recchiuto
Co-supervisor: G. Mottola, J. Callà

In partial fulfillment of the requirements for the degree of

Master's degree thesis in robotics

October 26th, 2023

Acknowledgements

I'd like to thank my thesis supervisor Prof. Antonio Sgorbissa, for his assistance during the entire project. His expertise and insightful feedback have been invaluable to the development of this thesis.

I'd also want to thank Dr. Jacopo Callà of JPDroni Srl., my corporate supervisor, for allowing to work on this project with the best possible equipment, as well as for his mentoring and practical insights. His expertise and outstanding feedback on the practical aspects of the drone-based inspection procedures have substantially expanded my understanding of the entire UAV based inspection operation.

My co-supervisor, Giovanni Mottola, deserves special recognition and heartfelt thanks for his outstanding patience and perseverance in assisting me in overcoming the hurdles encountered during the course of this thesis. Giovanni's direction and consistent support were critical in creating the final outcome of this project.

I would like to express my deep appreciation to my girlfriend, Alice, for her support and understanding, despite the physical distance between us during this academic year spent abroad. Her encouragement has been a constant source of motivation.

I would like to extend my sincere thanks to my colleagues Alessandro, Francesco, Luca, and Matteo for the great team work and sincere friendship that brought us together to the end of our challenging academic journey.

A special thanks goes to my life long friends Andrea, Filippo, Giacomo, and Irene. Together, we have experienced crucial life moments and taken decisions that contributed to both my academic and, more significantly, my personal growth.

Furthermore, I would like to thank my entire family for their patience and support shown throughout my academic journey up until now. A special thanks goes to my mother who, with great commitment and dedication, has helped me overcome numerous challenges encountered in my studies and in various aspects of life, both academic and beyond.

"Just as a flower blooms after enduring the harsh winter cold, a dream can only come true if one is prepared to endure the hardships that accompany its realization and make all the necessary efforts."
Daisaku Ikeda

Abstract

Unmanned Aerial Vehicles (UAVs), or drones, are a recent technology that have proven to be valuable in various professional contexts despite some limitations, particularly with regards to battery life. The scientific community expects that their use in daily life will become increasingly prevalent in the future. Research is currently advancing in multiple directions, with a focus on further developing their autonomous flying ability, which is a complex issue in the field of robotics. Even though they are still in the early stages of development, autonomous flying drones are already used in a wide range of fields, from the military to agriculture and commerce. This thesis explores a potential use for a UAV platform in the context of a specific inspection task. The study delves into the use of drones for autonomous monitoring and inspection of solar power installations by following pre-determined paths. The thesis examines this application in depth, breaking it down into two main sections. The first part is dedicated to finding the most efficient path for the drone to cover the entire solar plant by using satellite images and Machine Learning (ML) and geo-localization techniques to geolocalize the facility and determine the best path to cover it. The second portion is focused on identifying the optimal method of controlling the UAV to efficiently gather data along the designated path. Therefore, it will focus on the development of a visual based algorithm that uses visual-servoing control techniques and ML based features extraction to localize and detect solar panels, allowing the drone to safely follow the predetermined path. Simulations, in-lab tests validate the performance of the suggested strategies and finally, the algorithm will be tested in a real word environment.

Keywords: Machine Learning, Solar Plants, Unmanned Aerial Vehicles, Visual Serovoing.

Contents

Acronyms **xiv**

1 Introduction **1**
1.1 JP Droni . . . 1
1.2 Background . . . 1
1.3 Problem statement . . . 2
1.3.1 Planning the route . . . 3
1.3.1.1 Panel lines recognition . . . 4
1.3.1.2 Path calculation . . . 5
1.3.2 Autonomous flight . . . 6
1.4 Contents . . . 7

2 State of the art **9**
2.1 Drone classification . . . 9
2.1.1 Multirotors . . . 10
2.1.2 Fixed-wing drones . . . 10
2.1.2.1 Hybrid drones . . . 11
2.2 Quadcopter regulation . . . 12
2.2.1 Categories of certificates . . . 12
2.2.2 Open category . . . 13
2.2.3 Operational steps . . . 14
2.3 Applications for autonomous UAVs . . . 14
2.3.1 UAV-based last mile delivery . . . 14
2.3.2 Security and surveillance . . . 15
2.3.3 Search and rescue . . . 15
2.3.4 Agriculture monitoring . . . 16
2.4 Introduction to ML . . . 16
2.5 Artificial neural networks . . . 17
2.5.1 Weights . . . 18
2.5.2 Activation function . . . 18
2.6 Convolutional neural networks . . . 19

2.7 Segmentation . . . 20
2.8 Applications of NNs for PV plants . . . 21
2.8.1 FCN approach . . . 21
2.8.1.1 Model setup . . . 22
2.8.1.2 Results . . . 22
2.8.2 Semantic-Segmentation-Based approach . . . 22
2.8.2.1 DL server for segmentation . . . 22
2.8.2.2 Results . . . 24
2.8.3 U-net for satellite image segmentation . . . 24
2.9 Path identification . . . 24
2.9.1 TSP . . . 25
2.9.1.1 Historical background . . . 25
2.9.1.2 Definition and types of solutions . . . 25
2.10 CPP for PV plants . . . 26
2.10.1 SPP and DPP . . . 27
2.10.2 Cell-based CPP algorithms . . . 28
2.10.2.1 GBWC . . . 28
2.10.2.2 BECD . . . 28
2.10.2.3 GBSTC . . . 29
2.11 Drone flight and control . . . 29
2.11.1 Flight forces . . . 30
2.11.2 Quadrotor modeling . . . 30
2.11.3 Frames and states of the quadrotor . . . 31
2.11.4 Geometric and kinematic models . . . 32
2.11.5 Dynamic model . . . 32
2.11.6 Control model . . . 33
2.12 Autonomous flight for PV plant inspection . . . 33
2.13 Vision-based inspection strategy . . . 34
2.13.1 Lines and slope detection . . . 35
2.13.1.1 Lines . . . 35
2.13.1.2 Slopes . . . 35
2.13.2 UAV velocity controller . . . 36
2.13.2.1 Tracking procedure . . . 36
2.13.2.2 Identification of the end of a strip . . . 37
2.14 Combination of tasks . . . 37
2.14.1 Extended Jacobian . . . 37
2.15 Flight altitude . . . 39
2.15.1 Scale determination . . . 39
2.15.2 Experiments setup and results . . . 39
2.16 Post processing IM . . . 40
2.16.1 IM techniques . . . 40

2.17 Defects detection 41

3 Software and hardware architecture **43**
3.1 Hardware requirements 43
3.2 Software requirements 44
3.2.1 PC 44
3.2.1.1 Route planning 45
3.2.1.2 Bridge application 46
3.2.1.3 UAV Control 46
3.2.2 Android mobile phone 47
3.3 ROS architecture and interfaces 47
3.3.1 Interfaces description 47
3.3.2 UAV connection module inner interfaces 48
3.3.3 Drone control inner interface 49
3.4 Simulated interfaces and drone 50

4 Bridge application **52**
4.1 Application development 52
4.2 Application improvements 53
4.2.1 Tilt gimbal angle data transmission 54
4.2.2 Flight across rows of panels 54
4.2.2.1 Indoor flight 54
4.2.2.2 Outdoor flight 54
4.2.2.3 GPS position errors 55

5 Planning the flight route **57**
5.1 Current solution to the problem 57
5.2 Automatic solution to the problem 58
5.2.1 Satellite image acquisition 60
5.2.1.1 Coordinates selection 60
5.2.1.2 Image extraction 60
5.3 Deep Learning Instance Segmentation 61
5.3.1 YOLO 61
5.3.1.1 Object Detection Data-set 62
5.3.2 Detectron2 63
5.3.2.1 FPN + PointRend: Instance Segmentation . . . 64
5.3.2.2 Instance Segmentation Data-set 65
5.4 Image processing 66
5.4.1 First OpenCV processing 66
5.4.1.1 FindContour 66
5.4.1.2 MinAreaRect 67

5.4.2 Filtering and Standardizing . . . 67
5.4.3 OpenCV WP pixel coordinates identification . . . 68
5.4.3.1 BoxPoints . . . 68
5.4.3.2 WP identification . . . 68
5.4.3.3 Ordering the WPs within the panel lines . . . 70
5.4.4 Post-processing step: geo-localization . . . 72
5.5 Path optimization process . . . 73
5.5.1 TSP formulation . . . 73
5.5.1.1 Problem Formulation . . . 74
5.5.1.2 Problem constraints . . . 76
5.5.1.3 Additional solution methods . . . 79
5.5.2 Input data . . . 79
5.5.3 Output data . . . 79

6 UAV flight control **82**
6.1 Features extraction . . . 82
6.1.1 Detectron2 Panels segmentation . . . 83
6.1.1.1 Data-set images and annotation . . . 83
6.1.1.2 Model fine-tuning . . . 84
6.1.1.3 Feature extraction . . . 85
6.2 PID controller . . . 86
6.2.1 Feature interpretation . . . 86
6.2.2 Controller definition . . . 87
6.3 Non linear controller . . . 87
6.3.1 Feature interpretation . . . 87
6.3.2 Controller definition . . . 89
6.3.2.1 Reference system and Kinematic equations . . . 89
6.3.2.2 Proof of stability using Lyapunov criterion . . . 89
6.4 Task Stacking . . . 91
6.4.1 Pinhole model and camera calibration . . . 91
6.4.2 Feature interpretation . . . 91
6.4.3 Controller definition . . . 92
6.4.3.1 Interaction matrix . . . 93
6.4.3.2 UAV camera motion constraint . . . 94
6.4.3.3 Task based control and minimization problem . . . 95
6.4.3.4 Task stacking . . . 96
6.4.4 Inclined camera setting . . . 97
6.5 Across the panel indoor movement . . . 98
6.5.1 Trajectory controller . . . 99
6.5.2 Simple PID position based controller . . . 99

7 Experiments **101**
7.1 Semantic Segmentation evaluation . . . 101
7.1.1 Satellite images segmentation . . . 102
7.1.2 Aerial images segmentation . . . 102
7.2 Path optimization test . . . 103
7.2.1 Fixed time evaluation . . . 103
7.2.2 Optimal solution time over different maps . . . 104
7.2.3 Variation of path length given different time & iterations limits . . . 105
7.3 Control algorithm tests . . . 106
7.3.1 Simulated environment set-up . . . 107
7.3.2 Indoor test-room set-up . . . 108
7.3.3 Outdoor test set-up . . . 109
7.3.4 Simulation Tests . . . 110
7.3.4.1 Straight configuration . . . 110
7.3.4.2 Planar Rotation . . . 111
7.3.4.3 Vertical Rotation . . . 112
7.3.5 Indoor Tests . . . 113
7.3.5.1 PID controller; [Video Test 5] . . . 114
7.3.5.2 Lyapunov controller; [Video Test 5] . . . 115
7.3.5.3 Task Stacking controller; [Video Test 6] . . . 118
7.3.5.4 Angled panel following [Video Test] . . . 119
7.3.5.5 PID position based controller . . . 120
7.3.5.6 Across the panel movement over multiple panels [Video Test] . . . 121
7.3.6 Outdoor Tests . . . 123
7.3.6.1 20.5m / Aligned start / Non linear control [Video Test] . . . 123
7.3.6.2 10.5m / Aligned start / Non linear control [Video Test] . . . 124
7.3.6.3 23.8m / Angular off-set / Non linear control [Video Test] . . . 125
7.3.6.4 22m / Linear off-set / Non linear control [Video Test] . . . 126
7.3.6.5 20m / Linear off-set / Task Stacking [Video Test] 127
7.4 Links . . . 128

8 Possible improvements and conclusions **129**
8.1 Possible improvements . . . 129
8.1.1 Improving NN segmentation from satellite images . . . 129
8.1.2 Improving CPLEX optimization problem definition . . . 130

8.1.2.1 Constraint redefinition 130
8.1.2.2 Sub-tour embedded solutions 130
8.1.2.3 Human-based path finding tests 131
8.1.3 Pan-tilt gimbal control 131
8.2 Conclusions 132

A Hardware used **133**
A.1 DJI Mavic Pro: specifications 133
A.2 HP Omen computer 134

References **144**

List of Figures

1.1 Picture of a drone inspecting a PV plant taken during a test in Predosa (AL). 2

2.1 DJI models used in this thesis. 11
2.2 Fixed-wing UAV models. 11
2.3 YANGDA Nimbus VTOL Fixed Wing Drone [1]. 12
2.4 a: *supervised learning* of a CNN for the instance segmentation of a set of containers from areal images (Chap. 5). b: *unsupervised learning* for the K-means clustering of groups of solar panel rows (Chap. 5). c: *reinforcement learning* approach for autonomous UAV flight through gates [2]. 17
2.5 Schematic of a feed-forward NN. 18
2.6 A simple example of 2D convolution, with a 2×2 kernel and padding. 19
2.7 A simple example of pooling with a 2×2 filter. 20
2.8 Semantic segmentation for the identification of defects in PV panels [3]. 20
2.9 Example images from the Amir database [4]. 21
2.10 Schematic of the data processing method in [5]. 23
2.11 U-Net network structure [5]. 23
2.12 Different steps of the SPP algorithm [6]. 28
2.13 *.a* The steps for the BECD algorithm [7]. *.b*: the GBSTC algorithm [8]. *.c*: the GBWC algorithm [9]. 29
2.14 The quadrotor concept. The width of the arrows is proportional to the angular speed of the propellers [10]. 31
2.15 Quadrotor reference frame [10] 31
2.16 Schematic of the cascade control law for quadrotors. 34
2.17 Image pre-processing according to the steps in [11]. 36
2.18 Visual representation of a function h_i, from [12] 38
2.19 Example of IM, showing sidelap and endlap [13]. 41
2.20 Image of an actual hot-spot from the Pedrosa (AL) solar plant. . 42

3.1 Interfaces between the different hardware components. 44
3.2 High-level UML graph of the interfaces. 48
3.3 UML graph of the socket connection interfaces. 49
3.4 UML graph of the controller interfaces. 50
3.5 The quadrotor model. 51
3.6 Simulation environment. 51

4.1 Schematic of an inspection, with the drone velocity (in teal), camera stream (magenta) and altitude (red) from the proximity sensor and barometer 53
4.2 In teal: the WP-based movement across the panel. 55

5.1 ROS node which processes the image. 57
5.2 Pictorial summary of the image processing carried out by the module. 58
5.3 UML Activity Diagram of the node. 59
5.4 *.a*: *YOLOv8* Bounding Boxes;*.b*: *YOLOv5* Oriented Bounding Boxes. 62
5.5 *PointRend* segmentation comparison [14]. 65
5.6 *Point-Rend* segmentation. 65
5.7 Oriented boxes. 68
5.8 Side by side figures 69
5.9 Side by side figures with different Dub values. 70
5.10 Side by side figures with different path directions. 71
5.11 Incomplete panel line ending. 71
5.12 *a.*: TSP solution before the application of the constraint; *.b*: After the application of the constraint. 77
5.13 *K-menas* clusterized PV plant. 77

6.1 FPN Mask R-CNN network framework [15]. 84
6.2 Filtering step of the center panel. 85
6.3 PID's feature geometric interpretation. 86
6.4 Non linear controller feature geometric interpretation. 88
6.5 Frame of the DJI UAV. 89
6.6 Camera parameters, pinhole model. 92
6.7 (ρ, θ) feature geometric interpretation. 93
6.8 Shift, geometric interpretation. 98

7.1 Average path length calculated by each of the algorithms on four different maps. 104
7.2 Comparison of the average times to reach the optimal solution, for the BR and ILS algorithms. 105
7.3 Path length over time for the BR algorithm and Path length over number of iterations for the ILS algorithm. 106

7.4 Average time variation for different numbers of iterations, for the ILS. . . . 106
7.5 Three texture combinations. . . . 107
7.6 Screenshot of a simulation for different panel setups. . . . 108
7.7 Different iteration of the handmade simulated panel. . . . 109
7.8 Different Types of ground surfaces and drone markers. . . . 109
7.9 *a.*Error from the *top-view*; *b.* Error from the *lateral-view*; *c.* Rotation error over time. . . . 110
7.10 *a.*Error from the *top-view*; *b.* Error from the *lateral-view*; *c.* Rotation error over time. . . . 111
7.11 *a.*Error from the *top-view*; *b.* Error from the *lateral-view*; *c.* Rotation error over time. . . . 113
7.12 Top/lateral view | velocity of *Optitrack* data first PID test. . . . 114
7.13 PID Gains first test set. . . . 114
7.14 PID Gains second test set. . . . 114
7.15 Top/lateral view | velocity of *Optitrack* data third test. . . . 115
7.16 Top/lateral view | velocity of *Optitrack* data fifth test. . . . 115
7.17 Top/lateral view | velocity of *Optitrack* data first Lyapunov controller test indoor. . . . 116
7.18 picture of the camera misalignment. . . . 116
7.19 Top / lateral view | velocity of *OptiTrack* data second test. . . . 117
7.20 Top / lateral view | velocity of *Optitrack* data fifth test. . . . 117
7.21 Top / lateral view | velocity of *Optitrack* data second test. . . . 119
7.22 Top / lateral view | velocity of *Optitrack* data fourth test. . . . 119
7.23 Top / lateral view | velocity of *Optitrack* data sixth test. . . . 119
7.24 Prospective, top ,lateral view of the shifted visit. . . . 120
7.25 Position based controller graph from the top and lateral view. . . . 121
7.26 Prospective,top and lateral view of the two panels visit. . . . 122
7.27 ρ, θ and z altitude values perception for the non linear controller. 123
7.28 ρ, θ and z altitude values perception for the non linear controller controller. . . . 124
7.29 ρ, θ and z altitude values perception starting from an angular misalignment position | *a.* Image drone view at t_1; *b.* Image drone view at t_2; *c.* Image drone view at t_3. . . . 125
7.30 ρ, θ and z altitude values perception starting from a linear misalignment position.|*a.* Image drone view at t_1; *b.* Image drone view at t_2; *c.* Image drone view at t_3 . . . 126
7.31 ρ, θ and z altitude values perception for the Task stacking visual based controller. . . . 127

A.1 Mavic 2 Enterprise Dual . . . 133

List of Tables

2.1 Some examples of activation functions [16]. 19
2.2 Table summarizing the different defects and faults detectable at different heights [13]. 40

3.1 Libraries and packages required for the route planning. 45
3.2 Libraries and packages required for the bridge connection 46

4.1 Socket connections. 52

5.1 JSON comands 81

7.1 Root Mean Square errors for each of the controllers in the linear track. 110
7.2 Root Mean Square errors for each of the controllers in the *Tilted* track variations. 111
7.3 Root Mean Square errors for each of the controllers in the *Up and Down* track variations. 112
7.4 PID RMS errors for each of the tests. 114
7.5 RMS errors for each of the Lyapunov controller tests. 116
7.6 RMS errors for each of the Visual servoing controller tests. 118

A.1 DJI Mavic 2 Enterprise Dual specifications 134

Acronyms

AI	Artificial Intelligence
AL	Alessandria
AMO	Adaptive Multi-scale Optimization
ANN	Artificial Neural Network
BB	Bounding Box
BECD	Boustrophedon Exact Cellular Decomposition
BGWC	Grid-Based Wavefront Coverage
BR	Bottom Right
BVLOS	Beyond Visual Line Of Sight
CNN	Convolutional Neural Network
CoM	Center of Mass
CPP	Coverage Path Planning
CV	Computer Vision
DBSCAN	Density-Based Spatial Clustering
DC	Direct Current
DM	Distance Matrix
DoF	Degree of Freedom
DL	Deep Learning
DPP	Dynamic Point algorithm
EE	Earth Engine API
ENAC	*Ente Nazionale per l'Aviazione Civile* (national institution for civil aviation)
FCN	Fully Convolutional Neural Network
FPN	Feature Pyramid Network
FPS	Frames Per Second
GA	Genetic Algorithm
GM	Geometric Model
GNSS	Global Navigation Satellite System
GPU	Graphic Processing Unit
HSV	Hue-Saturation-Value
IDE	Integrated Development Environment
ILS	Iterated Local Search
IM	Image Mosaicking
IoU	Intersection over Union
IP	Image Processing
IS	Instance Segmentation
IT	Inference Time
LS	Local Search

JPEG	Joint Photo Expert Group
JSON	Javascript Object Notation
KF	Kalman Filter
KM	Kinematic Model
KML	Key-hole Markup Language
MA	Memetic Algorithm
MAV	Multirotor Areal Vehicle
MILP	Mixed Integer Linear Programming
ML	Machine Learning
NED	North-East-Down
NN	Neural Network
PID	Proportional-Integer-Derivative
PNG	Portable Network Graphic
PSR	Photo Scale Reciprocal
RC	Radio Controller
ReLU	Rectified Linear Unit
ResNet	Residual Network
RF	Representative Fraction
RGB	Red-Green-Blue
RMS	Root Mean Square
RTK	Real-Time Kinematics
SAM	Segment Anything
SP	Solar Panel
SPP	Static Point algorithm
SW	Software
TCP	Transmission Control Protocol
TIP	Traditional Image Processing
TL	Top Left
TSP	Traveling Salesman Problem
UAS	Unmanned Aircraft System
HW	Hardware
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
UI	User Interface
VLOS	Visual Line Of Sight
VTOL	Vertical Takeoff and Landing
WLAN	Wireless Local Area Network
WP	Way-Point
YOLO	You Only Look Once
ZSL	Zero Shot Learning

Chapter 1

Introduction

1.1 JP Droni

The thesis work is carried out in partnership with *JP Droni Srl*, a high-tech company that provides drone services to a wide range of clients, including industry professionals and individuals. They specialize in providing technical inspections, agriculture 4.0 solutions, LIDAR surveys, and video production services. Based in Genoa (Italy), they operate throughout Italy and Europe, maintaining high standards of quality and safety in all their activities. They have carried out flight operations in various countries, both within and outside of Europe, and have obtained the necessary certifications for these operations. In addition to their regular business operations, JP Droni also engages in research collaborations with universities and other research institutions to advance the field of UAV technology. For more information about JP Droni, we refer to their website [17].

1.2 Background

Solar energy is a sustainable and non-depletable source of energy supply. Such resource can be a viable substitute for fossil fuels and has the potential to significantly contribute to meeting global energy demands [18]. Photovoltaic (PV) solar panels are common and clean devices to harness this kind of renewable energy from sunlight. Worldwide, large-scale solar power infrastructures are being installed every day over very wide geographical areas where solar power can be exploited in the most effective way. These Solar Power (SP) plant infrastructures can cover many square kilometers of land where thousands of panels are installed (Fig. 1.1) [19].

Companies that manage large installations understand the need for frequent routine check-ups and extensive maintenance to keep the system running efficiently.

Figure 1.1: Picture of a drone inspecting a PV plant taken during a test in Predosa (AL).

Certain defects in PV modules can be caused by factors such as solar radiation, temperature changes, and environmental conditions, such as UV exposure, temperature cycles, and chemical reactions [20]. Harsh environments like deserts, with sand, wind and dust, can greatly reduce the performance of PV modules [21]. Recently, manufacturers have created thinner film solar cells with larger surface area, which makes the cells more delicate during transportation and installation. These thinner film solar cells are also highly sensitive and fragile [22]. Due to the panels' susceptibility to damage and their limited lifespan, it is often costly and dangerous to conduct on-site inspections using human labor. This creates an opportunity for using autonomous robotic inspection systems to improve the monitoring of these installations, preventing drops in energy production [23].

1.3 Problem statement

In recent years, there has been a growing interest among researchers in developing innovative methods for detecting degradation in photovoltaic plants. One approach that has been proposed is to use indicators based on resistance, which can provide valuable insights into the condition of these systems. As part of this research, many scientists have also begun to explore the potential of unmanned technology for monitoring and maintaining these large-scale power plants. Utilizing extensive flight campaigns, scientists have been investigating the feasibility of using UAVs to conduct regular inspections of PV plants, particularly those that are located in hard-to-reach areas or are otherwise difficult to access using traditional manned methods. As PV plants are getting larger, UAVs have become the go-to technology for monitoring and inspecting power plants [24].
This work aims at creating the foundation for the development of a software architecture capable of autonomously carrying out a full monitoring session of a

solar plant. The problem can be broadly summarized into 2 macro-sections, as discussed in the following subsections.

1.3.1 Planning the route

The first problem tackled is related to the automatic definition of a proper route for the drone to follow in order to cover up the entire PV plant area given a satellite image. In the following Chapter 2 some examples related to this matter are reported. This problem is taken care of in a very similar way in all the papers regarding these concepts, as detailed below:

1. **Image boundaries definition** (Subsections 2.8.1 and 2.8.2): given a satellite image of the solar plant, the first step is to extract the boundaries of the area of interest, namely the area covered by the panels. The image will then be transformed from a generic irregular shape to polygons [11]. This result is often achieved after multiple layers of image filtering, mainly carried out by image processing programs based on the *openCV* Python library using *Traditional Image Processing* (TIP) [6, 25] or *Deep Learning* (DL) approaches [4].

2. **Path calculation algorithm**: this part implies the application of an algorithm developed for path planning. This step is usually carried out either by a standard formulation of a *Coverage Path Planning* (CPP) problem or by a custom-made algorithm where global-coordinates-based (latitude-longitude) waypoints (WPs) are defined on the basis of the perimeter previously identified as the border of the PV plant (Sec. 2.10). It is important to specify that such algorithms do not take into account neither the position and orientation of the panels nor (for some cases) the battery capacity of the UAV in use. Indeed, these algorithms are only based on the geo-localization of the plant perimeter.

The company JP Droni performs PV plant inspections with DJI drones [26]. Their DJI UAV fleet support GNSS (Global Navigation Satellite System) sensors capable of identifying the position of a drone with higher or lower accuracy depending on the model and the GNSS signal condition in a specific area. Therefore, JP Droni takes advantage of the GNSS board to manually determine the global WPs connection for identifying the final path of the drone.

Starting from satellite images of the solar plant, the WPs are selected at the left and right (or top and bottom) extremes of the panels, depending on the plant orientation. The global orientation of the PV plants is usually either horizontal (west-east) or vertical (north-south). This implies that the photovoltaic panels are organized as lines of photovoltaic modules all oriented in the same direction

[27]. Choosing the WPs at the panel extremes ensures that the drone passes right on top of the panels. This feature is an advantage for the picture reconstruction of a plant for the evaluation of the efficiency conditions of the panels. The photos taken by the drone at a certain time stamp can be directly related to a specific position on top of a specific line of panels, making it very easy to locate a potential energy production issue.

However, this method intrinsically has three main disadvantages.

1. The manual connection of the WPs does not ensure to find the shortest path, especially when considering geometrically intricate PV plants where no trivial solution exists for connecting the points.

2. It does not take into account the battery life of the UAV.

3. Manually selecting all the WPs is long and tedious, especially when analyzing PV plants with large footprints.

The novelties introduced by this thesis aim to address and resolve the limitations associated with the already-established manual method. These advancements are achieved by incorporating the key concept of the manual technique into an automated solution that exploits *Machine Learning* (ML) and *optimization algorithms* maximizing the benefits offered by the existing method while minimizing the reliance on user commands. Additionally, the proposed approach takes into account the battery levels of the drones and ensures the planning of the most efficient and shortest flight path possible.

1.3.1.1 Panel lines recognition

Recognizing panel lines is an essential step for the inspection of PV plants. To tackle the problem, an object recognition technique based on *Neural Networks* (NNs) was explored in the first place. Object detection is a technology that falls within the domains of computer vision and image processing. Its aim is to identify and locate instances of specific semantic objects which can be part of a certain class, such as humans, buildings, or cars, within digital images and videos [28]. This method can be applied exploiting the use of different neural networks capable of performing such task. The first NN used for object detection was *You Only Look Once* (YOLOv8), eighth version. This is the last version of the YOLO NN proposed by the company Ultralytics. YOLO, using an end-to-end NN, makes predictions of bounding boxes and class probabilities all at once [29]. The issue with the use of this NN is related to the kind of output produced once an object is detected. The returned mask identifying the object is just a squared box with the same orientation as the image frame. Since the problem required a much higher precision, a *Convolutional Neural Network* (CNN) for *Instance Segmentation* (IS)

was used instead. Leveraging CNNs has demonstrated to be the most promising way to precisely detect objects from satellite images. Particularly, the *Detectron2* project called *Point Rend* was employed, as it proved to be particularly suited for addressing this problem [14]. Detectron2, developed by Meta researchers, is an advanced library renowned for its exceptional capabilities in object detection and segmentation [30]. Its integration into the thesis work has greatly enhanced the accuracy and efficiency of panel lines recognition (Chap. 5).
Furthermore, post-processing *Computer Vision* (CV) techniques based on the OpenCV library have been used for the identification of the drone's WPs [31].

1.3.1.2 Path calculation

Starting from the WPs thus found, three important points had to be addressed to determine the UAV's path.

1. *Geo-localizing the WPs*: *Keyhole Markup Language* (KML) files are used as a simple *User Interface* (UI) with Google Earth. The Google Earth Pro application allows the user to introduce markup elements such as *polygons*, *polygonal lines* and *key-points* on top of the satellite maps. The KML file related to a map can then be saved separately. The user will only have to create a folder within the KML file containing a polygon shape covering up the PV site and a key-point representing the starting point. This will be the only user interaction during the operation because all the geographical information contained in the KML file will be sufficient to locate the way-points and ensure the flight plan feasibility (Sec. 5.4.3).

2. *Connecting the WPs efficiently*: the connection of the WPs is carried out as a case of the well-known optimization problem called the *Traveling Salesman Problem* (TSP). The TSP is one of the most important and most studied combinatorial problems. Applications of TSP can be found in a wide variety of fields, from manufacturing to vehicle routing to integrated circuit design. *Mixed-Integer Linear Programming* (MILP) formulations have been studied extensively for TSP, yielding extremely effective results, to the extent that advances in computing power have made it possible to solve exact TSP instances with several thousand nodes [32]. However, one does not always have the time and computational means to use such methodological and algorithmic tools. For this thesis, the problem was approached from a different angle, namely, not focusing on the search of the optimal solution but considering time-saving approximate solutions. The algorithms used in the final system thus do not necessarily offer the optimal solution to the problem. *Meta-heuristics*, such as *local search*, *Iterated Local Search*, *simulated annealing*, and *rho-approximated algorithms* (Sec. 2.9.1) are part

of this category of approaches. These schemes can be adapted to many different optimization problems and have yielded excellent practical results in recent years. A few considerations were also made regarding the search of an optimal solution.

3. *Addressing the battery constraints*: considering the battery in the planning process is a key step forward for the effectiveness of the final path. If the battery level is too low to visit all the WPs in one tour, the drone will be able to plan a *clustered* path where each cluster of WPs is guaranteed to correspond to a feasible path. The key concept for addressing this problem is the modeling of **battery draining**.

   Thanks to the highly detailed satellite pictures retrieved from Google Earth, combined with the localization process, it is possible to associate the pixels side length to a measurement in meters. Therefore, it is straightforward to associate a total distance value to the calculated path. Such value will be compared to the drone's maximum feasible distance with a fully-charged battery pack. If the drone's maximum distance is shorter than the required distance for visiting every WP in one tour, the set of WPs is clustered in two subgroups. The same process and comparison are executed for each subgroup to check its feasibility. The feasible subgroups are added to the final path permutation, while the same clustering process is iterated over the non-feasible sub-tours.

   The process of partitioning WPs into subgroups is implemented using the *K-Means* clustering algorithm (Sec. 5.5.1.2). This ML-based classification technique aims to cluster data points into n groups with equal variance, while minimizing the within-cluster sum-of-squares criterion [33].

1.3.2 Autonomous flight

Thanks to the capability of DJI UAVs to connect geo-localized WPs through *Real-Time Kinematics* (RTK) technology, the final WPs path permutation could be enough for capturing panel images. As a matter of fact, the JP Droni company performs this kind of operation during image acquisition campaigns of PV plants. When the facility is on an area with a flat and regular surface, this method is very effective as confirmed by experience. However, this technique shows some issues when considering PV installations located on top of irregular terrain surfaces. In such a setting, the panel lines may not always be as straight and regular as in the previous case, given the highs and lows of the hills or the little shifts on the panels setting due to the terrain. Such changes may prevent a drone's camera to capture a well-centered image of the panel lines, especially during low-altitude flights.

To address this issue, a new ROS controller architecture was presented (Chap. 3). The system aims at controlling the entire drone-based inspection process. It manages the travel from one panel to the other through a basic WP connection. The motion following a panel is defined through three different controllers: a *simple PID controller* (Sec. 6.2), a *Non linear controller* (Sec. 6.3), and a *visual-servoing-/task-stacking controller* (Sec. 6.4). All three algorithms are based on a visual features extraction method (based on segmentation algorithms) and on the altitude sensors data coming from the UAV drones (Sec. 6.1.1). The control is capable of flying the drone at different heights, allowing it to identify different types of features which cannot be spotted at the current standard flying heights of 15 to 20 meters (Sec. 2.15). The data coming from the drone's sensors and the velocity controls sent to the drone are based on a connection established through a *bridge architecture* developed and perfected during previous thesis work. Some extra features were added to the bridge connection to perfect the given controls (Chap. 4).

1.4 Contents

This section provides an overview of the thesis layout, highlighting the different chapters and their contents and setting up the stage for diving into various topics such as the state-of-the-art, the global architecture, planning and control, experiments, and finally, the conclusions drawn, including some thoughts about the future developments of this work.

State-of-the-art (Chap. 2): here, some context and introduction is given to each of the topics touched throughout the thesis work, such as ML and DL basics, path coverage problems and their optimization techniques, quadrotor modeling and flight control, and image reconstruction strategies. Moreover, a *scoping review* regarding the concept of this work was conducted. This type of review method is particularly beneficial when seeking to gain a comprehensive understanding of a topic that is complex or diverse, by providing a clear overview of the available evidence [34]. After analyzing 52 papers, some of the most interesting ones were reported, together with related approaches to similar topics, and then relevant concepts were adapted to our specific applications.

Software and hardware architecture (Chap. 3): describing the software and hardware architecture of the interface and the advancements with respect to previous work.

Bridge application (Chap. 4): the bridge application is briefly explained and the advancements related to the exchanged data are reported.

Planning the flight route (Chap. 5): describing the extraction and geolocalization of satellite images, the localization of the WPs and the application of DL and CV techniques for their extraction, and the optimization techniques and strategies used for connecting the WPs.

UAV flight control (Chap. 6): three controller techniques (PID, Lyapunov non linear, visual servoing and task stacking based) are explained, along with the description of the visual feature extraction.

Experiments (Chap. 7): reporting the data collected during NN training and in simulated and real flight tests.

Conclusions and possible improvements (Chap. 8): where a few hints to new developments of this project are given.

The findings and contributions of this master's thesis are presented in the following sections, which provide additional depth about each of these topics.

Chapter 2

State of the art

The focus of this chapter is to gather and evaluate evidence from multiple study designs and methods found in the published literature to contextualize and explore the state of the art regarding the different aspects of the use of UAVs for autonomous monitoring and inspection. The aim is twofold: first, to define the basis of the DL-based method to identify the optimal UAV route, also including an introduction to ML and DL and to their role for this kind of application. Moreover, an overview is conducted on the different optimization techniques used for multiple types of WP connections. Secondly, attention is given to the existing visual control techniques applicable for controlling the drone flight, and also to the general modeling and low-level control of quadrotor drones. In addition to this, a section will be dedicated to the classification of drones based on function, size, missions and other elements of interest, including some examples of monitoring applications for drones. Regulations and limitations in relation to drone flight will also be discussed in the following sections.

2.1 Drone classification

Given their adaptable form factor, simplicity of control by human operators, autonomous capabilities, and a variety of possible applications, *Multirotor Areal Vehicles* (MAVs) have been quickly established as a platform for robotics research. Drones are useful in a wide range of sectors, including surveillance and monitoring tasks [35], environmental research, search-and-rescue missions, infrastructure inspections, precision agriculture, package delivery, and film-making. They are essential tools for gathering information, carrying out tests, mapping and navigating areas, and pursuing new directions in robotics research due to their agility, maneuverability, and capability to enter difficult-to-reach spaces [36].

There are many different ways to classify MAVs, depending on different aspects

of their platform. However, the most common classification is based on their mechanical architecture. The UAV form factor can be distinguished based on the propulsion strategy. Therefore, they can be classified as *rotating-wing* (or *multirotor*) models and *fixed-wing* (or *aircraft*) models.

2.1.1 Multirotors

Multiple benefits of multirotor UAVs lead to their broad use. First, they are extremely adaptable because of their ability to do *Vertical Take-Off and Landing* (VTOL) maneuvers in tight places and difficult terrains. Aerial photography, surveillance, delivery services, and disaster response are just a few of the areas in which these UAVs are used. Because of their adaptability and scalability, they can be manufactured in a variety of sizes, to accommodate different jobs or payloads. However, they do have a few drawbacks compared to fixed-wing alternatives. Indeed, multirotor UAVs have shorter flights and lower endurance, due to the nature of their flight mechanics, which requires more energy. They have a limited operating range and need frequent battery replacements or recharging periods, which can interrupt continuous activities. Moreover, multirotors usually have multiple points of failure. Components such as motors, propellers, electronic speed controllers, and power systems are critical for flight operations. In the event of any malfunction or failure in these components, the UAV's ability to fly can be compromised. These failures require regular maintenance and redundancy measures, to ensure the safe and reliable operation of multirotor UAVs.

Quadcopter (or *quadrotors*) (Fig. 2.1a) have four rotor blades that lift the drone off the ground through propulsion. This model has four actuators located behind each propeller blade. These motors are generally brushless *Direct Current* (DC), whose velocity can be controlled using basic *Proportional-Integral-Derivative* (PID) controllers, or other more advanced control schemes [37]. The quadcopter is the best-known mechanical architecture among multirotors, but many other models have been studied as well, for example **hexacopters** (Fig. 2.1b), with 6 propeller blades.

2.1.2 Fixed-wing drones

Differently from quadcopters, fixed-wing drones resemble traditional jet planes (Fig. 2.2). These drones require a runway or a launching platform to take off and land due to their propulsion made for forward flight. Because of their effective aerodynamic design, they provide advantages over multirotor alternatives like wide ground coverage for tasks such as aerial mapping, agricultural surveys, and animal monitoring. They also have higher battery autonomy and greater flying endurance, allowing them to function for longer periods of time between battery

(a) Quadcopter used for the experiments: DJI Mavic Enterprise 2 [26] (App. A.1).

(b) Exacopter used for the experiments: DJI Matrice 600 Pro [26].

Figure 2.1: DJI models used in this thesis.

(a) Albatross UAV [38].

(b) Delair UX11 UAV [39].

Figure 2.2: Fixed-wing UAV models.

changes. However, unlike multirotor UAVs, fixed-wing drones require wider space for takeoff and landing, making them less suitable for deployment in confined or densely populated areas. The need for a runway or specialized launching devices can limit their accessibility and operational flexibility. Furthermore, their larger form factors require careful consideration when it comes to transportation and storage logistics [36].

2.1.2.1 Hybrid drones

There are VTOL-type fixed-wing drones, also known as hybrid drones, which can perform forward flight like conventional fixed-wing drones while also being able to hover at a constant height like quadcopters. For instance, the *Nimbus* model (Fig. 2.3) resembles ordinary fixed-wing drones, but has four upward-facing propellers evenly spaced throughout its four quadrants. A forward-facing propeller and the

Figure 2.3: YANGDA Nimbus VTOL Fixed Wing Drone [1].

fixed wings create some lift, easing the strain on the upward-facing propellers.

Quadrocopters, fixed-wing, and hybrid drones can all use tilting rotors, which increase adaptability by allowing for directional adjustments in rotor thrust. However, the addition of tilting motors increases the control and actuation complexity. A thorough evaluation of the trade-off between versatility and complexity must be done before deciding on whether to include these extra capabilities in any drone.

2.2 Quadcopter regulation

For the sake of completeness, the main Italian regulations in force for UAV drone flight will be presented in this section. Note that these regulations vary from country to country [40].

2.2.1 Categories of certificates

The current European Union regulation provides indications and limitations to regulate and support remotely-controllable aerial vehicles and civil aviation. The *Regulation (EU) 2019/947* and *Regulation (EU) 2019/945* [41] establish a classification of *Unmanned Aircraft System* (UAS) operations based on risk level. The Italian reference for UAS drone flight regulation is the *"Ente Nazionale per l'Aviazione Civile"* (ENAC), the national institution for civil aviation. ENAC acts as the single authority for technical regulation, certification, supervision and control in the civil aviation sector in Italy, in compliance with the powers derived from the Navigation Code [42].

There are three official categories of UAS activities, described below.

Open category: describes UAS activities that follow particular guidelines, including operating inside the pilot's *Visual Line of Sight* (VLOS) and

having a maximum take-off mass of less than 25 kg for privately-constructed aircraft. This category is the most restrictive in terms of conditions and it allows the user to fly only in predetermined circumstances.

Specific category: this is the most typical UAS operation type, including complex tasks carried out by businesses or by independent specialists. This category allows the user to fly more freely than the previous one; moreover, operators with specific certificates may make specific requests to ENAC to perform operations in places or conditions that are not normally accessible or for which no explicit regulation exists.

Certified category: involves operations which entail protracted overflights over people, the transportation of people or dangerous items, and a high level of risk that can only be reduced by certifying the aircraft, the operator, and the pilot. This category represents the highest level in the field of drone piloting.

These categories all refer to *certified* drones, which must have geo-awareness and identification functions [43]. To date, however, there are no certified UAVs, which is why ENAC has extended the status of the limited categories (until 1 January 2024), allowing the piloting of non-certified drones.

2.2.2 Open category

The open category contains three sub-levels, regulated by two licenses. The three different levels are [44]:

A1: The drone weighs less than 500 grams. This allows flight over individuals, but not over crowds.

A2: Flight close to people. A minimum horizontal distance of 50 meters should be maintained between the drone and individuals. Additionally, if the drone is operated at a height above 50 meters, a horizontal distance at least equal to the altitude must be maintained. Also, the weight of the drone may not exceed 2 kilograms.

A3: The flight is allowed with a minimum horizontal distance of 150 m from residential, commercial and industrial areas and away from people.

The first license that can be obtained is for the A1-A3 levels, and the second is for the A2. These two licenses must be obtained to reach all three levels. Nevertheless, only VLOS operations are permitted in the "open" category, while *Beyond Visual Line of Sight* (BVLOS) operations are only allowed for the "specific" and "certified" categories.

2.2.3 Operational steps

Each country has to publish maps showing the particular geographic areas where drone flights are either entirely forbidden or where operation permission is required. In most countries, phone apps are developed to help users with such matters. These apps make it easier to find legal flight zones and avoid interfering with civilians and military aircraft. To provide a dedicated, low-altitude air traffic management service for all varieties of UAVs and related operations, the *D-Flight* portal has been developed for the Italian jurisdiction [45]. This application allows the operator to register the UAV with an identification code recognizable by Italian authorities and consult an aerial map for avoiding zones where flying is forbidden.

The standards *UAS.SPEC.050* and *UAS.SPEC.060* are established by *EU Regulation 2019/947* and underline the crucial factors that operators should be aware of before engaging in safe flying operations within the "open" and "specific" categories. These criteria apply to all aerial vehicles, independently of weight, function, and control mode (namely, whether it is autonomous or teleoperated). Autonomous flying is prohibited except for research work. As reported in [46], the law provides a thorough framework outlining the integration of automated control technology within a fleet of drones. This framework ensures accountable fleet management for a variety of tasks, including control management, surveillance, and contingency assessment. This is particularly important given the growing interest in autonomous driving, as it guarantees the responsible and successful application of automated control technologies.

2.3 Applications for autonomous UAVs

Commercial UAV units have been growing in popularity for application in multiple areas, including inspection, maintenance, controlled agriculture, and security monitoring, given their reliability and ease of use. Generally, wherever manned inspection is dangerous, uncomfortable, or simply inefficient, drones are a safer, cost- and time-effective alternative. In this subsection, a few applications for autonomous UAVs are briefly exposed.

2.3.1 UAV-based last mile delivery

During the last few decades, e-commerce and retail companies have been working intensively on deploying more advanced ways to cut shipping time to customers. Many newer companies, like Google and Amazon, as well as traditional logistics providers such as UPS, USPS, DHL, and FedEx, have been experimenting with drone delivery aiming at reducing costs and providing cheaper, faster, and more

efficient service [47]. One of the most researched fields in this context is *truck-drone combined* delivery. This feature adds the possibility of launching a UAV drone capable of delivering a payload to a certain location from a typical delivery truck. The aerial vehicle coupled with a standard ground vehicle provides multiple advantages regarding mailing speed and planning [48].

Starting from this, the concept of a *Smart Ambulance* was introduced, namely a medical ambulance equipped with a UAV. This feature allows launching the drone for a last-mile delivery of medical payloads and providing additional information regarding the rescued patients, making it easier to provide appropriate supplies to promptly take care of the emergency once the ambulance gets to the rescue site [49].

2.3.2 Security and surveillance

In the military, private, and public sectors, the use of UAVs for security and surveillance has significantly increased over the years. Target following, tracking, and border control have all been investigated as potential uses for UAVs. Researchers have created resource-usage management strategies like *Adaptive Multi-scale Optimization* (AMO) to improve the efficiency of UAV surveillance missions. Frameworks for UAV security systems have also been suggested, especially for marine and smart city applications. To find and follow small targets, robot surveillance systems use network platforms, task plans, and ML algorithms [50].

2.3.3 Search and rescue

Emergency services carry out *Search And Rescue* (SAR) operations to track people's positions during emergency situations in remote locations. Timing is a key factor for the success of the mission. Automated UAVs can increase the effectiveness of the operations. This application can be crucial during severe weather conditions, especially in rural contexts where streets and routes could get damaged. UAV units would completely change the timing and eventually the outcome of a mission where people have to be located and rescued. The application of drones, combined with an integrated data management architecture, can save lives, since external infrastructure for navigation and communication is usually unavailable during emergency situations of this kind [51]. The real-time data provided by a flying unit to an on-ground task force may also compensate the effects of street damage, traffic jams or generally hazardous terrain (typical scenarios during catastrophic events), which could compromise the feasibility of the rescue operation [52].

2.3.4 Agriculture monitoring

Drones are transforming the agricultural and farming industries and providing multiple advantages in the field. Farmers can improve their production, decision-making, and long term sustainability by integrating drones into their operations. Drones for agriculture give farmers useful information that can enhance their management decisions, maximize crop yields, and increase overall profitability. UAV-based inspections can more precisely grasp current terrain problems and provide targeted solutions based on highly accurate data by utilizing comprehensive information [53].

Passive applications like terrain mapping and scanning are usually employed for agricultural applications [17]. Moreover, active applications like crop spraying [54] and cattle herding [55] are also employed.

2.4 Introduction to ML

ML is a subset of *Artificial Intelligence* (AI) that allows systems to learn from data, without being explicitly programmed. This means that the system can improve its performance over time by analyzing data and making predictions or decisions based on that data. It is a method of teaching computers to recognize patterns and make decisions based on those patterns [56]. The core idea behind ML is to use algorithms that can learn from data, identify patterns and make decisions with minimal human intervention. This can be achieved through three main types of ML: **supervised learning**, **unsupervised learning** and **reinforcement learning** [57].

Supervised learning: starts from a set of data where the correct output is already known. The algorithm then attempts to identify the relationship between the input and the output. These types of problems are further divided into two categories: *regression*, where the output is a continuous value, and *classification*, where the output is discrete [58].

Unsupervised learning: doesn't have a predefined output. In this case, the algorithm tries to find patterns and structures in the input data without external guidance. These types of problems are mostly clustering problems, where the input data are grouped and labeled based on their relationships [59].

Reinforcement learning: involves an agent which learns a behavior through trial and error interactions with a dynamic environment. The agent learns by receiving rewards or penalties for certain actions and adapts its behavior

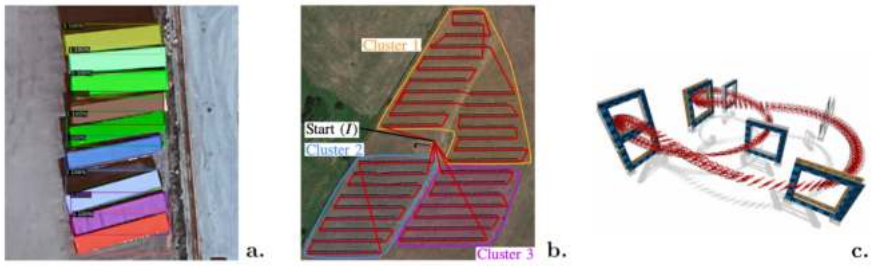


Figure 2.4: a: *supervised learning* of a CNN for the instance segmentation of a set of containers from areal images (Chap. 5). b: *unsupervised learning* for the K-means clustering of groups of solar panel rows (Chap. 5). c: *reinforcement learning* approach for autonomous UAV flight through gates [2].

accordingly. It is a method for teaching machines to make decisions in complex, uncertain and dynamic environments [60].

ML algorithms commonly fall under one of these macro categories. However, there are other, less commonly used types of ML, such as *semi-supervised learning*, which lies between supervised and unsupervised methods. This thesis work focuses on supervised and unsupervised learning applications. In Chap. 5, the specific algorithms that fit with the final method for recognizing solar panels and UAV path planning over satellite images will be described in more detail.

2.5 Artificial neural networks

Artificial Neural Networks (ANN) or NNs are a type of ML algorithms inspired by the structure and function of the human brain. They consist of interconnected nodes, or artificial neurons (modeled after the biological ones), that are able to learn and adapt to new data [56]. In this brief discussion, we focus on *feed-forward* NNs. The flow of information in the NN is then from the *input layer* (which receives the input data) through the *hidden layers* (which process and transform the input data) and finally to the *output layer* (which produces the final output or prediction), without looping back. Each artificial neuron is connected to all neurons in the next layer through weighted connections. The neuron receives as input the weighted sum of the inputs from the previous layer (Fig. 2.5), adds a bias term, and then applies an activation function before passing the information to the next layer [61, 62]. Each neuron thus performs an elementary operation as in the following equation [63]:

$$x_{m+1,k} = f\left(\sum_{i=1}^{n} w_{m,i,k} x_{m,i} + b_{m,k}\right) \tag{2.1}$$

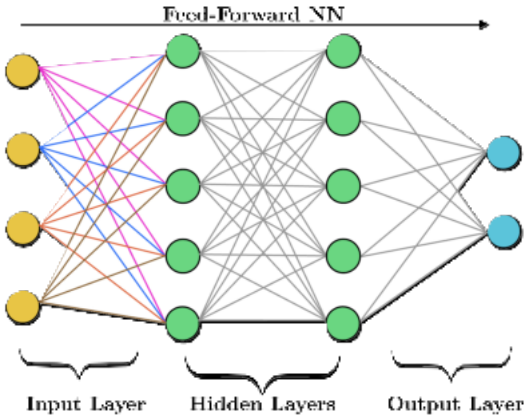


Figure 2.5: Schematic of a feed-forward NN.

In Eq. 2.1, $x_{m,i}$ is the ith neuron of layer m, $w_{m,i,k}$ is the ith weight of layer m (for the kth node of the following layer), $b_{m,k}$ is the bias of layer m (again for the kth node of the following layer), and $f(\bullet)$ is the activation function.

2.5.1 Weights

The weights assigned to the inputs are the parameters of the network that are learned during training, which is an important step of any supervised learning algorithm. This operation is needed to adjust the network weights to minimize the loss function. By doing this, the NN learns to recognize patterns in the input data and can make predictions on new (unseen) data. This is the essential principle of an ANN training process [63]. In most ML methods, the goal is to minimize a loss function [64]. For simplicity, the least-squares loss function can be considered [65]:

$$L = \frac{1}{N} \sum_{i=1}^{N} (y_P - y_T)^2 \tag{2.2}$$

where L is the loss function, N is the number of total observations used to compute the loss function, y_P is the the predicted output for a specific sample and y_T is the correspondent true value.

2.5.2 Activation function

The activation function is used to introduce nonlinearity in the network. There are several possible activation functions. The most common are the *REctified Linear Unit* (ReLU), the step and the sigmoid function (Tab. 2.1) but there are also many other ones [16].

Activation function	*Equation*
ReLU	$\max\{0,\ x\}$
Step	$\begin{cases} 1 & \text{if} \quad x \geq 0 \\ 0 & \text{if} \quad x < 0 \end{cases}$
Sigmoid	$\frac{1}{1+e^{-x}}$

Table 2.1: Some examples of activation functions [16].

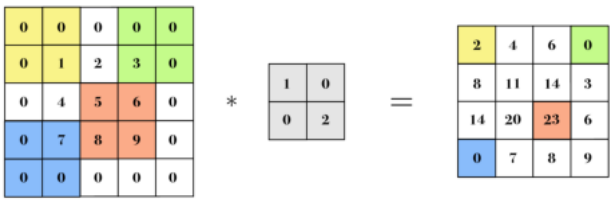


Figure 2.6: A simple example of 2D convolution, with a 2×2 kernel and padding.

2.6 Convolutional neural networks

CNNs are a type of NN that are designed specifically for image pattern recognition tasks. They are different from traditional ANNs in that they use convolutional layers that process the images with filters or kernels, creating 2D activation maps that extract specific features from the images, such as edges and vertical lines (Fig. 2.6). One of the main advantages of CNNs is that they can effectively compress the amount of data to be processed [66]. For example, in a dataset of 256×256 color images, a single neuron in the first hidden layer would require $(256 \times 256 \times 3) = 196.608$ weights, which would lead to a very large network which is complex to train. Additionally, having many parameters increases the risk of overfitting [67]. CNNs also employ a *pooling layer* which further compresses the amount of data, making the network more efficient. There are two types of pooling (Fig. 2.7):

- *average pooling*: returns pixel values within a specific region from the portion of image covered by the filter (Fig. 2.6).
- *max pooling*: returns the maximum value from the portion. It is usually preferred over average pooling (Fig. 2.7).

Finally, the last output is then flattened and sent to a standard NN.

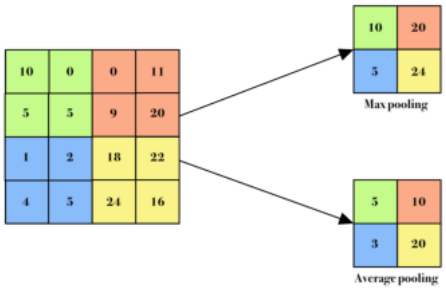


Figure 2.7: A simple example of pooling with a 2×2 filter.

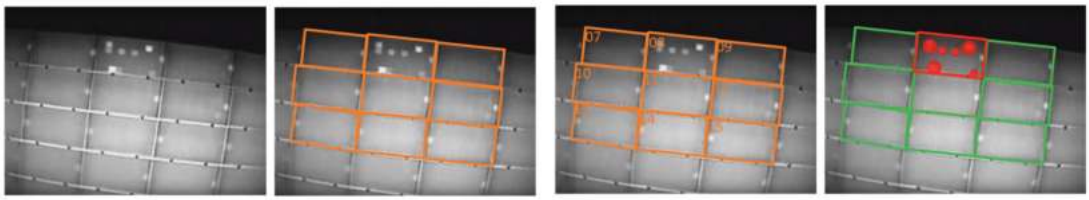


Figure 2.8: Semantic segmentation for the identification of defects in PV panels [3].

2.7 Segmentation

Semantic segmentation and *instance segmentation* are two techniques used in deep learning for image and video analysis. The goal of these algorithms is to assign a semantic label to every pixel in an image. This allows for a much more detailed and nuanced understanding of the scene, as opposed to just classifying the entire image as a whole. It differs from classic *object detection*, since the latter only draws a bounding box on the detected object. For the localization of the PV panel rows, segmentation approaches are much more accurate that simple bounding box detection, since with the former the actual outline of the panels can be extrapolated; also, this approach can be used for the detection of defects on the panels [3].

After the segmentation, it is necessary to localize a specific row in the given image. The minimum rectangle containing the panel is calculated from the mask that highlights the object in the image. Then, the center position and orientation angle of the rectangle are calculated and the coordinates of the center are returned, to locate the panel in the image [68].

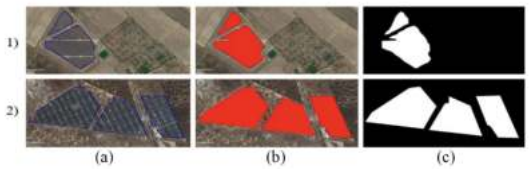


Figure 2.9: Example images from the Amir database [4].

2.8 Applications of NNs for PV plants

The main methods for extracting the boundaries of a PV plant identified in research papers on the topic are as follows.

Fully CNN: a *Fully Convolutional Neural network* (FCN) based on a Mask-RCNN deep network architecture is used for the boundary extraction [4].

Network-based architecture: a semi-automatic approach that uses image processing (starting from a CNN built on a server) for efficiently and precisely identifying the *Region of Interest* (RoI). An image processing layer is also added to the whole process [25].

2.8.1 FCN approach

The identification of the boundaries of a PV plant is necessary for creating a precise flight route that covers the entire area. FCNs are an efficient way to extract such boundaries. The FCN method utilizes a modified version of the traditional CNN architecture by replacing *fully-connected layers* with convolutional layers. The input for this method is a *Red-Green-Blue* (RGB) image of a PV plant, while the output is the predicted mask. In the first work presented, a *Mask-RCNN* was employed with a *VGG16* modified backbone. Mask-RCNNs are a method for instance segmentation tasks, utilizing a two-phase approach. The initial phase scans the image to generate potential object locations, while the second stage categorizes these proposals and generates pixel-level masks [69]. The VGG16 backbone includes connections in various directions, including upward, downward and lateral connections, resulting in the creation of diverse feature maps such as pyramid maps.

Since FCN is a supervised ML algorithm, a large and comprehensive dataset is required for training the algorithm [70]. The *Amir* dataset was thus presented, containing 3580 *orthophotos* from various large-scale PV plants located in 12 different countries. The dataset is diverse, featuring PV plants with varying capacities [4].

2.8.1.1 Model setup

A typical approach for boundary extraction with the *Mask-RCNN* method involves *down-sampling* the input image to obtain lower-resolution feature maps, which are then *up-sampled* using transposing convolutions to produce a full-resolution segmentation map. The weights of the model are initialized to values from *ImageNet*, a technique known as *transfer learning*, to improve accuracy and reduce losses during training and evaluation [4].

For the model optimization, the `binary_crossentropy` Python function evaluates the training loss. This function analyzes pixel by pixel the segmentation data and it compares it to the values provided by the ground truth dataset's data. The problem then reduces to a *Root Mean Square* (RMS) optimization problem [4].

2.8.1.2 Results

A comparison of TIP techniques against a new FCN method for boundary extraction of PV plants was conducted to implement an autonomous monitoring, aerial photogrammetry, and path planning system for aerial robots. The FCN technique was found to be more efficient and had higher accuracy than the TIP method; also, FCN did not require site-specific tuning parameters [4].

2.8.2 Semantic-Segmentation-Based approach

The second method concentrates on developing an efficient strategy for coverage path planning over PV plants using UAVs with the help of a semantic segmentation algorithm (located on a server) to identify the region of interest [25]. The algorithm is based on the following steps:

1. the raw satellite images (from *Google Maps*) of the PV plants are taken at a pre defined altitude from the ground.
2. the image is entered in the DL server, which will obtain a mask of the PV plant; this step is based on earlier work by the same group [5].
3. finally, a series of OpenCV functions are applied for post-processing.

The output masks were tested with a CPP method as shown in Fig. 2.10.

2.8.2.1 DL server for segmentation

A DL server is used to segment the plant (Sec. 2.7) and extract a mask from the Google Maps image. Before the segmentation, training samples are collected

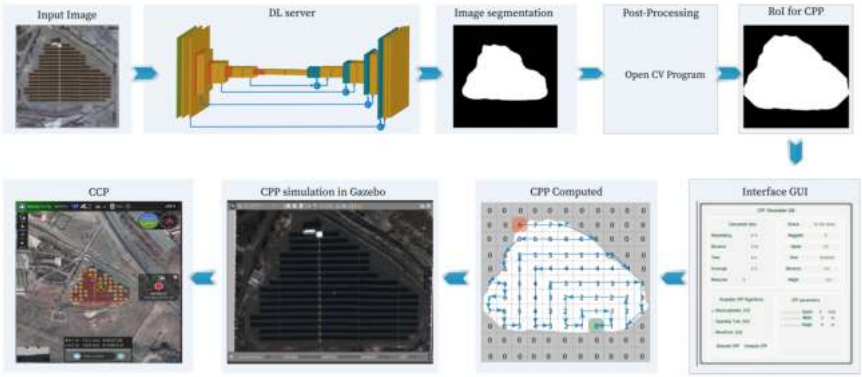


Figure 2.10: Schematic of the data processing method in [5].

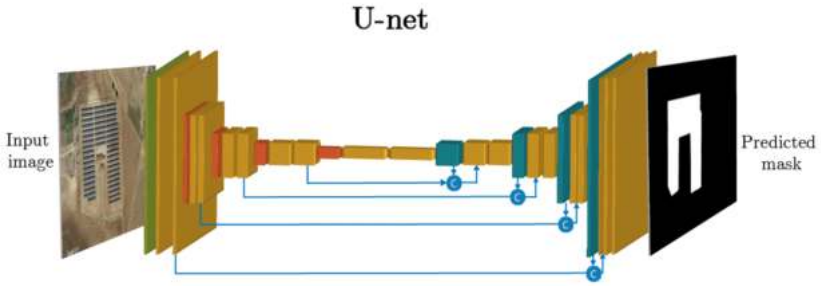


Figure 2.11: U-Net network structure [5].

from the Amir data-set. The samples are then converted into tagged image file format (.JPG) and mask image file format (.PNG) with a resolution of 240×320 and are entered into the CNN. The model chosen for the CNN server is the *U-net* network. This model uses two blocks (Fig. 2.11), respectively having decreasing and increasing layer sizes, thus forming a U-shaped architecture. The decreasing block is a typical CNN that reduces spatial information while increasing feature information. The increasing block combines feature and spatial information through up-sampling layers and transposed convolution. The platform used for this study is *Tensorflow* with *Keras* as back-end. For the model optimization, the `binary_crossentropy` function is used for the evaluation of the training loss.

The post processing involves a step-by-step image processing method using OpenCV functions. The method starts by applying morphological operators (`Erode` and `Dilate`) to the images, then uses the *FindCountours* function to extract their contours. The `ContourArea` function is used to find the area of the contour, then smaller areas (which may be false positives) are filtered out. The `ApproxPolyDP` function is used to approximate the contour to a shape with fewer vertices. Finally, the `Erode` operator is applied again to expand this shape, to

compensate for inaccuracies due to the CPP method and false positives of the DL server [25].

2.8.2.2 Results

The U-net network model was found to be effective in predicting the area of PV plants compared to other techniques like the FCN method used in Sec. 2.8.1. This method allows for faster processing and better performance in identifying PV plant areas, filtering out false positives. Moreover, the post-processing refinement step further improves the performance.

2.8.3 U-net for satellite image segmentation

As shown by the bibliographic analysis above, for satellite image segmentation tasks U-Net models are typically used. They perform well in a variety of image segmentation applications and are specifically created for semantic segmentation tasks. Also, complex and varied patterns (like various types of land, structures, roads, and water bodies) are frequently visible in satellite images. The U-Net model is well suited to handle such complex scenes due to its capacity to capture contextual information and small shapes [5].

2.9 Path identification

A goal of this thesis work is the determination of the most efficient route for UAV coverage based on a satellite image. Since UAVs have endurance limited by their batteries, an inspection method needs to deal with such limitation. For this, a path planning system that generates the best possible route in terms of length and time was implemented as an alternative to the standard (but slower) spiral flight approach [6].

As mentioned in Sec. 2.10, some works on path identification are based on connecting the WPs. For instance, the authors of [6] define a way to connect WPs on the border of a PV plant to cover the whole facility (Fig. 2.12). Since the drone flies at fairly high altitude in this application, the WPs are not correlated with the lines of panels, their number is low and a simple *points connection* algorithm can be applied. In this thesis, on the other hand, we set (at least) two WPs per line, so defining an optimal connection among them may be nontrivial, especially when the plant structure is irregular. Therefore, automatically finding the best possible path becomes a case of the *Traveling Salesman Problem* (TSP).

2.9.1 TSP

2.9.1.1 Historical background

The TSP can be described as follows: *"a salesman has to visit each of n different cities once and only once, starting from a base city and returning to it. Which path minimizes the total distance traveled by the salesman?"* [71]. This problem, a classical one in Operations Research, has been approached in many ways throughout the years. Starting from the simple question of establishing the best route for a single agent and generalizing to multi-agent systems (for instance, a combination of aerial and ground vehicles), the TSP is still an active area of research, with many applications in goods delivery.

2.9.1.2 Definition and types of solutions

The TSP is an *NP-hard* problem in *combinatorial optimization*. NP-hard defines a class of problems that are at least as computationally difficult as the hardest problems in NP, the complexity class of those decision problems which are solvable in polynomial time by a *nondeterministic Turing machine*. Despite its computational difficulty, various heuristics and exact algorithms have been developed for the TSP. These approaches allow for the exact solution of instances with tens of thousands of nodes and approximate solutions for millions of nodes.

There exist two types of TSPs: *symmetric* (where the length of the edges connecting two nodes is the same regardless of the traveling direction) and *asymmetric* (where the length of an edge depends on the direction).

The most studied approaches to solve the TSP are *exact algorithms*, which solve problems optimally. However, in the worst case, an exact algorithm for an NP-hard problem cannot run in *polynomial* time. Extensive research has been conducted to develop exact algorithms with *exponential* running times but low bases. One such method is the *branch and cut*, which involves solving *integer linear programming problems* through a combination of *branch and bound algorithms* and *cutting planes* that reduces the linear programming relaxations.

On the other hand, *heuristic techniques* are practical methods that do not guarantee optimal solutions but provide satisfactory approximations in a fraction of the time. Heuristics serve as shortcuts to reduce the computational load involved in reaching a result. Important examples of heuristic methods for solving the TSP are listed below [72].

Greedy method: it picks the locally optimal path, building a solution incrementally by consistently picking the closest unexplored node. Although it is one of the fastest algorithms, it frequently gets trapped in local minima and produces sub-optimal solutions. It is typically applied to provide a starting solution for more complex algorithms.

Local search (LS) and **Iterated local search** (ILS): LS methods search the neighborhood of the current solution to find an improving change. Common neighborhood search moves are: *two-opt* (removing two edges and reconnecting), *three-opt* (removing three edges) and *Lin-Kernighan* (adaptive choice of the number of edges to remove). These methods are simple but can slowly improve the solution. Hybrid methods that switch neighborhoods (to regularly change the initial solution) or combine multiple local searches are often more effective. One example of these algorithms is ILS, which iterates the LS algorithm to get out of local minima.

Simulated Annealing (SA): a probabilistic technique, analyzes random changes to the solution. With a decreasing likelihood of changes over time, it will occasionally accept worse solutions to get away from local minima. SA can use moves such as two-opt, three-opt, or swapping a random pair of nodes. While slower, SA has been successful for many applications.

Memetic Algorithms (MA): combine LS with strategies like *Genetic Algorithms* (GA). For TSPs, this can entail taking two solutions and combining them (the *crossover operator*), then enhancing the result with a LS. MA seeks to effectively integrate the local exploration of LS with the global exploration of evolutionary approaches. MA can offer high-quality solutions but may take many iterations to converge. *ϵ-approximate algorithms* ensure that the objective function for the solution is at most ϵ times the minimum. The goal is to find algorithms where the approximation ratio ϵ is as close as possible to 1 while maintaining a manageable computational complexity. A relevant example is the 3/2-*approximate Christofides*' algorithm.

In brief, there are many algorithms that can effectively solve the TSP, with different trade-offs between solution quality, speed, implementation difficulty and theoretical guarantees. Hybrid methods and MAs are currently the most successful approaches [73].

2.10 CPP for PV plants

After obtaining the mask for the PV plant, the second objective of the coverage path planning algorithm is to define the quickest route for scanning the infrastructure. CPP algorithms are used to optimize the flight path of an UAV for inspection tasks. Those algorithms plan the trajectory to ensure that the entire solar plant is covered during the inspection while minimizing flight time. CPP algorithms take into account factors such as the size and shape of the solar plant, the altitude and

speed of the UAV, and the type of sensor or camera being used for the inspection. The goal of CPP algorithms is to provide a systematic and efficient method to ensure that any defects or problems are detected and addressed in a timely manner [25]. The most common CPP algorithms are based on WP mapping. A few examples are:

Static Point Algorithm (SPP) [6]: an optimal initial path is generated through image processing techniques.

Dynamic Point Algorithm (DPP) [6]: checks if the UAV can complete the initial path, otherwise it defines a new path based on UAV sensor data.

Boustrophedon Exact Cell Decomposition (BECD) [74]: involves covering each cell in a *boustrophedon* pattern, moving back and forth until the entire area is covered.

Grid-Based Spanning Tree Coverage (GBSTC) [8]: subdivides the work-area into cells, then follows a *spanning tree* of the graph induced by the cells, while covering every point exactly once.

Grid-Based Wavefront Coverage (GBWC) [9]: finds a solution using an extension of the distance transform path planning method.

In the following, such techniques will be discussed evaluating the advantages and disadvantages of each. We assume that the CPP is done over the same images used to find the mask of the plant.

2.10.1 SPP and DPP

The SPP identifies the intersections where the border of the plant intersects a number H_l of parallel and uniformly-spaced horizontal lines; it holds

$$H_l = \frac{w}{h \times f - c} \tag{2.3}$$

where w is the solar plant width, h the UAV flight height, f is the portion of landscape captured by the camera with respect to the flight height, and c is a correction parameter introduced to increase precision. The distance N_r between the lines is computed as

$$N_r = \frac{y_M - y_m}{H_l} \tag{2.4}$$

where y_M and y_m are respectively the maximum and minimum y-axis coordinates of the solar plant in the image.

Figure 2.12: Different steps of the SPP algorithm [6].

These horizontal lines are then intercepted with the solar plant borders to find the actual WPs. The process is repeated for different rotation angles of the images: the path length is compared for every angle and the one corresponding to the shortest path is selected. The WPs thus found are then transformed from pixel coordinates in the image plane to actual coordinates [6].

The DPP begins on an initial path planned by SPP. The difference is in the online flight operation. For the DPP, if a defect is detected, the motion stops to let the UAV perform a closeup. Therefore, after every error-specific maneuver, the algorithm will reprogram the flight by a method, similar to the SPP, that takes into account the current battery level of the drone, thus finding an admissible path.

2.10.2 Cell-based CPP algorithms

The three algorithms presented in this subsection are based on a cell decomposition of the map.

2.10.2.1 GBWC

This is the earliest grid-based algorithm for CPP problems. This offline algorithm aims at finding a connection between an initial cell i in the grid and a final one f. An heuristic approach attributes a number to each cell of the grid through a *wavefront* process from cell f to cell i. This process assigns the value 0 to cell f and from there computes the grid distance of each cell from f (*distance transformation*) by successive clusters [75]. This method is applied until the assignment cluster reaches cell i. Once the distance transformation is calculated, the agent's path is determined starting from cell i and selecting at each step the adjacent cell yet to be explored with the highest number [8, 9].

2.10.2.2 BECD

The BECD method takes the free space and the obstacles and divides them into cells. These cells are then covered by the robot in a back-and-forth pattern, switching direction with 90-degree maneuvers. This approach improves upon the

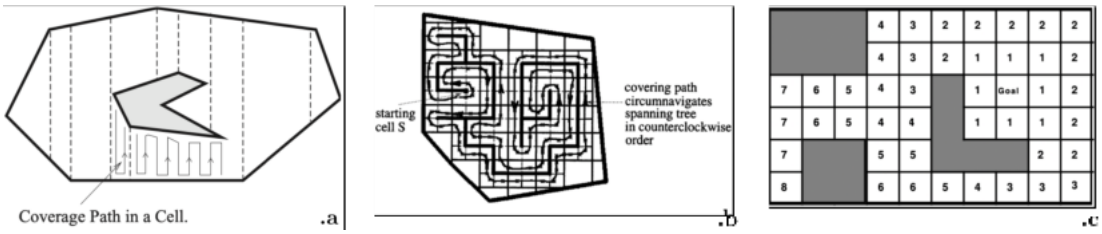


Figure 2.13: *.a* The steps for the BECD algorithm [7]. *.b*: the GBSTC algorithm [8]. *.c*: the GBWC algorithm [9].

trapezoid decomposition technique by leveraging the structure of the polygon to identify the beginning and end of an obstacle, resulting in fewer cells that don't require redundant steps, and allows for the coverage of areas with curved borders [7, 74] (Fig. 2.13.a).

2.10.2.3 GBSTC

The area to be explored is decomposed into cells of side l and the agent can move in perpendicular directions following the cell sides. Every cell is then divided into smaller ones of $l/2$ side length. The algorithm eliminates from the map all the occupied cells and those containing obstacles (Fig. 2.13.b).

The solution can then be found in different ways [8]:

off-line: the agent has complete knowledge of the surrounding environment.

on-line: the agent exploits sensor data to scan the environment and build a spanning tree.

ant-like: no knowledge of the environment is given to the agent, which however can leave markers during the coverage process to identify the already-visited areas of the grid in a heuristic fashion.

2.11 Drone flight and control

We now examine the control mechanisms for guiding drone flights. The papers discussed here analyze CV and DL models specifically designed for panel recognition, to ensure that the drone can accurately identify and track the panels using different visual servoing techniques. The aim is to extract essential visual features that will guide the drone in maintaining a steady trajectory and executing the pre-planned flight with the highest level of precision and efficiency. This point is especially critical when dealing with larger PV installations, where the

complexity of the system and the potential errors can be significant. The analysis of these advanced CV models and visual servoing techniques aims at minimizing the potential for errors and ensure that drones can perform their inspections efficiently and accurately [76].

2.11.1 Flight forces

For simplicity, let us consider cruise flight in a fixed-wing aircraft. When a system is flying at a steady altitude and constant (non-zero) velocity, it is subject to four forces [77].

Weight: due to aircraft and payload. This force is directed downwards.

Drag: opposite the direction of motion, generated from all surfaces affected by the flow of air.

Lift: determined mainly by the wings, but also by all other lifting surfaces. It is orthogonal to the direction of motion.

Thrust: generated by the propulsion system, which can be of any type (motor propeller, turbojet, turbofan and so on).

In a steady flight, these forces cancel each other out.

2.11.2 Quadrotor modeling

The drone used in this thesis work is the *DJI Mavic 2 Enterprise Duo* (App. A.1), a quadrotor model (Sec. 2.1.1).

In a multicopter configuration, each propeller generates thrust (due to lift), orthogonal to the plane of rotation of the blades. During hovering (namely, when the multicopter is at a fixed pose), the total thrust of all propellers balances the weight. On the other hand, in stable flight conditions (namely, when the multicopter is cruising at a constant altitude and speed), one component of the thrust balances the weight, and the other component causes the motion in the desired direction, balancing out the resistance that the multicopter experiences as it moves forward. Each propeller also generates torque (due to drag) [10]; thus, multicopters almost always have an even number of propellers, half of which rotate clockwise while the others rotate counterclockwise. This way, when all propellers have the same speed, the overall torque is zero and the drone does not rotate along the z (*yaw*) axis; to change the yaw angle, different propeller speeds are set, while the overall thrust keeps compensating for the weight (Fig. 2.14).

A quadrotor has six *Degrees of Freedom* (DoFs). There are 3 translational DoFs (*forward/backward*, *left/right*, *up/down*) and 3 rotational DoFs (*roll*, *pitch*,

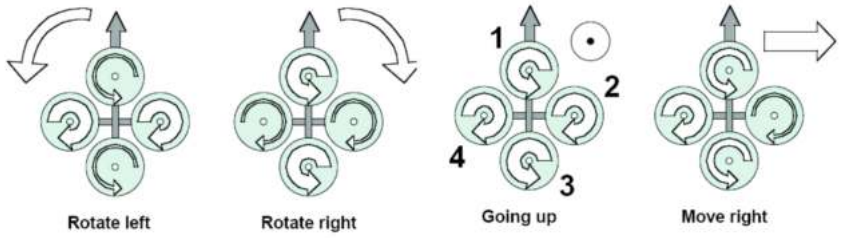


Figure 2.14: The quadrotor concept. The width of the arrows is proportional to the angular speed of the propellers [10].

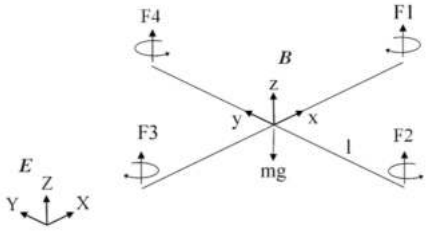


Figure 2.15: Quadrotor reference frame [10]

yaw). A quadrotor drone is however an *underactuated* robot, because it has more DoFs than the number of actuators or control inputs (namely, the four rotor speeds), meaning that it cannot control all of its DoFs independently. However, it is still able to perform a wide range of maneuvers by controlling the amount of force generated by each rotor.

In the following, the geometric, kinematic and dynamic models for quadrotor UAVs will be briefly explained. Some assumptions are made here: all the components (the drone frame and the propellers) are supposed to be rigid; the frame is symmetric, and the *Center of Mass* (CoM) of the drone is at the center of the frame; the thrust and drag of a propeller are proportional to the square of its velocity [36].

2.11.3 Frames and states of the quadrotor

The convention used for defining the frames (Fig. 2.15) is the same as the one in [10]. We introduce an earth-fixed frame E, having axes (X, Y, Z), and a body-fixed frame B, with axes (x, y, z) and x pointing toward rotor 1. The position vector $\mathbf{c} = [c_x, c_y, c_z]^T$ of the drone CoM identifies the drone position in space. The vector $\Gamma = [\phi, \theta, \psi]^T$ is the vector of Euler angles from the mobile to the fixed frame. The frame orientation is defined by a rotation matrix ${}^E\mathbf{R}_B \in \mathrm{SO}(3)$ from

B to E.

2.11.4 Geometric and kinematic models

We can write the 3×3 rotation matrix as

$$ {}^{E}\mathbf{R}_{B} = \begin{bmatrix} c(\psi)c(\theta) & c(\psi)s(\theta)s(\phi) - s(\psi)c(\phi) & c(\psi)s(\theta)c(\phi) + s(\psi)s(\phi) \\ s(\psi)c(\theta) & s(\psi)s(\theta)s(\phi) - s(\psi)c(\phi) & s(\psi)s(\theta)c(\phi) + c(\psi)s(\phi) \\ -s(\theta) & c(\theta)s(\phi) & c(\theta)c(\phi) \end{bmatrix} \tag{2.5} $$

where $c(\bullet) = \cos(\bullet)$ and $s(\bullet) = \sin(\bullet)$. The square magnitude of the velocity for any point is $v^2 = \dot{p}_x^2 + \dot{p}_y^2 + \dot{p}_z^2$, where $\dot{\mathbf{p}} = [\dot{p}_x, \dot{p}_y, \dot{p}_z]^T$ is the velocity vector of any point P on the drone in the global frame [10].

2.11.5 Dynamic model

In this work, the quadrotor dynamics are described through the *Euler-Lagrange* formalism [36]. We first introduce the Lagrangian function

$$ L(\mathbf{q}, \dot{\mathbf{q}}) = K(\mathbf{q}, \dot{\mathbf{q}}) - P(\mathbf{q}) \tag{2.6} $$

where $K(\mathbf{q}, \dot{\mathbf{q}})$ and $P(\mathbf{q})$ are respectively the kinetic and the potential energy of the system and $\mathbf{q}$ is the control input. The equations of motion are then derived from the Euler-Lagrange equations:

$$ \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) - \frac{\partial L}{\partial q} = \begin{bmatrix} \mathbf{F} \\ \boldsymbol{\tau} \end{bmatrix} \tag{2.7} $$

where $\mathbf{F}$ and $\boldsymbol{\tau}$ are the force and torque vectors, respectively. The dynamic model of the quadrotor for the translational DoFs, expressed in the local frame, can be written as:

$$ m\ddot{\mathbf{c}} + mg\mathbf{k} = \mathbf{F} \tag{2.8} $$

where m is the drone mass, g the gravitational acceleration, and $\mathbf{k} = [0, 0, 1]^T$. In matrix form, in the global frame we have [36]:

$$ m\ddot{\mathbf{c}} = \begin{bmatrix} 0 & 0 & -mg \end{bmatrix}^T + {}^{E}\mathbf{R}_{B} \begin{bmatrix} 0 & 0 & u \end{bmatrix}^T \tag{2.9} $$

where u is the total thrust vector magnitude.

The torque components acting on the drone due to the drag torque on each propeller is [10]:

$$ \begin{cases} \tau_x = bl(\Omega_4^2 - \Omega_2^2) \\ \tau_y = bl(\Omega_3^2 - \Omega_1^2) \\ \tau_z = d(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \end{cases} \tag{2.10} $$

where Ω_i is the speed of motor i, d is a constant coefficient defining the reaction torque due to rotor drag, b is a constant coefficient defining the thrust and l is the distance between the CoM and the rotation axis of the propellers. Another torque component is due to the rotation of the propellers, creating a gyroscopic effect:

$$\begin{cases} \tau'_x = J_r \omega_y (\Omega_1 - \Omega_2 + \Omega_3 - \Omega_4) \\ \tau'_y = -J_r \omega_x (\Omega_1 - \Omega_2 + \Omega_3 - \Omega_4) \end{cases} \tag{2.11}$$

where J_r is the motor inertia; the rotational velocity of the drone is defined by $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]^T$.

2.11.6 Control model

We now briefly explain the control laws for controlling a quadrotor flight. To overcome the DoF deficiency of quadrotor drones, a *cascade control* system architecture will be used. In cascade control, two or more controllers operate in series, with the output of one controller serving as the input to the next one. This type of control architecture is often used in quadrotor control because it allows for the implementation of separate controllers for different aspects of the motion, such as position and orientation. This can make the control system more flexible and efficient, allowing the quadrotor to respond more accurately to a wide range of control inputs and disturbances. Additionally, cascade control can make it easier to account for nonlinearities in the system and to compensate for uncertainties. The following three control layers are then needed:

Mixer-matrix: computes the rotors velocities Ω_1, Ω_2, Ω_3 and Ω_4 from the torques around the three axes and the thrust force of the quadrotor.

Attitude controller: sets the desired drone orientation using torque.

Position controller: enables the UAV to maintain a proper position by computing the desired angle configuration to input into the *Attitude controller*.

An example of this control scheme is in Fig. 2.16, where $[x_d, y_d, z_d]^T$ is the desired position, $[\psi_d, \phi_d, \theta_d]^T$ are the desired Euler angles, F is the thrust force, $[\tau_\psi, \tau_\phi, \tau_\theta]^T$ is the torque vector, and $[\omega_1, \omega_2, \omega_3, \omega_4]$ are the angular velocities of the rotors.

2.12 Autonomous flight for PV plant inspection

Another problem to tackle is creating a drone control system to inspect the PV installation area while following the given path. In the bibliographic analysis, we

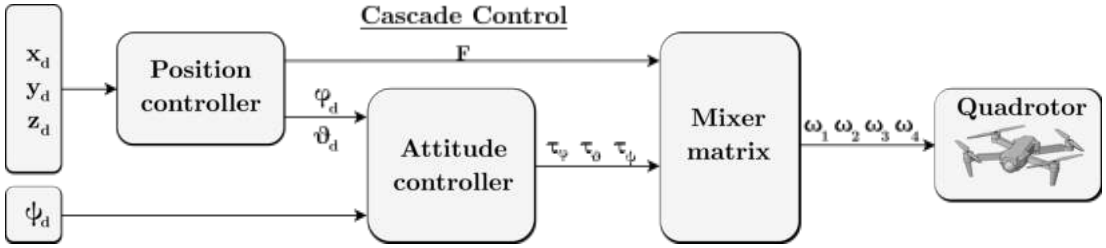


Figure 2.16: Schematic of the cascade control law for quadrotors.

gathered works for solving different related problems, which are discussed in the following sections.

- **Flight control**

  Vision based inspection strategy: in this section, a method is presented for using real-time video frames to acquire the linear features detected as the edges of single PV panel strings and then to calculate the flight direction and offset, resulting in an automatic tracking of PV arrays [11].

  Combination of tasks: since the method in the previous work is not based on a pre-defined trajectory, the combination between trajectory planning and flight control part is investigated.

  Flight altitude: discusses the correlation between the altitude of aerial photography and the ability to identify defects [13].

- **Image reconstruction**

  Post processing image mosaicking: develops an image post-processing tool for building digital maps for defect detection combining an *Image Mosaicking* (IM) algorithm with a CV module recognition algorithm [78].

2.13 Vision-based inspection strategy

A typical UAV-based inspection system aims to obtain infrared and visible images of solar panels to detect failures. Usually, the process of acquiring the images is performed either manually or through autonomous flight control using pre-determined WPs. The method presented here is a vision-based image acquisition strategy that does not rely on predefined WPs [11]. The flight control system will be used to help the drone follow the line of panels to accommodate variations in the way the modules are organized in rows.

2.13.1 Lines and slope detection

The algorithm is based on the detection of the panel edges, which will be the guidelines for the flight control. To extract this feature, the video from the downward-looking camera is processed. The data extraction is conducted in the *Hue-Saturation-Value* (HSV) color space, since the edges are more clearly visible [79]. Some pre-processing steps are performed on the image, namely:

1. *Resizing* the UAV camera image to a smaller size;
2. *Median filtering* the image;
3. *Screening* of the pixels belonging to the edges in *HSV* color space;
4. *Connecting* adjacent panels though morphological methods;
5. *Applying* the Canny edge detector to identify all the edges [11, 80].

2.13.1.1 Lines

Once the final binary image is extracted, a *Hough transform* [81] is used to identifying the straight lines in the image. The main concern with the application of this algorithm is the fact that some lines will likely be detected that do not correspond to the actual edges of the panels. A possible rejection strategy exploits the fact that the panel edges correspond to the longest lines. Another important feature is the line *slope* due to the tilted position of panels. Therefore, the lines to be detected can be extracted using thresholds on the line length l_l and slope k_l in the image frame:

$$\begin{cases} l_l > l_{th} \\ |k_l| < k_{th} \end{cases} \tag{2.12}$$

where l_{th} and k_{th} are the thresholds on the panel length and slope [11].

2.13.1.2 Slopes

The average slope k_a and the distance e_{pix} between the center of the image and of the line are computed as follows (Fig. 2.17):

$$\begin{aligned} k_a &= \left(\sum_{i=1}^{p} k_{upp} + \sum_{i=1}^{q} k_{low} \right) /(p+q) \\ e_{pix} &= \left(\sum_{i=1}^{p} d_{upp} \right) /(2p) + \left(\sum_{i=1}^{q} d_{low} \right) /(2q) - h_{img} \end{aligned} \tag{2.13}$$

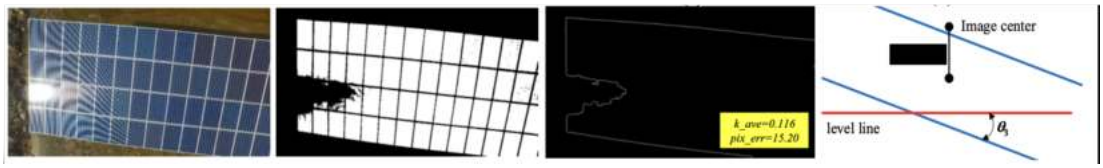


Figure 2.17: Image pre-processing according to the steps in [11].

where p is the number of upper edge lines, q is the number of the lower edge ones, h_{img} is the height of resized video images, k_{upp} and k_{low} are the upper and lower edge lines, and d_{upp} and d_{low} are the pixel distances between the k_{upp} and k_{low} centers with respect to the center of the image [11].

2.13.2 UAV velocity controller

This control action is be used over lines of PV panels and concerns the yaw angle of the camera's gimbal and the velocity of the UAV unit [11].

2.13.2.1 Tracking procedure

The coordinate frame considered fllows the *North East Down* (NED) convention. The direction of the PV panel strip is determined as:

$$\theta = \theta_1 + \theta_2 + \theta_3 \tag{2.14}$$

where θ_1 is the yaw angle of the UAV, θ_2 is the yaw angle of the camera gimbal, and $\theta_3 = \arctan(k_a)$ is the inclined angle of PV string in frame. In a standard configuration of the UAV, $\theta = 0$. The distance between the center of the image and of the strip is

$$d_{err} = \frac{e_{pix}}{h_{img} h \tan(0.5 f)} \tag{2.15}$$

The control objective is to minimize both the deviation of the *gimbal yaw* and d_{err} while keeping the UAV speed constant. The *gimbal yaw* is adjusted by an angle

$$\Delta\theta_2 = \theta_3 \tag{2.16}$$

The control action can be written as

$$\begin{cases} v_x = -v_0 \sin\theta + k d_{err} \\ v_y = v_0 \cos\theta \end{cases} \tag{2.17}$$

where v_0 is a velocity value and the parameter k is

$$k = \frac{b}{a + |d_{err}|} \tag{2.18}$$

where a and b are positive constants. The parameter k is added to limit the value of d_{err} [11].

2.13.2.2 Identification of the end of a strip

The identification of the panel strip is based on the pixel value of the binary image after the HSV conversion. The UAV has crossed the strip boundary if the mean value of the pixel values $x_i \in \{0, 1\}$ is greater than a threshold; the mean value depends on the amount of white pixels in the binary image, which in turn depends on the drone position [11]. With a similar method, one can also check if the UAV has reached the end of a panel line.

2.14 Combination of tasks

In a *visual servoing* approach, the simplest way to combine all tasks at once is through *task stacking* [12]. We define a minimization problem as follows:

$$\underset{\mathbf{u}}{\operatorname{argmin}} \|\mathbf{J}\mathbf{u} - \dot{\mathbf{e}}^*\|^2 = \sum_i \|\mathbf{J}_i\mathbf{u} - \dot{\mathbf{e}}_i^*\|^2 \tag{2.19}$$

where $\mathbf{u}$ is the input vector, $\dot{\mathbf{e}}^* = [\dot{\mathbf{e}}_1^* \quad \dot{\mathbf{e}}_2^* \dots]^T$ is the matrix of the time derivatives of the desired errors for all the tasks that we want to perform (defined as $e_i^* = u_i - u_i^*$), and $\mathbf{J} = [\mathbf{J}_1 \quad \mathbf{J}_2 \dots]^T$ is the *extended Jacobian matrix* which combines the interaction matrix $\mathbf{L}$ of the camera and the Jacobian $\mathbf{J}$ of the UAV.

This approach is a compromise between all the tasks that the UAV is required to do. The UAV thus tries to fulfill every task as well as possible, but the constraints cannot be all respected exactly, since all DoFs are constrained [12] due to the dimension of the control input for the trajectory tracking; here, we set the same *task priority* among the considered tasks.

2.14.1 Extended Jacobian

This method exploits the concept of prioritization of tasks with a variable hierarchy-based approach. Visual features are created based on the images captured by the drone that constrain the UAV movement when necessary. An example of this is maintaining a certain 2D image feature inside the camera field of view; here, we want to keep the line previously detected within defined bounds of the image. Considering one such feature s, an inequality constraint is set as $s^- \leq s \leq s^+$, with s^- and s^+ being the lower and upper bounds for feature s. We then define

$$s^* = \frac{s^+ + s^-}{2} \tag{2.20}$$

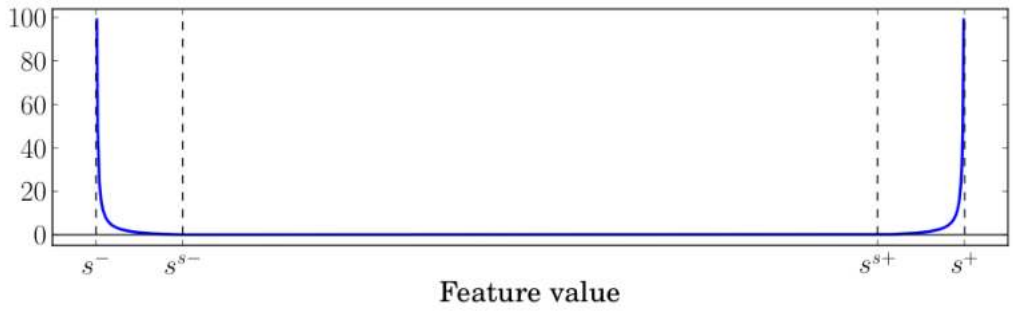


Figure 2.18: Visual representation of a function h_i, from [12]

as the desired value of the feature, midway between the bounds. The derivative $\dot{e}^*$ of the desired error between the current position of the feature and the desired one is then $\dot{e}^* = \dot{s}^*$. This new task will simply be stacked on top of the others.

The objective is to create a control action capable of keeping the feature within the desired bounds but acting only if the feature approaches them. To do so, matrix $\mathbf{H}$ is introduced within the stacking approach [12]. The derivative of the current feature vector can be written as $\dot{\mathbf{s}} = \dot{\mathbf{e}} = \mathbf{J}\mathbf{u}$. Both sides are then pre-multiplied by matrix $\mathbf{H}$, obtaining $\mathbf{H}\dot{\mathbf{e}} = \mathbf{H}\mathbf{J}\mathbf{u}$. The desired error, defined as $\dot{\mathbf{e}}^* = -\lambda\mathbf{e}$ is also pre-multiplied by $\mathbf{H}$, which is defined as:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_m & \mathbf{0}_n \\ \mathbf{0}_m & \mathbf{h}(\mathbf{s}) \end{bmatrix} \tag{2.21}$$

where m is the number of DoFs of the robotic system carrying the camera, n the number of extra constraints to keep it on track, and $\mathbf{I}_m$ is the $m \times m$ identity matrix. Matrix $\mathbf{h}(\mathbf{s}) = \text{diag}\,(h_1(\mathbf{s}), h_2(\mathbf{s}), \dots)$ contains n functions $h_i(\mathbf{s})$ whose values increase (Fig. 2.18) as the feature gets closer to the bounds of the area that it can occupy within the field of view.

The dimensions of matrix $\mathbf{J}$ need to be adapted to the dimensions of $\mathbf{H}$, thus $\mathbf{J}$ is stacked on top of an identity matrix $\mathbf{I}_n$. This structure lets matrix $\mathbf{H}$ regulate the *importance* of the correction tasks, since the values of the $h_i(\mathbf{s})$ will be much higher than those in the Jacobian, which guides the UAV flight along the predefined trajectory [12]. The optimization problem is then redefined as follows:

$$\underset{\mathbf{u}}{\text{argmin}} \|\mathbf{H}\mathbf{J}\mathbf{u} + \lambda\mathbf{H}\mathbf{e}\|^2 \quad \Rightarrow \quad \mathbf{u} = -(\mathbf{H}\mathbf{J})^+\lambda\mathbf{H}\mathbf{e} \tag{2.22}$$

This type of flight correction can be optimal to ensure a good scanning of long strips of PV panels, since the extra control action will interfere with the flight exclusively when needed.

2.15 Flight altitude

One of the main focuses of this work is to apply UAV inspection techniques at a lower flight altitude with respect to the state of the art. Here, we discuss the correlation between flight height and defect detection.

Aerial visual photography has been intensively exploited for defects detection in PV panel monitoring. Multiple types of defects and failures can be detected by visual sensors. These are generally not standardized: UAV units can carry different sensor sets and cameras depending on the inspection type. Another key difference is in the height at which a drone can fly. It is important, then, to define the advantages and disadvantages of flying at different heights [13].

2.15.1 Scale determination

Before inspection, one needs to be aware of the scale of aerial photography, to identify defects at an appropriate altitude, count the number of panels, and plan the mapping of the plant. The scale is used to correlate measurements from aerial images to actual measurements on the ground. Typically, the scale is determined using three main methods:

Representative Fraction (RF): ratio of the distances between two points, measured in the image and on the ground.

Photo Scale Reciprocal (PSR): inverse of the previous scale.

Relationship between the altitude from the camera lens and the focal length.

For all cases, the scale can be calculated based on the relationship between the altitude from the ground and the focal length of the camera lens [13].

2.15.2 Experiments setup and results

For the experiments conducted in [13], a small UAV platform (*PLP-610*) equipped with a *Nikon 1-v1* sensor and an onboard GPS was used to determine the lowest possible altitude for detecting specific failures and defects on PV modules in a real environment. The UAV flew at various altitudes to capture aerial photos. In [13], the UAV flew at heights ranging from 1 m to 20 m (in multiple increments of 1 m) above the PV modules. This provides insight into the relationship between the flight altitude and the detection of defects. Additionally, this experimental information is useful for determining the optimum height for calculating the scale.

Tab. 2.2 shows at what height different defects can be detected. The results indicate that the correlation between altitude and detection depends on various

Component	*Defects and failures*	*Detection height*
module	Snail trail	0-5 m
module	Dirty	10-15 m
module	White spot	15-20 m
module	Encapsulant discoloration	15-20 m
module	Different glasses	15-20 m
cell	Day4 technology	5-10 m
cell	Num. interconnect ribbon	10-20 m

Table 2.2: Table summarizing the different defects and faults detectable at different heights [13].

characteristics of the defect, such as its shape, size, color and location. As a result, specific defects can be detected at different altitudes during aerial visual inspection.

Low altitude flight is nowadays of key importance for industrial applications. In particular, for JP Droni, the objective is to develop an architecture capable of autonomously capturing all the bar codes of the PV panels. Since these codes are only a few millimeters in size, a very low flight altitude profile has to be kept during operation.

2.16 Post processing IM

When a UAV inspection is complete, a large image dataset is retrieved. This section shows how the IM processing algorithm provides more in-depth information regarding the analyzed plant. IM is a technique that combines multiple images, resulting in a more comprehensive view compared to individual ones; it can be used for various applications, such as resolution enhancement and motion detection. The algorithm provides important data from a PV plant analysis, like the number and location of the PV panels and the list of damaged ones [82].

2.16.1 IM techniques

Basically, IM techniques involve blending multiple overlapping aerial images to create a seamless, radiometrically balanced image without any visible internal boundaries. There are three types of IM:

1. *Uncontrolled mosaicking*: no horizon control is performed when patching the images together.

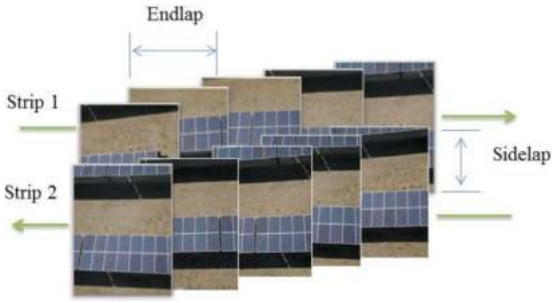


Figure 2.19: Example of IM, showing sidelap and endlap [13].

2. *Semi-controlled mosaicking*: images are enlarged and reduced to change the scale of the map and assembled with no rectification [83].

3. *Controlled mosaicking*: perspective distortion is removed before patching the images together.

According to industrial experience, to achieve good performances in PV plants (especially in large ones), *sidelap* (lateral image overlap) should be from 15% to 45%, while *endlap* (end of the picture overlap) should be from 55% to 65%. Generally, at least 40% endlap is necessary to obtain a decent overview of the string. For this reason, a vertically mounted camera is generally preferred over oblique mounting [82], as images taken along the vertical direction have significantly more overlap, which will help with the mapping and the 3D reconstruction of the environment [13]. Errors during image capture can heavily compromise its reconstruction. Two typical errors are *drift* (lateral displacement due to external causes like wind) and *crab* (alignment displacement due to incorrect sensor positioning).

2.17 Defects detection

The detection of defects is based on the processing of the image data captured by the drone. The identification of faults can be carried out based on CNN models trained specifically for this type of recognition. These models are either based on thermal [84] or RGB imaging [85]. This topic is deemed to be outside the scope of this thesis. Nevertheless, a defects detection system could easily be added afterward by either creating new classes (related to different types of defects) within the NN already in use or by training a specific NN for the identification of such faults.

Figure 2.20: Image of an actual hot-spot from the Pedrosa (AL) solar plant.

Chapter 3

Software and hardware architecture

The software and hardware components for the project are described in this chapter. The following sections focus on the hardware equipment, software components, and ROS architectures that incorporate the parts of the system design.

3.1 Hardware requirements

The hardware components that run the software architecture are as follows.

DJI Drone: the drone which performed all the tests is the *DJI Mavic 2 Enterprise Dual* model (App. A.1). This drone supports the *DJI mobile SDK*. The bridge between the drone and the computer is an Android application based on this SDK. Thus, any DJI drone supporting this SDK could interface with the software architecture developed here.

Radio controller (RC): it is the first interface with the drone. It communicates using a wireless connection with *OcuSync* technology, a transmission system developed by DJI.

Android mobile phone: once connected via USB to the RC, the Android mobile phone exchanges data between the drone and the laptop through a *WiFi*-based socket connection created by the custom app developed.

Linux-based Laptop: the data from the UAV is processed by the computer unit (App. A.2). The image processing is based on the segmentation driven by a NN running on the PC. This processing technique is computationally expensive and the use of a *Graphic Processing Unit* (GPU) is required. The *Detectron2* package used for detection requires graphics data-processing

based on *NVIDIA GPU boards* driven by *CUDA* for the exploitation of parallel programming capabilities.

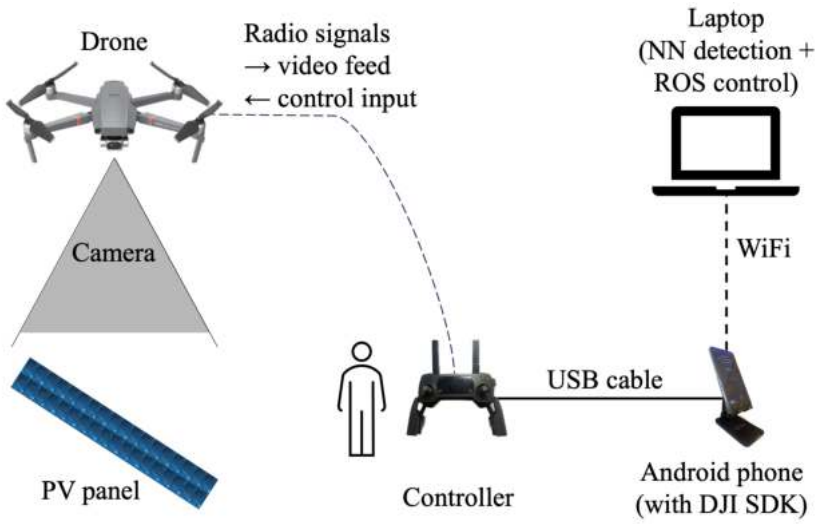


Figure 3.1: Interfaces between the different hardware components.

3.2 Software requirements

This section explains the apps, software versions, and libraries required to run the whole system. It is divided into subsections based on the hardware component on which the software is installed.

3.2.1 PC

The architecture can be conceptually divided into three sections, dedicated to *route planning*, the *ROS bridge*, and *UAV control*. Given the heavy computational load, a powerful computer is needed for such tasks. The computer specifications are listed in the (App. A.2) appendix. The computer has the following setup:

Linux: the version is *UBUNTU* 20.04 *LTS*. This distribution of is the only one fully compatible with the chosen ROS version;

Python 3: the last Python 3 version (3.11.4) is required;

ROS: the system runs the *Noetic* version of *ROS*;

CUDA: CUDA is a software layer that gives direct access to the parallel computational capabilities of the GPU. It can be installed only on devices supporting *NVIDIA* graphic cards. The installed version of CUDA is 1.17.

In the following, we present the installed libraries for running the system, which is divided in two parts that can work independently.

3.2.1.1 Route planning

The libraries and packages necessary for the architecture are reported in Tab. 3.1.

Name	Library/Package description	Version
PyTorch	PyTorch is a ML framework based on the *torch* library. It is the framework on which the library *Detectron2* is based.	1.10, *CUDA* 1.11.
Detectron2	A *Meta* project, evolved from *Detectron* and *maskrcnn-benchmark*. It is a Pytorch-based object detection and object segmentation library and allows *fine-tuning* many state-of-the-art, pre-trained NNs. The version has to match the PyTorch and CUDA ones.	0.6 with: *Torch* 1.10, *CUDA* 1.11.
Numpy	A Python library for scientific computing, defining multidimensional arrays, derived objects, and array operations.	1.25.0
Matplotlib	Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.	3.7
scikit-learn	A ML library based on NumPy, SciPy, and matplotlib. It is a simple and efficient tool for predictive data analysis.	1.2
Python-tsp	A library written in pure Python for solving TSPs. It can work with symmetric and asymmetric versions.	0.3.1
Rasterio	This library uses *GeoTIFF* format data to organize and store gridded raster datasets such as satellite imagery. In this project it was used for retrieving altitude information from a .TIFF file.	1.3
Pillow	A fork of the PIL library for Python image processing.	9.5.0
Pykml	A package for creating, parsing, manipulating, and validating KML files. KML is a language for managing geographic data.	0.1.0
JSON	A Python package for creating, parsing, manipulating, and validating *JavaScript Object Notation* (JSON) files, a lightweight format inspired by *JavaScript* object literal syntax.	3.11.4
OpenCV	A Python for computer vision, it includes hundreds of algorithms. The arrays are managed through the *NumPy* format array.	3.4.19
CPLEX	A high-performance mathematical programming solver for linear, mixed-integer and quadratic programming by *IBM*, it can solve huge optimization problems.	22.1.0

Table 3.1: Libraries and packages required for the route planning.

3.2.1.2 Bridge application

The bridge application is a communication channel first introduced in previous thesis work. It works as a data-bridge between the drone and the computer. It is a socket-based architecture where the RC, connected to a mobile phone, exchanges information from the UAV to the computer and viceversa. The communication interfaces are explained in Chap. 4. For the communication among these devices, multiple libraries, packages, and applications have been installed, as detailed in Tab. 3.2.

Name	Library/Package description	Version
Android Studio	The official *Integrated Development Environment* (IDE) for Android development by Google, it provides a set of tools and features for developers to create, debug, and deploy Android apps.	2021.1.1.19
CoppeliaSim / V-REP	A robotics simulator (formerly *V-REP*), it is based on a distributed control architecture: each object/model can be individually controlled via an embedded script, a plugin, a ROS node, a remote API client, or any other custom solution.	4.5.1 EDU
Numpy	See Tab. 3.1.	1.25.0
OpenCV	See Tab. 3.1.	3.4.19
Json	See Tab. 3.1.	3.11.4
Detectron2	See Tab. 3.1.	0.6

Table 3.2: Libraries and packages required for the bridge connection

3.2.1.3 UAV Control

The UAV control is based on two different control strategies depending on the type of flight. The UAV is required to either fly over the panels and conduct an image acquisition process, or to fly from a panel line to another without acquiring pictures of the plant. Accordingly, the control is divided in two *ROS actions*:

A **WP connection-based controller** for flights from one panel line to another. This implies implementing a connection between two WPs based on their geographical coordinates. This could be automatically carried out by the DJI software embedded in the drone system. However, for the indoor tests setup (chap.7) the GNSS signal necessary for the WPs connection was not reliable. Therefore, to simulate this capability, a PID controller was developed, both inside the *V-REP* simulation environment and in the *OptiTrack* motion capture system [86]. To interface with ROS, the *Optitrack ROS pakage mocap* [87] was added to the architecture.

Three different **vision based controllers** were developed and tested. To properly use them, an additional ROS package for camera calibration was installed.

All the control messages and techniques are based on the Numpy library and on the ROS libraries included in the packages.

3.2.2 Android mobile phone

The Android mobile phone is a key component for the bridge communication channel. Once connected to the RC (by a USB cable), it opens a two-way communication between the computer and the UAV for data exchange. The capability is provided by the Android app installed on the device through *Android Studio*. For compatibility reasons, the device needs a version of Android between 5.0 and 11.0.

3.3 ROS architecture and interfaces

The UML graph in Fig. 3.2 describes the structure of the software. The substructures are examined in further detail in the following subsection.

3.3.1 Interfaces description

Satellite data The data flow for the high-level architecture starts with the extraction of geo-coordinates for the identification of the WP positions. The *satellite data* module identifies latitude-longitude coordinate pairs and send them to the *drone control* module through a ROS based *service-client* connection on a service called */wp_coords*.

Drone control The *drone control* module, then, starts sending commands to the UAV socket connection module through a *publish-subscribe* ROS topic. These commands are based on the data extracted by the subscription to the *UAV connection* module which sends back the sensors data containing the information related to the current altitude and tilt camera angle (Chap. 4). The location data is the estimated position of the panel to be followed.

UAV connection The data extracted by the UAV connection is sent to the rest of the system on a publish-subscribe basis. The altitude information is calculated by two sensors, namely, a *barometer* and a *proximity sensor*, both of which stream on their own topic connection. The connection opens up the stream of images

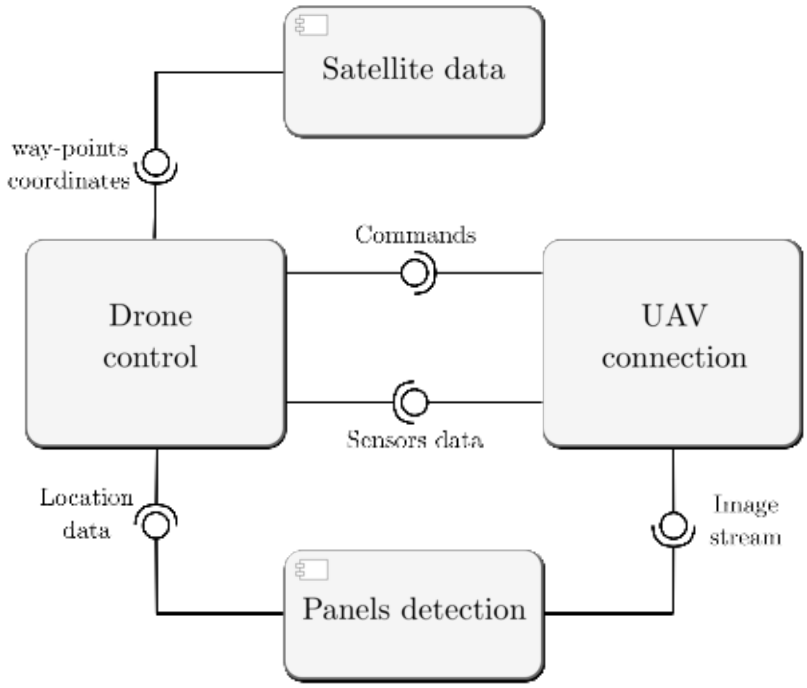


Figure 3.2: High-level UML graph of the interfaces.

from the drone camera to the PC. The image data is streamed with a compressed format which has to be decoded and then republished to another topic. The decompressed image is then published on the *Image stream* topic.

Panels detection Through multiple processing steps, this module calculates the panel position based on the aerial image and publishes it on the *location data* topic. Depending on which controller is activated and on the image modification required, the panels detection module subscribes to different image-streaming topics (Sec.6.4.1). Then, the *Drone control* module extracts the necessary information to compute the correct control input.

3.3.2 UAV connection module inner interfaces

Each box in Fig. 3.3 represents a ROS node contained in the UAV socket connection portion of the architecture. The following summary analyzes the sub-structures and connections.

Img Listener UDP This node receives the video stream from the bridge application and it republishes it to the topic */output/image_raw/compressed* with

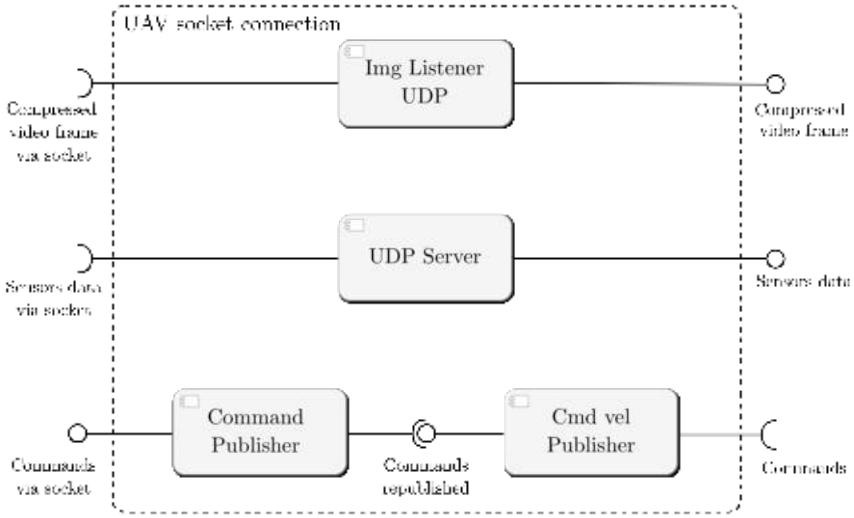


Figure 3.3: UML graph of the socket connection interfaces.

a JPEG compression. The *Republish* node from the ROS library then decompresses the image to a raw format on the */output/image_raw* topic.

UDP server This node opens the communication for the sensor data to be streamed on the ROS topics of the architecture. On a previous implementation of the bridge app, only the data related to the drone altitude was streamed. This information is published on the */barometer* and *ground_distance* topics. For this thesis work, the tilt angle of the camera gimbal was added.

Command publishers Commands are sent to the drone in two steps. First, the commands are passed to the *Cmd velocity Publisher* node which republishes the commands retrieved on the topic */command* to the topic */cmd_vel* at a rate of 15 Hz. This is required by the drone to accept the command (the drone accepts commands at frequencies from 5 to 25hz). The *command publisher* node then takes the stream from the *cmd_vel* topic and publishes it through a TSP socket connection to the drone.

3.3.3 Drone control inner interface

The interfaces within the drone controller portion of the architecture are based on two ROS custom actions for managing the autonomous movements of the drone (Fig. 3.4).

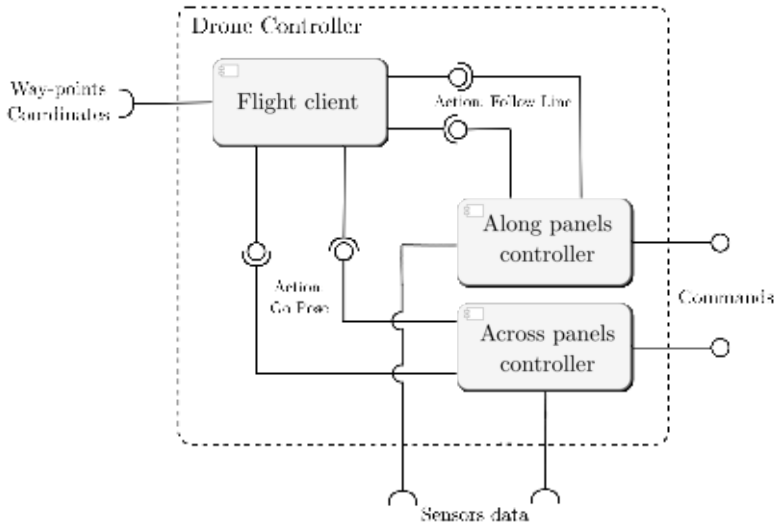


Figure 3.4: UML graph of the controller interfaces.

Flight client This node behaves as a single task parser for the controller. It receives the WP coordinates identified by the satellite data extraction node and executes an action to move the drone either along a single panel line (*follow_line action*) or from one panel line to another (*go_pose action*).

Along panels controller This is the visual servoing node; three different nodes can be used, depending on the control technique desired by the user (Fig. 3.4). All three nodes gather information from the sensors to create a control input published on the */commands* topic.

Across panels controller This node has similar interfaces as the previous one, but here the movements are not based on image data but only on telemetry and altitude information. This information is served to the controller through the *OptiTrack* data configuration managed by the dedicated package. This node could also be replaced by the WP-based DJI controller already included in the DJI mSDK (Sec. 4.2.2.2).

3.4 Simulated interfaces and drone

Preliminary flight tests were conducted in simulation on the latest version of *CoppeliaSim*. This simulator can simulate many types of robots, including UAVs.

The drone models can be controlled with different methods, such as embedded scripts, plugins, ROS nodes, and many others [88]. The drone model chosen for the simulation is the *quadcopter.ttm*, to which an ideal weightless camera and proximity sensor were added, to simulate the actual sensors of the real UAV (Fig. 3.5).

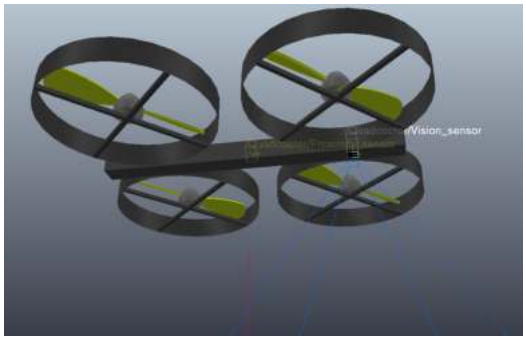


Figure 3.5: The quadrotor model.

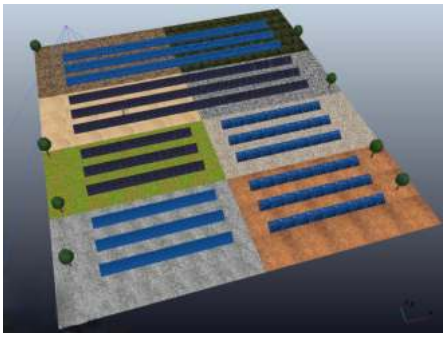


Figure 3.6: Simulation environment.

The ROS interfaces are coded in the *Lua* programming language. The script publishes to the designated ROS topics the following data:

/output/image_raw The image stream, already formatted in a raw standard, simulating the image stream of decompressed images from the actual drone.

/ground_distance The altitude data from the simulated proximity sensor.

/barometer_altitude The same message as the one published on the */ground_distance* topic.

/quadrotor_pose Publishing a *geometry_msgs/Pose* data containing the x and y coordinates of the quadrotor position, for simulating the telemetry data of the *OptiTrack* system.

To complete the network simulation of the socket connection, a subscription to the */command* topic is defined. Thanks to this configuration, the commands calculated by the actual controller used in the real-world simulation were sent to the simulated drone.

Chapter 4

Bridge application

The bridge app is a socket-based data bridge installed on a phone attached to the RC and connected to the computer managing the drone control. In the first part of this chapter, an outline of the already existing app is given. The second part describes the newly introduced features needed for the control techniques of Chap. 6.

4.1 Application development

Part of a previous thesis work was the further development of an already existing Android *bridge_DJI_ROS* mobile app, which establishes the bridge connection. This section introduces the main features of the app and how it works. The application was developed in Java using the Android Studio IDE. It is based on the mobile SDK developed to allow programmers to interface with DJI products (App. A.1).

The application develops a socket connection network for communication between the PC and the phone, exchanging sensor information and commands regarding the UAV flight. The socket connection is established on a *Wireless Local Area Network* (WLAN), opening the socket connections listed in Tab. 4.1.

Data	Type	Dimension	Rate	Protocol	Client/Server
Ground distance	Float	4 B	20 Hz	UDP	Phone/PC
Barometer altitude	Float	4 B	20 Hz	UDP	Phone/PC
Frame	JPEG	50 KB	Max	UDP	Phone/PC
Command	Custom	16 B	15 Hz	TCP	PC/Phone
Gimbal tilt	Float	4 B	5 Hz	UDP	Phone/PC

Table 4.1: Socket connections.

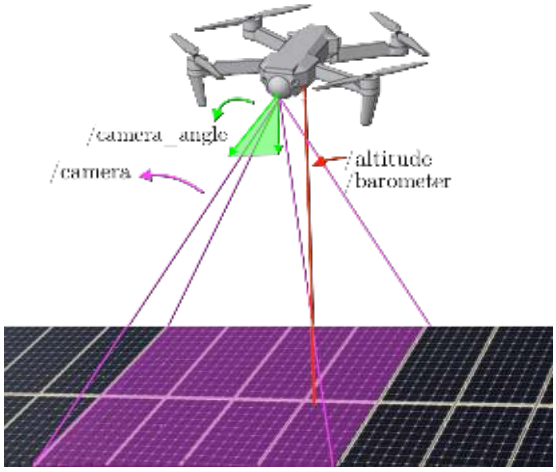


Figure 4.1: Schematic of an inspection, with the drone velocity (in teal), camera stream (magenta) and altitude (red) from the proximity sensor and barometer
.

Tab. 4.1 shows the *type*, *dimension*, *direction*, *protocol*, and *stream rate* of every data entity for each open socket. The difference between the types of transport layer protocol is due to the different nature of the messages exchanged between the two platforms.

Transmission Control Protocol (TCP): it ensures reliable, ordered, and error-checked data transmission between devices over networks. In case the received data is not correct, TCP re-sends the lost piece of information. This reliability is exploited for the PC-to-phone velocity publishing, to never lose velocity control signals.

User Datagram Protocol (UDP): it is a fundamental part of the Internet Protocol Suite and offers a simple, connectionless way of communication for networked devices. It offers speed and simplicity at the expense of data-reception reliability. This protocol is exploited for the phone-to-PC data transmission where sensors data (sampled at high frequency) is exchanged and error checking is superfluous.

4.2 Application improvements

New features, required for the developments in drone control presented in this thesis, were added to the socket connection network and to the application in general. The following subsections explain these features and how they interface with the already existing app.

4.2.1 Tilt gimbal angle data transmission

In Chap. 6, the visual-based control techniques for the UAV flight are explained. The visual servoing based on *task stacking* requires further data to be exchanged by the socket connection, namely, the tilt angle of the UAV. This information is extracted through the DJI SDK and republished on a 5 Hz UDP. With this, it is possible to adjust the control angle on tilted rows of panels for the task-stacking control (Sec. 6.4).

4.2.2 Flight across rows of panels

After reaching the end of a panel row, the UAV has to fly between the end of the line just tracked and the beginning of the following one. This flight across rows of panels is not based on the image stream, but on the UAV location. The following two subsections explain how this problem has been addressed in *indoor* and *outdoor* environments.

4.2.2.1 Indoor flight

As explained in Chap. 7, the experimental setup for the evaluation of the control algorithms proposed in this thesis work is in a protected indoor aviary inside the EmaroLab at the University of Genova. Experiments based on GPS location within indoor environments are not reliable due to signal corruption. To represent the movement of the UAV inside the indoor environment, the *OptiTrack* motion capture system installed in the test area was used (Sec. 6.5).

4.2.2.2 Outdoor flight

The outdoor UAV flight is based on the GPS signal. The current app collects the velocity commands calculated within the ROS architecture and passing them to the UAV, through a control mode called *VirtualStickControlMode*. The drone movement based on the GPS data is managed instead by a different control mode, called *WaypointMission*.

The TCP socket from ROS to the bridge app exchanges messages regarding the four command velocities for DJI drones (v_x, v_y, v_z and ω_z). For starting the *WaypointMission* mode, the ROS controller sends to the app a known modified version of the velocity message, where v_x, v_y and v_z are equal to the latitude, longitude, and altitude values of the objective point, and ω_z is equal to a specific key-value used to switch between modes. The latitude, longitude, and altitude values are then extracted to create a new WP instance, which is set as the next target point of *WaypointMission*, along with other mission configuration

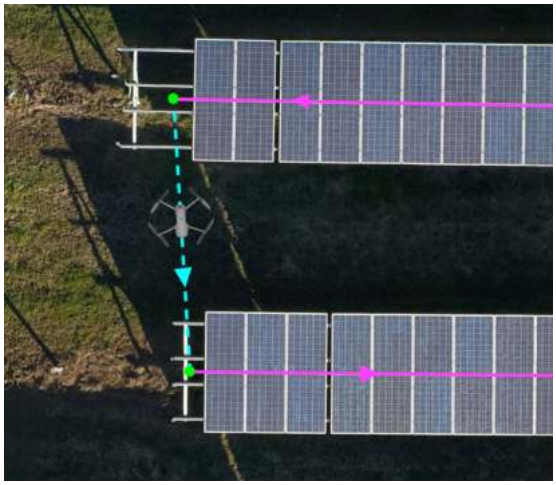


Figure 4.2: In teal: the WP-based movement across the panel.

parameters such as speed, heading, and actions. Then, the mission is started and the *VirtualStickControlMode* is suspended until reaching the target WP.

This feature could replace the current node used for movement across panels: an idle state in the node for moving along panels could wait for the end of the *WaypointMission*. While this feature has already been developed, it has not yet been tested because the indoor setup did not support GPS signal (Fig.4.2).

4.2.2.3 GPS position errors

From the specifications of the UAV model used for this thesis work (App. A.1), when starting the *WaypointMission*, the position error from the GPS signal could reach ± 1.5 m during active flight and ± 0.5 m during hovering. Therefore, the following problems could arise during outdoor applications.

Misalignment errors: when reaching a new WP, there could be issues in seeing the new panels inside the camera field of view, making it impossible to automatically start the inspection.

Battery issues (Chap. 5): when the drone's WP-based flight does not follow accurately the intended trajectory many times in a single visit, the total path length could be larger than the one considered while planning the flight. For example, the trajectory across the panels could be longer, or the UAV could be off the desired position, so a deviation is necessary during the visual servoing portion of the flight. These issues could lead to battery draining if not taken into account when choosing the maximum flight length (Sec.5.5).

Most of these issues can be solved by changing the drone model used to one supporting the RTK technology. This feature adds a high-precision positioning solution to improve the accuracy of GNSS-based navigation systems, particularly in applications such as mapping, where centimeter accuracy is required [26].

Chapter 5

Planning the flight route

This chapter presents a portion of the thesis project aimed at automating the extraction of satellite images of PV installations from *Google Earth* servers and generating an optimal path to visit them, considering the drones' battery limitations.
This portion of the project is developed both as a ROS node integrated in the project's software architecture (Fig.5.1) and also as a Python module which will be integrated into the ***JPVision*** application. *JPVision* is an under development application capable of autonomously plan and carry out all kinds of facilities and infrastructural inspections through the use of DJI UAVs.

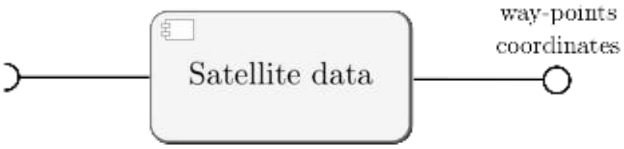


Figure 5.1: ROS node which processes the image.

5.1 Current solution to the problem

As mentioned in Sec. 4.2.2.2, DJI drones can follow pathways defined by geographical coordinate WPs. However, manually picking these points is time-consuming and inefficient. Moreover, manual picking implies a lack of systematic analysis to establish the best route for the drone since it is difficult to find a flight pattern which is optimized to account for battery draining. Advanced drone software and flight planning tools can be utilized to address these limitations.

The application developed for this thesis uses algorithms and optimization techniques to compute the most efficient path, taking into account aspects like *battery level*, *flying time*, and *distance* between WPs. These solutions, can dramatically improve the drone's efficiency, productivity, and overall flying performance by automating and optimizing the path planning process.

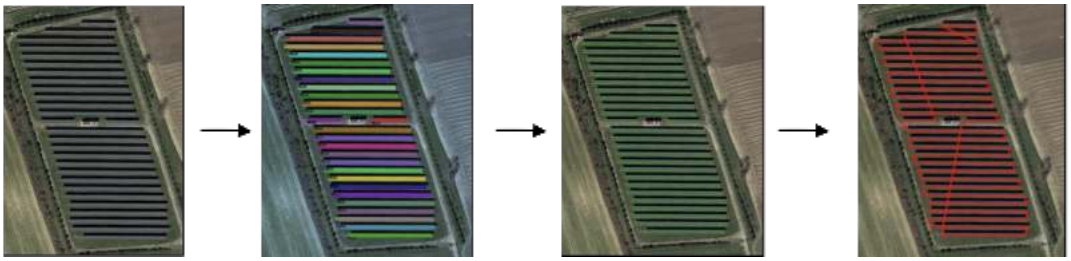


Figure 5.2: Pictorial summary of the image processing carried out by the module.

5.2 Automatic solution to the problem

In this section, the extraction, processing, and optimization of the data extracted by the satellite imagery will be explained in detail. The following *UML Activity Diagram* will schematize the whole process of extracting the images give specific geo-coordinates to the acquisition of the set of geographical WPs (Fig.5.3). The *Activity diagram* is a flowchart that represents the flow of control among the activity of a system. This kind of representation is necessary for understanding all the steps the node takes to extract the final information.

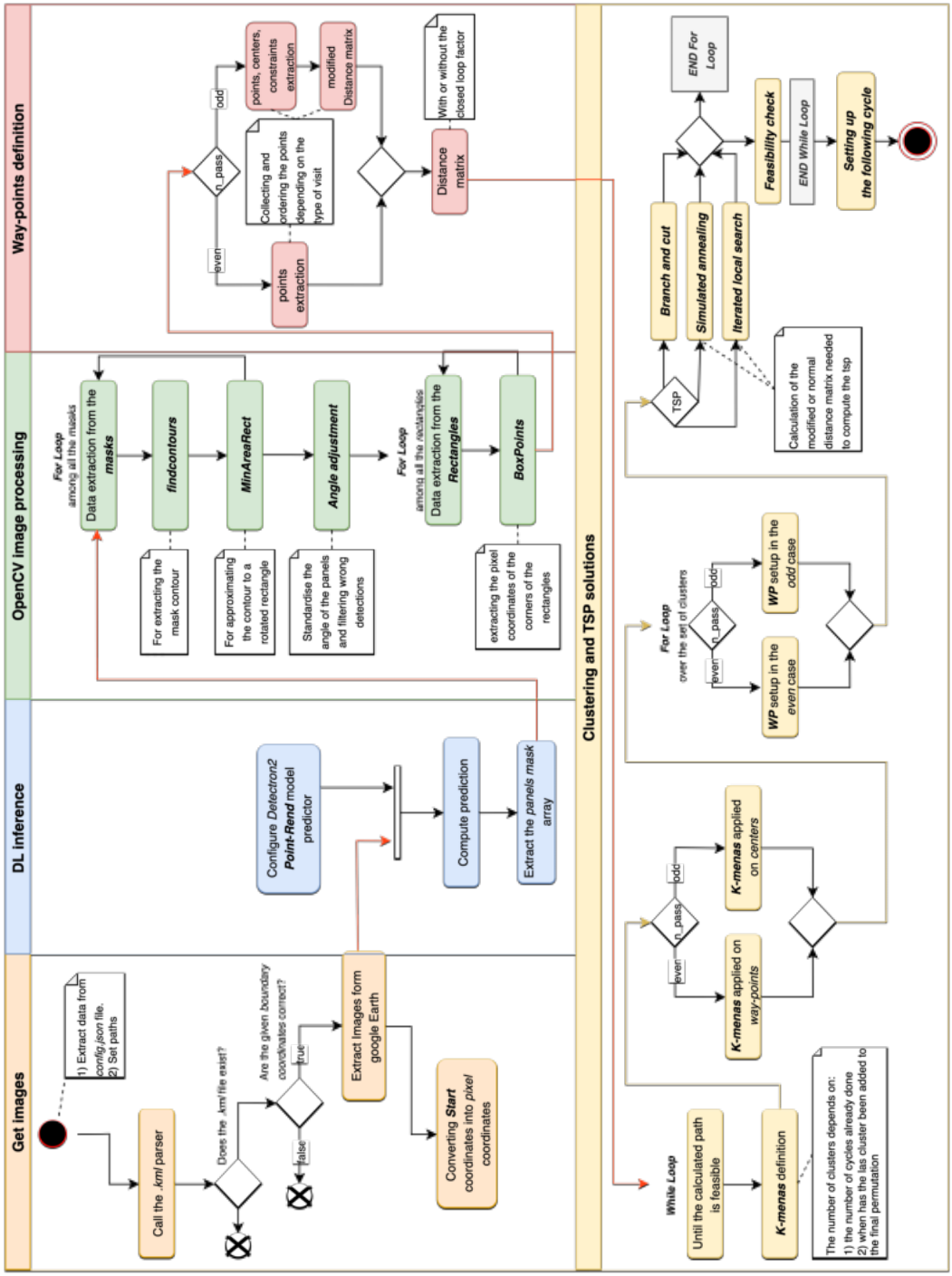


Figure 5.3: UML Activity Diagram of the node.

5.2.1 Satellite image acquisition

In this section the satellite-image acquisition will be discussed by focusing on the important steps for performing the process.

5.2.1.1 Coordinates selection

The image extraction process starts with the user's manual parsing of a *.kml* file. *.kml* is a *XML* based extension which displays geographic data of an Earth browser, such as *Google Earth*. The data which the file contains is:

A **polygon**: A closed polygonal line geo-localized in the world map. The user is required to create a polygonal shape around a certain geographical area containing the PV plant of interest.

A **key-point**:A geo-localized point in the world map. The user is required to set a *key-point* in a certain geographical location which identifies the starting point and charging-pit of the UAV.

The second step involves collecting the data for the image extraction from the *.kml* file. The *top-left* (TL) (corresponding to the *lowest and highest latitude and longitude* of the *polygon*) and *bottom-right* (BR) (corresponding to the corresponding to the *highest and lowest LAT, LON* of the *polygon*) coordinates are collected. These points define a rectangle containing every panels of the plant in the extracted image.

5.2.1.2 Image extraction

Once the TL and BR coordinates are extracted, the *sat-img.out* will execute the extraction of the satellite images. This executable was provided by the partner *JP Droni* as part of the *JPVision* project.
Given the TL and BR coordinates, the executable is capable of extracting the portion of land that lies within the two points from the *Google Earth* servers. The application developed by *JP Droni* is based on the *Earth Engine APIs* (EE). *Google Earth Engine* is a *Google* project that blends planetary-scale analysis capabilities with a multi-petabyte database of satellite photos of an accessible geographical datasets [89].
Google Earth and *Google Maps* export pictures using rectangular tiles each of which represents a different part of the Earth's surface. The zoom level of these image instances, goes from *level 0* to *level 20*. The side length corresponding to one pixel between the *21 levels* of zoom spans from $156km$ at *level 0* all the way to $0.15m$ at *level 20*. The given application is able to download a tile-stitched image with a zoom level of*20* which returns the highest detailed image possible.

5.3 Deep Learning Instance Segmentation

This section is dedicated to describing how to extract a binary image which distinguishes pixels depicting panel rows from background pixels within the previously found PV plant satellite image.
The following subsections will focus on explaining the steps that led to the choice of the NN model used, how it was trained, and how the training dataset was created.

5.3.1 YOLO

You Only Look Once (*YOLO*) is a popular object detection and image segmentation model. It was first developed by *Joseph Redmon* and *Ali Farhadi* at the *University of Washington* for their company *Ultralytics* [29]. It is one of the fastest and the most used algorithms for object detection porpoises. Due to its proven speed it was the first method exploited for the identification of the panels.

YOLO v8: Object Detection: Evolving from YOLOv5, YOLOv8 is the state-of-the-art NN model supporting *object detection*, *image classification*, and *instance segmentation* capabilities. The choice of this model comes from the first idea of identifying the position of the panels in the image through *object detection Bounding Boxes* (BBs). However, BBs do not account for any kind of object rotation in the image meaning that the boxes would always have to be parallel to the image frame. After discovering that photovoltaic plants are not perfectly aligned with the general configurations *North-South* or *East-West* (Roof installations are an example of this misalignment) this method was discarded since the final aim was to perfectly identify the extremes of the rows. However, during the first developments of the project, this issue was not considered as a big limitation because most of the PV pants are built following the correct orientations anyways. Therefore, the post processing steps were already able to roughly identify the end points of the lines of panels (Fig.5.4). The development of the *Path optimization* for the UAV flight was then conducted on the output of this first model.
Since *YOLOv8* supports a *segmentation* models, one of the NNs tested was the *YOLOv8m-seg* model. However, the obtained masks were not enough tight on the actual rows bounds.

YOLO v5: Rotated boxes Object Detection: The *YOLOv5-Oriented Bounding Boxes* (YOLOv5-OBB), is designed to yield predictions that better fit objects that are positioned at an angle in an image. Using *YOLOv5-OBB* is possible to detect lines of panels that are rotated on a given frame more tightly and accurately. *YOLOv5-OBB* is a single-stage rotation detector

based on the *RetinaNet*. The identification of the orientation is based on a *regression-based prediction branch* and a *CSL-based prediction branch*. More on this can be found inside the related paper [90].
The results obtained with the fine tuning of this model were way better than the results obtained with *YOLOv8* but the bounding boxes were still not very tight on the panels contours and some OBBs were not very much aligned with the rows they were identifying.

Figure 5.4: *.a*: *YOLOv8* Bounding Boxes;*.b*: *YOLOv5* Oriented Bounding Boxes.

5.3.1.1 Object Detection Data-set

Since no dataset of satellite images of photovoltaic rows of panels was found on the internet, a manual *image annotation* process was necessary to create a fairly big dataset for training the DL models. For training both the *YOLO v8* and *YOLO v5* models a *bounding box* dataset was created using the *LabelBox* online application. For the sake of this project, the tool was used uniquely for the annotation process and not for training the model.
In *supervised learning*, data annotating is the process of identifying raw data (in this case *images*) and adding one or more meaningful and informative *labels* to provide context so that a ML model can learn from it. To create a data-set two steps are necessary:

- *Collecting* a good amount of satellite images representing large scale solar plants.
- *Labeling* the images. Therefore, manually identifying the object drawing tight closed bound lines around them.

For the training of the *YOLO* models, the image acquisition process was skipped since an open-source data-set for the identification of PV plants contours was found on *Zenodo* [91]. However, a manual labeling process had to be carried out to locate the rows of panels in the images.
This first iteration of the dataset counts a total of *229 labeled images* divided into the three sub-sets: *Train set* (70%), *Validation set* 20%, *Test set* 10%. The dataset was then stored on the *Roboflow* database [92]. *Roboflow* is an online applicaiton that lets you store your own dataset on their platform. Thanks to an easy-to-use API interface it is possible to download it before the training step. Thanks to this application, it is possible to *resize the images* and *augment the dataset*.

Resizing: The resizing to a standard image resolution is an advantage when training a NN since it increases the over all *memory efficiency* of the process. This implies a faster convergence during training process. Moreover, having consistent image sizes ensures that *augmentation* operations can be applied uniformly across the entire dataset.

Augmentation: Image augmentation is a data processing technique for training NNs. It involves applying various transformations to the original training images, such as rotation, scaling, flipping, and brightness adjustments, to create new augmented versions of the data-set. This technique expands the size of the data-set without requiring additional labeling. It introduces more robustness to the model reducing overfitting. However, Augmentation can also lead to an increase in training time since each training epoch has to process a larger number of images. Moreover, excessive augmentation or inappropriate transformations may introduce unrealistic artifacts in the images which could mislead the NN training, leading to worse performances of the model itself.

The *resizing* for this data-set was set to an image resolution of 640×640.
The *augmentation* involved: *90° Rotate: Clockwise*, *90° Rotate: Counter-Clockwise*, *Image Flip*. Resulting in a data-set three times bigger than the original.

5.3.2 Detectron2

Detectron2 is an open-source deep learning framework developed by Facebook AI Research that mainly focuses on computer vision tasks, specifically *object detection*, *instance/semantic/pan-optic segmentation*, and *key-point* detection. It is based on *PyTorch* and provides a modular architecture, making it easy to customize and extend for different research needs and applications. *Detectron2*

offers state-of-the-art implementations of various cutting-edge object detection algorithms, such as *Faster R-CNN*, *RetinaNet*, and *Mask R-CNN*. The *Detectron2* repository [30] also includes a directory named *projects* where some Facebook research works are collected. One of these projects oriented towards *segmentation* is called ***PointRend*** which was used intensively to train a neural network capable of perfectly identify the orientation and the contour of the panels.

5.3.2.1 FPN + PointRend: Instance Segmentation

As aforementioned in Sec. 2.8.3, the *U-net* is a typical approach when it comes to the precise segmentation of small elements within satellite images. However, the article [93] of the famous repository [94] on *satellite images and deep learning* proposes a comparison between The *U-net* and the Basic *FPN + PointRend model* [14], which is reported to significantly improved segmentation performances over this type of imaging.
PointRend is capable of recognizing that different areas of the output mask which may require a different amount of information, in contrast to conventional CNNs that operate with regular grids of features. The regular grid representation of CNNs distributes Computational effort uniformly across all spatial positions, leading to excessive amounts of computation for easier areas and an under-balanced detail for complex regions. Additionally, *PointRend* makes predictions for a set of sampled points using an implicit function technique, gradually improving the prediction just for the subset of points (Fig.5.5). Moreover, it uses *bi-linear interpolation* to gradually enhance the resolution by starting with a coarse prediction that can be computed efficiently and only tweaking predictions for points that need higher resolution. With the help of these additional features, high-resolution segmentation can be added to current CNN-based image segmentation models, resulting in improvements in quantity as well as quality [14].
As a matter of fact, the studies' findings of the article [93] demonstrated that *Detectron2 FPN + PointRend* outperformed even the *U-Net* by a 15% increase in accuracy when comparing the results on the same validation dataset after the same fine tuning. These results demonstrate that using *PointRend* over *U-Net* may be beneficial in terms of output performance.

Therefore, two of the instance segmentation models were compared on a new data-set created for such task: Mask_rcnn_R_50_FPN and *pointrend_rcnn_R_50_FPN* both pre-trained on the *COCO* data-set.
As explained in the experiments Chap7, the best performing model with the *coco validation* resulted to be the *pointrend_rcnn_R_50_FPN*. Therefore, the fine tuned weights for the new class were used for the identification of the lines of panels.
Using this setup turned out to be a grate improvement in terms of tightness of the

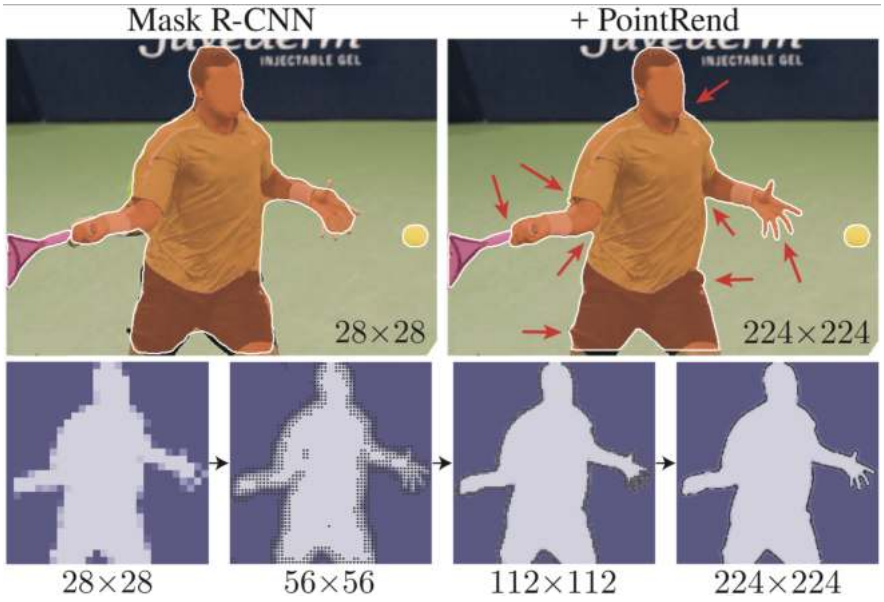


Figure 5.5: *PointRend* segmentation comparison [14].

masks (Fig.5.6). Thanks to the following *image processing* steps, the identification of the WPs turned out to be perfect in terms of positioning with respect to the panel lines.

Figure 5.6: *Point-Rend* segmentation.

5.3.2.2 Instance Segmentation Data-set

The dataset was developed in a similar way compared to the previously mentioned one. However, the labeling and the storing of the dataset were both carried out directly on *RoboFlow*. The most recent update of the *RoboFlow* software included a new key-feature regarding the labeling of *segmentation* based data-sets annotation. The newest *Meta AI* project called *Segment-Anything* (*SAM*) was included in the *Roboflow* workspace right after release.
SAM is an instance segmentation model released on April, 2023. SAM was trained on 11 million images and 1.1 billion segmentation masks. This model is addressed

as a *Zero-Shot Learning* (*ZSL*) model. It is possible to generate segmentation masks of any object without any prior training on any class [95]. This result is obtained by pre-training the model on a set of known classes and then giving it the task of generalizing to a specific collection of unknown classes without any additional training [92]. However, since the lines of panels in the images are very small compared to the picture of the PV plant, the SAM poly-line had to be corrected quite often.
Differently from before, the image collection was carried out manually. Thanks to a dataset from the solar inspection conducted by *JP Droni* back in 2020, it was possible to quickly locate many photovoltaic plants on the Italian territory from which the pictures of satellite images were collected at different satellite magnitudes. To improve the robustness of the neural network with respect to false positives, images without any solar panel were added.
The final dataset was composed of 470 images including more than 10000 annotations. Just like before, the dataset was split into the three sub-sets: *Train set*(70%), *Validation set* 20%, *Test set* 10%.
The *resizing* for this data-set was set to an image resolution of 640×640.
The *augmentation* involved: *90° Rotate: Clockwise*, *90° Rotate: Counter-Clockwise*, *Image Flip*, Rotate: Between -20° and +20°, *Brightness*: Between -20% and +20%. Resulting in a data-set three times bigger than the original with a total of 1112 images split in the sub-sets with the same percentages.

5.4 Image processing

As aforementioned in the previous sections of this chapter, the image *processing step* aims at filtering and extracting the set of WPs in geo-coordinates for visiting the entire PV plant. The following steps will explain how such activity is carried out in the code.

5.4.1 First OpenCV processing

5.4.1.1 FindContour

The first step is to identify the contours of the panel rows of each mask. This operation is based on the *contour identification* of an objects and it is carried out by the *OpenCV* function called `FindContour`. After its application computation a set of *contour elements* will be collected and sent to the following processing step.

5.4.1.2 MinAreaRect

The following step will concern the approximation of each of the just found *contours* into an *oriented bounding box*. The `MinAreaRect` is capable of executing such processing step. The oriented box element is composed by three different parameters: **Center** point described by a pair of (x, y) pixel coordinates, the **Width** and *Height* of the box, and the **Orientation Angle** of the box in the image.
Depending on the *Orientation Angle* the *Width* and *Height* of the boxes the identification of the correct orientation with respect to the image may not always be intuitive. The *orientation* value lies within $[-90, 0)$ deg with respect to the rectangle side representing its width, which lies the farthest to the right side of the image. If that side is rotated more than 90 deg, then the adjacent edge is considered as the width side, therefore it is used to calculate the angle from the horizontal. Therefore, a farther step is introduced to force the solutions into three different categories depending on the orientation-configurations:

East-West: In this case, regardless of the orientation, the *Height* value will be set as the smallest one between the given *width* and the *Height* values. This configuration is the most common among the Italian plants.

North-South: In the case of this configuration, the *Width* value will be the smallest and the *height* the highest.

Roof plants: Since the panels orientation follows the orientation of the building one of the two configurations will be adopted based on the average angle of the panels.

5.4.2 Filtering and Standardizing

The following step consists in adjusting the overall orientation of the rectangles obtained. Generally, the global orientation of the plant's rows remains constant through out the whole facility. This fact was exploited for filtering and refining the *MinAreaRect* output.
Taken every orientation of every panel of the set, the *Standard deviation* σ and *Average* μ were calculated. Thanks to this data, the following two steps where taken:

1. **Filtering** any wrongly identified shapes for the DL process.
2. **Standardizing** the remaining angles to have a similar orientation angle.

The *filtering* consists in eliminating from the collection the rectangles whose angle is out of a $\pm 2 * \sigma$ bound from the *average* only if the σ value is over a predetermined *threshold* of 3 deg. If the σ value is under this *threshold* value, no filtering of this kind is applied.
The *Standardizing* process consists into taking the remaining panels and calculating both *standard deviation* and *average*. Once obtained, a final average of the elements in the $\pm\sigma$ bound is collected. This angle is then given to all the elements that lie within the initially set bound. This adjustment will make the angle of the panels more coherent to the initial assumption of having them all arranged with the same global orientation.

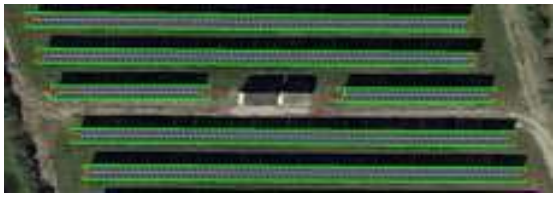


Figure 5.7: Oriented boxes.

5.4.3 OpenCV WP pixel coordinates identification

After managing the data contained in the *rectangles* representing the panels, the last few steps for the identification of the global WPs begin.

5.4.3.1 BoxPoints

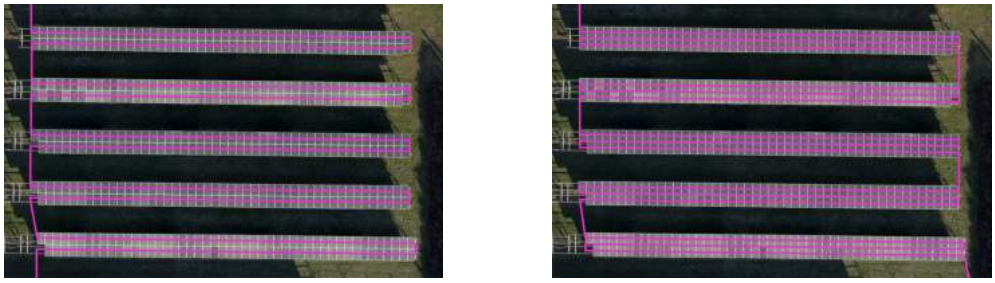
Starting from the information stored in the *rectangle* elements, it is possible to identify the coordinates of the corners of the rectangles thanks to the *OpenCV* function `BoxPoints`. Given the *rectangle* data as input to this function, four x, y *pixel coordinates* are given each one related to the rectangle's corners.

5.4.3.2 WP identification

The four corners of each rectangle are separated in two groups depending on the panels configuration:

Left-Right for *E-W*: Where the two points with the smallest x *pixel coordinate* and the two with the highest are coupled.

Top-Bottom for *N-S*: Where the two points with the smallest y *pixel coordinate* and the two with the highest are coupled.



(a) Two lines path. (b) Three lines path.

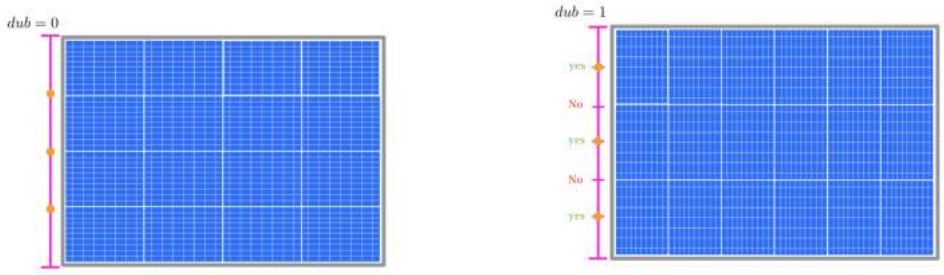
Figure 5.8: Side by side figures

(For the rest of this section, only the *East-West* configuration will be described. The *North-South* one works exactly in the same way just by switching *East* with *North* and *West* with *South*).
One imaginary segment is drawn between the two points of the *left* group. Depending on how many times does the drone have to pass over a single lines of panels the segment is divided into **n_pass** sub-segments. The WPs identifying the segments splits are then be projected on the opposite side's segment to identify the pixel WP coordinates necessary for the flight. The fact that the UAV may have to pass more than one time over the panel lines depends on the different type of information the user wants to extract from the visit. More specifically, the multiple passages are needed for the identification of the panels'*bar-codes*.

The standard flight height for a normal plant visit is around *15 to 20 meters* depending on how wide the line of panels is. at this height it would be impossible to spot those numbers which are identifiable only at an altitude of around 1.5 to 3 meters.
The issues with such flights plan is also related to how the modules have been arranged and oriented within the panel lines and where the *bar-code* is printed on the module.
Since there is not a pre determined way to install such panel lines, the flight plan has to change accordingly to the plant's panel lines configuration and bar-codes disposition. To accommodate such arrangements, the ***dub*** variable is introduced. This variable lets the user choose among two different types WPs distancing for flight plans with more than one single flight over the panel line. Focusing on one side of the line of panels, depending on the value of the *dub* Boolean two different configuration may take place:

dub variable set to 0: The side line is divided into *n_pass*+1 equal sub segments.

dub variable set to 1: The side line is divided into 2∗*n_pass* sub segments.



(a) Dub = 0 WPs. (b) Dub = 1 WPs.

Figure 5.9: Side by side figures with different Dub values.

These WPs are then divided into two sub-sets of *accepted* and *rejected* WPs. The Fig. 5.9 highlights the criteria of inclusion and exclusion of such WPs. The two accepted sets of *n_pass* WPs on each side of the panel line are going to be the actual flight WPs of the plan.

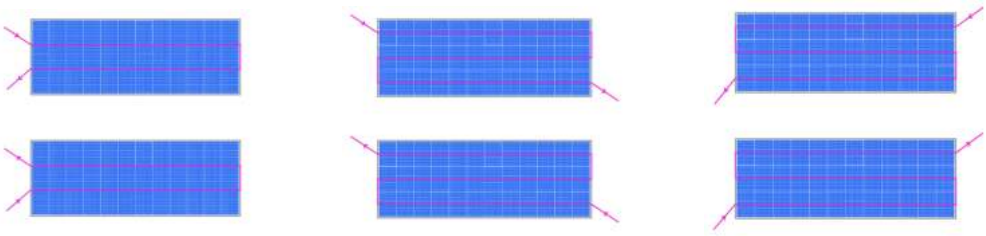
In the case of the dub = 1 the spacing closer to the edges of the row is smaller than the spacing between the inner WPs, whereas for the dub = 0 the spacing is exactly the same. By combining different values of the *dub* and *n_pass* variables the majority of the rows arrangements will always be covered by the UAV flight.

5.4.3.3 Ordering the WPs within the panel lines

The TSP optimization step only considers either the *left* (even *n_pass*) or *both* (odd *n_pass*) extremes of the line of panels. Therefore, in the case of $n\_pass > 1$, the WPs visiting order within the same row has to be decided.
For ordering the WPs the drone cannot go back to the charging station during an image capturing acquisition session of one line. Therefore, it has to approach the visit of one panel in a single *top-to-bottom* or *bottom-to-top* run (Fig.5.10). The choice between these two modalities is based on the following factors:

1. The position of the previous row visited,
2. the *evenness* or *oddness* of the *n_pass* value.

If *n_pass* is *Even* the only side considered during the WPs connection is the left side. Therefore, the decision on how to approach the panel is based exclusively on the position of the previous WP visited. If the y coordinate of the previous WP is higher than the *y-coord* of the center of the row, the visit will be of the *top-to-bottom* kind. The opposite thing holds in the case where the previous *y-coord* is lower than current one.



(a) Even paths. (b) Odd paths. (c) Odd paths.

Figure 5.10: Side by side figures with different path directions.

If *n_pass* is *Odd* there are more option for entering and exiting the single row. Calling the previous visited point (x_p, y_p) and the row's center one (x_c, y_c) the four different approaches are:

- Top-Left/Bottom-Right: When $y_p > y_c$ and $x_p < x_c$.
- Bottom-Right/Top-Left: When $y_p < y_c$ and $x_p > x_c$.
- Top-Right/Bottom-Left: When $y_p > y_c$ and $x_p > x_c$.
- Bottom-Left/Top-Right: When $y_p < y_c$ and $x_p < x_c$.

These predetermined orders make the optimization for the shortest path way more efficient since the WPs considered will be *n_pass* times less than the total. The WPs are then shifted $1m$ outside the lateral bound of the row's BB. This additional step provides an higher confidence of covering the entire line of panels since sometimes the initial boundary is not very tight on the bound. This kind of issue often happens with rows ends missing some modules as in Fig. 5.11.

Figure 5.11: Incomplete panel line ending.

The identification of the $1m$ shift the *Haversine formula*, which takes into account the curvature of the Earth, to calculate the distance in meters between the geo-localized WPs of the image. This distance is calculated between two well known couples of points *TL-BL* and *TL-TR*.
Since the pixel coordinates of these three points is known by the definition of the *.kml* file, and given these two distances in pixels, the average size of *pixel side* in meters is calculated. Finally, the value in pixel units of $2m$ is added to the width

value of every panel in order to make them a meter wider on each side.
the *pixel side* length in meters will also be exploited to measure the distance traveled by to connect for the computation of the optimal path. The *Haversine formula* is commonly used in navigation and is expressed in terms of spherical trigonometry. Here is the formulation of the equation:

$$a = \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1) \cdot \cos(\phi_2) \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right) \tag{5.1}$$

$$c = 2 \cdot \text{atan2}\left(\sqrt{a}, \sqrt{1-a}\right) \tag{5.2}$$

$$\text{distance} = R \cdot c \tag{5.3}$$

Where: $\Delta\phi$ is the difference in latitude between the two points, $\Delta\lambda$ is the difference in longitude between the two points, ϕ_1 and ϕ_2 are the latitudes of the two points, R is the radius of the Earth (in this case, it is multiplied by 1000 to get the distance in meters), atan2 is the two-argument arc-tangent function that calculates the angle from the *x-axis* to a point given its coordinates.

5.4.4 Post-processing step: geo-localization

Given that the image extraction of the satellite images is based on the *top-left* and *bottom-right* coordinates of the picture, the geo-localization of every other pixel in the map should be any easy task. However, some finer details have to be considered. The formulation to find the geographical coordinates of a given pixel is:

$$\begin{aligned} x &= tl[1] + \frac{wp[0] - shift\_x}{img.size[0] - 256}(br[1] - tl[1]) \\ y &= tl[0] + \frac{wp[1] - shift\_y}{img.size[1] - 256}(br[0] - tl[0]) \end{aligned} \tag{5.4}$$

Where: $tl[i]$ and $br[i]$ are the known *Lon-Lat* coordinates of the TL and BR points, $wp[i]$ is the pixel coordinate of the point of interest, $img.size[i]$ is the x, y pixel size of the image, $shift\_x$ and $shift\_y$ are x, y coordinates shift factor.

The Image extractor application provided by *JP Droni* has the limitation of have access only to satellite images up to may 2021. Eventually, whenever updated pictures of the territory are added to the *Google Earth* servers, some minor coordinate shifts happen from previous years image localization. This is due to multiple factors such as satellite misalignment. The adding of a $shift$ factor was necessary to fix such issue. The -256 added to the calculation accounts for a pixel image padding carried out by the image extractor to include bigger margins of the image. This extra room added to the image frame accounts for

possible maps changes. The $shift\_x$ and $shift\_y$ are needed to fix such issue. Depending on the year of acquisition of the image, the value of such shift can be changed to fit the current map in the best way possible (Sec.5.5.2).
The inverse of this function is applied to extract the geo-coordinates of *starting point* information extracted by the *.kml* to turn them into pixel coordinates. The formulation is:

$$\begin{aligned} pix[0] &= shift_x + \frac{(st[1] - tl[1])(img.size[0] - 256)}{br[1] - tl[1]} \\ pix[1] &= shift_y + \frac{(st[0] - tl[0])(img.size[1] - 256)}{br[0] - tl[0]} \end{aligned} \tag{5.5}$$

Where the $pix[i]$ indicates the pixel coordinates of the *starting point*.
Once all the pixel coordinates are transformed to geographic coordinates, a dedicated function transfers these information to a *.kml* file to view the path on *Google Earth*. This file will then enter the application that *JP Droni* uses to pass the geographical path to the UAV and compute the WPs connection with the custom *JPVision* app.

5.5 Path optimization process

Once all the WPs are collected, the following step is to optimize the path that connects them. The **objective** of the optimization is to minimize the flight time needed to complete the visit. The drone will have to **strictly pass over the row of panels** and it will have to keep track of the **battery draining up** as it is one of the most common limitations of electric vehicles. To approach the optimization of the connection, the *Traveling Salesman Problem* emerged as a viable solution. As aforementioned in Sec. 2.9.1, the objective of the TSP is to find the shortest possible route to connect multiple WPs and return to the starting point. For the problem formulation analyzed for this thesis work, the aforementioned additional constraints had to be included in the problem formulation. The definition of the modified problem will be described in the following sections.

5.5.1 TSP formulation

First of all, the implementation for the modified version of the problem started with a standard implementation of the *TSP* through the *IBM CPLEX* software. *CPLEX Optimizer* is a software by *IBM* capable of providing, high-performance mathematical programming solvers for linear, mixed-integer, quadratic, and

quadratic constraint programming problems [96]. The definition of the problem through the *CPLEX API*, will find the WPs connection through the optimal *branch and cut* method (Sec.2.9.1).

5.5.1.1 Problem Formulation

The *problem instance* is defined through the following variables:

C: as the set of all *WPs* to be visited.

c_i: as the *i-th node* representing the *WP*.

x_{ij}: the *binary decision variable*, which represents moving from c_i towards c_j. As a binary variable its values are $x_{ij} = \{1\} \vee \{0\}$.

A: as the arcs formed between c_i and c_j.

d_{ij}: as the distance between c_i and c_j.

To solve the problem the following three steps will be taken:

Step 1 : after collecting all the necessary WPs, the first step is to calculate the *Distance Matrix* (**DM**). The size of the matrix will be $n \times n$ (with n the number of WPs) with each i, j matrix spot representing the euclidean distance between the c_i and c_j WPs. Since this problem is defined as a symmetric TSP instance, the distance between the WP c_i and c_j (d_{ij}) will be the exact same as in the opposite direction d_{ji}.

$$\mathbf{DM} = \begin{bmatrix} 0 & d_{12} & d_{13} & \dots & d_{n1} \\ d_{12} & 0 & d_{23} & \dots & d_{n2} \\ d_{13} & d_{23} & 0 & \dots & d_{n3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} & d_{n3} & \dots & 0 \end{bmatrix} \tag{5.6}$$

Step 2 : The *CPLEX* modeling of the traveling salesman problem instance is based on two elements.

The **Objective Function**: Which represents the quantity that has to be minimize or maximize in the problem instance. In the case of the TSP the total distance Z has to be minimized.

$$\text{Min } Z = \sum_{\forall A} x_{ij} d_{ij} \tag{5.7}$$

The **Constraints**: Which ensure that the problem is modeled accurately and that the solution keeps into account the problem's requirements. The **tour**

constraint ensures that each way-point is visited exactly once in the single tour.

$$\sum_{j=1,j\neq i}^{n} x_{ij} = 1 \quad for:\ i = 1, \ldots, n \tag{5.8}$$

This equation means that for each WP c_i, the sum of the decision variables x_{ij} (representing outgoing edges from c_i) is equal to 1. It enforces that exactly one edge must leave each way-point.
The **Sub-tour Elimination Constraints** or *Big-M* was first used in the *Miller-Tucker-Zemlin* (MTZ) formulation of the TSP [97].

$$u_i \geq u_j + 1 - M(1 - x_{ij}) \tag{5.9}$$

It ensures that the differences between the values u_i and u_j (non-negative integer values representing the order of WPs i and j in the tour) are properly adjusted to eliminate the possibility of creating sub-tours. Choosing the appropriate value for M is important. It should be large enough that the constraint is always active when needed but not so large that it causes numerical issues or overly tight bounds in the optimization problem. For $M = n$, any tour could satisfy this constraint.

Step 3 : The *CPLEX* solver then uses its optimization algorithms to find the values of the decision variables that minimize the objective function while satisfying the constraints, thus providing an optimal or near-optimal solution to the TSP. To perform such solve, the optimization model previously defined is read and initialized by *CPLEX*, encompassing the objective function, constraints, decision variables, and their values. Based on the problem's nature, size, and specified solution approach (*mixed-integer linear programming*), *CPLEX* automatically chooses the most suitable algorithm. Then, the problem instance is pre-processed to reduce its complexity, minimize the size, and identify redundancies among the problem definition enhancing solving efficiency.
Throughout this process, *CPLEX* utilizes the branch-and-bound algorithm, dividing the problem into subproblems through branching and bounding the optimal solution within predefined ranges. This involves exploring different branches of a search tree to identify the best integer solutions.
whenever approaching the end of a solution for expiration time or because the optimal solution is found, *CPLEX* refines the solution with cutting planes supplementary constraints and heuristics as approximation methods. Once reached the end of the solve, the program provides information about the solve, including if the *final permutation is optimal or not.*

5.5.1.2 Problem constraints

However, the problem faced in this project is not a simple WP connection which a standard TSP solver could perform (Sec.5.5). Two extra constraint have to be taken into account:

Edge forcing: WPs located at the two sides of one single panel have to be *consequently visited* in order to ensure the passage over the rows. This constraint is not considered when performing the flight plan for even passages over the panels. In that case only the left side of the panel will be considered. this assumption is coherent with the nature of the problem since the entrance and exit side of the panel will always be the same.

Battery level: Meaning that the plan for the flight will have to take into account the UAV's need to be *charged* n times during the visit.

The following subsection will explain how these constraints were taken into account within the project and how they are embedded in the final solution.

Flight over the row of panels constraint : So far, the implementation of the TSP discussed in Sec. 5.5.1 matches a standard formulation of the Traveling Salesman Problem where the variable to minimize is the total distance flown by the UAV. At this stage the solution would have all the WPs connected without taking into consideration the passage over the panel (Fig.5.12.*a*). Therefore, a modification to the *distance matrix* was introduced alongside the introduction of en extra problem constraint. This new features penalize the UAV if it doesn't pass through these constrained WPs. The idea is to make the cost of not passing through the panels higher than the cost of the longest possible path in the original *distance matrix*. This is achieved by adding the maximum distance from the original matrix to the distances between all pairs of nodes that do not include a constraint. additionally, the problem constraint:

$$\sum_{k\neq i,k\neq j} x_{ik} - x_{ij} \leq \sum_{k\neq i,k\neq j} x_{jk} \quad \text{for } i,j \text{ constrained WPs} \tag{5.10}$$

ensures that if an edge from i to j is included in the tour, then for any other WP k, the path to k should go through j if an edge from i to j is chosen. This *indicator constraints* approach enforces a specific order among the constrained WPs in the tour, ensuring that they are visited in the desired sequence. This modification effectively makes the total cost of any path that does not pass through the constrained WPs higher than any path that does.

Electric vehicle constraint: Generally UAVs utilized for inspection porpoises are powered by batteries. Therefore, it is necessary to incorporate a constraint

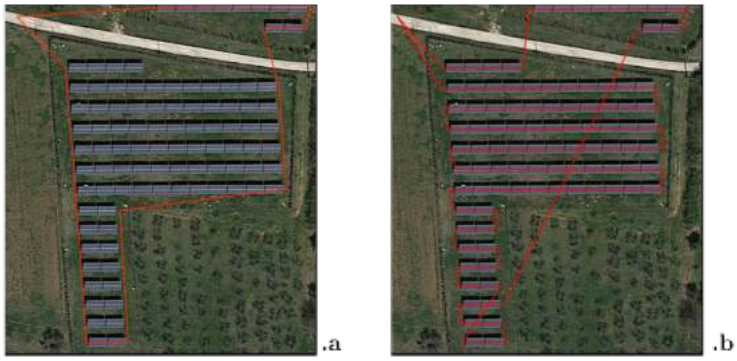


Figure 5.12: *a.*: TSP solution before the application of the constraint; *.b*: After the application of the constraint.

related to drone's autonomy into the task of determining the flight route. To this day, the inspections carried out by *JP Droni* do not take this constraint into account.
The solution proposed for this project is developed based on the concept of WPs clustering where the total amount of points is divides into sub-groups which are feasible to be visited within a single UAV charge. The algorithm at the base of the WPs clusterization process is **k-means**.

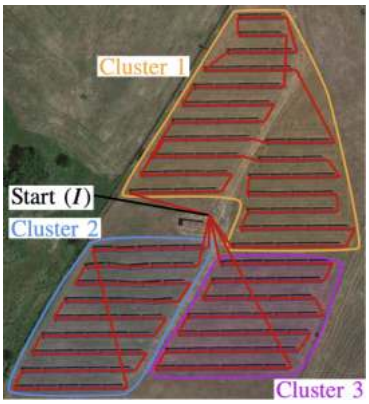


Figure 5.13: *K-menas* clusterized PV plant.

The *k-means* clustering algorithm represents a basic technique in *unsupervised learning* for addressing clustering tasks. It adheres to a straightforward procedure of categorizing a given dataset into k *clusters*, defined by the parameter k established in advance. This algorithm is categorized as *centroid-based*, wherein each cluster is linked to a centroid. Its principal objective is to minimize the total distances between data points and their corresponding clusters. The procedure

accepts an unlabeled dataset as input, divides the dataset into k clusters, and iterates until optimal clusters are identified.
In the process of finding the best UAV route, the *K-means* algorithm was iterated according to the assumption that in the case where the best WPs permutation found considering the full set of WP was longer than the maximum UAV flight endurance, sooner or later a charging pit-stop has to be performed. Therefore, the *K-means* algorithm would be applied with $K = 2$ only if it is not possible to cover the entire plant in a single go. The WPs connection will then be performed on both the two new sub-clusters. At this stage the following outcomes could take place:

1. **Both sub-paths distances are independently compliant with the full UAV charge**: In this case both the sub-tours will be added to the final WPs-permutation and the search would terminate.

2. **Only one sub-path is independently compliant with the full UAV charge**: Only the feasible cluster would be assigned to the final permutation while the other will be divided into $K = 2$ sub-clusters with a *K-means* clusterization.

3. **Both sub-paths distances are not independently compliant with the full UAV charge**: This case implies that the algorithm will be run again on the full set of WPs but with $K = 3$.

For the last two cases, the new sub-tours will then be checked again for feasibility to keep this iteration going. Once all the sub-clusters are checked as feasible, the permutation with the shortest flight-tours and the lowest charging pit-stops will be obtained.
Three assumption were considered for calculating the sub-clusters with the *K-means* algorithm:

1. The **charging point is unique** and it coincides with the *starting position*.

2. The drone's **battery** is represented as the *overall operational flight distance*, measured in meters. This value is determined by various factors, including weather, wind conditions, current battery level, drone type, and more. Users have the flexibility to input a specific endurance value before the flight, considering these factors, to plan the most suitable route that aligns with the available battery capacity. As mentioned in Sec. 5.4.3, the length in meters is calculated through the number of pixels covered by the *path-line* connecting the WPs of a sub-tour.

3. The total distance covered by the *sub-tour path* takes into consideration every connection among the WPs including the starting point.

4. The points considered for the clusterization vary depending on the number of passages over the panels. If the *n-pass* is *odd*, the point considered for the clusterization is the center point of the panel. In the case where the *n-pass* is *even* the left side point will be considered as the point to cluster since it matches with the point considered during the TSP WPs connection.

The *K-means* algorithm was implemented in the project through the library *Scikit-Learn* [33].

5.5.1.3 Additional solution methods

The solution achieved with the *CPLEX TSP* model is not the only one usable in the code to solve the WPs connection. The user can choose among two other solution algorithms: the **Iterated local search** and the Simulated Annealing. Both of these algorithms take the modified *Distance Matrix* for odd *n_pass* and the normal one for the even case. Since these are *heuristic algorithms*, the problem's solution cannot be guaranteed to be optimal. However, they are designed to find good solutions in a reasonable amount of time. Thus, while these heuristic methods can offer effective solutions, they may not always produce the absolute best solution for the problem at hand (Sec.2.9.1). These functions are provided by the library *python-tsp* [98].

5.5.2 Input data

Before using this module, the user must prepare two files first:

input.KML: which has to be an input file including the *polygon* shape encapsulating the entire solar plant and the starting location data marked as a *Key-point* all inside the same folder.

config.JSON: The *JSON* file is a *flight settings file* which has to be modified before every planning session in order to tweak every parameter of the system (tab.5.5.3).

5.5.3 Output data

Once the process is finished and the final permutation is found, three *Outputs* will be produced:

<Name of the project>_output.kml file: The *.kml* file given as output will include all the information related to the path the drone will take to cover the whole plant. The array of WP coordinates is stored into a *poly-line* instance which identifies the position of all the WP's as their vertices. The

information on the WPs also include the *altitude* asked by the user. The information related to the altitude can be stored in a *.tiff* file within the project. If this information is required the user has to insert this file in the folders of the project. This information has to geographically match with the coordinates corresponding to the WPs in order to include such data in the final file.

<Name of the project>_output.csv file: This file identifies all the information related to the flight in a *.csv* table. Other then the coordinates of the WPs and their order of visit, it includes also the following extra data related to the WP itself:

- The panel number to which that WPs correspond to,
- The position number of the way-point with respect to the order of visit of the WPs covering the single panel,
- The altitude data if required by the user,
- The charging pit-stops if any.

ROS message: with all the coordinates to the following nodes of the project.

Table 5.1: JSON comands

paths	**UAV type**	**Flight**	**Detection**	**Plot**	**altitude**	**TSP**
Dir name	Battery[a]	n_pass[b]	th_det[e]	shift_x[g]	altitude [h]	tsp_type [i]
Input name		th[c]	th_overlap[f]	shift_y[g]		th_time [l]
tif name		dub[d]				opened [m]
csv name						n_loops[n]

Table note

[a] **Battery level** of the UAV expressed as a quantity in meters.

[b] **Number of passages** over the panels.

[c] **Lateral distance** in meters from the panels.

[d] **Type of distancing** between the passage over the panel.

[e] Percentage of **confidence** of the panels detection.

[f] Percentage of **overlapping** among the panels detected.

[g] **X-axis Y-axis** coordinates image **shifting** for adjusting to new satellite acquisitions.

[h] 1 if the *altitude* is included in the collected data. 0 otherwise. An extra *.tiff* with the altitude information of the portion of land analyzed has to be included

[i] *Type of TSP solution*: *branch_and_cut_solution*, *iterated_local_search*, *simulated_annealing*.

[l] **Time** in seconds implied for the search of the optimal path for the *branch and cut solution*.

[m] 1: if the trip doesn't take into account the fact that the drone has to go back to a certain location for charging and finishing the loop.
0: if it cares about going back to the starting position at the end of every sub-trip.
Option valid only for Heuristic searches (*Iterated local search* or *simulated annealing*)

[n] How many **iterations** the *iterated local search* has to perform?

Chapter 6

UAV flight control

As products meant for commercial use, neither of the models of the DJI fleet has the possibility to let the programmer have access to low-level hardware controls such as the rotational velocity of the quadrotor's motors.
The commands received by the UAV are only higher level velocity inputs related to the *linear velocity* along all three *axis* and the *angular velocity* along the *z-axis*. On top of this assumption, three control techniques were developed for the *Visual based* UAV flight: A **Simple PID Controller**, a **Lyapunov non linear Controller**, and a **Visual servoing and task stacking based controller**. All of them are based on a visual features extraction and sensors data coming from the UAV through the *DJI-ROS Bridge* application (Chap.4).
In this chapter, all of these control techniques, including the *trajectory based control* employed for the indoor coordinates based movements are explained.

6.1 Features extraction

To create the control input on a *visual servoing* bases, the first step is to find and extract reliable features from the image stream. The extracted features will have to identify a reliable representation of the positioning of the panels row to compute the correct control input capable of make the UAV positioning it self correctly with respect to the followed line. The required row image characteristics are the following:

1. The depicted panel will have to be as much as possible *horizontally-aligned* and *centered* within the image frame.

2. The images will have to be taken approximately at the same height in order to keep similar proportions.

3. The images must not have motion blurred effects.

4. For optimal performance of Image Mapping in PV plants, the image overlaps must meet specific criteria: **sidelap** (lateral image overlap) should range from 15% to 45%, and **endlap** (end of the picture) should range from 55% to 65%. Typically, a minimum total *endlap* of 40% is essential to achieve a satisfactory string overview (Sec.2.16).

The control input will be computed according to these requirements.

6.1.1 Detectron2 Panels segmentation

For identifying the position of the panel in the image, a *Detectron2 instance segmentation NN* was used. The following subsection will explain how the images collection and labeling campaign took place for the custom data-set, why the *Mask R50-FPN R-CNN* model was used over the *Mask R101-FPN R-CNN*, and how the feature were extracted from the segmentation data.

6.1.1.1 Data-set images and annotation

The custom data-set is composed of 1368 labeled pictures coming from different sources:

- *Robflow* free data-sets of areal images of solar plants [92],
- Imaging from collection campaigns conducted by *JP Droni*:
  - The first one captured by their *DJI mavic 3* with a vertical image setting on a country side PV plant. The panels' vertical alignment is a result of their previous method of capturing images, where the drone used to fly linearly over the panels instead of the current lateral approach.
  - The second one captured with their *Matrice 300 RTK* on a roof top PV plant. In this case, the panels are horizontal with respect to the camera frame.
- Spear online images of solar panels and panels rows in order to make the dataset more complete and versatile.
- Images coming from the *Coppelia simulation*. These images enhanced the ability of the drone to identify panels during the simulation tests.

All of these images weren't labeled as segmentation data-sets. Therefore, all the annotation was again conducted manually. Thankfully, just like with the data-set for the satellite images segmentation, the *MetaAI SAM* model helped accelerating

the annotation process by a lot [95].
To enhance the segmentation capabilities of the models the following data-set *augmentation* steps were applied: *90° Rotate: Clockwise*, *90° Rotate: Counter-Clockwise*, *Image Flip*, and ±30% *brightness* variation. The final data-set was three times bigger than the original. The images are then divide in the three sets with a division of 80%, 20%, and 10% for the *Train*, *Validation*, and *Test* set. An image resizing to the square dimension of $640x640$ was also applied.

6.1.1.2 Model fine-tuning

The *R50-FPN Mask R-CNN* and *R101-FPN Mask R-CNN* models were initially tested for the fine tuning. The *R50* and the *R101* are the short names for the two different backbone architectures of the models. The *R* refers to a specific type of CNN called *ResNet* (Residual Network). 50 and 101 represent the number of layers of the NN. For both networks have the the same feature extraction technique called Feature Pyramid Network (FPN). It is a CNN feature capable of enabling better segmentation results across different resolution by creating and combining a pyramid of feature maps. The *Mask R-CNN* is the extension of the the *faster R-CNN* which combines the *object detection* with *semantic segmentation* capabilities to add the ability of the network to to predict *instance segmentation* masks.

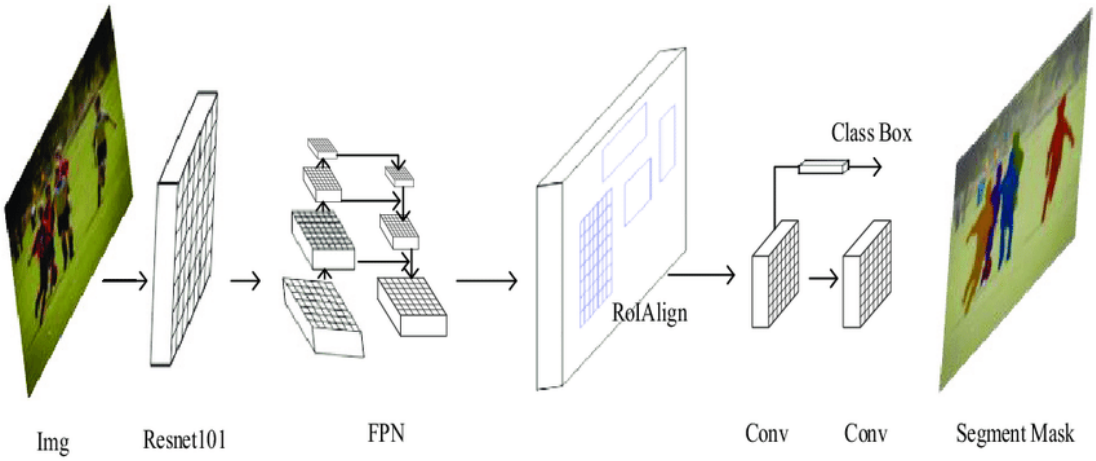


Figure 6.1: FPN Mask R-CNN network framework [15].

The main difference between the two models is the backbone. Because the *ResNet-101* has more layers and capacity, it is potentially more capable of distinguishing fine features in images, which could lead to better segmentation performances especially with complex scenes [15].
However, the trade-off of using such network over the *ResNet-50* is an higher

computational effort and and a longer training time since deeper network require more memory for computation and training. Moreover, for the same hardware capabilities, these issues lead also to a longer inference time. Therefore, given the resource intensive behavior of the *R101* was rejected for the *R50* even though it performed better in terms of *score-evaluation* (More on the evaluation in Chap 7).

6.1.1.3 Feature extraction

The feature extracted from the segmentation is a ***line*** passing through the middle of the row of panels of the image. In order to make the drone's camera capture the row, the features representing the panels had to be detailed enough to make the UAV fly aligned with the row (Fig. 6.2.*a*).
The extraction of this feature comes from an *OpenCV* post-processing step. In the same fashion as the post-processing step explained in Sec. 5.4.1, from the panels' segmentation masks provided by the NN (Fig. 6.2.*b*), the *FindContour* (Fig. 6.2.*c*) and *MinAreaRect CV2* function2 (Fig. 6.2.*d*) will approximate the contour of the mask into a rectangle. From there, they will extract the reliable information related to the row of interest which are:

- (x, y) coordinates of row's center with respect to the center of the image.
- θ angle of the row with respect to the horizontal line of the image.

For each of the following sections related to the visual servoing techniques, the different interpretations to the line representation in relation to the control techniques will be explained.

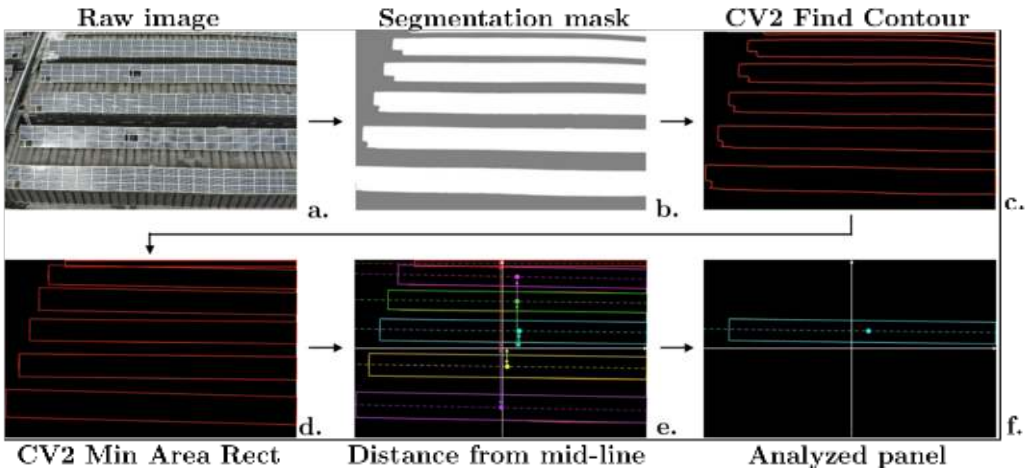


Figure 6.2: Filtering step of the center panel.

Each visual-based control technique deals with the crucial issue of the simultaneous observation of multiple panels. Considering that the flight tour generally takes place at an altitude between 15 to 20 meters, it's possible that the image frame will include multiple rows of panels arranged in parallel (Fig. 6.2.*a*). Since it is not possible to occlude the view of the surrounding panels, the problem had to be fixed during through a processing step.
The one line that has to be followed is the one closer to the center line of the image frame. Starting from this assumption, the masks excluded from the image frames are the ones that are the farther away from the center line. To measure this distance, the absolute value $|y|$ of the y coordinate of the center of each panel was compared (Fig. 6.2.*e*). The smallest one of all will determine the considered mask-coordinate (Fig. 6.2.*f*).

6.2 PID controller

6.2.1 Feature interpretation

In the case of the *PID controller*, the data extracted from the run time image of the panels mentioned in the previous section is not modified. This means that the $[y_s, \theta]$ coordinates are use directly for the control. The following picture depicts this geometrical interpretation.

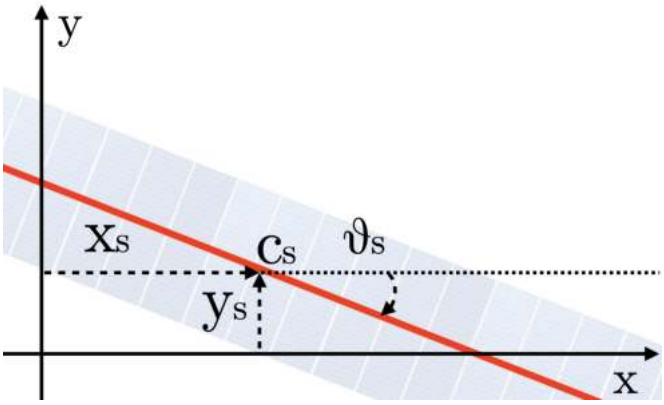


Figure 6.3: PID's feature geometric interpretation.

in Fig. 6.3, C_s represents the center of the line feature with x_s and y_s its pixel coordinates in the image frame and θ_s the angle the panel makes with respect to the *x-axis*. In order to make the control input based on this coordinates independent from the image resolution, the (x_s, y_s) couple are normalized respectively to the pixel width and pixel height of the image.

6.2.2 Controller definition

For computing the control input, the PID controller uses only the y_s and θ_s values from the image evaluation and the z value directly measured from the DJI sensors. Independently from the v_y of the drone, the PID controller will take care of each one of the three measurements independently. Therefore, the y_s, z, and θ_s, coordinates misalignment will be controlled exclusively on the drone's v_x, v_z, and ω_z velocities with PID terms (Eq.6.1).

$$\begin{cases} v_x = K_{p,x} y_s + K_{i,x} \int_0^t y_s d\tau + K_{d,x} \frac{dy_s}{dt} \\ v_z = K_{p,z} z + K_{i,z} \int_0^t z d\tau + K_{d,z} \frac{dz}{dt} \\ \omega = K_{p,\theta} \theta_s + K_{i,\theta} \int_0^t \theta_s d\tau + K_{d,\theta} \frac{d\theta_s}{dt} \end{cases} \tag{6.1}$$

This control strategy ensures the alignment of the drone's camera with the rows but it does not take into consideration the continuous lateral movement which is fundamental for capturing the entire surface of the row of panels. Therefore, setting the v_x velocity value as dependent to the y_s and θ_s miss-alignment could be a good start for ensuring the system stability.

The v_y velocity is then set as inversely proportional to the combination of the y_s and ω_s coordinates. The UAV velocity will be equal to the maximum chosen speed only during perfect alignment. This caution improves the ability of the drone to align with the panels but at the same time it does not ensure a constant velocity profile along the visited row. This issue could cause problems when trying to perform a *timed capture of the panel images*.

6.3 Non linear controller

6.3.1 Feature interpretation

The distance between the center and the line considered for this control action is not the difference between the center of the box encapsulating the panel and the center of the image, but rather the shortest distance between the center of the image and the panel line as shown in Fig. 6.4.

Where the l vector is defined as $l = P_{\min} - O$ with $P_{\min}$ being the closest point of the line to the center of the image frame and the center of the image frame with coordinates $(0, 0)$.
In order to uniquely define the point $P_{\min}$ the following steps have be taken:

Defining the panel *line* equation: Starting from the given angle θ_s and the center point C_s of the line, the first step is to find the angular coefficient m

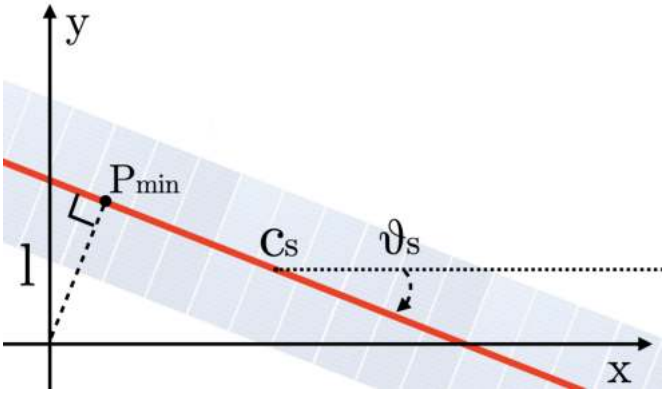


Figure 6.4: Non linear controller feature geometric interpretation.

of the line as:

$$m = tan(\theta) \tag{6.2}$$

Given the center point of the rectangle encapsulating the panel (x_s, y_s) which is by definition laying on the line, the equation turns out to be:

$$y = m(x - x_s) + y_s \tag{6.3}$$

Find intersection: The P_{min} point is then found as the intersection of the panel line with its perpendicular line passing through the origin, namely:

$$\begin{cases} y = m(x - x_s) + y_s \\ y = -\frac{x}{m} \end{cases} \tag{6.4}$$

solving for (x, y) we find the intersection point as:

$$\begin{aligned} x_{\text{min}} &= \frac{m(mx_s - y_s)}{1 + m^2} \\ y_{\text{min}} &= \frac{mx_s - y_s}{1 + m^2} \end{aligned} \tag{6.5}$$

Vector length: Once the coordinates of point P_{min} are found, it is trivial to calculate the length of vector l as the distance between P_{min} and the origin O:

$$|l| = \sqrt{(x_{\text{min}} - 0)^2 + y_{\text{min}} - 0)^2} \tag{6.6}$$

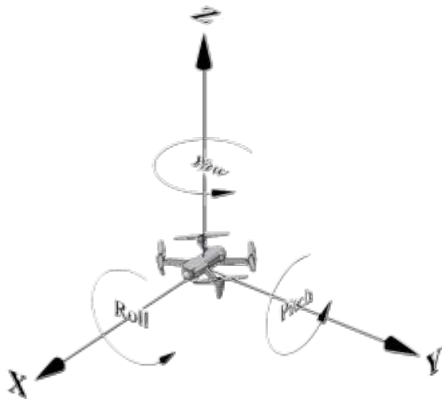


Figure 6.5: Frame of the DJI UAV.

6.3.2 Controller definition

The algorithm used in this section exploits the concept of the *Lyapunov stability theorem* [99] and the reinterpretation of the work proposed in [100] and re-proposed in a previous thesis work.

6.3.2.1 Reference system and Kinematic equations

As explained in Sec. 2.11.2, the *kinematic equations* of any quadrotor model derive from the differentiation of the geometric equations. The *kinematic equations* of the DJI model are:

$$\begin{cases} \dot{x} = v_x \cos(\theta) - v_y \sin(\theta) \\ \dot{y} = v_x \sin(\theta) + v_y \cos(\theta) \\ \dot{z} = v_z \\ \dot{\theta} = \omega \end{cases} \tag{6.7}$$

6.3.2.2 Proof of stability using Lyapunov criterion

The main concept of the *Lyapunov stability criterion* [99] states that:
Given a non-linear system of the kind:

$$\dot{\mathbf{x}} = \underline{\mathbf{f}}(\mathbf{x}, \mathbf{u}) \tag{6.8}$$

The point $\mathbf{x} = \underline{0}$ is asymptotically stable if there exists a *Lyapunov function* $V(\mathbf{x})$ such that:

- $V(\mathbf{x}) > 0$: *positive definite.*
- $\dot{V}(\mathbf{x}) < 0$: *negative definite.*

This criterion is the basis of the proof of stability of the system taken into consideration.
The state variables of the system are x, y, θ as the information coming from the image stream and z as the altitude coming directly from the altitude sensors. The idea is to stabilize the x and θ coordinates around 0 and the z coordinate around a constant altitude value while keeping the the *y-velocity* $\dot{y}$ constant in time. To simplify the calculations the z state variable will be considered as $\tilde{z} = z - z_0$ with z_0 the altitude threshold to maintain during flight.
Staring from the *positive definite* $V(\mathbf{x})$ Lyapunov function:

$$V(\mathbf{x}) = \frac{\theta^2}{2} + \frac{\tilde{z}^2}{2} + \frac{x^2}{2} \tag{6.9}$$

We can compute the derivative as:

$$\begin{aligned} \dot{V}(\mathbf{x}, \mathbf{u}) &= \dot{\theta}\theta + \dot{\tilde{z}}\tilde{z} + \dot{x}x \\ &= \theta\omega + \tilde{z}v_z + x[v_x \cos(\theta) - v_y \sin(\theta)] \end{aligned} \tag{6.10}$$

The proposed input is then going to be defined to ensure the $\dot{V}(\mathbf{x}, \mathbf{u})$ to be *negative definite*. Therefore, the following composite inputs are defined:

$$\begin{cases} V_x = \dot{x} = v_x \cos(\theta) - v_y \sin(\theta) \\ V_y = \dot{y} = v_x \sin(\theta) + v_y \cos(\theta) \end{cases} \tag{6.11}$$

Thanks to these composite inputs, the system can be now be written as:

$$\begin{cases} \dot{x} = V_x \\ \dot{y} = V_y \\ \dot{z} = v_z \\ \dot{\theta} = \omega \end{cases} \tag{6.12}$$

By imposing asymptotically stable inputs we then obtain:

$$\begin{cases} \dot{x} = V_x = -K_x x \\ \dot{y} = V_y = V_0 \\ \dot{z} = v_z = -K_z \tilde{z} = -K_z(z - z_0) \\ \dot{\theta} = \omega = -K_\theta \theta \end{cases} \tag{6.13}$$

By substituting the values of the the V_x and V_y composite inputs and solving for $\mathbf{u}$, the control input will ensure the $\dot{V}(\mathbf{x}, \mathbf{u})$ to be *negative definite*, namely:

$$\mathbf{u} = \begin{cases} v_x = -xK_x \cos(\theta) + v_0 \sin(\theta) \\ v_y = v_0 \cos(\theta) + xK_x \sin(\theta) \\ v_z = -K_z \tilde{z} \\ \omega = -K_\theta \theta \end{cases} \tag{6.14}$$

Combining eq. 6.14 with eq. 6.10:

$$\begin{aligned}\dot{V}(\mathbf{x},\mathbf{u}) = &- K_\theta\theta^2 - K_z\tilde{z}^2 + \\ &+ x[(-xK_x\cos(\theta) + v_0\sin(\theta))\cos(\theta) - \\ &- (v_0\cos(\theta) + xK_x\sin(\theta))\sin(\theta)] = \\ = &- K_\theta\theta^2 - K_z\tilde{z}^2 - K_x x^2 < 0\end{aligned} \tag{6.15}$$

Therefore, ensuring the stability of the system.

6.4 Task Stacking

6.4.1 Pinhole model and camera calibration

The pinhole camera model simplifies the relationship between *real-world 3D scenes* and *2D images* using triangle similarity principles. It's based on the concept that light from objects passes through the camera's pinhole and reaches the sensor. This establishes a mathematical connection between object size in the image measured in pixels and their actual physical dimensions measured in meters. This connection is determined by a coefficient, which represents the ratio between the sensor-to-pinhole distance and the pinhole-to-object distance. Despite its simplicity, this correlation can be employed to deduce information about the real world. This is achievable by utilizing known camera parameters and calibrating the recognition system [83].

Thanks to a previous camera calibration conducted with the *camera_calibration* ROS Package for a previous thesis project, the camera parameters of interest are saved inside of a YAML file. Moreover, a visual servo calibrate package, has been developed for setting up the camera parameters and compensate the distortion. The two nodes of the package will first extract and publish the calibration data contained in the YAML file, and the second node will subscribe to the */CameraInfo* topic and provide an undisturbed video stream with a suitable aspect ratio.

6.4.2 Feature interpretation

Traditional image-based control schemes use the image-plane coordinates of a set of points to define the set of visual features s. The image measurements are the pixel coordinates of the set of image points, and the camera intrinsic parameters are used to go from image measurements expressed in pixels to the features.
Exploiting the *Pinhole model* we can establish the connection between the real 3D world feature to the 2D image plane, namely for a 3D point with coordinates

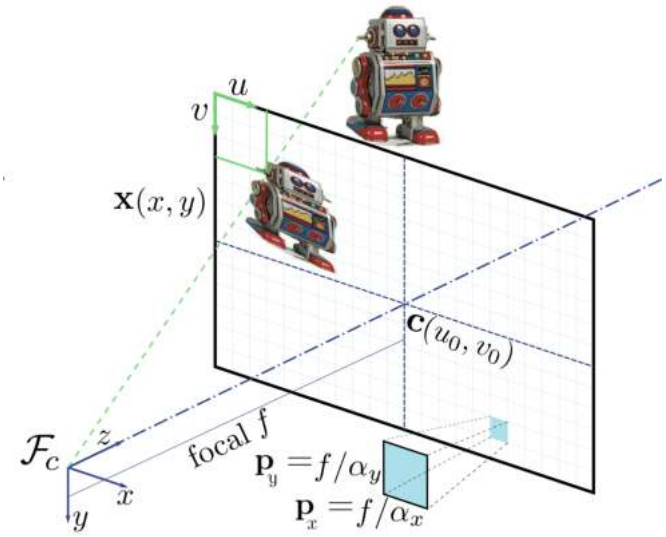


Figure 6.6: Camera parameters, pinhole model.

$\mathbf{X} = (X, Y, Z)$ in the camera frame, which projects in the image as a 2D point with coordinates $\mathbf{x} = (x, y)$ we have:

$$\begin{cases} x = X/Z = (u - u_0)p_x \\ y = Y/Z = (v - v_0)/p_y \end{cases} \tag{6.16}$$

Where (u, v) are the coordinates of the image point expressed in pixel unit, (u_0, v_0, p_x, p_y) are camera parameters describing the center of the image in pixel coordinates (u_0, v_0), and (p_x, p_y) the ration between the focal length f and the size of the pixel (α_x, α_y) (Fig.6.6).
Then, in the image plane the (ρ, θ) features are extracted. These two features will uniquely define the panel line in the image plane. In Fig. 6.7 the polar coordinates ρ, θ represent respectively the module of the l vector found in Sec. 6.3.1 and the angle it makes with the horizontal line.

6.4.3 Controller definition

Visual servoing techniques involve utilizing data from one or multiple cameras to govern the motion of a robotic system. This enables a broad spectrum of tasks aimed at determining what the system should see with respect to what it actually sees. This control can encompass any subset of the system's n degrees of freedom. Regardless of the sensor's arrangement, which may span from a camera attached to the robot's end effector to several cameras observing the scene, the objective is to select a set of k visual data points to track during the control. Therefore, a control law is then developed to guide the visual data $\mathbf{s}(t)$ towards a desired

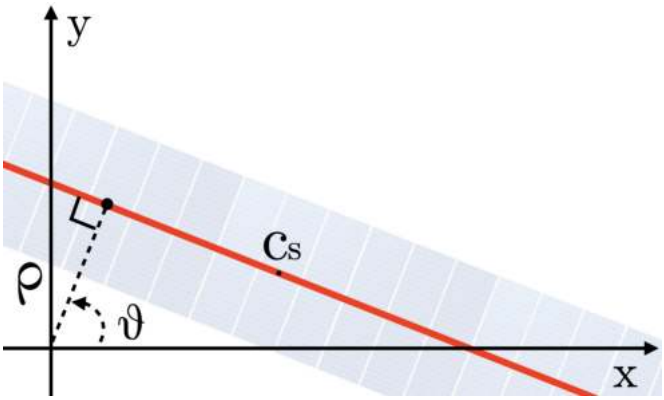


Figure 6.7: (ρ, θ) feature geometric interpretation.

value $\mathbf{s}^*$, which defines the successful completion of a task [12].
With an ideal feature extraction, the camera feedback could be directly fed into the control system considering the sensor feedback as ground truth feature. In a real system where the detection is not guaranteed to be precisely identified, some precaution have to be taken in order to avoid any kind of issues on the given control whenever a false detection is encountered. To manage such issues, a possibility would be to average out every element of the image data over a given detection interval. This approach would partially mitigate the effect of wrong detections. However, the best approach to such a problem would be to use a well suited implementation of the *Kalman-filter* capable of discern genuine features from erroneous detections. One of the key advantages of using it is related to its capability to account for the expected behavior of the feature evolution in time allowing the system to filter out noise or false positives that may be detected. By continuously updating its estimate value based on the incoming camera feedback and the expected feature behavior, the *Kalman-filter* would provide a robust mechanism for maintaining accurate feature values through adapting to changing conditions over time considering the possibility of features evolving in time. Its ability to handle time-varying dynamics ensures that the control system remains responsive and reliable in the face of changing circumstances.

6.4.3.1 Interaction matrix

The *interaction matrix* L is a key concept for visual servoing. It represents how the changing of position and orientation of the camera relates to changes in the visual features, namely:

$$\mathbf{s} = \mathbf{s}(\mathbf{p}(t)) \tag{6.17}$$

Where $\mathbf{p}(t)$ represents the the pose at the instant t between the camera and its environment. The differentiation of s enables the possibility to determine the connection between changes in the visual data and the relative movements between the camera and the scene:

$$\dot{\mathbf{s}} = \frac{\partial \mathbf{s}}{\partial \mathbf{p}} \dot{\mathbf{p}} = \mathbf{L}\mathbf{v}_c \tag{6.18}$$

Where $\mathbf{L}$ is a $k \times 6$ matrix (with k equals to the features DoFs), referred to as the *interaction matrix* associated with $\mathbf{s}$ and $\mathbf{v}_c$ is the is the velocity twist of an ideally free to move camera, with respect to the scene.
The matrix $\mathbf{L}$ can be written based on any visual feature constructed from configurable geometric primitives. This is done by defining the equation representing the primitive's nature and configuration in the scene, its projection onto the image plane, and the relation between the 3D primitive and its image.
In the case of a line feature described with the polar coordinates (ρ, θ) (minimal representation with $k = 2$) the $\mathbf{L}_l^{2\times 6}$ matrix can be written as:

$$\mathbf{L}_l = \begin{bmatrix} \mathbf{L}_\rho = \begin{bmatrix} \lambda_\rho \cos(\theta) & \lambda_\rho \sin(\theta) & -\lambda_\rho \rho & (1+\rho^2)\sin(\theta) & -(1+\rho^2)\cos(\theta) & 0 \end{bmatrix} \\ \mathbf{L}_\theta = \begin{bmatrix} \lambda_\theta \cos(\theta) & \lambda_\theta \sin(\theta) & -\lambda_\theta \rho & -\rho\cos(\theta) & -\rho\sin(\theta) & -1 \end{bmatrix} \end{bmatrix} \tag{6.19}$$

With:

$$\begin{aligned} \lambda_\theta &= A\sin(\theta) + B\cos(\theta) \\ \lambda_\rho &= A\rho\cos(\theta) - B\rho\sin(\theta) - C \end{aligned} \tag{6.20}$$

Where A, B, C and $D(\neq 0)$ represent the parameters used to describe the equation of a plan in the camera frame, namely:

$$AX + BY + CZ + D = 0 \tag{6.21}$$

More on this in[12].

6.4.3.2 UAV camera motion constraint

Since, the camera used for this thesis project is mounted on a drone, the relation between $\mathbf{s}$ and the UAV twist cannot simply be the same as eq.6.18. Therefore, it is necessary to find the relationship between the camera twist and the UAV twist and adapt it to the degrees of freedom of the drone. The aim of this will be to map the camera velocity $\mathbf{v}_c$ to the control input $\mathbf{u} = [v_x, v_y, v_z, \omega_z]$.
The first step will be to find the transformation ${}^c\mathbf{W}_b$ that allows for the change of reference for two twists rigidly connected, namely:

$${}^c\mathbf{W}_b^{(6\times 6)} = \begin{bmatrix} {}^c\mathbf{R}_b & ({}^c\mathbf{t}_b)_x \, {}^c\mathbf{R}_b \\ \mathbf{0} & {}^c\mathbf{R}_b \end{bmatrix} \tag{6.22}$$

With:

$$ {}^{c}\mathbf{R}_b = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad {}^{c}\mathbf{t}_b = \begin{bmatrix} 0.14 \\ 0 \\ -0.04 \end{bmatrix} [m] \tag{6.23} $$

Where ${}^{c}\mathbf{R}_b$ and ${}^{c}\mathbf{t}_b$ are respectively the fixed *rotational matrix* and *translational vector* between the body frame of the drone $< b >$ and the camera frame $< c >$. The next step is to map $\mathbf{v}_b^{(6\times1)}$ to the control input $\mathbf{u}^{(4\times1)}$. Therefore, the $\mathbf{A}^{(6\times4)}$ *Adapter matrix* is introduced.

$$ \mathbf{v}_b = \mathbf{A}\mathbf{u} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_z \end{bmatrix} \tag{6.24} $$

The $\mathbf{J}_s$ matrix will encapsulate all the steps to create the relationship between the variation of the feature line in the image frame and the UAV velocity input u, namely:

$$ \dot{\mathbf{s}}^{(2\times1)} = \mathbf{L}\mathbf{v}_c = \mathbf{L}\ {}^{c}\mathbf{W}_b\mathbf{v}_b = \mathbf{L}^{(k\times6)}\ {}^{c}\mathbf{W}_b^{(6\times6)}\mathbf{A}^{(6\times4)}\mathbf{u}^{(4\times1)} = \mathbf{J}_s^{(k\times4)}\mathbf{u} \tag{6.25} $$

6.4.3.3 Task based control and minimization problem

Generally, the control of mobile robots is based around the inference of the robot's state based on the platform's sensors feedback. This kind of applications imply the use of sensor fusion methods, such as the *Kalman filter* which in this case, would provide an accurate estimation of system's state (the robot's position with respect to the environment).
Unlike this type of position control, *task based control* computes the control action directly within the sensors space (task space). This concept entails that the robot knows what its sensor perceptions should ideally be. Given this assumption, the robot no longer deduces its position from sensor data but, it computes a *proportional control action* aimed at driving its sensor perceptions towards the goal sensor values, namely:

$$ \mathbf{e} = \mathbf{s} - \mathbf{s}^* \tag{6.26} $$

Where $\mathbf{s}^*$ is the desired feature value, $\mathbf{s}$ is the current sensor's perception, and $\mathbf{e}$ the error between the two. Since $\mathbf{s}^*$ is a fixed desired value, the time derivative of $\mathbf{e}$ will be equal to $\dot{\mathbf{s}}$. As aforementioned in Sec. 6.4.3.1 $\dot{\mathbf{s}}$ can also be written as:

$$ \dot{\mathbf{s}} = \dot{\mathbf{e}} = \mathbf{J}_s\mathbf{u} \tag{6.27} $$

To obtain a proportional control law it is possible to impose a desired behavior of the time derivative of the error as:

$$\dot{\mathbf{e}}^* = -\lambda \mathbf{e} \tag{6.28}$$

which implies a controlled behavior of the error as:

$$\mathbf{e}(t) = \mathbf{e}^{(-\lambda t)} \tag{6.29}$$

To impose such behavior the following *control input minimization problem* is computed:

$$\text{argmin}_{\mathbf{u}} \; ||\dot{\mathbf{e}} - \dot{\mathbf{e}}^*||^2 = ||\mathbf{J}_s \mathbf{u} + \lambda \mathbf{e}||^2 \tag{6.30}$$

In the ideal case, the solution to the problem is found as:

$$\mathbf{J}_s \mathbf{u} + \lambda \mathbf{e} = 0 \tag{6.31}$$

obtaining the minimal control input:

$$\mathbf{u} = -\lambda \mathbf{J}_s^+ \tag{6.32}$$

6.4.3.4 Task stacking

To achieve the final objective of following the row of panels, the UAV had to take care of the following three tasks:

1. *Aligning* the camera frame with the the row of panels,

$$\dot{\mathbf{e}}_1^* = -\lambda \begin{bmatrix} \rho - \rho^* \\ \theta - \theta^* \end{bmatrix} = -\lambda \begin{bmatrix} \rho \\ \theta - \theta^* \end{bmatrix} \qquad \mathbf{J}_{1|\dot{\mathbf{e}}_1^*} = \mathbf{J}_s \text{ with } \mathbf{L} = \mathbf{L}_l \tag{6.33}$$

2. *Keeping* the flight at a constant height,

$$\dot{\mathbf{e}}_2^* = -\lambda [z - z^*] \qquad \mathbf{J}_{2|\dot{\mathbf{e}}_2^*} = [0 \quad 0 \quad 1 \quad 0] \tag{6.34}$$

3. *Moving* along the row of panels.

$$\dot{\mathbf{e}}_3^* = [v_y^*] \qquad \mathbf{J}_{3|\dot{\mathbf{e}}_3^*} = [0 \quad 1 \quad 0 \quad 0] \tag{6.35}$$

This means that the objective vector $\mathbf{s}^*$ can be written as:

$$\mathbf{s}^* = \begin{bmatrix} \rho^* \\ \theta^* \\ z^* \\ v_y^* \end{bmatrix} \tag{6.36}$$

since the dimensions of $\mathbf{s}^*(4 \times 1)$ are equal to the number of degrees of freedom the problem is considered as perfectly constrained. This means that all the tasks can be satisfied simultaneously. Therefore, the three tasks can be regrouped in a single one *without* any kind of *compromise* in the UAV's movement, namely:

$$\dot{\mathbf{e}}^* = \begin{bmatrix} -\lambda\rho \\ -\lambda(\theta - \theta^*) \\ -\lambda(z - z^*) \\ v_y^* \end{bmatrix} \qquad \mathbf{J} = \begin{bmatrix} \mathbf{J}_{\rho|1} \\ \mathbf{J}_{\theta|1} \\ \mathbf{J}_2 \\ \mathbf{J}_3 \end{bmatrix} = \begin{bmatrix} J_{\rho|0} & J_{\rho|1} & J_{\rho|2} & J_{\rho|3} \\ J_{\theta|0} & J_{\theta|1} & J_{\theta|2} & J_{\theta|3} \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{6.37}$$

The control input corresponding to the new task can be computed by solving the minimization problem discussed in Sec. 6.4.3.3.

6.4.4 Inclined camera setting

Solar panels are not positioned parallel to the ground. They are rather disposed at an angle towards the sun's path. This configuration, allows them to receive more direct sunlight throughout the day optimizing the amount of solar energy that the panels can capture [101]. A new objective for *JP Droni* would be to capture panel images with a camera angle that corresponds to the tilt of the solar panels on the plants. The following adjustments will accommodate the control techniques to this new image acquisition setting which will capture the row's modules with a 90 deg relative angle.
Given the new camera tilt angle, the drone will have to partially shift the previous trajectory in order to keep the row of panels centered in the image frame as shown in Fig.6.8. To find the value of such *Shift* the following trigonometric computations are computed.

The first part involves finding the c_1 cathetus of Fig.6.8. Since the α angle and the b_1 length correspond respectively to the panels' tilt angle and the flight altitude, the two dimensions are known. Therefore, it becomes easy to deduce the length of c_1 as:

$$c_1 = b_1 \tan \alpha \tag{6.38}$$

The second step is to remove the c_2 cathetus' length to obtain the total length of the backwards shift.
Given the knowledge of the angle α the length of the panel (L) and the lowest point of the panel from the ground (H_0) are known, it is possible to find the length of b_2 as:

$$\begin{aligned} c_3 &= L/2 \sin \alpha \\ b_2 &= c_3 + H_0 \end{aligned} \tag{6.39}$$

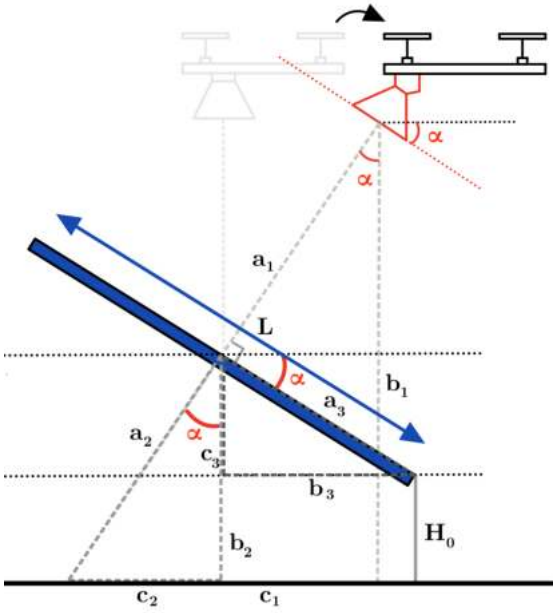


Figure 6.8: Shift, geometric interpretation.

To then find c_2 as:

$$c_2 = b_2 \tan \alpha \tag{6.40}$$

The backwards shifting is then computed as a perpendicular distance in meters from the previous line of panel as Shift $= c_1 - c_2$. This value is then converted into geographical coordinates to adjust the planned trajectory (Chap.5).
Some adjustments were made also on the controller since the rigid complex *UAV+camera* previously considered, has undergone some geometrical changes. Specifically, the camera is not pointing to the ground with a 90 deg angle but it's pointing towards the panel with a specific angle α. Therefore, the only thing that will change will be the rotation matrix ${}^c\mathbf{R}_b$ in the transformation ${}^c\mathbf{W}_b$ that allows for the change of reference for two twists rigidly connected, namely:

$${}^c\mathbf{R}_b = \begin{bmatrix} 0 & -1 & 0 \\ \cos(\alpha) & 0 & \sin(\alpha) \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix} \tag{6.41}$$

6.5 Across the panel indoor movement

As aforementioned in Sec. 4.2.2.1, the set-up for a portion of the tests were made in a indoor environment. Therefore, the GNSS signal cannot be sufficiently reliable to conduct the *Way-Point mission* to move to the following point (as explained in

Sec. 4.2.2.2). This section will briefly describe how such movement was achieved during the indoor tests.
To represent the location based movement of the UAV inside the indoor environment, the *OptiTrack* motion capture system installed is the test area was used. The *OptiTrack* system is a 8 infrared camera setup capable of providing real-time data on the position and the orientation of the sensorized objects located within the calibrated area. Thanks to the *OptiTrack Motive* application [86] installed on the computer managing the position data, it will be possible to stream the information via socket and publish it as a *ROS topic* thanks to the *optitrack* ROS package [87]. The *Across panels control node* will simulate the WP mission through the design of a simple position based flight controller to move the UAV to the desired location.

6.5.1 Trajectory controller

The first thing to do is to find the WPs that map the solar system. Thanks to the *Motive* software (capable of creating rigid bodies instances within the environment), the locations of the handmade rows of panels were identified in the environment 7.3.2. The two ends locations were then raised up to the height of the UAV flight cruise in order to simulate the acquisition of a coordinate.
Once the UAV finishes inspection of one of the rows, the *ROS Action client* will conclude the *Vision Based action* and it will start the *Position Based* one. Thanks to the just recovered positions, the drone will be controlled to go to the following panels-end.
To address this problem a trajectory to move from the current position of the drone to the end of the following line of panels is planned. The trajectory generation was first based on the definition of a *quintic poly-line* capable of connecting the two specified points [102]. Subsequently, the velocity profile was sampled at a rate of $15Hz$ and then sequentially transmitted to the UAV.
However, the drawback of employing this straightforward approach is that the velocity signals are sent to the drone in an open-loop way.
A different approach is based on a simple PID position based controller.

6.5.2 Simple PID position based controller

The PID controller relies on the run time position continuously streamed by the *Optitack* system onto a given socket. The *Optitack* ROS package installed on the computer will read on the UDP socket providing a */StampedPose* message on the */<name_robot>/world* topic. Therefore, once the *UAV_rigidbody* is created, The absolute position of its frame with respect to the world frame is published in the architecture.

The final-goal position is identified as a *Goal frame* with respect to the world frame. Since both the UAV's and goal frames are described with respect to the world frame, it was easy obtain the position of the *goal frame* with respect to the current *drone frame* as:

$$ {}^{d}\mathbf{T}_g = {}^{w}\mathbf{T}_d^{-1} \, {}^{w}\mathbf{T}_g \tag{6.42} $$

Where ${}^{d}\mathbf{T}_g$ is the transformation matrix between goal frame seen from drone's frame, ${}^{w}\mathbf{T}_d$ is the transformation matrix between the drone's frame seen from world frame, and ${}^{w}\mathbf{T}_d$ is the transformation matrix between the goal frame seen from world frame.
The the goal of the control action will be to align the two frames to make the UAV reach the final destination. To achieve this alignment, the *PID controller* comes into place.
The ${}^{d}\mathbf{T}_g$ matrix represents the position error between the current position of the drone with respect to the goal position. Therefore, the position error used for the controller are directly extracted from it. The controller will be represented as:

$$ \begin{cases} v_x = K_{p,x} x_e + K_{i,x} \int_0^t x_e d\tau + K_{d,x} \frac{dx_e}{dt} \\ v_y = K_{p,y} y_s + K_{i,y} \int_0^t y_e d\tau + K_{d,y} \frac{dy_s}{dt} \\ v_z = K_{p,z} z_e + K_{i,z} \int_0^t z_e d\tau + K_{d,z} \frac{d_e}{dt} \\ \omega = K_{p,\theta} \theta_e + K_{i,\theta} \int_0^t \theta_e d\tau + K_{d,\theta} \frac{d\theta_e}{dt} \end{cases} \tag{6.43} $$

Where the position vector is extracted as $[x_e, y_e, z_e]^{-1} = {}^{d}\mathbf{T}_g[:, 3]$ and θ_e is calculated as $\theta_e = arctan((\ {}^{d}\mathbf{T}_g[1,0])/(\ {}^{d}\mathbf{T}_g[0,0]))$ (given the *Yaw, Pitch, Roll* rotation matrix representation where $R[1,0] = \cos(Pitch)\sin(Yaw)$ and $R[0,0] = \cos(Pitch)\cos(Yaw)$). The goal will be considered reached whenever the origin of the UAV frame enters an area of $10cm$ radius around the goal point.

Chapter 7

Experiments

This chapter discusses the steps for the setup of the experiments and the evaluation of the results. The first part discusses the evaluation of the NNs used, while the second one focuses on the evaluation of the control algorithms used to guide the UAV flights.

7.1 Semantic Segmentation evaluation

When creating ML applications, it is a standard approach to partition the dataset into three sub-sets: *train set*, *validation set*, and *test set*. As explained later, this division is meant to compare and evaluate how different NNs perform when fine-tuned over the same dataset. The evaluation of the results is usually based on the *confusion matrix*. Since this evaluation was not available within the setup used, an alternative technique was exploited for assessing the semantic segmentation accuracy, namely, the *Intersection over Union* evaluation (IoU). It is equal to the intersection of the predicted segmentation surface and the ground truth segmentation surface (in the manually segmented image), divided by the union of said areas, namely:

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|} = \frac{TP}{TP + FP + FN} \tag{7.1}$$

where TP, FP, and FN represent respectively the *true positive*, the *false positive*, and the *false negative* areas. The IoU is calculated using the validation set and the results are compared between the fine-tuning of different NNs performed exclusively on the train set. The best performing one will be retrained over the combination of the train set with the validation set and tested on the test set.

The results are then reported with the following metrics:

- the mean average precision (mAP), which denotes the average percentage of accurate predictions across all the evaluation prediction;

- the $AP50$, which is the percentage of accurate predictions, with a minimum IoU of at least 50%;
- the $AP75$, the percentage of predictions having IoU $\geq 75\%$;
- $mAPs$, $mAPm$, and $mAPl$, which are respectively the average precision for small, medium, and large detected objects [103].

7.1.1 Satellite images segmentation

After evaluation, the final NN chosen for the satellite image segmentation (Chap. 5) was the *pointrend_rcnn_R_50_FPN*, pre-trained on the *COCO* data-set [93]. Evaluating this NN against the *Mask_rcnn_R_50_FPN* model showed that the former has better performance than the unmodified version of the network (Sec.5.3.2). The *evaluation* results are listed in the following table: The evaluation

	mAP	**AP50**	**AP75**	**mAPs**	**mAPm**	**mAPl**
PointRend	72.1936	90.1970	84.4292	53.3027	79.8116	85.5761
Mask	47.973	74.922	56.733	30.070	59.457	47.249

difference is clearly noticeable also in a visual comparison between the two set of segmented photos.

After the training of the PointRend network was performed on the train set and validation set, the results for the test carried out on the test set are as in Tab. 7.1.1. The results of the test seem worse than the previous ones. A possible

mAP	**AP50**	**AP75**	**mAPs**	**mAPm**	**mAPl**
70.6961	91.6826	83.1612	51.5163	76.1567	80.4988

explanation could be that, when a NN is only trained on the training set, it can be expected to *overfit* to the training data. Overfitting to the train set means that the NN "memorizes" the data rather than catching the underlying patterns (which is the real goal of the training). Adding the validation set during training provides the model with a broader range of examples, enabling the acquisition of more robust features, therefore reducing the risk of overfitting.

7.1.2 Aerial images segmentation

A similar comparison has been done over the NNs for the run-time detection of the panels from the areal images. The performances of the *mask_rcnn_R_50_FPN* and *mask_rcnn_R_101_FPN* pre-trained on the *COCO* data-set [30] were evaluated.

	mAP	AP50	AP75	mAPs	mAPm	mAPl
50	94.9457	98.8106	97.8568	85.0495	77.0602	96.0152
101	96.5157	98.8516	98.8516	92.5253	81.8912	97.0473

The evaluation table is in Tab. 7.1.2. It is easy to see that the *mask_rcnn_R_101_FPN* model performed slightly better than the *mask_rcnn_R_50_FPN* network. However, since the camera feedback is essential for the control action, the *Inference Time* (IT) had to be considered. The IT represents how much time it takes to apply the segmentation mask on an image. This consideration is not taken into account when choosing the NN in the previous section, because the computational time for the path optimization is orders of magnitude larger than the time needed to infer the mask; therefore, the PointRend network was chosen there, given the large difference in terms of performance evaluation (IoU).

In this case, the frame rates of the two NNs are 15 *Frames Per Second* (FPS), for the R_50, and 5 FPS, for the R_101. Given the small difference between the segmentation quality and the large difference in the fps rate, the *mask_rcnn_R_50_FPN* was chosen over the *mask_rcnn_R_101_FPN*. The NNs inference time was based on the results obtained with the PC described in appendix A.2.

7.2 Path optimization test

As mentioned in Chap. 5, the TSP for connecting the given WPs can be solved using CPLEX (mixed integer linear programming), iterated local search, or simulated annealing. This section evaluates the performance of the three different algorithms in different scenarios, based on the number of panel rows of the plants and the UAV endurance.

7.2.1 Fixed time evaluation

The first evaluation is based on fixing the solve time of each of the three solvers and then comparing the results among four different PV plants of different dimensions. For the data collection, each algorithm has been run 30 times over each map. The box plots in Fig. 7.1 represent the collected data. Since there is no control on the time it takes to solve the TSP with the simulated annealing algorithm implementation used, the solving time was fixed with respect to the average time taken to solve the problem with this method.

From the graphs, it is easy to see how the *CPLEX* solution is outperformed by the heuristic ones. That is because the branch-and-cut method takes a considerable amount of time to explore the entire solution space of the problem instance aiming

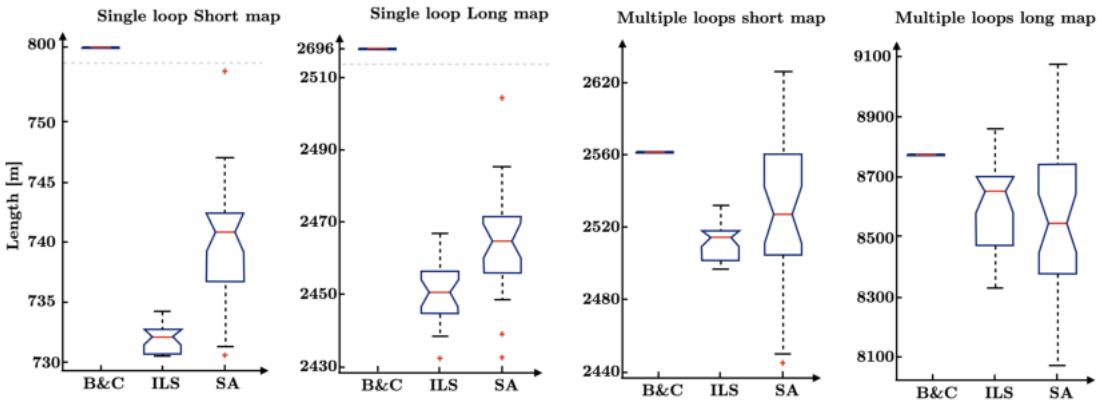


Figure 7.1: Average path length calculated by each of the algorithms on four different maps.

to find the optimal solution. In contrast, the heuristic solvers are designed to quickly find a solution which is not guaranteed to be optimal. The *CPLEX* solution starts its optimization process from an initial solution. If the initial solution is the same (or very similar) in each run, it can lead the solver to explore the same or similar regions of the solution space. This fact is noticeable in the given data, where the standard deviation of each branch-and-cut solution is null.

When comparing the two heuristic solutions, the iterated local search seems to find solutions with lower variance compared to the SA. Moreover, the average solutions of the ILS are better than the SA in three cases out of four. ILS normally performs a local search from n starting solutions (with n being the number of iterations), which can lead to better solutions for space exploration. SA, on the other hand, explores the space using probabilistic acceptance criteria, and its performance can be more susceptible to initial conditions and parameter settings. The higher variance of the SA algorithm could also be due to the algorithm's probabilistic nature, since it involves unpredictability in its search process. Therefore, SA outcomes may vary more throughout several runs, whereas ILS tends to be less sensitive to initial conditions.

7.2.2 Optimal solution time over different maps

Given that the ILS is the best performing heuristic algorithm, the time to optimally solve the TSP over the same PV plant (with an increasing number of panel rows) was compared with the time it takes for the optimal *CPLEX* solver to find the optimal solutions. The values shown in Fig. 7.2 represent the time-averages (over 30 runs) of the ILS and B&C solvers for each problem instance.

The semilogarithmic graph clearly shows how quicker the ILS solver is on

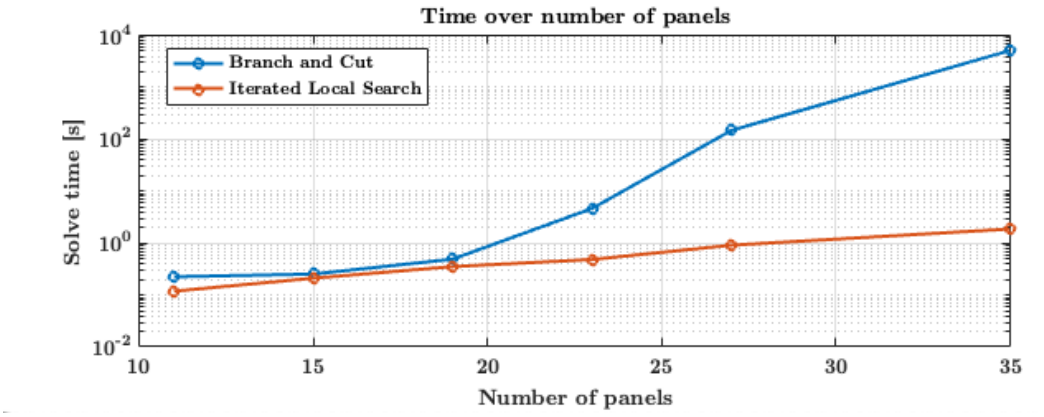


Figure 7.2: Comparison of the average times to reach the optimal solution, for the BR and ILS algorithms.

average. Since all the problem instances used for this comparison were composed of relatively few WPs, the ILS was capable of finding the global minima of the problem. However, given the heuristic approach of the solver, it is not possible to be always sure of finding the global minimum with such approach. Especially with larger instances of the problem, the ILS could eventually find local minima. Therefore, solving the problem with the *CPLEX* solver ensures the identification of the optimal solution, at the expense of an higher solving time. However, the issue with the B&C CPLEX solution is shown in the trend of the graph: the exponential increment in the number of possible solutions leads to an exponential growth of the number of branches, resulting in an exponentially-growing solve time.

7.2.3 Variation of path length given different time & iterations limits

The B&C and ILS can return different solutions when given, respectively, a different solve time limit and a fixed number of iterations. The two graphs in Fig. 7.3 show the trends for both algorithms, going respectively from 1 second and 1 iteration to the respective values needed to find the optimal solution.

Given the heuristic nature of the ILS algorithm, the best path does not always correspond to the highest number of iterations. Since the two graphs are in different units, it is hard to compare them. Therefore, the average solve time for each iteration step is reported. Once again, the ILS solve time is drastically shorter than the *CPLEX* one, confirming the conclusions drawn in Sec. 7.2.2.

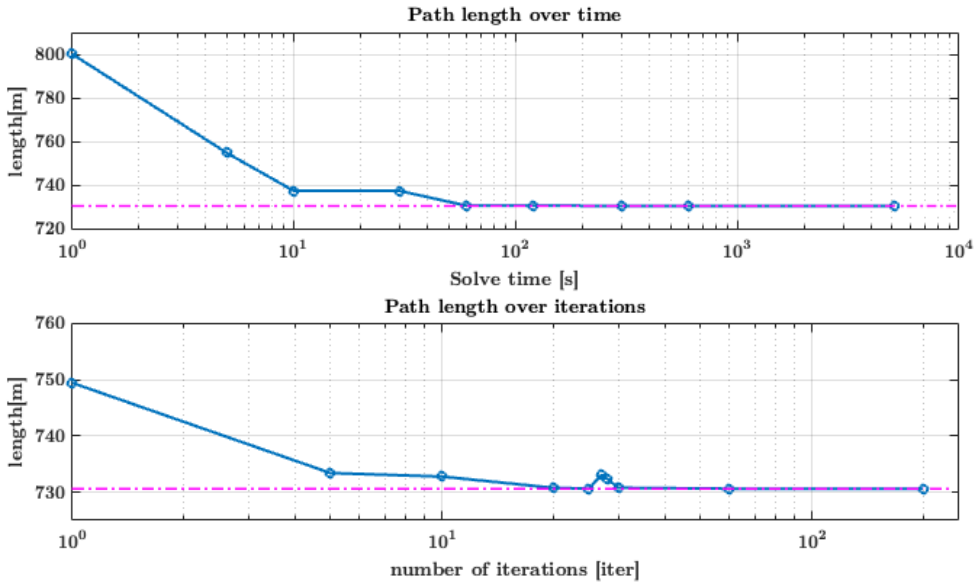


Figure 7.3: Path length over time for the BR algorithm and Path length over number of iterations for the ILS algorithm.

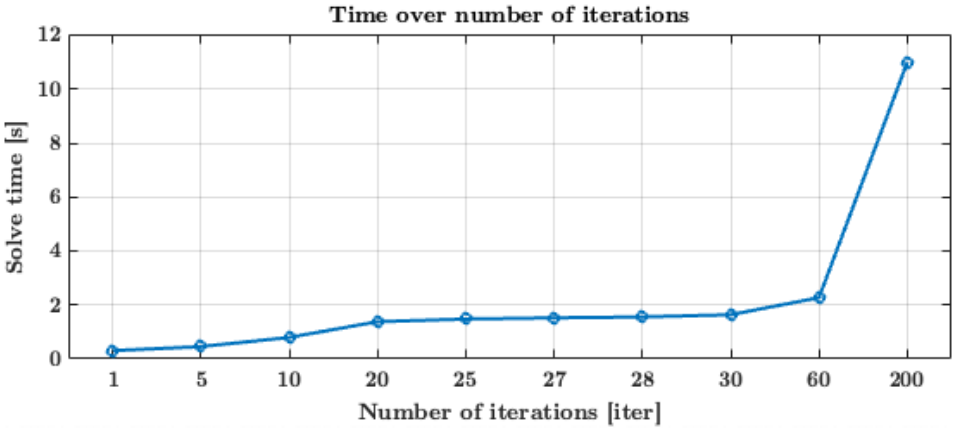
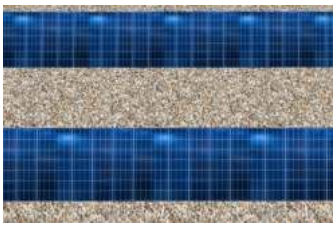


Figure 7.4: Average time variation for different numbers of iterations, for the ILS.

7.3 Control algorithm tests

This section is dedicated to the description of the data acquisition and results of the flight trail runs performed in the simulated, indoor, and outdoor environments. The first subsections will be dedicated to the description of the physical and simulated set-ups while the second one to the data evaluation of the test-flights.



(a) Ground and panel texture 1.

(b) Ground and panel texture 2.

(c) Ground and panel texture 3.

Figure 7.5: Three texture combinations.

7.3.1 Simulated environment set-up

In the simulation experiments, many different environments where a solar plant may be placed are represented, by combining eight different ground textures (three types of gravel, sand, dirt, mud, and two types of grass) below eight different 25 by 15 meter panels (Fig. 3.6). The panels are defined as a $(17/40) \times 2 \times 0.1$ meter prismatic object at an angle of 30 deg with respect to the ground. This inclination can vary for different solar plants, but is usually the same for every panel line inside the same plant. The simulated panel lines have four different textures, to simulate multiple panel types (Fig. 7.5).

Since this environment represented a real disposition of solar panels, is was used to deploy and tests the entire software architecture combining the vision based control to the position based one. An example of this could be checked out in this video.
Moreover, another simulated environment was arranged to test the capabilities of the three controllers in different simulated settings. The surface over which the flights were performed was dividend into 9 rows of panels each one containing a different configuration of two contiguous lines of panels (Fig.7.6), namely:

- A **straight** row between the two lines.
- ± 5 deg and ± 10 deg of angle with respect to *z-axis* of the world frame between the first and second line. This configuration represents a possible variation of the panels arrangement.
- ± 5 deg and ± 10 deg of angle with respect to *y-axis* of the world frame between the first and second line. This configuration simulates a plausible change of slope of the terrain.

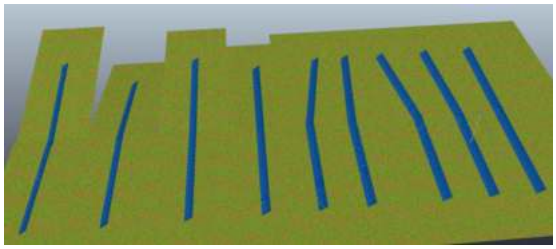


Figure 7.6: Screenshot of a simulation for different panel setups.

7.3.2 Indoor test-room set-up

The tests have been conducted inside of the UAV flight dedicated area in the university's *EMARO-LAB*. The dimensions of the area are $5 \times 4 \times 2.7$ and it was equipped with the aforementioned *OptiTrack* system (Fig.7.7, 7.8). However, the motion capture sensors did not completely cover the region. As a result, sometimes the drone found itself in this shadow area during some test flights. Moreover, a few issues regarding the *DJI* embedded obstacle detection raised up. The *DJI security control layer* is a precaution system embedded in the *DJI* software which allows the drone to avoid hitting the obstacles that it could encounter during flight. Whenever the direction of the command would drive the UAV towards an obstacle, the protection layer would not proceed with the given input resulting with the robot stopping its flight therefore interfering with the tests performances. To test out the the control algorithm inside the environment, a few elements had to be added to the test area to simulate a real world application. Some fabricated rows of panels have been made to mimic a real-world application. The first panel prototype was made from platified sheets of paper printed on both sides with the texture of different panel modules (Fig.7.7.*b*). These textures were identical to those used in the V-REP simulation: One represented a darker panel with no reflections and the other, a lighter color with some light artifacts indicating the possible light flares caused by the sun's reflection (Fig.7.7.*c*)). The Sheets of paper were then arranged in lines to represent the rows of panels.
The second version of the handcrafted panels were then made by printing the same two textures on both sides of a continuous sheet of plastic for advertising banners (Fig.7.7.*c*). This improvement made the detection much more reliable during tests.

For simulating the tilted panels, a scaled down rack was built as well. The objective was to tilt platified sheet panels to a constant angle. Therefore, the sheets have been stick onto a plastic surface into groups of three. This sheet was then tilted with an hand cut plastic stand. The issue with this hand made solution was related to the coupling between the panels and the stand. Since part of the stands would partially stick out from the front of the rows of panels,

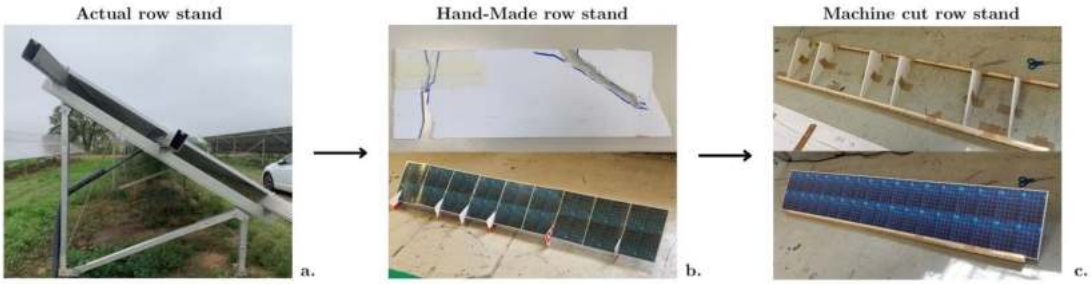


Figure 7.7: Different iteration of the handmade simulated panel.

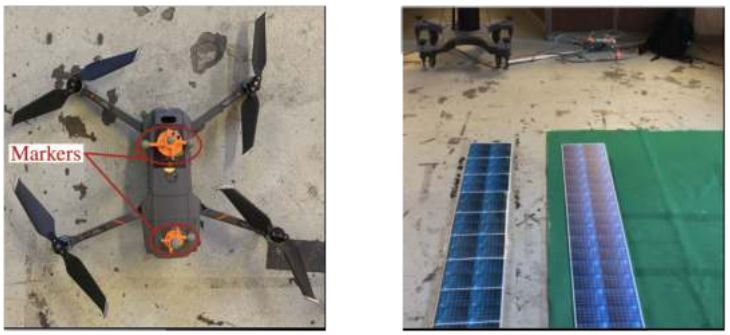


Figure 7.8: Different Types of ground surfaces and drone markers.

they interfered with the detection resulting in some issues with the control. To simulate the different types of ground some of the tests have been carried out on different color surfaces (Fig.7.8).

7.3.3 Outdoor test set-up

As a last proof for the performance capabilities of the system, a few outdoor tests were conducted during a one day field trip to an actual PV plant. The facility is located in a rural location called *Predosa(AL)*. JPDroni has a special partnership with the owner of the site who lets them carry out tests simulations in exchange of periodical inspections of the facility status.
The plant is the one depicted in the satellite image (Fig.5.7). It is composed of 28 rows of panels distributed in a *east-west* communication each one built as a 2 lines of solar panels modules. The tilt angle of the rows is fixed to 30 deg (Fig.7.7.*a*).

7.3.4 Simulation Tests

The following graphs will represent the position and orientation of the camera for the three different algorithms applied at constant lateral speed and constant altitude with respect to the ground. This representation will help identifying the algorithm most capable of following the row of panels regardless of the variations on the configurations. The *Root Mean Square* (RMS) of the errors in the $[X, Z]$ *top-view plane* and $[X, Y]$ *lateral-view plane* registered during the tests of the three controllers is calculated on the performances of the previously mentioned tests. The Performance of the controllers could be improved by tuning the controller gains and consequently improving their ability of convergence. A manual recursive approach was exploited to tune these parameters. However, manual tuning can be time-consuming, and the benefits obtained may only be ideal for certain operating situations. Furthermore, environmental conditions such as wind, temperature, and humidity might alter manual adjustment, causing the system to operate differently than planned. Alternatively, more standardized tuning procedures, such as the Ziegler-Nichols method, can be employed without personal tuning to obtain the optimal gains for a given system [104].

7.3.4.1 Straight configuration

[mm]	[X, Z] *err*			[X, Y] *err*		
	Lyap	*PID*	*VIS*	*Lyap*	*PID*	*VIS*
Linear	0.0431	0.0471	0.0434	0.0081	0.0120	0.0120

Table 7.1: Root Mean Square errors for each of the controllers in the linear track.

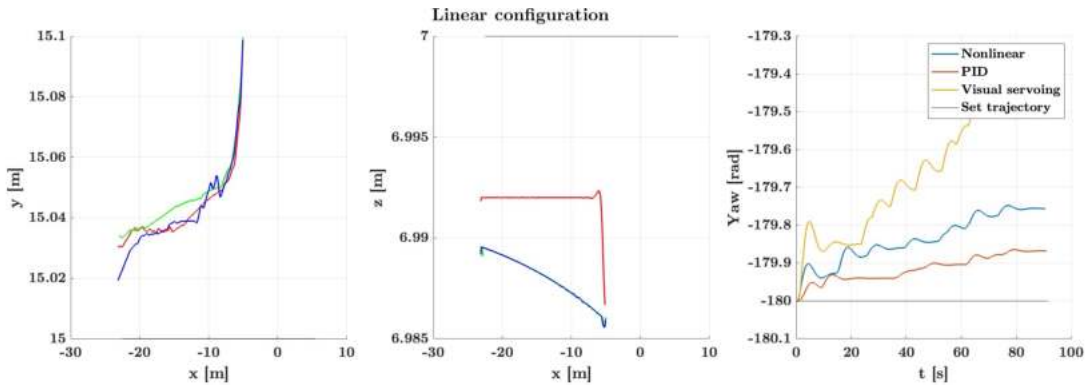


Figure 7.9: *a.*Error from the *top-view*; *b.* Error from the *lateral-view*; *c.* Rotation error over time.

From the error graphs (Fig.7.9) and the RMS reported on Tab. 7.1, it is possible to see how all of the control algorithms kept the position of the camera stable over the line of panels. The only slightly noticeable difference is related to the (Fig.7.9.*b*) graph related to the altitude profile. As it is possible to see the *Lyapunov* controller results to have a slightly less variable altitude trajectory with respect to the other two algorithms which they show a less linear profile. The constant off-set shown for all of the algorithms is related to the structure of the simulated Drone. Since there is a small offset between the position of the camera and the position of the altitude sensor on the drones' body frame, the alignment of the camera of the drone with the mid-line of the panel, prevents the altitude sensor to also be aligned with the panel's mid-line. Therefore, a constant altitude offset is noticeable for every algorithm.

7.3.4.2 Planar Rotation

[mm]	[X, Z] *err*			[X, Y] *err*		
	Lyap	*PID*	*VIS*	*Lyap*	*PID*	*VIS*
+10 deg	0.0510	0.0423	0.0482	0.0081	0.0119	0.0121
−10 deg	0.0390	0.0608	0.0388	0.0082	0.0120	0.0121
+5 deg	0.0508	0.0436	0.0515	0.0081	0.0120	0.0120
−5 deg	0.0301	0.0393	0.0309	0.0082	0.0120	0.0120

Table 7.2: Root Mean Square errors for each of the controllers in the *Tilted* track variations.

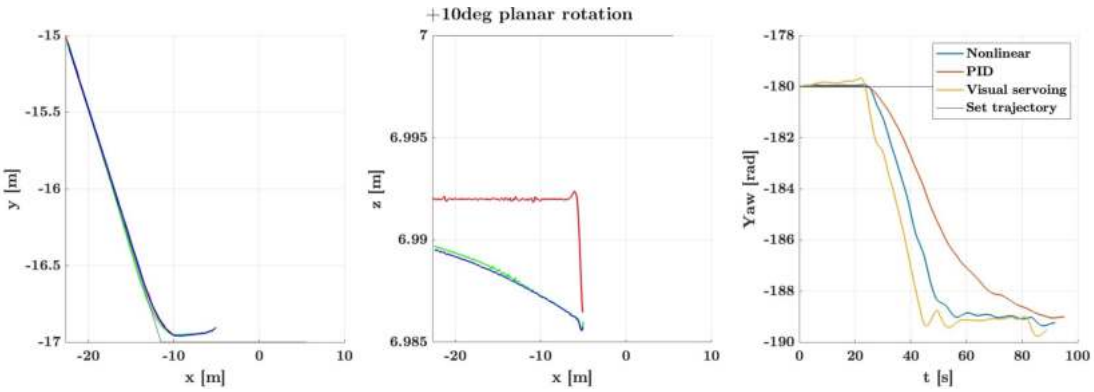


Figure 7.10: *a.*Error from the *top-view*; *b.* Error from the *lateral-view*; *c.* Rotation error over time.

Since the graphs of all four of the tests of this section were very similar, only the test with the 10 deg panel rotation was reported (Fig.7.10).

The biggest difference among the three graphs is visible in the rotation angle (Fig.7.10.*c*). The visible variation represented in the graph is given by the sudden change of angle of the panels configuration. Eventually, all three of the algorithms drive the drone to change the camera orientation to the tilted direction. However, as it is noticeable for all four panel configuration, The *Lyapunov non linear controller* with the *Task stacking visual based controller* are the fastest to reach the new configuration with respect to the *PID controller*. The slow angle correction of the *PID controller* may be due to a wrongly chosen gain values combination which leads to a slow convergence of the angle. Moreover, given the different nature of the two controllers, it is possible to notice how The *Lyapunov* one has a smoother convergence to the desired angle compared to the *Task stacking* based one. Concerning the *top-view* graph (Fig.7.10.*a*), all coordinates controllers have similar behavior. At first glance, all controllers seam to have an initial *off-set* along the *y-axis*. This is due to the line detection algorithms. The CV algorithm would detect the panel mid-line by drawing the tightest rectangular shape around the detected panel contour. In the case of a curved panel, the approximation into a rectangle does not match the tight angle of the panel configuration. Therefore, the linear error with respect to the panel's mid-line is due to a combination of the starting off-set of the drone and the gradual rotational adjustment of the trajectory to adapt to the approximated detected mid-line.

7.3.4.3 Vertical Rotation

[**mm**]	[**X**, **Z**] ***err***			[**X**, **Y**] ***err***		
	Lyap	*PID*	*VIS*	*Lyap*	*PID*	*VIS*
↑ 10 deg	0.0605	0.0613	0.0663	0.1845	0.1844	0.1848
↓ 10 deg	0.0466	0.0428	0.0577	0.0770	0.0765	0.0778
↑ 5 deg	0.0366	0.0393	0.0381	0.0949	0.0947	0.0952
↓ 5 deg	0.0336	0.0403	0.0387	0.0825	0.0822	0.0827

Table 7.3: Root Mean Square errors for each of the controllers in the *Up and Down* track variations.

Also for the case of the ground altitude variation, the difference between the graphs was very small. Therefore, only the +5 deg graph was reported for brevity. The starting position of the camera drone was right at 180 deg for all four of the tests. The visual difference between the three algorithms in Fig.7.11.*c* is to be considered acceptable since the degree's minimal increment is equal to ±1 deg and the difference between the goal and the angle oscillations range around the 1 deg. The most noticeable difference is related to the second graph (Fig.7.11.*b*)

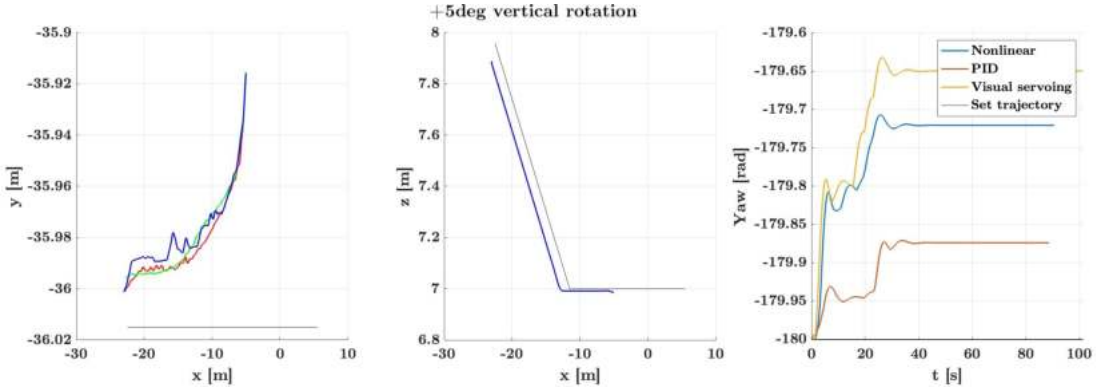


Figure 7.11: *a.*Error from the *top-view*; *b.* Error from the *lateral-view*; *c.* Rotation error over time.

representing the vertical trajectory of the drone over the lateral movement. In this view all three of the algorithms behave in a similar way but they all seam to induce a constant off-set on the *x-axis* with respect to the position of the panels setting. The difference was most likely due to a wrong identification of the construction of the second portion of the panel. During the creation of the scene the panel was slightly shifted towards the negative side of the *x-axis* with respect to the previously decided position of the panel reported in the graph.

7.3.5 Indoor Tests

The indoor tests were conducted, to check if the given gains were accurate even in a real world setting, to verify if the drone was properly moving accordingly to the given inputs, to simulate the across the panel movement and the tilted camera variation.
These evaluation were possible thanks to the run time feedback received from the *OptiTrack* system. The following subsection will summaries the results of the most representative tests for each of the controllers. The different tests were conducted with different initial conditions to check if the algorithms would properly work. An evaluation of the RMS error on the position of the Optitrack UAV frame during the flight tests with respect to the goal position was also conducted. This method of measurement is representative of the flight quality considering a constant shift between the camera position and the the Optitrack reference system relative to the global frame. The error will be represented as $(\rho_e, \mathbf{y}_e)$ where ρ_e represents the error on the plane (X, Z) of the simulation environment and y_e the error on the vertical axis Y with respect to the altitude goal position.

7.3.5.1 PID controller; [Video Test 5]

[mm]	***Test1*** 7.12	***Test2***	***Test3*** 7.15	***Test4***	*Test5* 7.16
ρ_e	34.51	33.68	26.43	41.31	18.36
$\mathbf{y}_e$	16.93	23.44	51.00	47.20	30.53

Table 7.4: PID RMS errors for each of the tests.

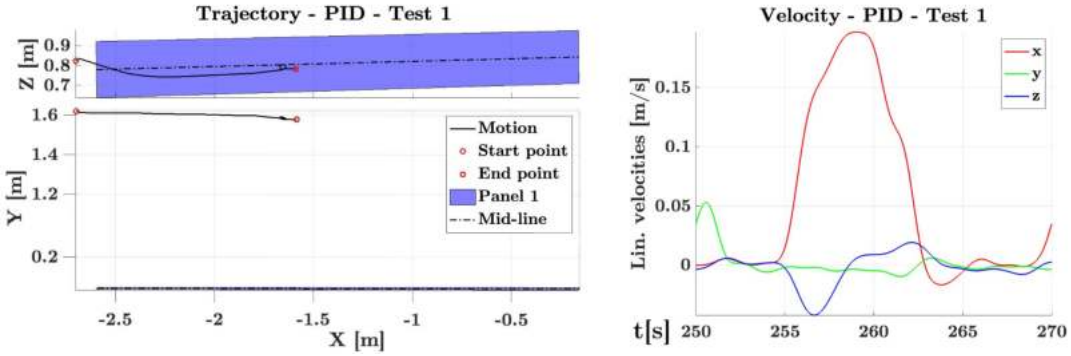


Figure 7.12: Top/lateral view | velocity of *Optitrack* data first PID test.

	$\mathbf{v_x}$	$\mathbf{v_z}$	$\omega_\mathbf{z}$
P	0.001	0.01	0.06
I	0.002	0.001	0.001
D	0.002	0.001	0.001

Figure 7.13: PID Gains first test set.

	$\mathbf{v_x}$	$\mathbf{v_z}$	$\omega_\mathbf{z}$
P	0.02	0.06	0.05
I	0.01	0.003	0.0054
D	0.001	0.012	0.01

Figure 7.14: PID Gains second test set.

The first three tests of the *PID controller* were made with the set of gains listed in Tab. 7.13.
As it is possible to notice in the graph 7.12, the flight stopped in the middle of the panel. The reason for this was the detection of an obstacles around the trajectory. The embedded *DJI control layer* would first induce some ripples in the movement (visible in the graph) and then the complete passage to an hoovering state. To avoid such issue the second test flight was conducted at the same lateral speed goal of $0.2m/s$ but with an altitude goal of $1.5m$ compared to the previous $1.6m$. Regardless of the new flight altitude the same issue happened again. The same thing happened once again with the third try where the goal was set to the higher altitude of $1.9m$ (Fig.7.15). From the footage recorded during the tests it was possible to notice that the table and tripod that were holding up the recording

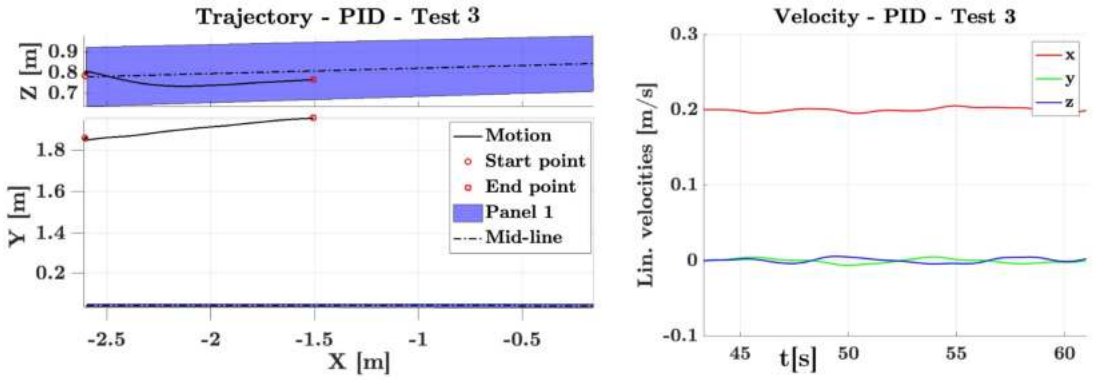


Figure 7.15: Top/lateral view | velocity of *Optitrack* data third test.

phone were interfering with the obstacle avoidance layer. Therefore, two other tests were conducted on a shorter and farther panel from the construction.

For brevity, only the graphs of the best RMS values test (Tab.7.4) were reported (Fig.7.16). The set of gains associated to the best performing are the ones listed in Tab.7.14. The lateral goal velocity was again $0.2m/s$ at a $1.8m$ goal height. The minimal increment for the altitude barometer sensor is $\pm 0.1m$ and it was noticed to not always be very precise since the very accurate altitude recorded by the *OptiTrack* sensors were several times different compared to the ones reported by the sensor it self.

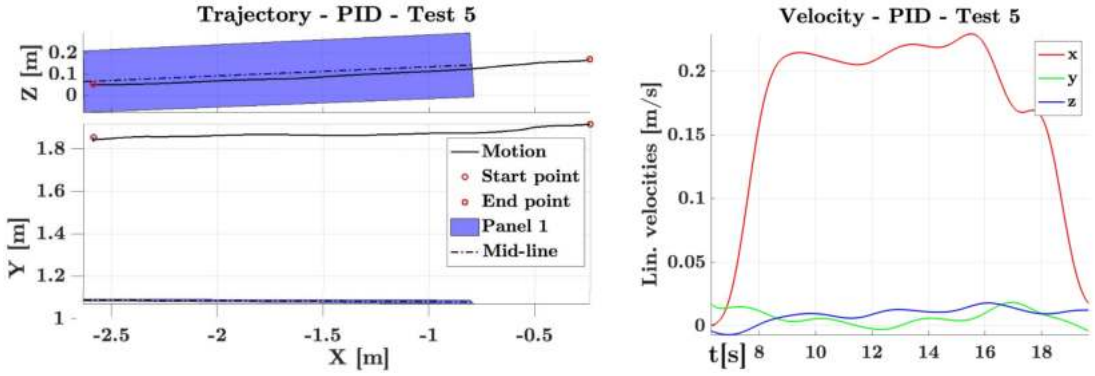


Figure 7.16: Top/lateral view | velocity of *Optitrack* data fifth test.

7.3.5.2 Lyapunov controller; [Video Test 5]

The first few test with the Lyapunov controller were made in similar conditions as the PID tests. The initial gains for the non linear controller were $K_x = 0.01$, $K_z = 0.01$, and $K_\theta = 0.01$. As it is shown in the graphs in Fig. 7.17, the test with

[mm]	*Test1* 7.17	*Test2* 7.19	*Test3*	*Test4*	*Test5* 7.20
ρ_e	55.62	178.83	94.64	35.87	33.25
$\mathbf{z}_e$	49.01	11.52	52.18	51.12	10.84

Table 7.5: RMS errors for each of the Lyapunov controller tests.

speed goal equal to $.2m/s$ and altitude goal at $1.7m$ recorded the same ripples and sudden stop of the trajectory as the PID test. However, since the RMS of this test was fairly low and the both the trajectories seam to be converging two other tests were carried out on the same panel configuration taking into consideration only the first part of the flight. The second test at the lower altitude of $1.6m$

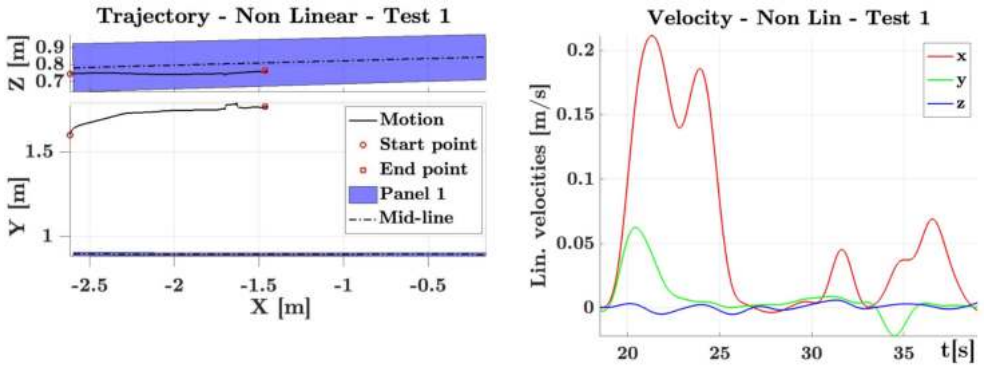


Figure 7.17: Top/lateral view | velocity of *Optitrack* data first Lyapunov controller test indoor.

showed a very interesting behavior of the drone. In the graph 7.19 referring to the *top-view* A trajectory completely wrong is reported. A more accurate analysis of the body of the drone showed a strange behavior of the drone's gimbal camera; After a few continuous flight the gimbal would reset to an *offset-ed* initial position corresponding to a *pan angle* as shown in picture 7.18.

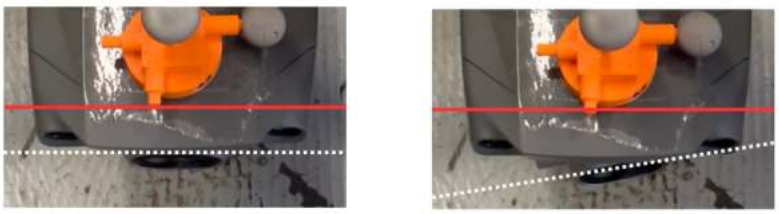


Figure 7.18: picture of the camera misalignment.

Since this issue kept happening, it became the cause of many delays in the multiple test sessions because the only way to reset this issue was to turn off the UAV and therefore connecting it again to ROS through the Bridge app.

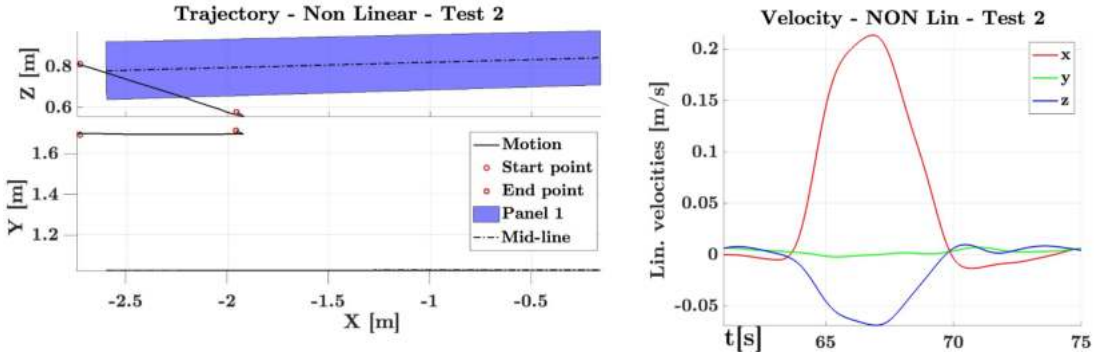


Figure 7.19: Top / lateral view | velocity of *OptiTrack* data second test.

Few more tests with the same gains but different goal heights were then performed (see table for the RMS of the third test in Tab. 7.5). As it is visible in the graphs the second set of gains seams to have brought even more stability to the UAV flight. The gains are for the following case equal to $K_x = 0.02$, $K_z = 0.01$, and $K_\theta = 0.05$. In the 7.20 test, the lateral velocity was set to be again equal to $0.2m/s$ and the altitude $1.8m$. In this case the panel used for the test was farther from the obstacle. Therefore, the control input didn't stop the drone from passing over the middle of the panel finishing the visit of the entire row. The altitude was kept stable at the give goal accordingly to the $\pm 0.1m$ barometers minimum threshold. However, it is possible to notice a slight miss alignment towards the end on the trajectory. This behavior might be due to a wrongly detected panel which ruined the final trajectory.

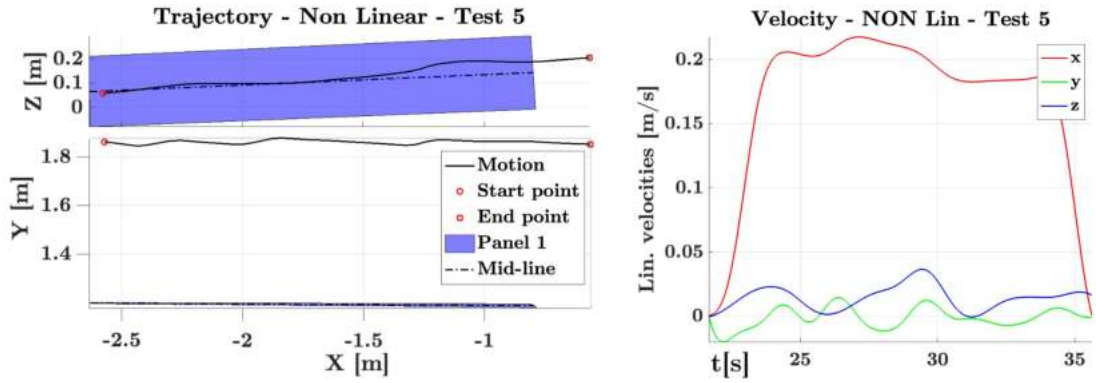


Figure 7.20: Top / lateral view | velocity of *Optitrack* data fifth test.

7.3.5.3 Task Stacking controller; [Video Test 6]

[mm]	*Test1*	*Test2* 7.21	*Test3*	*Test4* 7.22	*Test5*	*Test6* 7.23
ρ_e	50.23	41.46	32.61	33.77	40.69	39.80
$\mathbf{z}_e$	31.32	30.25	59.58	46.64	49.29	10.70

Table 7.6: RMS errors for each of the Visual servoing controller tests.

Also for the first few tests conducted with the *Visual Servoing Controller*, the flight was stopped by the presence of the obstacle. The λ gain matrix for the proportional control was equal to:

$$\lambda = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0.005 \end{bmatrix}$$

The following graphs will show first the best test performed on the panel close the obstacle (Fig.7.21), a test performed at an higher velocity goal (Fig.7.22), and the last test will show a clear detection error on a panel farther from the obstacle (Fig.7.23).
By looking at the first graph 7.21 it is possible to notice how the trajectory of the drone was stable around mid-line of the panel ensuring a good picture acquisition of the row. The goal velocity was again set to $0.2m/s$ and the altitude $1.6m$ which has been kept stable during the flight within the sensor's threshold of $\pm 0.1m$.

The second reported test (Fig.7.22) shows the best performance in terms of the line following. This was the first case where the goal velocity was set at an higher value of $0.4m/s$. Regardless, of the good results obtained in this test, the velocity for the remaining indoor tests was set back to $0.2m/s$ for safety reasons. The goal altitude for this test was $2.2m$ to try to avoid the obstacle stopping the test at mid-flight. Though, since the test started at a lower altitude, the RMS was high throughout the test. However, it is shown how the goal altitude was finally reached once reached the goal of the panel.

The last graph shows the flight data obtained for the last test conducted on the shorter panel. As it is possible to see from the position graph (Fig.7.23), the *z-axis* deviation brought the drone out of the trajectory because of a clear a wrong panel identification. This is probably due to the shape of the custom panel which sometimes was mistaken by the network given the not so representative nature of the panel modules.

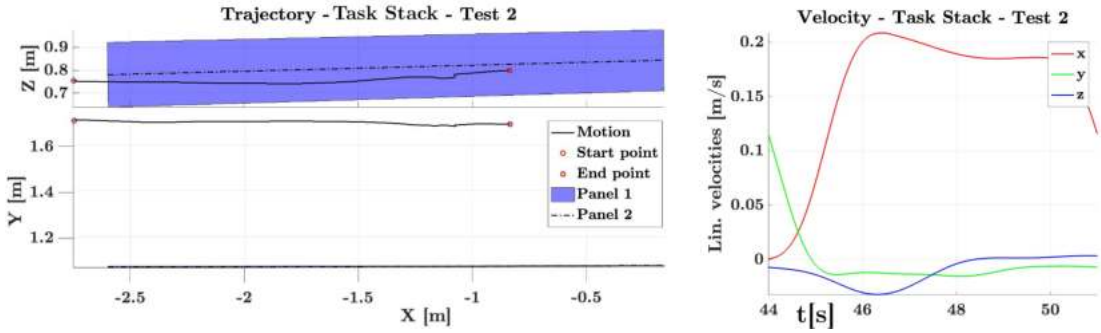


Figure 7.21: Top / lateral view | velocity of *Optitrack* data second test.

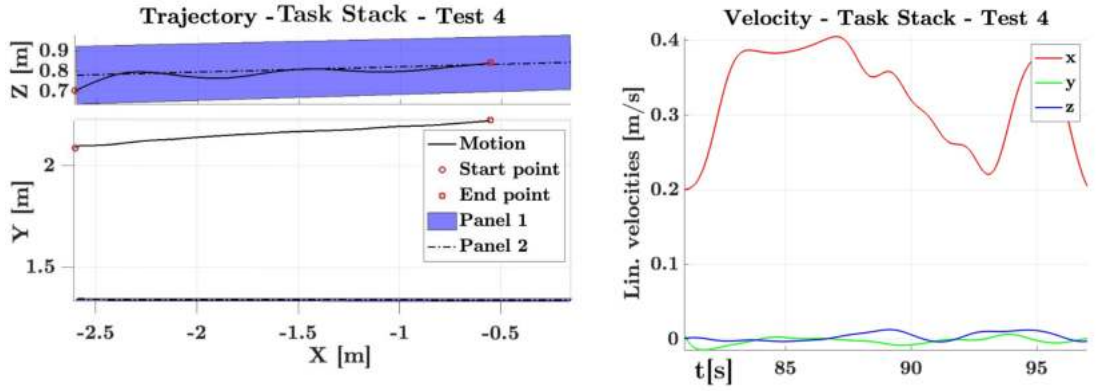


Figure 7.22: Top / lateral view | velocity of *Optitrack* data fourth test.

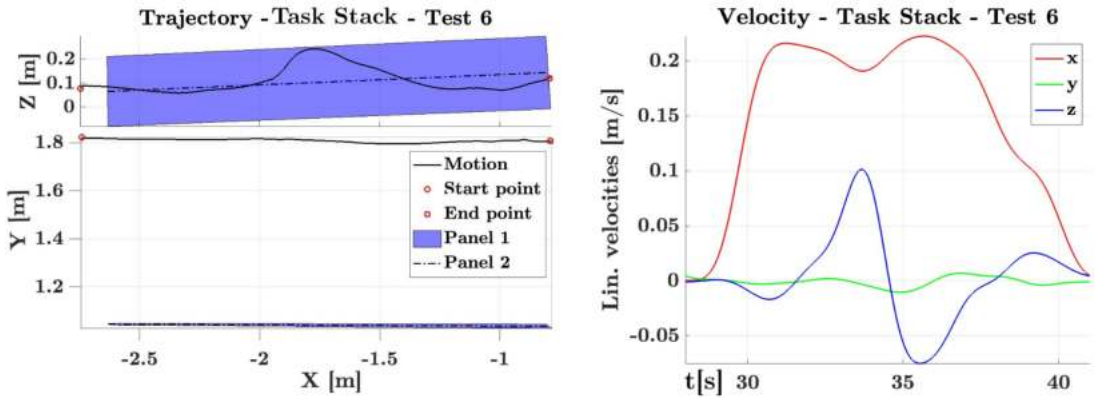


Figure 7.23: Top / lateral view | velocity of *Optitrack* data sixth test.

7.3.5.4 Angled panel following [Video Test]

This flight test represents the best performance of the Visual Seroving controller for the visit of an angled panel. To simulate the actual tilt of the solar panels,

the row had an angle of +30 deg around the *x-axis*. Therefore, having the UAV's height goal set at 1.6m, the distance along the z-axis was chosen accordingly to the geometry explained in Sec. 6.4.4.
The following graphs show the entire flight tour of the drone during the visit (Fig.7.24).

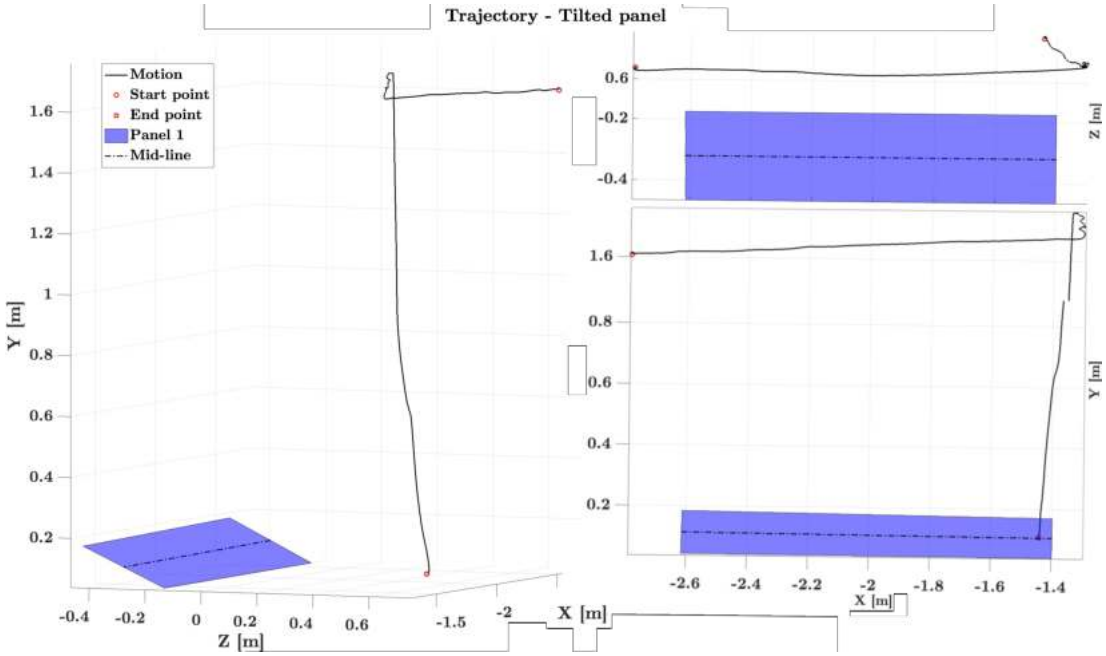


Figure 7.24: Prospective, top ,lateral view of the shifted visit.

7.3.5.5 PID position based controller

The (Fig.7.25) graph shows the position based controller used to simulate the GPS-driven control for across the panels movements.
The graph represents the entire *WP mission* where three WPs had to be reached. The green circles in the top and lateral view show the three goal areas the drone had to reach. To make sure that the goal position was properly reached, the drone had to record the presence of the drone inside of a spherical goal area of 1[dm] radius around the goal position for more than 2[s]. If such position was reached, the *ROS action* managing the drone movement would have set the following point of the list as the goal position. The PID gains used were exactly the same as the ones used for the PID visual based controller in Tab. 7.14.

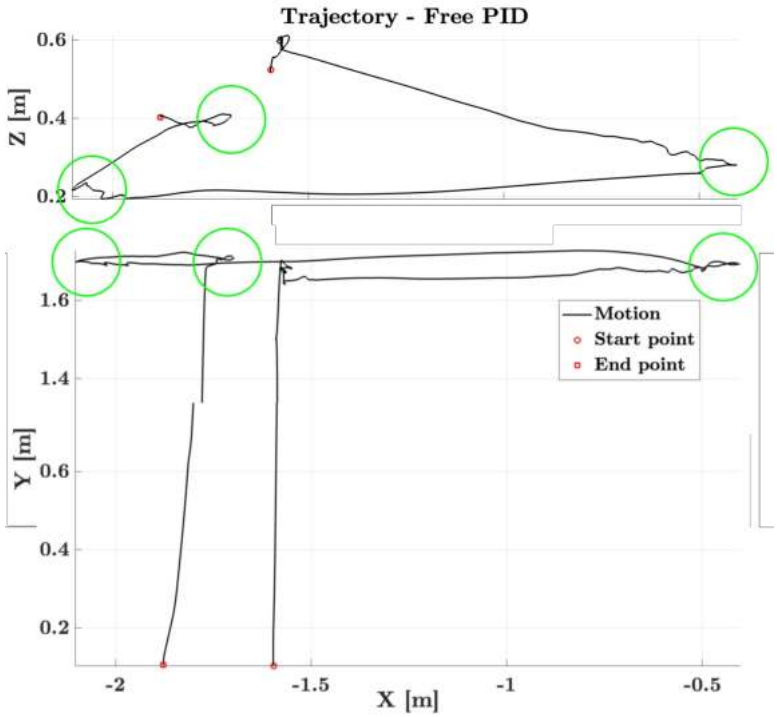


Figure 7.25: Position based controller graph from the top and lateral view.

7.3.5.6 Across the panel movement over multiple panels [Video Test]

The following graph representation shows the combination of the lateral movement of the drone based on the *OptiTrack* data simulating the GPS and the Visual Servoing controller. Both of the panels were visited at the lateral speed of $0.2m/s$ and at the altitude of $1.8m$ and $1.6m$. This difference makes it easy to distinguish well the three different trajectories in the lateral view (Fig.7.26).

From the top view is possible to see how well the first panel was followed compared to the following one. Given the initial starting offset given by the *across the panels* movement, the second panel had a noticeably different trajectory given the fact that it had to impose a correction due to the drone's starting location. The switch between the *across the panels* movement and the *Visual servoing* controller was controlled by the custom ROS action depicted in Chap. 3.

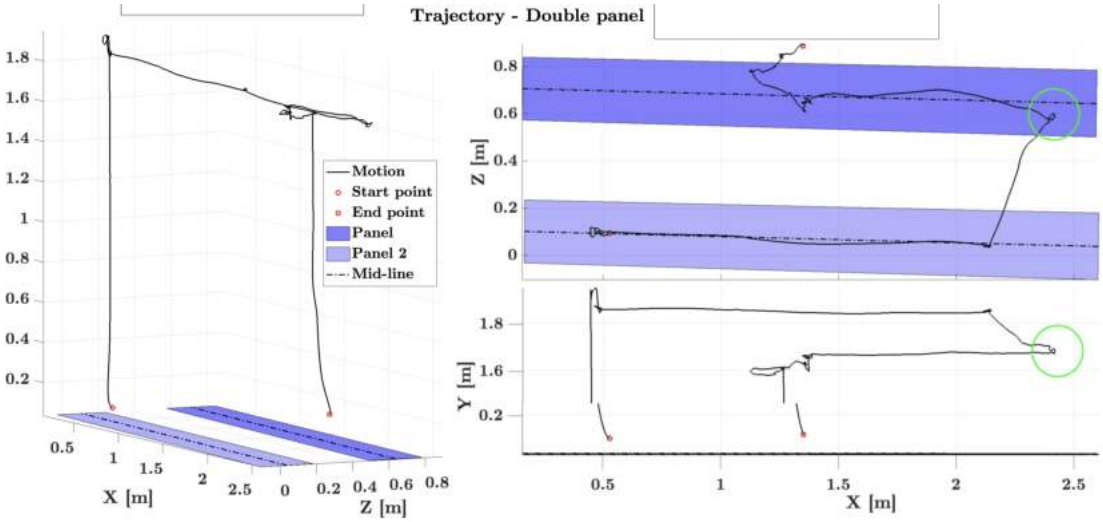


Figure 7.26: Prospective,top and lateral view of the two panels visit.

The video link Here shows the same test with a tilted panel visit in the simulated environment described in Chap. 3.

7.3.6 Outdoor Tests

These tests focused on conducting a trail run of the *Lyapunov* and *Task stacking* controller making the drone's flight start at different relative initial positions with respect to the panel and conducting the flight inspection at different flight heights and and different lateral velocities.

7.3.6.1 20.5m / Aligned start / Non linear control [Video Test]

These are the results of the first experiment of the session. The flight was performed at the altitude of 20.5m and at a lateral velocity of 1.5m/s. As it is possible to see from the graphs representing the ρ and θ image position error of the perceived panel, that the detections were very stable during the analyzed portion of the visit. Therefore, it was possible for the drone to align pretty quickly with the panel ensuring a good image capturing. The DL network worked way better outside compared to the performance indoor. This is because the NN was trained on actual areal images captured in real life settings (Chap.6).

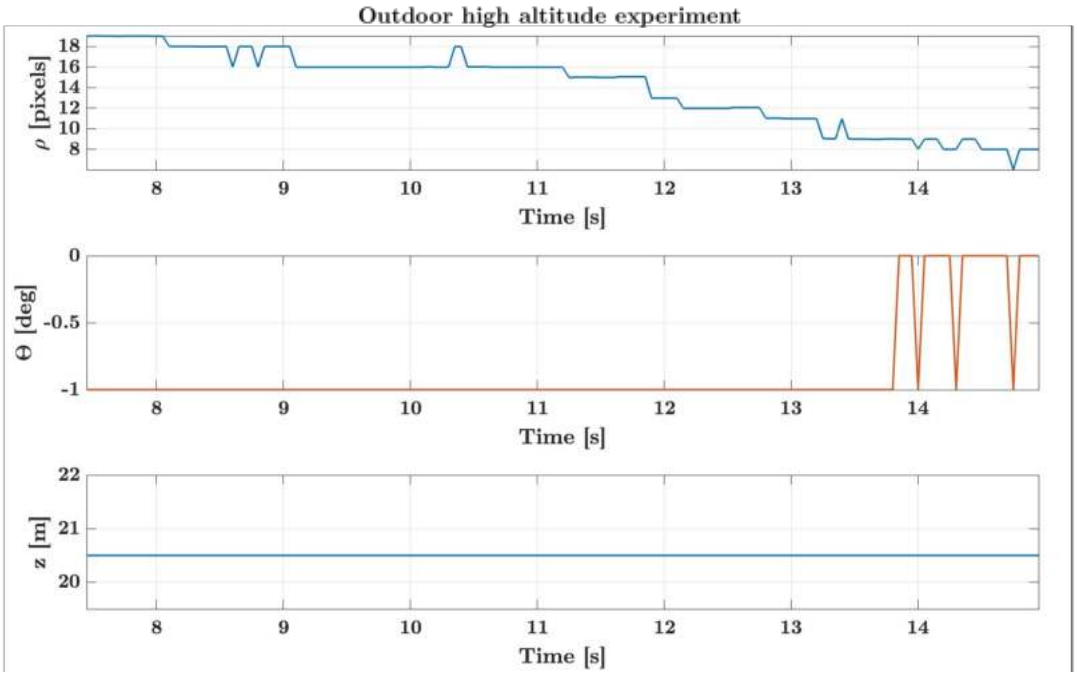


Figure 7.27: ρ, θ and z altitude values perception for the non linear controller.

7.3.6.2 10.5m / Aligned start / Non linear control [Video Test]

From the graph, it is possible to see how this performance was not as clean as the previous one. Since the drone was so close to the panel the CV processes extracting the goal line of the panel have often mistaken the alignment of the panel to be vertical instead of horizontal. Since the objective would be to fly lower on the panels, these detection errors are crucial for the functioning of the setup at lower altitudes. Therefore, a solution would be to both train the NN on images of the panels taken at a lower height or implementing the Kalman filter for rejecting the wrong detections.

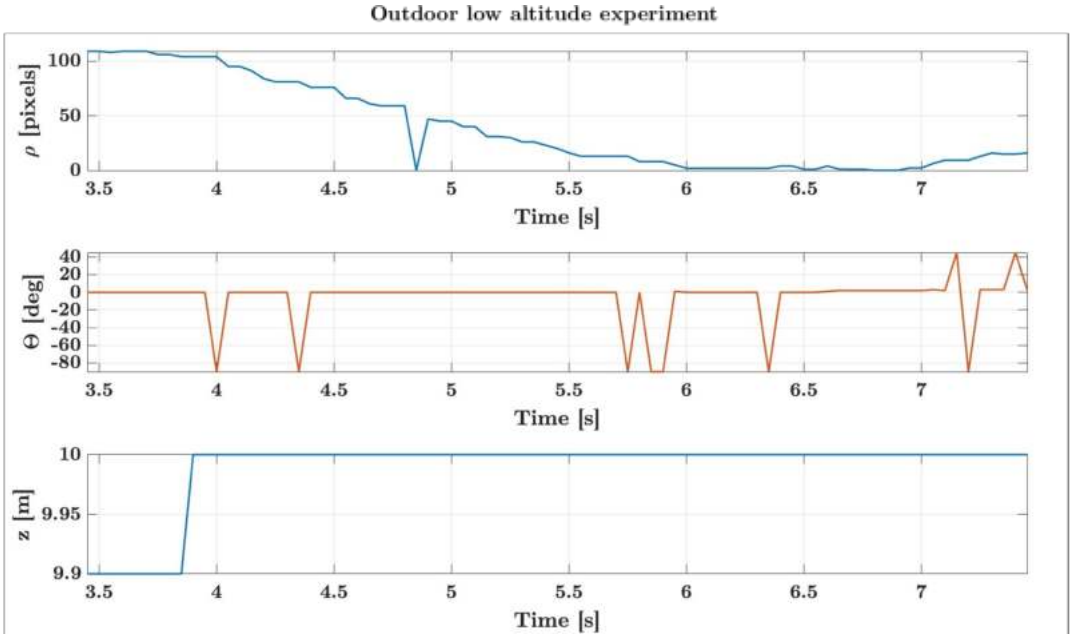


Figure 7.28: ρ, θ and z altitude values perception for the non linear controller controller.

7.3.6.3 23.8m / Angular off-set / Non linear control [Video Test]

This test had the objective of showing how the drone would behave when staring the visit of from a rotated position with respect to the desired line. As reported in the title of the section, the test was conducted at the fixed height of $18m$ with a constant lateral speed of $1.5m/s$. As it is possible to see from the graph, by using the *Lyapunov non linear controller*, the drone was capable of perfectly aligning its angle with the line of panels in around 10 seconds since the beginning of the test. The sudden oscillations visible in the ρ error graph are mainly due to small detection errors coming from the identification of the panels location. To deal with such detection error two solution concerning the application of filtering layers are reported in Sec. 6.4.3.

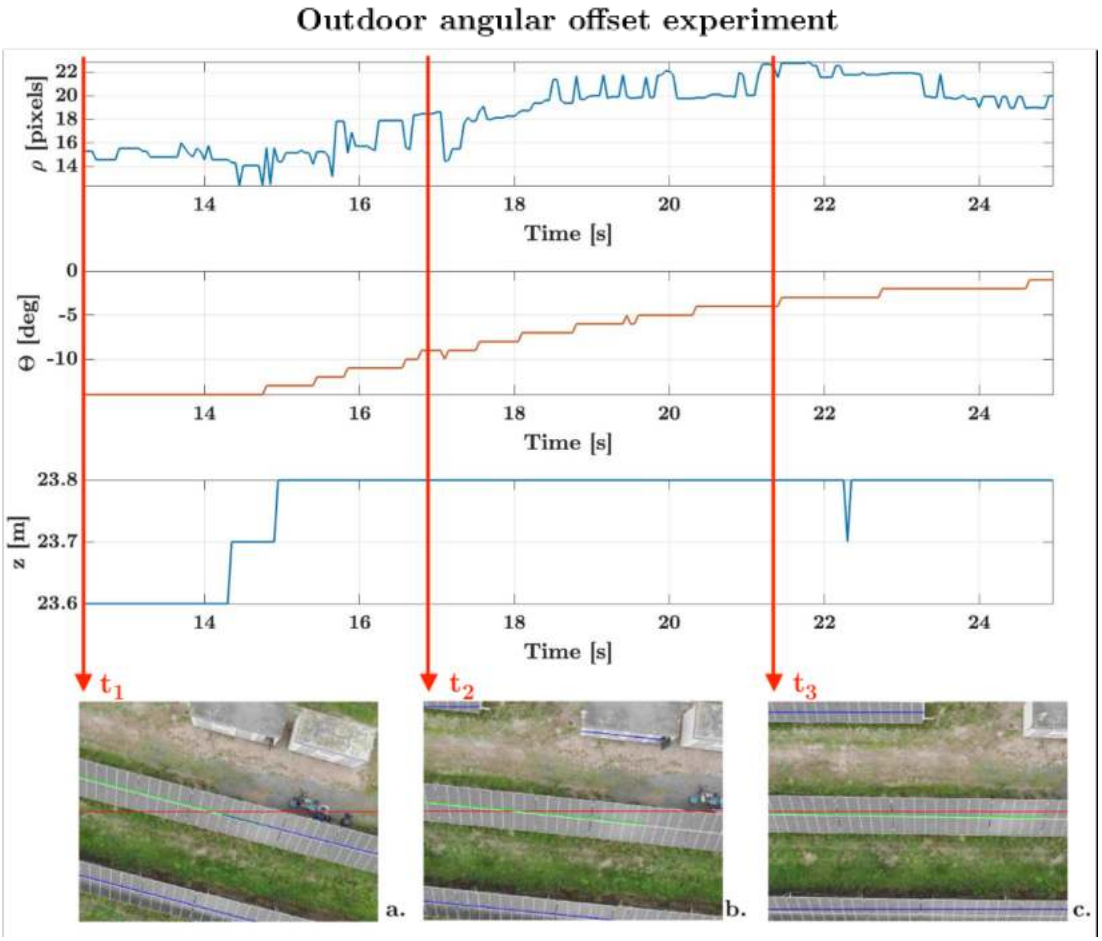


Figure 7.29: ρ, θ and z altitude values perception starting from an angular misalignment position | *a.* Image drone view at t_1; *b.* Image drone view at t_2; *c.* Image drone view at t_3.

7.3.6.4 22m / Linear off-set / Non linear control [Video Test]

This test had a staring condition concerning a linear offset from the panel line. As it is visible from the first graph of picture 7.30, the linear offset was gone after just 3 seconds of the *Lyapunov controller* action. Since the initial offset did not concern any angle misalignment the graph is arguably stable around a very small constant value.

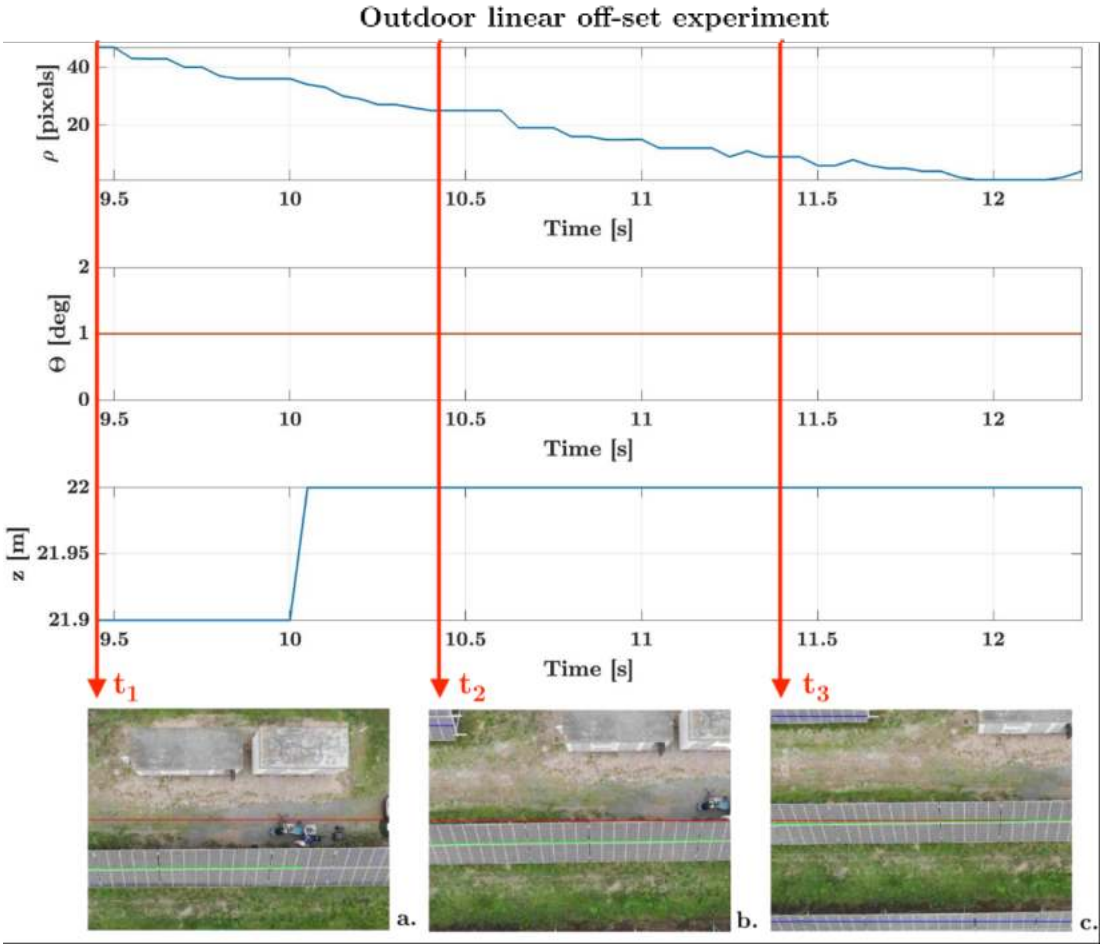


Figure 7.30: ρ, θ and z altitude values perception starting from a linear misalignment position.|*a.* Image drone view at t_1; *b.* Image drone view at t_2; *c.* Image drone view at t_3

7.3.6.5 20m / Linear off-set / Task Stacking [Video Test]

A similar task was performed for the Visual servoing controller As it is possible to see, with a few oscillations the controller was able to stabilize the flight around the mid-line of the panel just like for the non linear controller.

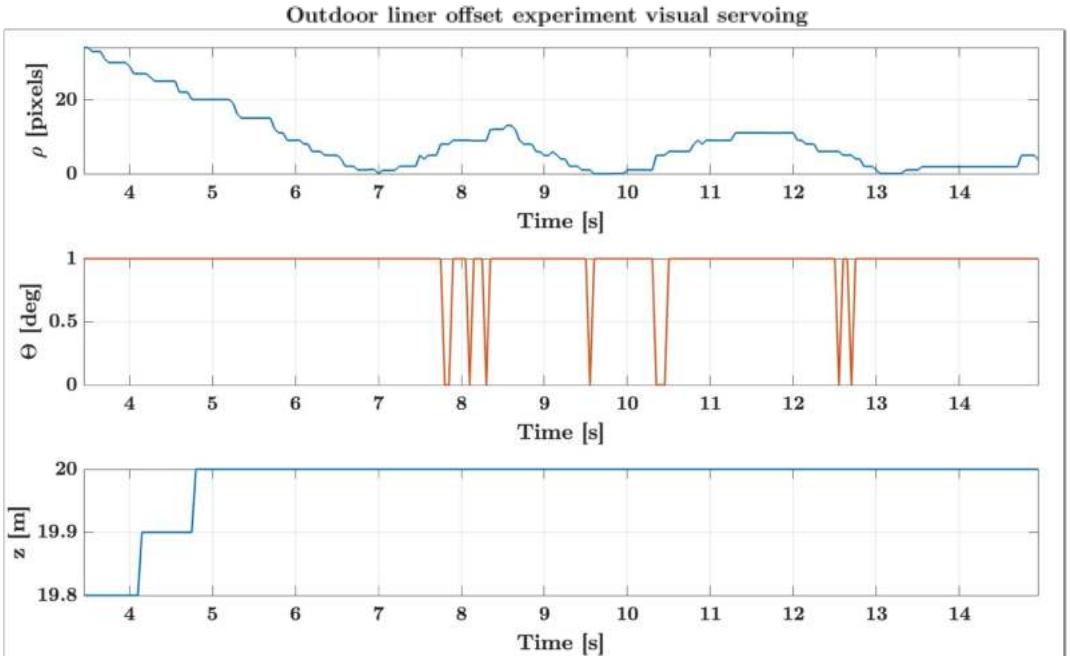


Figure 7.31: ρ, θ and z altitude values perception for the Task stacking visual based controller.

7.4 Links

The following links will connect to the YouTube videos, GitHab repositories, and Roboflow datasets related to the project:

Idoor tests

1. PID control single panel: https://youtu.be/uKugrg64Ir0.
2. Non Linear control single panel: https://youtu.be/DtzYR1kEhzE.
3. Task Stacking control single panel: https://youtu.be/2Tk-TND5spk.
4. Double panel: https://youtu.be/6SYh28fkWW4.
5. Task Stacking Tilted Panel: https://youtu.be/RXTMHEeZkJo.

Outdoor tests

1. High altitude, Non Linear controller: https://youtu.be/-W3krIS3pxM.
2. Low altitude, Non Linear controller: https://youtu.be/inCPx4gH0Zg.
3. Linear off-set, Non Linear controller: https://youtu.be/yoWSaf9Ue9E.
4. Linear offset, Task Stacking controller: https://youtu.be/bMub7nXpUME.
5. Angular offset, Non Linear controller: https://youtu.be/lFhh4Qirmuc.
6. Demo Flight with external cameras: https://youtu.be/4BKfuR4rpHI.

Simulation environment test Click https://youtu.be/-SZw5Kpt7hA to see the test of the entire project running in simulation. In the scene it will be possible to see both the coordinates based and the visual servoing control in action.

GitHub links (Not all repositories are updated at their last iteration).

1. *Drone control*: https://github.com/Fabioconti99/insp_panels.
2. *Route planning*: https://github.com/Fabioconti99/planning_routes.
3. NN training: https://github.com/Fabioconti99/Train_script_detectron2.
4. *V-rep environment*: https://github.com/Fabioconti99/V-rep_UAV_environment.

Roboflow Datasets

1. *Satellite Data-set*: https://universe.roboflow.com/fabio-conti/solar-panles-instance-segmentation.
2. *Aerial Data-set*: https://universe.roboflow.com/fabio-conti/solar-panels-mavic.

Chapter 8

Possible improvements and conclusions

8.1 Possible improvements

The following section outlines future developments that could improve the performance of the presented project, highlighting how these enhancements would take advantage of different system capabilities to increase the efficiency.

8.1.1 Improving NN segmentation from satellite images

As mentioned in Section 5.3.2, the dataset for the *pointrend_rcnn_R_50_FPN* NN has a bit under 500 images. Regardless of the data augmentation, such a short dataset can significantly limit the fine-tuning process and the overall performance of the NN. DL models, especially complex ones like PointRend, require a substantial amount of data to generalize different inputs and capture meaningful patterns in the given data. With a limited dataset, the network may struggle to learn representative features, leading to overfitting and poor generalization on unseen data. To achieve better performances, it would also be necessary to optimize the *hyperparameters* to fine-tune model weights.

Another strategy to improve the segmentation performance is to start the fine-tuning from a different NN. Many satellite image segmentation techniques are known [105], such as the *CloudRCNN* model. This NN outperforms earlier models thanks to added auxiliary branches, such as a *deconvolution decoder*, feature fusion module, and spatial attention module to enhance semantic segmentation and improve mean IoU compared to Mask R-CNN and PointRend [106].

8.1.2 Improving CPLEX optimization problem definition

8.1.2.1 Constraint redefinition

In the modified TSP considered in this work, the distance matrix is modified to force the passage over predefined edges. However, problems with irregular node distributions may be more difficult to solve, resulting in more complex patterns in the optimal tour, making it more difficult for the optimization algorithms to identify effective solutions and possibly increasing solution time. Thus, the constrained edge between the WPs on a single panel may be represented more efficiently to overcome this limitation.

A simpler problem instance with fewer WPs would be much easier for CPLEX to solve. Instead of interpreting each row of panels as a pair of WPs, the solution could incorporate the WP coupling and the cost of entering and exiting each WP. The constrained connection between a couple of WPs must be explicitly defined within the problem formulation. The new solution approach would involve two new features:

1. calculating the weight of an unvisited constrained WP, considering both the actual Euclidean distance from every other WP and the distance between the current WP and its coupled WP;

2. embedding a constrained travel to the WP couple.

This approach avoids considering this extra connection as an option based on the modified distance matrix, ultimately reducing the computational effort of the solver.

8.1.2.2 Sub-tour embedded solutions

To respect battery constraints, the *k-means* clustering method was employed. While this tactic can be very efficient, it is not necessarily the optimal approach. Introducing the docking station instance within the CPLEX problem definition could be beneficial. A possible modeling of energy expenditure is in [107], where a relationship is defined between the power consumption of the DJI Mavic 2 Enterprise (App. A.1) and relevant quantities such as weight, distance, and flight speed, using a polynomial curve fit based on the angular speed of the propellers. This model was then embedded within a CPLEX instance which represented multiple charging stations that would charge the batteries during the operation. Adding such a feature to the problem interpretation in this thesis work would allow finding the optimal routing of the UAV within the given plant.

Finally, another possible advancement is to identify the optimal placement for the recharging station.

8.1.2.3 Human-based path finding tests

Experiments involving a group of participants could be valuable for further assessing the effectiveness of the optimization algorithms presented. Volunteers would be asked to manually connect WPs, taking into account aspects such as the battery level and the forced edges. An app would be created for administering the test and keeping track of the results. This program would assist the users in choosing different paths while adhering to the stated rules. The gathered data would be analyzed with a focus on two critical aspects: the total path distance and the time required for human operators to find a solution. Comparing the results with those from the optimization algorithms will provide additional evidence of the efficiency of this work over manual approaches.

8.1.3 Pan-tilt gimbal control

For the visual servoing, the pan and tilt velocity controls of the camera gimbal could be added to the input vector. Indeed, the DJI Mavic 2 is equipped with a gimbal mount that connects the camera to the body of the UAV. This mount adds two DoFs to the camera velocity twist, allowing to control the pan and tilt. These allow to add new control capabilities such as:

1. automatically adjusting the camera tilt angle and the drone position, given the tilt angle of the panel rows;

2. helping to follow the lateral rotation of the panel rows.

This implies that new image features representative of such tasks have to be found and extracted from the RGB camera stream.

The new DoFs add two values to the control input vector namely, the velocity values of the camera angles: $[\dot{q}_p, \dot{q}_t]$. The control input vector finally becomes $\mathbf{u} = [v_x, v_y, v_z, \omega_z, \dot{q}_p, \dot{q}_t]$; compare with Eq. (6.14). The camera velocity $\mathbf{v}_c$ then becomes

$$\begin{aligned} \mathbf{v}_c = & \ {}^c\mathbf{W}_b\mathbf{v}_b + \mathbf{J}_q\dot{\mathbf{q}} = \\ = & \ {}^c\mathbf{W}_b\mathbf{A}\begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_z \end{bmatrix} + \mathbf{J}_q\dot{\mathbf{q}} = \\ = & \begin{bmatrix} {}^c\mathbf{W}_b^{(6\times6)}\mathbf{A}^{(6\times4)} & \mathbf{J}_q^{(6\times2)} \end{bmatrix}\mathbf{u}^{(6\times1)} = \\ = & \ \mathbf{J}_u^{(6\times6)}\dot{\mathbf{u}}^{(6\times1)} \end{aligned} \tag{8.1}$$

where matrix $\mathbf{J}_q$ represents the extra lines needed for the transformation between the camera frame and the drone body frame. The $\mathbf{L}^{(6\times6)}$ interaction matrix can now include the two additional DoFs, thus

$$\dot{\mathbf{s}} = \mathbf{L}^{(6\times6)}\mathbf{J}_u^{(6\times6)}\dot{\mathbf{u}}^{(6\times1)} = \mathbf{J}_s^{(6\times6)}\dot{\mathbf{u}}^{(6\times1)} \tag{8.2}$$

8.2 Conclusions

The goal of this thesis work was to create a prototype of a comprehensive software architecture capable of autonomously carrying out the planning and UAV flight control stages for a solar plant inspection mission. From user input through the finished mission, the software gives a rapid, precise, and optimized solution to the problem.

Planning and control are the two main elements of this work, which is meant as a proof of concept for the final design of the software. Nonetheless, the overall approach works nicely under the test conditions. JP Droni will also utilize a modified version of the stand-alone planning app for their WP-based inspection operations. The upgraded version of the application includes a manual step that allows the user to visualize and alter the .KML file defining the rectangles that contain the panel rows. This adds another layer of control to the NN detection, which sometimes can identify the panels rows incorrectly in the photos.

The autonomous flight control architecture, together with the different controller methods, provides the foundation for an alternative solution to the inspection issues caused by uneven terrain surfaces and disalignments in the panel lines. This system has proven capable of performing accurate image capturing during flight inspections, overcoming the constraints of the WP-based method already in use.

The experiments conducted proved the practicality and viability of the proposed methods. The thesis work, including NN training, simulated tests, and physical flight experiments, provides a good example of the system capabilities in a real-world setting.

Finally, the work emphasizes the necessity of UAV-based autonomous flight for improving monitoring and maintenance of solar plants and other facilities. The software architecture for route planning and autonomous flight control lays a basis for the complete automation of this procedure. While the current regulatory landscape presents some constraints, it is anticipated that, as technology evolves and demand for intelligent systems grows, autonomous inspection techniques similar to those proposed here will gain relevance and acceptance in the renewable energy sector, encouraging a sustainable and efficient energy future.

Appendix A

Hardware used

A.1 DJI Mavic Pro: specifications

Figure A.1: Mavic 2 Enterprise Dual

The relevant specifications for the drone used in this thesis work (from the datasheet by DJI) are reported in Tab. A.1.

DJI offers a mobile SDK for custom programming their products, which however are only compatible with mobile devices and limit the applicability of any apps thus built to other DJI systems. Additionally, using machine learning methods for visual recognition operations is difficult, as it requires a level of GPU performance which is not provided by the SDKs. To address this issue, a software interface has been developed using the mobile SDK to bridge data between the DJI controller and a Linux computer via socket connections [108].

Aircraft		**Camera**	
Weight (no accessories)	899 g	*Max bit-rate*	100 MB/s
Max take-off weight	1100 g	*Field of view*	85°
Max speed	72 km/h	*Photo format*	JPEG
Max ascent speed	5 m/s	*Video format*	MP4, MOV
Max descent speed	3 m/s	*Image size:*	
Max angle	35°	*4:3*	4056×3040
Max wind speed	38 km/h	*16:9*	4056×2280
Temperature range	From −10 to 40 °C		
Max operating height	6000 m		
Max distance:			
FCC transmission	8 km		
CE transmission	5 km		
Battery life (hovering)	About 29 minutes		
Charging time	About 90 minutes		

Table A.1: DJI Mavic 2 Enterprise Dual specifications

A.2 HP Omen computer

The computer used for this application is part of the OMEN line by HP (model: *17-ck1026nl*) and has the following specifications:

- *CPU*: Intel Core $i9 - 12900HX$
- *Solid state memory*: 2 TB
- *SDRAM*: SODIMM DDR5 32 GB
- *GPU*: NVIDIA GeForce RTX 3080 Ti, 16 GB of RAM

References

[1] Yangda. Yangda pictures. url = https://www.yangdaonline.com/, 2023. x, 12

[2] Yunlong Song, Mats Steinweg, Elia Kaufmann, and Davide Scaramuzza. Autonomous drone racing with deep reinforcement learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1205–1212. IEEE, 2021. x, 17

[3] A. Greco, C. Pironti, A. Saggese, M. Vento, and V. Vigilante. A deep learning based approach for detecting panels in photovoltaic plants. In *ACM International Conference Proceeding Series*, 2020. x, 20

[4] Amir Mohammad Moradi Sizkouhi, Mohammadreza Aghaei, Sayyed Majid Esmailifar, Mohammad Reza Mohammadi, and Francesco Grimaccia. Automatic boundary extraction of large-scale photovoltaic plants using a fully convolutional network on aerial imagery. *IEEE Journal of Photovoltaics*, 10(4):1061–1067, 2020. x, 3, 21, 22

[5] Andrés Pérez-González, Álvaro Jaramillo-Duque, and Juan Bernardo Cano-Quintero. Automatic boundary extraction for photovoltaic plants using the deep learning u-net model. *Applied Sciences*, 11(14):6524, 2021. x, 22, 23, 24

[6] A.M. Moradi Sizkouhi, S. Majid Esmailifar, M. Aghaei, A.K. Vidal De Oliveira, and R. Ruther. Ieee photovoltaic specialists conference. In *Autonomous Path Planning by Unmanned Aerial Vehicle (UAV) for Precise Monitoring of Large-Scale PV plants*, pages 1398–1402, 2019. x, 3, 24, 27, 28

[7] Howie Choset, Kevin M Lynch, Seth Hutchinson, George A Kantor, and Wolfram Burgard. *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005. x, 29

[8] Yoav Gabriely and Elon Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of mathematics and artificial intelligence*, 31(1):77–98, 2001. x, 27, 28, 29

[9] Alexander Zelinsky, Ray A Jarvis, JC Byrne, Shinichi Yuta, et al. Planning paths of complete coverage of an unstructured environment by a mobile robot. In *Proceedings of international conference on advanced robotics*, volume 13, pages 533–538, 1993. x, 27, 28, 29

[10] Samir Bouabdallah. Design and control of quadrotors with application to autonomous flying. Technical report, Epfl, 2007. x, 30, 31, 32

[11] Z. Xi, Z. Lou, Y. Sun, X. Li, Q. Yang, and W. Yan. International symposium on distributed computing and applications for business engineering and science. In *A Vision-Based Inspection Strategy for Large-Scale Photovoltaic Farms Using an Autonomous UAV*, pages 200–203, 2018. x, 3, 34, 35, 36, 37

[12] François Chaumette, Seth Hutchinson, and Peter Corke. Visual servoing. In *Springer Handbook of Robotics*, pages 841–866. Springer, 2016. x, 37, 38, 93, 94

[13] A. Di Tommaso, A. Betti, G. Fontanelli, and B. Michelozzi. A multi-stage model based on YOLOv3 for defect detection in PV panels based on ir and visible imaging by unmanned aerial vehicle. *Renewable Energy*, 193:941–962, 2022. x, xiii, 34, 39, 40, 41

[14] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. Pointrend: Image segmentation as rendering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9799–9808, 2020. xi, 5, 64, 65

[15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. xi, 84

[16] Andrinandrasana David Rasamoelina, Fouzia Adjailia, and Peter Sinčák. A review of activation function for artificial neural network. In *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 281–286. IEEE, 2020. xiii, 18, 19

[17] Paolo Scuteri Jacopo Callà. JP droni online web page. `https://www.jpdroni.it`, 2023. 1, 16

[18] Gail E Tverberg. Oil supply limits and the continuing financial crisis. *Energy*, 37(1):27–34, 2012. 1

[19] Ehab Salahat, Charles-Alexis Asselineau, Joe Coventry, and Robert Mahony. Waypoint planning for autonomous aerial inspection of large-scale solar farms. In *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*, volume 1, pages 763–769. IEEE, 2019. 1

[20] Edson L Meyer and E Ernest Van Dyk. Assessing the reliability and degradation of photovoltaic module performance parameters. *IEEE Transactions on reliability*, 53(1):83–92, 2004. 2

[21] Timothy M Walsh, Zhengpeng Xiong, Yong Sheng Khoo, Andrew AO Tay, and Armin G Aberle. Singapore modules-optimised PV modules for the tropics. *Energy Procedia*, 15:388–395, 2012. 2

[22] Paula Mints. The commercialization of thin film technologies: Past, present and future. In *2010 35th IEEE Photovoltaic Specialists Conference*, pages 002400–002404. IEEE, 2010. 2

[23] Ricardo A Marques Lameirinhas, João Paulo N Torres, and João P de Melo Cunha. A photovoltaic technology review: History, fundamentals and applications. *Energies*, 15(5):1823, 2022. 2

[24] M. Aghaei, F. Grimaccia, C.A. Gonano, and S. Leva. Innovative automated control system for PV fields inspection and remote control. *IEEE Transactions on Industrial Electronics*, 62(11):7287–7296, 2015. 2

[25] Andrés Pérez-González, Nelson Benítez-Montoya, Álvaro Jaramillo-Duque, and Juan Bernardo Cano-Quintero. Coverage path planning with semantic segmentation for UAV in PV plants. *Applied Sciences*, 11(24):12093, 2021. 3, 21, 22, 24, 27

[26] DJI. DJI pictures. "url:https://www.dji.com/it", 2023. 3, 11, 56

[27] A. Barbón, C. Bayón-Cueli, L. Bayón, and V. Carreira-Fontao. A methodology for an optimal design of ground-mounted photovoltaic power plants. *Applied Energy*, 314:118881, 2022. 4

[28] Stamatia Dasiopoulou, Vasileios Mezaris, Ioannis Kompatsiaris, V-K Papastathis, and Michael G Strintzis. Knowledge-assisted semantic video object detection. *IEEE Transactions on Circuits and Systems for Video Technology*, 15:1210–1224, 2005. 4

[29] Glenn Jocher. YOLOv8 - Ultralytics. https://ultralytics.com/yolov8, 2023. 4, 61

[30] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. 5, 64, 102

[31] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 5

[32] David L Applegate, Robert E Bixby, Vašek Chvátal, William Cook, Daniel G Espinoza, Marcos Goycoolea, and Keld Helsgaun. Certification of an optimal tsp tour through 85,900 cities. *Operations Research Letters*, 37(1):11–15, 2009. 5

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 6, 79

[34] Hilary Arksey and Lisa O'Malley. Scoping studies: towards a methodological framework. *International journal of social research methodology*, 8(1):19–32, 2005. 7

[35] Samuele Depalo, Carmine Tommaso Recchiuto, and Antonio Sgorbissa. Applications of unmanned aerial vehicles in maritime transportation industry: A scoping review. *Available at SSRN 4334518*, 2023. 9

[36] Isabelle Fantoni. Modelling of terrestrial and aerial vehicles. Technical report, IEEE, 2016. 9, 11, 31, 32

[37] Vitaliy Sigarev, Tatiana Kuzmina, and Alexey Krasilnikov. Real-time control system for a dc motor. In *2016 IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW)*, pages 689–690. IEEE, 2016. 10

[38] Appliedaeronautics. Applied aeronautics pictures. "url = https://www.appliedaeronautics.com/", 2023. 11

[39] Delair. delair pictures. "url = https://delair.aero/delair-commercial-drones/professional-mapping-drone-delair-ux11/", 2023. 11

[40] Therese Jones. International commercial drone regulation and drone delivery services. Technical report, RAND Santa Monica, CA, USA, 2017. 12

[41] Raphaela Chakravarti, Seira Iwai, and Suhara Wijewardane. Strategies to improve the social acceptability of drones. *Book print*, 2021. 12

[42] Enac. Enac definition. "url:https://www.enac.gov.it/", 2023. 12

[43] Claudia Stöcker, Rohan Bennett, Francesco Nex, Markus Gerke, and Jaap Zevenbergen. Review of the current state of UAV regulations. *Remote sensing*, 9(5):459, 2017. 13

[44] Ente Nazionale per l'Aviazione Civile. Direzione centrale regolazione aerea. dispense enac patente a1-a3, regolamento uas-it edizione 1 del 4 gennaio 2021. *Online resurce*, 2021. 13

[45] National aviation authorities. D-flight, 2023. 14

[46] Sergio Bemposta Rosende, Javier Sánchez-Soriano, Carlos Quiterio Gómez Muñoz, and Javier Fernández Andrés. Remote management architecture of UAV fleets for maintenance, surveillance, and security tasks in solar power plants. *Energies*, 13(21):5712, 2020. 14

[47] V Bryan. Drone delivery: DHL'parcelcopter'flies to German isle. *Reuters*, 1(September 24), 2014. 15

[48] Joanna Stern. Like amazon, ups also considering using unmanned flying vehicles. *ABC News, Dec*, 3, 2013. 15

[49] University of Genoa. Smart Ambulance project. https://life.unige.it/progetto-smart-ambulance, Year Accessed. 15

[50] Irwin O. Reyes, Peter A. Beling, and Barry M. Horowitz. Adaptive multiscale optimization: Concept and case study on simulated UAV surveillance operations. *IEEE Systems Journal*, 11(4):1947–1958, 2017. 15

[51] Haris Balta, Janusz Bedkowski, Shashank Govindaraj, Karol Majek, Pawel Musialik, Daniel Serrano, Kostas Alexis, Roland Siegwart, and Geert De Cubber. Integrated data management for a fleet of search-and-rescue robots. *Journal of Field Robotics*, 34(3):539–582, 2017. 15

[52] Teodor Tomic, Korbinian Schmid, Philipp Lutz, Andreas Domel, Michael Kassecker, Elmar Mair, Iris Lynne Grixa, Felix Ruess, Michael Suppa, and Darius Burschka. Toward a fully autonomous UAV: Research platform for indoor and outdoor urban search and rescue. *IEEE robotics & automation magazine*, 19(3):46–56, 2012. 15

[53] Paolo Tripicchio, Massimo Satler, Giacomo Dabisias, Emanuele Ruffaldi, and Carlo Alberto Avizzano. Towards smart farming and sustainable agriculture with drones. In *2015 international conference on intelligent environments*, pages 140–143. IEEE, 2015. 16

[54] Cancan Song, Zhiyan Zhou, Ying Zang, Lingli Zhao, Wenwu Yang, Xiwen Luo, Rui Jiang, Rui Ming, Yu Zang, Le Zi, et al. Variable-rate control system for UAV-based granular fertilizer spreader. *Computers and Electronics in Agriculture*, 180:105832, 2021. 16

[55] Kate J Yaxley, Nathan McIntyre, Jayden Park, and Jack Healey. Sky shepherds: A tale of a UAV and sheep. *Shepherding UxVs for Human-Swarm Teaming: An Artificial Intelligence Approach to Unmanned X Vehicles*, pages 189–206, 2021. 16

[56] Tom M Mitchell and Tom M Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997. 16, 17

[57] Taiwo Oladipupo Ayodele. Machine learning overview. *New Advances in Machine Learning*, 2:9–18, 2010. 16

[58] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised learning. In *Machine learning techniques for multimedia*, pages 21–49. Springer, 2008. 16

[59] Horace B Barlow. Unsupervised learning. *Neural computation*, 1(3):295–311, 1989. 16

[60] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996. 17

[61] Zhi-Hua Zhou. *Machine learning*. Springer Nature, 2021. 17

[62] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938, 2018. 17

[63] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014. 17, 18

[64] Russell C Eberhart. *Neural network PC tools: a practical guide*. Academic Press, 2014. 18

[65] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*, 2017. 18

[66] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015. 19

[67] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern recognition*, 77:354–377, 2018. 19

[68] Panqu Wang, Pengfei Chen, Ye Yuan, Ding Liu, Zehua Huang, Xiaodi Hou, and Garrison Cottrell. Understanding convolution for semantic segmentation. In *2018 IEEE winter conference on applications of computer vision (WACV)*, pages 1451–1460. Ieee, 2018. 20

[69] Jeremiah W Johnson. Adapting mask-rcnn for automatic nucleus segmentation. *arXiv preprint arXiv:1805.00500*, 2018. 21

[70] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 21

[71] Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM (JACM)*, 9(1):61–63, 1962. 25

[72] Silvano Martello and Maria Grazia Speranza. *Ricerca operativa per l'economia e l'impresa*. Società Editrice Esculapio, 2022. 25

[73] Christian Nilsson. Heuristics for the traveling salesman problem. *Linkoping University*, 38:00085–9, 2003. 26

[74] Howie Choset and Philippe Pignon. Coverage path planning: The boustrophedon cellular decomposition. In *Field and service robotics*, pages 203–209. Springer, 1998. 27, 29

[75] Heinz Breu, Joseph Gil, David Kirkpatrick, and Michael Werman. Linear time euclidean distance transform algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(5):529–533, 1995. 28

[76] G. Roggi, A. Niccolai, F. Grimaccia, and M. Lovera. A computer vision line-tracking algorithm for automatic UAV photovoltaic plants monitoring applications. *Energies*, 13(4), 2020. 30

[77] Robert Thomas Jones. *Classical aerodynamic theory*. Number 1050 in 1. National Aeronautics and Space Administration, 1979. 30

[78] F. Grimaccia, S. Leva, and A. Niccolai. PV plant digital mapping for modules' defects detection by unmanned aerial vehicles. *IET Renewable Power Generation*, 11(10):1221–1228, 2017. 34

[79] Dongqing Li. *Encyclopedia of microfluidics and nanofluidics*. Springer Science & Business Media, 2008. 35

[80] Lijun Ding and Ardeshir Goshtasby. On the canny edge detector. *Pattern recognition*, 34(3):721–725, 2001. 35

[81] John Illingworth and Josef Kittler. A survey of the hough transform. *Computer vision, graphics, and image processing*, 44(1):87–116, 1988. 35

[82] M. Aghaei, S. Leva, and F. Grimaccia. PV power plant inspection by image mosaicing techniques for ir real-time images. In *Photovoltaic Specialists Conference*, volume 2016-November, pages 3100–3105, 2016. 40, 41

[83] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003. 41, 91

[84] Álvaro Huerta Herraiz, Alberto Pliego Marugán, and Fausto Pedro García Márquez. Photovoltaic plant condition monitoring using thermal images analysis by convolutional neural network-based structure. *Renewable Energy*, 153:334–348, 2020. 41

[85] X. Li, Q. Yang, J. Wang, Z. Chen, and W. Yan. Intelligent fault pattern recognition of aerial photovoltaic module images based on deep learning technique. In *International Multi-Conference on Complexity, Informatics and Cybernetics*, volume 1, pages 22–27, 2018. 41

[86] shaikh Aslam. OptiTrack Motive. https://optitrack.com/software/motive/, 2023. 46, 99

[87] shaikh Aslam. ros-drivers/mocap_optitrack. https://github.com/ros-drivers/mocap_optitrack, 2023. 46, 99

[88] Eric Rohmer, Surya P. N. Singh, and Marc Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326, 2013. 51

[89] Noel Gorelick, Matt Hancher, Mike Dixon, Simon Ilyushchenko, David Thau, and Rebecca Moore. Google earth engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment*, 2017. 60

[90] Xue Yang and Junchi Yan. Arbitrary-oriented object detection with circular smooth label. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VIII 16*, pages 677–694. Springer, 2020. 62

[91] Jiang Hou, Yao Ling, and Liu Yujun. Multi-resolution dataset for photovoltaic panel segmentation from satellite and aerial imagery, 2021. 63

[92] B. Dwyer, J. Nelson, J. Solawetz, and et al. Roboflow (version 1.0) [software]. https://roboflow.com, 2022. computer vision. 63, 66, 83

[93] Akasapu Hemanthika. Detectron2 FPN + PointRend model for amazing satellite image segmentation, 2023. Medium. 64, 102

[94] Robin M. Cole. satellite-image-deep-learning, 2023. 64

[95] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023. 66, 84

[96] IBM ILOG CPLEX. V12. 1: User's manual for CPLEX. *International Business Machines Corporation*, 46(53):157, 2009. 74

[97] C. E. Miller, A. W. Tucker, and L. R. Zemlin. Integer programming formulations and traveling salesman problems. *Journal of the ACM*, 7:326–329, 1960. 75

[98] Fillipe Gonçalves. python-TSP: Python library for travelling salesman problem. https://github.com/fillipe-gsm/python-tsp, 2023. 79

[99] Alan SI Zinober. *Variable structure and Lyapunov control*, volume 193. Springer, 1994. 89

[100] Alexandre S Brandao, Felipe N Martins, and Higor B Soneguetti. A vision-based line following strategy for an autonomous UAV. In *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, volume 2, pages 314–319. IEEE, 2015. 89

[101] Sun Jianping. An optimum layout scheme for photovoltaic cell arrays using PVSYST. In *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, pages 243–245. IEEE, 2011. 97

[102] Luigi Biagiotti and Claudio Melchiorri. *Trajectory planning for automatic machines and robots*. Springer Science & Business Media, 2008. 99

[103] Jan Hosang, Rodrigo Benenson, Piotr Dollár, and Bernt Schiele. What makes for effective detection proposals? *IEEE transactions on pattern analysis and machine intelligence*, 38(4):814–830, 2015. 102

[104] Vishakha Vijay Patel. Ziegler-nichols tuning method: Understanding the pid controller. *Resonance*, 25(10):1385–1397, 2020. 110

[105] Neha Bagwari, Sushil Kumar, and Vivek Singh Verma. A comprehensive review on segmentation techniques for satellite images. *Archives of Computational Methods in Engineering*, pages 1–34, 2023. 129

[106] Gonghe Shi and Baohe Zuo. Cloudrcnn: a framework based on deep neural networks for semantic segmentation of satellite cloud images. *Applied Sciences*, 12(11):5370, 2022. 129

[107] Luca Morando, Carmine Tommaso Recchiuto, and Antonio Sgorbissa. Social drone sharing to increase the UAV patrolling autonomy in emergency scenarios. In *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 539–546. IEEE, 2020. 130

[108] Daniel V Ruiz, Leonardo A Vidal, and Eduardo Todt. For the thrill of it all: A bridge among linux, robot operating system, android and unmanned aerial vehicles. *arXiv preprint arXiv:2006.11656*, 2020. 133