Tesis Doctoral
Ingeniería de Automática, Electrónica y de Telecomunicación

Visual Perception System for Aerial Manipulation: Methods and Implementations

Autor: Pablo Ramón Soria
Director: B.C. Arrue y A. Ollero

Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Tesis Doctoral
Ingeniería de Automática, Electrónica y de Telecomunicación

Visual Perception System for Aerial Manipulation: Methods and Implementations

Autor:

Pablo Ramón Soria

Director:

B.C. Arrue y A. Ollero

Profesor Titular

Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

2018

Tesis Doctoral: Visual Perception System for Aerial Manipulation: Methods and Implementations

Autor: Pablo Ramón Soria
Director: B.C. ARRUE y A. Ollero

El tribunal nombrado para juzgar la Tesis arriba indicada, compuesto por los siguientes doctores:

Presidente:

Vocales:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

A mi familia
A mis profesores

Acknowledgements

First of all, I have much to thank to my supervisors Begoña C. Arrue and Anibal Ollero for all the support, time and patience that I received from them. I appreciate the support and trust that Begoña put on me. She encouraged me to start this project and helped me to achieve it.

During my Phd I had the opportunity to do a research internship in the Australian Centre for Field Robotics (ACFR) in Sydney. I want to thank Prof. Robert Fitch who guided my research during the stay. Also, I really appreciate the time and effort that Wolfram Martens dedicated to me. He taught me to be more strict in mathematical terms in my research and introduced me in the probabilistic world of Gaussian processes.

Also, I would like to thank to all colleagues in the Group of Robotics, Vision and Control (GRVC) who I worked with over these years. Particularly, I appreciated the time spent with Manuel Perez, Ricardo Lopez, Javier Rubiales and Alejandro Gomez. It has been really amusing developing with them and discussing my work. They helped me a lot carrying out the experiments and had an endless patience.

I must make an special mention to my family. They have provided me their support and time day by day. I want to thank my parent because he guided me in my career as engineer since I can remember. And also to my mother for her unconditional support every day, whatever happen during life. At last but not at least, I sincerely want to thank Ana, my love, for faithfully supporting me during this long journey.

Pablo Ramón Soria
December, 2018

Abstract

Technology is growing fast, and autonomous systems are becoming a reality. Companies are increasingly demanding robotized solutions to improve the efficiency of their operations. It is also the case for aerial robots. Their unique capability of moving freely in the space makes them suitable for many tasks that are tedious and even dangerous for human operators.

Nowadays, the vast amount of sensors and commercial drones makes them highly appealing. However, it is still required a strong manual effort to customize the existing solutions to each particular task due to the number of possible environments, robot designs and missions. Different vision algorithms, hardware devices and sensor setups are usually designed by researchers to tackle specific tasks. Currently, aerial manipulation is being intensively studied to allow aerial robots to extend the number of applications. These could be inspection, maintenance, or even operating valves or other machines.

This thesis presents an aerial manipulation system and a set of perception algorithms for the automation aerial manipulation tasks. The complete design of the system is presented and modular frameworks are shown to facilitate the development of these kind of operations.

At first, the research about object analysis for manipulation and grasp planning considering different object models is presented. Depend on the model of the objects, different state of art grasping analysis are reviewed and planning algorithms for both single and dual manipulators are shown.

Secondly, the development of perception algorithms for object detection and pose estimation are presented. These allow the system to identify many kind of objects in any scene and locate them to perform manipulation tasks. These algorithms produce the necessary information for the manipulation analysis described in the previous paragraph.

Thirdly, it is presented how to use vision to localize the robot in the environment. At the same time, local maps are created which can be beneficial for the manipulation tasks. These maps are are enhanced with semantic information from the perception algorithm mentioned above.

At last, the thesis presents the development of the hardware of the aerial platform which includes the lightweight manipulators and the invention of a novel tool that allows the aerial robot to operate in contact with static objects.

All the techniques presented in this thesis have been validated throughout extensive experimentation with real aerial robotic platforms.

Contents

Notation

$\mathbb{R}$	Real numbers
$\mathscr{O}$	three dimensional object
$\mathscr{S}$	Surface of object $\mathscr{O}$
$\mathbb{S}$	Approximate surface of $\mathscr{S}$
$\mathscr{P}$	Polytope of surface $\mathscr{S}$
$V = \{v_i\}$	Vertices of polytope $\mathscr{P}$
$E = \{e_l\}$	Edges of polytope $\mathscr{P}$
$F = \{f_m\}$	Faces of polytope $\mathscr{P}$
GP	Gaussian Process
$GPIS$	Gaussian Process Implicit Surface
$f \sim GP(m(\cdot), K(\cdot,\cdot))$	Joint distribution of variables driven by a Gaussian Process with mean $m(\cdot)$ and covariance $K(\cdot,\cdot)$
$\mathscr{D} = \{(x_i, y_i)$	Set of data points in 3D with associated scalar values y_i
$exp(\cdot)$	Exponential function
$sin(\cdot)$	Sinusoidal function
$cos(\cdot)$	Cosine function
ε	Error associated to a process
$x \sim N(0, \sigma^2)$	Variable distributed with a Gaussian or Normal distribution
μ	Mean of Gaussian distribution
σ	Variance of a Gaussian distribution
$\mathscr{D}^+ = \{(x_i, y_i)\}$	Extended set of data points in three dimensions with associated scalar values and normal vector direction
∇	Derivative term
K	Covariance matrix of data points in GP
K^{**}	Covariance matrix of query points in GP
K^{*}	Cross-covariance matrix between data points and query points in GP
$\frac{\partial f(\mathbf{x})}{\partial y}$	Partial derivative of x repect y
$diag$	Diagonal matrix

$min\_dist(A,B)$	Minimum distance between set of points A and B
F_i	Three dimensional normal force
τ_i	Three dimensional torque force
ω_i	Six dimensional vector representing a force wrench composed by F_i and τ_i
$S(t)$	Cross-product matrix
$\mathscr{R}$	Multi-link Serial Robot
$\Theta = \theta_1, ..., \theta_n$	Set of scalar variables describing joints of Serial Robot $\mathscr{R}$
$J(X)$	Jacobian matrix of X
Δ	Increment
$\angle(x,y)$	Angle between x and y
$\sum_{i=1...N_k} x_i$	Sum of all x_i variables
$\prod_N^{n=1} x_i$	Product over all x_i variables
$a \in A$	a belongs to A
$P(x\|y)$	Conditional probability of x given y
$P(x=a)$	Probability of $x=a$
$\{w_1, w_2, ..., w_i, ..., w_N\}$	Set of words defining a document $\mathbb{W}$
$\mathbb{D} = \{W_1, W_2, ..., W_M\}$	Corpus or set of documents
z	Latent topic
$z = (1, ..., K)$	Set of latent topics
θ	Variational parameter of the distribution of the latent variables
γ	Latent variables of the Latent Dirichlet Distribution
ϕ	Variational parameter of the distribution of the topics
x_k^i	state variable
$\dot{x}_k^i$	first derivative of state variable
$\ddot{x}_k^i$	second derivative of state variable
DF_i	Data-Frame
CF_i	Cluster-Frame
χ^2	Chi square probability distribution

Acronyms

2D	Two Dimensional (or dimensions)
3D	Three Dimensional (or dimensions)
3DSC	3D Shape Context
ACFR	Australian Centre for Field Robotics
AEROARMS	AErial RObotic system integrating multiple ARMS and advanced manipulation capabilities for inspection and maintenance
BA	Bundle Adjustment
ANN	Artificial Neuronal Network
BOW	Bag Of Words
BRIEF	Binary Robust Independent Elementary Features
CAD	Computer-Aided Drafting
CNN	Convolutional Neuronal Network
CRP	Chinese Restaurant Process
CUDA	Compute Unified Device Architecture
CWS	Cone Wrench Space
DH	Denavit Hartenberg
DL	Deep Learning
DLS	Damped Least Squares
DOF	Degrees Of Freedom
DP	Dirichlet Process
DPMM	Dirichlet Process Mixture Model
EKF	Extended Kalman Filter
FAST	Features from accelerated segment test
FLANN	Fast Library for Approximate Nearest Neighbors
FPFH	Fast Point Feature Histograms
FPS	Frames Per Second
F-RCNN	Fast(er) Region-Based Convolutional Neuronal Network
gICP	Generalized Iterative Closest Points

GPS	Global Positioning System
GP	Gaussian Process
GPIS	Gaussian Process Implicit Surface
GPU	Graphics Processing Unit
GRVC	Group of Robotics Vision and Control
GWS	Grasp Wrench Space
HSV	Hue Saturation Value
IBVS	Image-Based Visual Servoing
ICP	Iterative Closest Points
IK	Inverse Kinematic
IR	InfraRed
IMU	Inertial Measurement Unit
KNN	K-Nearest Neighbor
LDA	Latent Dirichlet Allocation
LM	Levenberg Marquardt
MCMC	Markov Chain Monte Carlo
MH	Metropolis Hastings
NARF	Normal Aligned Radial Feature
ORB	Oriented FAST and rotated BRIEF
OWS	Object Wrench Space
PBVS	Position-Based Visual Servoing
PCA	Principal Components Analysis
PCL	Point Cloud Library
PFH	Fast Point Feature Histograms
PWM	Pulse-Width Modulation
RANSAC	Random Sample Consensus
RBF	Radial Basis Function
RCNN	Region-Based Convolutional Neuronal Network
RGB	Red Green Blue
RGB-D	Red Green Blue Depth
ROS	Robot Operating System
SBA	Sparse Bundle Adjustment
SHOT	Signature of Histograms of Orientations
SIFT	Scale Invariant Feature Transform
SURF	Speed-Up Robust Features
SLAM	Simultaneous Localization and Mapping
SSD	Single Shot MultiBox Detector
SVM	Support Vector Machine
TCVF	Temporal Convolution Voxel Filtering
TWS	Task Wrench Space
UAV	Unmanned Aerial Vehicle
YOLO	You Only Look Once

1 Introduction

1.1 Motivation and Objectives

Most living beings use vision to perform even the most insignificant task. Finding theirselves in the environment, recognizing places, moving or even grasping objects would be impossible without vision.

For a long time, scientists and researchers have paid special attention to develop methods and algorithms that, somehow, replicate the capabilities of living beings in term of perception. These algorithms are targeted to be applied to machines, allowing them to perceive to accomplish different tasks in an autonomous manner. These machines are called robots. Autonomous robots require for a perfect symbiosis between the hardware and the software, and the synchrony of a variety of algorithms to, among several things, perceive the environment, control its actuators and interact with the environment.

The motivation of this dissertation is to develop the necessary vision algorithms to allow robots to perform manipulation tasks. Particularly, the work developed in this research focuses on aerial robots, i.e., robots that fly. Researching with this kind of robots has many requirements and specifications. Some of them are the same that with any other kind of robot. However, aerial robots usually suffer from more and stricter constraints, and require more robust algorithms due to the fragility of these platforms and their minimal payload capabilities.

Nonetheless, aerial robots, so-called drones or Unmanned Aerial Vehicles (UAVs), are suitable for a wide range of applications which take place in inaccessible locations and are typically dangerous to human operators such as power line inspections [1, 2], wind-turbine maintenance [3], inspection of different structures in facilities [4] or photogrammetry [5]. In the later years, they have been prove to be useful for manipulation tasks too[6].

In order to perform these tasks, it is necessary to provide the robot with various capabilities such as localization, planning, and, in general, smart capabilities. It also requires to equip them with the appropriate hardware to perform the desired tasks. The purpose of introducing robots in these applications is to provide a more efficient solution to these problems, to reduce the costs (both time and monetary) and definitely reduce the decay on the quality due to faults in human operators caused by the monotony of some of these

tasks. Instead, the operator will be placed in a more important role which can not be done by robots.

The advances in mechatronics and the continuous rise of smaller electronic devices have favored the rise of applications with these aerial platforms. Particularly, the development of small electronic devices such as embedded computers and lightweight sensors have paid an essential role in this topic.

Multirotors platforms are common platforms in the field of aerial robotics. Some of their beneficial capabilities are their reduced cost compared with other similar solutions such as helicopters, their ability to move freely in the space and hover closer to the desired location, and their scalability in size and quantity.

However, these platforms are not exempted of challenges. One of the main limitations of these platforms is their endurance. The maximum time flight is highly dependent on the platform's configuration and its payload. The development of new lithium polymer battery technology has been utterly important improving this endurance. However, this fact is still a strong limitation for many applications.

In recent years, a new trend in aerial research has appeared: the aerial manipulation. Aerial manipulation is based on the concept that installing a manipulator on an aerial robot expands the number of applications that these platforms can perform. In the beginning, applications involving UAVs were merely perceptive. Cameras and computers can be easily set up in the robots, so they can explore areas and process images on board or just relay them to a ground computer. Now, the concept is to enable UAVs to interact with the environment, so the robot can, not only detect or perceive something but also, perform actions.

This thesis focus on providing these aerial robots for the vision capabilities to perform manipulation tasks. Aerial manipulation is a very challenging area of research. Aerial robots are delicate and have strong payload limitation which implies that algorithms must be as much light as possible and effective to prevent the robot crashing. Figure 1.1 shows two platforms developed by the Group of Robotics, Vision, and Control of the University of Seville under the framework of AEROARMS European Project [7]. Particularly, the platform on the left has been designed and built during the development of this thesis.

Manipulation implies to interact with physical objects using manipulators. This can be, for example, grasping objects, moving them, or just keeping in contact with some stiff object. In order to do so, the robot needs to be able to detect, locate and analyze how to interact with the objects. Additionally, in order to perform this tasks, it is necessary to plan the actions and movements of the manipulators and to prevent them from colliding with the environment and the platform itself.

As mentioned before, this thesis focus on the development of an aerial platform with manipulation capabilities. For that purpose, a variety of vision algorithms has been developed and applied in real aerial platforms developed by the author.

In order to perform a manipulation task, many subtasks are needed. At first instance, it is required to detect the object to be manipulated. Then, it is necessary to have an estimation of the position of the objects relative to the robot. At the same time, robots' manipulators or arms are required to be located to manipulate the object. A principal aspect that is going to be discussed in this dissertation is how objects can be modeled. These models are used in several aspects of manipulation such as object detection or grasp planning.

Figure 1.1 Aerial manipulators developed by the GRVC team.

Additionally to the skill of detecting and recognizing objects in the environment, making the robot able to locate itself in the space, detecting the physical world that surrounds it, is what makes it possible to move and to plan reasonable movements. Moreover, detecting objects and placing them into the space makes it possible to interpret the surrounding and influences on how to move from one place to another.

The capability of self-location is utterly required in robotics to allow fully autonomous development of tasks. This problem receives the name of Simultaneous Localization and Mapping (SLAM) and has been very popular in mobile robotics over the last decades. Global Localization systems offer a partial solution to this problem. However, the low accuracy rates of traditional GPS solutions, the sensitivity of these systems to atmospheric conditions and the disability of working indoor due to the interference of the infrastructures have nefarious effects on this localization system. For this reason, many researching efforts have been paid to provide robots with other methods for localization using a broad spectrum of sensors. Most common sensors are visual sensors or camera-based sensors. These provide color images which can be used in similitude with human eyes to compute relative information about the localization of the platform. Nowadays, many vehicles use laser-based sensors. These devices are of high accuracy and have been used effectively, for example, in autonomous navigation of cars. However, the price of these sensors is extraordinarily high, and their weight is not affordable for all the platforms.

All the mentioned capabilities are necessary to move to the next generation of autonomous robots. Each of them, fulfill a specific requirement without which the operation cannot be granted. This dissertation will cover some of these aspects as will be described later in this section. Special consideration needs to be made when working with UAVs. As aforementioned, these platforms have strong limitations regarding design and up-lift weight. It is because the aerial robot's flight in an unstable equilibrium in the air balancing

the gravity forces with the multiple rotors. Contrary to helicopters, small propellers of multirotors are highly less efficient, so the trade-off between the thrust and the energy consumption is more compromised. Nevertheless, the multi-rotor design overweight the mono-propeller design concerning maneuverability, flexibility and fault tolerant.

Figure 1.2 shows some of the existing concept models of aerial manipulation platforms. Each of these platforms has been designed to accomplish different manipulation tasks.

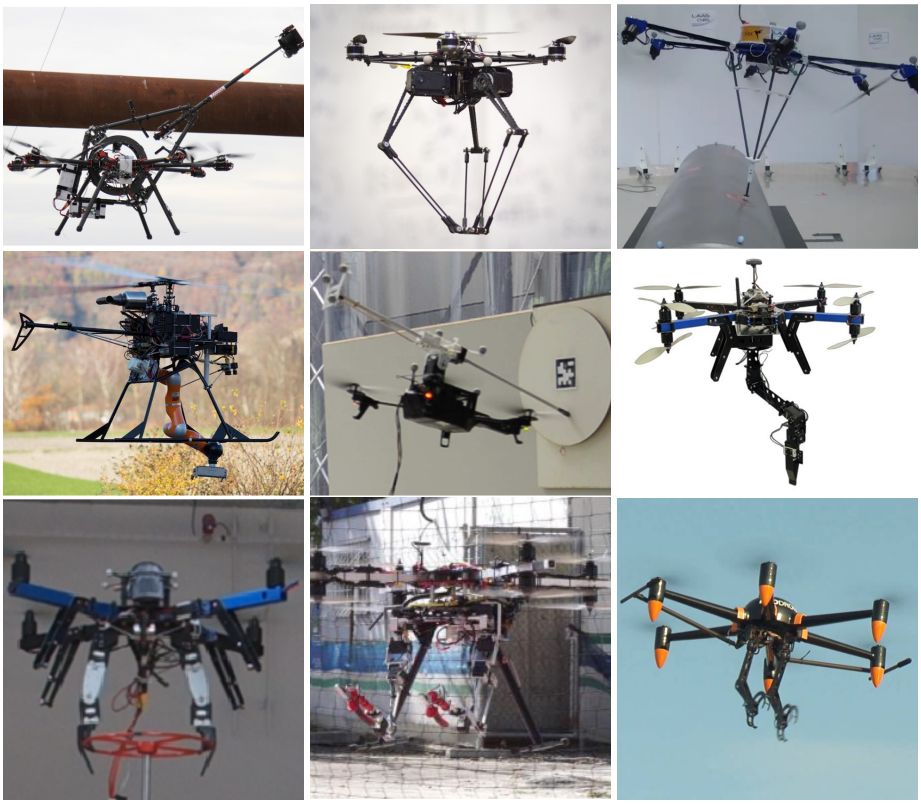


Figure 1.2 Examples of aerial platforms with manipulators.

Nevertheless, payload limitation influences, not only the design of the aerial platforms but also the computing hardware. On board computers are required to be smaller and less power-drainer due to the limitations in space and battery consumption. For these reasons, the design of the algorithms and software implementation has inherited limitations regarding memory usage, power resources, and power consumption.

Therefore, this thesis has the following objectives:

- To analyze different methodologies for modeling objects concerning the visual capabilities of the aerial robot to perform manipulation tasks. It includes analyzing the types of data that on board sensors can acquire and to elaborate methods to define a virtual abstraction of the object to be manipulated by the robot.

- To implement and compare algorithms to evaluate and generate more efficient ways to carry out manipulation tasks to grasp different objects. These algorithms should take into consideration the models, mentioned in the previous objective, concerning the visual sensors.
- To design and develop computer vision algorithms that allow the aerial system to detect and locate objects in its environment. These algorithms should exploit the sensor capabilities and be robust to changes in illumination and appearance.
- To provide the aerial robot for a system to perceive the environment and its own location. Additionally, the system should be able then to locate individual instances of objects in the environment.
- Robustness and speed have to be explicitly considered in the proposed methods and algorithms, such that they can be used in real applications. The system should be applied to any object and environment without increasing the complexity of the algorithms. Thus they can be applied to any manipulation task.
- To design and develop the required hardware devices to test and validate the proposed methods and algorithms in real conditions. This includes the preparation of the aerial platforms, the embedded computers and electronics, and the arms. Special consideration will be paid to the manipulators to be lightweight and to have the necessary capabilities to perform the manipulation tasks.

During the development of the thesis, the author developed hardware for two aerial platforms. Additionally, a contact tool has been developed which docks to pipes to provide the relative position of the platform as an alternative to visual localization algorithms.

1.2 Thesis Framework

The core research of this dissertation has been developed under the framework of two projects. The European project AEROARMS (Project ID: 644271), or AErial RObotic system integrating multiple ARMS and advanced manipulation capabilities for inspection and maintenance, under the program H2020. And the national project ARM-EXTENDED (DPI2017-89790-R). The primary motivation of these projects is to develop a robotic aerial system with arms with advances manipulation capabilities.

Additionally, the author did an internship in the Australian Center for Field Robotics (ACFR), where he developed vision algorithms for manipulation in the context of fruit harvesting.

This goal is difficult to achieve because of the complexity of operating with UAVs. During the development of the thesis, various experiments and demonstrations have been shown under the Framework of AEROARMS project. These experiments were autonomous and semi-autonomous, and showed the perception and manipulation capabilities the aerial platforms developed and implemented during this thesis.

1.3 Thesis Outline

The thesis is organized in the following chapters:

- **Chapter 2** describes the methods and algorithms that have been explored and implemented during the thesis to model objects by the robot's perception system. These models are mathematical definitions of the objects. These are classified into deterministic and probabilistic models. Each specific model determines how sensor data is used to detect or manipulate the object. The chapter analyzes how the models are used to compute and evaluate the best way to manipulate them. The content of this chapter has been published in [8], [9] and [10].
- **Chapter 3** describes how to detect and locate different objects in the environment. This objective is critical to allow the robot to interact with the objects. The algorithms are ordered an classified according to object models aforementioned in Chapter 2. The methods exploit 2D and/or 3D information obtained from the sensors to achieve this objective. The chapter ends with an analysis of various object detection algorithms using machine learning techniques. The content of this chapter has been published in [11], [8], [9], [12],[13] and [14].
- **Chapter 4** Introduces two different methods for simultaneous localization and mapping for UAVs applications. The first method is based on sparse feature clouds obtained from a low-cost stereo camera. This algorithm fuses the visual information together with the Inertial Measurement Unit (IMU) of the aerial robot to elaborate a local map. The second method is based on depth-sensing cameras and proposes a flexible framework for simultaneous localization and mapping. The framework has been evaluated in different standard datasets, and two cases of uses are shown: mapping of the pillars of a bridge for future inspection purposes, and an object-based semantic SLAM using the deep learning techniques used in Chapter 3. The content of this chapter has been published in [11].
- **Chapter 5** exposes the platforms that have been developed during the course of the research. This chapter is important as thanks to the development of these platforms, the author was able to test in real conditions the developed methods and algorithms. Additionally, a novel end-effector which is integrated into the manipulator is described too. This tool was conceived as an alternative to the vision algorithms to obtain the relative position of the platform. The content of this chapter has been published in [11], [8] and [15].
- **Chapter 6** Summarizes the conclusions of the research developed by the author and discusses the research lines for the future work.

Figure 1.3 shows visually the structure of the thesis and the influences between the chapters.

The contributions in this dissertation are not only focused in advancing state of the art methodologies but also implementing them in real-world applications in UAVs for advanced manipulation tasks. As it is well-known, validating algorithms in physical setups requires extra efforts in comparison with simulation environments or single runs

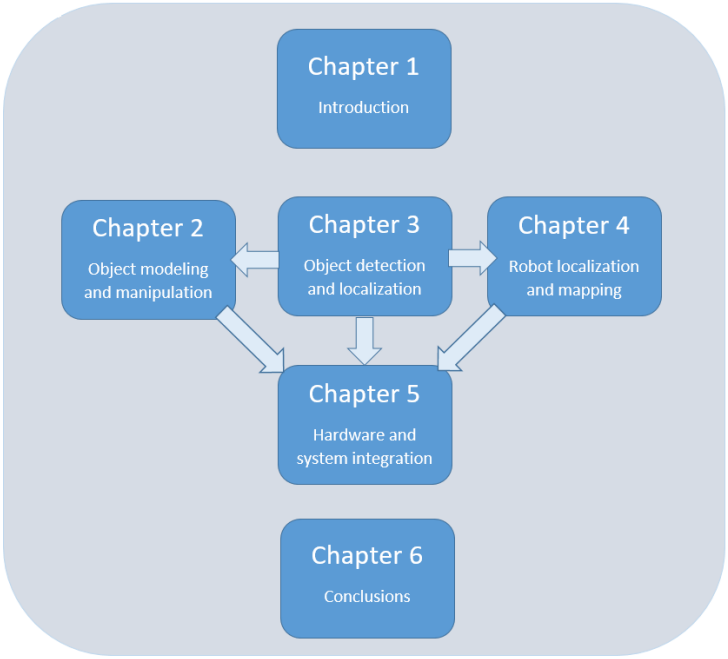


Figure 1.3 Thesis outline.

of datasets. The purpose is to provide as much as possible a ready to use platform with current technology to potential transfer to industry.

It is worth to mention that all the algorithms and implementations has been validated with real platforms and real data which made the research more challenging.

1.4 Contributions

The main contributions of this dissertation are the development of:

- Computer vision algorithms for object detection and location for aerial manipulation.
- Visual aided object analysis for object manipulation using manipulators.
- Computer vision framework for robot localization and mapping of the environment
- Development of two aerial platforms with manipulation capabilities and a variety of end-effectors.

These contributions are aligned with the motivation and goals of the thesis. The dissertation summarizes the following publications:

1. Ramon Soria, P.; Bevec, R.; Arrue, B.C.; Ude, A.; Ollero, A. "Extracting Objects for Aerial Manipulation on UAVs Using Low Cost Stereo Sensors". Sensors 2016, 16, 700.

2. Ramon Soria, P.; Arrue, B.C.; Ollero, A. "Detection, Location and Grasping Objects Using a Stereo Sensor on UAV in Outdoor Environments". Sensors 2017, 17, 103.
3. P. Ramon Soria, B.C. Arrue and A. Ollero, "A 3D-Printable Docking System for Aerial Robots: Controlling Aerial Manipulators in Outdoor Industrial Applications," in IEEE Robotics & Automation Magazine (RAM). doi: 10.1109/MRA.2018.2884744
4. Ramon Soria, P., B.C. Arrue, Anibal Ollero. Grasp Planning and Visual Servoing for Aerial Dual Manipulator outdoors - (Under submission). Elsevier - Engineering. 2018.
5. W. Martens, Y. Poffet, P. R. Soria, R. Fitch, and S. Sukkarieh, "Geometric Priors for Gaussian Process Implicit Surfaces," in IEEE Robotics and Automation Letters, vol. 2, no. 2, pp. 373-380, April 2017.
6. Ramon Soria P., Sukkar F., Martens W., Arrue B.C., Fitch R. "Multi-view Probabilistic Segmentation of Pome Fruit with a Low-Cost RGB-D Camera". ROBOT 2017: Third Iberian Robotics Conference. ROBOT 2017. Advances in Intelligent Systems and Computing, vol 694. Springer, Cham
7. Prada Delgado J., Ramon Soria P., Arrue B.C., Ollero A. "Bridge Mapping for Inspection Using an UAV Assisted by a Total Station". ROBOT 2017: Third Iberian Robotics Conference. ROBOT 2017. Advances in Intelligent Systems and Computing, vol 694. Springer, Cham
8. A. Suarez, P. R. Soria, G. Heredia, B. C. Arrue and A. Ollero, "Anthropomorphic, compliant and lightweight dual arm system for aerial manipulation," 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, 2017, pp. 992-997.
9. Pablo Ramón Soria and Begoña C. Arrue. "Aerial robotics manipulation" – Chapter 4.7 "Object Detection and Probabilistic Object Representation for Grasping with Two Arms". Editors: Anibal Ollero and Bruno Siciliano. Springer Tracts on Advanced Robotics – 2018.

Additionally, during the course of this thesis, fruitful collaborations have led to many other publications which are not discussed in this dissertation but are also related with the development of vision or perceptive algorithms, or related with aerial platforms:

1. Castaño ÁR, Real F, Ramón-Soria P, et al. "Al-Robotics team: A cooperative multi-unmanned aerial vehicle approach for the Mohamed Bin Zayed International Robotic Challenge". J Field Robotics. 2018;1–21.
2. Fran Real, Arturo Torres-Gonzalez, Pablo Ramón Soria, Jesús Capitán and Aníbal Ollero. "UAL: an abstraction layer for unmanned vehicles". International Symposium on Aerial Robotics (ISAR). June 12th. 2018. Philadelphia, USA. 2018.
3. P. R. Soria, A. F. Palomino, B. C. Arrue and A. Ollero, "Bluetooth network for micro-uavs for communication network and embedded range only localization," 2017 International Conference on Unmanned Aircraft Systems (ICUAS), Miami, FL, USA, 2017, pp. 747-752.

4. P. Ramon Soria, Begoña C. Arrue, Jose Joaquin Acevedo, Anibal Ollero. "Visual surveillance system with multi-UAVs under communication constraints". Robot 2015: Second Iberian Robotics Conference. December 2015
5. J. M. Aguilar, Pablo Ramon Soria, B.C. Arrue, A. Ollero. " Cooperative Perimeter Surveillance Using Bluetooth Framework Under Communication Constraints" Advances in Intelligent Systems and Computing book series (AISC, volume 694)
6. Pedro J. Sanchez-Cuevas, Pablo Ramon-Soria, Begoña Arrue, Anibal Ollero, Guillermo Heredia. " Robotic System for Inspection by Contact of Bridge Beams Using UAVs". Sensors 18, no. 12. Bridge Structural Health Monitoring and Damage Identification.
7. José Joaquín Acevedo, Manuel García, Antidio Viguria, Pablo Ramón, Begoña C. Arrue, Anibal Ollero. "Autonomous Landing of a Multicopter on a Moving Platform Based on Vision Techniques". Advances in Intelligent Systems and Computing book series (AISC, volume 694)

2 Object manipulation analysis for aerial robots

2.1 Introduction

The use of aerial vehicles for manipulation tasks requires the development of efficient and reliable capabilities. Many robot operations typically consist of just perceiving their environment. However, in order to move to the next generation of automated missions with robots, it is vital to provide them with actuation capabilities with the environment. Manipulation of objects is a challenging task that needs the robot to detect, track and analyze these objects. This Chapter is devoted to perception methods and algorithms that allow the robot to model and interact with movable objects.

The chapter is structured as follow. Section 2.2 introduces four different methods to model objects concerning the input data and the mathematical definition. The selection of the object model influences how these are treated later in the manipulation analysis. Section 2.3 describes algorithms that analyze how the objects should be manipulated taking into account the object and the robot configuration. Finally, Section 2.4 exposes how previous information is used to grasp using dual serial manipulators. All the methods exposed in this chapter has been implemented an integrated in an Open-Source library described in Appendix A.2.

2.2 Object modeling

Manipulation with robots has been studied extensively and tackled in different ways. This section describes how objects can be modeled concerning the visual information from sensors. The selection of this model influences the way objects are analyzed for their manipulation. During the development of this thesis, many models have been studied. Each of these models has particular advantages and disadvantages which will be discussed during the rest of this section.

There are many characteristics of objects that need to be taken into consideration. Nevertheless, it is common to simplify the objects with a representation of its envelope or surface. Let $\mathscr{O}$ be a generic three-dimensional object, and $\mathscr{S}$ the surface of object $\mathscr{O}$. The surface encloses the volume of the object, and it is its interface to be manipulated. Also, objects have mass and consequently weight. This is directly translated into inertial and gravity forces.

The interaction between manipulators and objects occurs through its interface, i.e., the surface. These interactions are contact points, and they need to be properly characterized to perform manipulation tasks. The model includes the position and orientation of the contact points which are mainly defined by the surface of the object. This position and orientation define the normal forces that are mutually exerted to and from the object and the hand. Additionally, the material of both, manipulator and object, determine the friction coefficient term. This coefficient is used to compute the magnitude of the tangential forces in soft contacts.

This section focuses on the mathematical modeling of object surfaces. These models are categorized into deterministic and probabilistic. Particularly, under the classification of deterministic models there can be found dense point cloud-based model, sparse feature based point cloud model, and mesh-based surfaces. These deterministic models do not handle uncertainty, but are very convenient due to their simplicity, and are widely used. For the probabilistic model, this chapter focus on the use of Gaussian Processes (GP) to handle uncertainty on objects' surfaces. This model has been widely used and has promising applications. The author of this thesis paid special attention to this probabilistic model and various publications have been written using it.

2.2.1 Point cloud-based object modeling

Point clouds might be the simplest model for objects surfaces. This representation is widespread as point clouds are the typical output data generated by range sensors such as depth cameras, stereo cameras, and laser devices. Many processing libraries, such as PCL [16], handle with this structure of data.

Lets denote $\mathbb{S} \in \mathscr{S}$ the approximation of object surface being,

$$\mathbb{S} = \{V = \{v_i\}, \quad i = 1...N \quad , \quad v_i \in \mathscr{S}\}$$

This is a list of points containing 3d information that is assumed to be on the surface of the object.

Nowadays several devices that acquire dense point clouds exist in the market. Let's classify them in three categories: passive stereo cameras, active stereo cameras, and mixed stereo cameras.

Passive stereo cameras have a calibrated baseline that allows computing the disparity map between the images that they capture. This disparity is then used to perform a pixel-wise triangulation of points which results in a point cloud. Additionally, as the depth is computed from color images of cameras, point clouds can be colored, providing them with additional information. This kind of cameras are usually less accurate than active stereo cameras but works well in almost any condition. However, they are based in the visible spectrum of light. Thus, their performance is poor in low light conditions.

The second kind of devices, active stereo cameras, are devices composed by an emitter, usually IR-light emitter, and a sensor sensitive to that emission. These kinds of cameras emit a structured pattern to the environment, so-called structural light. The pattern is then captured by the camera to be analyzed. The distortions in the pattern are used as features for computing the depth map. Contrary to passive stereo cameras, this devices cannot assign color to point clouds as typically IR cameras are grayscale. Nevertheless, these devices are usually equipped with a color camera that, once calibrated, is possible to project the color image onto the point cloud to paint it. These devices are usually more accurate than passive stereo devices. However, their main drawback is that IR-light is usually blinded with sunlight. So these devices have strong limitations for working outdoors.

Other modern devices use a combination of both systems, structural light, and stereo cameras to obtain the benefits of both systems.

During the development of this work, there have been tested many devices such as Intel real sense R200, SR300, D435; Microsoft Kinect v1 and v2; Artec EVA; StereoLab ZED and ZED mini; and other handmade stereo cameras using Logitech webcams. Figure 2.1 shows an example of typical data from this kind of sensors.

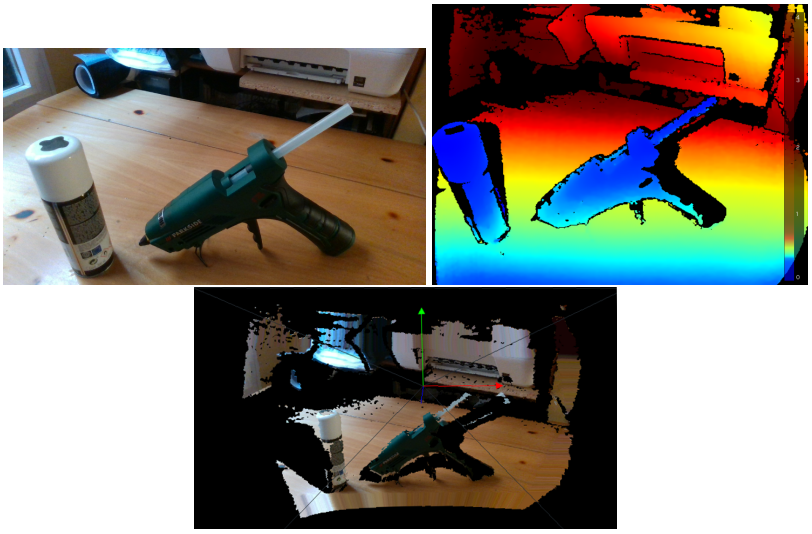


Figure 2.1 Example of data stream from an active stereo camera. From left to right it can be seen: the color image, the depth image resulting from the IR-emitter and the IR-sensor and eventually the constructed point cloud. This picture has been taken with an Intel RealSense SR300.

Point clouds are widely extended nowadays in robotics. They provide a quantitative representation of objects and the environment. Additionally, there exist a considerable number of algorithms that use point clouds to model objects or the environment. Many algorithms such as RANdom SAmple Consensus (RANSAC) [8, 17] or Iterative Closest Points (ICP) [18, 19] are used for model fitting and use directly point clouds obtained from

depth sensors. Thus there is no need to adapt the type of data. As an example, authors in [20] used point clouds to do shape fitting and track objects position for later generating grasps.

2.2.2 Sparse featured point clouds

The model introduced in the previous section is widely extended, and its geometrical representation is very intuitive. However, this representation is massive in terms of data. Sometimes, it is convenient to have 3D information but in a more compact way. Sparse point clouds contain scattered tridimensional information about the object or the environment, so they are less time-consuming. Additionally, each of the 3D points contained in the cloud usually has attributes or information that differentiates them.

Depending on the way this cloud is obtained, this additional data can be varied. This information is usually called descriptor, and each of the points is considered a feature. Features, or keypoints, are extensively studied in perception and can be obtained from both 2D images or 3D clouds. In the following paragraphs, a brief introduction of the features and descriptors is presented. Then, it is explained how these features can be used to model objects to perform the manipulation tasks.

Image features (or 2D features) have been widely used and studied. Features are computed in two steps. First, a feature detector algorithm chooses the set of stand out pixels in the images, typically corners or blobs. Then, a feature descriptor algorithm takes the local information of the pixel to generate a distinctive signature for that specific pixel. Stereo systems can use these 2D features to compute the disparity between the pair of images, and triangulate the 3D location of that specific points. As a consequence, the 3D point has both geometrical information and a distinctive signature that holds descriptive information about the vicinity of the point.

Several new feature detectors and descriptors have been developed over the last decades, to name a few SIFT [21], SURF [22], ORB [23], DAISY [24], BRIEF [25], FAST [26], shi-Tomasi [27]. These methods have been designed according to different objectives, as being faster or more robust. SIFT is a well-known detector and descriptor, and it has been proven to be robust to scales, rotations, and translations on images. However, it needs a high computational time in comparison with other features. Authors in [28, 29] proposed optimizations to speed it up but it is sometimes still not fast enough, or their approximation causes some losses of information.

Vision algorithms for UAVs have a strong requirement concerning speed as they need a faster response for the control loop than ground robots. For this reason, faster features such as ORB should be used.

Feature descriptors are specifically designed to describe points. This description is used to be able to keep track of those points in different images. The correlation between different features over a stream of images is commonly called feature matching. Each descriptor in one image can be compared with any other descriptor in a second image using a distance calculation such as Norm L2 or Hamming distance. The strategies to find the matching between two sets of descriptors is varied and starts from force brute algorithms to expanding trees or the well-known Fast Library for Approximate Nearest Neighbors (FLANN) [30].

Figure 2.2 shows an example of sparse cloud creation from a set of 2D images using ORB feature detector and descriptor. First three images show the matches during the sequence of images. Eventually, the last picture shows the point cloud obtained.

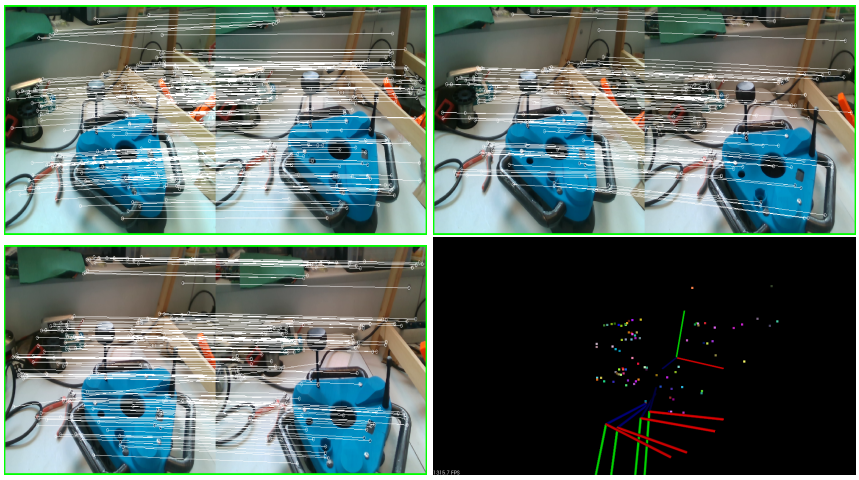


Figure 2.2 Example of 2D feature detection, description and matching. A sequence of images is used to compute an sparse featured cloud by the triangulation of features matched between the images..

With the rise of 3D point clouds, other kinds of descriptors have appeared. 3D detectors and descriptors use the local 3D information of dense point clouds to compute this distinctive description. Moving from dense point clouds representation to sparse representations can be highly beneficial. At first, it is intuitively less time consuming as the point cloud reduces its size drastically. Additionally, each of the points of the new sparse point cloud intrinsically collects the geometrical information about its vicinity.

3D keypoints have similar benefits to the 2D ones. Moreover, many of them are adaptations of bidimensional detectors. Several authors [31, 32, 33, 34] investigated the benefits and limitations of 3D keypoint detectors. Some of the well-known 3D detectors are Harris3D [35], which is a variation of the Harris corner detector. Similarly to the 2D case, it measures local changes in different directions applying different patches. Other detector is SIFT3D [36], that is a variant of SIFT. As its planar predecessor, it uses Hessian to define interest points. Finally, NARF (or Normal Aligned Radial Feature) [37], which locates features in areas where the surface is stable and the neighborhood contains large surface changes.

Some of the most used 3D descriptors are 3D Shape Context (3DSC) [38], this descriptor uses a 3D spherical grid which is divided into several portions. Then a weighted sum of points falling in each portion is used to generate the descriptor. Fast Point Feature Histograms (FPFH) [39] performs a multi-dimensional histogram with the information of each point. This variant is more robust than the previous version PFH due to the variety of density in local portions of the cloud. Signature of Histograms of Orientations

(SHOT) [40] first computes a local reference frame which is used to align the information of neighboring points. Then the local region is divided into volumes in which a histogram is computed. Finally, the descriptor is computed by concatenating the histograms. Authors in [41] proposed a variant of the SHOT descriptor which takes into account possible color information in the point clouds.

Similarly to bidimensional features, it is possible to perform a feature matching using the 3D descriptors. Figure 2.3 shows an example of feature matching using features computed from a dense point cloud. It can be clearly seen that this method overtake dense method for the alignment as they perform matching using a lower number of points and the matches go straight to the corresponding object. Top figures show the features computed in a cloud belonging to an object (top left figure) and features computed in a cloud of a scene (top right figure). The bottom figure shows the matches between them.

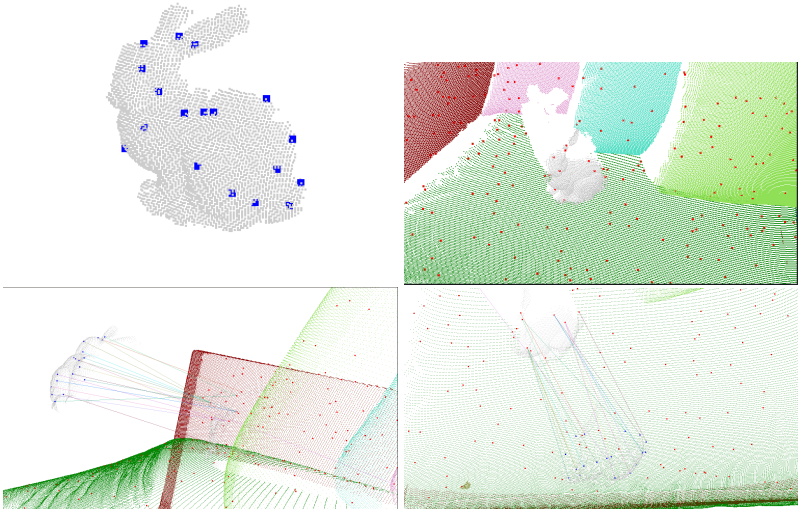


Figure 2.3 Example of 3D features in an scene with Stanford's Bunny in a couch. Figure a) shows the a point cloud of the bunny and a feature cloud computed using SIFT3D, then for each feature FPFH is computed. Similarly, Figure b) shows the same result for an environment. Figures c) and d) shows the matches between the features that can be used later for an alignment process.

The above 3D sparse representation has been extensively used to detect objects and compute their location. Gordon et al [42] uses a 3D representation of objects using sparse cloud computed with 2D SIFT features to recognize specific learned objects in new images and to use the 3D information to obtain their pose in the space relative to the camera. Tsai et al [43] proposed an algorithm that uses 3D features and descriptors to create descriptions of particular objects. Then, the same kind of signature is used in new shots of scenes captured using RGB-D cameras to seek for the object and to compute its location.

Particularly, the author of this thesis developed an algorithm for Aerial Manipulation that will be described in section 3.4 using this model and that was published in [8].

2.2.3 Mesh based object modeling

In this section, mesh models are introduced. Previous point-based models present a problem, which is that there is not a complete representation of the surface but an approximated representation using scattered points which are assumed to lie on the surface. However, these point clouds are still hollowed. Meshes offer an alternative in which points are connected with edges and faces. Meshes have been traditionally treated in mathematics as Polytopes $\mathscr{P}$ (N-dimensional geometrical objects). Moreover, this model is widely extended in the world of computer graphics (3D animation and computer games), thanks to which, meshes have been popularized, and there is much research in this area. Particularly, traditional GPU rendering pipelines use meshes as input streams. In this section, the mathematical definition of 3D meshes is briefly introduced and it is shown how this model is used to model objects.

In a nutshell, surfaces are continuous feasible 2D manifolds in $\mathbb{R}^3$, which represents the envelope of objects. Let $\mathscr{S}$ be a specific surface, and $\mathbb{S}$ a discrete approximation of the surface such that $\mathbb{S} \in \mathscr{S}$. This approximation is characterized by,

$$\mathbb{S} = \begin{cases} V = v_i & i = 1...N & \& & v_i \in \mathscr{S} \\ E = e_l = (v_i, v_j) & & \forall & i \neq j \\ F = f_m = (v_i, v_j, v_k) & & \forall & i \neq j \neq k \end{cases} \tag{2.1}$$

Each v_i is named a vertex, each e_l is an edge and f_m a facet (typically named a face in 3D). This discretization usually makes the problem more tractable than managing continuous functions in the space.

Meshes' complexity can be of any kind. Miller et al. [44] propose the use of a set of primitive shapes, such as spheres, cylinders, cones, or boxes to approximate objects. Each simplified shape has a set of predefined grasping strategies. This simplification makes the problem tractable, and it is particularly useful for a fast generation of grasps. An example of usage can be seen in Figure 2.4

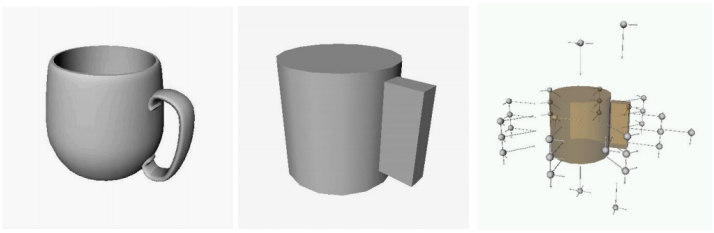


Figure 2.4 Example of grasps using shape primitives as meshed models. Resource from [44]. First image shows a mesh model of a mug, second model shows an approximation using a cylinder and a rectangle. Third picture shows some resulting approaching points for grasping the object..

In [45], authors propose an algorithm to extract objects from the scene which shape fits into primitive shapes for grasping them. However, this method needs the object to be simple itself and cannot be used in case of more complex shapes.

Other algorithms consider the use of any shape. However, it is important to take into consideration the convexity of the objects to check the feasibility of the grasps. Authors in [46] consider meshed objects that are non-convex. The algorithm proceeds using an approximate convex decomposition of the object for planning grasps.

Meshes are also a widespread model to represent objects surfaces. An advantage respect to point clouds is that faces area very compact representation. This can be interpreted as point cloud models with linear interpolations between sets of points, i.e., the faces. However, like previous models, it is deterministic and does not handle uncertainty intrinsically.

2.2.4 Probabilistic object modeling

This section introduces a new model for objects surfaces using a probabilistic representation called Gaussian Processes (GPs). Introducing this probabilistic model allows to incorporate uncertainty in the manipulation algorithms. This uncertainty can be, for example, caused by the errors in the visual detection algorithms, or even the uncertainty in the control of the manipulators.

At first, the mathematical definition of the model is presented. Then, the contributions of the author in this field are introduced, which are the use of geometrical priors for the inference of the surfaces, and the use of non-fixed variances in the data points for the inference. Additionally, all the methods and algorithms have been implemented an wrapped-up in the OpenSource library grasping_tools introduced in Appendix A.2.

Gaussian Processes for object modeling

Williams and Fitzgibbon in [47] propose an algorithm that uses Gaussian Processes for the reconstruction of 3D objects in a probabilistic framework denoted Gaussian Process Implicit Surfaces (GPIS). In this approach, the surface of the object is assigned to a level set of the regressed function using the data. This gives a probabilistic representation of the surface of the object in contrast with previous algorithms that use deterministic shapes. GPIS are also used in the literature for smoothing input data [48] or fusing data from different sensors [49, 50]. In [49, 50, 51] authors extend GPIS by integrating surface normals to improve the reconstruction of the object.

A Gaussian Process is a probabilistic model in which each of an arbitrarily large set of variables is assumed to be jointly normally distributed. The following equations use the notation given in [9]. In GP models the joint distribution of all these variables is defined as

$$f \sim GP(m(\cdot), K(\cdot,\cdot)) \tag{2.2}$$

where f is a vector of random variables, $m(\cdot)$ denotes the mean function and $K(\cdot,\cdot)$ the kernel or covariance function.

Given a set of data points in 3D,

$$\mathscr{D} = \{(x_i, y_i), \forall i = 1,...,N; x_i \in \mathbb{R}^3 \text{ and } y_i \in \mathbb{R}\} \tag{2.3}$$

it is possible to perform a regression [52] for any point $x^* \in \mathbb{R}^3$. This regression is computed as follow

$$f^* = \mu_f^* + K^{*T} \cdot K^{-1} \cdot (Y - \mu_f) \tag{2.4}$$

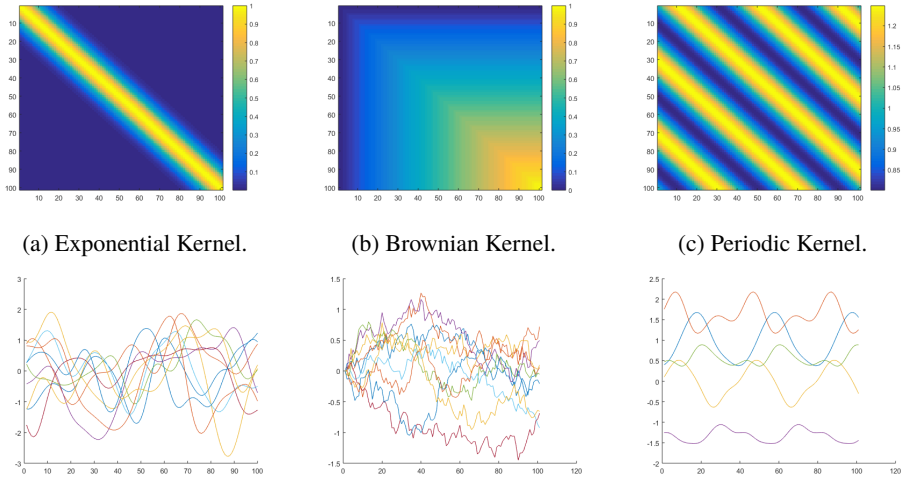
$$\Sigma^* = K^{**} - K^{*T} \cdot K^{-1} \cdot K^*, \tag{2.5}$$

where $Y \in \mathbb{R}^N$ is a concatenation of the data values y_i, $K \in \mathbb{R}^{N \times N}$ is the covariance matrix between the data points, $K^* \in \mathbb{R}^{N \times 1}$ is the covariance between the data points and the query point, $K^{**} \in \mathbb{R}$ is the covariance matrix evaluated for the query point, $\mu_f \in \mathbb{R}^N$ is a vector containing the concatenation of the mean function evaluated at the data points and $\mu_f^* \in \mathbb{R}$ is the mean function evaluated at the query point.

The kernel function $K(\cdot,\cdot)$ defines the correlation between data and any query points. Some examples of kernel functions are:

$$\begin{aligned} \text{Exponential kernel} \quad & K(x,y) = exp(-\tfrac{(x-y)' * (x-y)}{2*l^2}) \\ \text{Brownian kernel} \quad & K(x,y) = l * min(x,y) \\ \text{periodic kernel} \quad & K(x,y) = exp(-\tfrac{2*sin(\frac{x-y}{2})}{l^2}) \end{aligned} \tag{2.6}$$

Figure 2.5 shows the resulting GPs using different kernels (and assuming $m_f(\mathbf{x}) = 0$). One of the benefits of using this probabilistic model is that it is possible to sample from the GP to generate different possible functions as shown in the same figure.

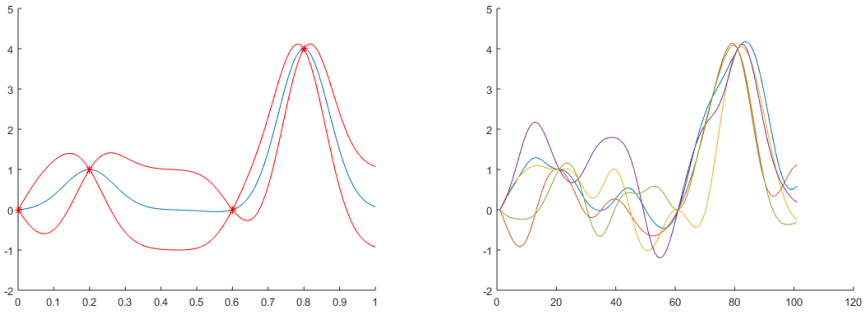


(a) Exponential Kernel. (b) Brownian Kernel. (c) Periodic Kernel.

(d) Exponential Samples. (e) Brownian Samples. (f) Periodic Samples.

Figure 2.5 Different kernels and samples.

Figure 2.6 shows an example of 1D regression. The plot on the left shows the mean values of the distribution (blue) and the associated variance (red). Red stars represent the data points (or observations). The plot on the right shows some examples of sampled functions from the posterior distribution.



(a) Posterior distribution.

(b) Samples from posterior.

Figure 2.6 Gaussian Process Regression.

The previous model assumes that there is no uncertainty in the data, for that reason the variance in data points(red stars in figure) is always zero. It is possible to introduce noise to the process by adding a term to the value function $y = f(x) + \varepsilon$ such that $\varepsilon \sim N(0, \sigma_y^2)$. This modification in the value function is translated in a small modification in the covariance matrix, resulting in $K_y = K + \sigma_y^2 * I$

Up to this point, 1D GPs have been introduced. GPs can be extended to any dimension as f is a scalar function that can be of any shape. For the purpose of this thesis, the surface of the object is modeled as a level-set of f function in the 3D space, so that

$$f = \begin{cases} Interior, & \text{if } f < 0 \\ Surface, & \text{if } f = 0 \\ Exterior, & \text{if } f > 0. \end{cases} \tag{2.7}$$

According to the previous statement, all observed data points $x_i \in \mathscr{D}$ are hence set to $y_i = 0$ as they are assumed to belong to the surface, with some associated noise $\varepsilon \sim N(0, \sigma_y^2)$

Additionally, to the assumption that observation points lie on the surface, i.e., they have a value equal to zero, it is possible to add surface normals to the observed data-points. This information can be captured by the camera or computed from the observed data points. This extra information can feed the GP regression by extending the mean and covariance functions with derivative terms. Derivatives of a GP are also Gaussian. Thus the covariance functions between data points and derivatives can also be computed [53, 54].

Hence, it can also be inferred the normals on new points.

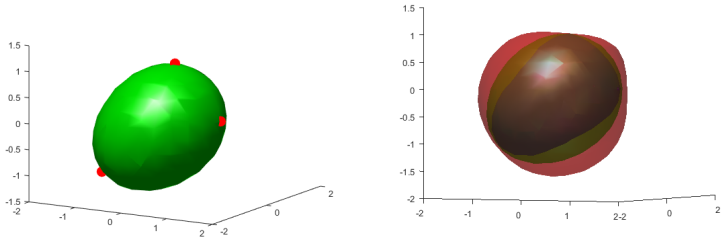
$$\begin{cases} \mathrm{K}\left(\frac{\partial f(\mathbf{x})}{\partial x_i}, f(\mathbf{x}')\right) & = \frac{\partial k(\mathbf{x},\mathbf{x}')}{\partial x_i} \\ \mathrm{K}\left(\frac{\partial f(\mathbf{x})}{\partial x_i}, \frac{\partial f(\mathbf{x}')}{\partial x'_j}\right) & = \frac{\partial^2 k(\mathbf{x},\mathbf{x}')}{\partial x_i \partial x'_j}. \end{cases} \tag{2.8}$$

As a result, the dataset $\mathscr{D}$ is extended as

$$\mathscr{D}^+ = \{(x_i, y_i), \forall i = 1...N, x_i \in \mathbb{R}^3 \text{ and } y_i \in \mathbb{R}^4\}... \tag{2.9}$$

with y_i being the concatenation of the function value at x_i (usually zero for GPIS) and the components of the normal $[\nabla x, \nabla y, \nabla z]$. Equations 2.4 and 2.5 are modified using $Y \in \mathbb{R}^{4N}$, $K \in \mathbb{R}^{(4N)\times(4N)}$, $K^* \in \mathbb{R}^{(4N)\times 1}$, $K^{**} \in \mathbb{R}^{4\times 4}$, $\mu_f \in \mathbb{R}^{4N}$ and $\mu_f^* \in \mathbb{R}^4$.

Figure 2.7 shows the result of a regression in 3D. This Figure only shows iso-surfaces of level zero and $\pm\sigma$, as for obvious limitations it is not possible to show the scalar function in the 4th-dimension. This regression was computed using an exponential kernel, and a set of four 3D data points. The regression fits the data points and smoothly predicts the most probable shape, resulting in an oval figure.



(a) Zero-level iso-surface.

(b) Zero-level and 3σ variances iso-surfaces.

Figure 2.7 Gaussian Process Regression in 3D. This figure shows: on the left, the zero-level iso-surface of the mean function of a 3D regression using an spherical prior; on the right it is shown the mean on green and the variance of the mean on the zero-level and the iso-level variance surface shown with the red (outside) and blue (inside) surfaces.

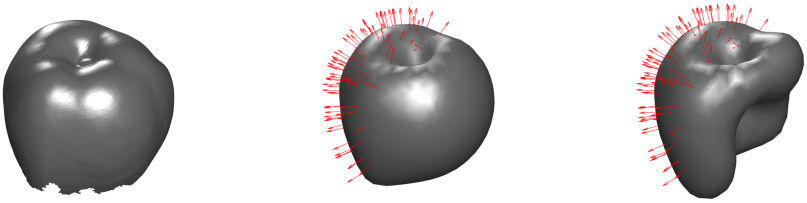
Most of the works focus on the use of constant mean functions for the regression of the data. However, in some situations, it is possible to have clues of the approximate shape or a primitive shape. In this dissertation, prior shapes have been introduced in the mean function of the GPIS process to improve the regression or kriging [55]. Particularly, in [9] following prior shapes were explored: ellipsoids, cylinders and finite planes.

The first prior is the ellipsoidal. It is a generalization of the spherical prior for object shapes with non-isotropic dimensions. Let a,b,c be the semi-major axes of an ellipsoid in 3D, and $\mathbf{A}_E = \text{diag}[a^{-2}, b^{-2}, c^{-2}]$. For any $h \in \mathbb{R}$, the mean function

$$m_S(\mathbf{x}) = \frac{h}{2}\left(\mathbf{x}^T \mathbf{A}_E \mathbf{x} - 1\right) \tag{2.10}$$

satisfies the zero-level set assumption for the surface of an ellipsoid defined by a,b,c. However, unless $a = b = c$, the magnitude-one assumption for the surface gradients will be violated at a non-empty set of points, regardless of how h is selected. Possible choices are, for example, setting h equal to either of a,b,c, resulting in correct gradient magnitudes in the corresponding direction, or setting h equal to their mean.

Figure 2.8 shows an example of surface reconstruction for an apple with ellipsoidal prior versus constant level mean function. The first figure shows the ground truth captured by an Artec EVA device. For the reconstruction, only half of the points have been used. These points are represented with the red arrows behind the apple. Second figure shows the reconstruction using GPIS and the ellipsoidal prior function. It shows that the second half of the apple is reconstructed using the GPIS. It can be observed that the use of the prior shape is highly beneficial against the use of a constant level function. The prior adds the necessary information to reconstruct the volume of the apple. While, as shown in the third picture, the constant level prior completely loses this information. This result is very important in robotics manipulation, as the later analysis of the shape for the manipulation task may fail, if the estimation of the surface has significant errors.



(a) Complete shape. (b) Using geometrical prior. (c) Using constant level prior. .

Figure 2.8 Reconstruction of a scanned apple with and without geometrical prior from observations on half of the surface.

The second prior is the cylindrical. It is interpreted as the limit of an ellipsoid with one of the major axes going to infinity, resulting in a scale matrix $\mathbf{A}_C = \text{diag}[a^{-2},b^{-2},0]$ for $c \to \infty$, for instance. Figure 2.9 shows the reconstruction of a tree trunk using a cylindrical prior. In this example, the data points belonging to the tree trunk were manually segmented from the remaining data points.

At last, while the introduction of implicit surfaces was motivated to model closed surfaces, it may also be useful to use planar GPIS priors. The planar mean prior can be used for example to represent tables, floors or walls. In this model, the mean function is set to

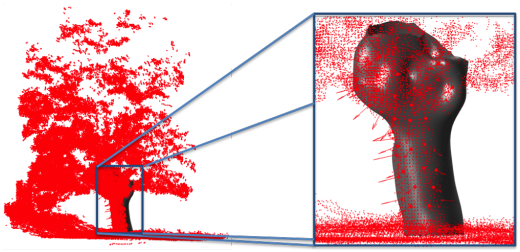


Figure 2.9 Reconstruction of a tree trunk using a cylindrical prior.

$$m_P(\mathbf{x}) = \mathbf{n}^T\mathbf{x}, \tag{2.11}$$

where $|\mathbf{n}|_2 = 1$, describes a linear prior with planar zero-level set. The resulting surface intersects with the origin, and its gradient magnitude is 1 everywhere, thus satisfying the gradient magnitude assumption. In particular, let $\mathbf{n} = [0,0,1]^T$, so that $m_P(\mathbf{x}) = x_3$ describes a linear function in 3D, with constant gradient pointing in positive x_3-direction. The zero-level set is then given by the x_1,x_2-plane, with the infinite half-volumes for $x_3 < 0$ and $x_3 > 0$ interpreted as the "inside" and the "outside" of the object. A natural application of this prior is for representing a ground surface in 3D-pointclouds, with positive x_3 pointing upwards.

Unfortunately, some objects, such as a chair or a lamp, cannot be represented with primitive shapes. In these cases, GPIS can still be applied using a zero-mean prior. Future implementation will integrate more complex priors named composite priors. These priors will be piecewise-defined functions in the d dimensional space. The main limitation of this priors is that they need to be continuous and their first derivatives too to ensure proper behavior of the GP.

Combining point clouds with GPIS for pose uncertainty

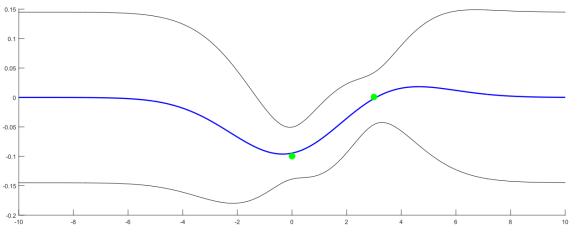
Up to this point, two methods have been presented to model the mean function of GPIS: a constant mean level function for $m(\cdot) = c$; and the use of geometrical functions as primitive shapes for the mean functions such as spheres, ellipsoids, improving the reconstruction of the expected volume of the object according to the prior. The use of prior knowledge in the shape of the object is undoubtedly helpful for grasping, contrary to standard constant-mean functions that produce in holes in the volume of the objects which is undesirable for the grasping process.

In this work, instead of modifying the mean function, a method for adding prior information in the form of extra data is proposed. Let $\mathbb{S}$ be a set of points that models the shape of the targeted object, and $\mathbb{D}$ a set of points obtained from an input sensor, which is assumed to belong to the object. A variance term is inferred for each point in the model data set according to

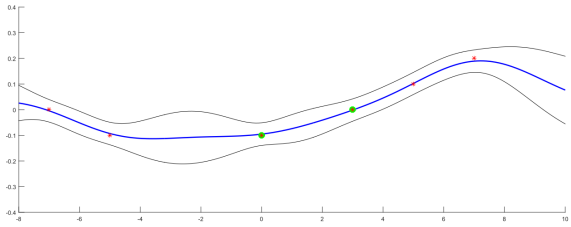
$$\sigma(x_i^{model}) = \sigma_{alignment} * e^{min\_dist(x_i^{model}, x_j^{obs})} \tag{2.12}$$

being $\sigma_{alignment}$ the variance of the correspondences used for the alignment of the model and $min\_dist(x_i^{model}, x_j^{obs})$ is the minimum distance between the point and the observed points.

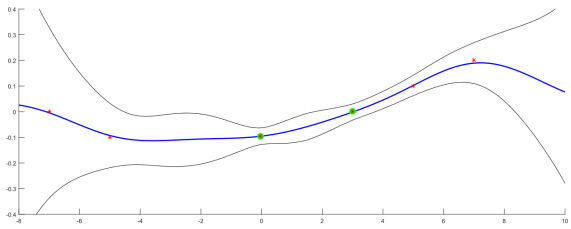
For simplicity, a 1D example is considered first. Figure 2.10a shows the standard regression using a constant mean function and a fixed observation noise set for each point. Figure 2.10b shows an example of regression using a model aligned with the data. However, this does not take into account that points that lie far away from the observations cannot have the same confidence that those points that are close to them. For this reason, a varying sigma depending on the distance to the data points is introduced as shown in Figure 2.10c.



(a) No model and fixed sigma noise.



(b) Model and fixed sigma noise.



(c) Model and varying sigma.

Figure 2.10 1D example of varying sigma for prior model for GPIS. The green circles correspond to real observations, the blue line the mean function and the pair of black lines represent the variances on the value of the function. Red asterisks are the data points from the aligned model.

This method allows the introduction of custom complex mean shapes for the GPIS regression, by using the point clouds as virtual data with varying variance. Comparing Subfigures 2.10a and 2.10c it can be clearly seen how both have the desired mean shape, but the proposed model produces larger sigmas for unconfident points that are further away from the observations.

The same process can be applied for 3D shapes as shown in Figure 2.11. At first, an initial model is placed with the observed point cloud using any alignment algorithm, such as RANSAC or ICP, resulting in an initial pose for the object. However, these algorithms might have alignment errors, so the variance term on the points of the model is introduced. It can be seen that the mean shape of the GPIS bends towards the model points reconstructing the shape of the object correctly. This would not be possible with only a constant mean function and the observations. Other authors use the thin plate covariance kernel which minimizes the changes in the gradients, but it still fails when the gaps in the observed surfaces are large [9].

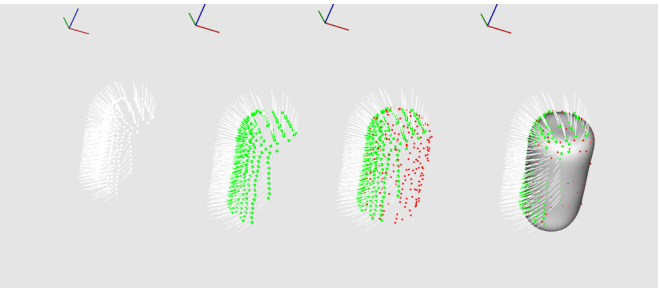


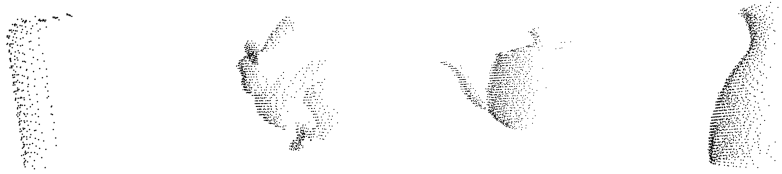
Figure 2.11 Sequence for 3d object. Firstly input cloud is received, then the model is aligned and voxelized to reduce the number of points used for the GPIS regression. Finally, the regression is performed obtaining the shape of the object.

Experiments have been performed first with virtual objects, similar to those that can be found in public datasets [56]. In order to test the algorithm, partial observations (in the form of point clouds) are generated using BlenSor [57]. BlenSor is a modified version of Blender 3D tool that can simulate different depth sensors such as Kinect or Time of Flight cameras (ToF).

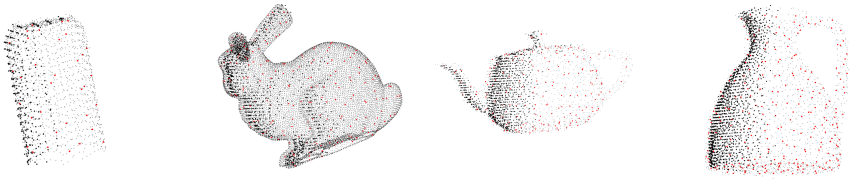
The computation time of GP regressions scales poorly with the size of the input data. For this reason, the input point cloud of the model can be downsampled using any kind of filtering. A super-voxelization algorithm presented in [58] is chosen, but any other simple voxel filtering or downsampling algorithm can also be used.

Figure 2.12 shows the results of the surface reconstruction using the proposed algorithm in comparison with the result without using the prior shape model and a constant mean level function.

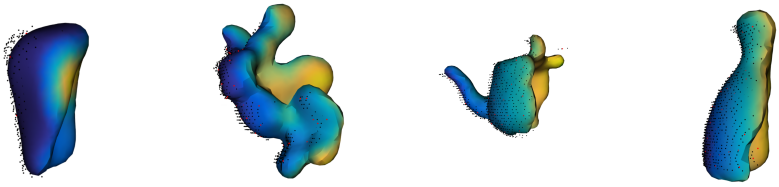
Note that the complete surface reconstruction is not needed for the grasping process with GPIS and it is computationally expensive. For these reasons, it may not be computed



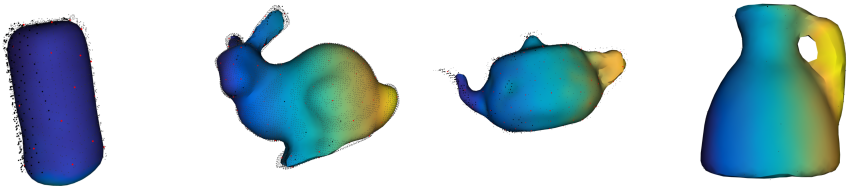
(a) Coke input data. (b) Bunny input data. (c) Tool input data. (d) Jar input data.



(e) Coke aligned & voxels. (f) Bunny aligned & voxels. (g) Teapot aligned & voxels. (h) Jar aligned & voxels.



(i) Coke non-model surface. (j) Bunny non-model surface. (k) Teapot non-model surface. (l) Jar non-model surface.



(m) Coke our surface. (n) bunny our surface. (o) Teapot our surface. (p) Jar our surface.

Figure 2.12 Simulation examples of surface reconstruction for different objects using. First row shows a simulated input cloud with a partial observation. Subfigures in second row have the result of the alignment of the model with the input cloud (small white dots) and the supervoxelization (in red dots). In third row there are the computed surfaces of the objects using our method. Finally, last row shows the reconstruction of the surface using a GPIS with a constant mean level.

in the grasping process, and it is shown for demonstration purposes using the algorithm described in [9].

The above probabilistic model has been used to model the surface of a crawler robot for grasping during the AEROARMS projects. The experiments have been carried out in a hexacopter platform equipped with an Intel RealSense D435 camera, as shown in Figure 2.13.

Figure 2.13 Aerial platform used for the experiments..

Figure 2.14 show captures of a real experiment. At first instance, the object has been detected and its pose estimated using the 3D model and the point cloud acquired by the camera. Then, an approximated reconstruction of the object's surface has been computed using the GPIS model.

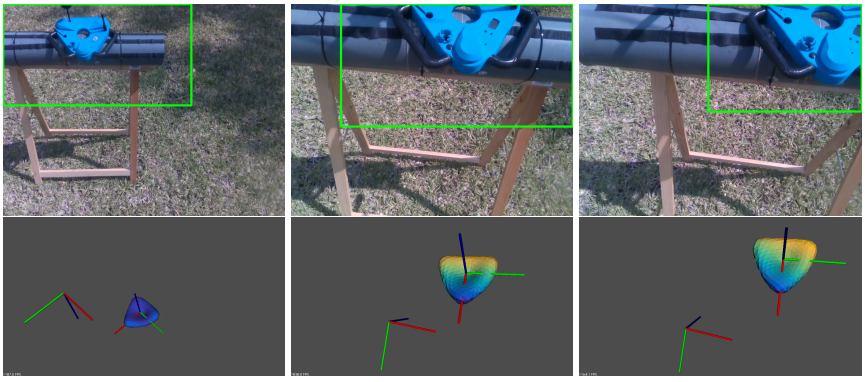


Figure 2.14 Snapshots of outdoor experiments. Reconstruction of the complete surface with colored covariance from the GPIS.

The GPIS model calculates the covariance term for each of the points on the approximated surface of the object. These covariances allow computing some probabilistic metrics for manipulation such as the probability of force closure (described later in Section 2.3.1),

which gives the probability of a grasp to be stable. Figure 2.14 shows the experiment in three different instants. From left to right columns; the crawler is completely observed, for this reason, the uncertainty is uniform and low close to the observation points. In the second column, the crawler started leaving the frustum of the camera, the alignment algorithm still have enough information for acquiring an approximate pose of the target object, but as part of the object is not observed, the uncertainty increased on the backside. In the third column, the top-right part of the crawler is out of camera's sight. Thus the uncertainty increased in the right side too.

As conclusion, the computation of this covariance terms in the surfaces allows taking into account both the shape of the object and which part is better to be used for grasping in situations where the object is wholly observed.

Remark on GP parameters: The hyperparameters for the kernel function and mean function were selected ensuring that the reconstruction of the object is reasonable. A broader study about the parametrization and learning of the hyperparameters is proposed as future work.

2.3 Object manipulation - Grasp generation and evaluation

In order to perform manipulation tasks, the robot needs to compute how to grasp objects efficiently. This section introduces how to generate grasps efficiently and a variety of methods to evaluate the quality of these grasps. All the methods and metrics introduced in this section has been implemented and integrated in the OpenSource library *grasping_tools* introduced in Appendix A.2

2.3.1 Grasp Quality metrics

The manipulation of objects requires the analysis of objects' shape as explained in the previous section. However, to manipulate, it is necessary to determine the appropriate contact points for the manipulator's end-effectors. The computation of these points depends on the object surface model. The combination of them generates a grasp, but not all the possible grasps are good candidates to manipulate objects. Thus it is necessary to evaluate them to be able to achieve desired tasks.

Grasps are evaluated with quality metrics[59]. This section summarizes the most commonly used metrics. These can be categorized in regards to the relative position of the contact points, the global hand configuration and other variety of aspects.

Before analyzing the different state of art quality metrics, some fundamental concepts are introduced in this section.

As aforementioned, the robot exert forces on the object through its surface on the contact points. These contact forces can be modeled in three different ways[59, 60, 61], as shown in Figure 2.15

- **Punctual contact point without friction**. In this model, the force exerted on the object is only applied in the direction of the normal.
- **Punctual contact point with friction**. In this model, a force is applied in the direction of surface's normal. Additionally, forces may be exerted in the tangential

direction. These tangential forces are usually modeled with Coulomb's friction model and are usually discretized with a convex cone.

- **Soft contact point**. This model assumes the same forces the Punctual contact point with friction but with an additional torque around the direction of the normal.

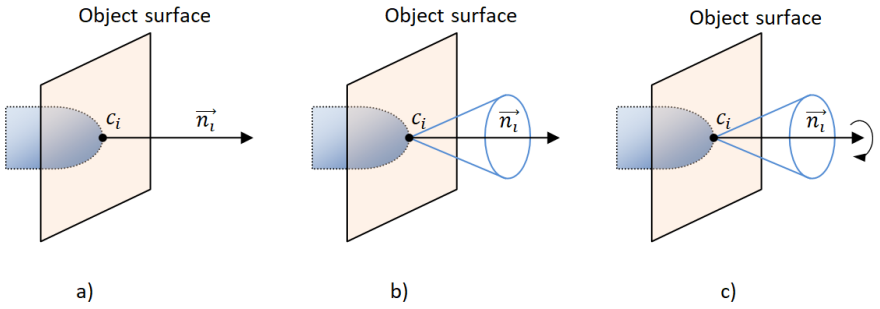


Figure 2.15 Contact point types.

These forces on the object are usually represented in a vector so called wrench. A wrench ω_i is a $6D$ vector which contains the $3D$ vector of forces F_i and a second $3D$ vector with the torque τ_i exerted by F_i from the center of the object c to the contact point p_i being $\tau_i = F_i \times (p_i - c)$. Thus,

$$\omega_i = \begin{pmatrix} F_i \\ \tau_i \end{pmatrix} \tag{2.13}$$

As mentioned before, tangential forces may appear. These forces, if they exist, oppose the movement. Thus they do not have an explicit representation. Instead, a convex cone of possible wrenches containing all the possible tangential forces is used. This cone is represented in the *wrench space* which is commonly named *Cone Wrench Space* (CWS)

$$CWS_i = \left\{ \omega_i \middle| \omega_i = \begin{pmatrix} F_i \\ F_i \times (p_i - c) \end{pmatrix}, \|F_i - (F_i \cdot n_i)n_i\| \leq \mu(F_i \cdot n_i) \times \|F_i\| \right\} \tag{2.14}$$

The summary of all the n CWS exerted by the manipulator on the object is called *Grasp Wrench Space* (or GWS).

$$GWS = \left\{ W \middle| \omega_i = \sum_{i=1}^{n} \omega_i \in CWS_i \right\} \tag{2.15}$$

A complete evaluation of the cone increases the complexity of the grasp analysis. Thus it is usually approximated to a polyhedral convex cone as shown in Figure 2.16

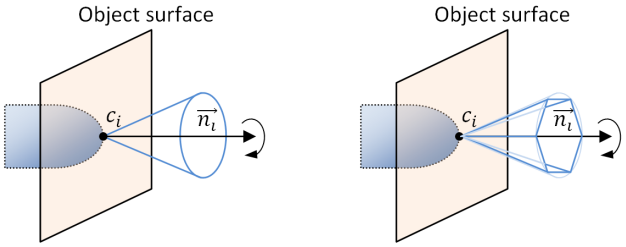


Figure 2.16 Approximate wrench cone by a convex polyhedral cone.

For the grasp analysis these contact point information information is stored in a matrix called *Grasp Matrix* (G). It is a $\mathbb{R}^{6x6n}$ matrix that is composed of n partial matrices ($G_i \in \mathbb{R}^{6x6}$), being n the number of contact points. It is computed as follow,

$$G_i = \begin{pmatrix} R_i & 0 \\ S(p_i - c)R_i & R_i \end{pmatrix} \tag{2.16}$$

being R_i a $\mathbb{R}^{3x3}$ matrix which represents the orientation of the contact point, c the center of the object, p_i the position of the contact point and $S(t)$ the cross-product matrix given by,

$$S(t) = \begin{Bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{Bmatrix} \tag{2.17}$$

Eventually, the complete grasp matrix is the concatenation of all the grasp matrices,

$$G = \{G_n \| ... \| G_n\} \tag{2.18}$$

Quality metrics derived from the Grasp Matrix

Most of the quality metrics based on on the geometrical position of the contact points are computed from the *Complete Grasp Matrix*, which encloses fundamental geometrical properties of the object and the grasps such us the size or the geometrical distribution of the contact points.

A grasp is stable if its Grasp Matrix is full rank, or in other words, if it has six singular values (σ_i) by the positive square root of GG^T. If any of the singular values turn to be zero the grasp become unstable and incapable of resisting external wrenches.

$$G = U \cdot \Sigma \cdot V^* \qquad being, \qquad \Sigma = \begin{bmatrix} \sigma_1 & 0 & ... & 0 \\ 0 & \sigma_2 & ... & 0 \\ 0 & 0 & ... & 0 \\ 0 & 0 & ... & \sigma_6 \end{bmatrix} \tag{2.19}$$

Concerning the singular values of the grasp matrix, this section focuses on three measures: the minimum value of the singular values, the volume of the ellipsoid in the wrench space and the grasp isotropy index.

As the singular values are required to be strictly larger than zero, the minimum value of the singular values indicates if the grasp is close or not to a singular grasp.

$$Q_{min} = \sigma_{min}(G) \tag{2.20}$$

The volume of the ellipsoid in the wrench space provides a general measure of the global contribution of the contact forces which is essential to be maximized. However, as all singular values are equally weighted, it is not possible to determine the contribution of each of the contact forces.

$$Q_{volume} = \sqrt{det(GG^T)} = \sigma_1 ... \sigma_n \tag{2.21}$$

The last metric (grasp isotropy index) tries to determine if the grasp present a similar behavior in all the direction. If the index is close to 1 it means that the grasp can be used for any task, i.e., it resists external wrenches, equally, in any direction. However, if the index tends to zero, it means that it presents a lousy behavior against some direction of the external wrenches.

$$Q_{isotropy} = \frac{\sigma_{min}(G)}{\sigma_{max}(G)} \tag{2.22}$$

The principal advantage of the grasp quality metrics based on the grasp matrix is that they are relatively easy to compute. However, the information that they give is vague in some situations and qualitative. For these reasons, the following section introduces other quality metrics that are widely used and provide more quantitative information for the manipulation tasks.

Quality metrics from other aspects

A grasp is said to be good if it can resist external wrenches in any direction. One common property to ensure this condition is the force-closure. Nguyen defined in [60] *"A grasp on an object B is a force closure grasp if and only if we can exert arbitrary force and moment on object B by pressing the fingertips against this object. Equivalently, any arbitrary motion of object B will be resisted by a contact force from the fingers, which means that B cannot break contact with the fingertips without some non zero external work. That is, the total freedom of B is zero."*

As mentioned before, the forces exerted by the manipulator on the object are wrenches which are usually approximated by convex cones in $\mathbb{R}^6$. The combination of all the convex cones defines the Grasp Wrench Space. The boundary of the GWS can be enclosed by with 6D convex hull of all the wrenches. Authors in [62] proof that a necessary condition for a grasp to have force closure is that the convex hull needs to enclose the origin of the wrench space.

There are two common ways to construct the convex hull of the wrench space concerning the constraint of fingers' force. The first assumes that each finger has the same force and is

limited to 1, i.e., $\|f_i\| \leq 1 \forall i = 1...n$. The second model assumes that the sum of all forces is limited and normalized to 1, i.e., $\sum_i^n \|f_i\| \leq 1 \forall i = 1...n$.

Ferrari and Canny in [63] proposed a new quality metric based on the convex hull of the GWS: the distance from the origin to the closest facet of the polytope ($\mathscr{P}$) of the approximate convex hull of the GWS. Geometrically, this measurement is equivalent to the radius of the largest sphere centered on the origin that can be fully contained in $\mathscr{P}$. It is usually called *largest minimum resisted wrench* (or *lmrw*). Figure 2.17 shows the *lmrw* for both of fingers' force models. Additionally, if $lmrw > 0$, it means that the origin is contained by the convex hull, so the grasp has force closure too without extra computations.

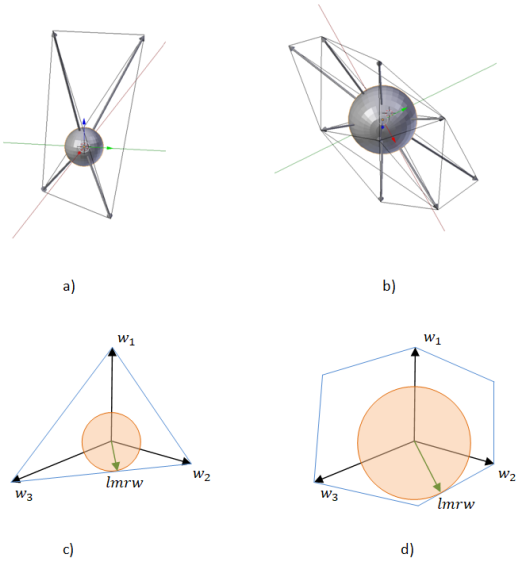


Figure 2.17 Largest minimum resisted wrench using both models of fingers' force.

This measure is widely used and very convenient as it grants that the grasp has force-closure and also defines the weaker direction of the grasp. An alternative quality metric has been proposed in [64] which is the volume of the complete GWS. This metric is invariant to the orientation of the grasp, but it has a high computational cost and does not indicate the poorest direction that the grasp can resist.

All the quality measures introduced, up to this point, assume that external wrenches, i.e., perturbations, may occur in any direction. Nevertheless, in many situations, it is possible to assume a previous knowledge in the way perturbations are exerted. Li and Sastry [64] introduced the concept of *Task Wrench Space* (TWS) as a set of wrenches that can be specified for a given task. This TWS is a polytope $\mathscr{P}$ which shape is usually a 6D ellipsoid.

However, it is not always possible to explicitly define the TWS or do it accurately. Previous work [65, 66] extended it in case of partial known of the tasks. They introduced

that there is always a set of forces that can be known such as the gravity or the forces that arise by the acceleration of the object, this is called *Object Wrench Space* (OWS). In general, TWS and OWS are more convenient, as the lmrw is more restrictive.

An additional, grasp quality metric is introduced which derives from the force closure metric: the probability of force closure [67, 68]. The probability of force closure is defined as the probability of a grasp with uncertain contact points and normals of having force closure. Its analytical solution is complicated, so usually, this metric is obtained by Monte Carlo sampling. Each contact point is sampled, and then it is computed if it has force closure. The probability can then be approximated by evaluating N samples. This quality metric can be applied in the probabilistic framework of GPIS thanks to the possibility of random sampling processes with the definition of the underlying GP model.

Previously described quality measurements are based in mathematical definitions of geometrical properties of the contact points of the manipulators with the object. However, recently, a new paradigm has appeared in robotics manipulation as in many other fields of robotics. Neuronal networks can be used to generate approximate grasps by a supervised training of sets of labeled data. Contrary to previous mathematical algorithms, these algorithms infer clues of the possible grasps from the collections of data.

Caldera et al [69] provide a comprehensive review of Deep learning algorithms for grasp detection. As an example, authors in [70] used RGB-D to fed a neuronal network to compute feasible gripper-like grasps (as shown in Figure 2.18). They introduce a structured regularized approach to improve the efficiency of the network taking into account the different nature of the color and depth channels.

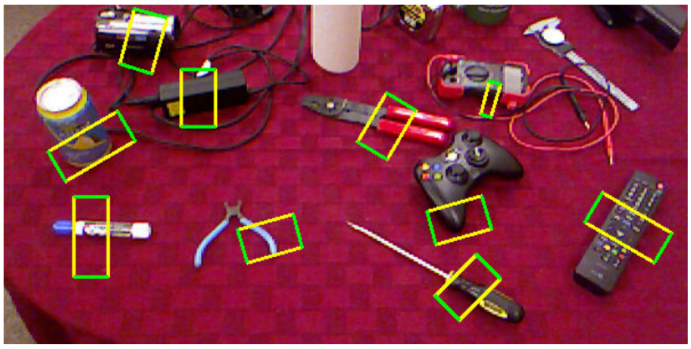


Figure 2.18 Example results on a cluttered scene using deep learning. Rectangles correspond to feasuble grasps for the robot. Resource from [70].

2.3.2 Grasp generation - Deterministic shape models

Once objects surfaces are modeled, it is possible to determine processes for generating grasps. The first deterministic model that is going to be used is the mesh-based model. As stated before, a mesh is a set of faces that encloses a volume. Author in [71] uses a bound box to generate a set of rays which are cast from a bounding box towards the target object.

Each ray collides once with the surface, and the surface normal is computed. Figure 2.19 shows an example for a crawler robot in the AEROARMS project.

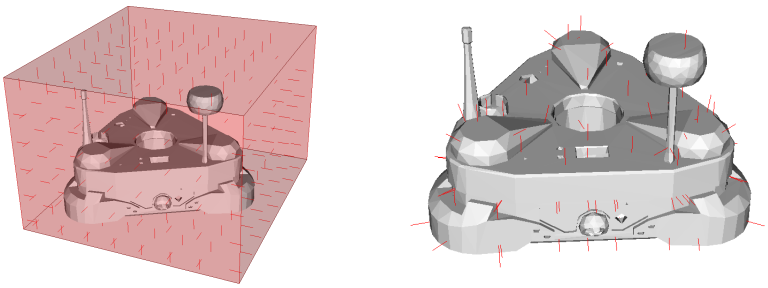


Figure 2.19 A simple way of parameterizing the grasp search space..

The method above computes a set of possible contact points for grasping the object. Then, knowing the kinematic model of the robot and its restrictions it is possible to compute feasible grasps.

However, this methodology might fail to generate feasible grasp for objects with complex shapes. A simple method has been implemented in this thesis to take into account any non-convex shape extending the previous procedure. For this purpose, the rays are collided not only with the first encounter facet but also to all the possible facets. Figure 2.20 shows an example of ray tracing of an object with an U-shape. If gripper's size is smaller than a size, then the object is un-graspable for the robot using previous method. In this section, a slight modification of the algorithm is proposed, which takes into consideration all the possible internal nooks on the surface.

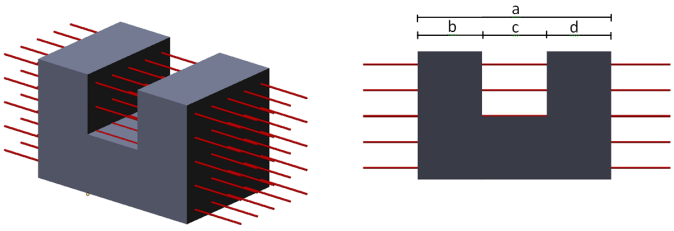


Figure 2.20 Example of non-convex object for grasping algorithm. Red lines represent the rays used to trace the possible grasp points..

In general, terms, let be a single ray passing through an object. The number of collisions is

$$n_{collisions} = n_{folds} \cdot 2$$

thus the number of combinations of opposite contact points that might generate a grasp is

$$n_{grasps} = \sum_{i=1}^{n_{folds}} i$$

Ray tracing algorithms have been intensively studied, especially in the field of computer graphics [72]. In this work, faces are composed of just three vertices. Let be a triplet of points forming a facet $V = v_1, v_2, v_3$ and a ray defined by two points p_0 and p_1, it is possible to determine if the ray intersects the triangle by using Möller–Trumbore intersection algorithm [73].

This contact point candidates generation might be time intensive if the mesh of the object is too complicated. For this reason, once the ray tracing is performed, the untransformed results are stored into a binary file, so if the same object or a different instance of the same object is grasped later, this database is recovered, avoiding all the ray tracing computations.

As aforementioned, meshed representations are very common and this algorithm can be used for objects of any kind. Figure 2.21 shows examples of grasps for a parallel gripper for different objects. Gripper exerted forces are represented as red cones in the contact points.

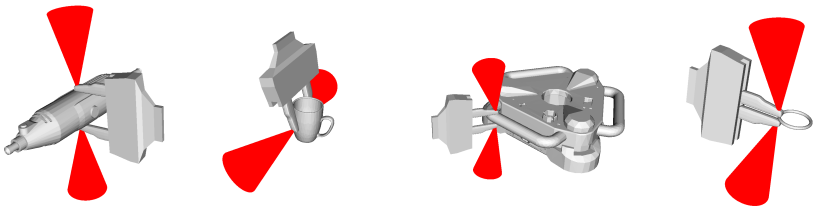


Figure 2.21 Grasp generation for a parallel gripper for meshed objects, from left to right: drilling tool, mug, crawler robot and ring. Red cones represent wrenches exerted on contact points of the grasps.

The other common deterministic object model is the point cloud. A similar procedure can be applied to them. The only difference is that point clouds are face-less representations of objects. Thereby, it is necessary to make some modifications.

The algorithm starts with the same steps. A set of initial points with directions are defined to perform the ray tracing algorithm (as shown in Figure 2.19). However, in this case, it is not possible to detect collisions with faces. Therefore, an octree [74] is used to get a fast estimation of the approximate collisions of the rays with the expected surface of the object. Figure 2.22 shows examples of grasp generation using point clouds for a parallel gripper. Red cones represent the wrenches on the contact points of the grasps.

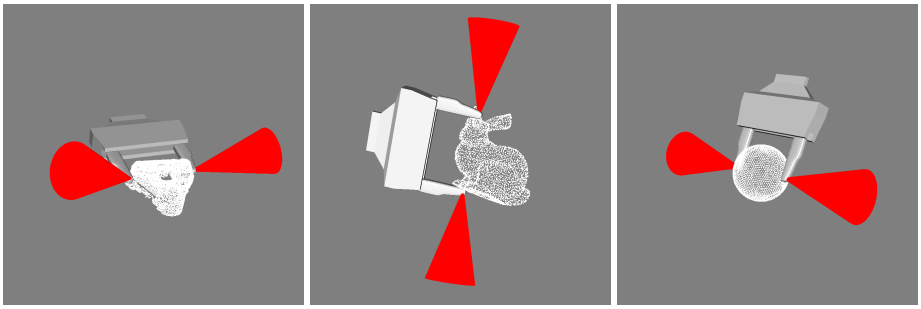


Figure 2.22 Examples of grasps generation using point clouds for a parallel gripper. From left to right, a crawler, Standfor's Bunny, and a sphere. Red cones represent the wrenches on the contact points..

2.3.3 Grasp generation - Probabilistic shape models

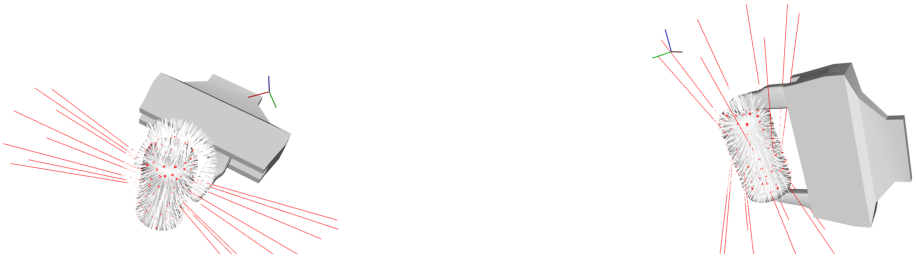
As aforementioned, one of the fundamental steps of the grasping process is the generation of grasp candidates. There are several algorithms to compute feasible candidates given the robot/hand kinematics. Usually, authors assume that the surface is known when generating the contact points as shown in previous sections. In this section, it is assumed that there is not an exact model of the object surface. The GPIS model presented in Section 2.2.4 is used for grasping. In these experiments, a parallel gripper is used, but a similar process can be extended for generating grasps for other manipulators. This work has been presented in [10] and implemented in the Open-Source library presented in Appendix A.2

The grasping process is outlined in algorithm 1. Initially, a random point away from the observations is selected. Next, using the GPIS information and a gradient descent algorithm, it is moved toward object's surface, being $f(x)$ the function to be minimized. As $f(x) > 0$ means outside, the algorithm just need to converge to $f(x) = 0$. Then, an opposite point to the initial one is taken and converged to the surface too. Figure 2.23 shows examples of grasps generated.

Algorithm 1 Grasp Generation using GPIS for parallel gripper

```
1: Choose random init point.
2: while is point on surface? do
3:     gradient ← GPIS(point)
4:     movepoint
5: end while
6: Choose opposite point on the previous sphere
7: while is point on surface? do
8:     gradient ← GPIS(point)
9:     movepoint
10: end while
```

As mentioned in the previous section, a new Grasp quality metric can be introduced in

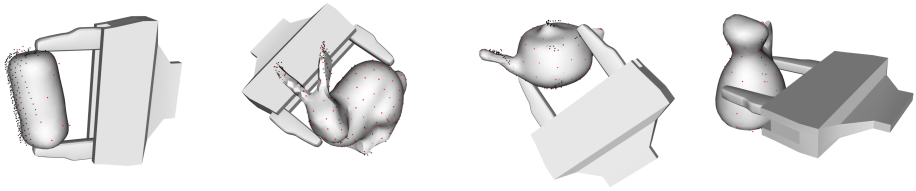


(a) (b)

Figure 2.23 Examples of grasps generated on a cylinder. The reconstructed surface is represented by a gray mesh. Red lines belong to the convex cone of forces generated in the grasping points..

this model thanks to the intrinsic uncertainty in this shape model. Previous work [67, 75] introduced the concept of Probability of Force closure out of the context of GPIS. In that work, authors use polygons or splines to model the contour of the objects and then handle the uncertainty with a probability distribution. The probability of force closure is computed by sampling the contact points and their normals using a probability distribution and then performing Monte-Carlo integration. This concept has been later used in the context of GPIS [76] as at each point of the space it is possible to infer a Gaussian probability distribution.

Being able to extrapolate the entire shape of the object in the GPIS regression allows generating grasps. As the variance of the surface increases with the distance to observations, the grasp algorithm tends to choose not only the stable (as in the deterministic shapes) but also the best stability regarding the probability of force closure. Figure 2.24 shows generated grasps for the simulated object obtained from the grasp planning process. For this figure, the objects were observed only from one side. The GPIS reconstructed the complete shape so the algorithm can compute efficiently feasible grasps.



(a) Coke input data. (b) Bunny input data. (c) Tool input data. (d) Jar input data.

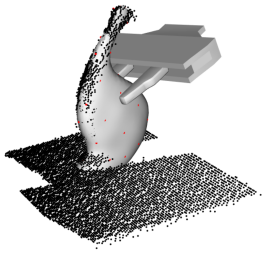
Figure 2.24 Generated grasps for simulated objects using the proposed method for reconstructing the surface. The surface is computed for showing purposes not for computing the grasps..

The same methodology was applied for real data, in which a Kinect Depth sensor was used to generate the object's models as well as to acquire the input clouds to the algorithm. Figure 2.25, shows generated grasps for different real objects.

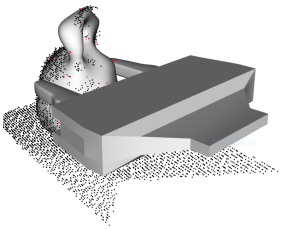
(a) Real jar.

(b) Real teapot.

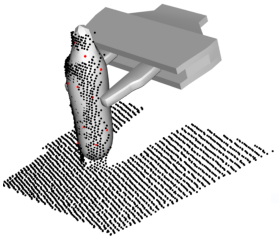
(c) Real spray.



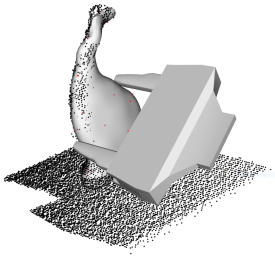
(d) Example grasp jar.



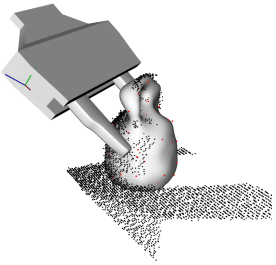
(e) Example grasp teapot.



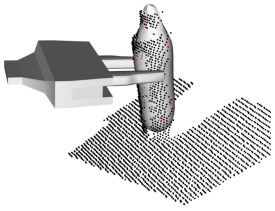
(f) Example grasp spray.



(g) Example grasp jar.



(h) Example grasp teapot.



(i) Example grasp spray.

Figure 2.25 Generated grasps for real objects using the proposed method for reconstructing the surface. The surface is computed for showing purposes not for computing the grasps..

2.4 Grasp planning with dual manipulators

Mechanical and mathematical models of robot hands and their interaction with the object are a fundamental aspect of the analysis of robotic manipulation. The vast amount of combinations of object and hand configurations makes this research challenging. This section describes how the grasping algorithms described above are used to perform grasp planning tasks with a pair of complete serial manipulators with grippers. This section presents the implementation and results of the work presented in [13]. In this work, the aerial platform perform a visual servoing to grasp objects using a pair of manipulators installed in the aerial robot. The object detection will be presented in Section 3.6.3 and the dual manipulator platform will be described in Section 5.3.

Particularly, the manipulators used in this work are part of an open-hardware & open-source project called *Hecatonquiros* introduced in Appendix A.3, which aims for general purpose, cheap and easy to use robotic arms. These are designed to be light-weight. Thus relatively small UAVs can carry them. These are 3D printed reducing the overall cost of its production as the material es relatively cheap and does not need any post-processing step, so can be assembled easily. The cost of a single arm is $\sim$ 150$ (including the smart serial servos). The project also provides for a library based on OpenRAVE [71] for the kinematic solvers and has support to ROS to perform simulations before and while doing real experiments. The overall description of the complete hardware, including the multicopter, is in Section 5.

The previous section introduced how to grasp general objects using parallel grippers taking into account, just, the limitations of these. In this section, complete serial manipulators are modeled to perform the grasp generation and grasp planning.

At first instance, a set of feasible grasps are generated taking into account its shape. These initial grasps are computed based on any of the previously mentioned methods, either for a deterministic or probabilistic model. Each of these possible grasps needs an extra requirement. These grasps need to be reachable, i.e., the arms should be able to place its end-effector in the target position. In order to determine this reachability, a best suitable distance has been computed according to arms configuration. Once the grasps are computed and its reachability computed, these are then arranged according to their properties, such as the largest-minimum resisted wrench, described previously in Section 2.3.1.

In order to compute the inverse kinematic of the arms, two methods have been used. The first one is a Fast-IK solver developed by Rosen Diankov [77]. The second one is an iterative method using the mathematical definition of arms' kinematics.

Let a multi-link robot manipulator $\mathscr{R}$ be specified by the scalar variables $\Theta = \theta_1, ..., \theta_n$ describing its joints states. The end of each links of the body have a certain position $S = s_1,...,s_n$. These positions can be described using Denavit-Hartenberg formulation [78] as a chain of transformations computed by a set of parameters $DH_i = \{\alpha_i, a_i, d_i, \theta_i\}$. Each DH_i element is a matrix which transform between link $i-1$ to i, i.e. forward kinematics. Figure 2.26 shows a particular arm configuration with 6DoF developed in this research.

The purpose is to obtain a set of adequate joints Θ that places the *end-effectors* on a target location T_t, or Inverse Kinematics (IK). This problem has been widely studied. Unfortunately, it does not have a straightforward solution in most of the situations. Iterative

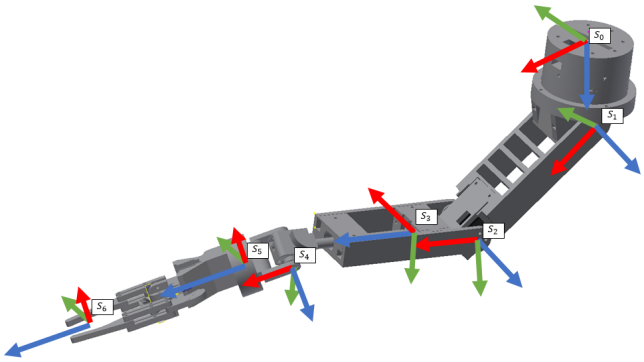


Figure 2.26 6DoF arm part of Hecatonquiros project, developed by the Ph.D. candidate during this research. The model will be introduced in more detail in Chapter 5.

methods [79] and sampling methods [80] perfectly suit to problem. Forward kinematic is given by functions $\Theta \mapsto S(\Theta)$ that are $\mathbb{R} \mapsto \mathbb{R}^3$. These functions can be linearly approximated using Jacobian matrices ($J(\Theta)$) close to current state Θ. The velocities can be expressed as,

$$\dot{S(\theta)} = J(\Theta)\dot{\Theta} \qquad\qquad J(\Theta) = \left(\frac{\delta s_i}{\delta \theta_j}\right)_{i,j}$$

The final purpose is to guide the arms towards the target grasps using the visual information the sensors. Particularly, in this work a Jacobian damped-least square (DLS) gradient descent method [81, 82] is applied to ensure the convergence of the arms towards the target position. This method has proved to be more robust to inverse and pseudo-inverse methods [82] near to instabilities and singularities in the Jacobian matrices. Let X_k and Q_k be the target position of orientation for the end effector given by the grasp planning and updated by the vision algorithm. The aim is to update manipulator's pose by updating its joints. The DLS proceed as follow

$$error = \begin{cases} X_k - X_{k-1} \\ Q_k - Q_{k-1} \end{cases} \qquad\qquad J_c = \begin{bmatrix} J_X \\ J_\Theta \end{bmatrix}$$

$$\Delta\theta_k = (J_c^T(\theta_k) \cdot J_c(\theta_k) + \lambda^2 \cdot I)^{-1} \cdot error \tag{2.23}$$

being X_k and Q_k the target position and orientation of the end-effector at instant k; θ_k current joints' angles of the arm; J_X and J_Q are the corresponding position and orientation jacobians at current state; and λ the damping coefficient to reduce the issues related to the

inversion of the matrix. λ needs to be large enough to ensure that the algorithm behaves adequately close to singularities but not too large to grant a good convergence rate.

An alternative to this method is the Fast-IK solver. It has the benefit of giving fast solutions, and the implementation is given and generalized to any kind of robots. However, in some situations, it leads to abrupt changes in joints coordinates to reach the same position. Contrary, the DLS method grants smooth transitions between any change in position because of being iterative as long as the parameters are well tuned to grant the convergence rate.

At last but not at least, working with a pair of manipulators require for additional constraints. The principal one is the need to avoid self-collisions, collisions with the object and collisions between the arms. However, checking collisions of any non-convex object is not trivial. The use of convex-hull is advantageous in some situations as it is relatively used to check collisions within convex-hulls, but this simplification can be rough in some applications. Modern approaches split the objects into a set of convex hulls [83, 84]. It results in a more accurate solution while preserving the advantages and mathematical simplifications of working with convex hulls. In this work, the implementation given by OpenRAVE is used [77].

Figure 2.27 shows some examples of grasps generated using the described methodology.

Figure 2.27 Examples of dual grasping using the proposed methodology for the dual manipulator represented in OpenRAVE QtCoin visualizer..

Figure 2.28 shows the value of the joints during two visual servoing tests. The dashed line corresponds to the target joints and the solid line to the actual joints values. A change in grasp due to the previous target being unreachable is shown in the middle of the figure. Figure 2.29 shows snapshot of a test-bench experiment and a real flying experiment from both the point of view of the robot and from outside.

In this section, it has been demonstrated that it is possible to perform grasping with dual arms using the grasp analysis described in this chapter. Despite the oscillations of the aerial platform due to the control system, the DLS method ensures the right convergence of the inverse kinematic and smooth the trajectories in real time. Thanks to DLS, it is not necessary to compute explicit trajectories as far as the target positions are ensured to not collide with any object.

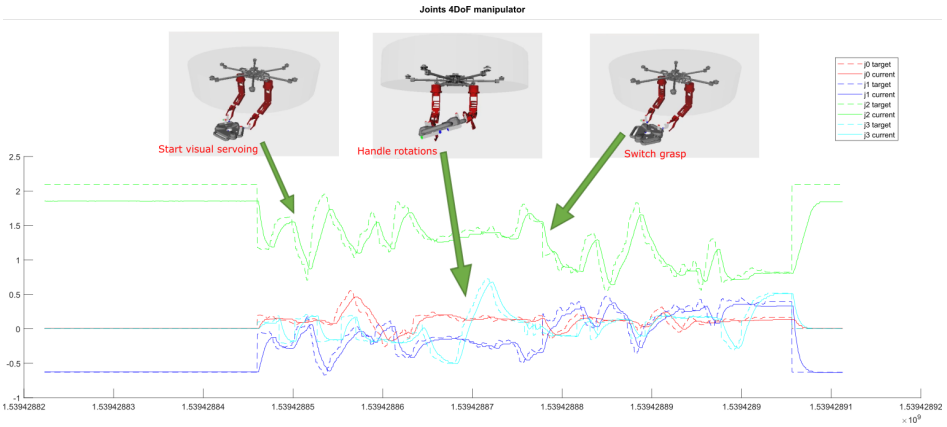


Figure 2.28 Joints values during experiments. At first instance the arms follow the grasps target. Up to a certain point, the object rotates so the system switch to another feasible grasp..

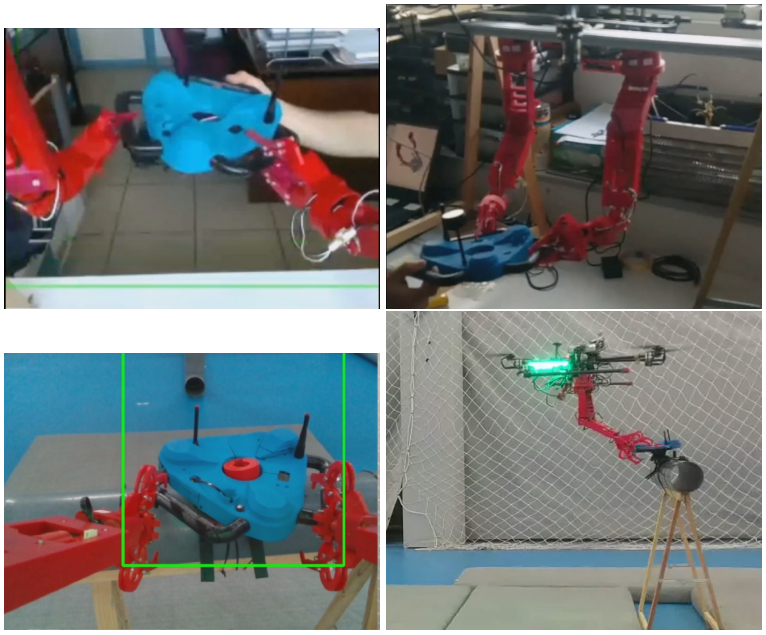


Figure 2.29 Complete system running. Top pictures shows the system in a test-bench and bottom pictures shows it running in a real autonomous flying experiment.

3 Object detection and localization for aerial manipulation

3.1 Introduction

Humans and animals have an inherent ability to identify and locate instances of objects to be able to interact with them. However, this simple skill implies a set of subprocesses that are not intuitive for machines. Robust and reliable object detection and recognition, as well as full pose estimation in any scene, are critical tasks to achieve robotic manipulation. Furthermore, contrary to the ground or fixed manipulators, aerial manipulation implies extra robustness and speed to be able to produce more agile movements.

Both object detection and tracking have been tackled with a wide range of techniques. Some simpler straight algorithms use a fixed set of markers to accurately locate objects by placing these tags stuck onto the surface [85, 86]. However, this methodology limits the applications and is poorly scalable.

Some techniques are based on detecting the object using Euclidean distances between the objects [87, 88]. Most frequent algorithms apply feature detectors and descriptors to generate sparse descriptive models that are identified later in any scene [42][43]. Finally, many current researchers in this area focus on using Deep Learning techniques for object detection [89, 90], object tracking by recurrent networks [91] and even object pose estimation [92].

This chapter introduces techniques to tackle the problem of object detection and localization for aerial manipulation tasks. The chapter is divided in six sections organized according to the typology of the algorithms. Section 3.2 shows the implementation and results of a distance-based object detection and localization algorithm. Section 3.3 shows how to use dense point clouds with ICP to detect objects in scenes composed of dense point clouds. Section 3.4 shows an algorithm to create sparse models of objects using feature clouds and how to detect and locate them efficiently. In Section 3.5 a probabilistic algorithm is described to perform multi-object detection and localization under the

framework of Gaussian Processes Implicit Surfaces. Finally, Section 3.6 shows different machine learning algorithm for object detection using monocular cameras.

3.2 Distance based object detection

Some of the more basic algorithms for object detection are the geometric or distance-based algorithms. These algorithms take the 3D information given by the robot's sensors to create clusters and locate the objects. The information used in these algorithms is diverse. Not only 3D geometric distances are used but also color information, so-called color space; or further adding curvature data to the 3D geometric information.

A well-known algorithm is K-means [93]. This algorithm aims to partition a set of observations into a defined number of clusters according to the nearest mean of the cluster. More formally, each cluster has a *Normal* distribution associated to evaluate the membership of each data point to a cluster, which is a very intensive problem. Nevertheless, several heuristic algorithms produce good results to guarantee the convergence to a local minimum by iterative processes.

The primary challenge of this algorithm is its initialization, i.e., selecting the right amount of clusters and their initial parameters. Authors in [94] propose an efficient algorithm for selecting the initial parameters of the centroids by sampling a weighted distribution which segregates the initial clusters by maximizing their distance. An extension of the k-means algorithm, called X-mean algorithm [95], was developed to update the number of clusters iteratively. This improvement is of high importance as in some situations it is not possible to know a priori the number of existing clusters.

Another object detection algorithm based on Euclidean distances is shown in [88]. This algorithm uses an efficient representation of the space using a Kd-tree and uses an approximate nearest neighbors method to measure the distance between the points. Then the algorithm iteratively grows clusters of points that are close in range and ends when the complete point cloud has been explored.

The author of this thesis used this technique to segment candidate objects in [11]. The remainder of this section introduces it. In that work, the authors focus on using cheap stereo cameras. These pair of images are used to compute a sparse 3D point clouds. The data was obtained with flights of a real aerial robot carrying the stereo system. Section 4.2 will explain with more detail the elaboration process of the point clouds and the creation of a map representing the environment by the consecutive alignment of the point clouds.

An example of input point cloud from the system is shown in Figure 3.1. This input cloud contains points that belongs to objects, the background, and even noise points that do not exist. Thus, the input cloud needs to be filtered.

At first instance, the noise is removed using and standard outlier removal [16]. Then the floor is subtracted from the point cloud using a RANSAC algorithm [96] which finds the best set of points in the point cloud that match a planar model, i.e., the floor.

The resulting cloud is free of noise points and the points that are assumed to belong to the floor. Then the Euclidean-based distance segmentation algorithm is used to get a list of candidate clusters that are assumed to be objects. Finally, clusters that meet the minimum number of points required are selected as candidate objects. The idea behind this is that a

Figure 3.1 Input cloud and corresponding image of the scene for the detection of objects.

tight cluster of features that is not part of the floor could represent an object, assuming the objects are close to each other. Figure 3.2 shows the clusters obtained from an experiment.

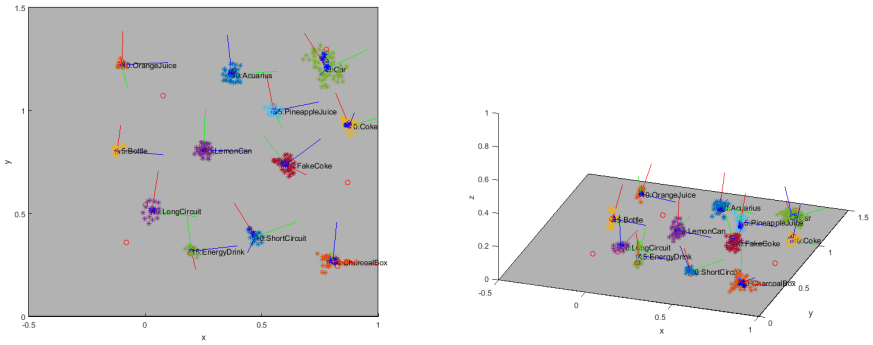


Figure 3.2 Reconstruction of the objects in the scene (colored clusters) compared to ground truth (red circles). The frame in each object represents the PCA result with the red axis representing the dominant axis. .

After the candidates are extracted and clustered, as shown in Figure 3.2, their points are projected onto the images from the cameras. The projected points correspond to their respective objects as can be seen in Figure 3.3.

In order to do so, it is necessary to know the position of the UAV in the 3D map and the calibration of the cameras. Let C be a cluster of points belonging to an object, K the calibration matrix of the camera, and T the pose of the camera with respect to the map, being

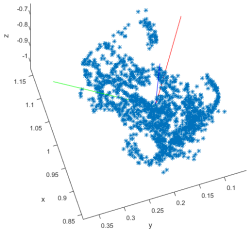
Figure 3.3 Left and right images with projection of the points belonging to candidate objects, surrounded by a convex hull. .

$$C = \{p_i \forall i = 1...N\} \quad K = \begin{bmatrix} f_x & 0 & c_X \\ 0 & f_y & c_y \\ 0 & 0 & s \end{bmatrix} \qquad T = \begin{bmatrix} R|T \end{bmatrix}$$

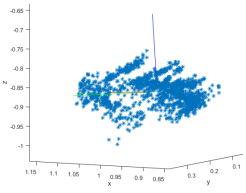
the projection of every single point of the cluster using a pinhole model of the camera

$$\begin{bmatrix} x_{im} \\ y_{im} \\ 1 \end{bmatrix} = K \cdot T \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.1}$$

Once all the points are projected onto the image, a convex hull that envelops the object is generated, for each cluster of 2D points. These 2D clusters can be used later in an object classification algorithm. Particularly, subsection 3.6.1 introduces the algorithm used to classify the objects from the footprint in the image. Figure 3.4 shows a close up view of an extracted object.



(a) Top view of object's cloud.



(b) Side view of object's cloud.

(c) Sample picture of real object.

Figure 3.4 Example of an extracted objec.

3.3 Object detection by dense point cloud alignment

Point cloud model for objects was introduced previously in Section 2.2.1. Detecting objects using this model typically implies an alignment process. 3D point cloud alignment consists of finding the right 6DoF rigid transformation existing between two point clouds in some coordinate system. A large variety of algorithms exists [97] to solve the alignment problem using different properties and methodologies. Some of these algorithms are even able to handle non-rigid deformations [98] of the point clouds during the alignment process. This section will focus on the alignment of point clouds using ICP and particularly a variant called generalized-Iterative closest points or gICP [99]. Additionally, color information has been added to the process to make the alignment process more robust. This section focuses on the work developed in article [13] for the detection of objects using dense point clouds.

The alignment problem usually involves two point sets, the target which is considered static and is used as a reference for the second point set, the source. Let's denote them $T = \{p_i = [x_i, y_i, z_i] \forall i = 1...N\}$ and $S = \{p_j = [x_j, y_j, z_j] \forall j = 1...M\}$. At first instance the algorithm seeks for possible matching candidates. It means, for each point in the source cloud, creating a list with the closest points in the target cloud.

Once each point in the source cloud is paired with the target cloud, a set of filters is used to reject possible bad matches. Simplest ICP algorithm just looks for distance correspondences between the points. Other versions use some kind of local information in the point clouds to seek for lines, planes or any other feature that can be used to reject false matches between points, making the process more robust to local minima. The presented implementation uses following correspondence rejectors: maximum distance (Eq.3.2), normal compatibility (Eq.3.3), a custom implementation of color rejector using HSV (Eq.3.4), and a one to one correspondence rejector.

$$s_{dist} = min(\sqrt{(x_i - \hat{x}_j)^2}) \tag{3.2}$$

$$s_{norm} = min(cos(\angle(n_i, \hat{n_j}))) \tag{3.3}$$

$$s_{color} = min(\sqrt{(r_i - \hat{r}_j)^2} + \sqrt{(g_i - \hat{g}_j)^2} + \sqrt{(b_i - \hat{b}_j)^2}) \tag{3.4}$$

As a brief introduction, a set of examples are shown in the following paragraph. Given the two sets of point clouds and their matches (typically named correspondences), ICP tries to iteratively performs small steps to align both sets of points by looking for correspondences between the clouds.

ICP alignment algorithms strictly need to have an initial estimation because they are susceptible to fall in local minimums. Moreover, if the algorithm is used to locate individual objects. This happens due to the vast amount of redundant data, which makes possible that the source cloud fit to several locations in the target cloud. Figure 3.5 shows the results of the ICP algorithm for object detection. Left figure show a good result when the estimation is close to the right position. Right figure shows a bad result due to the bad initial estimation.

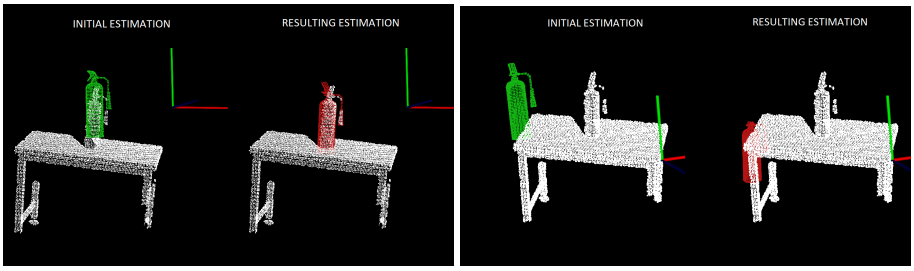


Figure 3.5 ICP algorithms are sensitive to the initial estimation of cloud's pose. Figures show two example of object alignment with different initial locations for the same object on a scene. Left picture shows a good initialization and right picture a bad initialization. .

The rest of this section shows the results of the object detection and location algorithm using dense point clouds developed in [13]. Particularly, the model of the object to be detected corresponds to a mock-up of a crawler robot that is part of the European Project AEROARMS [7]. Figure 3.6 shows an instant of the algorithm in which the crawler robot is detected in outdoor environments on the top of a fragment of a pipe.

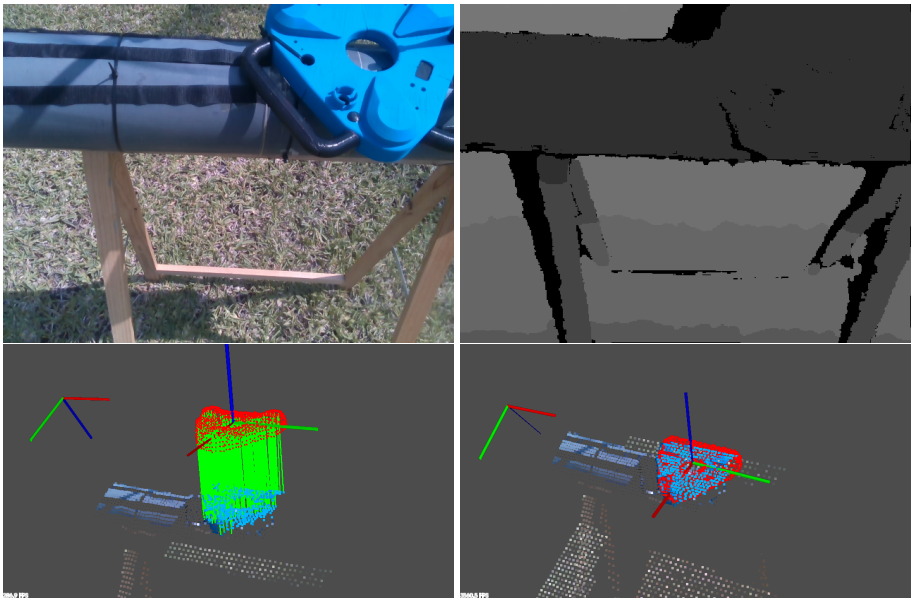


Figure 3.6 Top figures show the a pair of RGB-D data from a RealSense D435 device. Bottom left figure shows the final matches between a dense 3d point cloud model of the case of the crawler and bottom right figure shows the resulting alignment. The coordinate system of on the point cloud represent the final position of the robot in the coordinate system of the camera.

The complete fitting algorithm is stated in algorithm 2. The algorithm is complemented with an Extended Kalman Filter (EKF) to smooth the results of the alignment and to predicts next possible position of the object to guarantee the convergence.

Algorithm 2 Object fitting with EKF estimator for orientation consistency with objects with symmetries

```
1: if initialized? then
2:     EKF ← previous_estimate
3:     pose ← predict_current()
4:     refine_estimate(plose, input_cloud, model)
5: else
6:     initial_estimate()
7:     init_ekf()
8: end if
```

The complete application is described in the article referenced above. The first part of it consists on a convolutional neuronal network (CNN) which detects the crawler in the RGB image providing a guest of objects pose. This algorithm is introduced later in Section 3.6.3. This particular object has three symmetry planes which can produce a twist in the estimation of 60 deg. In order to prevent this problem, once the initial estimate has been computed, the EKF is used to have the initial estimation of the position and the rotation, giving a better initial condition for the refining alignment algorithm and preventing undesired twists.

The system works in UAV's coordinate system. It implies that even if the target object is static respect to the scene, it moves relative to the robot, due to the movement of the UAV. It is assumed that the relative pose has a simple kinematic movement,

$$X = X_{k-1} + \dot{X}_{k-1} \cdot \Delta t + \frac{1}{2}\ddot{X}_{k-1} \cdot \Delta t$$

and that the observation equation is the result of the alignment algorithm being

$$Z_k = \hat{X}_k$$

Figure 3.7 shows the result of object's pose estimation algorithm. This position estimation is used later to grasp the object as described in Section 2.

The use of the Kalman filter and the refinement of the ICP provide high accurate results of the position of the object to be grasped. Additionally, it is computed in real time, which is critical for the right execution of the manipulation. Section 3.6.3 will introduce an object detection algorithm for 2D images which provides in which fragment of the image is the object located. Providing this candidate region enhances the speed of the algorithm as it reduces the size of the point cloud which is used for the alignment process.

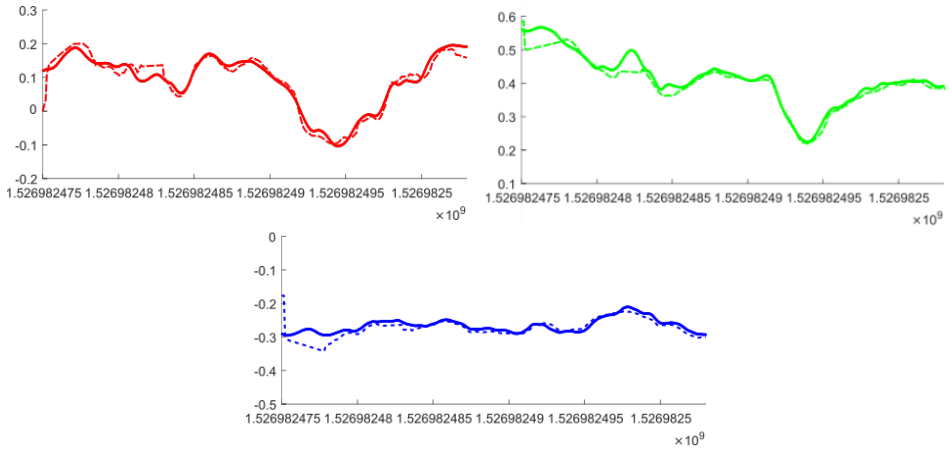


Figure 3.7 Errors in cartesian coordinates of object's pose estimation using point cloud model with ICP and EKF pose filtering.

3.4 Feature-based object detection

Feature-based point clouds, as introduced in 2.2.2, are sparse point cloud in which each point contains local geometrical information about its vicinity. Featured clouds are very convenient as their size is hugely lower than the ones used in ICP algorithms and they contain reliable information that overcomes this shortage of points.

This section describes the implementation of a feature-based object detection algorithm developed in [8]. This algorithm is divided into two stages: the learning stage (or offline stage), in which the model of the object is created, and the localization stage (or online stage), in which the object is detected and its position is estimated. Figure 3.8 summarizes the pipeline of the algorithm in both the modeling and localization stages.

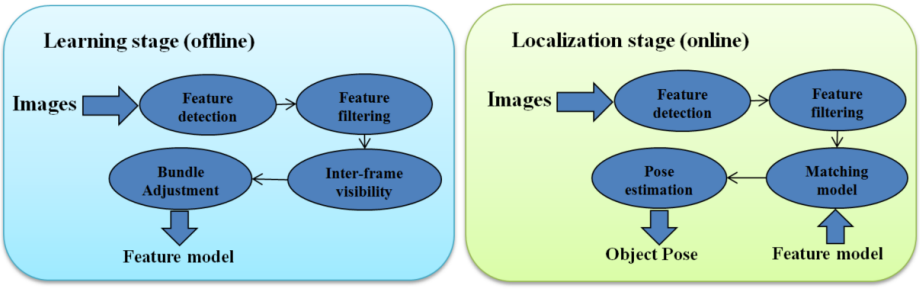


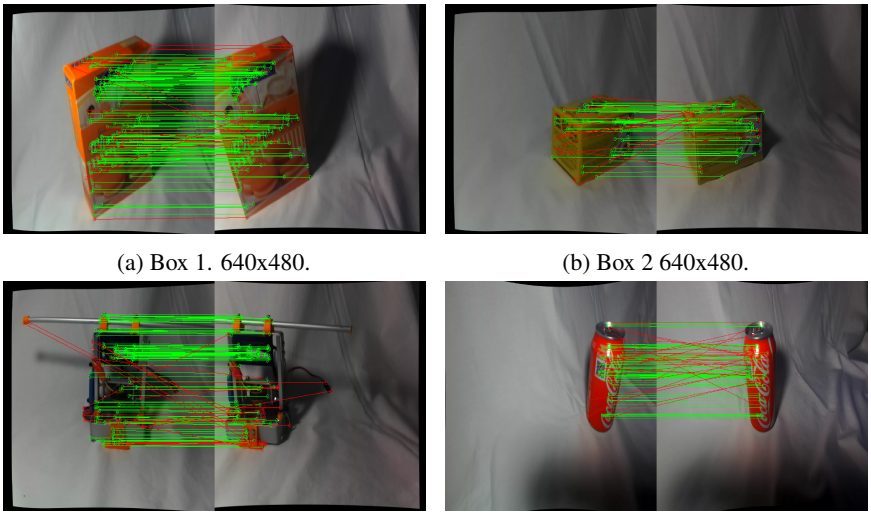
Figure 3.8 Pipeline of both stages of the algorithm..

The learning stage, or offline stage, generates a model of an object from a set of images. The presented method differs from previous approaches by the use of a stereo camera [42, 100], which automates the learning process (providing 3D information of the real-world

scale) and improving the filtering of outliers. Particularly, a ZED stereo camera from StereoLabs[1] has been used.

Section 2.2.2 introduced some of the most used state-of-art two-dimensional features. In this work, the performance between FAST-SIFT and ORB are evaluated, and quantitative results have been provided to show the efficiency of the system in different situations. The performance of the image feature detection and matching for the model creation and object position estimation depends on the combination of the detector and descriptor chosen. Nevertheless, for both the object modeling and detection algorithms, the features are entirely exchangeable. Later in this section, the processing time for the different combinations of detectors and descriptors is shown.

As mentioned before, a ZED stereo camera is used for image acquisition. This camera has a wide-angle lens, so a rectification of the images is needed to detect and match the features correctly. Initially, features are detected on each pair of images. Then these features are matched with either a FLANN [30] (for SIFT) matcher or a force brute matcher (for ORB). The first advantage of using stereo images is to improve the filtering of matches, making it more robust to outliers. Despite, it needs more processing time as features are computed in both images. The resulting inliers are assumed to be key features of the object. This makes the algorithm more robust to outliers as these are rejected at the beginning of the process. Figure 3.9 shows examples of feature filtering using stereo, as described in this paragraph.



(a) Box 1. 640x480.

(b) Box 2 640x480.

(c) Tool 640x480.

(d) Can 640x480.

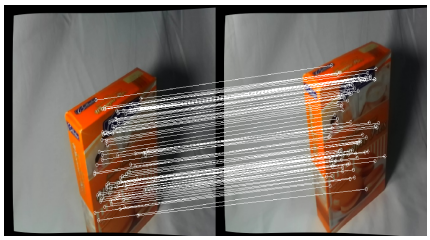
Figure 3.9 Filtering bad features using known stereo geometry and sequential filtering..

The last step of the object modeling is to perform a bundle adjustment (BA) [101] with the features and camera positions to reconstruct the right shape of the object. The camera

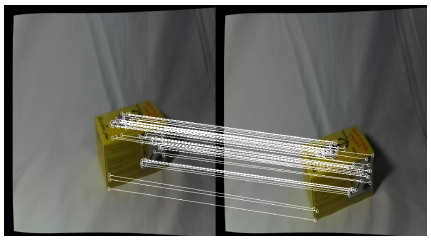
[1] *https://www.stereolabs.com/*

positions where the images were taken are also obtained. However, for the proposed method this information is not used. The BA consists of a global optimization using Levenberg & Marquardt [102] algorithm to minimize the re-projection errors on feature points in a set of images.

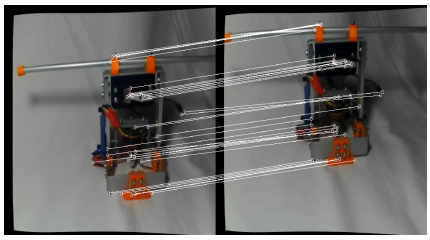
In order to perform the BA, it is necessary to correlate the points within all the images. It is assumed that all the pictures from the dataset are arranged as they were captured. Then, the features are matched sequentially to obtain the inter-frame visibility of the features. With this step, the relations within sequential frames are obtained. Figure 3.10 shows consecutive matches of features for the creation of the inter-frame visibility matrix for different objects.



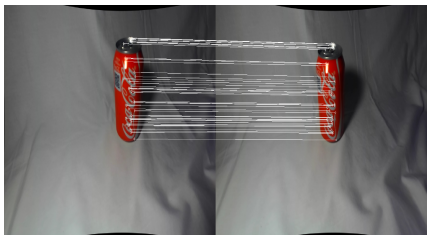
(a) Whoopies box. 640x480.



(b) Gena box 640x480.



(c) Drilling tool 640x480.



(d) can 1280x720.

Figure 3.10 Features are associated sequentially to compute the inter-frame visibility of the features. Then, following algorithm 3, the visibility between non-sequential frames is computed..

Additionally, it is necessary to obtain the rest of the relations within all the frames. Let us denote $P = \{p_k \forall k = 1...K\}$ a vector in which each p_k is a vector with the features on the frame k; M a matrix that at each element m_{ij} contains a vector with the matches between the frame i and the frame j. All the elements $m_{i(i+1)}$ are filled from the sequential match of frames. Then the rest of elements of m_{ij} above the diagonal, i.e., $j > i$ can be filled using algorithm 3. An improvement can be made by detecting loop closure; however, for most of the cases, this method is enough to reconstruct the object, as the camera is always pointing to the same place. A diagram of this visibility problem is shown in Figure 3.11

Consider a set of N 3D points, observed from K cameras (at T_k position and R_k orientation). Then, given the correlations between the projections of the 3d points into the cameras, an optimization is performed minimizing the errors. Using the previously defined matrix M of matches between frames makes it possible to generate a unique list of 3D

Algorithm 3 Correlate back matches

```
for offset = 2, offset < K do
    for i = 0, i < K − offset do
        j = i + offset
        for match in m_{i(j−1)} do
            if match is visible in m_{(j−1)j} then
                add match in m_i j
            end if
        end for
    end for
end for
```

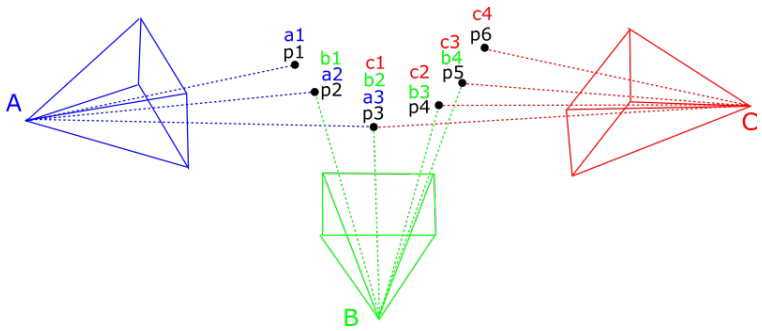


Figure 3.11 Diagram of elements in the Bundle Adjustment problem. A, B and C represent the position of the camera from where the observations were taken. $p_i \forall i = 1...6$ are six features in the space and $a_i, b_i and c_i$ are the features observed by each of the positions..

points and the inter-visibility of the points within the frames. Gordon and Lowe [42] showed that placing all the cameras at the same distance from the origin on the Z-axis and all the projections in the XY-plane are enough conditions to ensure the convergence of the BA. It is important to highlight that it is necessary to keep track of the descriptors of the features, as they need to be stored with the 3D points as part of the model of the object. Figure 3.12 shows how a model has iteratively been constructed using the BA.

Moreover, since the inter-visibility matrix was obtained, it is possible to compute the number of times that each point appears, i.e., in how many images each point is observed. Therefore before computing the BA, an additional filtering can be done to improve the performance.

Some of the features can be badly matched or just not matched. Hence, this could produce duplicated points that might complicate the convergence of the algorithm. To avoid this, removing points that appear in less than k images can improve and speed up the

(a) Starting state. (b) First iteration. (c) Iteration 5. (d) Iteration 20.

Figure 3.12 Features are associated sequentially to compute the inter-frame visibility of the features. Then, following algorithm 3, the visibility between non-sequential frames is computed..

convergence of the BA. The minimum value for k is 2, as the points need to appear in at least two images to be able to triangulate it. On the other hand, increasing this parameter too much is not possible, because the BA solver might not be able to solve the problem if the number of observed variables is lower than the number of variables in the problem. Therefore, in this work, k is set to 3.

Once the BA process is performed, a 3D model of the object is obtained. However, as described by in [103], because of the optimization algorithm, the points are not scaled according to the real size. Authors in [103] record an additional dataset in which the position and orientation of the object are known, then a second optimization algorithm is performed to obtain the right scale of the model.

In contrast, in the proposed method, with the use of stereo cameras, this new dataset is not needed. As the correlation of all the points is known, it is possible to get the projections on both the left and right images at each frame. P^m are the points obtained from the BA and P^t is a cloud reconstructed from features of a frame k using the known stereo geometry. Therefore, it is possible to estimate the transformation T between them using an SVD-based estimator. If k is the current frame, N_k is the number of points seen on that frame, p_i^m is the point i on the model and p_i^t is the triangulated point from the stereo pair; the score of each transformation is computed as

$$score = \sum_{i=1...N_k} (\|p_i^m - p_i^t\|)/N_k$$

The transformation T which produces the minimum score is used to scale the model to the real-world size. Being

$$T = \begin{bmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

the scale factor can be computed as

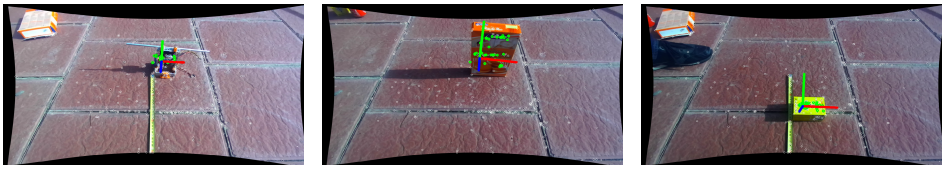
$$s = [s_x, s_y, s_z] = \begin{cases} s_x = \|[a_{11}, a_{21}, a_{31}]\| \\ s_y = \|[a_{11}, a_{21}, a_{31}]\| \\ s_z = \|[a_{11}, a_{21}, a_{31}]\| \end{cases}$$

As the model of the object is sparse, it is hard to compute information about how to grasp it reliably. But it can be set manually to ensure a correct manipulation. As the object modeling is performed offline, it is realistic to choose it manually at this stage. Nevertheless, the detection of the grasping points can be analyzed depending on the manipulator and the geometry of the object using various quality metrics [59] if some extra information is given about its shape.

The remainder of this section introduces the online detection of the object in new images and the position estimation. First of all, using the camera calibration, the acquired images are undistorted. The same feature detector and descriptor as the one used in the modeling stage is used to extract features in both input images. Then, the features in the pair of images are matched. As described before, the known parameters of the stereo calibration are used to filter the outliers. Hence, the remaining points are stronger as they appear in both cameras and they are easy to match.

At this point, there is a set of point candidates on the scene that becomes part of the object. To detect it and estimate its position a PnP (or Perspective-n-Points) formulation is used. Let $P = \{[x_i, y_i, z_i], \forall i = 1...N\}$ be a set of 3d points and $U = \{[u_i, v_i], \forall i = 1...N\}$ be their projection on the camera plane. The objective is to find the Rotation R and translation T of the object in the camera's coordinates (knowing the calibration parameters of the camera) by minimizing the re-projection error of the points. Particularly, a RANSAC implementation is used, which is less sensitive to local minima and more robust to outliers.

To be able to start the PnP problem, it is necessary to match the features in the scene with the feature model of the object. In order to do that, each descriptor is matched with the points in the scene and then filtered to remove outliers. The inliers are used in the PnP problem to detect the position of the object. Figure 3.13, shows screen-shots of results outdoors with a featured floor.



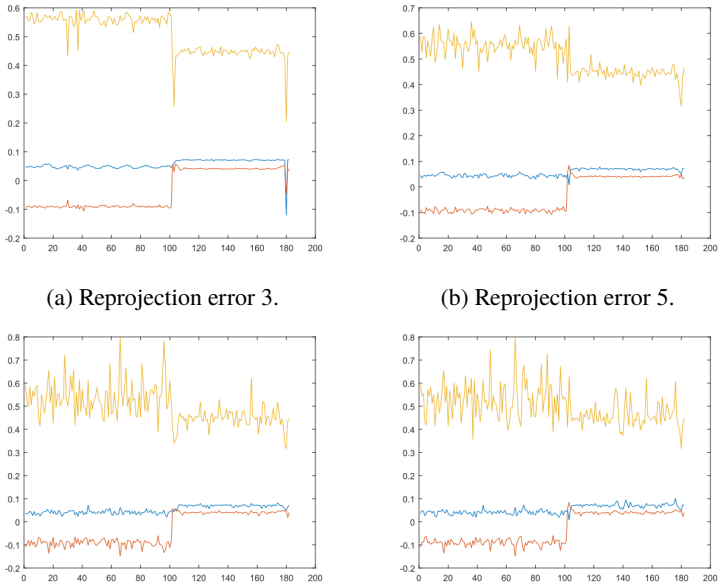
(a) Drilling tool. (b) Box 1. (c) Box 2.

Figure 3.13 Examples of detection and position estimation of objects outdoors. White thin circles are candidate features in the scene. Green thick circles are the features assigned to the object and the coordinate system is the representation of the position of the object. It depends on the coordinate system chosen at the modeling stage.

The PnP process was analyzed regarding the confidence parameter and the reprojection error parameter. These two parameters affect the performance of the algorithm in both time and pose estimation. To give a numerical idea of the influence of the parameters, table 3.1 summarizes the average time for the algorithm varying the parameters using FAST and SIFT. Figure 3.14 shows the estimated position of an object varying the reprojection parameter. It can be seen that the estimation on the z-axis is worst when the reprojection error increases.

		reprojection error			
		3	5	7	8
conf.	0.99	0.031	0.028	0.025	0.024
	0.999	0.036	0.031	0.027	0.026
	0.9999	0.039	0.034	0.026	0.028

Table 3.1 PnP problem times.



(a) Reprojection error 3.

(b) Reprojection error 5.

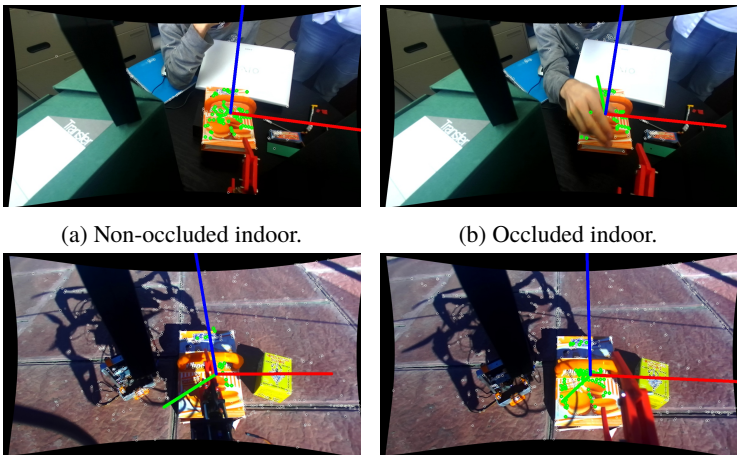
(c) Reprojection error 7.

(d) Reprojection error 8.

Figure 3.14 Result of pose estimation algorithm varying the reprojection error. Due to the increase of this parameter, the position is less accurate. However, as described in table 3.1, it is slightly faster.

The algorithm is also proved to be robust to occlusions. The position of the object can be reconstructed with a small fraction of points of the model. Subfigures 3.15a, 3.15b, show the estimated position of an object occluding partially by a person. Also, in Subfigures 3.15c, 3.15d one can see how the position of the object remains stable even with the

robotic arm occluding the object during the grasping trajectory. It is noticeable that the position is more stable than the orientation against partial occlusions.



(a) Non-occluded indoor. (b) Occluded indoor.

(c) Non-occluded outdoor. (d) Occluded outdoor.

Figure 3.15 Testing detection and pose estimation against partial occlusions..

It was observed that the use of a FAST detector and SIFT descriptors produced the best results. In the learning stage, these features produced accurate models. Subsequently, the position estimation was recovered, in both indoor and outdoor environments, easier than with the other feature descriptors. However, the computation time of this descriptor is too high. In simple scenes, a frame speed within 8 and 13 is obtained. However, outdoors, due to the light conditions and the texture of the floor, the FPS decreased drastically within 2 and 4 FPS. This happens due to the large increment of features detected by FAST. This fact enlarges the number of descriptors that need to be computed and the RANSAC PnP solver takes longer to converge.

The second best option is to use FAST detector and rBRIEF descriptor. The size of rBRIEF descriptors is smaller than SIFT ones, so the time spent by the matching algorithm is lower too. For this pair detector-descriptor 15-25 FPS were achieved. However, this descriptor showed the worst behavior against variations in the scale.

This algorithm has been intensively tested in several environments with different objects. Videos [2] [3] shows the summary results submitted to [8]. Additionally, the algorithm has been used in the project AEROARMS to enable UAVs to perform aerial manipulation tasks[4] [5] [6].

[2] *https://youtu.be/P5krV80lAdk*
[3] *https://youtu.be/eL2mBwhQxtA*
[4] *https://www.youtube.com/watch?v=G4kDpBnqGH0*
[5] *https://www.youtube.com/watch?v=H6CzL5ZJmpM*
[6] *https://www.youtube.com/watch?v=XVi5ZGTlFTM*

3.5 Object detection and location using probabilistic model

As presented in Chapter 2, probabilistic models are an alternative to deterministic representations of objects in robotics. Particularly, probabilistic representations of point clouds are essential for active methods such as targeted exploration [104], decision making in grasping contexts [105] or probabilistic object detection [106]. In this section, the object detection and location is formulated as a data association problem, where each point n is assigned a label a_n, associating it to one of a set of objects present in the scene. Particularly, this work has been published in [12] and the target objects to be detected are fruits for harvesting which is an interesting manipulation problem in robotics.

The surface of fruits are modeled by Gaussian process implicit surfaces [47], where observations of object surfaces are interpreted as points in the zero-level set of an underlying GP. Let $\mathrm{D} = \{x_n, y_n\}$ for $n = 1, \cdots, N$, be a set of observations y_n of $f(x_n)$, characterized by the covariance function $k_{\mathrm{N}}(x_m, x_n)$. Observations y^* of $f^* = f(x^*)$ at a query location x^* are distributed according to

$$y^* \sim \mathrm{N}(\langle y^* \rangle, \sigma^{*2}), \tag{3.5}$$

with mean and variance

$$\langle y^* \rangle = \mu_{y_D} + k^{*T} K^{-1} y_{\mathrm{D}}, \tag{3.6}$$

$$\sigma^{*2} = k^{**} - k^{*T} K^{-1} k^*, \tag{3.7}$$

where $y_{\mathrm{D}} \in \mathbb{R}^{N \times 1}$ denotes the concatenation of the observations for all data points, and $K \in \mathbb{R}^{N \times N}, k^* \in \mathbb{R}^{N \times 1}, k^{**} \in \mathbb{R}$ denotes covariance terms. Further information can be found in [47, 107, 108].

This section takes into account the possible existence of multiples objects, so mixtures of GPs are considered instead of a single instance. The association of data points to different GPs is modeled by a Dirichlet process (DP). DP mixture models have the advantage that they readily deal with data sets where the number of latent clusters is not known. Each GP is then characterized by a set of hyperparameters, which can be used to represent the surface properties as well as prior shape and location [107].

Analytical representations of the resulting probability space are intractable, even for reasonably small problems, as it consists of all possible combinations of part associations. As a result, Markov-Chain Monte Carlo method is employed to sequentially explore the probability space by producing samples according to a Gibbs-sampler.

Gibbs sampling is a Markov Chain Monte Carlo (MCMC) technique commonly used to make statistical inference. *Monte Carlo* term refers to algorithms that involve performing simulations using probabilistic choices. *Markov Chain algorithms* are a class of algorithms for sampling from a probability distribution based on the construction of a Markov chain, which is a random process that undergoes transitions from one step to another. Let z^k be a set of states. It is desired a function g that makes a probabilistic choice to go from one state to another. That choice is done according to a transition probability $P_{trans}(z^{(k+1)}|z^{(0)}, z^{(1)} \ldots z^{(k)})$. Performing these transitions is called probabilistic walking

through the states. A Markov Chain Monte Carlo algorithm is defined so that the next state z^{k+1} only depends on the current state z^k,

$$P_{trans}(z^{(k+1)}|z^{(0)},z^{(1)}...z^{(k)}) = P_{trans}(z^{(k+1)}|z^{(k)}) \tag{3.8}$$

To apply Gibbs sampling the state z need to have two or more dimensions, i.e., $z = \{z_1, z_2, ..., z_D\}$ *with* $D > 1$. The basic idea is that, rather than picking the next state completely, the probabilistic choice is split for each of the $d \in D$ dimensions, where each choice depends on the other $D-1$ dimensions. The walk proceeds as follow:

1: $z^{(0)} := \{z_1^{(0)}, z_2^{(0)}, ..., z_D^{(0)}\}$
2: **for** $t = 1 to T$ **do**
3: **for** $d = 1 to D$ **do**
4: $z_d^{t+1} \sim P(z_d|z_1^{(t+1)},...,z_{d-1}^{(t+1)},z_{d+1}^{(t)},...,z_D^{(t)})$
5: **end for**
6: **end for**

The conditional distribution can be obtained using its definition as:

$$P(z_d|z_1^{(t+1)},...,z_{d-1}^{(t+1)},z_{d+1}^{(t)},...,z_D^{(t)}) = \frac{z_1^{(t+1)},...,z_{d-1}^{(t+1)},z_d^{(t)},z_{d+1}^{(t)},...,z_D^{(t)})}{z_1^{(t+1)},...,z_{d-1}^{(t+1)},z_{d+1}^{(t)},...,z_D^{(t)})} \tag{3.9}$$

It is known that samples from a DP can be generated by sampling from the so called Chinese Restaurant Process (CRP): the prior distribution of a data point's association is conditioned on the association of all other data points and is given by

$$p(z_n = k|\mathbf{z}_{\setminus n}, \alpha) = \frac{N_k}{N-1+\alpha} \tag{3.10}$$

$$p(z_n = K+1|\mathbf{z}_{\setminus n}, \alpha) = \frac{\alpha}{N-1+\alpha}, \tag{3.11}$$

where a_n denotes the association of point n, $\mathbf{z}_{\setminus n}$ denotes the association vector for all other points, N denotes the total number of points (including point n), N_k denotes the number of points associated to component K, and α denotes the Dirichlet process concentration parameter. The second expression $p(z_n = K+1)$ represents the probability that point n is associated to a new object without any points associated. The algorithm randomly selects a data point $\{\mathbf{x}_n, y_n\} \in \mathscr{D}_M$ and removes it from its current GP, before computing the likelihoods with respect to all GPs currently present in the scene, i.e., with a number of associated data points $N_k > 0$. The GP likelihoods are weighted by N_k according to (3.10) and (3.11) and used to compute posterior association probabilities $p(z_n = k)$ by normalisation. Our algorithm produces a desired number of samples for the scene segmentation, which can be used to draw probabilistic conclusions about the scene segmentation and the location and shape of objects.

The previous concepts are extended to the problem of reconstructing multiple objects in cluttered scenes. This work aims to exploit the prior object shapes described in Sec-

tion 2.2.4, and the compactness of objects for a probabilistic association of point cloud observation to different objects.

Let $\mathscr{D} = \{x_n, y_n; \forall n = 1...N\}$ be a set of observations (called parts), $\mathscr{K}$ denotes the number objects in the scene and $A = [A_1,...,A_N]$ the association of the parts to the objects. Each object k is characterized for its prior parameters θ_k.

It is assumed that the number of objects present in the scene is not known a priori and the analytical representation of the scene is intractable. The association probabilities A_n are modeled using a Dirichlet Process Mixture Model (DPMM) which accounts that the number of objects is not known a priori. In order to solve the problem, MCMC methods can be used to generate samples. The previously described Gibbs-sampling method is used, which has been applied to similar problems. The whole process is summarized in algorithm 4.

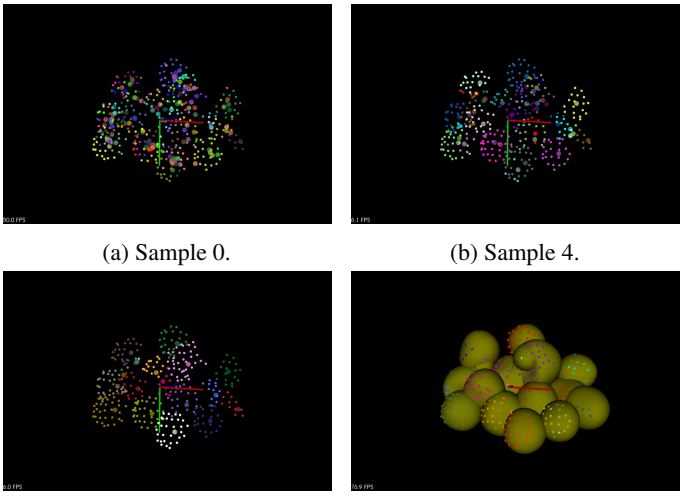
Algorithm 4 Multi-Object Sampler

```
1:  init prior data
2:  for all samples do
3:      for each part n do
4:          remove part from associated object
5:          for each object k do
6:              compute association probability
7:          end for
8:          sample association A_n
9:          if object exist then
10:             add part to object
11:         else
12:             create new object
13:         end if
14:     end for
15:     for each object k do
16:         if has parts associated then
17:             sample prior parameters θ_k
18:         else
19:             remove object
20:         end if
21:     end for
22: end for
```

Given an initial state of the system, the sampler first iterates over the parts. For each of the N parts, it is computed the probability of belonging to an existing object. The sample for a new association of the part is done according to a Chinese Restaurant Process (CRP) which weights the different objects with a Dirichlet concentration parameter α. Consequently, there is a possibility that the part is associated to a new object instead of an existing one. One of the parts are updated, all the object without parts associated are removed. Then the algorithm cycles through all the objects updating its parameters. The update of parameters is performed sampling according to the current state. There is not any

limitation about how that sample is performed. Currently, two different kinds of samplers are considered, 1) constructing a normal distribution using the part information; 2) using the Metropolis-Hastings (MH) algorithm [109].

Figure 3.16 shows samples of the described multiobject sampler in a scene with multiple fruits. Only spherical prior was used with radius fixed at 0.3 m. The last figure also shows the representation of the mean of the underlying GP process. It means as described before representing a 3D surfaces. Usually, plotting an iso-level surface of the GP is expensive. For that reason, it was developed a quick plotter using a seed and spread algorithm which has been implemented in the library described in Appendix A.2.



(a) Sample 0. (b) Sample 4.

(c) Sample 60. (d) Sample 100.

Figure 3.16 Random samples and Zero-level of GPIS after one hundred samples for a scene with cluttered fruits and vegetables. The algorithm is initialized with the same number of objects than parts. The last picture shows one of the samples with the reconstructed GPIS objects.

3.5.1 Multiview probabilistic object detection and location

This algorithm has been tested for the probabilistic segmentation of apple vines for harvesting. The vine-like structure of apple trees on a trellis comprises leaves, branches, and fruit. The leaves can occlude the fruit and also induce noise in the observed point clouds. For these reasons, the input point clouds are firstly filtered with standard noise removal. At this stage, the clouds have fragments of the apple, but morphologically these are identical to leaves since both are small planar surfaces.

In order to achieve better results in the probabilistic segmentation a set of point clouds are taken from multiple points of view to build up a 3D reconstruction of the apple vine using a registration algorithm. A pose planner generates camera poses such that the scene is observed from different angles. Figure 3.17 summarises the multiview probabilistic segmentation process. It is assumed that the robot is positioned in front of the apple

branches. First, a snapshot is taken and used to infer the relative angle of the apple "vine" with respect to the arm pose. Using this angle, the algorithm computes candidate poses reachable by the arm, and these are arranged on a portion of a sphere. Then, at each step of the algorithm, a pose is selected that maximizes the distance from the history of past poses granting a good convergence of the mapping process. From each viewpoint, a new cloud is taken, which is registered using ICP plus information from the arm controller that provides an initial estimate of the point cloud's exact pose.

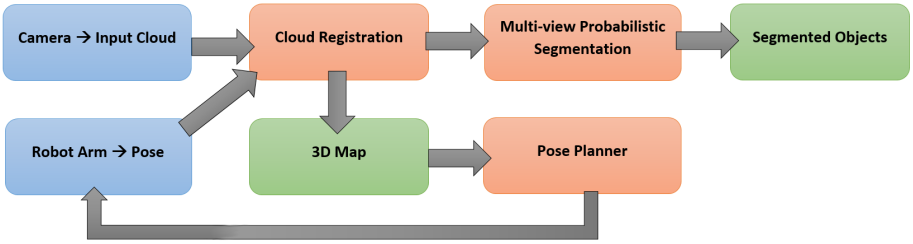


Figure 3.17 System block diagram.

Finally, the map of the scene is filtered by color using the Euclidean distance between colors and a model value in HSV. Then, it is simplified using the voxel cloud segmentation algorithm described in [58] before being introduced into the Dirichlet segmentation process.

The artificial apple trellis has a dimension of 1.5m by 1.5m with the same characteristics as typical orchard trellises. The initial position of the arm with respect to the trellis is unknown at the beginning of each experiment. The only assumption is that the trellis is in the range of the camera (the Intel RealSense SR300 has a maximum range of 1.5m).

The first row in Figure 3.18 shows the input cloud that is fed into the Dirichlet segmentation algorithm. In the second row, the result of the segmentation process is shown, where different colors are used to show the association to the different objects.

Finally, Figure 3.19 shows additional results of the probabilistic segmentation with different configurations and clusters of apples. All the apple vines were reconstructed entirely, and all the apples were successfully segmented using 1000 iterations of the Gibbs Sampler. The centroids of the apples are distributed over the samples within a range of 3 *cm*. This range depends on the parameters of the sampler, and a specific centroid depends on the selected iteration step, as these are updated sequentially to ensure convergence. The surfaces of the apples fit well to the data, demonstrating accurate reconstruction close to the observed data points, as required for harvesting and manipulation.

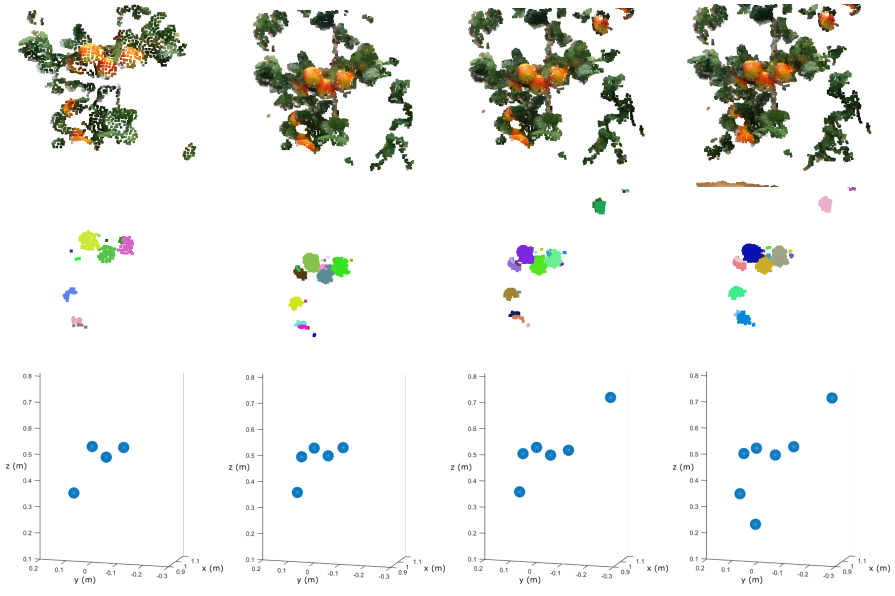


Figure 3.18 Segmentation results for an increasing number of viewpoints from left to right. The first row shows the registered point clouds for multiple views, the second row shows the labeled groups representing the segmented apples, and the last row displays the centroids of apples that were segmented with a sufficiently large number of parts associated..

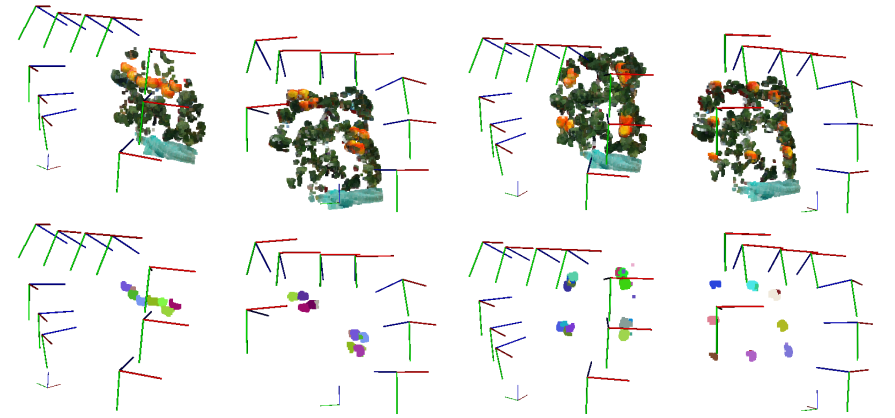


Figure 3.19 Segmentation results for different distributions of the apples over the vine..

3.5.2 Extending probabilistic object segmentation using to multiple different object's priors

In previous Chapter, it has been introduced how objects' surface can be modeled using the GPIS probabilistic model (Section 2.2.4) and how can be used this model together with a Gibbs Sampler to segment multiple instances of objects [12]. This section presents how to implement a more complex sampler to introduce several mean priors on the Gibbs sampler to segment objects of many kinds.

A a new function is introduced after line 14 in Algorithm 4 called $samplerPriorType()$. This function computes the likelihood of each prior for each object with it currently associated parts. Then it is sampled for the priors according to the likelihoods (computed using the information of the associated parts to the object). Figure 3.20 shows an example of the Multiobject-Multi-Prior sampler using ellipsoidal and infinite plane priors.

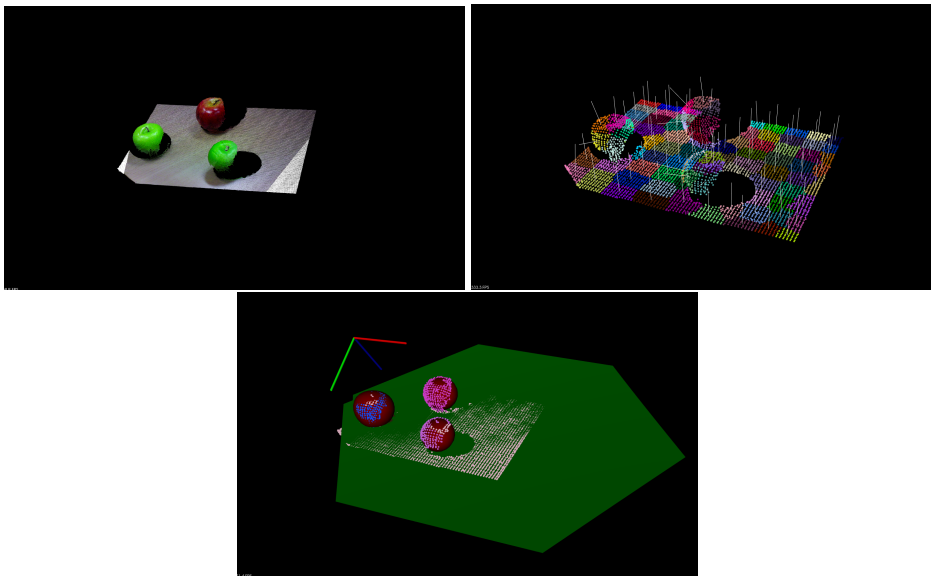


Figure 3.20 Segmentation of objects using multiple priors, in this case spherical and planar priors.

General poses of observed objects are immediately integrated into the prior functions, as outlined in the following.

A translation of the object by a vector μ is realised by shifting the mean function by an offset, $m^{\mu}(\mathbf{x}) = m(\mathbf{x}-\mu)$. For example, the mean function of a spherical object prior, translated by μ yields

$$m_S^{\mu}(\mathbf{x}) = \frac{r}{2}\left((\mathbf{x}-\mu)^T \mathbf{A}_S (\mathbf{x}-\mu) - 1\right). \tag{3.12}$$

Furthermore, anisotropic objects, such as ellipsoids, cylinders or planes, exhibit rotational degrees of freedom in $SO(3)$, in addition to translation. For a prior mean function $m(\mathbf{x})$, the result of a translation $\mu \in \mathbb{R}^3$ and rotation $\theta \in SO(3)$ is readily given by the transformed mean function $m^{\mu,\theta}(\mathbf{x}) = m(\mathbf{R}_\theta(\mathbf{x}-\mu))$, where $\mathbf{R}_\theta$ denotes the rotation matrix induced by θ. For example, the mean for a general ellipsoid reads

$$m_E^{\mu,\theta}(\mathbf{x}) = \frac{h}{2}\left((\mathbf{x}-\mu)^T \mathbf{A}_E^\theta (\mathbf{x}-\mu) - 1\right), \tag{3.13}$$

where $\mathbf{A}_E^\theta = \mathbf{R}_\theta^T \mathbf{A}_E \mathbf{R}_\theta$. In the case of a newly observed set of surface points and given an object prior function, μ and θ represent a set of unknown parameters that need to be inferred, for example via maximising the log-likelihood of the data.

3.6 Machine learning for object detection

Machine learning refers to a field of algorithms that allows the computer to generate behaviors or results by the generalization of information. The number of applications for machine learning is enormous. This section focuses on the use of machine learning algorithms for object detection and classification. These methods have been evolved along the years adapting to the resources of the technologies. First designed algorithms differ radically from what it is considered today "learning" from some adopters.

K-Nearest Neighbors (KNN)[110] is one of the simplest machine learning algorithms categorized as lazy learning type. KNN algorithm uses a set of data to classify new data points based on similarity measures. Others long-tradition algorithms for machine learning are decision trees and random forests. Decision trees [111] are non-parametric learning algorithms which are graph-based classifiers. These can be used to infer the category of new input data related to a input dataset by splitting the datasets into ramified groups in which each set of branches at the same level distinguish are different in some kind of attribute.

This section describes the methods explored and used during the research of the author. At first, the Bag of Words (BoW) model with a State Vector Machine (SVM) as a classifier and Latent Dirichlet Allocation (LDA) are presenteds. These use of these methods were generalized before the explosion of Neuronal Networks and Deep learning and have been proved to be robust and accurate in many applications. At last, a review of different Convolutional Neuronal Networks for object detection is presented.

3.6.1 Bag of Words model and State Vector Machine

This section introduces a classical approach for image classification so called Bag-of-words model. This model has been used as a first approach for instance object classification during this thesis. Particularly, it has been applied in in [11]. At first instance, it is briefly explained how the algorithm works. Then, qualitative results are shown. Current implementation in c++ has been wrapped up in the library described in Appendix A.1.

Images are $NxMx3$ matrices of unstructured data. Managing all this amount of information is not tractable. Thus, images are usually encoded in feature vectors. Features

are pieces of information that contain local information of the image, i.e., corners, lines, etc. This codification of information is a considerable reduction of dimensionality which simplifies the task of classification. The bag-of-words model (BoW)[112] simplifies the images into an histogram of codified features. BoW methodology is commonly used to classify text documents where the occurrence of each word is used as a feature for training a classifier. In similarity with the text model, image features are interpreted as words. Each object is represented by a histogram of visual descriptors, computed by detecting features in the image. Figure 3.21 shows a simplified representation of the bag of words model.

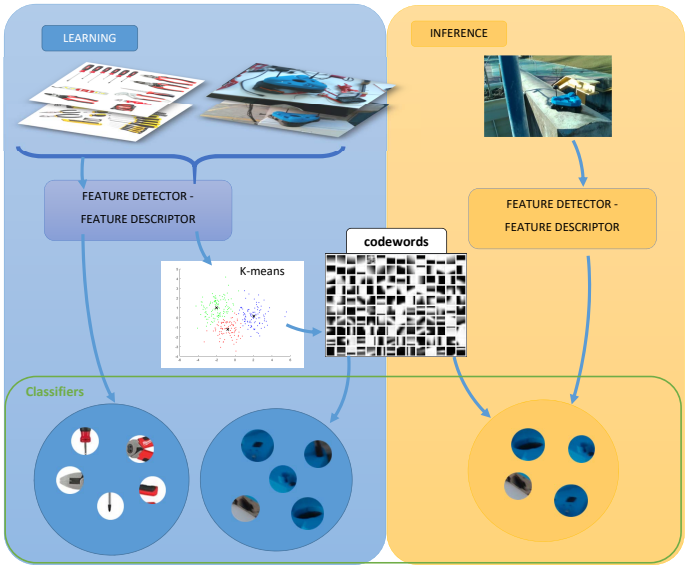


Figure 3.21 Bag of words representative model for image.

BoW model requires a vocabulary (so-called codewords), which is a set of representative descriptors that are used as a reference to quantify features in the images. The vocabulary is generated during an offline training process of the classifier. The algorithm detects all features in the training set. Then a clustering algorithm (typically k-means) is used to extracts the representative set of *words* by N clusters.

During the training process, a classifier is optimized using positive and negative object samples. The resulting object detection and recognition system return the labels of the detected objects, which the drone uses if a specific object must be picked up or located. By training the object to recognize object categories, a novel object can also be classified. In this work, SVM has been used as classifier.

During the 1990s, before the explosion of modern neuronal networks, SVM [113, 114] was one of the best solutions as a classifier due to its high efficiency. In a nutshell, given a set of multidimensional points which belong to two categories, a linear SVM seeks for the hyperplane that divides the set of points so that each side contains the most significant

possible fraction of points belonging to the same class. This hyperplane divides the space so for each new point can be categorized.

However, linear SVM classifiers assume that the data is linearly separable. Thus they are not able to classify real-world data which, usually, it is not. Left picture of Figure 3.22 shows an example of circular distributions of data which are not linearly separable. In order to solve, Vapnik [114] proposed what it is called "kernel trick". It consists of applying a feature space mapping to the points, so they became linear separable by the SVM. There are many kernels such as the polynomial kernel or the Radian basis function (RBF). Right picture in Figure 3.22 shows how the problem becomes linear separable in 3D by applying the polynomial kernel.

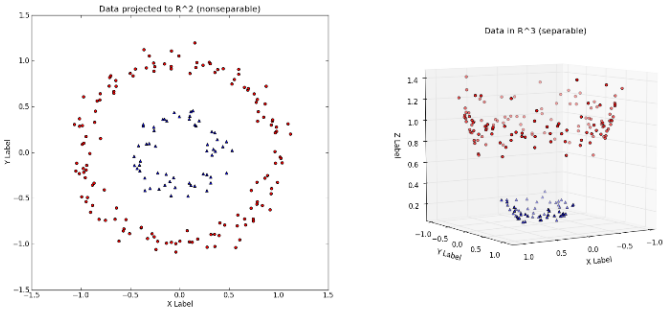


Figure 3.22 Example of non-linear separable data and application of polynomial kernel trick to make it linear separable in a higher dimensional space.

In this research [11], the SVM classifier was trained using 16 individual objects based on the BoW model. A second classifier has been trained with the objects were grouped by categories (Figure 3.23): cans; juice boxes; circuits; cars; boxes; Some of the objects were very similar and hard to distinguish from certain angles, e.g., the original coke and generic copy, juice boxes of the same brand but different flavors, since they possess intentionally similar appearance. Shi-Tomasi corner detector has been used in combination with the SIFT descriptor [21] to model words.

(a) Category *cans*. (b) Category *juice box*. (c) Category *circuits*.

Figure 3.23 Some sample categories of objects used for the BoW-SVM algorithm in [11].

The algorithm has been tested in four test scenarios (*Laboratory*, *Street1*, *Street2*, *Testbed*). In the *Laboratory* scenario had an uniform background. In this scenario the high-level classification of objects in categories was excellent, while the specific object class seems to produce less accurate results. It should be noted that no color information was used for training and recognition, although it could improve recognition results of individual objects.

In the other scenarios, the recognition rates were lower, and we attribute this to our feature extraction implementation. A square bounding box around the objects is used to extract features instead of only the convex hull. During the training the background was plain, but those examples with a textured floor, a lot of the background is included in the bounding box, and it is vibrant with visual features. Still, it can be seen that the categories were detected better than the individual objects.

Table 3.2 summarizes the results of our object extraction method in real flights. In our scenarios the number of total points in the local map stopped increasing significantly after this point, meaning the area had been mostly inspected and the results have converged. However, this is an empirically derived value that depends on the diversity of the observed scene and the flight path the UAV takes to inspect it.

	Categories			Objects		
Dataset	Precision	Recall	F-Score	Precision	Recall	F-Score
Laboratory	1	1	1	1	0.5	0.667
Street 1	0.429	0.6	0.6	0.333	0.4	0.36
Street 2	0.783	0.4	0.53	0.75	0.33	0.462
Testbed	0.429	0.5	0.462	0.2	0.167	0.182

Table 3.2 Results of the object extraction method.

This algorithm has been integrated into a complete system that computes a local map of the environment and subtracts patches on images that are evaluated by the BoW-SVM algorithm. A show-case video can be seen in this video[7] showing both the mapping and the learning capabilities. The Chapter 4 will introduce in more detail the mapping algorithm.

Note on multiple object instances. Typically, BoW-SVM algorithm is applied to an image or a patch of images. In work presented in [11], authors used a bottom-up approach where candidate objects are extracted from a reconstructed local map of the scene and then recognized using 2D information from the images. It means that the object detection algorithm is only applied to single patches of objects previously computed by the projection of the 3D object segmented in the scene.

Nevertheless, this algorithm can be applied without previous 3D knowledge of the scene. In order to do that, it is typically applied a sliding window methodology in which many patches of the image are extracted sequentially with different sizes, and then these are processed by the classifier as shown in Figure 3.24. A threshold filters the results, and then a Non-max suppression algorithm is used to erase redundant results.

[7] *https://www.youtube.com/watch?v=cRBgsHr0pHA*

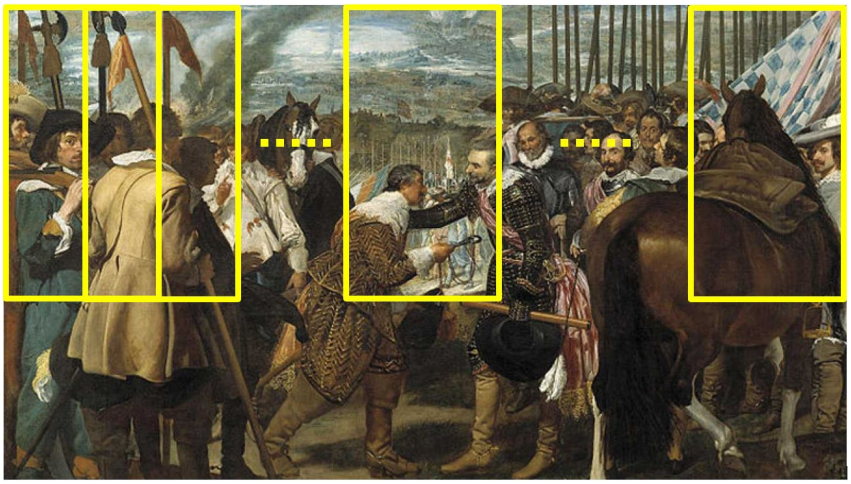


Figure 3.24 Example of sliding window in the Picture *La rendición de Breda o Las lanzas* by Velazquez.

3.6.2 Latent Dirichlet Allocation, a text-oriented non-supervised image classification

There is a prolonged confrontation between two branches of object classification researchers [115]: the ones that support generative models and the ones supporting discriminative models. Support Vector Machines, described in the previous section is a discriminative model. This kind of algorithms directly tries to learn the posterior distribution of the probability of some kind of feature vector to belong to a specific class ($p(z|X)$). However, there exists another alternative. Generative classifiers try to infer the joint distribution $p(z,X)$ to be able to make any prediction using Bayes rules to compute $p(z|X)$ to pick the most likely category z.

Previous section describes an algorithm that learns to classify images starting from a set of pre-categorized images. These methods are called supervised machine learning and imply that each input vector (or image) are tagged manually. Sometimes this is not possible or even intractable. For these reasons, there are other classes of algorithms that can classify information without human intervention nor labeling, which are called unsupervised machine learning.

This section introduces Latent Dirichlet Allocation (LDA) [116, 117, 118], an unsupervised approach for topic modeling. This algorithm has been implemented an integrated in the OpenSource library introduced in Appendix A.1.

LDA intent to discover latent semantic topics in collections of documents. To make it easier to understand first, this model is based on the concept that text documents that discuss similar concepts usually use similar words. Thus, it is possible to establish categories by identifying these group of words. Typically, by accounting for the frequency of usage of the words within the corpus of different documents, it is possible to discover latent topics. Documents are modeled as a random mixture over latent topics, being each topic characterized by a particular distribution of words.

Consider a document a finite mixture over underlying topics. The documents are composed of a sequence of N words denoted by $W = (w_1, w_2, ..., w_i, ..., w_N)$. Each word is a discrete $V-$dimensional unit of data. A corpus is a collection of M documents denoted by $\mathbb{D} = (W_1, W_2, ..., W_M)$. Eventually, a topic $z = (1, ..., K)$ is a probability distribution over a vocabulary of V words. There are only two hyper-parameters which control the expected behavior of the probabilistic distributions over topics α and words β as shown in Figure 3.25.

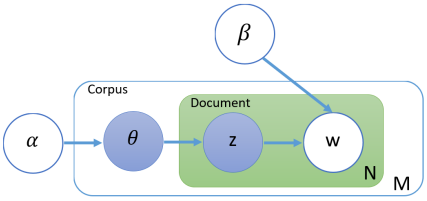


Figure 3.25 Generative model of Latent Diritchelet Allocation algorithm for topic modeling.

The method aims to compute the posterior distribution of the hidden variables (z and θ) given a document.

$$p(\theta, z|w, \alpha, \beta) = \frac{p(\theta, z, w|\alpha, \beta)}{p(w|\alpha, \beta)} \tag{3.14}$$

Unfortunately, this inference is, in general, intractable due to the couple between θ and β [117].

In order to solve this problem, there are many alternatives. One option is to use a variational inference algorithm. The basic idea of convexity-based variational inference is to obtain a lower bound on the log-likelihood, or to consider a family of lower bounds indexed by a set of variational parameters. This solve the coupling between θ and β obtaining a family of distributions on the latent variables θ and z characterized by the distribution (Figure 3.26)

$$q(\theta, z|\gamma, \phi) = q(\theta|\gamma) \prod_{N}^{n=1} q(z_n|\phi_n)$$

At this point, it is possible to define an optimization algorithm that determines the optimal values of the variational parameters γ and ϕ given a set of finite data. The complete methodology is described in in [117, 119]

A second alternative is to use Gibbs sampling [120, 121]. The problem is still to infer equation 3.14. The previous method proposes a simplification in the problem to obtain a tractable solution. This alternative is based on a Monte Carlo sampling methodology [122, 123]. In this case, the variables of interest are the topics of portions of the documents θ_d, the distributions of words per topic ϕ^z and the specific topic of each word z_i.

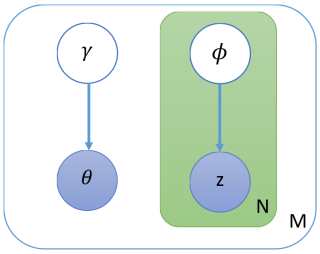


Figure 3.26 Generative model of Latent Diritchelet Allocation algorithm for topic modeling.

Given an initial random assignment of topics over the words z_i and the observed variables α, β and the words W, the algorithms try to improve the categories by performing continuous Monte-Carlo steps.

$$p(z_i|z_{i-1},\alpha,\beta,W)$$

Another sequential optimization alternative is using an Expectation-Maximization algorithm (EM) [124, 125]. This algorithm iteratively computes the probability distribution of the words in a document (or image) d_j

$$p(w_i|d_j) = \sum_{K}^{k=1} p(w_i|z_k)p(z_k|s_j)$$

by performing two steps. The expectation step of the topic distribution is

$$p(z_k|w_i,d_j) = \frac{p(w_i|z_k)p(p_z|d_j)}{\sum_{l=i}^{K} p(w_i|z_l)p(z_l|d_j)}$$

next is, a step to maximize the likelihood of the data

$$L = \prod_{i=i}^{M}\prod_{j=1}^{N} p(w_i|d_j)$$

A typical representation of LDA is using a geometrical representation of the latent space [126]. In the case of using three topics, it is a triangular representation in which each vertex correspond to one of the topics and the distance to them indicates the proximity or similarity to that topic.

Figure 3.27 shows the case of a classification of a set of images from the Caltech 101 Objects dataset [127] for four different categories. The picture shows the effect of varying both alpha and beta in the results of the distribution of documents after the training. Each point is represented with a different color according to the real label of the image.

Figure 3.28 shows the optimal result for the classification. It can be seen that each color tends to a specific corner or the tetrahedron.

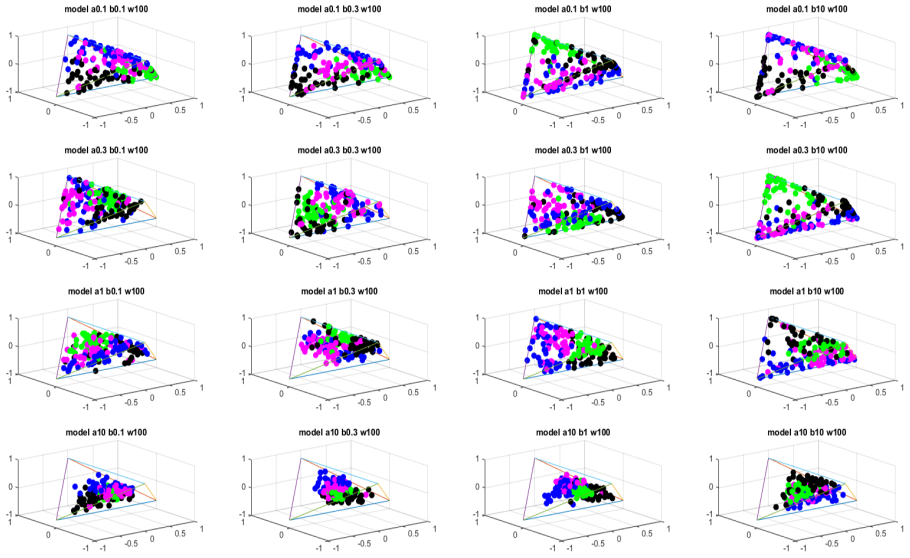


Figure 3.27 Geometrical representation of the documents of the LDA according to their classification. Figure shows the effect of varying the hyper-parameters of the model.

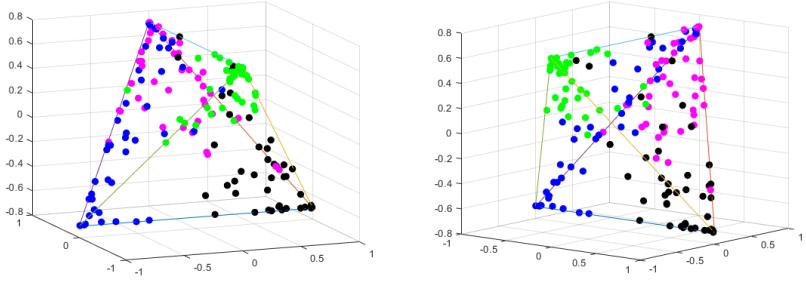


Figure 3.28 Geometrical representation of the documents of the LDA according to their classification. Figure correspond to the results with the optimal parameters for four categories of the Caltech101 dataset.

The resulting model can be used to infer labels on new images. Figure 3.29 shows examples of inferences on new images by the trained model. Each row corresponds to a different category (camera, cougar, motorbike, and faces). It can be seen that motorbike is the category with less confusion while cougars, faces, and cameras have one mislead result.

Figure 3.29 Inference on new images of Caltech101 dataset using pLSA trained model.

The same algorithm has been used to train a dataset of tools (described in the following section). The results have been tested using a sliding window algorithm to detect and locate different tools as shown in the following video[8].

[8] *https://www.youtube.com/watch?v=SuHmORfo8-U*

3.6.3 Neuronal Networks and Deep Learning

Even though classical methods are proved to have fair results in many applications, there are many issues related to their design, flexibility, and robustness. One of the primary drawbacks of classical methods is that they rely, up to a certain point, on a human design in terms of feature selection. For example, the previously described algorithms, BoW and LDA, can discriminate between different sets of objects by labeling histograms of features which are computed for every single image. These are even able to infer the classification of unseen data. However, in order to perform this classification, it is necessary to provide them with the histograms of features. It means that it is needed to predefine which features are used to describe the objects, the size of the vocabulary to cluster the features and the size of the histogram which is used as a signature for each object.

Artificial Neuronal Networks (ANNs) suffered from the same problem in their beginnings. ANNs are algorithms which topology inherites somehow from biological neurological systems [128]. ANNs are composed by a minimal entity called neuron or perceptron [129]. These small pieces consist of an input, a weight, an activation, and an output; simulating a biological neuron. Having several of this networks connected creates what is called a fully connected network. The first layer is the input layer and the last layer the output layer.

However, these networks can model just linear classifications. Adding multiple intermediate layers (or hidden layers) starts adding non-linear behaviors to the network allowing them to model more complicated information.

The revolution of neuronal networks come with the invention of Convolutional Neuronal Networks (CNN). In a nutshell, these networks have a preprocessing set of layers called convolutional layers which perform a set of convolutions which are weighted too and compute by their selves thousands of possible features which fed the fully connected layers which are responsible for the later classification. The key power of these networks is that, for the first time, the algorithm learns itself the proper features to make the right classification, contrary to classical approaches in which the feature selection needed a carefully human-design. This fact, and the impressive advances in hardware technology (particularly advances in GPU devices) made CNN powerful tools for image processing.

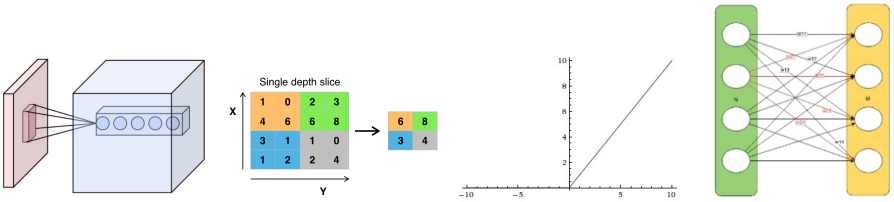


Figure 3.30 Different layers involved in modern artificial neuronal networks. From left to right: Convolutional layer, max pooling layer, ReLU, and fully connected layer.

Since then, the number of networks and variations have exploited. In the field of

object detection, most used networks are Faster R-CNN [130], YOLO [131, 132], Mask-RCNN [133], and SSD [134], which has been tested during this research.

In this thesis, CNNs have been used as a tool for detecting objects robustly to perform manipulation tasks. During the development of this thesis the author explored two different frameworks for deep learning: caffe [135] or TensorFlow [136] which are ones of the most used tools.

The remainder of this section show quantitative results of the use of various network topologies in two different datasets for different applications. The first application consists of the elaboration of a dataset of hand tools to be grasped by the UAV. The results given by the network is used later for creating a labeled 3D map with the location and category of the tools, which will be introduced in Section 4.5. Evaluates different networks to choose the appropriate one for a real time grasping application described in [13].

Evaluation of different nets in a custom Dataset of hand-tools

This brief section compares the results of different state-of-art object detectors in a custom dataset for detecting hand-tools. The purpose of this dataset is to allow the aerial platform to grasp different tools to use them or to transport them to a human operator for human-robot cooperation. Figure 3.31 shows in a shot some examples of the training set. This set has been manually labeled and, up to now, contains four labels which are: screwdriver, wrench, plier and hammer.

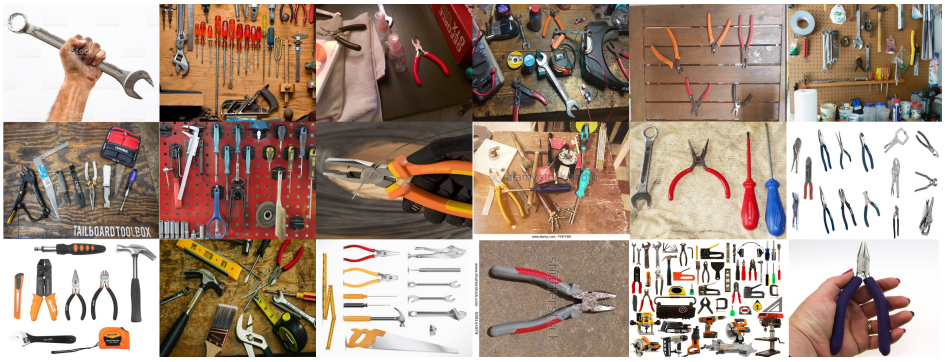


Figure 3.31 Sample images from hand tools dataset.

CNN is widely used nowadays. These algorithms provide a quick and easy solution to many intractable problems such as is the case of object detection. If the target object is known, just a labeled database of images is needed to train the network. However, these algorithms are extensively heavy regarding mathematical operations, but they are highly parallelizable. Thus, Graphics Unit Processors are usually used to boost their performance. Ones of the most used frameworks for this purpose are TensorFlow [136] or Caffe [135]. They provide for an extensive "model zoo" with implementations of newest networks. This speed up the testing and development process. However, it is vital to be taken into consideration that the final algorithm will be carried out by the onboard computer of the drone, which may have some limitations such as CUDA compatibility.

The results thrown in this section have been calculated using TensorFlow model zoo. Specifically, the dataset has been tested with four networks, Faster R-CNN [130], SSD [134], and YOLO [131, 132] (this network has a different implementation and is not embed in Tensorflow, instead it can be found in[9]).

R-CNN is one of the first region based convolutional neuronal networks. As mentioned before, one of the main problems of object detection algorithm is how to accurately propose regions where the objects can be located to be able to identify them. Due to the large pixel resolution, the number of possible combination is enormous. Girshick et al. proposed an algorithm that for each instance it selectively generates 2000 regions. It was called region-proposal step. This fixed number of regions is then warped and fed the CNN which computes the features, and then a classifier put the labels on the object. This algorithm showed promising results, but its computational time makes it unusable for almost any application. Later, Girshick proposed a new version of the algorithm called Fast-RCNN. In this work, the author improved the region proposal phase by using a net to generate a convolutional feature map. Then, this map is used to generate the region proposals.

Later in 2016 two new networks appeared which revolutionized the concept of object detection; YOLO (You Only Look once) and SSD (or Single Shot Detector). Both networks are based on the same concept, processing in the same net both the regions and classifying them, contrary to FRCNN. This fact makes both networks remarkably faster than previous algorithms. Additionally, as the whole image is processed, the network is fed with all the contextual information, a fact that does not happen if the regions are extracted before, and the net is fed only with the regions. Figure 3.32 shows the structure of both networks. YOLO takes the whole input image, processes it and divides it in *SxS* grids, for each grid it computes *B* bounding boxes. Also, in the same pipeline, a classification score is generated. It makes an all-in-once region proposal and classification. Nowadays, a faster version of YOLO is the fastest object detection network. SSD appeared to be a trade-off between accuracy and speed. It has a single network which computes a rich feature map and then the information from different layers fed a convolutional layer that generates the regions and the classification scores.

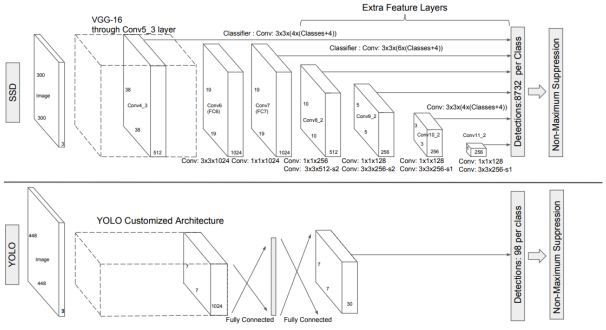


Figure 3.32 Structure of SSD versus YOLO networks.

[9] *https://pjreddie.com/darknet/*

Each the networks introduced had a particular improvement, it is also essential to take into account that every quartile an improvement appears in this field, either by a new software enhancement or by new hardware version. For this reason, this section tries to compare and analyze the strengths and weakness of each of this net designs objectively.

Method	mAP (0.5IOU)	mAP (0.75IOU)	Recall
F-RCNN inception_v2	0.5192	0.2717	0.4576
F-RCNN restnet50	0.3891	0.0978	0.3954
F-RCNN restnet101	0.4242	0.1063	0.4132
YOLO v3 tiny	0.69 (avg.)		0.65
YOLO v3 ssp	0.77 (avg.)		0.70
SSD inception_v2	0.5847	0.3557	0.5006
SSD mobilenet_v1	0.6923	0.4670	0.5945

Table 3.3 Detection scores for the hand tools dataset for the tested networks..

Despite being small, this dataset is quite challenging as the shapes and colors of the different tools are highly varied. Also, there are many similitudes in the shapes. There are many images in the dataset that are very challenging because hand-tools appear cluttered or in cases, as shown in Figure 3.33.

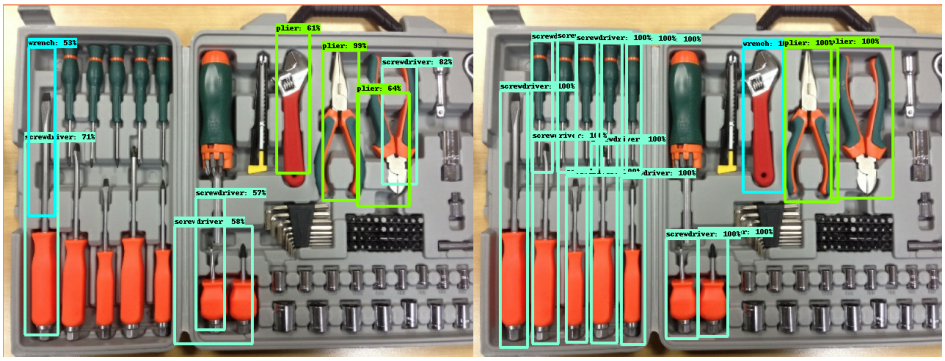


Figure 3.33 Detection results of F-RCNN in the datasets tool for difficult image. Left picture shows the detection results and right picture shows the ground truth.

Crawler detection using CNN for aerial manipulation

In this section, a set of CNNs are tested for the detection of a crawler robot. The purpose is to find the a network that is robust and able to run in real time to grasp the crawler using the aerial manipulator. Once the crawler robot is detected, the algorithm described in Section 2.4 is used to perform the grasping.

This section describes the algorithm for the detection and tracking of the target object. The algorithm is summarized in Figure 3.34, yet introduced in the previous section of this chapter. The algorithm is based on two stages, at first instance, an object detection convolutional neuronal network (CNN) is applied to produce object candidates. Then

an alignment algorithm is used to compute the exact location of the target object. This section focuses on the integration of the neuronal network in the pipeline, as Section 3.3 introduced the alignment algorithm previously.

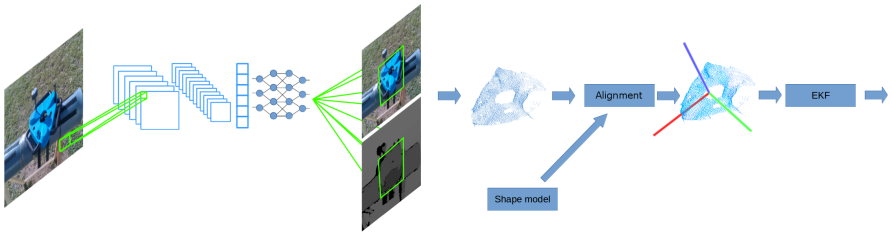


Figure 3.34 Scheme of the visual algorithm for the object detection and pose estimation of the target object.

The results of the object detection and pose estimation are filtered and used by a grasp planning and visual servoing module as described in Chapter 2.

In order to train and evaluate the task of the detection of the crawler robot, a custom intensive dataset has been created. The dataset contains images of the crawler robot in different environments with different light conditions, backgrounds, and occlusions. Figure 3.35 shows some pictures of the dataset in different environments.

Figure 3.35 Sample images from crawler dataset.

Three popular algorithms have been tested in three different devices to choose the best option that fits the needed speed for the manipulation task and the payload limitations of the aerial manipulator. In particular, the CNN chosen were: FRCNN, SSD300, and YOLO (tiny YOLO v2); in the devices: laptop with a GTX1070, a Jetson TX1 and an Intel NUC with an iris GPU. Two first devices have CUDA capabilities as they use a Nvidia GPU, the third one, however, can not use common frameworks that use CUDA. For this reason, an

OpenCL implementation of YOLO[10] has been tested. Table 3.4 summarizes the averaged computational times of the different detection algorithms in the different devices.

Table 3.4 Computational time in seconds of the different algorithms in the tested devices..

	F-RCNN	SSD	YOLO (OpenCL)
Laptop	0.067	0.027	0.0103
Jetson TX1	0.47	0.113	0.051
Intel NUC	-	-	0.053

It is seen that the performance on the laptop overtakes the results of the other devices. However, due to the strict payload limitations, while operating with UAVs, just the other two devices can be considered. At first, YOLO runs a bit faster in Jetson Tx1 device. However, CPU speed of Intel NUC highly overtakes Jetson's CPU capabilities. As the whole system requires many processes to run many tasks, the last platform was chosen. Figure 3.36 shows example of results in different environments of the algorithm online.

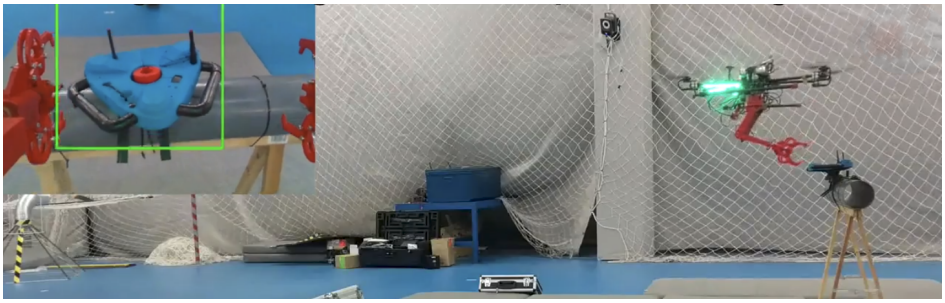


Figure 3.36 Scheme of the visual algorithm for the object detection and pose estimation of the target object..

The overall system proves to be able to perform the mission autonomously. The system detects and track the object using the algorithms described in Section 2.2.2 and Section 3.6.3. Then, having the pose of the object, it computes a set of good feasible grasps as described in Section 2.3.2. These grasps, together with the continuously updated pose of the object is used to perform a PBVS algorithm, described in Section 2.4, which moves the arms towards the grasp and perform the grasping. The results can be seen in the following video [11]

[10] *https://github.com/ganyc717/Darknet-On-OpenCL*
[11] *https://www.youtube.com/watch?v=nXYlzqwM8kA*

4 Mapping and localization for aerial manipulation

4.1 Introduction and Related Work

This Chapter addresses the topic of Localization and Mapping algorithms for aerial robots with manipulation capabilities. Two methodologies has been applied to solve this problem, each of them described in a separated section. Both sections presents the implementation carried out by the author for the localization of the UAV and the creation of a local map of the environment, and have been integrated in the OpenSource library described in Appendix A.1. The presented work focuses on algorithms to work in a local area. It means that the purpose is not to elaborate large-scale maps but smaller portions to allow the robot to execute manipulation tasks. Having this map, the robot can localize itself, the objects, and take into account possible collisions between the manipulators and the environment.

Self-localization in the environment is one key skill required for the full automation of the robot. To perform dexterous manipulation tasks, knowing the specific set up where the task is going to be performed is important to locate the target objects or to prevent the manipulators to collide. Additionally, this information may be useful for planning movements from one place to another, avoiding obstacles, among other tasks.

The task of localization usually involves two concepts. The first is the representation of the environment which is typically called map. The second one is the localization of the robot in that map.

Maps can be of many kinds, depend on the data used for its creation. Maps can be a CAD designed model of a factory, building or room, which can be used as a reference to localize the robot [137, 138]. Other kinds of maps are based on frequency beacons placed in the environment which are used to triangulate the position of the robot [139]. Alternatively, other models use three-dimensional point clouds which are computed while the robot moves in the environment [140, 141].

Maps can also hold information about where the data was obtained. In graph-based map models [142, 143] the observations of the environment and the positions where the robot

took the data compose a graph with nodes and edges. In this model, each edge corresponds to a particular observation of the environment in a specific pose of the object. Then each side of the edge corresponds to a robot pose on one side and a particular landmark on the other side (Figure 4.1).

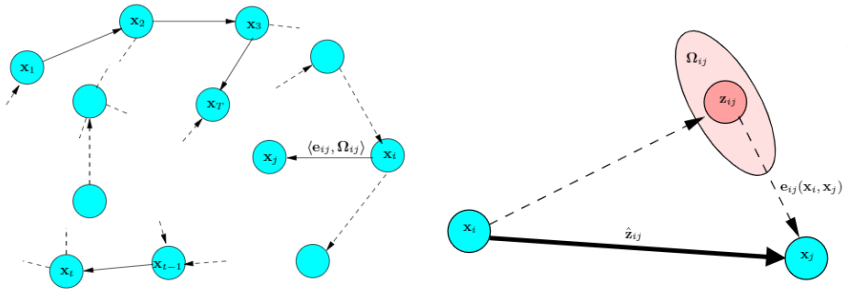


Figure 4.1 Graph-based SLAM model. Resource from [143].

Traditionally, one of the common localization methods for humans is based on the global reference frame of the earth, labeled with Latitude and Longitude. This position is commonly provided by a Global Positioning System (GPS) [144, 145]. In recent years, the European Commission put a special effort into developing a new positioning system, Galileo [145]. The new global positioning system is currently integrated into many platforms with promising results. These global localization systems are based on the triangulation of the position of the sensor using the signals from satellites. In order to receive this signal, it is necessary to have "line of sight" of a number of satellites which are distributed in the orbit of the Earth. The set of connected satellites is called constellation.

However, due to the "line of sight" constraint, this localization system is not suitable for all the situations. Indoor applications, or applications in environments with poor GPS coverage, require for other localization systems. The use of sensors to compute the localization of the robot and the map concurrently is frequently called SLAM, or Simultaneous Localization And Mapping. SLAM algorithms, typically use visual or laser sensors to obtain information about the environment and associate it sequentially to compute the incremental transformation between batches of data. Many methodologies and frameworks [146, 147, 148, 149, 150] has been developed which produce good enough solutions to this problem nowadays.

Sequential correlation of the information from the sensors is usually called odometry, or visual odometry if the system uses cameras. Typically, to make visual odometry algorithms more robust, the information given by the perception sensors is fused with other sensors available in the robots. These can be for example the GPS when it is available, encoders in the axis of the actuators or Inertial Measurements Units (IMUs). The use of Bayesian filters [151] such as EKF [152] or Particle filter [153] for robot localization integrating these sensors is widely extended.

However, even these method reduces the incremental error of conventional odometry algorithms. They still suffer the problem of drifting. In order to solve it, many authors

studied the use of optimization algorithms to minimize the errors in the maps and reduce the global drifts produced over time. These processes usually enclose two methods or phases. One of them is the optimization itself. The second one is the so-called loop-closure, which consists of the location of closed loops during the execution of the algorithm. This closing loop methods allow the optimization algorithm to minimize the error enclosed in the loop, thus reducing the long-term biases.

The sequential computation of the position of the robot, i.e., odometry, algorithms usually produces, not only the localization of the robot but also a set of points or entities of interest that are part of the environment. The accumulation of all these interest points is what is called map and, as aforementioned, they can be of any kind. This map also suffers from the shifts in the localization, and it is also improved by the optimization algorithms to produce a more accurate and robust map. 4.2 shows an example of map representation and its improved version after the optimizations.

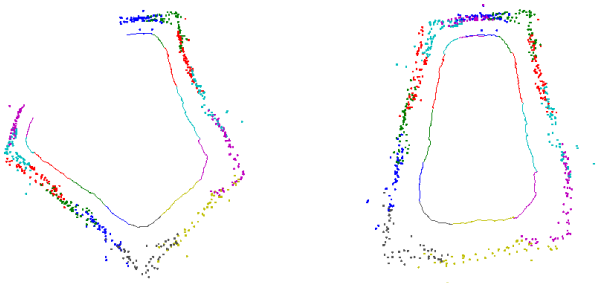


Figure 4.2 Effect of loop closure and global optimization in SLAM. Resource from [154].

Loop closing is very related to a concept called visual place recognition [155] in this context. It is a challenging problem due to a large number of ways in which the appearance of a place can arise. Many solutions [156, 157] propose a probabilistic appearance-based approach using a Bag of Words model. However, these methods usually rely on the invariance of the descriptors in the scene. Light changes or smog might profoundly affect the descriptors.

Previous loop closure algorithms are based on hand-crafted features such as ORB. Recent authors explore the use of deep learning techniques to auto-generate in a more efficient way the representation of the images for detecting loops [158, 159, 160].

Once loops closures are detected, the global optimization algorithm turns into action. The optimization process depend on the information captured by the sensors and the representation of the map in the SLAM algorithms. One of the most popular ways of representing the maps in the environment is the graph-based map [143]. This representation has been popularized, and many frameworks exist which provide a quick implementation to be integrated, particularly g2o [161] is one of the most used frameworks.

Bearing all the above in mind, it is possible to make a theoretical classification of SLAM methodologies: online SLAM and Full SLAM. Online SLAM refers to those algorithms that just pay attention to the computation of the most recent pose of the camera and map.

Full SLAM methods intend to estimate the full trajectory of the robot and the complete map of the environment. The latter requires a more complex structure as all the information is continuously being updated while the methods run.

Regarding the practical application of the SLAM methodologies, the selection of the sensors highly influences the algorithms. There are several kinds of sensors that can be used for this applications, but these can be classified in three classes as shown in Figure 4.3

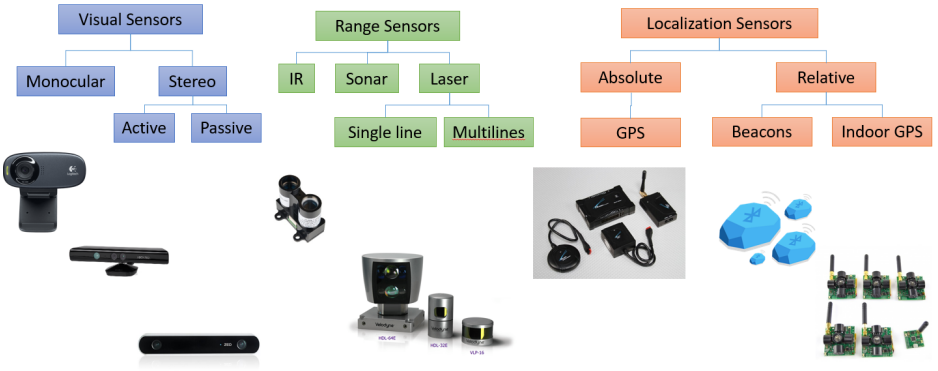


Figure 4.3 Classification of sensors for SLAM algorithms.

This chapter focuses on the use of sensors of the first kind, i.e., visual sensors, for SLAM applications. Particularly, the work presented focuses on the use of stereo sensors. This kind of sensors have a good trade-off between the resolution and accuracy versus price. In the remainder of this section, this kind of sensors may also be called RGB-D sensors, as they provide both color images and an estimation of the depth of the points.

Typical RGB-D cameras such us Microsoft Kinect works with infrared (IR) structural light. The camera projects a pattern with a laser which is captured by an IR sensor. The deformation of this pattern is then used to build a depth map efficiently. However, the main drawback of these cameras is that they barely work outdoors in daylight due to the radiance of the sun.

Passive stereo cameras use a pair of calibrated cameras [162][163] to compute the depth of the scene by comparing the disparity between the images. Graphics Processing Units (GPU) have played an essential role in this scope. Recent devices exploit GPU capabilities to build quickly dense disparity maps from calibrated stereo devices [164][165]. Cameras, such as Zed by Stereo Lab, revolutionized SLAM world as these devices work where structural light devices cannot.

Newer devices combine both technologies by adding a pair of IR sensors in the RGB-D cameras. These devices use the structural light where the pattern is available and exploit the stereo capabilities in the presence of sun's radiation. An example of this technology is the new devices created by Intel under the name of Intel RealSense [166].

The remainder of this chapter proceeds as follow. Section 4.2 focuses on a first online SLAM approach based on sparse featured clouds computed using low-cost unsynchro-

nized stereo cameras. This implementation integrates the visual odometry with inertial information from the IMU of the drone to elaborate a sparse map while computing the position of the robot. Section 4.3 introduces a SLAM Framework based in RGB-D sensors. This Framework proposes an architecture for the full SLAM problem which includes optimization algorithms and loop closure detection. Section 4.4 shows a set of validation results from different datasets of the proposed SLAM Framework. Finally, Section 4.5 shows how the SLAM framework can be used together with machine learning algorithms to elaborate a semantic map with objects instances for later manipulations tasks.

4.2 Online SLAM method using stereo cameras

This section presents a simultaneous localization and mapping algorithm for UAVs using low-cost unsynchronized stereo cameras. This work has been presented in [11] and used for the detection of objects described in Section 3.2.

This first approach is based on the creation of sparse feature clouds and its sequential alignment to estimate the position of the robot. It is assumed that the source of the images is not fully reliable. For this reason, it proceeds at the beginning with a set of filters to reject blurry images and possible outliers. For the robustness of the algorithm, it has been implemented an EKF which fuses the results of the visual odometry with the inertial information of UAV's IMU. Figure 4.4 summarizes the pipeline of the algorithm. Concerning the map, it is a keyframe-free representation. It means that poses are not stored and no optimization algorithm is used to refine the map. Instead, the map is stored as a raw set of feature points filtered with a voxelized operation over time.

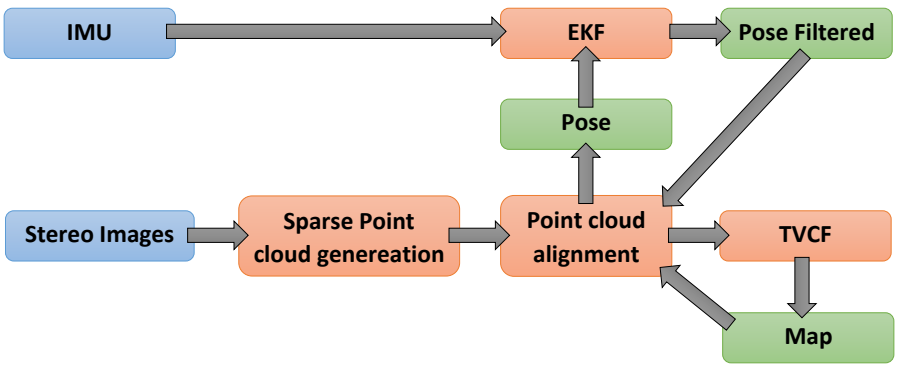


Figure 4.4 Epipolar geometry..

The generation of the point clouds from the pair of unsynchronized cameras is divided into four steps:

1. Evaluation of input images
2. Visual feature detection in the left image.

3. Template matching over the epipolar line in the right image.
4. Triangulation.

At first instance, the cameras are prompted to return new images. The time difference between them can be up to $1/FPS$. Thus, if the movement of the drone above the objects is slow, the time drift can be ignored. However, it is essential to check if the quality of the image is good enough for the creation of the point cloud. This quality is measured in terms of blurriness. Particularly, the score proposed in [167] is used to evaluate the quality of the pair of images. If some of both of the images is blurry or there is a large time drift between them, the quality of the cloud decrease drastically. An example of good and bad point clouds is shown in Figure 4.5.

Figure 4.5 On the Top a good input image is shown and the corresponding point cloud, generated from the stereo pair. On the bottom a blurry image results in a very poor cloud. A similarly bad cloud is also generated when images are mistimed too much..

Once the pair of images is accepted, the algorithm proceeds computing visual features in the left picture. These features are assumed to be there in the right one. The pair of cameras is adequately calibrated, thus the extrinsic constraint and the epipolar geometry can be

used to look for keypoint matches. A template window is slid across the epipolar line and compared to the template of each corresponding keypoint. If the matching score is lower than a threshold, the keypoint pair is then triangulated (Figure 4.6). All the triangulated keypoints build the point cloud.

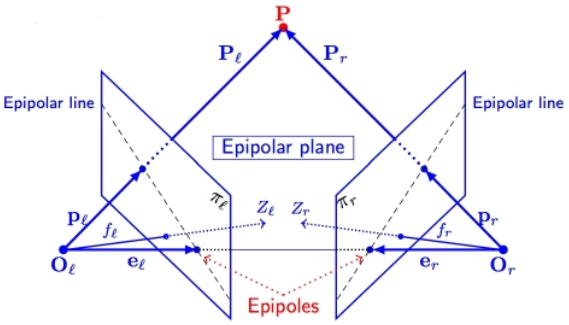


Figure 4.6 Epipolar geometry..

Any feature detector can be used (See Chapter 2.2.2). In the work presented in this section, Shi-Tomasi corner detector [27] has been chosen because these features are cheap to compute. The cost function used for the template matching algorithm is the squared sum of differences.

The resulting point clouds might contain noisy points due to bad matches, triangulation and calibration errors, mistimed stereo images (Unsynchronized stereo produces a delay between the captured frames), occasional rolling shutter effect because of vibrations, and even some partially blurry images that get through to this point. For this reason, it is necessary to process these clouds before adding them to the map. The author developed a method that processes sequential point clouds in both spatial and time dimensions before adding the result to the map.

The point clouds are filtered in two steps: (1) Spatial filtering: Isolated particles or small clusters of particles are considered noise (using [168]) and the remaining points are transformed into a grid of cubic volumes of equal size, also called voxels, where a voxel is occupied if at least one point from the point cloud belongs to it. [169]; (2) Time filtering: filtering method over time is proposed using sequential voxel point clouds stored into memory, also called history. The occupancy of each voxel is checked in each cloud in history so that only voxels that have a higher probability of being occupied by a real point will be kept. We call this method Temporal Convolution Voxel Filtering (or **TCVF**).

Given a set of N consecutive point clouds PC_i. The goal is to obtain a realistic representation of the environment by filtering out incorrect points. The algorithm 5 describes the process.

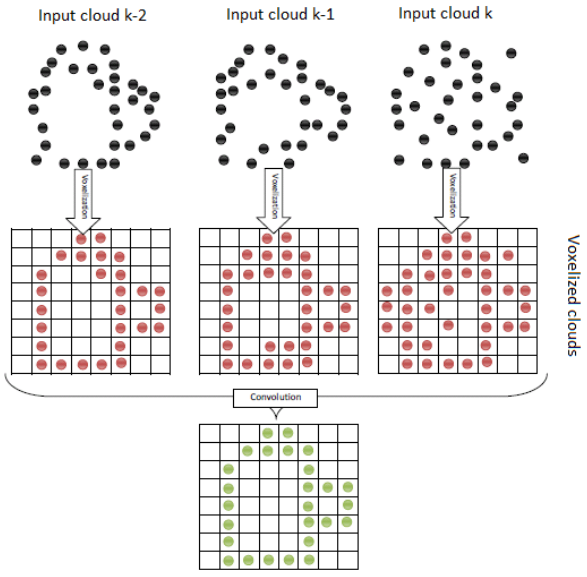
TCVF adds a new cloud to the history in each iteration and evaluates the clouds kept in the history at that moment. The result of this operation is then added to the map. By discretizing the space, the number of points for computation is reduced, which reduces the computational time. An occupancy requirement of 100% is used throughout the entire history, making this calculation a simple binary operation of occupancy check, which is

Algorithm 5 Probabilistic Map Generation

```
MAP ← empty
for i ∈ [0,N] do
    PC_I ← filter(PC_I)
    PC_I ← align(PC_I)
    PC_I ← voxel(PC_I)
    addToHistory(PC_I)
end for
for point in PC_i do
    if point ∃ in PC_i with i ∈ [0,N] then
        MAP add point
    end if
end for
```

very fast and is only evaluated on occupied voxels, making this method computationally light. The number of operations is $O(nk)$, with n the number of occupied voxels in the smallest cloud in the history and k the history size. The voxel size is predetermined and represents the resolution of our map. Figure 4.7, shows a schematic of a 2D example using a history size of three.

Figure 4.7 A 2D example of our Temporal Convolution Voxel Filtering for history size of 3. Only voxels occupied in the entire history are passed through the filter..



At the start of the application, the drone acquires the first point cloud and initializes an

empty local map. Since noise is not desirable, the TCVF algorithm is used to add points to the map. TCVF needs to fill first the entire history with sequential point clouds in order to determine whether specific points exist. However, the drone is not static, so from the camera's point of view, the points might move, even though they represent the same actual static point. The camera origin of a point cloud effectively represents the relative position of the drone to the detected scene.

The sequential clouds must be aligned in order for TCVF to work. The effect of aligning sequential clouds is also an assessment of the updated position of the drone. Iterative closest point (ICP) algorithm is used to minimize the distances of pairs of closest points in an iterative fashion, to align point clouds. However, ICP algorithms have difficulties detecting the correct transformation between two sequential point clouds if the change in pose between them is substantial.

This problem can be solved by using IMU data from the drone, to provide an assessment of the pose change and feed this to the ICP algorithm. Unfortunately, it is not possible to rely solely on the IMU data for positioning in GPS-denied environments, because it tends to drift quickly. Luckily the ICP result gives us an estimation of the drone position, so we implemented an algorithm to fuse the information from the IMU and the ICP result, to estimate the position of the drone in the map.

Traditionally, an Extended Kalman Filter (EKF) is used to fuse the visual and inertial data. The result of using an EKF is a smoothed pose estimation. There are several implementations of this idea [170, 171, 172, 173]. In particular, in [173] the effect of the biases in the IMU is studied and a solution provided. Suppose that the system's state is:

$$X_k = \{x_k^x, x_k^y, x_k^z, \dot{x}_k^x, \dot{x}_k^y, \dot{x}_k^z, \ddot{x}_k^x, \ddot{x}_k^y, \ddot{x}_k^z, b_k^{\ddot{x}}, b_k^{\ddot{y}}, b_k^{\ddot{z}}\} \tag{4.1}$$

and the observation's state:

$$Z_k = \{x_k^x, x_k^y, x_k^z, \ddot{x}_k^x, \ddot{x}_k^y, \ddot{x}_k^z\} \tag{4.2}$$

while the equations for the system and the observation are:

$$\begin{cases} x_k^i = x_{k-1}^i + \Delta t \cdot \dot{x}_{k-1}^i + \frac{\Delta t}{2} \cdot \ddot{x}_{k-1}^i, & i = x,y,z \\ \dot{x}_k^i = \Delta t \cdot \ddot{x}_{k-1}^i, & i = x,y,z \\ \ddot{x}_k^i = \ddot{x}_{k-1}^i, & i = x,y,z \\ bias_k^{\ddot{x}^i} = \frac{T}{T+\Delta t} \cdot bias_{k-1}^{\ddot{x}^i} + \frac{\Delta t+T}{T+\Delta t} \cdot (C_1 + C_2), & i = x,y,z \end{cases} \tag{4.3}$$

$$\begin{cases} X_k^i = Z_k^j, & i = j = 0...2 \\ X_k^i = Z_k^j, & i = 0...2, j = 3...5 \end{cases} \tag{4.4}$$

Introducing these equations into the EKF allows predicting the current state of the system. This information is used to locate the cameras in the environment. It is also used to provide a guess in the next iteration of ICP, by taking the current state and assessing the drone's position after Δt. The orientation is taken directly from the IMU since it is provided by the compass and does not drift. The process is summarized as follow:

1. The previous state X_{k-1} is used to obtain $\tilde{X}_k$, which is a rough estimation of the current position of the robot.

2. If the stereo system has captured good images, a point cloud is generated and aligned with the map using $\tilde{X}_k$ as the initial guess. The transformation result of the alignment is used as the true position of the drone $\hat{X}_k$. The obtained transformation is compared to the provided guess and discarded if the difference exceeds a predefined threshold.

3. If the stereo system has not captured good images, it is assumed that $\tilde{X}_k$ is a good approximation of the state, so $\hat{X}_k = \tilde{X}_k$

4. The EKF merges the information from the ICP $\hat{X}_k$, with the information from the IMU, $\hat{\hat{X}}_k$, and the resulting X_k is the current filtered state.

Figure 4.8 shows how the fusion of ICP information and IMU information gives more robust and accurate results than using only IMU data or ICP results separately. The RGB refer to XYZ respectively. Dotted lines is the estimation of position using just IMU information. As mentioned in Section 4.1, this tends to drift due to the accumulation of errors. Dashed lines are the estimation using only ICP. These results are initially good if all the input clouds are confident and input images are not blurry. But, un iteration 120 the algorithm converges to a wrong solution, shifting the estimation of the position of the robot. Finally, solid lines are the estimations of the drone position from the fusion algorithm. It returns stable and robust estimates of the position of the drone. The progress of building the local map of the scene can be seen in Figure 4.9.

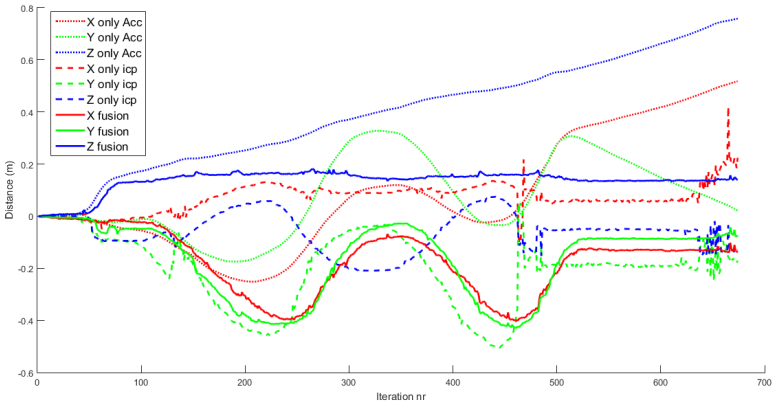


Figure 4.8 Comparison of drone positioning using the EKF with only IMU data; only ICP results; fused IMU data and ICP results. Using only IMU data, the position drifts away quickly. Using only ICP results the position has several bad discrete jumps and doesn't correspond to actual motion. Using the fused data the position corresponds to actual motion..

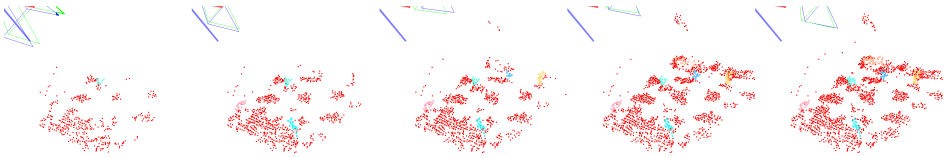


Figure 4.9 Local map at different iterations. .

The author used this technique to elaborate the map that is used for detecting objects in the environment as described in Sections 3.2 and 3.6.1. Additionally, the work is part of the publication [11].

4.3 Full SLAM Framework using RGB-D sensors

In this section, it is shown a general purpose framework for full SLAM using RGB-D sensors which exploit both the RGB and Depth pictures for localizing the robot and creating a dense map. This framework has been developed and implemented as open-source resource as is described in Appendix A.1. The purpose is to make a modular framework to allow fast development of applications that requires SLAM. Additionally, each of the modules is coded in such way that can be replaced with other with similar functionalities, allowing the integration of new methods and algorithms. The purpose is to allow developers to focus in single problems while being able to exploit all the functionalities. Figure 4.10 shows the complete pipeline of the algorithm and the modules involved in it.

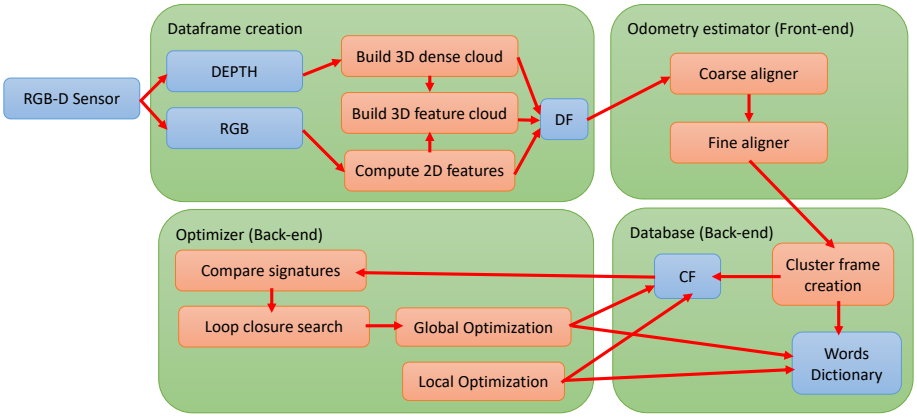


Figure 4.10 RGB-D SLAM pipeline.

This framework has been developed based on the advantages of existing SLAM Frameworks such as [146, 161, 174]. Additionally, the lessons learned from the methodology described in the previous section have been beneficial. This framework intends to solve

the full SLAM problem, obtaining the localization of the robot over time and a keyframed map based in graphs. This new structure allows to perform not only the visual odometry estimation of the pose of the UAV but also perform optimizations in it improving the elaborated map and making the odometry more robust to sequential drifts.

This section is structured according to the modules shown in Figure 4.10. At first instance, it is introduced how the data is captured from the sensors to elaborate a bundle pack called Data-Frame which contains useful information for the SLAM algorithm at each instant. Then, the visual odometry algorithm is described using the Data-Frames to acquire an estimation of the actual position of the robot. The obtained pose is stored in the Data-Frame for using it later. The sequential acquisition of the position of the robot is usually called front-end of a SLAM algorithm. The latter part of the SLAM algorithm is called back-end and consist on two different modules. The first module, or Database, is responsible for the management of the results of the front-end to elaborate a graph-based map of the environment. This module selects highlighting Data-Frames and store them as nodes of the graph, that are called in this framework Cluster-Frames. Additionally, the strongest features are stored separately in a dictionary, these features are called words. Finally, while the map is being created two different processes perform optimizations in the map. The first optimization occurs locally in the vicinity of the position of the robot. This local optimization fixes small drifts and eliminates bad words in the dictionary. A second global optimizer works looking for loop closures and fixing drifts in the position of the robot on a larger scale.

4.3.1 Sensors and DataFrame creation

Each instant of time, the camera captures important information that is used for locating the robot, and potentially to be used in other tasks. This information is used into computes robot position. The unit of data of the framework is called Data-Frame and contains the following information:

- a color image captured by the device.
- a depth image captured by the device or computed by some algorithm.
- a unique identifier ID of the Data-Frame.
- list of features obtained from the color image
- colorized point cloud obtained from the deprojection of the depth and color image.
- sparse point cloud obtained from the deprojection of features computed in the color image
- the pose calculated by the odometry estimator.

The information can be either obtained from the sensors or computed online. Active stereo cameras usually provide in the same shot the color and the depth image, while passive stereo cameras required the depth map to be computed in a separate pipeline. Once both images are obtained, two processes compute two clouds. The first cloud consists of a sparse cloud obtained by the deprojection of the features detected in the color image.

The second cloud is the complete deprojection of the depth image, obtaining a dense point cloud which can be used to perform a finer alignment process in the odometry estimator.

The rest of the information is gradually filled in the pipeline. Lately, the Data-Frame is given to the Database to store it for optimizations or just data query.

4.3.2 Odometry estimator

In SLAM, odometry is the basic operation which computes a fast, but rough localization of the robot. In this work, the odometry estimator relies on visual clues from an RGB-D device, which are stored in the Data-Frame. Algorithm 6 summarizes the pipeline of the odometry estimator using visual clues.

Algorithm 6 Visual odometry estimator

1: $df_i.rgb, \quad df_i.depth \leftarrow camera\_grab()$
2: $df_i.features \leftarrow compute\_features(df_i \leftarrow rgb)$
3: $df_i.feature\_cloud \leftarrow compute\_rough\_feature\_cloud(df_i \leftarrow depth)$
4: $df_i.matches\_prev \leftarrow match\_features(df_{i-1} \leftarrow features, df_i \leftarrow features)$
5: $\{df_i.pose, inliers\} \leftarrow rough\_transform\_ransac(df_{i-1}, df_i)$
6: **if** $df_i.n\_inliers > inliers\_threshold \quad \& \quad is\_valid(transformation)$ **then**
7: $\quad validate\_dataframe()$
8: **else**
9: $\quad reject\_dataframe()$
10: **end if**

As mentioned before, at each instance, the algorithm captures information from the camera and store it into a Data-Frame. The first step then is to subtract visual clues from the color images and create a point cloud with descriptors. Visual features have been intensively mentioned in previous in Sections 2.2.2 and 3.4 for their use in object detection and modeling. Any of the features can be used in this framework, but ORB has been chosen due to the computation limitations in UAVs' onboard computers.

Once the features are computed on the RGB image, their location in pixels on the 2D image is transformed to 3D using the depth picture and the calibration of the sensor. This generates a point cloud in which each point has a descriptor, called feature cloud. Then, these descriptors are used to match the point clouds of sequential Data-Frames.

Up to this point, feature clouds are computed related to the camera's frame. It means that they are referred to a local coordinate system attached to the camera. In order to compute the relative position of sequential cameras all the information computed above is used. There is a large variety of method for computing the incremental transformation between two camera frames using visual clues. In this work, the 3D information in the sparse point cloud is used, together with the matches between sequential features, to obtain the relative position of the cameras in a robust manner.

A rough classification of algorithms can be gradient descent and sampling algorithms. Gradient descent algorithms are usually characterized for obtaining more accurate solutions with the main drawback of needing a proper initialization. This drawback arises from their definition as gradient descent methods minimize a global function which may fall in local minimums. In contrast, sampling based algorithms are characterized for being more robust and not to fall in local minimums, but solutions are usually less accurate. Additionally, gradient descent usually needs the definition of step sizes and convergence functions which makes their convergence time undefined in some situations. While sampling algorithms just need the definition of a fixed number of samples which bounds computational time.

Our odometry estimator uses RANSAC [96] with a refinement stage to take advantage of its robustness and obtaining a more accurate solution. The matches between sequential feature clouds are used to sample possible transformations and eventually the one with more fitness score is used. The algorithm computes the relative transformation between the pair of frames, and then adds it to the previous Data-Frame pose to compute current position.

A further refinement can be performed in the alignment using the dense clouds stored in the Data-Frames. Given the initial robust estimation of the position of the camera, gradient descent algorithms such as ICP are optionally used. This step produces much more accurate results, and by the use of previous rough position the probability to converge increases. However, it is usually more time-consuming. Thus, if the computational resources are limited, this step can be skipped.

Additionally to the visual clues, it is possible to integrate extra information into the system to improve the estimation of the odometry. Similar to the algorithm exposed in Section 4.2 the IMU information can be acquired to be fused with the EKF or using a particle filter.

4.3.3 Database module

This section presents the Database module. This module is responsible for the efficient creation of the map. The map is composed of two principal components: Cluster-Frames and words. Cluster-Frames are those Data-Frames that are chosen to be representative. Words are those feature points which are strong and repetitive, so they are stored as representative points. These two components are then used to generate a graph-based map in which Cluster-Frames and words are nodes in the graph, and the edges are the corresponding projections of the words into the specific Cluster-Frames.

Words contain the following data:

- Data-Frame DF_i in which the word has been observed.
- a descriptive descriptor that is used for local optimization and detection of loop closures.
- its position in the space.
- its projection at each Data-Frame and Cluster-Frame
- list of invalidated projection.
- cluster in which word has been observed for optimizations

Cluster-frames are containers of information from different data-frames. These act as melting pots of information which is enhanced over time from input data. Additionally, keeping the similarity with keyframes, these are used as reference positions for the localization algorithm, but these are prepared to collect (or cluster) the information from related data-frames. Cluster-Frames contains the following information:

- unique identifier ID
- Position and orientation
- signature
- covisibility
- words associated
- associated data-frames

All the sequential data is gathered in Data-Frames. However, storing every single Data-Frame is extremely memory consuming and highly redundant. To solve this problem, information from sequential Data-Frames is collected in the Cluster-Frames. These contain the useful information needed for the localization problem and also to elaborate a map of the environment.

For each input Data-Frame, the database computes a representative a signature which describes its visual information. This signature is used to determine its membership to a Cluster-Frame. A signature is assigned to clusters too, in order to unequivocally identify them. Meanwhile, the database selects those features that are repetitive and correlate them which the clusters. If several features are observed between different clusters, these clusters are considered to have covisibility. This covisibility is important to perform later the optimizations algorithms.

Being able to efficiently store all the information acquired in the environment to build a map simultaneously to the localization of the robot is vital. For this purpose, the more useful information is stored in a graph. This graph-based map based holds the necessary information about the structure of the map. In the proposed framework, each node is represented with a Cluster-Frame, and each landmark on the map is a Word. Each Cluster contains the words which has a projection in the cluster. The information is not stored intrinsically in a graph shape, but the graph can be quickly built from the data. As mentioned in the introduction, the purpose is to make the framework modular. Thus, this fact is useful to not to particularize to a specific algorithm, keeping the framework flexible for new algorithms and methodologies.

4.3.4 Optimizer Module

As it is well-known, the sequential alignment of the data from the sensors (or visual odometry in this case) is prone to produce drifts overtime while computing the localization of the robot. This section introduces two levels of optimizations to reduce the drift. The first optimization level occurs locally at the same time that the odometry estimation. The second level of optimization involves long-term improvements which turn into action when loop closures are detected. For this purpose, the graph-based map representation is used.

This optimizations prevent the dead-reckoning and minimize the drift in the position of the platform. In this work, the optimization is performed using g2o [161] framework.

The first optimization, or Local optimization, is based in the covisibility between the cluster-frames in the vicinity of current robot pose. The optimization consists of performing Bundle Adjustment (BA) using the last information in the graph map. BA algorithm has been introduced previously in Section 3.4 for the optimization of the reconstruction of objects using sparse clouds. In this section, Sparse Bundle Adjustment [101] is used similarly to minimize the re-projection of the words onto the images of the cluster-frames to reduce the drifts produced by the visual odometry.

All the information captured previously is stored in a graph in which each node is denoted x_i and represent a cluster-frame of the database, and holds a pose in the space and a unique identifier. Also, let be z_{ij} the observation of a specific feature j observed from cluster-frame i. Finally, the constraints between the nodes and the observations are denoted with an edge e_{ij}.

In general terms, it is possible to construct a cost function to minimize the errors in the positions of the clusters with the following form:

$$F(x) = \sum_{(ij)\in\mathscr{C}} e_{i,j}\Omega_{ij}e_{ij}x^* = argmin_x F(x) \tag{4.5}$$

Where $\Omega_{i,j}$ represents the mean and information matrix of the constraints represented by the projections of the features observed and the poses of the cameras held by the cluster-frames. If the initial information is good enough, the algorithm is ensured to converge. The basic idea behind is to approximate the error function by its first order Taylor expansion in the initial guess $\hat{x}$

$$e_{ij}(\hat{x}_i + \Delta x_i, \hat{x}_j + \Delta x_j) \simeq e_{ij} + J_{ij}\Delta x \tag{4.6}$$

For the local optimization, let be CF_i a cluster-frame in the map which holds the observed features and its pose. Also, let the covisibility be

$$CF_{covisibility} = \{CF_j \forall j = 1...N\}$$

A set of cluster-frames which share a minimum of observed features, i.e., they share a visible part of the environment, so $j = 1...N$ represent the co-visibility of the cluster-frame. The bundle adjustment is performed in that specific subset of cluster-frames to improve the poses of all the clusters in the co-visibility locally and consequently the representation of the features or map-points.

Up to this point, the algorithm produces a map and robot's localization using visual odometry and local optimizations. However, the accumulation of small errors in the system ended up biasing the real location of the robot. To solve this problem, loop-closing algorithms detect when the robot re-visited someplace and correlate its position with the position in the past. By adding this connection in the graph-map, the global optimization algorithm is able to erase the drift produced by the odometry, fixing the past trajectory and consequently the map generated. With this concern, two methods have been explored.

The first method is based on the construction of a similarity matrix between the existing Cluster-Frames and has been implemented and integrated in the framework. Each time a cluster i is created, the loop-closure detector computes a similarity score between it and the other Cluster-Frames in the database based on their unique signature [156] and place it in a $\mathbb{R}^{NxN}$ matrix. Figure 4.11 show the score matrix computed in a dataset. The left picture shows the raw matrix with the similarity scores. The right picture shows an optimized version using the algorithm proposed in [175]. Authors proposed an analysis of the singular values of the score matrix to remove those values that are not significant. The algorithm proceeds by computing the singular value decomposition, then reconstructing the matrix but setting those non-significant singular values to 0.

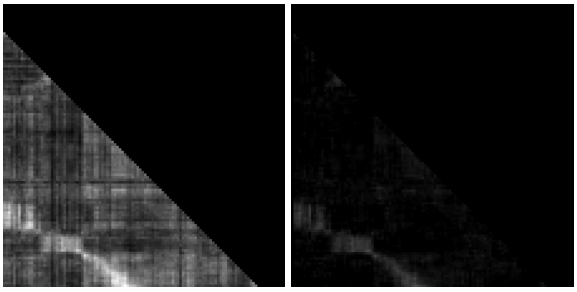


Figure 4.11 Score matrix from signature to detect loop closures. Left matrix shows the raw scores and right matrix shows the optimized matrix using rank reduction..

Then the loops are detected by applying Water-Smith algorithm [176] in the lower triangular fragment of the rank reduced matrix. Figure 4.12 shows the resulting binary matrix. The sequence of all positive values (i, j) correspond to a loop in the sequence.

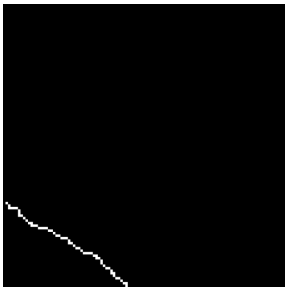


Figure 4.12 Result of the Water-Smith algorithm for loop closure detection.

However, this method scales poorly in extensive experiments due to the need for creation of the score matrix for each Cluster-Frame with all the existing Cluster-Frames. For this reason, another alternative has been explored. The loop closure detector developed by Dorian et al. in [156] has been used as an alternative, which is less time-consuming and scale better with large experiments. This algorithm detects loops by comparing signatures of Cluster-Frames in a database. The comparison between images is made more efficiently

by the use of trees. Then, if the comparison passes a threshold, the geometrical distribution of the features used for computing the signature is compared using RANSAC. If the RANSAC algorithm gives a solution and the number of inliers is above a threshold, the loop is considered detected, and both nodes are connected in the graph.

Eventually, once a loop is detected by either of the aforementioned methods, the framework proceeds to optimize those cluster-frames that are included in it. This is done, again in our framework, using Sparse Bundle Adjustment. In this case, all the frames integrated into the loop are introduced in the minimization problem while the rest remain unused. Once the optimization algorithm has finished, all the poses are updated and the branches attached to to the optimized loop are moved too according to the base of the branch.

Additionally, after the optimization process, all the edges in the graph-map are analyzed. The optimization algorithm returns a score based on the probability of the deviation of the measurements using a Chi-Square distribution (χ^2). If the score of an edge, i.e. a projection ,is above a threshold, it is invalidated, assuming that it is a bad match. Removing this bad matches is essential, as if they grow over time, they can make the algorithm diverge even if the initial condition is good. This elimination process also occurs in the local optimization to reduce the mismatches while the map is being created.

4.4 Experimental Validation of the Framework

This section presents the validation experiments of the proposed SLAM Framework. The purpose is to evaluate the accuracy of the system using different datasets. However, it is difficult to elaborate custom datasets for aerial platforms due to the need for accurate ground truth to compare the results against. For these reasons, the first two subsections evaluate the framework using two standard datasets. The last subsection presents an experiment that the author was able to record with a custom aerial platform under a bridge, in which the position of the Drone was recorded using an external positioning tool called Total Station as ground truth. Finally, a semantic slam approach is presented using the algorithms presented in Chapter 3.

4.4.1 Experiments in Microsoft 7-scenes RGB-D Datasets

This section shows the experimental results on the 7-scenes dataset by Criminisi et al.[177]. This dataset was recorded using a handheld Kinect RGB-D camera. The datasets consist of several sequences in seven different scenarios. They are shipped with a ground truth obtained with the KinectFusion[178]. The only objection is that they do not provide any calibration file for the camera but a raw focal length ($f_x = f_y = 585$) and principal point ($c_x = 320$, $c_y = 240$).

Table 4.1 summarizes the results of the proposed SLAM framework with and without optimizers. As there are a total of 50 sequences, only the first two of each scene are shown in this table. As the framework only stores the cluster frames, these are the one compared against the ground truth. An example of map reconstruction can be seen in Figure 4.13. Each Cluster-Frame is represented with a red number, and the green lines represent the covisibility between them.

	Avg. error	
dataset	visual odometry	loop closure
chess - seq01	0.043091	0.035963
chess - seq02	0.050865	0.0497788
fire - seq01	0.095764	0.027084
fire - seq02	0.085799	0.043113
heads - seq01	0.080163	0.038983
heads - seq02	0.071898	0.046259
office - seq01	0.254695	0.041244
office - seq02	0.345024	0.124132
pumpkin - seq01	0.080049	0.084454
pumpkin - seq02	0.098603	0.068412
redkitchen - seq01	0.060107	0.059255
redkitchen - seq02	0.168820	0.096546
stairs - seq01	0.206929	0.046391
stairs - seq02	0.057993	0.035623

Table 4.1 Average error in microsoft 7-scenes dataset.

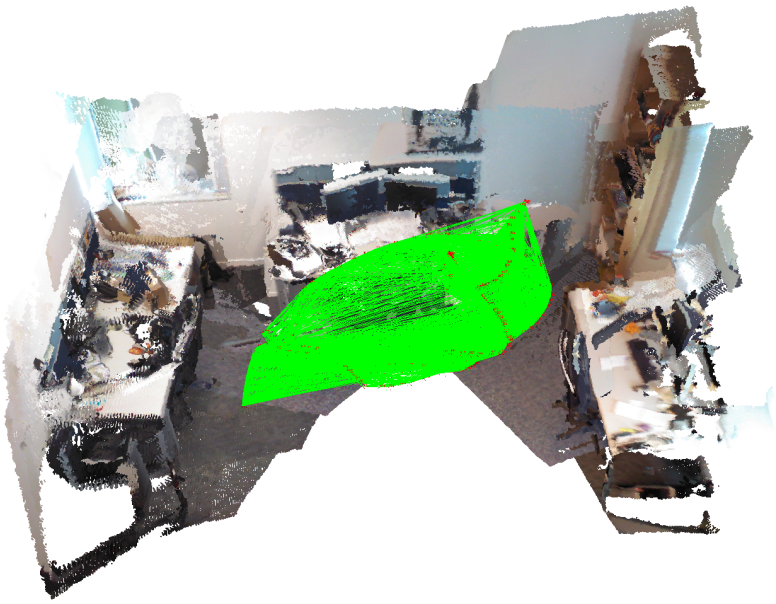
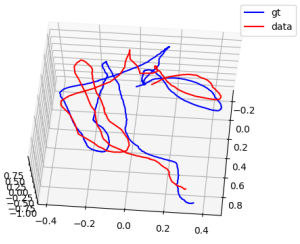


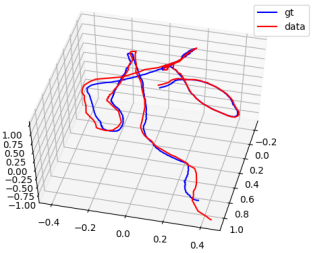
Figure 4.13 Sample map reconstruction in the Office sequence 01.

Additionally, Figure 4.14 show the 3D trajectories of the cameras in some sample datasets against the given ground truth. For clearness of the chapter, the rest of the trajectories can be found in Appendix A.1.

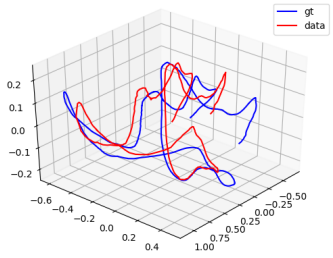
It can be clearly seen that the optimizations pay an essential role in both the localization



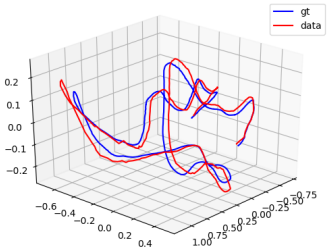
(a) Fire sequence 02.



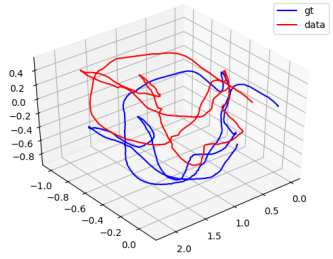
(b) Fire sequence 02 - with optimizations.



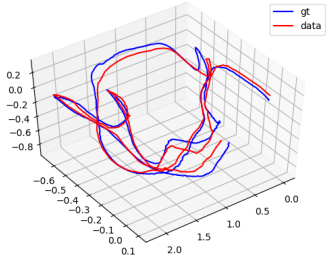
(c) Heads sequence 01.



(d) Heads sequence 01 - with optimizations.



(e) Office sequence 01.



(f) Office sequence 01 - with optimizations.

Figure 4.14 Sample trajectories computed by the SLAM framework versus ground truth in Microsoft 7 scenes. Pictures in left column show the results without optimizations. Right column picture shows the results using both local and global optimizations..

and the mapping of the environment. If the optimizations are applied, the average errors in the cluster frames are reduced between 25% and 65% depend on the dataset.

4.4.2 Experiments in TUM RGB-D Datasets

The TUM RGB-D benchmark [179] provides a good number of datasets for evaluating the accuracy of the localization of the RGB-D sensor in several sequences. These datasets are composed by a set of color and depth images acquired with a Kinect camera. Additionally, the authors provide a highly accurate position of the camera using an indoor motion capture system.

Table 4.2 shows the average errors of the algorithm in some of TUM's RGB-D datasets. An example of map reconstruction can be seen in Figure 4.15.

	Avg. error	
dataset	visual odometry	optimizations
fr1_xyz	0.021423	0.009089
fr1_floor	0.089290	0.083565
fr1_room	0.208355	0.178355
fr2_xyz	0.011387	0.006515
fr2_desk	0.159498	0.138194
fr2_desk_person	0.076316	0.052962

Table 4.2 Average error in freiburg datasets just with VO, and with VO and optimizations..

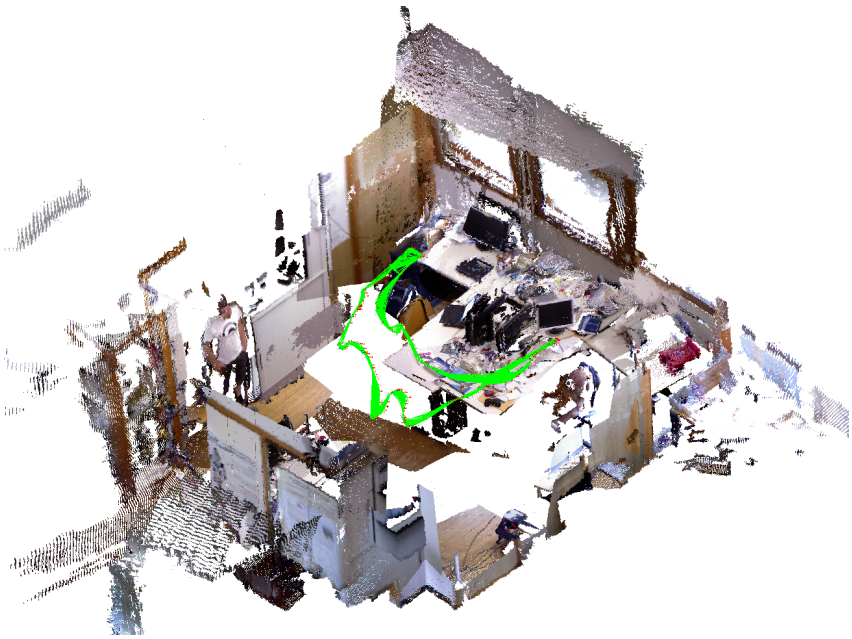
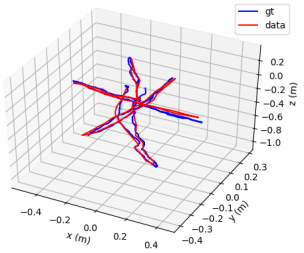
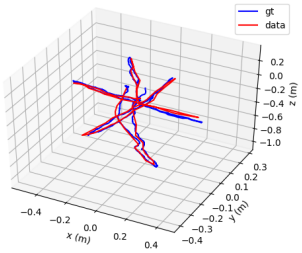


Figure 4.15 Sample map reconstruction in the freiburg1_room dataset.

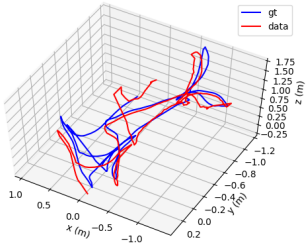
Additionally, Figure 4.16 show the 3D trajectories of the computed by the algorithm with just visual odometry and with the optimization module running, respectively.



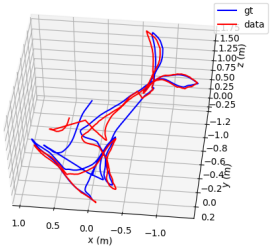
(a) Dataset Freiburg 2 XYZ .



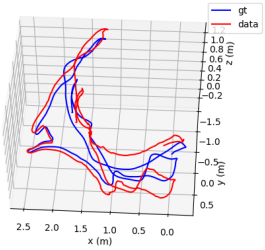
(b) Dataset Freiburg 2 XYZ - with optimizations.



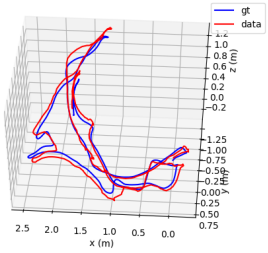
(c) Dataset Freiburg 1 floor.



(d) Dataset Freiburg 1 floor - with optimizations.



(e) Dataset Freiburg 1 room.



(f) Dataset Freiburg 1 room - with optimizations.

Figure 4.16 Freiburg datasets just visual odometry.

Similar to previous datasets, the results are unequivocally better if optimizations are applied. However, it is worth to mention that the results in the dataset fr2_desk_person, results deteriorate in the last fragment of the datasets due to the person moving in the scene. In this case, the visual odometry seems to be more robust thanks to the robust inlier rejection by RANSAC. However, during the last fragment of the dataset, the person moves

the objects in the table, which causes mismatches in the past projections of the words. In future implementations of the framework, the system will watch out this temporal-spatial mismatches of dynamic objects to prevent the optimization to diverge.

4.4.3 Custom dataset - Flying under the bridge

This subsection presents the results of the reconstruction of the pillars of a bridge using our framework. The results are compared against the position captured by a Leica Total station. This device tracks a prism located in the UAV and records its position. Thanks to this device, it is possible to obtain a ground truth of the experiments in almost any environment if the platform keeps in the line of sight of the ground device. Figure 4.17 shows the 3D map reconstructed by the SLAM algorithm and the trajectory of the UAV compared against the ground truth. An average error of $0.1056(m)$ is obtained in the experiment shown.

This experiment is important not only for introducing a different context but for being a real outdoor dataset using a flying platform. With it, the author wants to show the capabilities of the framework in the target circumstances concerning the movements and vibrations typical of the aerial platforms.

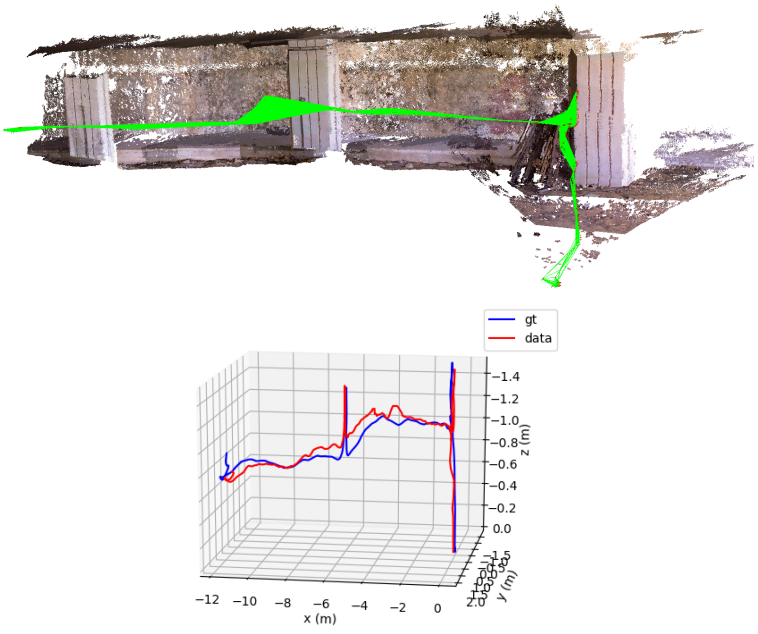


Figure 4.17 Bridge pillars mapping.

4.5 Semantic labeling for manipulation and Augmented data for human operators

In this section, results from the object detection algorithms presented in Chapter 2 are used to enhance 3D maps. With this work, it is pretended to fully automate the manipulations tasks to be developed by the aerial platform, as the UAV will be able to, at first instance, locale itself in the environment, and to locate the target objects in the environment. Thus, at last instance, the robot could perform manipulation task as described in Chapter 3.

This information is highly valuable because it provides information about the location of the objects in the scene. With this knowledge, the robot can plan for different tasks such as transporting the objects from one place to another. This work focuses on the localization of the objects in the scene. Once the objects are located, the rest of the algorithms presented will be used to grasp the specific objects.

Semantic labeling [180] refers to the field of computer vision which categorize 2D or 3D data into human-comprehensible information. In some way, computer vision always tries to categorize the information of the environment, but in this particular context, the objective is, in general terms, to place a label on physical objects which can be useful for some task. This task is very correlated with segmentation tasks in robotics. Usually, segmentation is performed using visual information which characterizes an object to isolate it from the environment. It is possible to use contextual information to help this segmentation. This has been recently studied by using deep learning techniques [181].

Custom datasets have been recorded in which several instances of objects appear in the scene. The algorithm computes the localization of the robot at the same time that it builds up a map of the environment. In parallel, each image is processed using the object detection neuronal network, described in Section 3.6.3 to detect instances of objects. These instances in 2D are projected onto the point clouds to generate semantic 3D information.

Each time that an object is detected with a score higher than a threshold, it is projected onto the point cloud generating a set of points in which the object is assumed to be. Then, using Principal Component Analysis (PCA) the minimum oriented bounding box containing the points is computed using the following equation.

$$
\begin{aligned}
P = X_i &= (x_i, y_i, z_i) \forall i = 1...N \\
k(x,y) &= \frac{\sum_{i=i}^{N} (x_i - \bar{x}) * (y_i - \bar{y})}{N} \\
K &= \begin{bmatrix} k(x,x) & k(x,y) & k(x,z) \\ k(y,x) & k(y,y) & k(y,z) \\ k(z,x) & k(z,y) & k(z,z) \end{bmatrix} \\
PCA &= \{\sigma_{\lambda_i}(K), E_{\lambda_i}(K) \quad \forall i = x,y,z\}
\end{aligned}
\tag{4.7}
$$

These detections are associated with current Cluster-Frame to be stored in the map. Then, it could happen in subsequent instants that the same object is detected by the network, and consequently, a second bounding box is obtained. In order to prevent several instances of

the same object to appear in the scene, for each new detection, the volumetric Intersection over Union (IoU) is computed by

$$\begin{aligned} V_1 &= width_1 * height_1 * altitude_1 \\ V_2 &= width_2 * height_2 * altitude_2 \\ V_{intersec} &= V_1 \cap V_2 \\ IoU &= \frac{V_{intersec}}{V_1 + V_2 - V_{intersec}} \end{aligned} \tag{4.8}$$

This value measures the overlap between the two regions, and if the overlap is significant, it means that two predictions belong to the same object. If a new detection has a null or low IoU score, a new instance of an object is added to the scene and attached to the cluster frame. This link to the cluster frame is very important because if the cluster pose is updated due to the optimization algorithm, the pose of the detected object needs to be updated too.

Figures 4.18 and 4.19 show the reconstruction of an indoor office in which there different mockups of the crawler robot shown in Section 3.6.3. The results of the trained networks showed in that section are used to obtain the semantic information.

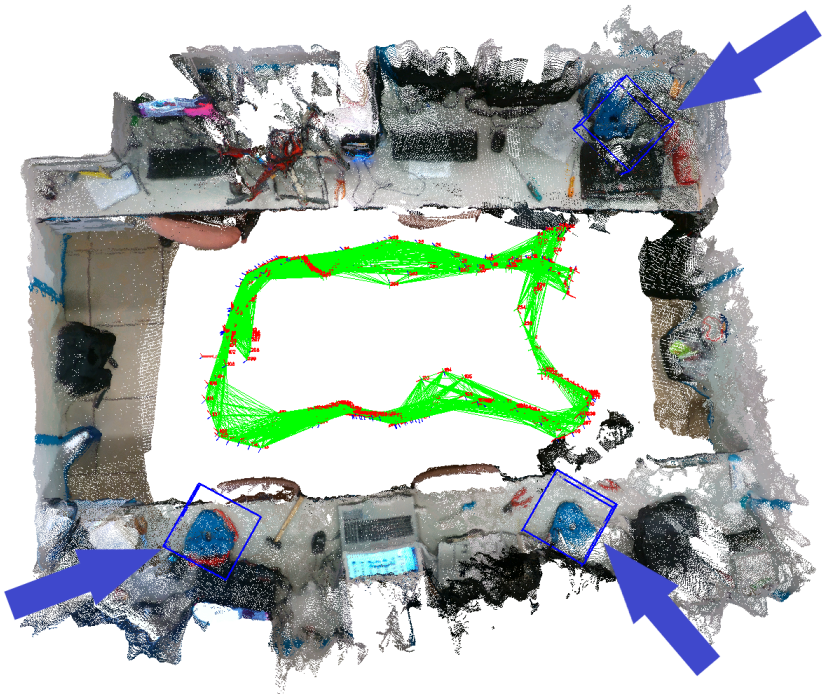


Figure 4.18 Top view of office dataset with semantic information from the neuronal network trained with the crawler dataset.

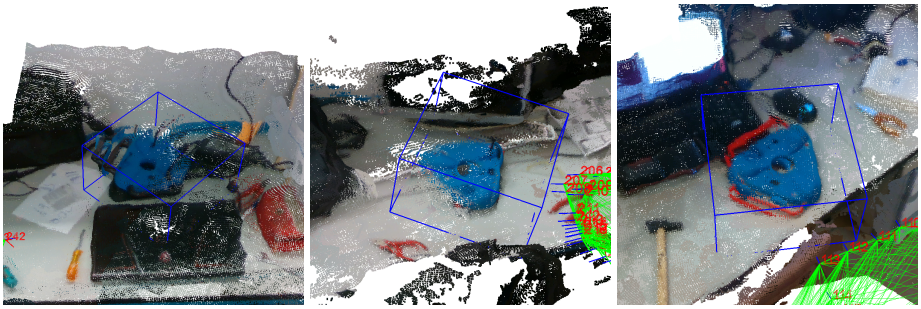


Figure 4.19 Close-up view of crawler detections .

In the same scene, different hand tools were placed. Using the trained network shown in Section 3.6.3, a second run of the experiment was performed obtained the results shown in Figures 4.20 and 4.21.

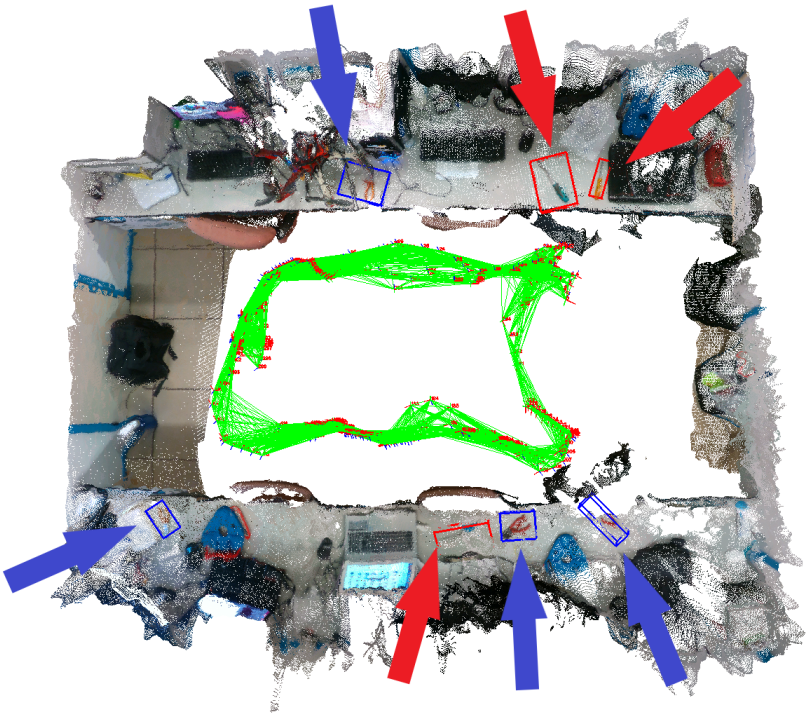


Figure 4.20 Top view of office dataset with semantic information from the neuronal network trained with the hand-tools dataset.

Future work in this researching line will include more complex networks which detect, not only the bounding box in the 2D image but the exact pixels that belong to the object.

Figure 4.21 Close-up view of hand-tools detections.

This information is advantageous because the 3D projection of the points includes only the specific points that belong to the object, contrary to bounding boxes that may include points that belong to the background. Further conclusions are thrown in Chapter 6.

5 Aerial manipulator platforms and general system architectures

5.1 Introduction

Multirotors have been proved to be practical solutions for developing inspection and maintenance tasks [6, 182, 183]. Contrary to fixed-wing UAVs, multirotors can move freely in the space and hover at a desired position to develop any tasks.

Many approaches exist in state-of-art to the problem of aerial manipulation. At first, it is important to consider the expected specifications that need to be accomplished by the aerial robot. Then, the UAV needs to be equipped with an adequate set of tools to accomplish these tasks.

First approaches provided the aerial platforms with a single gripper placed just at the bottom of the UAV. A well-known example of this kind of aerial manipulators is Yale's helicopter [184, 185] Dollar et al. researched the use of a helicopter with a built-in gripper downside for transportation purposes. However, the previous platform is large and not suitable for many operations. Authors in [186] proposed a smaller design using a quadcopter which was equipped with a gripper downside too. In that article, authors show the design and the resulting platform which has been tested indoor using a VICON system. However, these kinds of actuators can merely be used to grab-and-drop applications. Many other small low-complexity grippers for multirotors have been designed [187, 188]. Other grippers can also be magnetic [189] reducing the complexity and the mechanical parts.

Other authors, studied the use of manipulators to exert forces to objects or surfaces. Authors in [190] studied the design of a tiltrotor with a sensorized rod to control the force exerted. Another example is manipulators with force control as shown in [191] in which authors developed a 1DoF manipulator with exerting controlled forces to walls for bridge inspection. In these works, authors govern the position of the aerial platform to exert controlled forces to objects. In the AEROARMS project [7] a contact force inspection tool was developed which actively uses the force exerted to control the position of the UAV.

Conversely, a passive tool for controlling the position of the multirotor is shown later in Section 5.4.

New generations of aerial manipulators are provided with full serial robotic arms to extend the capabilities and applications of these platforms. In this case, more complex operations can be performed as the embedded manipulators have more dexterity and allow the platform to remain static while the manipulators are moving to perform an operation. Robotic manipulators are a wide area of research. However, due to the nature of this research, this document focuses on lightweight manipulators due to the strict payload limitations of aerial vehicles.

Parallel manipulators [192, 193] have been recently studied for their use in aerial platforms. The main advantage of this kind of manipulators is that they are relatively fast and accurate being able to maintain robustly its position against perturbations. However, the reachability of these arms is limited. Serial arms are suitable for more applications. Many articles explored the use of a single arm installed in the aerial platform to carry out a variety of tasks. K. Kondak et al. developed a helicopter equipped with an industrial robotic arm to perform manipulations task as shown in [194][195]. Currently, a rising interest in aerial manipulation leads to researches to place not one but two manipulators in the same platform to allow it to perform dexterous manipulation with both hands. Authors in [196] proposed the use of a pair of manipulators to turn valves in industrial environments. Dual manipulators [197, 198] configuration has been found to be very versatile to enhance manipulation tasks.

In this dissertation, aerial manipulators with embedded serial robotic arms are the case of study. These are more efficient and flexible, concerning the reachability, to perform a large variety of tasks which imply dexterous manipulation.

Eventually, it is not hard to understand, that flying aerial platforms with embed manipulators introduce an extra challenge. These attachments produce static and dynamic instabilities on the platforms. Additionally, the interaction with physical objects during the manipulation tasks, or even the interaction with fixed objects such as walls, induces more problems.

For a better understanding, this Chapter focuses on the hardware implementation and the architecture of the software used during the research to test all the manipulation algorithms exposed in previous chapters. A proper control design that integrates the movements of the manipulators in the dynamic model of the multirotor is beneficial for the stability and control of the platform. However, these concepts are out of the scope of this research.

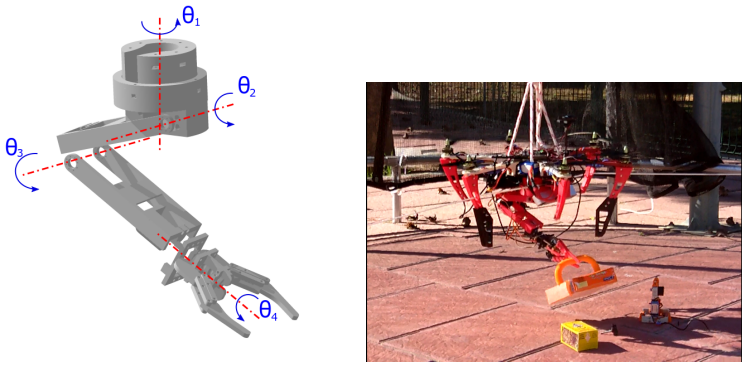
The remainder of this Chapter is divided into four sections. Section 5.2 introduces the first platform developed; a small hexacopter with a single manipulator. Section 5.3 shows the last platform, a medium size hexacopter with larger payload capabilities with a dual manipulator embedded. Section 5.4 presents a custom designed tool which is used as end-effector of one of the manipulators shown in Section 5.3. This tool docks to a structure and is used to compute the relative position of the UAV to the attachment point. One of the purposes of this docking tool is to perform manipulation tasks with a second manipulator while it remains in contact with a stiff object.

5.2 First approach with single manipulator

The first platform developed during this research was based on a commercial hexacopter DJI F550 with customized engines to maximize the thrust. This platform was called Aquiles. The hexacopter model was chosen for two reasons; the first one is that it has more power than smaller multirotors, which is very convenient for carrying the payload of the arm and the onboard computer for running the algorithms. Additionally, hexacopter configuration is more stable and robust to failures, as it can work better in case of losing one engine or propeller.

In order to evaluate any algorithm, an onboard computer has been embedded in it. To ensure enough computational power, the computer that was used was an INTEL NUC5i7RYH [199]. This compact computer has a CPU i7 3.1 GHz and 8 GB of RAM. An Arduino Uno [200] board was added as an interface between the computer and the manipulator. Further, a specific holder for cameras is placed close to the arm to perform the visual manipulation tasks. Finally, a Pixhawk[1] autopilot is used to control the platform. This hardware has the advantage of being open-source and relatively easy to command from the onboard computer.

The requirements of the robotic arm were: to be lightweight, to have as long range as possible and having 4-DOF to accomplish the grasping task. Figure 5.1a shows the CAD design of the parts ,which were built by 3D printing. Joints axis and angles have been highlighted. The arm is actuated using PWM servos controlled by the onboard computer through the analog pins of the Arduino. These servos have a strength of 25 Kg/cm and are placed directly in the joints. Finally, Figure 5.1b shows the whole structure that we built.



(a) Cad design of the 3d printed parts. (b) Final built platform.

Figure 5.1 Model of robotic arm for the aerial robot..

Special care was taken in the design of the first rotation joint. In order to prevent the actuator of that joint to hold the whole weight of the arm and the grasped object, it has been designed to hold the load structurally as shown in Figure 5.2. This structure is greased to reduce the friction as much as possible.

[1] *https://pixhawk.org/*

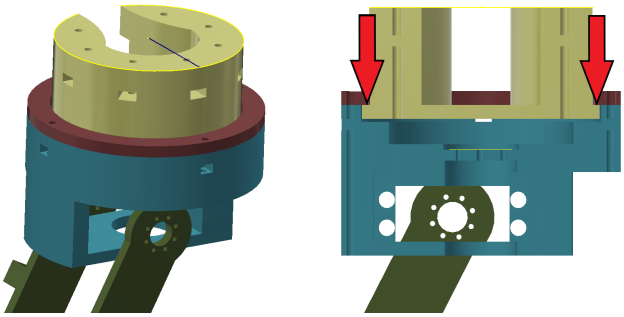


Figure 5.2 Details of first joint. Left picture shows the general assembly of the joint. Right picture shows a cut-view. The red arrows show the structure that holds the weight of the arm..

Due to the simplicity of the design, the inverse kinematic of the arm can be computed analytically by:

$$[\theta_1,\theta_2,\theta_3] = F(x,y,z) = \begin{cases} \theta_1 = atan(y/x) \\ \theta_2 = acos((l_2^2 - d^2 - l_1^2)/(-2*d*l_1)) \\ \theta_3 = acos((d^2 - l_1^2 - l_2^2)/(-2+l_1*l_2)) \end{cases}$$

being, $d = \sqrt[2]{p^2+z^2}$ and $p = \sqrt[2]{x^2+y^2}$. The additional joint at the end-effector provides for an extra rotation to accommodate the orientation of the gripper for the grasping operation.

Additionally, the arm is provided with a gripper as end-effector. It is crucial to design a gripper with enough strength to prevent the objects to slip or fall when manipulating them. Bearing this in mind, it was designed with a worm drive to pull a bar which closes the gripper. This mechanism is significantly stronger than those grippers in which the servo actuates a connecting rod directly. Figure 5.3 shows a close up picture of grippers assembly CAD. The arm had a total weight of 0.72 Kg, including the servos, wiring, and electronics.

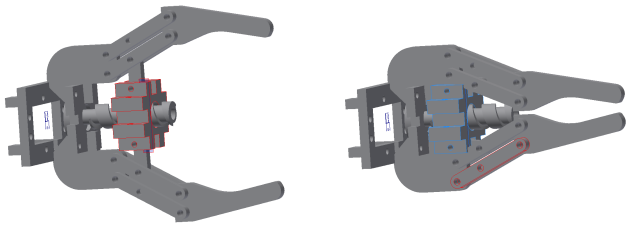


Figure 5.3 Gripper design with worm drive actuator.

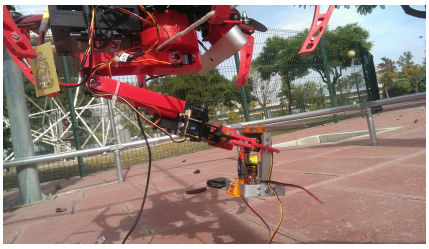
Additionally, a transformation is needed between the coordinate system of the camera

holder and the coordinate system of the arm. This transformation is composed of a translation between the centers of the coordinates and a simple spin on the X axis:

$$T^{C\leftarrow A} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & cos(\alpha) & -sin(\alpha) & t_y \\ 0 & sin(\alpha) & cos(\alpha) & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The parameters of the transformation are experimentally obtained as $\alpha = 30$, $t_x = 0.06m$, $t_y = 0.1m$, $t_z = 0$.

(a) Secure structure for testing..



(b) UAV grasping the drilling tool..

Figure 5.4 The left figure shows the safety structure. The UAV is hanging from the top. The right image shows a close-up picture of the bottom side of the robot after grasping the drilling tool..

This hexacopter has been employed in real experiments, and mainly was used for the validation of the grasping algorithm explained in Section 3.4, published in article [8].

5.3 Aerial dual manipulator

The second platform was also a coplanar hexacopter. However, in this case, the platform is larger than previous one, thus having more stability and strength, which is necessary to accomplish the payload requirements to carry the pair of manipulators. This platform was called Belerofonte. Figure 5.5 shows the actual platform. The electronic devices were placed in the center below the autopilot to balance the center of mass, and the arms have been placed below too. The frame is an hexacopter designed and built by DroneTools SL [2]. The rest of the hardware, as well as the arms and the software, has been designed and developed by the author.

The platform has a Pixhawk[3] autopilot which is responsible for the robot's flying control. Additionally, an Intel NUC computer, in which all the systems run, has been installed at the bottom. Finally, a camera holder is placed between the arms to place any camera to perform query images for the vision algorithms. Particularly, two cameras have been tested with this platform: an Intel Real-Sense camera and a ZED Stereo Camera.

[2] *http://www.dronetools.es*
[3] *https://pixhawk.org/*

Figure 5.5 Figure shows the arms built in the multirotor.

The arms were refurbished taking into account the problems that raised in previous versions. First of all, the traditional PWM servos were replaced by digital serial servos. These presents many advantages compare with previous ones. Particularly, they can measure the actual load that these are suffering, which is useful to be able to code a fail-safe stop if the servos are getting overloaded. Additionally, they are controlled over the same wire bus, reducing the number of cables that are needed in PWM servos.

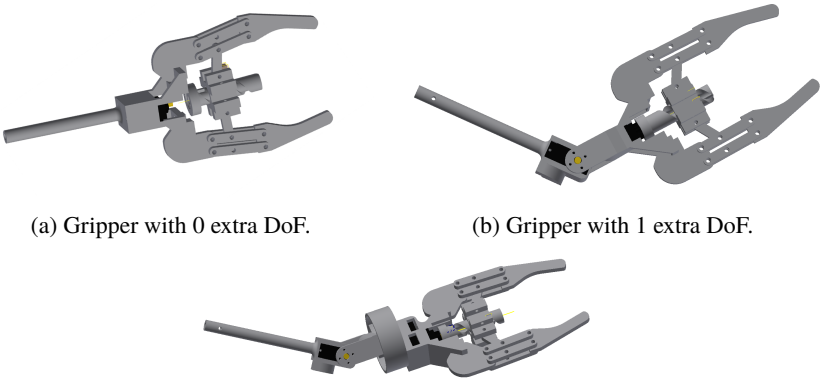
In this design, special attention was paid to make the system as reusable as possible. For this reason, the end effector was designed with a quick release mechanism (shown in Figure 5.6) with a screw to quickly change between different tools.

Figure 5.6 Quick release system for attaching different tools, i.e. the gripper or the docking tool..

Then three different grippers have been designed each of them giving extra degrees of freedom. Figure 5.7 shows the design of the different grippers. All of them use the worm drive to actuate the gripper.

These arms are part of an open-source project called *hecatonquiros* [4] which aims for general purpose, cheap and easy to use robotic arms. These are designed to be light-weight, thus relatively small UAVs can carry them. These are 3D printed reducing the overall cost of its production as the material is relatively cheap and does not need any post-processing step, so can be assembled. The cost of a single arm is $\sim$ 150$ (including the smart serial servos). The project also provides for a library based on OpenRAVE for the kinematic

[4] *https://github.com/bardo91/hecatonquiros/*



(a) Gripper with 0 extra DoF.

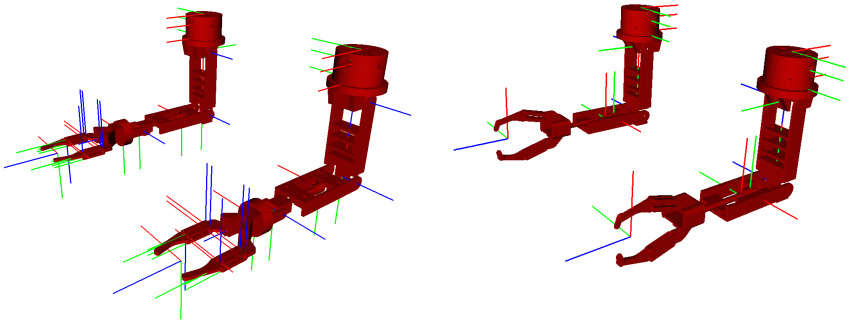
(b) Gripper with 1 extra DoF.

(c) Gripper with 2 extra DoF.

Figure 5.7 CAD design of different grippers designed for the aerial manipulation.

solvers and has support to ROS for performing simulations before the real experiments. More detailed information about the project is presented in Appendix A.3.

Figure 5.8a and Figure 5.8b show the coordinate frames defined for the robot, which will be used to transform from the detection of the camera to move the end-effectors of the arms. Figure 5.9 shows the kinematic reachability of the pairs of arms which will be used to estimate the appropriated position of the robot to perform the manipulation tasks.



(a) 6DoF arms model in OpenRAVE and coordinate frames.

(b) 4DoF arms model in OpenRAVE and coordinate frames.

Figure 5.8 Cad models of the 6Dof and 4DoF manipulators for the online simulations and planning.

Detailed information about the kinematics and inverse kinematics of these manipulators has been previously introduced in Section 2.4.

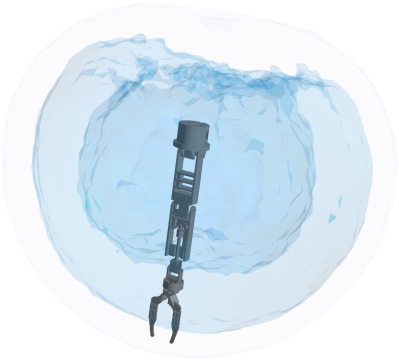


Figure 5.9 Kinematic reachability of the aerial Manipulator..

This platform has been used in many experiments for the AEROARMS project. Particularly, it has been used to validate the grasping algorithm shown in Sections 2.4 and 3.6.3, published in article [13].

5.4 Contact positioning tool for manipulation tasks

This section describes the hardware design of the novel tool, so-called docking tool, which is used to control the position of the aerial platform by contacting a rigid object. The tool has multiple free degrees of freedom which are sensorized. Other approaches [7] actively exert forces towards the rigid objects, overloading the engines and consuming more energy. Conversely, this tool is underactuated being more efficient and more stable.

At first, the development of tools for aerial robots is usually more constrained than in-ground systems due to the payload limitations and stability issues. In this work, the following assumptions are adopted:

- Perturbations produced by wind are relatively small.
- The UAV has a low-level controller which input is the desired speed in Cartesian coordinates, and it outputs motors speed.

In order to test this tool, the aerial platform presented in the previous section was used. The right arm is provided with a gripper to perform different manipulation tasks. The docking tool is attached to the left one. This tool provides the position of the UAV relative to its attachment base. These measurements are used for stabilizing it close to the manipulation space. Figure 5.10 shows the aerial robot with all the tools. The work presented in this section has been published in [15].

5.4.1 Docking tool model

The tool consists of a passive multi-link arm with sensors in the joints to measure the angles between the links. The main criteria during the design of the tool were to minimize

Figure 5.10 Dual arm aerial robot with gripper and docking tool..

the total weight and the friction in the joints, reducing the torques exerted on the arm and subsequently on the UAV. To reduce the weight, the structural parts are designed to thin and hollow. The components are 3D printed using ABS, making it lightweight and easier to replace. Furthermore, the production costs are lower than using aluminum or carbon fiber, and the components do not need to be built, machined or post-processed.

The tool is not actuated, i.e., it does not need any motor, making it lighter and getting rid of batteries. Figure 5.11 shows the CAD model of the tool. Bearings have been placed in the joints to minimize the friction. These are made of acetal plastic which is ten times lighter than common metal bearings.

It is composed of five joints. The base joint (or θ_0) provides a rotation on the Z axis. The following two joints (θ_1 and θ_2) compose a two-link arm that gives to the robot free 3D movement on the work zone. Joint θ_3 are set to provide an extra degree of freedom allowing the robot to remain parallel to the floor, independently of the position of the two-link section. Finally, the last joint (θ_4) adds to the robot another DoF, to make it able to maintain the heading effortlessly.

The kinematic model is shown in

$$T_{UAV} = f(\theta_0, \theta_1, \theta_2, \theta_3, \theta_4) = T_0 \cdot T_1 \cdot T_2 \cdot T_3 \cdot T_4 \tag{5.1}$$

being,

$$T_0 = \begin{bmatrix} c\theta_0 & -s\theta_0 & 0 & 0 \\ s\theta_0 & c\theta_0 & 0 & 0 \\ 0 & 0 & 1 & l_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

$$T_i = \begin{bmatrix} 1 & 0 & 0 & l_i \\ 0 & c\theta_i & -s\theta_i & 0 \\ 0 & s\theta_i & c\theta_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \forall i = 1,2,3 \tag{5.3}$$

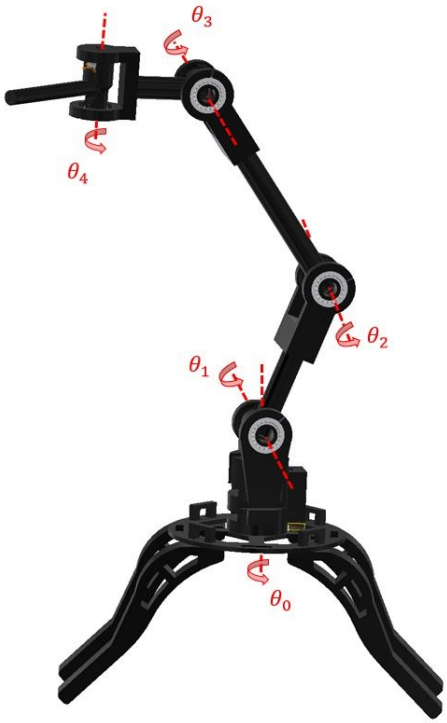


Figure 5.11 CAD model of docking tool with base for attaching to pipes with joints edges illustrated.

$$T_4 = \begin{bmatrix} c\theta_1 & 0 & s\theta_1 & 0 \\ 0 & 1 & 0 & l_4 \\ -s\theta_1 & 0 & c\theta_1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.4}$$

The docking tool system has 5 DoF for the drone's movement. An additional joint in the axis of the last bar provides free rotation related to the roll of the UAV. However, this rotation is significantly small due to the assumption of small perturbations. For this reason, the joint is not included in the design.

The joints of the tool are provided with potentiometers that are used for measuring the angles. The voltage signals from the potentiometers are measured by an electronic device connected to the onboard computer. Then, the signals are mapped to the angles, because the voltage in the resistances changes linearly, the mapping of variables is a linear map.

The sensors are wired using internal holes on the joints as shown in Figure 5.12. This minimizes the forces exerted by the cables on the joints.

The angles measured from the potentiometers are used to estimate the current pose of the UAV. Together with the information of the arms, these measurements are used to close

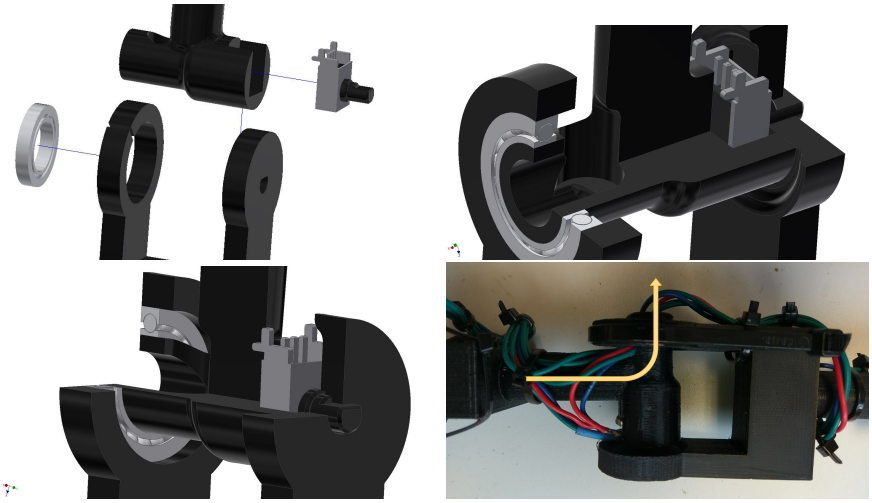


Figure 5.12 a) shows an figure with the component exploded. b) shows part of the joint sliced to see the internal holes and the components assembled. Finally, d) shows a close picture of the wiring system, passing through the holes that can be seen in previous figures..

the loop of the control system described in section 5.4.2. Figure 5.13 shows a 3D virtual visualization at different times of real experiments.

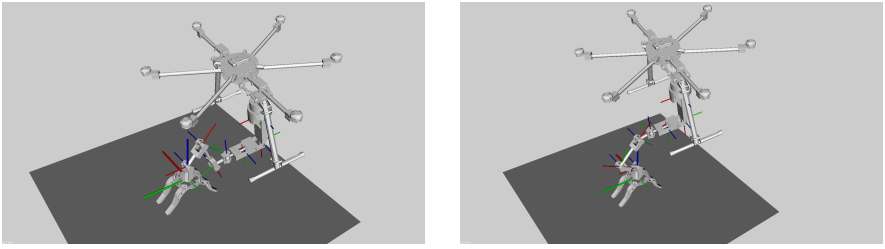


Figure 5.13 Online virtual visualization of the aerial platform with the docking tool during the experiments..

Furthermore, the base of the docking tool is provided with two additional components shown in Figure 5.14. The first is a 3-axis accelerometer, and the second one is a servo that locks the rotation of the base during the flight to improve the coupling phase.

The use of the accelerometer is essential to guarantee that the 6D pose of the end-effector can be used to control the UAV. As the arm that holds the docking tool remains parallel to the horizontal plane of the platform, it is important to know the orientation of the base of the docking tool to properly compute not only the position but the full pose. If for any reason during the attachment, the base is not entirely parallel it will lead to a rotation in the end-effector.

The second component, the servo, has been integrated to prevent the base from rotating while the platform is flying. As the joints are not actuated, the base tends to rotate, thus making difficult the docking stage. Then, the purpose of the servo is to lock until the platform has docked.

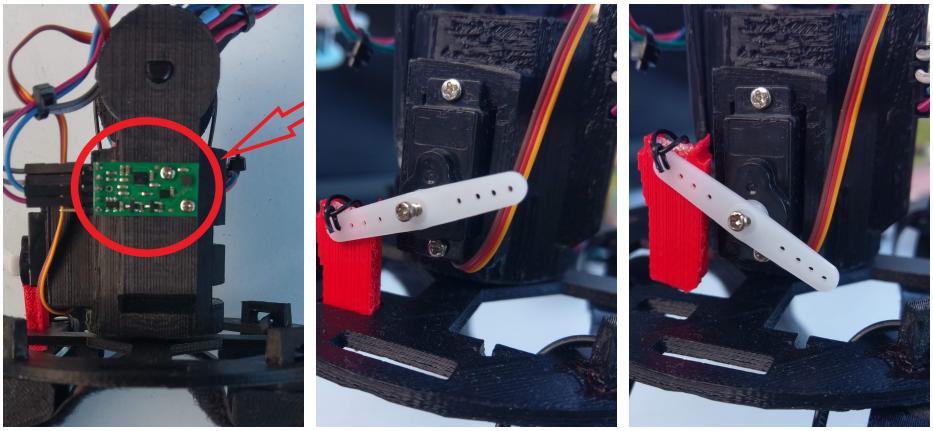


Figure 5.14 Left picture shows the accelerometer placed in the base joint to measure the orientation of the base while docked. The other two pictures shows the servo that locks the base of the docking tool until it is placed..

More detailed specifications of the docking tool components are summarized in Table 5.1.

Table 5.1 Specifications of the low-cost Docking tool.

Potentiometers resistance($K\Omega$)		20
Potentiometers angle range(deg)		270
Operating Voltage (V)		5
Power consumption (W)		0.25
Servos		SG90 Pro 9g
Accelerometer		L3GD20H and LSM303D Carrier
Longitudes (m)	l_0	0.071
	l_1	0.105
	l_2	0.155
	l_3	0.07
	l_4	0.075
Total weight (g)		150
Base material		Plastic ABS

Finally, Figure 5.15 shows the workspace of the end-position of the docking tool and the quality of each of the positions. The quality is evaluated using the distance of the joints to the saturation points. This distance is mapped to a value within 0 and 1, and all

the values are multiplied, obtaining a single bounded value. It means that if one of the joints get close to a saturation limit, the overall quality of the position decreases. The values are represented using a heat-map scale, being blue the best value and red the worst. These values are intended to be used in future work to perform a smarter control of the system trying to keep the position of the docking tool in a well-conditioned volume of the workspace.

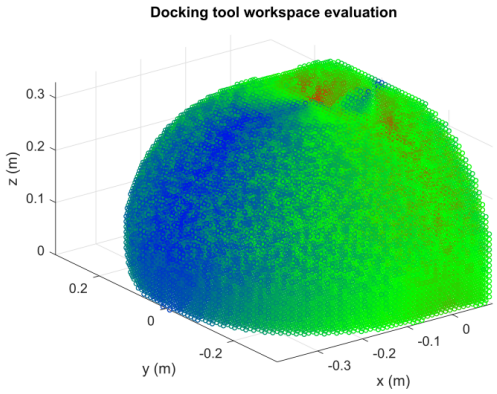


Figure 5.15 Reachability of the end-position of the docking tool with heatmap colors evaluating the closeness of the joints to the saturation limits.

5.4.2 Controlling the position with contact point

In this Section, a case of use of the docking tool proposed is shown. The position provided by the tool is used to estimate the relative pose of the end-effector of the second arm regards the attached position. Thanks to this information, it is possible to perform a Position Based Servoing (PBS) algorithm to keep the end-effector in a fixed point or location.

A cascade control system is proposed for positioning the aerial robot. Figure 5.16 shows the controller structure. The inner loop corresponds to the internal controller provided by the px4 software. It consists of Cartesian speed control that translates from the desired velocity to the corresponding actuation on the motors. The outer loop uses the position obtained by the docking tool to produce a target speed to control the robot.

The outer controller is a PID tuned to provide quick responses to the perturbations on the UAV and the drifts of the internal controller (generally due to errors in the internal estimators of the px4 software: GPS errors, IMU drifts, etc.). The position of the docking tool is compared against a target position to compute an error $e(t)[m]$. This error in position fed the controller which produces at the output a control signal $u(t)$ which is a reference speed in $[ms]$ that fed the autopilot.

While the tool is docked, it throws measures of the relative position from its base. The error that feeds the control loop is computed by the difference in position of the current position and a reference position. In order to smooth the state of the robot, the data obtained from the tool is filtered using an Extended Kalman Filter[201].

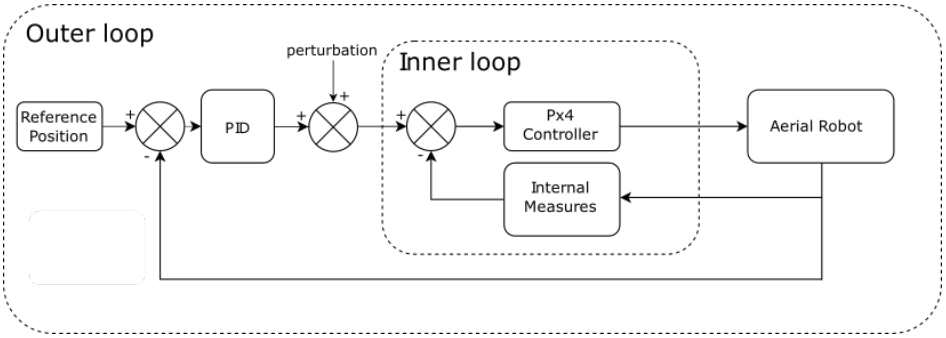


Figure 5.16 Scheme of the cascade control system..

The PID was coded with an anti-windup block to avoid large oscillations due to the integral factor. Additionally, the output speeds are saturated to prevent abrupt control signals due to the derivative terms. Table 5.2 contains the values of the PID parameters. These values were tuned from the experiments, starting from a controller flying free in the air and then tuning the parameters with the information from the docking tool.

Table 5.2 PID parameters.

	K_p	K_i	K_d	Anti-windup	Signal saturation (m/s)
X	0.8	0.01	0.7	10	2
Y	0.8	0.01	0.7	10	2
Z	0.3	0.03	0.7	10	1

5.4.3 Experimental Setup

In addition to the components shown in Section 5.4.1, the aerial robot needs other devices to perform the experiments. Figure 5.17 shows all system components. The potentiometers on the joints of the docking tool are connected to the analog inputs of the microcontroller. The Arduino is connected to an onboard Intel NUC computer which is used as the main computer. The lectures of the sensors are gathered on it, to produce the control signals that are sent to the autopilot (Pixhawk). The autopilot receives the target speed and controls the multi-rotor. Additionally, a power supply system is added to feed each of the devices at appropriate voltages.

A set of practical conclusions were obtained during the first stages of the development process:

- In order to increase the arms operation range and to prevent internal collisions a foldable landing gear was built-in.
- It was observed that signals of potentiometers saturate before reaching the mechanical extremes. This reduces the tool workspace. Particularly, the mechanical range is 170°, and the signal range is 150°. For this reason, it is good to keep the UAV in a position in which the joints are not in the limits.

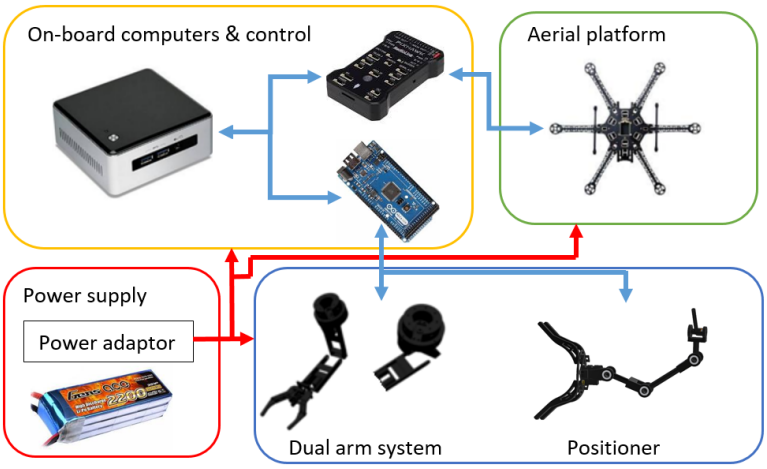


Figure 5.17 Components of the autonomous docking system..

- The docking tool joints are not actuated. During the experiments, the tool hangs until it is docked. Particularly, the base joint can rotate. For this reason, that joint is locked with a micro servo which unlocks the joint once the tool is placed.

5.4.4 Test-bench and tool characterization

A static experiment has been carried out in a test-bench. The purpose is to measure two variables: the accuracy and frequency of the measurements. The data provided by the docking tool is compared against a ground truth obtained by a *Leica Total Station MS50*[5] (or TS). It uses a laser that locates a *prism* within an error of $0.3 - 0.4$ millimeters.

The docking tool is placed on a table and the *prism* is attached at the end of the positioning system as shown in Figure 5.18.

Figure 5.19 shows the results in the test-bench. The end position of the tool is moved to describe a cross in 3D, trying to perform the movements over each axis independently. Figure 5.19a shows the end position of the tool, the solid line is the position measured by the docking tool, and the dashed line is the position measured by the TS taken as ground truth; Figure 5.19b shows the difference between both measures. The mismatch in the Z axis, within time counts 2375 and 2675, corresponds to the fact that the values of the potentiometers saturate when going down. The joints exceed the allowed range which leads to bad angle measurements. Similar effects can be seen at the limits of the movements in the X and Y axis, but in these cases, the effect is slightly noticeable. This happened intentionally in this experiment to show the saturation effect. Nevertheless, because the workspace and the closeness of the joints to the saturation limits are known at any time, this effect can be mitigated while flying.

[5] *https://leica-geosystems.com/products/total-stations*

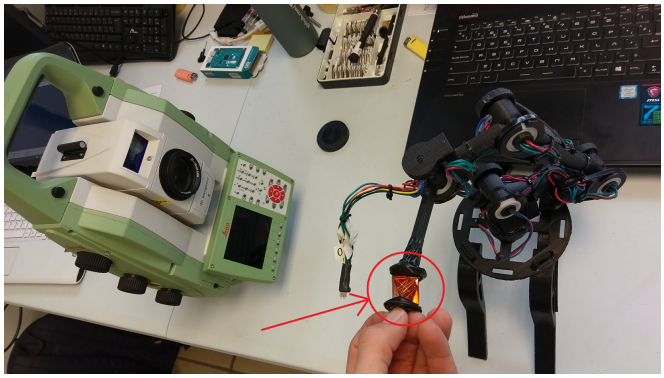
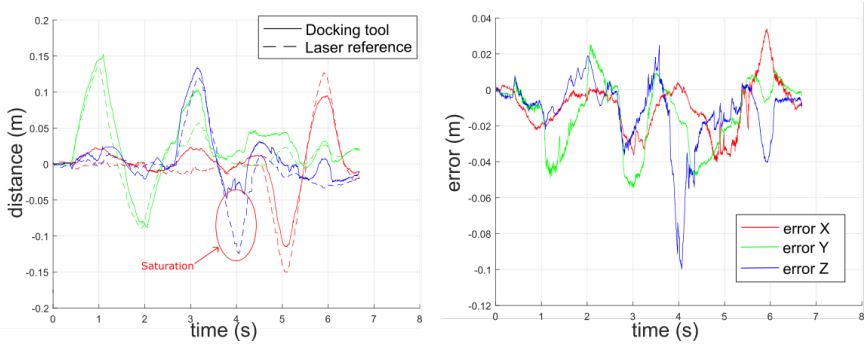


Figure 5.18 Test-bench with laser system for measuring the accuracy of the docking tool..



(a) Position measurements vs ground truth.

(b) relative error.

Figure 5.19 5.19a shows the X, Y, Z components of the trajectory of the end-effector during the experiment in the test-bench measured by the docking tool and the laser system (X: red; Y: green; Z: blue). 5.19b shows the relative error between the measurements. The relative error has been zoomed for the clearness of the figure. .

5.4.5 GPS positioning characterization

An outdoor first experiment was executed using the GPS to obtain the position reference for the control loop. This experiment is shown to characterize the magnitude of typical errors using this common positioning device and to compare with our positioning system. The real position of the robot is measured with the Total Station. Figure 5.20 shows the error in the position of the UAV according to a fixed set point measured with the Total Station. The experiments were performed in a clear day with very low wind conditions with the hardware specified in Section 5.4.3

In this experiment, it can be observed that the errors can be large in some situations. Moreover, these experiments were performed outdoors on a clear day. Thus, there could be worst conditions were the GPS could be denied or noisy, that would induce larger

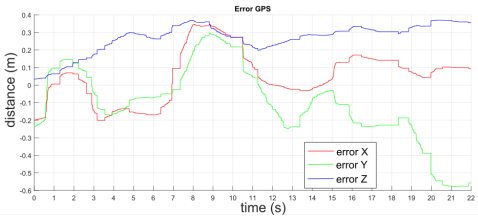


Figure 5.20 Position of the UAV measured by the total station during a hovering test using GPS as position estimator. .

errors and put the platform and the environment in danger. This inaccuracy exceeds the workspace of the arms, making difficult any kind of inspection or manipulation task in the target zone.

5.4.6 Docking and autonomous control

During the experiment [6], the UAV takes off, moves to the target position in order to attach the docking tool. Then, it starts measuring the relative position of the robot and performs the autonomous control. The position of the UAV is also acquired using the TS as ground truth. Figure 5.21 shows snapshots of different experiments of the robot docked to a pipe.

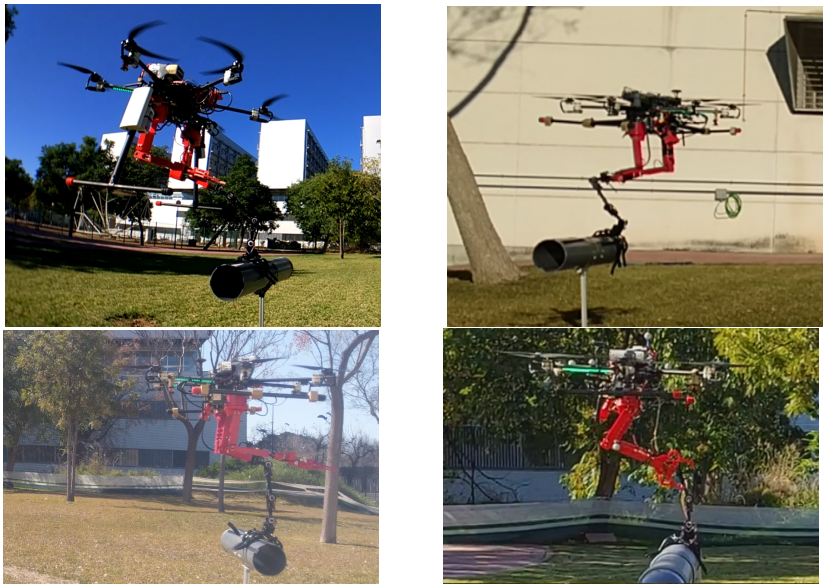
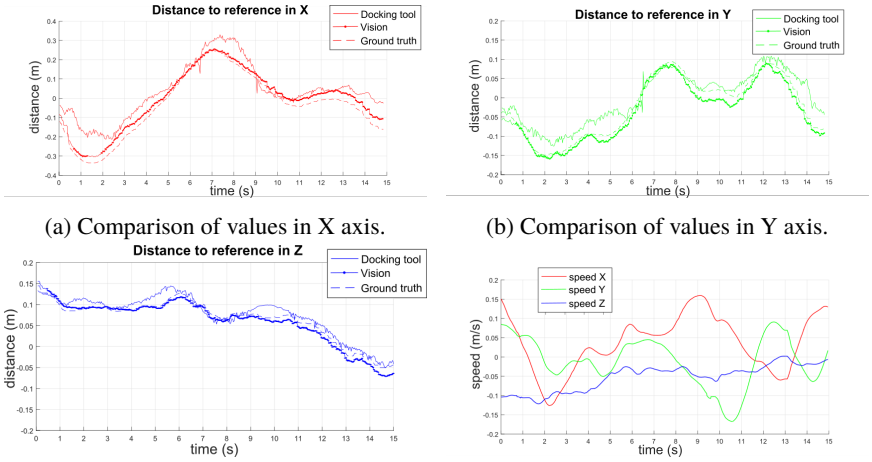


Figure 5.21 Snapshots of robot docked to a pipe during different experiments. The joints of the docking tool, passively, adapt to the UAV position, which can vary due to external perturbations..

[6] *https://youtu.be/Vk9G7lb_r6I*

Additionally, a camera has been attached to the docking tool to compare the results with a monocular vision system (ORB_SLAM2[174]). In Figure 5.22 the data recorded during an experiment using the docking system can be observed. Figures 5.22a, 5.22b and 5.22c show the difference between the current position and the reference position measured by the different localization systems. The solid line represents the difference in position by the docking tool. The dashed line the one measured by the total station and the solid line with dots using the vision algorithm.



(a) Comparison of values in X axis.

(b) Comparison of values in Y axis.

(c) Comparison of values in Z axis.

(d) Control speed generated by the controller.

Figure 5.22 5.22a, 5.22b and 5.22c compare the errors between current UAV position and a reference position measured from the docking tool (solid line) and the TS (dashed line) and the vision system (solid line with dots) in the three axis. 5.22d shows the speed control generated for the outer loop of the cascade controller.

Figure 5.22d, shows the control signals produced by the PID generated by the outer loop of the cascade controller from the error in position obtained by the docking tool.

Table 5.3 compares the errors and deviations of the different localization systems studied against the docking tool during the experiments at each axis. It is evident that just relying on GPS is unsafe for the platform. It can be observed, that both the visual algorithm and the docking tool provides similar measures. However, the visual localization algorithm consumes a lot of computer resources, and it gives the location of the robot up to 25 Hz. Conversely, the measurements obtained from the docking system achieve a frequency of 1200 Hz which overtakes the vision speed. Moreover, the computer vision approach depends strongly on the light conditions.

5.4.7 Manipulation while keeping in contact

A last experimental set up has been proposed to enhance the usefulness of the docking tool. As aforementioned, the aerial platform has two built-in manipulators. The purpose of

Table 5.3 Errors and standard deviations during the test experiments..

	GPS		Vision		Docking Tool	
	μ	σ	μ	σ	μ	σ
x (m)	0.164	0.079	0.022	0.027	0.034	0.032
y (m)	0.153	0.123	0.012	0.032	0.036	0.028
z (m)	0.179	0.085	0.023	0.026	0.040	0.039
avg. speed (Hz)	10		25		1200	

the docking tool is to provide to the aerial platform a robust position estimation from the attaching point to then be able to perform any task with the second arm. In the experiment shown in this section, the second manipulator is commanded to place its end-effector to a predefined location. The target is to perform a position based servoing (PBS) to minimize the error between the real position of the end-effector and the target position. To reduce the position error, the equations shown in Section 2.4 are used.

In order to measure the real position of the end-effector a reflector prism is placed to be detected by the total station. Using the docking tool and the control loop proposed in Section 5.4.2 the arm tries to fix the position of the end-effector in the target pose as shown in Figure 5.23.

Figure 5.23 Outdoor experiments of the position based servoing of the manipulator using the position given by the docking tool.

The expected relative position from the attachment point computed by the docking tool has been compared against the reference position obtained by the total station. Figure 5.24 the positions obtained.

Additionally, the results of the joints state of the manipulator are shown in Figure 5.25, resulting from the PBS algorithm to move the arm towards the target point. Dashed lines represent target joints of the manipulator, and the solid lines are the actual joints at each instant.

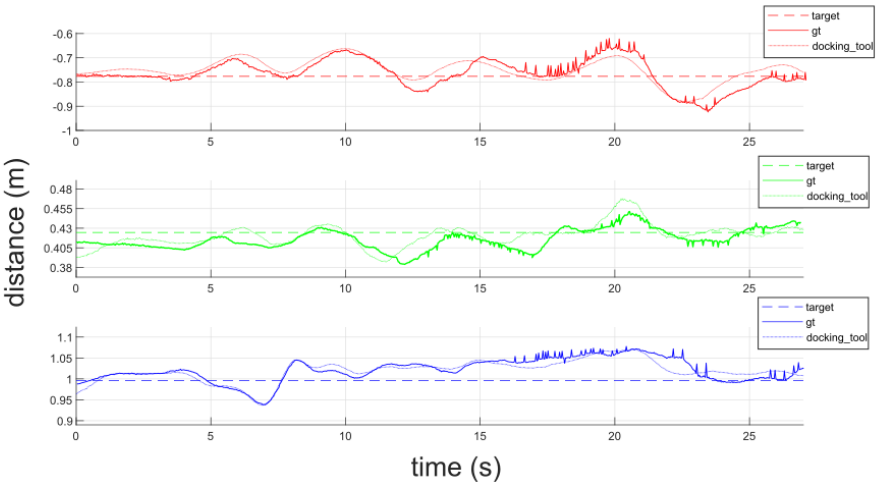


Figure 5.24 Position of the end-effector measured by the docking tool and the kinematic model of the system, and measured by the ground truth.

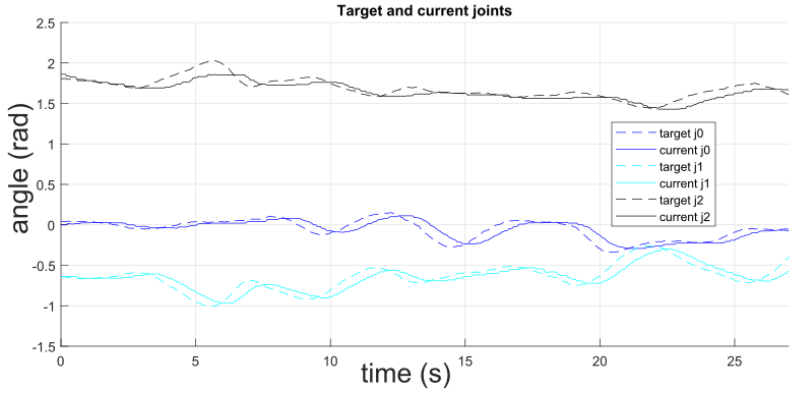


Figure 5.25 Target joints angles and current joints angles during the experiment.

6 Conclusions and Future Work

This chapter presents the main contributions of the thesis and analyses and highlights the results. Moreover, it discusses the advantages and disadvantages of the proposed methods and algorithms. Finally, future developments are introduced to overcome the drawbacks.

Unmanned Aerial Vehicles have a clear and key future in many applications. However, there are many challenges that need to be overtaken to fully integrate them among us. Especially, if these are expected to cooperate with humans. One of the central pillar for full automation is the development of an adequate vision system. Any limitation in the perception capabilities of the robots will obstruct in the rest of functionalities of the system. For this reason, it is the belief of the author that robots require substantial improvements in their perception capabilities to allow them to interact with the environment to perform any task.

The presented aerial system intended to allow these aerial robots to perform autonomous operations in industrial environments for inspection and maintenance tasks. The contributions of this thesis center in the perception capabilities for interacting with specific objects, either detecting them or analyzing the best way of manipulating them. Nevertheless, there is a long way to make them future-proof and capable of achieving the same capabilities than human operators in these environments.

6.1 Contributions and conclusions

This thesis presented a set of useful algorithms for the automation of aerial robots in manipulations tasks from the point of view of the perception and the manipulation analysis of physical objects. The thesis has been arranged coherently gathering all the knowledge acquired during the research of the author over these years. This knowledge has been published in many articles, and each of them contains the information in specific applications which are then split to accommodate the presented structure.

Chapter 2 contributes with algorithms to describe objects and analyze the manipulation of them. At first, several object models have been presented. The right selection of these models is essential to determine the later algorithms for the manipulation analysis. Special attention has been paid to the probabilistic model of Gaussian Processes Implicit

surfaces about which the author has been working intensively during his research stay in the Australian Center for Field Robotics and with which he has published various articles. Later, in the same Chapter, most of the common manipulation analysis has been presented and which are also implemented in the framework that is shown in Appendix A.2. At last but not least, practical applications of different manipulation methodologies are presented for manipulation tasks with real aerial platforms. Related to this work, the following conclusions can be drawn:

- Point cloud-based and mesh-based object models provide for reasonably good results for object manipulation. However, these models are quite simple and do not handle uncertainty.
- GPIS can hold a resolution-less representation of objects which has information about the confidence of where the surface lies. Thus, the manipulation tasks can be evaluated from the probabilistic point of view, making the robot system to chose those grasps that maximizes the probability of succeed.
- Contrary to fixed manipulators, in order to perform manipulations tasks with aerial robots, it is critical to take into account dynamic targets. This chapter introduced how to use a DLS method to dynamically update the position of the arms to grasp objects that move relative to the frame of the robot.

Chapter 3 summarizes a variety of perception algorithms that allow the aerial robots to detect the objects a that are required to be manipulated somehow. The presented algorithms cover from the classical deterministic models using point clouds and meshes to state-of-art deep learning algorithms, covering some alternatives using the GPIS probabilistic model. The purpose of these sections is to allow the robot to perceive and track their-selves objects of any nature to automate the inspection and maintenance tasks. From this presented work, the following conclusions can be drawn:

- Most of vision based algorithms for object detection and pose estimation requires for a learning stage and supervised data. In many applications, it can be assumed that the object to be manipulated is already known. However, for future exploration applications, it might be useful to explore detection algorithms such as the one introduced in Section 3.2 which are able to detect object instances without a previous definition of them.
- Feature-based object detection is quite extended and has proven to be robust and accurate. However, the model overfits to a specific object, making this algorithm poorly scalable.
- Finally, among the different supervised methods, DL neuronal networks have overthrown state of the art algorithms. However, despite the advances in electronics, neuronal networks have strong limitations, and most of the state-of-art networks cannot run in small embedded computers on the UAV.

Chapter 4, explore two methodologies to give the robot the capability of locating itself in the environment and to create a descriptive map of the surrounding, so-called SLAM. At first, a localization system based on a pair of cheap web-cams is presented. Visual

features are exploited in the images captured by both cameras to create a sparse feature cloud. The sequential computation of these feature clouds is used to estimate the position of the robot at each instant. This visual localization algorithm has been improved by fusing the information of the internal inertial measurement unit using an Extended Kalman filter. It has been shown that the fusion of both information is highly beneficial, as the single use of the IMU produces large drifts over time and the sequential alignment of the feature clouds might induce abrupt movements in case of failures. Later in the same chapter, a framework based on RGB-D cameras is proposed to perform the simultaneous localization and mapping. This framework has been designed in well-differentiated modules which are described in Appendix A.1. The purpose is to create a flexible framework for fast development of many applications concerning the available resources and sensors in the aerial platform. The framework has been evaluated using the TUM-RGBD datasets. Additionally, a custom dataset of a bridge is presented to show the results with real outdoor data. The last section in this chapter has introduced a method for integrating the perception results from Chapter 3 into the maps computed in the proposed framework. This semantic information can be beneficial for the automation missions with aerial robots. To conclude with, the following conclusions can be drawn:

- It is undeniable that sensor fusion in SLAM algorithm is vital. Exploiting UAV's internal inertial sensors improves the transients and smooth the results of the visual systems.
- Modularly is a key aspect in fast prototyping. While developing the full SLAM framework, it has been found to be more efficient to split the implemented functionality properly. This fact makes every single component reusable and the whole framework easy to use.

Eventually, in Chapter 5 all the aerial platforms, manipulators and end-effectors developed during this research have been presented. Since the beginning, all the algorithms and methodologies developed have been applied and tested in real aerial platforms. This fact has been essential because, as mentioned several times, researching for aerial applications has very strict limitations, and it is critical to grant that the algorithms accomplish, not only the intended results, such as object detection and manipulation but also the required specifications of speed and accuracy. At first, a small hexacopter with a single arm is shown. This platform was used in article [11] for the detection of multiple objects using a low-cost stereo system, and in article [8] for object detection and manipulation. Later, a larger hexacopter is presented which has two manipulators. Finally, a special end-effector tool, so-called docking tool is presented which is intended to be used in aerial manipulation tasks which requires the aerial platform to have an estimation of the position relative to an area of interest. In particular, experiments are presented in which the tool docks to a pipe and the second manipulator performs position based servoing to control its position. Following conclusions can be drawn from the development of this chapter:

- Design of aerial manipulators required for a strong trade-off between the size and capabilities. Enlarging the size of manipulators affect to the payload and compromise the stability. But also, it affect to the computation capabilities. Strong computers are heavier and consumes more power which translates to larger batteries.

- Concerning the docking tool, it has been shown that in some situations, simpler physical devices can provide practical solutions. In this, case the positioning system is provided by a tool as end-effector, leaving the rest of resources available for other tasks such as vision or manipulation.

6.2 Future work

It is our belief that this work has established a baseline for future developments in the field of aerial manipulation. Additionally, three open-source frameworks have been implemented. These are: a grasping tool framework which structure is shown in Appendix A.2; an open-hardware and open-source repository for cheap 3D printed manipulators for aerial platforms which are shown in Appendix A.3; and a general vision library for monocular and depth cameras which is shown in Appendix A.1.

As the nature of the dissertation is varied, many research lines arose as future work. Nevertheless, future developments will strongly consider a deeper usage of deep neuronal networks for each of the tasks. Currently, all the methods developed in the analysis of manipulation tasks are based on traditional geometrical analysis of the topology of the objects. However, cutting-edge methods exploit the usage of deep learning to automate the generation of grasp candidates more intuitively. These methods require the use of labeled datasets to generalize the way objects can be manipulated or grasped. Other methods instead, leave the algorithms to learn themselves by the use of reinforced learning algorithms. These are based on the definitions of policies and rewards functions to define how goods are grasps to, similarly to humans, let the robot train itself and adjust its internal parameter to perform the tasks.

Concerning the object detection and tracking algorithms, recent state of the art algorithms also exploit the usage of deep learning for improving the results. Particularly, we found interesting the integration of both pipeline of images, the color, and the depth, into a single neuronal network to exploit all the information in one row. This fact might be useful as current implementations have two split algorithms, one for the object detection and another one for the estimation of its pose. Merging both will improve the efficiency of the algorithm in term of speed but also, it is expected that the neuronal network exploits the visual and the spatial information all in the same algorithm.

A SLAM framework has been developed to give a highly flexible set of tool for testing and integration localization and mapping capabilities to any platform in any new application. However, there are many new development lines that can be:

- Parallelization of optimization pipelines. Optimization has been coded separately to allow a parallel implementation of the SLAM algorithms. However, the current application has been implemented sequentially. It will be of high interest to implement an application with these modules running in parallel to not to hinder the visual odometry pipeline.

- Threaded loop closure detection. The loop detection module is implemented separately. However, the main application for SLAM has this code implemented sequentially too. A thread-safe implementation of loop closure detection will also improve the performance in real applications.
- Integrating new state estimators such as Unscented Kalman Filter or Particle Filter
- Recently, many loop closure detectors based on Deep Learning algorithm have appeared. It might be interesting to research on them and implement different backends for loop closure detection.

Eventually, regarding the augmented information for the SLAM algorithms, i.e., semantic SLAM. The method exposed showed the results for localizing instances of objects. However, this semantic information can be further extended. Possibly, the most interesting line of research concerning this issue is the use of pixel-wise semantic deep learning algorithms to completely categorize the point clouds used in the SLAM for the reconstruction of the environment. This information could be useful to enhance the understanding of the environment to perform the manipulation tasks.

Appendix A
Open resources developed

A.1 RGBD TOOLS

This section presents RGBD_TOOLS[1] as an added value contribution to this thesis. RGBD_TOOLS is an open-source library that targets to allow faster development of real-world autonomous aerial robot applications. The library and the API is coded in C++ has has compatibility with other open-source libraries such as OpenCV, PCL, and ROS.

One of the challenges in developing real-world applications is the need for integrating all the different modules. Additionally, if the application is moving from a testbench or proof of concept to a real application, there is usually a need of refactoring the code to adapt to the new interfaces. In RGBD_TOOLS, all the classes and functions have been designed for the simplicity of the user. Additionally, an special attention has been paid to those functionalities that can be, theoretically, exchangeable, and these have been implemented with such flexibility. In that manner, for example, all the camera interfaces are exchangeable, saving time while switching between simulations or datasets and the real application.

The library is in a continuous development process. Up to now, it is composed by five core modules which are shown in Figure A.1 which will be described in the rest of this section. The list of modules are:

- Camera Wrapper
- SLAM Module
- State Filtering Module
- Machine Learning Module
- Other Tools

Further information and resources (such as tutorials and examples) can be found in the github webpage of the library.

[1] *https://github.com/Bardo91/rgbd_tools*

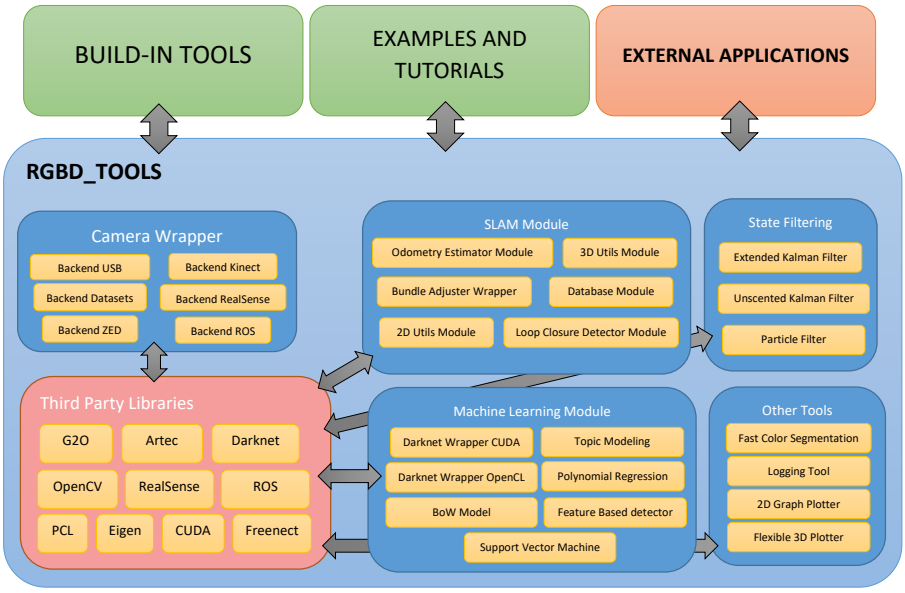


Figure A.1 Architecture of RGBD_TOOLS library.

A.1.1 Camera Wrapper Module

Handling with different cameras or switching between real devices, simulations or datasets is always a source of refactoring in real applications. In order to make the system flexible to any change in this device the Camera Wrapper module provides an standardized API that allows to switch between multiple camera back-ends without needing to change any piece of code. This is possible thanks to the use of factories and the generalization of the interface using standard JSON files. Figure A.2 shows the hierarchical structure of the cameras.

The cameras has been called StereoCameras since the library focuses in devices that provides not only color images but depth images too. Nevertheless, the system handle monocular cameras too.

The module is composed by the following backends:

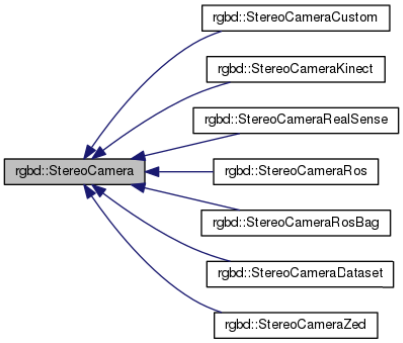


Figure A.2 Hierarchical architecture of cameras to allow flexible changeability.

- Backend USB: This backend provides for a general interface with cameras that are compatible with Universal Video Class (UVC) drivers. This backend targets for custom build monocular or stereo systems and provide basic built-in implementations of algorithms to compute the disparity between the cameras and to construct the point clouds for any application. This fact has a goal to allow fast tests and development of applications which point of interest is not the set up but another results which uses this information such as SLAM, object detection, collision avoidance, etc.
- Backend Kinect: This backend provides a fast integration of the freenect open-source library into the framework. The third-party library provides for the color image and depth image which directly fed the interface. Then using the internal calibration of the camera, the backend build the point cloud for the user making it simple to use.
- Backend RealSense: This backend provides a fast integration of the librealsense library into the framework. This backend has compatibility with both version 1.X and 2.X. As Intel Real-Sense devices are varied, the implementation intend to cover all the configurations. The third-party library provides for the color image, depth image and point cloud which directly fed the interface.
- Backend ZED: This backend provides a fast integration of ZED library by StereoLab into the framework. This backend requires for the additional dependency of CUDA to allow the fast computation of the pointclouds, if not, just the pair of stereo images are provided. The third-party library provides for the color image, depth image and point cloud which directly fed the interface.
- ROS Interface: This implementation has two different backends. ROS [202] is one of the most wide used open-source libraries for robotics applications that exists in the world. The library is built with modules and has many resources for a large variety of applications. Particularly, one of the most important modules is a communication system based on topics which is used to communicate different programs in the same or different computers. A backend which reads the data from this topic is provided to integrate the library into this framework. Additionally, ROS library provides for bunches of data called ros-bags. The second backend provides an interface to this files, reading all the information from them to fed the applications.
- Backend Datasets: Common datasets on internet consists on bunches of color and depth images arranged. This backend provides for an interface to this standard datasets to allow the users to tests their applications without needing to run the real experiments online. This backend is particularly useful for proving concepts, testing new algorithms, or during the development process.

A.1.2 SLAM Module

The purpose of this section is to introduced the framework given by the library to allow fast development of applications using it and the possibilities that it offers. A deeper analysis of the state of art and algorithms is provided in Chapter 4.

The tools has been coded taking into account the modular and agile development philosophy of the library. The Module is composed by six packages: Odometry estimator, Database, Loop-Closure detection, Bundle Adjustment, 2D utils and 3D utils.

The odometry estimator consists in a common interface for different odometry estimators. The purpose is to abstract the common behavior of odometry algorithms to make them easily exchangeable to allow a fast integration and also testing of different algorithms without changing the overall behavior of applications. This fact is extremely useful for researching as in many cases, the topic of research focuses in this specific topic and it is hard to test the final improvement in a SLAM algorithms which requires more complex implementations of the rest of the functionalities. Current implementation is based in computing visual odometry by using RGB-D sensors such as ZED, RealSense devices or Kinect.

The Database module, provides for an implementation of the storage requirements needed for a Graph-based visual SLAM algorithm. This module is responsible of the storage of the data and for the creation of the visual landmarks that are used for the localization algorithms, Loop-Closure detectors and optimization algorithms. Future developments in this module will pay attention to the efficient storage of the data to improve the dynamic memory allocation making applications faster and able to run large-scale algorithms.

The Loop-closure module is an interface to simplify the usage of algorithms that are able to detect when the camera has visited again the same location, which means that a loop is detected. Similar to previous implementations, this module can be freely applied if convenient and has a flexible structure. Particularly, two different implementations are currently provided. The first one is based in the Smith-Waterman algorithm and the second is based in the usage of Binary-BoW model for faster implementations.

An indispensable functionality for any SLAM algorithm are the optimizations. Bundle Adjuster module intends to provide an easy interface for one of the most common used optimization algorithms in these applications. Nowadays, several libraries exists that gives this functionality. However, as in many other functionalities, each of these libraries has different interfaces and requirements. This module integrates three different implementations of Bundle Adjustment optimization algorithm, each of them has different targets, but all provide good results for this purpose. Particularly, these backends are g2o, cvSBA and ROS_SBA. The generalization of the interface, allow the integration of the optimization algorithm with almost any knowledge about the algorithms, making it extremely easy to apply and making faster the development of SLAM implementations.

Eventually, 2D and 3D utils module contains various implementation of utilities that are used by the other modules, but that can be used separately for similar purposes. Using them, testing new algorithms or methodologies can be easier, unifying the interfaces.

A.1.3 State Filtering Module

The goal of this module is to provide easy to use implementations of state estimators for robotics applications. All the implementations are based in bayesian algorithms which are widely used in this field. Particularly, three implementations are provided: The well-known Extended Kalman Filter (EKF), an implementation the Unscented Kalman Filter and a general interface for deploying Particle Filters.

A.1.4 Machine Learning Module

The Machine Learning module ships with various implementations of learning algorithms fully integrated with the framework and with C++ API. The huge popularity of Deep learning in robotics have produced the apparition of a lot of frameworks to deploy their usage. In many of these framework, Python has been chosen as programming language due to its simplicity and portability. However, many users have experienced troubles while integrating them into custom applications due to the large amount of dependencies and the variety of implementations. For this reason, the Machine Learning module has the purpose of providing interfaces that are easy to deploy and integrate in real-world applications. This module has also implemented some popular methods such as BoW with SVM, probabilistic LDA and object detection neuronal networks.

A.1.5 Other Tools Module

This last module integrates several tools that are not intrinsically related with the aforementioned modules. The most important implementations are:

- Fast Color Segmentation: This tool, provides for an easy to use implementation with speed improvements of the algorithm proposed in [203]. The algorithm segmentate the color space by using binary operations which are quite faster than common color clustering algorithms. This implementation has been used in [204] in the MBZIRC robot competition.
- Logging Tool: It provides for a basic logging functionality with colored interface for simple user interfaces and log storage for external applications.
- 2D Graph Plotter: It provides for a simple 2D plotter based in OpenCV to show graphs or values over time. This tool has been found to be useful for debugging purposes, due to its simplicity and light dependencies.
- 3D Gui: It provides for an easy to use 3D plotter based in PCL. It is thread-safe and can be called across the application easily without needing to keep track of its instantiation. Fact that make it versatile.

A.2 GRASPING TOOLS

This appendix presents GRASPING_TOOLS[2] architecture used to supplement the contributions of this thesis. GRASPING_TOOLS is an open-source library for autonomous grasps generation in robotics. The library and the API is coded in C++ and implements the different object modeling shown in Chapter2 and the how to grasp objects according to those models. Figure A.3 shows the content of the library.

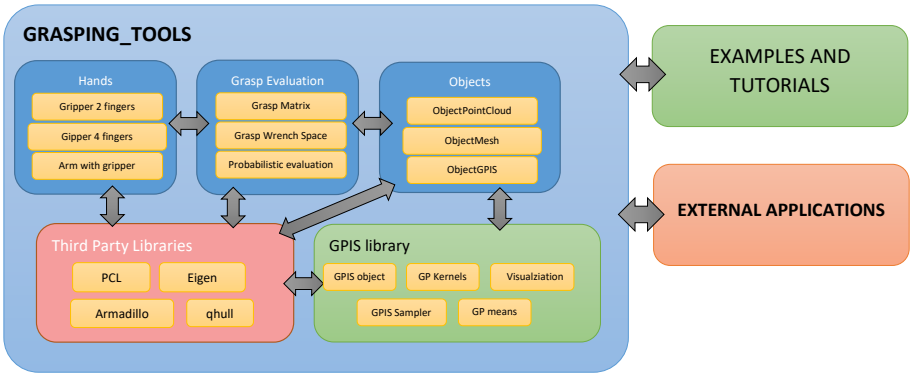


Figure A.3 Architecture of GRASPING_TOOLS library.

The library is composed by 3 core modules which purpose is to interface 3 different aspects of the manipulation: the objects, the manipulators and the evaluation of the grasps.

- Objects module: The purpose of this model is to provide an intuitive interface to model objects for manipulation tasks. As introduced in Chapter 2 objects typologies are: point cloud based, mesh based, and probabilistic models using GPIS. These models are used to simulate object's surface to generate grasps for planning manipulation tasks.
- Manipulators module: this module defines the interface for manipulators. Manipulators are entities that can generate grasps given any object of the previous module. Three basic manipulators are defined in this module, but more can be implemented and easily integrated with few work.
- Grasp Evaluation module: this module is encapsulate the results of combining the objects and manipulators. The pillar of this module is the grasp. Then different quality metrics can be computed to it to evaluate the feasibility of the grasps and be able to plan for the best option for each manipulation task.

[2] *https://github.com/Bardo91/grasping_tools*

A.3 HECATONQUIROS

This appendix exposes the open-hardware and open-software resource developed during this research focuses in the development of general purpose arms and tools for aerial manipulators. hecatonquiros [3] library contains different models of arms and end-effectors easy to build using additive manufacturing techniques (i.e. 3d printing). These models have been developed for the aerial manipulation tasks described in this dissertation. Figure A.4 shows the combined hardware-software architecture of the library.

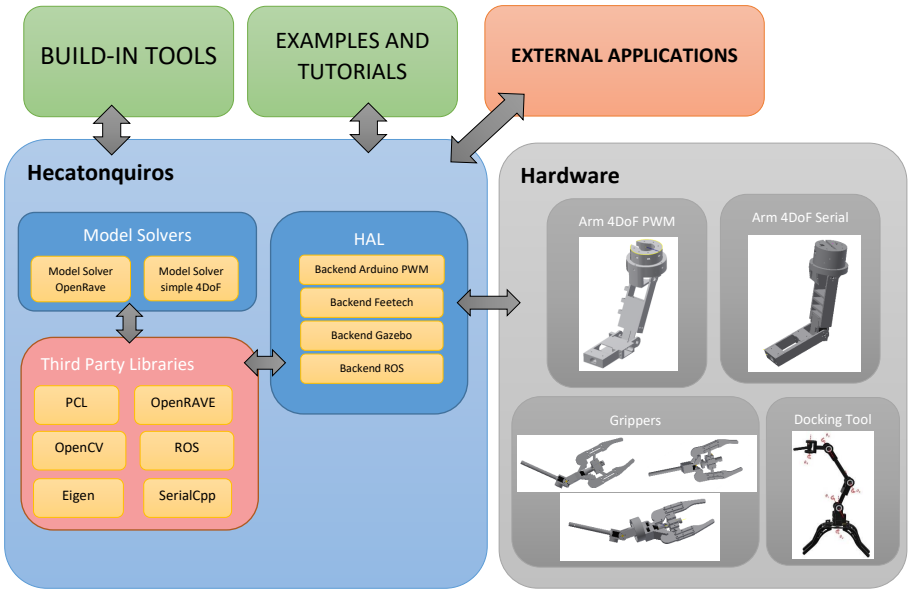


Figure A.4 Architecture of Hecatonquiros library.

Concerning the hardware design, two different arms have been designed. The main difference between current arms designs is the electronics. The first version was developed using PWM servos controlled by the onboard computer through and arduino. These servos gave fair enough results, but they have the main disadvantage of not providing any feedback of the real position of the joints. This fact can lead to inaccuracies in the estimated position of the end-effector. For this reason, the second design uses serial servos which have bidirectional communication between the computer and each of the engines. Being able to query the actual angle of the joints makes possible to know the real position of the end-effector, even if the target position is not reachable.

[3] *https://github.com/ViGUS/hecatonquiros*

Additionally, a set of end-effectors are were designed. Firstly, three different gripper with different degrees of freedom to complement the ones of the arms. Secondly, the design of the docking tool shown in Chapter 5 is included too.

Eventually, the library provides for a C++ interface for controlling the arms. These interface is standardized, so it can be use for any of the manipulators without changing the code. This fact simplifies the development of new applications and transitioning from simulation to real arms. The Hardware Abstraction Layer (HAL) is the module responsible of this task. It implements backed for both of the aforementioned arms but also for manipulators simulated in Gazebo or any other manipulator with ROS interface.

The library includes also two ways of computing the kinematics of the manipulators. The first option is a simple model solver for the configurations with 4DoF. This implementation has a fast algorithm based in the specific distribution of the joints of the arms. A second option uses OpenRAVE as backend for solving the kinematics of the arm. This second option can be used for any of the configurations.

List of Figures

List of Tables

Bibliography

[1] N. Pouliot, P. Richard, and S. Montambault. Linescout technology opens the way to robotic inspection and maintenance of high-voltage power lines. *IEEE Power and Energy Technology Systems Journal*, 2(1):1–11, March 2015.

[2] Leena Matikainen, Matti Lehtomäki, Eero Ahokas, Juha Hyyppä, Mika Karjalainen, Anttoni Jaakkola, Antero Kukko, and Tero Heinonen. Remote sensing methods for power line corridor surveys. *ISPRS Journal of Photogrammetry and Remote Sensing*, 119:10 – 31, 2016.

[3] L. Wang and Z. Zhang. Automatic detection of wind turbine blade surface cracks based on uav-taken images. *IEEE Transactions on Industrial Electronics*, 64(9):7293–7303, Sept 2017.

[4] Randa Almadhoun, Tarek Taha, Lakmal Seneviratne, Jorge Dias, and Guowei Cai. A survey on inspecting structures using robotic systems. *International Journal of Advanced Robotic Systems*, 13(6):1729881416663664, 2016.

[5] JF Reinoso, JE Gonçalves, C Pereira, and T Bleninger. Cartography for civil engineering projects: Photogrammetry supported by unmanned aerial vehicles. *Iranian Journal of Science and Technology, Transactions of Civil Engineering*, 42(1):91–96, 2018.

[6] F. Ruggiero, V. Lippiello, and A. Ollero. Aerial manipulation: A literature review. *IEEE Robotics and Automation Letters*, 3(3):1957–1964, July 2018.

[7] A. Ollero Baturone, G. Heredia, A. Franchi, G. Antonelli, K. Kondak, A. Sanfeliu Cortes, A. Viguria, J. R. Martinez de Dios, F. Pierri, J. Cortes, A. Santamaria-Navarro, M. A. Trujillo Soto, R. Balachandran, J. Andrade-Cetto, and A. Rodriguez Castano. The aeroarms project: Aerial robots with advanced manipulation capabilities for inspection and maintenance. *IEEE Robotics Automation Magazine*, pages 1–1, 2018.

[8] Pablo Ramon Soria, Begoña Arrue, and Anibal Ollero. Detection, location and grasping objects using a stereo sensor on uav in outdoor environments. *Sensors*, 17(12):103, Jan 2017.

[9] W. Martens, Y. Poffet, P. R. Soria, R. Fitch, and S. Sukkarieh. Geometric priors for gaussian process implicit surfaces. *IEEE Robotics and Automation Letters*, 2(2):373–380, April 2017.

[10] Pablo Ramón Soria and Begoña C. Arrue. *"Aerial robotics manipulation" – Chapter 4.7 "Object Detection and Probabilistic Object Representation for Grasping with Two Arms"*. Springer Tracts on Advanced Robotics, 2018.

[11] Pablo Ramon Soria, Robert Bevec, Begoña C. Arrue, Ales Ude, and Aníbal Ollero. Extracting objects for aerial manipulation on uavs using low cost stereo sensors. In *Sensors*, 2016.

[12] Pablo Ramon Soria, Fouad Sukkar, Wolfram Martens, B. C. Arrue, and Robert Fitch. Multi-view probabilistic segmentation of pome fruit with a low-cost rgb-d camera. In Anibal Ollero, Alberto Sanfeliu, Luis Montano, Nuno Lau, and Carlos Cardeira, editors, *ROBOT 2017: Third Iberian Robotics Conference*, pages 320–331, Cham, 2018. Springer International Publishing.

[13] Pablo Ramon Soria, B.C. Arrue, and Anibal Ollero. Grasp planning and visual servoing for aerial dual manipulator outdoors - (under submission). *Elsevier - Engineering*, 2019.

[14] A. Suarez, P. R. Soria, G. Heredia, B. C. Arrue, and A. Ollero. Anthropomorphic, compliant and lightweight dual arm system for aerial manipulation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 992–997, Sept 2017.

[15] P. Ramon Soria, B.C. Arrue, and A. Ollero. A 3d-printable docking system for aerial robots: Controlling aerial manipulators in outdoor industrial applications. *IEEE Robotics Automation Magazine*, pages 1–1, 2019.

[16] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *Proc. of IEEE ICRA*, pages 1–4, 2011.

[17] Michele Fenzi, Ralf Dragon, Laura Leal-Taixé, Bodo Rosenhahn, and Jörn Ostermann. 3d object recognition and pose estimation for multiple objects using multi-prioritized ransac and model updating. In Axel Pinz, Thomas Pock, Horst Bischof, and Franz Leberl, editors, *Pattern Recognition*, pages 123–133, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[18] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, 2001.

[19] Hao Men, Biruk Gebre, and Kishore Pochiraju. Color point cloud registration with 4d icp algorithm. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 1511 – 1516, 06 2011.

[20] Naresh Marturi, Marek Kopicki, Alireza Rastegarpanah, Vijaykumar Rajasekaran, Maxime Adjigble, Rustam Stolkin, Ales Leonardis, and Yasemin Bekiroglu. Dynamic grasp and trajectory planning for moving objects. 08 2018.

[21] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[22] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. *SURF: Speeded Up Robust Features*, pages 404–417. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[23] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2564–2571, Washington, DC, USA, 2011. IEEE Computer Society.

[24] E. Tola, V.Lepetit, and P. Fua. A Fast Local Descriptor for Dense Matching. In *Proceedings of Computer Vision and Pattern Recognition*, Alaska, USA, 2008.

[25] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. *BRIEF: Binary Robust Independent Elementary Features*, pages 778–792. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[26] E. Rosten, R. Porter, and T. Drummond. Faster and better: A machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):105–119, Jan 2010.

[27] J. Shi and C. Tomasi. Good features to track. In *Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[28] Faraj Alhwarin, Danijela Ristić-Durrant, and Axel Gräser. *VF-SIFT: Very Fast SIFT Feature Matching*, pages 222–231. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[29] Michael Grabner, Helmut Grabner, and Horst Bischof. *Fast Approximated SIFT*, pages 918–927. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[30] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

[31] R. Hänsch, T. Weber, and O. Hellwich. Comparison of 3D interest point detectors and descriptors for point cloud fusion. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 57–64, August 2014.

[32] S. Filipe and L. A. Alexandre. A comparative evaluation of 3d keypoint detectors in a rgb-d object dataset. In *2014 International Conference on Computer Vision Theory and Applications (VISAPP)*, volume 1, pages 476–483, Jan 2014.

[33] V. K. Ghorpade, P. Checchin, L. Malaterre, and L. Trassoudaine. Performance evaluation of 3d keypoint detectors for time-of-flight depth data. In *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1–6, Nov 2016.

[34] Jingdao Chen, Yihai Fang, and Yong K. Cho. Performance evaluation of 3d descriptors for object recognition in construction applications. *Automation in Construction*, 86:44 – 52, 2018.

[35] Ivan Sipiran and Benjamin Bustos. Harris 3d: a robust extension of the harris operator for interest point detection on 3d meshes. *The Visual Computer*, 27(11):963, Jul 2011.

[36] A. Flint, A. Dick, and A. v. d. Hengel. Thrift: Local 3d structure recognition. In *9th Biennial Conference of the Australian Pattern Recognition Society on Digital Image Computing Techniques and Applications (DICTA 2007)*, pages 182–188, Dec 2007.

[37] Bastian Steder, Radu Bogdan Rusu, Kurt Konolige, and Wolfram Burgard. Narf: 3d range image features for object recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, October 8, 2010 2010.

[38] Andrea Frome, Daniel Huber, Ravi Kolluri, Thomas Bülow, and Jitendra Malik. Recognizing objects in range data using regional point descriptors. In Tomáš Pajdla and Jiří Matas, editors, *Computer Vision - ECCV 2004*, pages 224–237, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[39] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE International Conference on Robotics and Automation*, pages 3212–3217, May 2009.

[40] Samuele Salti, Federico Tombari, and Luigi Di Stefano. Shot: Unique signatures of histograms for surface and texture description. *Computer Vision and Image Understanding*, 125:251 – 264, 2014.

[41] F. Tombari, S. Salti, and L. Di Stefano. A combined texture-shape descriptor for enhanced 3d feature matching. In *2011 18th IEEE International Conference on Image Processing*, pages 809–812, Sept 2011.

[42] Iryna Gordon and David G. Lowe. *What and Where: 3D Object Recognition with Accurate Pose*, pages 67–82. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[43] C. Tsai and S. Tsai. Simultaneous 3d object recognition and pose estimation based on rgb-d images. *IEEE Access*, 6:28859–28869, 2018.

[44] A. T. Miller, S. Knoop, H. I. Christensen, and P. K. Allen. Automatic grasp planning using shape primitives. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 2, pages 1824–1829 vol.2, Sept 2003.

[45] Matthias Nieuwenhuisen, Jörg Stückler, Alexander Berner, Reinhard Klein, and Sven Behnke. Shape-primitive based object recognition and grasping. In *ROBOTIK*, 2012.

[46] Efrain Lopez-Damian, Daniel Sidobre, and Rachid Alami. Grasp planning for non-convex objects. In *International symposium on robotics*, volume 36, page 167. unknown, 2005.

[47] Oliver Williams and Andrew Fitzgibbon. Gaussian process implicit surfaces. In *Gaussian Processes In Practice*, June 2006.

[48] M. P. Gerardo-Castro, T. Peynot, F. Ramos, and R. Fitch. Robust multiple-sensing-modality data fusion using gaussian process implicit surfaces. In *Proc. of FUSION*, pages 1–8, 2014.

[49] S. Dragiev, M. Toussaint, and M. Gienger. Gaussian process implicit surfaces for shape estimation and grasping. In *Proc. of IEEE/RSJ ICRA*, pages 2845–2850, May 2011.

[50] M. Björkman, Y. Bekiroglu, V. Högman, and D. Kragic. Enhancing visual perception of shape through tactile glances. In *Proc. of IEEE/RSJ IROS*, pages 3180–3186, Nov 2013.

[51] Soohwan Kim and Jonghyuk Kim. *GPmap: A Unified Framework for Robotic Mapping Based on Sparse Gaussian Processes*, pages 319–332. Results of the 9th International Conference, Field and Service Robotics. Springer International Publishing, 2015.

[52] Christopher KI Williams and Carl Edward Rasmussen. Gaussian processes for regression. In *Advances in neural information processing systems*, pages 514–520, 1996.

[53] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[54] R. Adler. *The Geometry of Random Fields*. Society for Industrial and Applied Mathematics, 2010.

[55] Michael L Stein. *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media, 2012.

[56] The princeton shape benchmark. In *Proc. of the Shape Modeling International 2004*, SMI '04, pages 167–178, Washington, DC, USA, 2004. IEEE Computer Society.

[57] Michael Gschwandtner, Roland Kwitt, Andreas Uhl, and Wolfgang Pree. BlenSor: Blender Sensor Simulation Toolbox Advances in Visual Computing. volume 6939 of *Lecture Notes in Computer Science*, chapter 20, pages 199–208. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2011.

[58] Jeremie Papon, Alexey Abramov, Markus Schoeler, and Florentin Wörgötter. Voxel cloud connectivity segmentation - supervoxels for point clouds. In *Proc. of the 2013 IEEE CVPR*, CVPR '13, pages 2027–2034, Washington, DC, USA, 2013. IEEE Computer Society.

[59] Máximo A. Roa and Raúl Suárez. Grasp quality measures: review and performance. *Autonomous Robots*, 38(1):65–88, Jan 2015.

[60] V. D. Nguyen. Constructing force-closure grasps in 3d. In *Proceedings. 1987 IEEE International Conference on Robotics and Automation*, volume 4, pages 240–245, Mar 1987.

[61] Beatriz León, Antonio Morales, and Joaquin Sancho-Bru. *Robot Grasping Foundations*, pages 15–31. Springer International Publishing, Cham, 2014.

[62] B. Mishra, J. T. Schwartz, and M. Sharir. On the existence and synthesis of multifinger positive grips. *Algorithmica*, 2(1):541–558, Nov 1987.

[63] C. Ferrari and J. Canny. Planning optimal grasps. In *Proceedings 1992 IEEE International Conference on Robotics and Automation*, pages 2290–2295 vol.3, May 1992.

[64] Zexiang Li and S. Sastry. Task oriented optimal grasping by multifingered robot hands. In *Proceedings. 1987 IEEE International Conference on Robotics and Automation*, volume 4, pages 389–394, Mar 1987.

[65] Nancy Pollard. Parallel methods for synthesizing whole-hand grasps from generalized prototypes. In *Technical Report AI-TR 1464*, 01 1994.

[66] C. Borst, M. Fischer, and G. Hirzinger. Grasp planning: how to choose a suitable task wrench space. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 1, pages 319–325 Vol.1, April 2004.

[67] V. N. Christopoulos and P. Schrater. Handling shape and contact location uncertainty in grasping two-dimensional planar objects. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1557–1563, Oct 2007.

[68] J. Weisz and P. K. Allen. Pose error robust grasping from contact wrench space metrics. In *2012 IEEE International Conference on Robotics and Automation*, pages 557–562, May 2012.

[69] Shehan Caldera, Alexander Rassau, and Douglas Chai. Review of deep learning methods in robotic grasp detection. *Multimodal Technologies Interact*, 2018.

[70] Ian Lenz, Honglak Lee, and Ashutosh Saxena. Deep learning for detecting robotic grasps. *CoRR*, abs/1301.3592, 2013.

[71] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[72] Andrew S Glassner. *An introduction to ray tracing*. Elsevier, 1989.

[73] Tomas Möller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *J. Graph. Tools*, 2(1):21–28, October 1997.

[74] Hanan Samet. Implementing ray tracing with octrees and neighbor finding. *Computers And Graphics*, 13:445–460, 1989.

[75] B. Kehoe, D. Berenson, and K. Goldberg. Toward cloud-based grasping with uncertainty in shape: Estimating lower bounds on achieving force closure with zero-slip push grasps. In *2012 IEEE International Conference on Robotics and Automation*, pages 576–583, May 2012.

[76] Jeffrey Mahler, Santosh Patil, Ben Kehoe, Joran van den Berg, Matei Ciocarlie, Pieter Abbeel, and Kenneth Goldberg. Gp-gpis-opt: Grasp planning with shape uncertainty using gaussian process implicit surfaces and sequential convex programming. 2015:4919–4926, 06 2015.

[77] Rosen Diankov and James Kuffner. Openrave: A planning architecture for autonomous robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, 79, 2008.

[78] Antonio Barrientos. Fundamentos de robótica. Technical report, McGraw Hill, 2007.

[79] A. Goldenberg, B. Benhabib, and R. Fenton. A complete generalized solution to the inverse kinematics of robots. *IEEE Journal on Robotics and Automation*, 1(1):14–20, March 1985.

[80] Nicolas Courty and Elise Arnaud. Inverse kinematics using sequential monte carlo methods. In Francisco J. Perales and Robert B. Fisher, editors, *Articulated Motion and Deformable Objects*, pages 1–10, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[81] Wampler II and Larry Leifer. Applications of damped least-squares methods to resolved-rate and resolved-acceleration control of manipulators. 110, 03 1988.

[82] Samuel Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. 17, 05 2004.

[83] Mukulika Ghosh, Nancy M. Amato, Yanyan Lu, and Jyh-Ming Lien. Fast approximate convex decomposition using relative concavity. *Comput. Aided Des.*, 45(2):494–504, February 2013.

[84] K. Mamou and F. Ghorbel. A simple and efficient approach for 3d mesh approximate convex decomposition. In *2009 16th IEEE International Conference on Image Processing (ICIP)*, pages 3501–3504, Nov 2009.

[85] Guillermo Heredia, AE Jimenez-Cano, I Sanchez, Domingo Llorente, V Vega, J Braga, JA Acosta, and Aníbal Ollero. Control of a multirotor outdoor aerial manipulator. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 3417–3422. IEEE, 2014.

[86] Joseph DeGol, Timothy Bretl, and Derek Hoiem. Chromatag: A colored marker and fast detection algorithm. *arXiv preprint arXiv:1708.02982*, 2017.

[87] Shashank Bhatia and Stephan K Chalup. Segmenting salient objects in 3d point clouds of indoor scenes using geodesic distances. *Journal of Signal and Information Processing*, 4(03):102, 2013.

[88] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Robot Manipulation*. Springer Publishing Company, Incorporated, 2013.

[89] Chunhu Li, Bo YANG, and Chun-hu LI. Deep learning based visual tracking: A review. *DEStech Transactions on Computer Science and Engineering*, 07 2017.

[90] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *arXiv preprint arXiv:1807.05511*, 2018.

[91] Anton Milan, Seyed Hamid Rezatofighi, Anthony R Dick, Ian D Reid, and Konrad Schindler. Online multi-target tracking using recurrent neural networks. In *AAAI*, volume 2, page 4, 2017.

[92] Thanh-Toan Do, Ming Cai, Trung Pham, and Ian D. Reid. Deep-6dpose: Recovering 6d object pose from a single RGB image. *CoRR*, abs/1802.10367, 2018.

[93] David MacKay. An example inference task: clustering. *Information theory, inference and learning algorithms*, 20:284–292, 2003.

[94] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[95] Dan Pelleg and Andrew W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 727–734, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[96] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Graphics and Image Processing*, 24(6):381–395, 1981.

[97] G. K. L. Tam, Z. Cheng, Y. Lai, F. C. Langbein, Y. Liu, D. Marshall, R. R. Martin, X. Sun, and P. L. Rosin. Registration of 3d point clouds and meshes: A survey from rigid to nonrigid. *IEEE Transactions on Visualization and Computer Graphics*, 19(7):1199–1217, July 2013.

[98] Haili Chui and Anand Rangarajan. A new point matching algorithm for non-rigid registration. *Computer Vision and Image Understanding*, 89(2-3):114–141, 2003.

[99] Aleksandr Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-icp. In *Robotics: science and systems*, volume 2, page 435, 2009.

[100] Alvaro Collet Romea, Dmitry Berenson, Siddhartha Srinivasa, and David Ferguson . Object recognition and full pose registration from a single image for robotic manipulation. In *IEEE International Conference on Robotics and Automation (ICRA '09)*, May 2009.

[101] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*, ICCV '99, pages 298–372, London, UK, UK, 2000. Springer-Verlag.

[102] Donald W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[103] Alvaro Collet, Dmitry Berenson, Siddhartha S. Srinivasa, and Dave Ferguson. Object recognition and full pose registration from a single image for robotic manipulation. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, ICRA'09, pages 3534–3541, Piscataway, NJ, USA, 2009. IEEE Press.

[104] H. van Hoof, O. Kroemer, and J. Peters. Probabilistic segmentation and targeted exploration of objects in cluttered environments. *IEEE Trans. Robot.*, 30(5):1198–1209, 2014.

[105] Joni Pajarinen and Ville Kyrki. Decision making under uncertain segmentations. In *Proc. of IEEE ICRA*, pages 1303–1309, 2015.

[106] C. Cadena and J. Kosecká. Semantic parsing for priming object detection in indoors RGB-D scenes. *Int. J. Robot. Res.*, 34(4-5):582–597, 2015.

[107] W. Martens, Y. Poffet, P. R. Soria, R. Fitch, and S. Sukkarieh. Geometric priors for Gaussian process implicit surfaces. *IEEE Robotics and Automation Letters*, 2(2):373–380, April 2017.

[108] Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

[109] Gareth O Roberts and Adrian FM Smith. Simple conditions for the convergence of the gibbs sampler and metropolis-hastings algorithms. *Stochastic processes and their applications*, 49(2):207–216, 1994.

[110] Min-Ling Zhang and Zhi-Hua Zhou. A k-nearest neighbor based algorithm for multi-label classification. In *Granular Computing, 2005 IEEE International Conference on*, volume 2, pages 718–721. IEEE, 2005.

[111] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[112] Gabriella Csurka, Christopher Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In *ECCV Workshop on statistical learning in computer vision*, pages 1–22, 2004.

[113] Vladimir Naumovich Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.

[114] Vladimir Vapnik. *The nature of statistical learning theory*. Springer-Verlag, 1995.

[115] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 841–848. MIT Press, 2002.

[116] Kuan-Yu Menphis Chen and Yufei Wang. Latent dirichlet allocation, 2007.

[117] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[118] Amr Ahmed, Moahmed Aly, Joseph Gonzalez, Shravan Narayanamurthy, and Alexander J Smola. Scalable inference in latent variable models. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 123–132. ACM, 2012.

[119] Colorado Reed. Latent dirichlet allocation: Towards a deeper understanding, 2012.

[120] Tom Griffiths. Gibbs sampling in the generative model of latent dirichlet allocation. 2002.

[121] Ian Porteous, David Newman, Alexander Ihler, Arthur Asuncion, Padhraic Smyth, and Max Welling. Fast collapsed gibbs sampling for latent dirichlet allocation. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 569–577. ACM, 2008.

[122] Radford M Neal. Markov chain sampling methods for dirichlet process mixture models. *Journal of computational and graphical statistics*, 9(2):249–265, 2000.

[123] Sonia Jain and Radford M Neal. A split-merge markov chain monte carlo procedure for the dirichlet process mixture model. *Journal of computational and Graphical Statistics*, 13(1):158–182, 2004.

[124] Thomas Hofmann. Unsupervised learning by probabilistic latent semantic analysis. *Machine Learning*, 42(1):177–196, Jan 2001.

[125] J. Xu, G. Ye, Y. Wang, G. Herman, B. Zhang, and J. Yang. Incremental em for probabilistic latent semantic analysis on human action recognition. In *2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance*, pages 55–60, Sept 2009.

[126] Thomas Hofmann. Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, pages 289–296, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[127] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer vision and Image understanding*, 106(1):59–70, 2007.

[128] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[129] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[130] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[131] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[132] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[133] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.

[134] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[135] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[136] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow:

Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[137] Aniket Murarka and Benjamin Kuipers. Using cad drawings for robot navigation. In *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, volume 2, pages 678–683. IEEE, 2001.

[138] Frederic Bosche, Carl T Haas, and Carlos H Caldas. 3d cad drawing as apriori knowledge for machine vision in construction. In *Proceedings of the 1st Annual Inter-University Symposium on Infrastructure Management, Waterloo, Ontario, Canada, August*, volume 6. Citeseer, 2005.

[139] Yang Song, Mingyang Guan, Wee Peng Tay, Choi Look Law, and Changyun Wen. Uwb/lidar fusion for cooperative range-only slam. *arXiv preprint arXiv:1811.02854*, 2018.

[140] Ben Bellekens, Vincent Spruyt, Rafael Berkvens, and Maarten Weyn. A survey of rigid 3d pointcloud registration algorithms. In *AMBIENT 2014: the Fourth International Conference on Ambient Computing, Applications, Services and Technologies, August 24-28, 2014, Rome, Italy*, pages 8–13, 2014.

[141] François Pomerleau, Francis Colas, Roland Siegwart, et al. A review of point cloud registration algorithms for mobile robotics. *Foundations and Trends® in Robotics*, 4(1):1–104, 2015.

[142] Sebastian Thrun and Michael Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, 25(5-6):403–429, 2006.

[143] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[144] L Cordesses, P Martinet, B Thuilot, and M Berducat. Gps-based control of a land vehicle. In *Proceedings of the IAARC/IFAC/IEEE International Symposium on Automation and Robotics in Construction, ISARC*, volume 99, pages 22–24, 1999.

[145] David Obdržálek. Robot localization. 2012.

[146] Felix Endres, Jürgen Hess, Jürgen Sturm, Daniel Cremers, and Wolfram Burgard. 3-d mapping with an rgb-d camera. *IEEE Transactions on Robotics*, 30(1):177–187, 2014.

[147] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[148] Montiel-J. M. M. Mur-Artal, Raúl and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[149] Viorela Ila, Lukas Polok, Marek Solony, and Pavel Svoboda. Slam++-a highly efficient and temporally scalable incremental slam framework. *The International Journal of Robotics Research*, 36(2):210–230, 2017.

[150] Cyril Roussillon, Aurélien Gonzalez, Joan Solà, Jean-Marie Codol, Nicolas Mansard, Simon Lacroix, and Michel Devy. RT-SLAM: A generic and real-time visual SLAM implementation. *CoRR*, abs/1201.5450, 2012.

[151] Dieter Fox, Jeffrey Hightower, Lin Liao, Dirk Schulz, and Gaetano Borriello. Bayesian filtering for location estimation. *IEEE pervasive computing*, (3):24–33, 2003.

[152] S. Y. Chen. Kalman filter for robot vision: A survey. *IEEE Transactions on Industrial Electronics*, 59(11):4409–4420, Nov 2012.

[153] Sebastian Thrun. Particle filters in robotics. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 511–518. Morgan Kaufmann Publishers Inc., 2002.

[154] Brian Patrick Williams, Mark Joseph Cummins, José Neira, Paul Newman, Ian D. Reid, and Juan D. Tardós. An image-to-map loop closing method for monocular slam. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2053–2059, 2008.

[155] Stephanie Lowry, Niko Sünderhauf, Paul Newman, John J Leonard, David Cox, Peter Corke, and Michael J Milford. Visual place recognition: A survey. *IEEE Transactions on Robotics*, 32(1):1–19, 2016.

[156] Dorian Gálvez-López and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012.

[157] E. Garcia-Fidalgo and A. Ortiz. On the use of binary feature descriptors for loop closure detection. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–8, Sept 2014.

[158] Yi Hou, Hong Zhang, and Shilin Zhou. Convolutional neural network-based image representation for visual loop closure detection. *CoRR*, abs/1504.05241, 2015.

[159] Xiang Gao and Tao Zhang. Unsupervised learning to detect loops using deep neural networks for visual slam system. *Autonomous robots*, 41(1):1–18, 2017.

[160] Xiwu Zhang, Yan Su, and Xinhua Zhu. Loop closure detection for visual slam systems using convolutional neural network. In *Automation and Computing (ICAC), 2017 23rd International Conference on*, pages 1–6. IEEE, 2017.

[161] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g 2 o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3607–3613. IEEE, 2011.

[162] L. M. Paz, P. PiniÉs, J. D. TardÓs, and J. Neira. Large-scale 6-dof slam with stereo-in-hand. *IEEE Transactions on Robotics*, 24(5):946–957, Oct 2008.

[163] M. Tomono. Robust 3d slam with a stereo camera based on an edge-point icp algorithm. In *2009 IEEE International Conference on Robotics and Automation*, pages 4306–4311, May 2009.

[164] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. Deepstereo: Learning to predict new views from the world's imagery. *CoRR*, abs/1506.06825, 2015.

[165] Xiao Song, Xu Zhao, Hanwen Hu, and Liangji Fang. Edgestereo: A context integrated residual pyramid network for stereo matching. *Asian Conference on Computer Vision (ACCV)*, 2018.

[166] Leonid Keselman, John Iselin Woodfill, Anders Grunnet-Jepsen, and Achintya Bhowmik. Intel realsense stereoscopic depth cameras. *CoRR*, abs/1705.05548, 2017.

[167] P. Marziliano, F. Dufaux, S. Winkler, and T. Ebrahimi. A no-reference perceptual blur metric. In *The International Conference on Image Processing*, volume 3, pages III–57–III–60, 2002.

[168] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz. Towards 3d point cloud based object maps for household environments. *Robotics and Autonomous Systems*, 56(11):927–941, 2008.

[169] Y. Roth-Tabak and R. Jain. Building an environment model using depth information. *Computer*, 22(6):85–90, June 1989.

[170] Héctor García de Marina, Felipe Espinosa, and Carlos Santos. Adaptive uav attitude estimation employing unscented kalman filter, foam and low-cost mems sensors. *Sensors (Basel, Switzerland)*, 12(7):9566–85, 2012.

[171] J. Lobo, J. Dias, P Corke, P Gemeiner, P Einramhof, and M Vincze. Relative pose calibration between visual and inertial sensors. *The International Journal of Robotics Research*, 26(6):561–575, 2007.

[172] M Nießner, a Dai, and M Fisher. Combining inertial navigation and icp for real-time 3d surface reconstruction. *Eurographics*, pages 1–4, 2014.

[173] C. Di Franco, G. Franchino, and M. Marinoni. A biased extended kalman filter for indoor localization of a mobile agent using low-cost imu and usb wireless sensor network.

[174] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. volume 33, pages 1255–1262, 2017.

[175] Kin Leong Ho and Paul Newman. Detecting loop closure with scene sequences. *International Journal of Computer Vision*, 74(3):261–286, Sep 2007.

[176] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197, 1981.

[177] B. Glocker, S. Izadi, J. Shotton, and A. Criminisi. Real-time rgb-d camera relocalization. In *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 173–179, Oct 2013.

[178] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 559–568, New York, NY, USA, 2011. ACM.

[179] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[180] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, and Jose Garcia-Rodriguez. A review on deep learning techniques applied to semantic segmentation. *arXiv preprint arXiv:1704.06857*, 2017.

[181] L. Ran, Y. Zhang, and G. Hua. Cannet: Context aware nonlocal convolutional networks for semantic image segmentation. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 4669–4673, Sept 2015.

[182] Paul Oh Stjepan Bogdan Matko Orsag, Christopher Korpela. *Aerial Manipulation*. Springer, 2018.

[183] Hossein Bonyan Khamseh, Farrokh Janabi-Sharifi, and Abdelkader Abdessameud. Aerial manipulation—a literature survey. *Robotics and Autonomous Systems*, 107:221 – 235, 2018.

[184] P. E. I. Pounds, D. R. Bersak, and A. M. Dollar. Grasping from the air: Hovering capture and load stability. In *2011 IEEE International Conference on Robotics and Automation*, pages 2491–2498, May 2011.

[185] S. B. Backus, L. U. Odhner, and A. M. Dollar. Design of hands for aerial manipulation: Actuator number and routing for grasping and perching. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 34–40, Sept 2014.

[186] J. Qi, J. Kang, and X. Lu. Design and research of uav autonomous grasping system. In *2017 IEEE International Conference on Unmanned Systems (ICUS)*, pages 126–131, Oct 2017.

[187] D. Mellinger, Q. Lindsey, M. Shomin, and V. Kumar. Design, modeling, estimation and control for aerial grasping and manipulation. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2668–2673, Sept 2011.

[188] V. Ghadiok, J. Goldin, and W. Ren. Autonomous indoor aerial gripping using a quadrotor. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4645–4651, Sept 2011.

[189] Usman A. Fiaz, Mohamed Abdelkader, and Jeff Shamma. An intelligent gripper design for autonomous aerial transport with passive magnetic grasping and dual-impulsive release. 07 2018.

[190] C. Papachristos, K. Alexis, and A. Tzes. Efficient force exertion for aerial robotic manipulation: Exploiting the thrust-vectoring authority of a tri-tiltrotor uav. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4500–4505, May 2014.

[191] T. Ikeda, S. Yasui, M. Fujihara, K. Ohara, S. Ashizawa, A. Ichikawa, A. Okino, T. Oomichi, and T. Fukuda. Wall contact by octo-rotor uav with one dof manipulator for bridge inspection. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5122–5127, Sept 2017.

[192] Todd W Danko, Kenneth P Chaney, and Paul Y Oh. A parallel manipulator for mobile manipulating uavs. In *Technologies for Practical Robot Applications (TePRA), 2015 IEEE International Conference on*, pages 1–6. IEEE, 2015.

[193] Jonathan Hodgins. *H-Delta: design and applications of a novel 5 degree of freedom parallel robot*. PhD thesis, 2018.

[194] K. Kondak, F. Huber, M. Schwarzbach, M. Laiacker, D. Sommer, M. Bejar, and A. Ollero. Aerial manipulation robot composed of an autonomous helicopter and a 7 degrees of freedom industrial manipulator. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2107–2112, May 2014.

[195] M. Laiacker, F. Huber, and K. Kondak. High accuracy visual servoing for aerial manipulation using a 7 degrees of freedom industrial manipulator. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1631–1636, Oct 2016.

[196] Christopher Korpela, Matko Orsag, and Paul Oh. Towards valve turning using a dual-arm aerial manipulator. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 3411–3416. IEEE, 2014.

[197] A. Suarez, G. Heredia, and A. Ollero. Design of an anthropomorphic, compliant, and lightweight dual arm for aerial manipulation. *IEEE Access*, 6:29173–29189, 2018.

[198] Prodrone. *https://www.prodrone.jp*, 2016.

[199] Small mini pc. *http://www.intel.es/content/www/es/es/nuc/overview.html*, 2016.

[200] Arduino. *https://www.arduino.cc*, 2016.

[201] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[202] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[203] James Bruce, Tucker Balch, and Manuela Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 3, pages 2061–2066. IEEE, 2000.

[204] Ángel R. Castaño, Fran Real, Pablo Ramón-Soria, Jesús Capitán, Víctor Vega, Begoña C. Arrue, Arturo Torres-González, and Aníbal Ollero. Al-robotics team: A cooperative multi-unmanned aerial vehicle approach for the mohamed bin zayed international robotic challenge. *Journal of Field Robotics*, 0(0).