Technical University of Crete, Greece
School of Electrical and Computer Engineering

Aerial and Ground Robot Collaboration for Autonomous Mapping in Search and Rescue Missions

Dimitrios Chatziparaschis

Thesis Committee

Associate Professor Michail G. Lagoudakis (ECE)

Professor Michalis Zervakis (ECE)

Associate Professor Panagiotis Partsinevelos (MRE)

Chania, October 2018

Πολυτεχνειο Κρητης

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Συνεργασία Εναέριων και Επίγειων Ρομπότ για Αυτόνομη Χαρτογράφηση σε Αποστολές Έρευνας και Διάσωσης

Δημήτριος Χατζηπαράσχης

Εξεταστική Επιτροπή

Αναπληρωτής Καθηγητής Μιχαήλ Γ. Λαγουδάκης (ΗΜΜΥ)

Καθηγητής Μιχαήλ Ζερβάκης (ΗΜΜΥ)

Αναπληρωτής Καθηγητής Παναγιώτης Παρτσινέβελος (ΜΗΧΟΠ)

Χανιά, Οκτώβριος 2018

Abstract

Nowadays, Humanitarian Crisis scenarios occur on daily basis and typically require immediate rescue intervention. In most cases, the scene conditions may be prohibitive for human rescuers to provide instant aid, because of hazardous, unexpected, and human-unfriendly situations. Those scenarios are ideal for autonomous mobile robot systems to act on, by searching and even rescuing individuals, therefore enhancing rescuers' actions and keeping them safe. In this thesis, we present a ground-aerial robot collaboration approach, in which a quadcopter and a humanoid robot solve a search-and-rescue scenario locally, without any GNSS/GPS system dependencies. Specifically, the quadcopter uses a combination of Simultaneous Localization and Mapping (SLAM) and OctoMapping approaches to extract a 2.5D occupancy grid map of the unknown area in relation to the height of the humanoid robot. At the same time, the quadcopter searches for the humanoid robot in the field and localizes it in the map frame. The humanoid robot awaits for a goal position in the created map and executes a path planning algorithm to estimate the footstep navigation trajectory for reaching the goal. As the humanoid robot navigates, it localizes itself in the map using an adaptive Monte-Carlo Localization algorithm by combining local odometry data with spatial observations from the quadcopter. Finally, the humanoid robot performs visual human body detection using camera data through a Darknet pre-trained neural network. The entire project has been implemented within the Robot Operating System (ROS) and is available as an open source package. The proposed robot collaboration scheme has been tested both in interior and exterior physical environments under real-time conditions. The main advantage of the proposed scheme is the joint-ability to perceive the unknown scene from the air using the quadcopter, while at the same time performing close inspections on the ground using the humanoid robot.

Περίληψη

Στις μέρες μας, καθημερινά συμβαίνουν σενάρια ανθρωπιστικής κρίσης και τυπικά χρήζουν άμεσης διασωστικής επέμβασης. Στις περισσότερες περιπτώσεις, οι συνθήκες αντιμετώπισης μπορεί να είναι απαγορευτικές για τους διασώστες στη παροχή άμεσης βοήθειας, εξαιτίας των επιβλαβών, απρόβλεπτων και μη φιλικών προς τον άνθρωπο καταστάσεων. Τα σενάρια αυτά είναι ιδανικά για τη δράση αυτόνομων κινητών ρομποτικών συστημάτων, καθώς μπορούν να αναζητήσουν και να διασώσουν ανθρώπους σε ανάγκη, ενισχύοντας έτσι το έργο των διασωστών, κρατώντας τους ασφαλείς. Η παρούσα διπλωματική εργασία παρουσιάζει μια προσέγγιση συνεργασίας ενός επίγειου και ενός εναέριου ρομπότ, στην οποία ένα τετρακόπτερο και ένα ανθρωποειδές ρομπότ αντιμετωπίζουν τοπικά ένα σενάριο έρευνας και διάσωσης, χωρίς εξάρτηση από σύστημα GNSS/GPS. Ακριβέστερα, το τετρακόπτερο χρησιμοποιεί τον συνδυασμό των Simultaneous Localization and Mapping (SLAM) και OctoMapping μεθόδων για να εξάγει ένα 2.5D occupancy grid χάρτη της άγνωστης περιοχής σε σχέση με το ύψος του ανθρωποειδούς ρομπότ. Ταυτοχρόνως, το τετρακόπτερο αναζητεί το ανθρωποειδές ρομπότ μέσα στον χώρο και προσδιορίζει τη θέση του μέσα στο σύστημα συντεταγμένων του χάρτη. Το ανθρωποειδές ρομπότ αναμένει μια επιθυμητή θέση ως στόχο μέσα στον χάρτη και εκτελεί έναν αλγόριθμο path planning για να οργανώσει την διαδρομή του στον χώρο, κάνοντας την εκτίμηση τοποθέτησης των βημάτων και πατημάτων του μέχρι να φτάσει στον τελικό στόχο. Τέλος, το ανθρωποειδές ρομπότ πραγματοποιεί οπτική αναγνώριση ανθρώπων χρησιμοποιώντας ένα προ-εκπαιδευμένο νευρωνικό δίκτυο Darknet πάνω στα δεδομένα των καμερών του. Η συνολική εργασία έχει υλοποιηθεί μέσω του Robot Operating System (ROS) και είναι διαθέσιμη ως πακέτο ανοιχτού κώδικα. Η προτεινόμενη προσέγγιση συνεργασίας έχει δοκιμαστεί σε εσωτερικά, αλλά και εξωτερικά, περιβάλλοντα, σε συνθήκες πραγματικού χρόνου. Το βασικό πλεονέκτημα της προτεινόμενης προσέγγισης είναι η συνδυαστική ικανότητα αντίληψης μιας άγνωστης περιοχής από αέρος με τη χρήση του τετρακόπτερου, παράλληλα με την ικανότητα κοντινών παρατηρήσεων στο έδαφος χρησιμοποιώντας ένα ανθρωποειδές ρομπότ.

Acknowledgements

First of all, I am grateful to my professors Michail Lagoudakis and Panagiotis Partsinevelos for letting me do this research in their laboratories. I would like to thank them for their guidance and support throughout this work and beyond that.

My friends and colleagues in both SenseLab and Kouretes teams, who shared with me amazing experiences in the recent years.

My family, my close friends and especially, Angelos A., Stelios M., George Vogia., George T. and George Vougioukas, as nothing would be the same without them.

Contents

List of Figures

Chapter 1

Introduction

Nowadays, Humanitarian Crisis scenarios occur on daily basis and typically require immediate rescue intervention. In most cases, the scene conditions may be prohibitive for the human rescuers to provide instant aid, as they have to act on extreme environments, like air-poisoned environments, radioactive environments or collapsed buildings. In addition, sometimes there is a need to immediate examine a wide area, in order to determine an approaching scenario.

Many organizations and research teams are developing rescuing robots in order to assist "human" crisis intervention teams. These mobile robots can be equipped with a variety of sensors, actuators and embedded processing units, depending on the scenario they need to assist. As a result, due to their specialized structure they can achieve sufficient maneuverability in the terrain that where designed for, and thus can perform mapping, searching and reconnaissance procedures by processing the captured data from their sensors. These robots can either be autonomous or remotely controlled, depending the current setup, and thus enhance the rescuers' actions.

Although, in cases of collapsed buildings or dense obstructed areas, the robot remote control may be limited or inapplicable, so the robot needs to be able to determine its own pose in the unknown environment. Moreover, assuming that the area map is given we can not ensure that its state is up to date to the current environment state, specifically after occurred crisis scenarios. Hence, the mobile robots need to be able to construct the map of their surrounding environment and also localize themselves in it, to adapt on environment changes without human supervision. Not to mention, in multi-robot scenarios it is critical also for the robots to be able to recognize and localize other robots

in the field, in respect to them or in respect to the generated map coordinate system.

Additionally, assuming that the mobile robots are aware of the generated map and their positions in it, path planning approaches ensure that given a target location in the map frame, the optimal path will be estimated for them to follow. Also at the same time, during the robots' exploration in the world environment, human detection methods should be applied, as locating a missing person is one of the primary tasks of Search and Rescue scenarios.

Consequently, rescuing mobile robots need to be specially tuned and programmed to be able to face problems as a human rescuer would do, and most importantly to able to cooperate with other rescuing robots, in order to accomplish their missions faster and more efficiently.

1.1 Thesis Contribution

This thesis describes a ground-aerial robot collaboration approach, in which a quadcopter and a humanoid robot cooperate to solve a real-world Search and Rescue scenario. Specifically, we assign specific roles on each robot, in order to cooperate to locate a human in need. By assuming that the ground robot can not take any spatial measurements, we approach its localization problem through the aerial robot's view. Also, the ground robot is responsible for locating the victim in the area and also for estimating its optimal path to given target in the map.

The quadcopter performs a combination of 2D and 3D mapping methods in order to extract a 2.5D map specifically made for the humanoid robot's navigation. The quadcopter uses a multi-stereo camera system in order to extract spatially depth information and then use an octree mapping method, by which can construct a 3D world representation. At the same time, a SLAM approach is performed to estimate quadcopter's poses in the generated map and also constructs a 2D map, based on laser ranging data. As a result, a world representation is obtained and thus this information is transfered on the humanoid robot, to proceed its mission.

Furthermore, the quadcopter simultaneously with the aforementioned mapping approaches, searches also for the humanoid robot. Since the ground robot is located, the quadcopter can calculate its pose within the map frame, by using an AR Marker and by referencing the ground robot position with respect to itself. In addition, a dedicated node is implemented for the quadcopter to be able to transform its LiDAR spatial data to the

most recently acquired ground robot position. Given this, the ground robot can percept the world, through quadcopter "vision" system, and thus to perform its localization in the current map. The ground robot localization method is performed by an adaptive Monte Carlo particle filtering method.

Finally, the humanoid robot uses a footstep planner to obtain the optimal path to a given map location. Through its navigation, a neural network, namely Darknet, is used as its weights are pre-trained on human imagery dataset.

This collaborative approach is designed to make the best use of the platform specifications, with respect to their computational capabilities and onboard sensory. The entire project has been implemented within the Robot Operating System (ROS) and is supported from any ground-aerial multi-robot systems, as long as proper adjustments and sensors are acquired.

1.2 Thesis Outline

In Chapter 2 we present all the background information needed for this thesis. We give an overview of the possible requirements that a Search and Rescue scenario has and how it can be approached by a multi-robot cooperation system. Specifically, we explain the usability of the ROS middleware in robots' communication and cooperation, and provide basic information about the specifications of the selected robots for this scenario. Moreover, there will be sufficient sensor model explanation of each sensor, and there will be references to basic knowledge about robot localization and mapping in those kind of scenarios. In Chapter 3 we state our cooperative multi-robot approach, referencing also other single-robot or multi-robot approaches that have been implemented for navigation, exploration and mapping scenarios. In Chapter 4, we describe in detail the implementation steps of the proposed cooperation method, starting with the establishment of robot communication and finishing the localization of human victims in the unknown environment. Throughout this Chapter, the performance of implemented steps is presented, as we validate the entire system with real-world experiments. This work is concluded in Chapter 5, in which future plans for extending our approach are presented.

Chapter 2

Background

At the beginning of this Chapter, we will describe briefly the technical and software specifications of the used robot platforms and how they interact through the ROS middleware. In addition, basic knowledge will be provided about Localization and Mapping problems, from a probabilistic approach, and specific discussion will be made on the sensory measurement models. Also, a detailed description will be given on camera parameters extraction, through the calibration process, to be able to test and validate stereo correspondence algorithms and 3D map reconstructions, in our approach in Chapter 4.

2.1 Robots

2.1.1 Aldebaran Nao Humanoid Robot

Aldebaran Nao is a programmable, medium-sized humanoid robot that is designed and manufactured by a French robotics company, the Aldebaran Robotics[1]. The first product release was in 2004 and since then there have been six version upgrades. Nao robot became popular mostly from the RoboCup soccer competition, the Standard Platform League (Figure 2.1), in which research robotic teams participated and programmed Nao robots to play 5 on 5 soccer games autonomously.

[1] https://www.softbankrobotics.com/

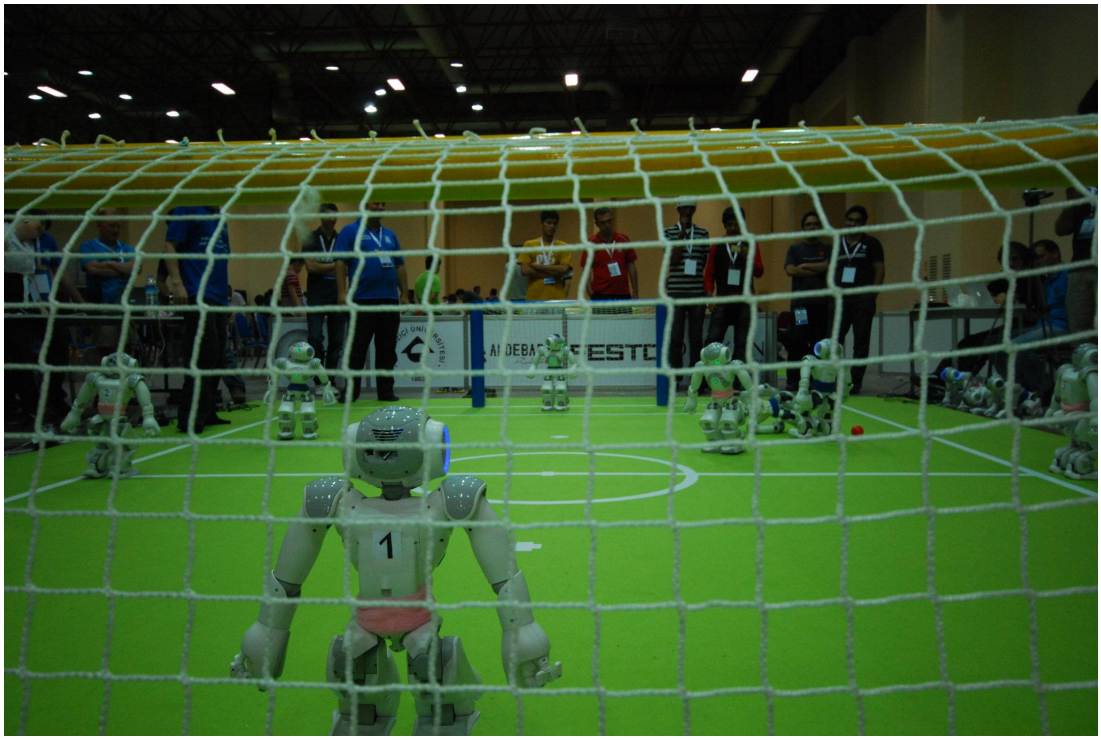
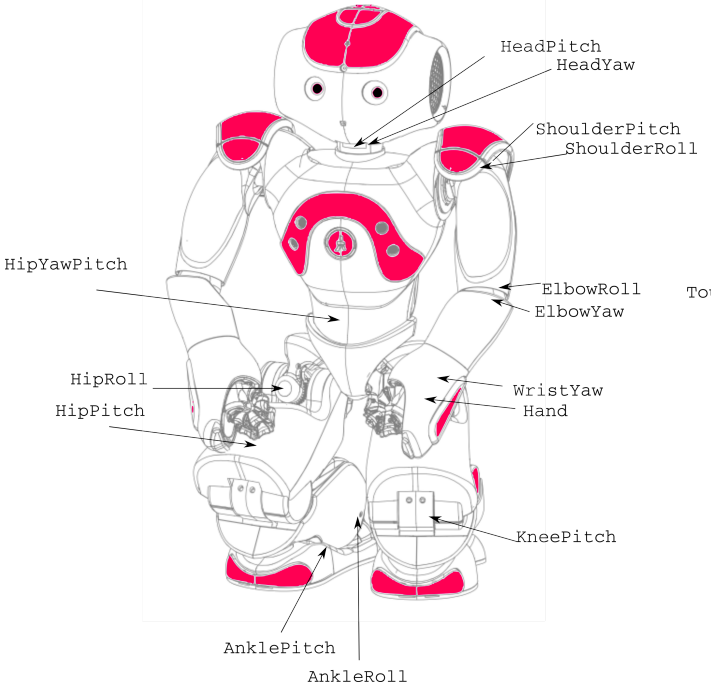


Figure 2.1: RoboCup SPL 2011 *(Image from Kouretes team archive)*[1]

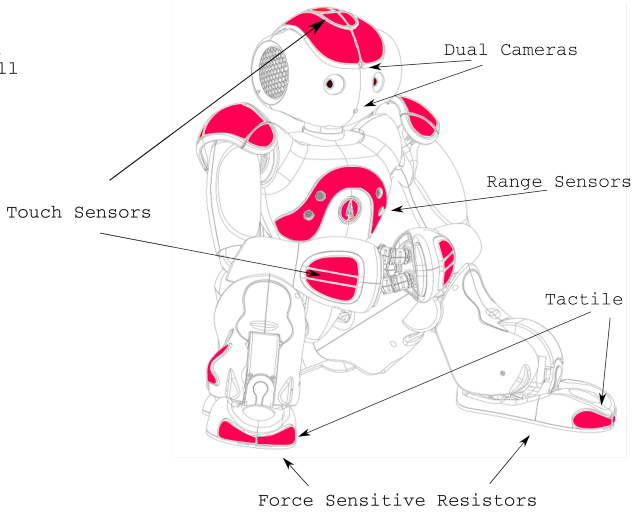
On the technical side, Nao V5 has a build in ATOM Z530 processor at 1.6 GHz clock and 1 GB of RAM, which makes it capable of completing simple tasks and also 2GB of flash memory with an Embedded GNU/Linux distribution. It is powered by a 6-cell li-ion battery and has a IEEE 802.11g network card by which can communicate wirelessly or wired with RJ-45 Ethernet connection.

In addition, Nao is consisted of a variety of actuators and sensors. Specifically, it has 26 motors in his frame, which provides 25 degrees of freedom. There are 2 in the head, 5 in each arm, 5 in each leg, one in each hand and one in the pelvis(on which there are two joints which are coupled in one servo and cannot move independently). Each joint comes with a magnetic rotary encoder, which provides pose information about the specific joint in precision of 12 bits per 0.1^o angle. On the other side, the sensors that Nao is equipped , are, tactile sensors, touch sensors, ultrasonic sensors, force sensitive resistors, cameras and an inertial unit system. Figures 2.2a and 2.2b shows the motors' and sensors' position, respectively.

[1] http://www.intelligence.tuc.gr/kouretes/web/index.php/el.html



(a) Nao Motor units



(b) Nao Sensor units

Figure 2.2: Aldebaran Nao robot

On the software side, as previously mentioned the Aldebaran Nao has an embedded computer system which runs a Gentoo-alike GNU/Linux distribution. The basic framework that runs on the Nao operating system and controls the robot, is named NAOqi[1]. The NAOqi framework is a cross-platform, supports cross-language and also provides introspection, which means the framework knows which functions are available in the different modules and where. It supports parallelism, resources, synchronization and events, that the robot need to perform any programmed task. NAOqi, also, allows homogeneous communication between different modules (motion, audio, video), homogeneous programming, and homogeneous information sharing.

To conclude, NAOqi supports software development in C++ and Python and has a fully developed ROS driver, which was made by Freiburg's Humanoid Robots Lab and Armin Hornung[2]. Through this driver, all Aldebaran's NaoQI API parts are wrapped and makes it available for use in ROS.

[1] http://doc.aldebaran.com/2-5/index_dev_guide.html
[2] https://github.com/ros-naoqi

2.1.2 DJI Matrice 100 (M100)

Matrice 100 is a quadcopter, that is designed and manufactured by DJI[1] company, mainly for developers. It's a fully programmable drone, as it hosts its own SDK (the DJI SDK) and has an onboard processing unit for programming, the *Manifold* (shown in Figure 2.3).

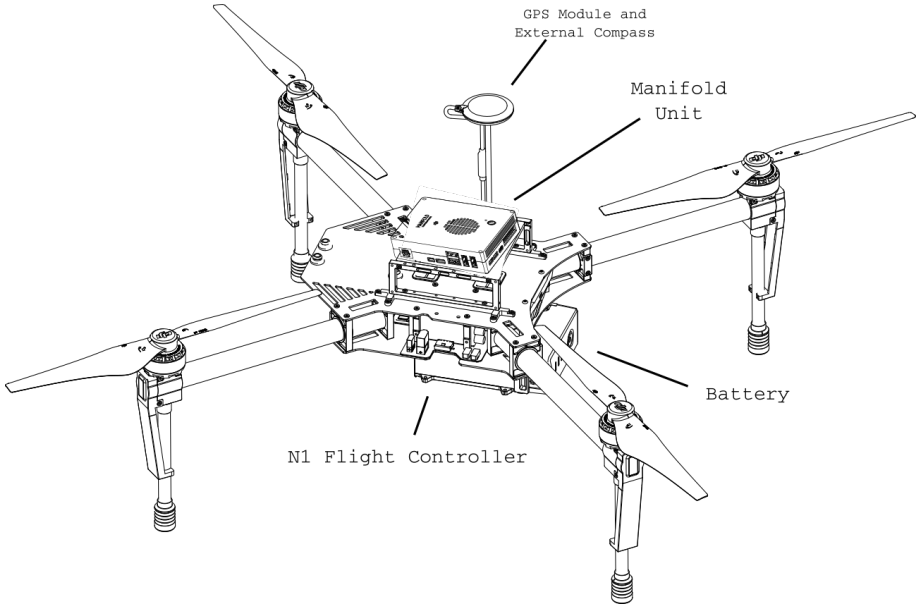


Figure 2.3: DJI Matrice M100 quadcopter sketch

In specific, the *Manifold* unit is an embedded computer especially designed for the DJI onboard SDK. *Manifold* packs a Quad-core ARM processor along with a low-power NVIDIA Kepler-based GeForce graphics processor, so its capable of completing any simple or more demanding task, that it will be assigned to. Also, it has a built-in special version of Ubuntu GNU/Linux operating system, which is developed to have fully DJI SDK accessibility and control. The *Manifold* can natively run the DJI Onboard SDK and have real-time access to flight data, raw sensor data and perform any behavior that is programmed to follow. Equipped with Ethernet, Mini-PCIe, UART, SPI, I2C and USB ports, the *Manifold* allows connections to a wide array of sensors, depending on the scenario is going to follow.

[1] https://www.dji.com/

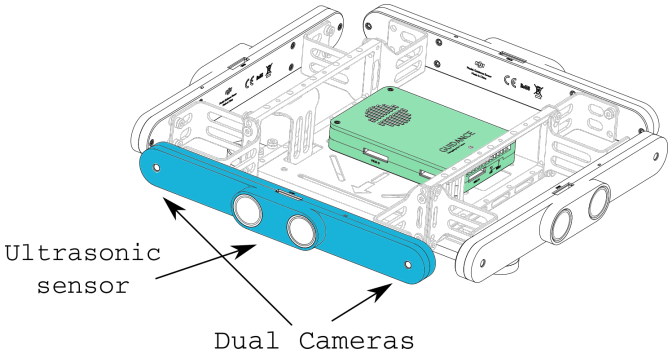


Figure 2.4: DJI *Guidance* system

In our approach, we will use M100 equipped with a LiDAR sensor and the DJI *Guidance* system (which is illustrated in Figure 2.4). The entire system is shown in Figure 2.5.

The *Guidance* system consists on 5 similar sensor units (showing one of all with blue color in Figure 2.4) and each one is equipped with an ultrasonic sensor and a pair of visual cameras (stereo camera system) giving 320x240 grayscale images. Those 5 sensor units are positioned towards to each direction, except the upwards. Also, in *Guidance* core (appeared with green color) there is an IMU built-in and a high-precision visual odometer, by which can measure the velocity of the sensor, to an accuracy of within centimeters.

To summarize, M100 quadcopter makes a great solution for our search and rescue scenario in an unknown area, as with its onboard processing capabilities and sensor data, can provide a sufficient belief about the viewing environment. Both DJI onboard SDK and *Guidance* sensor supports ROS[1,2], making M100 quadcopter ideal for our ROS cooperative scenario setup.

[1] https://github.com/dji-sdk/Onboard-SDK-ROS
[2] https://github.com/dji-sdk/Guidance-SDK-ROS

Figure 2.5: The DJI M100 quadcopter equipped with the *Manifold* unit, a LiDAR sensor and the *Guidance* system.

2.2 Robot Operation System (ROS)

The Robot Operation System, namely ROS[1] [1], is a framework developed by the *Willow Garage*[2] robotics research team, in order to facilitate the design, development and maintenance of robotic applications. Specifically, it is a middleware that allows different devices "coexist" in the ROS environment and interact with each other, through a peer-to-peer network. Each device in this environment represents a process/node and it is able communicate with an another one, through ROS communication infrastructure. This communication can be established via,

- ROS *Topics* publication or subscription. A ROS *Topic* is an asynchronous way of node to stream data. Also, a node in order to receive the data from a topic, it

[1] http://www.ros.org/
[2] http://www.willowgarage.com/

has to subscribe on the corresponding ROS topic. Any node can post any amount of topics and subscribe also. Moreover, more than one nodes can subscribe on the same topic or even more publish on the same one. In general, publishers and subscribers are not aware of each others' existence.

- ROS *Services*. In contrast with topics, in case of a "Remote Procedure Call" (RPC) request/reply interaction, there is a need of two-way communication. Especially, given a node with a providing service, a client firstly calls the service by sending a request and awaits service node's reply, after the service initialization. However in services' case, only one node can advertise a service with a certain name.

- Via *Parameter Server*. This way of exchanging data is used for sharing static data or non-binary data, like configuration files. Basically is a shared, multi-variate dictionary that is accessible from all nodes to store and retrieve data from.

For example, a ROS environment could be a single humanoid robot that publishes on topics the information about all its sensors and joints' poses. It could also, publish a topic of the localization process that executes on its embedded processor. Furthermore, it could provide two services, like setting the camera parameters or resetting the belief of its position, in which an another computer or robot on the same ROS environment could call. Also, there could be a scenario including of two connected robots in the same ROS environment, which through communication and cooperation will explore an unknown area. In this scenario, those robots can share information through their published topics about their state and belief and by the data fusion they can cooperate and navigate in the unknown environment and accomplish certain tasks.

In addition, ROS is using a language-neutral interface definition language (IDL) to describe the *messages* that are published through topics. Basically, a ROS *message* represents the value of a single data that is posted from a ROS node at that time, like a sensor measurement, a joint state, a diagnostic ROS message or more. Therefore it consists on unique fields that describe it and the IDL uses short text files to compose them.

A sensor_msgs/Range ROS Message

uint8 $ULTRASOUND = 0$
uint8 $INFRARED = 1$

std_msgs/Header *header*
    uint32 *seq*
    time *stamp*
    string *frame_id*
uint8 *radiation_type*
float32 *field_of_view*
float32 *min_range*
float32 *max_range*
float32 *range*

The left appeared box shows the structure of a Range ROS message, that a range sensor node can publish. Every range message comes with *header* parameter that includes the *time stamp* and *frame* information, which identifies it uniquely in time and gives information of the frame that was it taken in respect to.

Also parameters as *min_range*, *max_range*, *field_of_view* and *radiation_type* define the specifications of the range sensor and the measured value and the parameter *range* gives the distance reading in the current time.

Furthermore, as ROS framework initially have been developed to be used in complex and large-scale robot applications, it utilizes also a variety of tools that help to manage its complexity. The tools that come along with ROS infrastructure consist on ROS environment real time configuration and monitoring, package logging and debugging, data visualization tools (like RViz [2]) and the transformation system *tf* [3].

To conclude, ROS framework usability, wide-range of capabilities and its open-source philosophy are some of the reasons for choosing it as the basis framework of our robotic scenario. Also, as it was mentioned before, ROS is fully supported on the selected robots (2.1.1 and 2.1.2), making it ideal for the application development. In particular, our vision is to create a generic collaboration package of an aerial and ground robot, to cooperate, map and locate individuals in an unknown area. Through ROS framework, this project can be applied in every dual robot (aerial-ground) scenarios, since the robots have the specified sensors and have made the appropriate modifications.

2.3 Localization and Mapping

2.3.1 Mobile Robot Localization

Mobile robot localization is the problem of determining the pose of a robot in a known environment-map. During a navigation scenario the robot should be able to locate itself in the exploring area, namely establish the correspondence of its local coordinate system to the global (map coordinate system), in order to express the location of any surrounding objects or even obstacles in a relation to it. The localization problem can be solved by exploiting robot sensors data, either from motion or the perception. Motion sensors provide information about the robot movements-displacements (known as odometry) and the perception sensors provide spatial information relative to the robot position.

The localization problem can be classified in three categories, depending on the type of knowledge that is available from the first moment of the navigation process to the run-time. Also, due to high uncertainty in motion and perception, a probabilistic approach is needed to formulate the localization problem and often is expressed as an Bayesian estimation problem. In specific, the localization categories are,

- **Position tracking**. In position tracking, we assume that the initial robot pose is known and we solve the localization problem through the odometry information. By compensating the robot's motion model uncertainty, we update the local robot pose belief in relation with the initial pose. Hence, this method rely on the assumption that the robot's motion model error is small and can be approximated.

- **Global localization**. The problem of the robot unawareness of its initial position in a known map and the need of locating itself in it, is known as the global localization problem. The robot must exploit the data from it sensors to infer its location in the known map and then proceed with the position tracking approach.

- **Kidnapped robot problem**. The kidnapped robot problem, as an extension of the global localization problem, assumes that, as the robot navigates and "knows" its location it is teleported or kidnapped to a different location than it was, without getting noticed. The main goal of the solution in this problem, is to make the robot realize that it was kidnapped, in order to initialize its belief about its pose in the map. This problem is more difficult than the global localization problem, as the robot might believe that it know where it is, but it does not. The practical

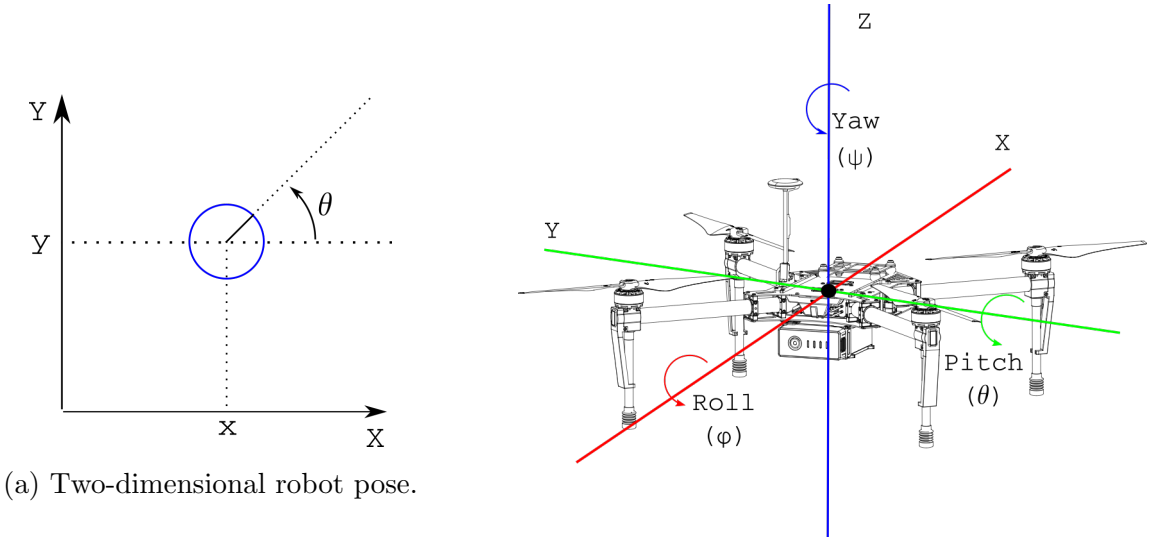
importance of this problem is to benchmark the awareness and the adaptivity of the localization algorithm to global localization failures.

Robot Pose

The configuration space is defined as the state space of a robotic system, which is described by its position, orientation and joint angles. In path planning scenarios, we search for a path that transits from the current configuration state to a desired one. In three-dimensional space, the configuration of a rigid robotic system is described by six variables, the three-dimensional Cartesian and the three Euler angles, describing the translation and rotation of this body relative to an external coordinate frame respectively. Likewise, in planar environments, the kinematic state of a robot is described by three variables, the two translational in each dimension and one that defines the angular orientation. The state (pose) of the mobile robot at the time step k, in each case is described as,

$$\mathbb{R}^3 : x_k = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} and\, \theta_k = \begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix} \tag{2.1}$$

$$\mathbb{R}^2 : x_k = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{2.2}$$



(a) Two-dimensional robot pose.

(b) Three-dimensional robot pose.

Figure 2.6: Robot poses in 2D and 3D global coordinate systems.

Motion Model

Mobile robots are equipped with actuators (servos, hydraulic drives, motors and more) and use them to move and navigate, in the exploring area. However, the actuator usage can be noisy and may result to a slightly different motion than the purposed one. This introduces error to the *motion model* which is added overtime to the odometry information, affecting in this way the localization validity. To avoid this, mobile robots often are equipped with extra sensors to measure the occurred deviation. The information about the action that actually took place (locomotion action) in any step time is available after the action is occurred. Thus, given the current *state* x_k and the action u_k (also known as *control* input), the robot can make the transition to a new *state* $x_{k'}$. The probabilistic model of this transition, is named *motion model* and is formed as,

$$P(x_k|x_{k-1}, u_k) \tag{2.3}$$

This model describes the posterior distribution of the robot pose x_k in time step k, given its previous pose x_{k-1} by performing the action u_k. The *motion model* adopts the Markov property of the memoryless stochastic process, as it depends only on the previous robot state and the current action.

Sensor Model

A sufficient localization process can't be implemented without having any sensor measurements from the robot's surrounding environment. In particular, given a robot without any external sensor and based only on the *motion model*, the problem of global localization, as well the kidnapped robot problem, can not be apriori solved. So, the mobile robot must be equipped with sensors, like a range finder, a camera or even a global position system, to be able to locate features in its environment and estimate its position into a generated map.

In addition, despite the sensors' importance in localization, sensors are also subject to noise. Therefore, the noise in robot's sensor measurements can be described in probabilistic terms, as a conditional probability distribution,

$$P(z_k/x_k) \tag{2.4}$$

The resulted distribution (2.4), is called the *sensor model* and describes the likelihood of making the z_k measurement in k step, given the pose x_k at the same step. Similarly, this model adopts the Markov property, as *motion model* in (2.3).

2.3.2 Occupancy Grid Mapping

In *Localization* Section [2.3.1], we assumed that the robot's exploring environment is known, as the localization is performed in reference to the known map. Instead, in most explorations cases and especially in Search and Rescue cases, the searching area is unknown or at least, in case of building, may differs from the given building's blueprints.

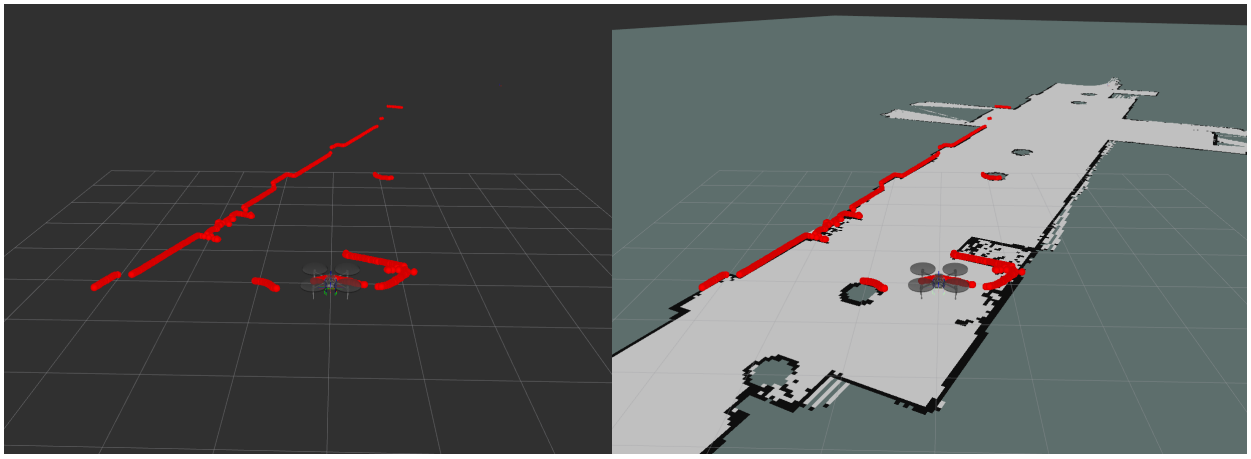
Thus, the ability of a robot to do the mapping in real-time during its navigation, can be a huge advantage under exploration situations, as the robot can always be informed for the current state of its environment and can adapt to changes without human supervision. Although, the mapping process can be more acquiring than the localization process, because of the,

- **Hypothesis space**. The hypothesis space, of all generated maps, becomes huge as the size and resolution of the map increases. The huge size of the hypothesis space makes approaches like Bayes filtering inapplicable, because it's infeasible to calculate the full posteriors over maps.

- **SLAM problem**. As the robot navigates in the environment, it accumulates errors in odometry, and thus inserts more uncertainty to the localization process. At the same time, the robot must construct the map by combining the previous pose states, and thus perform the mapping . The *simultaneous localization and mapping* (SLAM) approach solves those two problems simultaneously and will be discussed in depth in the following Subsection [2.3.3].

- **Perceptual ambiguity**. Many places in an unknown area can be spatially similar to other, as they appear in sensor measurements (observed features). This event causes false positives in correspondences extraction and inserts error in the mapping process.

- **Closing loops**. Closing cycles can be of the most common problem in mapping process. If a robot visits a previously mapped area through a different path, the accumulated odometric error can be huge, and thus affect the mapping accuracy.



(a) Visualized laser scan beam endpoints (b) Occupancy Grid map

Figure 2.7: Occupancy Grid Mapping in Mineral Resources Engineering building in the Technical University of Crete.

In particular, an occupancy grid map is an evenly, in space, gridded map, in which every grid value define the occupancy of the current grid block. Those values can be considered also as random variables and there are algorithms that implement their approximate posterior estimation. Figure 2.7b shows the occupancy grid map that is created

from sequential laser range finder data using the SLAM algorithm. The gray-scale color of each grid cell corresponds to the likelihood of occupancy. The black-colored grid cells represents high occupation probability and the white ones high certainty in free space. Also, gray-colored grid cells represent the prior.

2.3.3 Simultaneous Localization and Mapping (SLAM)

The simultaneous localization and mapping (SLAM) problem is defined as the situation of a mobile robot, that neither its pose or the map is known. This is the most common scenario that can occur, in which a robot starts to explore an unknown area. Specifically, the only information that the robot has is, its actions $u_{1:k}$ and measurements $z_{1:k}$. By solving the SLAM problem, the mobile robot can perform the mapping process the same time as it localizes itself in the generated map, and thus can navigate to a totally unknown environment.

The SLAM problem can be divided in two forms, the *online* SLAM and the $full$ SLAM. Particularly,

- The *online* SLAM deals with the estimation of the posterior over the momentary pose along with the map, the actions and measurements that have been made since the start,
$$p(x_k, m|z_{1:k}, u_{1:k}) \tag{2.5}$$
- The $full$ SLAM seeks the posterior over the entire path of the mobile robot, instead of its current position,
$$p(x_{1:k}, m|z_{1:k}, u_{1:k}) \tag{2.6}$$

Furthermore, the SLAM problem has to do with two estimation problems, a continuous and a discrete. The continuous component has to do with the estimation of the observation location in the map and the robot's own pose variables. On the other side, the discrete estimation component has to do with the observations correspondence. In specific, it can be described as a discrete problem in which we assume if the observation have previously made or not. The goal to solve the SLAM problem, is to estimate the full posterior (2.5),(2.6). By estimating the full posterior, we have all the information about the map, the mobile robot pose and it's trajectory path. However, this goal is

usually infeasible, due to previously referenced problems, so the SLAM algorithms rely on approximations methods.

2.4 Sensor Measurement Models

As mentioned before, robots are equipped with a plethora of sensors that help them percept the world around them, in order to modify their operations accordingly and be able to perform more complex tasks. In this section, the sensor measurement models of the laser range scanner and the visual cameras' will be described in detail, as they are going to used in the localization and mapping processes. In particular for the camera part, there will be a full description in basic knowledge on monocular vision system and how we "transition" in stereo vision system. Also, information about camera calibration procedures will be given, separately.

2.4.1 LiDAR Sensor Measurement Model

LiDAR system, which stands for Light Detection and Ranging system, is a 3D laser-scanning device that can provide distance information on the scanning area. Nowadays, these sensors are used on many mobile platforms (autonomous cars, satellites, airplanes and more) for different scenarios, in which is necessary to determine the absolute position and orientation of the mobile platform in relation to the generated ranging data, for a certain purpose.

Specifically, a LiDAR system emits rapid pulses of laser light in a scanning range, captures the reflected laser light back to its receiver and stores the reflection time. Consequently, the distance d of the laser scanner sensor from the reflected surface can be found from,

$$d = \frac{\textit{Speed of light * Time of Flight}}{2}$$

In most cases, LiDAR sensors are two-dimensional, meaning it could only scan a single horizontal slice of the world. Also, they are wide angle (scanning range above 180^o with a fixed angular resolution, so they can provide almost surrounding spatial information. Of course, there are LiDAR sensors which may have more than one laser-scanner systems and they can measure distances in three dimensions in the world space, not only in single plane. Those sensors generates data in a point cloud [4] form.

(a) The LiDAR Hokuyo UTM-30LX-EW

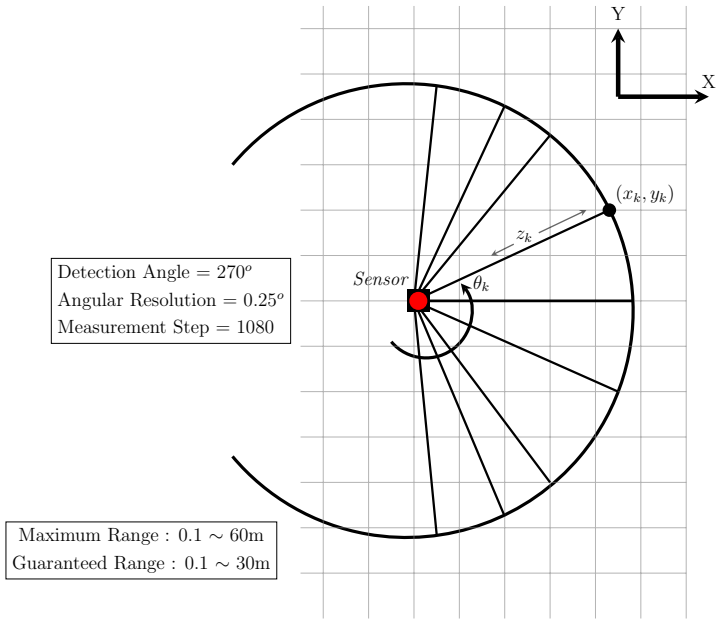


Figure 2.9: Hokuyo UTM-30LX-EW diagram of scanned area and specifications.

For our scenario, we use a Hokuyo laser scanner sensor (appeared on Figure 2.8a),

which provides 1080 range measurements in a full scan (270^o scanning angle with 0.25^o of angular resolution; see Figure 2.9). Specifically, the scanning plane of this LiDAR is parallel with the XY plane and provides raw distance measurements z_k ,θ_k in polar coordinate system. Those measurements can be expressed in Cartesian coordinate system as,

$$\begin{pmatrix} x_k \\ y_k \end{pmatrix} = \begin{pmatrix} z_k^t \times cos\theta_k \\ z_k^t \times sin\theta_k \end{pmatrix} \tag{2.7}$$

where z_k^t the kth-measured end point at kth-scanned angle θ_k, in step time t (Figure 2.9). In the above equation (2.7), all measurements have zero z-axis values because they lie on the same plane.

So, let $x_t = (x, y, \theta)$ be the robot pose at time t. Also, we denote the relative location of the LiDAR sensor in the robot's local coordinate system by $(x_{k,sens}, y_{k,sens})$ and the angular orientation of the sensor beam relative to the robot heading direction as $\theta_{k,sens}$. In global coordinates, the LiDAR measurements z_k^t can be expressed through the following trigonometric transformation, as,

$$\begin{pmatrix} x_{z_k^t} \\ y_{z_k^t} \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + R(\theta) \begin{pmatrix} x_{k,sens} \\ y_{k,sens} \end{pmatrix} + z_k^t \begin{pmatrix} cos(\theta + \theta_{k,sens}) \\ sin(\theta + \theta_{k,sens}) \end{pmatrix} \tag{2.8}$$

where $R(\theta)$ the rotation matrix, to align the LiDAR sensor with the the robot's heading direction.

However, LiDAR scanning process comes with noise, which must be included in the measurement model (2.7) . The basic types of source of noise, are,

- **Measurement noise.**. This category has to do with measurement miscalculations. The measurement noise can be modeled with a zero-centered Gaussian of the measurement point and the closest to it map obstacle point. Given the LiDAR measurement coordinates $(x_{z_k^t}, y_{z_k^t})^T$ and the nearest object in map m, the sensor noise will form as,

$$p_{hit}(z_k^t | x_t, m) = \epsilon_{\sigma_{hit}}(dist) \tag{2.9}$$

  where parameter $dist$, is the Euclidean distance of the measurement $(x_{z_k^t}, y_{z_k^t})^T$ and the closest to it obstacle point.

- **Sensor failures**. The used laser-scanner system (Figure 2.8a) provide its raw measurements z_k^t in millimeters. In case of a measurement that is smaller than 23mm, there was a measurement error occurred, probably due to light interference or noise. Similarly, if the range measurement is above 60m, there was either no obstacle detected or the obstacle's surface had low reflectivity. Those cases are discarded from the measurement model, through a discrete probability distribution in which,

$$p_{fail}(z_k^t|x_t, m) = \left\{ \begin{array}{c} 1 \text{ , if z<23mm or z>60m} \\ 0 \text{ , else} \end{array} \right\} \tag{2.10}$$

- **Unexplained random measurements.** A uniform distribution of random noise in measurements. This random noise distribution will be expressed with p_{rand}.

So, the final algorithm for computing the likelihood of a range finder scan using the Euclidean distance of the closest point to the measurement and including the referenced noise models of the sensor failures and random measurements, is presented in table 1 below.

Algorithm 1 Likelihood Field Range Finder Model

```
q=0
for all k do
    if z_k^t ≠ z_max then
        x_z^k = x + x_{k,sens} cosθ − y_{k,sens} sinθ + z_t^k cos(θ + θ_{k,sens})
        y_z^k = y + y_{k,sens} cosθ + x_{k,sens} sinθ + z_t^k sin(θ + θ_{k,sens})
        dist^2 = min_{x',y'} { sqrt((x_{z_t^k} − x')^2 + (y_{z_t^k} − y')^2) | ⟨x', y'⟩ occupied in m }
        q = q · ( z_hit · prob(dist, σ_hit + z_rand / z_fail) )
    end if
end for
return q
```

The function **prob()** computes the probability of *dist* under a zero-centered Gaussian distribution, which is characterized by standard deviation σ_{hit}.

2.4.2 Pinhole Camera Measurement Model

The Pinhole camera model is basically a camera model without a lens and with a single tiny hole, called the aperture. As its simplicity and the lack of lenses, this camera allows only the correctly aligned light rays pass through its aperture and "projects" them as an inverted image on the opposite surface of the camera, on the projective plane.

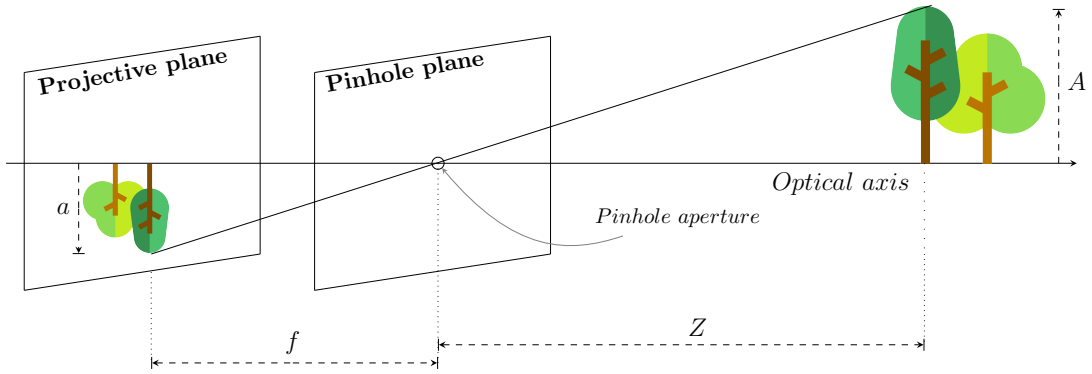


Figure 2.10: The Pinhole Camera Model

As Figure 2.10 shows, the size of the projected image relative to the distant object depends on variable f, the camera's focal length. The focal length of a pinhole camera, is the distance from the pinhole aperture to its projective plane. From the figure we can conclude by similar triangles theorem, that,

$$\frac{-a}{A} = \frac{f}{Z} \Leftrightarrow -a = f\frac{A}{Z} \tag{2.11}$$

Camera Matrix

Likewise, a equivalent pinhole camera model of Figure 2.10 appears in Figure 2.11, in which the projective plane is replaced with the pinhole plane and vice versa. This figure is equal to the first one, but simplifies the math computations, as the projected object appears upright. The basic idea is that, the image plane represents a slice of all those light rays that end up passing through the pinhole aperture.

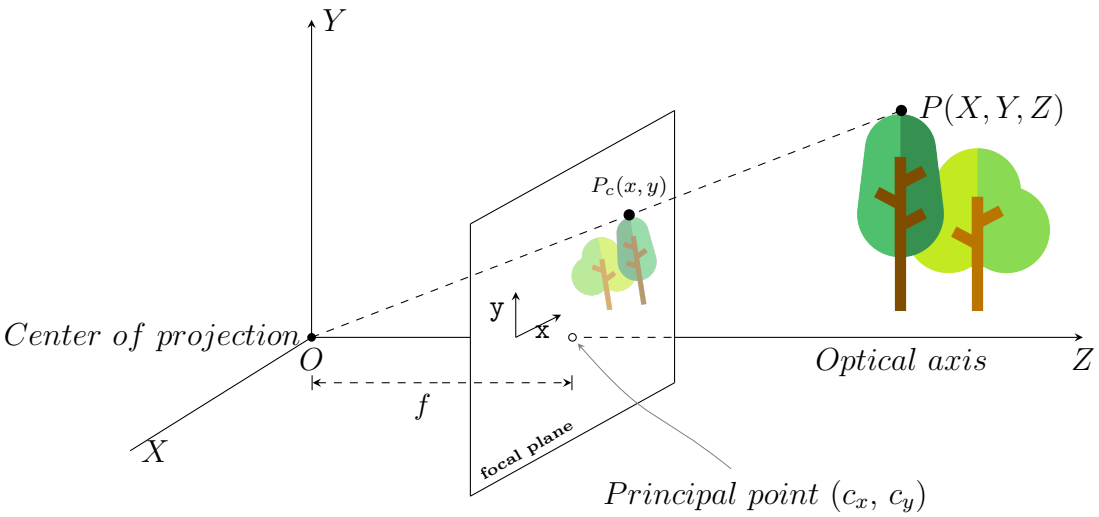


Figure 2.11: Rearranged Pinhole Camera Model

As the new image plane (focal plane) is f distance away from the center of projection, by the similar triangles theorem it is true that,

$$\frac{x}{X} = \frac{y}{Y} = \frac{f}{Z} \Leftrightarrow \quad x = f\frac{X}{Z} \quad and \quad y = f\frac{Y}{Z} \tag{2.12}$$

The equations in (2.12) are expressed in metric units. Although, the unit of the image coordinates is pixels, thus there is a need to define two parameters to make the transform from metric to pixels unit system. Specifically, we define two parameters each one for each axis, D_x and D_y respectively, because cameras could have different pixel resolution on each axis. In addition, the above assumptions were made given that the principle point is equivalent to the origin of the 2D image coordinate system. This happens when the center of the camera's chip is perfectly aligned with the optical axis. Hence, we define two parameters, c_x and c_y, to include the 2D translation-displacement of the center of the coordinate system from the optical axis, in pixels unit.

So, the corresponding image coordinates, from the projection of the 3D world point on the projective plane, are,

$$\mathtt{x} = D_x * f\frac{X}{Z} + c_x \quad and \quad \mathtt{y} = D_y * f\frac{Y}{Z} + c_y \tag{2.13}$$

It is mandatory to define two different focal lengths in equations (2.13), each for every axis on the image plane. Therefore, by using homogeneous coordinates, we can express

P_c as,

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = \begin{pmatrix} D_x f_x & 0 & c_x \\ 0 & D_y f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \mathrm{K}P \tag{2.14}$$

In equation (2.14) the P_c is expressed in homogenous coordinates, so to extract its real projected position (`x,y`) on the image plane, we must divide x,y with w. The matrix K also includes the skew parameter and the scale factor, $alpha_c$ and s_x. The skew parameter is non-zero when the image axes are not perpendicular (centering imperfection). In this approach (which is derived from Heikkila and Silver [5]), we assume that the skew parameter is zero and the scale factor is one.

$$\mathrm{K} = \begin{pmatrix} D_x s_x f_x & alpha_c & c_x \\ 0 & D_y f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.15}$$

The matrix K (2.15) is known as the *Camera Intrinsics* [5] matrix.

Lens and Distortion Matrix

At the same time, as we mentioned before, pinhole camera has no lens in its structure. Therefore, the amount of light rays that passes through the pinhole aperture is small in a period of time. This fact causes time delays in scene imaging, because of lesser light rays gathering, which implies time delay in total scene projection. To solve this problem, we use a lens to make more light rays accumulate on our sensor, over wider area and focus that light to converge the point of projection. Nowadays, lenses are used in every camera and come in different specifications of aperture and focal length, depending on the scenario they are going to be used. However, lenses may introduce some aberrations, known as distortions, in our 3D to 2D scene projection, because of their manufacturer defects or their displacement comparative with the camera sensor. Those distortion coefficients must be included in our camera model, to correct the mapped 2D `x,y` (2.13) pixel coordinates.

The most common distortion effects are the radial distortion and the tangential distortion.

Radial Distortion Radial distortion is caused when a lens deflects more the light rays that are farther away from its center, than those who are. This distortion can be found mostly on spherical lens and it causes an inward or outward displacement of an image plane point `(x,y)` (2.13) from its real position. The distortion is equal to zero at the optical center of the projective plane of the camera and increases while it moves to the peripheral.

Also, it can be classified in two categories, the barrel and the pincushion distortion (see Figure 2.12), in which the first distortion appears in wide-angle lens while the second one appears low-end telephoto lens.

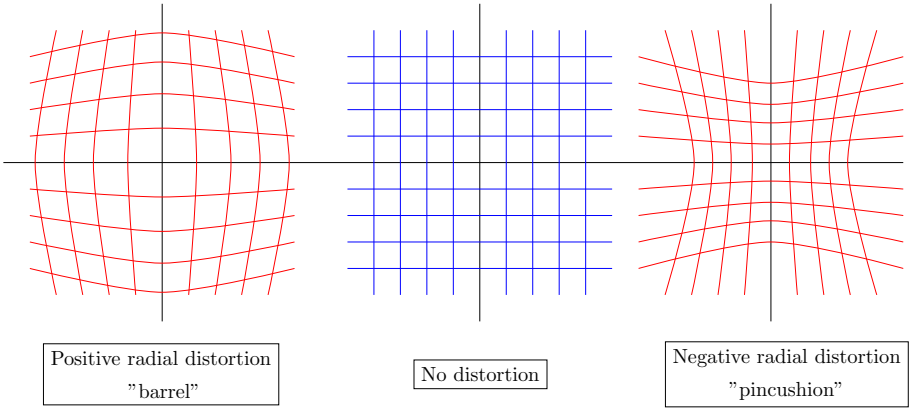


Figure 2.12: Radial Distortion categories

For this reason, to correct this kind of distortion, we apply a simple polynomial transformation, which is affected symmetrically by the distance from the optical center. Specifically, this distortion model can be described by the first few terms of a Taylor series expansion around $r = 0$, in which r is the radial distance and the distortion function becomes a polynomial in the neighborhood of $r = 0$.

$$f(r) = a_0 + a_1 r + a_2 r^2 + a_3 r^3 + \dots \tag{2.16}$$

In (2.17), due to no distortion on the optical center, $f(0) = 0$, the a_0 is zero. Additionally, because of the distortion symmetric behavior around the center, every coefficient of odd power will be zero. So, the equation (2.17) will form as,

$$f(r) = a_2 r^2 + a_4 r^4 + a_6 r^6 + \dots \quad \text{where} \quad r = \sqrt{x^2 + y^2} \tag{2.17}$$

The polynomial parameters that will be necessary to model those radial distortions will belong to the first three coefficients, r^2,r^4 and r^6. Hence, the radial distortion can be defined using the following expressions for x,y,

$$\delta x^{(r)} = x(k_1 r^2 + k_2 r^4 + k_3 r^6) \quad \text{and} \quad \delta y^{(r)} = y(k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.18}$$

Finally, all the distorted points will be corrected by increasingly displaced inward as far as they are placed from the optical center.

Tangential Distortion Tangential distortion is caused when the lens is not parallel to the imaging plane of the camera. As you can see in Figure 2.13, image plane is projected and distorted, because of lens misalignment comparative with the CMOS chip.

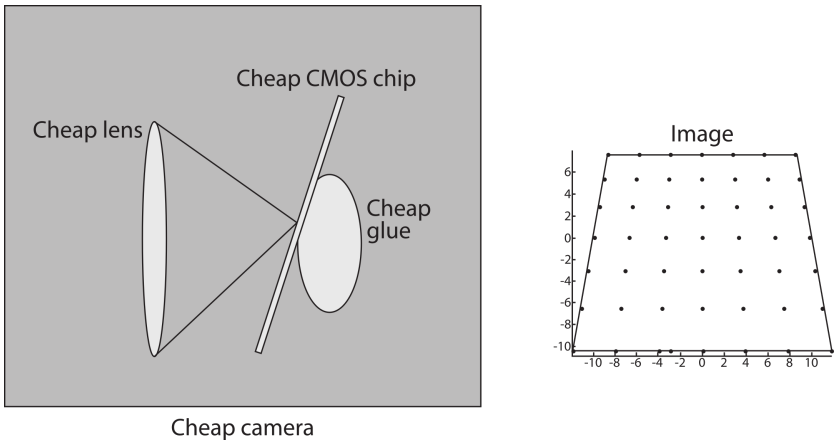


Figure 2.13: Tangential Distortion (*Image from Sebastian Thrun*)

This distortion is characterized by two parameters, p_1 and p_2. So the expression of this distortion for x and y, is often written in the following form,

$$\delta x^{(t)} = 2p_1 xy + p_2(r^2 + 2x^2) \quad \text{and} \quad \delta y^{(t)} = p_1(r^2 + 2y^2) + 2p_2 xy \tag{2.19}$$

To summarize, the proper camera model for an accurate calibration can be acquired by combining the pinhole model (2.13) with the correction of radial (2.18) and tangen-

tial (2.19) distortion components. So the undistorted-corrected pixel coordinates of a projected 3D object on the projective plane of the camera, are given by,

$$\mathtt{x} = D_x\Big(x + \delta x^{(r)} + \delta x^{(t)}\Big) + c_x \quad \text{and} \quad \mathtt{y} = D_y\Big(y + \delta y^{(r)} + \delta y^{(t)}\Big) + c_y \tag{2.20}$$

Also we define the matrix D, containing all distortion parameters from above, as,

$$\mathrm{D} = \big[k_1, k_2, p_1, p_2, k_3\big] \tag{2.21}$$

This matrix may include more distortion coefficients, if the distortion model differs from the above. The matrix D (2.21) is also known as the *Distortion* matrix.

The *Camera Intrinsics* matrix K (2.15) and the *Distortion* matrix D (2.21), are uniquely defined for each camera, because they depend only on its physical construction and specifications (not by the camera position in the viewed scene).

Extrinsic parameters

In Figure 2.11, the optical axis is total perpendicular to the image plane and the center of projection is at origin of the XYZ coordinate. In most cases, this assumption doesn't imply, as the camera coordinate system is slightly rotated and translated, compared to its ideal position as it appears in Figure 2.11. So we define two matrices R and T, to transform any coordinate of a point in (X,Y,Z) into the coordinate system which is fixed with respect to the camera. Let R be the 3x3 rotation matrix for the optical axis to coincide with the Z-axis and T be the 1x3 translation vector to measure the translation of camera's center of projection from the XYZ coordinate system origin.

So the rigid motion equation that relates a 3D world point to the camera's 3D coordinate system is,

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = R \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + T \text{ , where } R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{13} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \text{ and } T = \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \end{pmatrix} \tag{2.22}$$

Once the coordinates of a 3D world point is expressed in the camera reference frame, it may be projected on the image plane using the intrinsic camera parameters. So, the

perspective transformation of (2.14), will form as,

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = \mathrm{K} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{13} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \quad \Leftrightarrow \quad wP_c = \mathrm{K}[R|T]P' \quad , \text{where } P' = (X, Y, Z, 1). \tag{2.23}$$

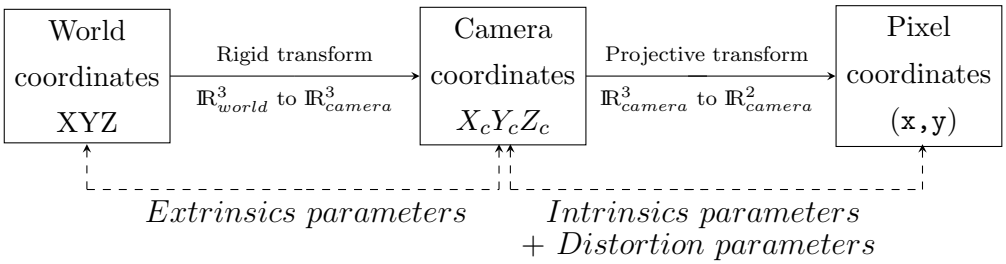


Figure 2.14: Total flow diagram of 3D world point projection on the image plane, in pixel coordinates.

In conclusion, Figure 2.14 shows the complete coordinate transformation of a world point (X,Y,Z) to be projected on the image plane of the camera.

Camera Calibration

Camera calibration is the process of estimating the intrinsic (2.15) and extrinsic (2.22) parameters of a camera. As we mentioned before in Section [2.4.2], the knowledge of those matrices is necessary for accurate mapping 3D world points on image plane pixel coordinate system as well as extracting metric information from the viewed scene. One of the most known calibration methods is published by Zhengyou Zhang's in [6]. According to this technique, the only requirement for a camera calibration is a planar pattern being shown in two or more camera photos. There is no need to know the motion of the camera around the planar pattern neither their relative positions, as long as the camera observes the whole pattern and there are at least two photos being taken with different viewing angles.

So, let M be the 3x4 projection matrix from (2.23),

$$M = \mathrm{K}[R|T] \quad and \quad (2.23) \Leftrightarrow \begin{pmatrix} x \\ y \\ w \end{pmatrix} = M \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.24}$$

Additionally, it is known that *planar homography* is defined as the projective mapping from one plane to an another, up to a scale factor. If we assume that,

$$q = \begin{pmatrix} \mathtt{x} \\ \mathtt{y} \\ 1 \end{pmatrix} \quad and \quad Q = \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.25}$$

we can express their homography as,

$$q = sHQ \;\text{ as it is true that }\; p_{dst} = Hp_{src} \text{ with } p_{src} = \begin{pmatrix} x_{src} \\ y_{src} \\ 1 \end{pmatrix} \; and \; \begin{pmatrix} x_{dst} \\ y_{dst} \\ 1 \end{pmatrix} \tag{2.26}$$

The homography matrix H is defined as , H=$\mathtt{K}[R\ t]$. Its definition is derived in two parts; the projection (the camera's intrinsics matrix $\mathtt{K}$) and the physical transformation (the information about the viewed plane transformation to the image plane $[R\ t]$). Also we notice that, during the calibration process we care for the Q position on the pattern plane than its real world position. Therefore, we can assume that the object plane is characterized by Z=0, without any loss of generality in calibration procedure. Eventually,

$$\begin{pmatrix} \mathtt{x} \\ \mathtt{y} \\ 1 \end{pmatrix} = sH \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} = s\mathtt{K} \begin{bmatrix} R & t \end{bmatrix} \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} \tag{2.27}$$

By breaking R into three 3x1 columns,

$$(2.28) \Leftrightarrow \begin{pmatrix} \mathtt{x} \\ \mathtt{y} \\ 1 \end{pmatrix} = s\mathtt{K} \begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix} \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} = s\mathtt{K} \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \tag{2.28}$$

So the 3x3 matrix $\mathtt{K}[r_1\ r_2\ t]$ is the homography that maps the pattern planar calibration point (X,Y,0) to image coordinates `(x,y)` in pixel units. In addition, according to Zhang's [6] approach, to find the camera intrinsic and extrinsic parameters, starts with a closed-form solution followed by nonlinear optimization, assuming that there is no distortion applied. Specifically, we collect the homography of each view of the planar pattern and with preceding equations we generate a B 3x3 matrix that contains the info of the camera parameters and can be solved in closed-form. We must also point out that we need two or more images of the planar pattern because we need to fit the homography matrix to each view of the planar pattern (two views of 4 planar points is enough to solve the calibration problem - e.g a 3x3 chessboard pattern). Then, we estimate the distortion coefficients (mentioned in Section [2.4.2]) by stacking all the generated distortion equations of the projected points x,y and applying linear least-squares approximation. Finally, we refine all parameters by doing a complete maximum likelihood estimation, to minimize the image residual errors.

2.4.3 Stereo Cameras Measurement Model

Stereo Camera rigs are used on basic range imaging scenarios, where a 2D map is constructed showing distances around a specific point, and even more on scenarios that world scene depth estimation and 3D reconstruction is needed. This camera setup consist on two (or more) cameras that are fixed positioned from each other and are viewing the same scene. Their placement setup simulates the human binocular vision system as we use both our eyes to have a 3D perception of the world.

Specifically, when a 3D world point is projected on the 2D image plane of the camera loses its depth information. By using the camera matrix to do the reverse operation, we end up with a line in three dimensional space, on which the 3D point lies, but we have no information about its exact position. So this is the main reason that we use a second camera, that is viewing the point from a different angle and position, to extract its depth information by following some important steps that will be mentioned below.

i. **Distortion removal**. As we have to deal with common cameras, there may be radial or tangential lens distortion that should be removed. Lens distortion models and undistortion process is described in Subsection [2.4.2].

ii. **Stereo rectification**. The purpose of this step is to approximate and define the transformation model from the right camera to the left, as a result to row-align and

rectify the image planes. This step is important to facilitate the stereo matching step.

iii. **Stereo Matching**. This step consists on finding same features on both camera image planes (*correspodance*) and constructing a map of differences in x-coordinates of corresponding points, the disparity map. It is necessary for the image planes to be rectified (step *ii.*) for better results.

iv. **Reprojection**. Reprojection stands for the reverse process of 3D to 2D world point projection, as we can "re-project" a 2D image plane point (from pixels) to 3D space (in metric units). To conclude, as we have constructed the disparity map and we know the fully geometric arrangement of the cameras setup, we can extract metric information and construct the depth map.

Subsequently, the aforementioned sections will be shortly presented with some additional stereo imaging theory references.

Epipolar Geometry

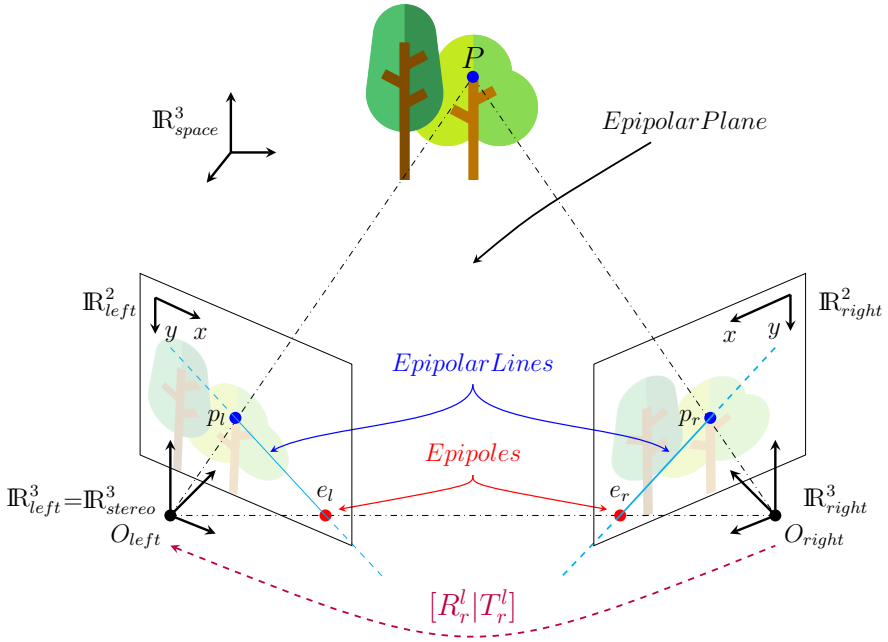


Figure 2.15: Binocular Vision and Epipolar Plane

As Figure 2.15 shows, we assume that we have a two undistorted pinhole cameras setup, with their camera matrices K_i. At first, we need to clarify the different coordinate systems that coexist in this setup. Let $\mathbb{R}^3_{left}$ and $\mathbb{R}^3_{right}$ be the cameras' coordinates systems, $\mathbb{R}^2_{left}$ and $\mathbb{R}^2_{right}$ their image plane (pixel) coordinate systems and $\mathbb{R}^3$ the world coordinate system. Due to previous definitions, we can express the transformation of a point P in right camera coordinate system to left, as a rigid 3D Euclidean transformation,

in which,

$$P_l = R^l_r P_r + T^l_r \tag{2.29}$$

where R^l_r and T^l_r the rotation and translation that is needed to transform a point from $\mathbb{R}^3_{right}$ to $\mathbb{R}^3_{left}$ coordinate system. The computation of those two matrices, that describe the rigid 3D Euclidean transformation from right camera's coordinate system to left, will be the stereo calibration main purpose.

The basic feature of the binocular vision system is the epipolar geometry. In Figure 2.15, let Π_l and Π_r be the image planes of left and right camera, with p_l and p_r the projection of world point P on each of them respectively. In epipolar geometry, epipoles are defined as the intersection points of baseline O_lO_r with the two image planes Π_l and Π_r. The plane in space which is formed by the world point P and the two epipoles points on each image plane, is called the epipolar plane.

As we may notice, there is a connection between the use of epipolar geometry and the depth problem we initially stated. The world point P, as we previously mentioned, loses its depth information when its projected to either Π_l or Π_r plane. The only information that we get is a 3D line, on which P lies on. In Figure 2.15 those lines are the O_lP and $O_rP \in \mathbb{R}^3$ and we can notice that they are projected respectively on Π_r and Π_l. The projections of those lines on each image plane, is called in epipolar geometry as epipolar lines. In addition, for every viewed and projected point (feature as referred in epipolar geometry) p_i on the image plane of one camera, the corresponding point $p_{i'}$ of the other camera, must lie along the epipolar line. This can be expressed mathematically as,

$$p_i^T F p_{i'} = 0, \quad \text{where } i = \text{left or right with } i' \text{ its opposite.} \tag{2.30}$$

In equation (2.30) the parameter F, is called fundamental matrix and contains information about the geometry of the two cameras relative to each other in pixel units. The computation of F is similar to the computation of the homography, that was described

in Subsection [2.4.2], by providing a set of known correspondences. If the fundamental matrix is known we can derive for a number of image points, the corresponding epipolar lines on the other image plane.

Stereo Cameras Calibration

Stereo calibration, in contrast with single camera calibration (explained in Subsection [2.4.2]), deals with the estimation of the geometrical relationship of the two cameras in the world space. Especially, stereo calibration aims on calculating the rotation R and translation T matrix that relates the right camera to the left camera. These matrices are needed to make the two image planes coplanar, which will be useful on the following step of *Stereo Rectification* (Subsection [2.4.3]).

Let P be a world point, which is viewed from both cameras. The point P can be written in each cameras' coordinate system, $\mathbb{R}^3_{left}$ and $\mathbb{R}^3_{right}$, as,

$$P_l = R_l P + T_l \quad \text{and} \quad P_r = R_r P + T_r \tag{2.31}$$

Also, at the start of Epipolar Geometry Section [2.4.3] we defined the transformation from right camera's coordinate system to left camera', as a rigid 3D Euclidean transformation, with the form of $P_l = R^l_r P_r + T^l_r$ (2.29). So, the stereo calibration procedure starts with separate left and right single camera calibration, as described in Subsection [2.4.2] and then uses the above equations (2.31 and 2.29) in the resulted calibration equations to solve for the rotation and the translation parameters between the two cameras. It also applies the Levenberg-Marquardt iterative algorithm to compute the local minimum of the reprojection of the planar pattern located points for both camera views, to conclude with the optimal R and T matrices.

Stereo Rectification

As the stereo rig is calibrated, we know that the image planes of the two cameras can be transformed to be coplanar, i.e all their points belong on the same plane. The purpose of *Stereo Rectification* process is determine a transformation so the image planes will become coplanar and every epipolar line will be parallel to one of the image axis (usually the horizontal axis). Consequently, the search of feature correspondences will be done along the horizontal lines of the rectified images, simplifying the stereo correspondence problem (which will be described in next subsection).

Specifically according to J-Y. Bouguet approach [7] to achieve a coplanar and row-aligned stereo rig system, initially we need to define a matrix R_{rect} that will align the epipolar lines horizontally. Firstly as we know R matrix from the calibration step, we split it half in two rotation matrices r_l and r_r. Each camera will do the half rotation of R, so their principal axis will end up parallel to the vector sum of their original principal axis, which had been pointing in the start. Also, by transposing the two image planes in the same plane, the epipoles of left and right image plane, converge to infinity. So to construct R_{rect}, we create a rotation matrix that is initialized with the direction of an epipole e_1, which is along the translation vector between the two camera's centers of projection,

$$e_1 = \frac{T}{\|T\|} \tag{2.32}$$

For the next vector e_2, we choose a direction orthogonal to the principal axis, which can be given by the cross product of e_1 and the direction of principal axis. With normalization, we can express e_2 as,

$$e_2 = \frac{\begin{bmatrix} -T_y & T_x & 0 \end{bmatrix}^T}{\sqrt{T_x^2 + T_y^2}} \tag{2.33}$$

And the last vector, which must be orthogonal to e_1 and e_2, can be found from their cross product, $e_3 = e_1 \times e_2$.

So the matrix R_{rect} that will rotate the left camera to align the epipolar lines horizontally and take the epipoles to infinity, is,

$$R_{rect} = \begin{pmatrix} (e_1)^T \\ (e_2)^T \\ (e_3)^T \end{pmatrix} \tag{2.34}$$

Thus, the row alignment of the two cameras can be expressed as,

$$R_l = R_{rect} r_l \text{ and } R_r = R_{rect} r_r \tag{2.35}$$

So the rectified camera matrices $\mathtt{K}_{l\_rect}$ and $\mathtt{K}_{r\_rect}$ combined with the projection matrices P_l and P_r, will be,

$$P_l = \mathtt{K}_{l\_rect} P'_l = \begin{pmatrix} f_{x\_l} & alpha_{c\_l} & c_{x\_l} \\ 0 & f_{y\_l} & c_{y\_l} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.36}$$

$$and \tag{2.37}$$

$$P_r = \mathtt{K}_{r\_rect} P'_r = \begin{pmatrix} f_{x\_r} & alpha_{c\_r} & c_{x\_r} \\ 0 & f_{y\_r} & c_{y\_r} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.38}$$

In the above resulted equations, the translation matrices P'_l and P'_r are related to the position of the current camera's optical center in relation to the left camera's. The parameter Tz is 0, because both image planes lies on the same plane and the left camera have always Tx and Ty equal to 0. For the right camera, Ty = 0 (as we assume that we are in the same horizontal line) and $Tx = -f_{x\_r}B$, where B is the baseline between the cameras.

At this point we can define the reprojection matrix, which can "re-project" an image pixel coordinate point into 3D world coordinate system (referenced in fourth step of stereo imaging process [4]).

$$Q = \begin{pmatrix} 1 & 0 & 0 & -c_{x\_l} \\ 0 & 1 & 0 & -c_{y\_l} \\ 0 & 0 & 0 & f \\ 0 & 0 & -\frac{1}{T_x} & \frac{c_{x\_l} - c_{x\_r}}{T_x} \end{pmatrix} \text{ , where } \frac{c_{x\_l} - c_{x\_r}}{T_x} = 0 \text{ if the principal axis intersect in infinity.} \tag{2.39}$$

Hence, the 3D reprojection of a (x,y) image plane point, as we know the disparity, is,

$$Q \begin{pmatrix} \mathtt{x} \\ \mathtt{y} \\ disparity \\ 1 \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix} \text{ , with the world point } P = \begin{pmatrix} X/W \\ Y/W \\ Z/W \end{pmatrix} \text{ in world metric units.} \tag{2.40}$$

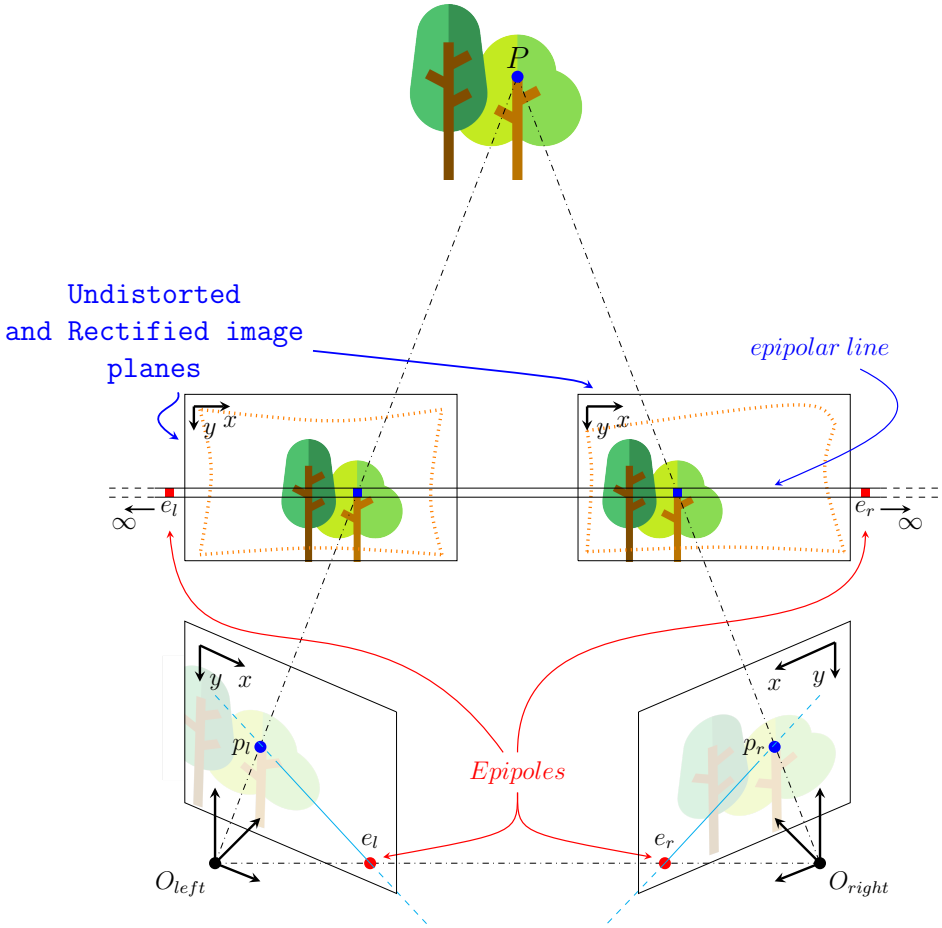


Figure 2.16: Stereo Rectification

Finally, we choose new image centers and set new image borders, to maximize the available common-overlapping viewing area.

Triangulation

Given a perfectly undistorted, rectified and geometrically known positioned stereo camera rig, depth estimation problem can be solved through the *frontal parallel* arrangement model (Figure 2.17). In this setup, we assume that the image planes are coplanar (calibration outcome, Subsection [2.4.3]) and row-aligned (rectification outcome, Subsection [2.4.3]).

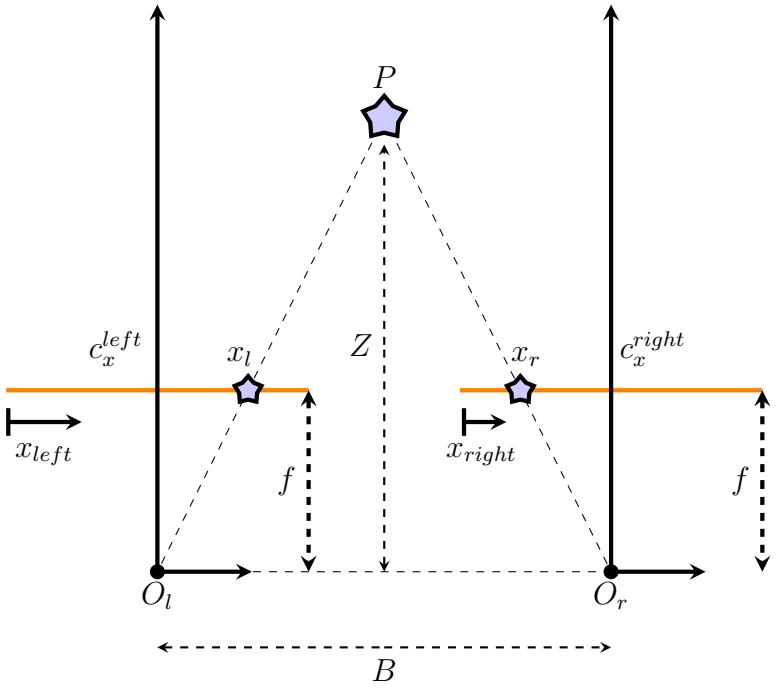


Figure 2.17: Frontal Parallel arrangement

By using of similar triangles theorem, in Figure 2.17, we have,

$$\frac{B-(x_l-x_r)}{B} = \frac{Z-f}{Z} \Leftrightarrow Z(x_l-x_r) = fB \Leftrightarrow$$
$$\Leftrightarrow Z(x_l-x_r) = \frac{fB}{disparity}$$

As we can observe, depth Z is inversely proportional to the disparity between of the cameras' views. Thus, for large disparity values, small disparity differences doesn't change the depth as intensively as as in smaller disparity values. This means that stereo depth estimation is more accurate in shorter distances from the rig, than in bigger and farther ones.

Stereo Correspondence (Stereo Matching)

As mentioned previously in the start of this chapter and specifically in *Stereo Rectification* [2.4.3] and depth Estimation through *Triangulation* [2.4.3], the fundamental problem of

stereo imaging process, is the *Stereo Correspondence*. This part has to do with the detection of similar-corespondent features between the two cameras' image planes, to proceed with the extraction of the disparity map. Given the disparity map combined with the undistorted and rectified cameras' image planes, we can estimate and build the depth map of the viewed scene.

Stereo matching methods can be derived in two categories, the local and the global. Local methods are based on intrinsic parameters of the selected regions that are trying to match and limit their searches on small image regions. Instead, global methods use local methods to observe a more generative-global image pattern, which must satisfy a physical constraint.

In particular, local methods perform the matching process either by discrete feature extraction and matching or by correlating small image patches that are described by a pattern. Feature-based local matching algorithms, in spite their accuracy and rapidness, require high computational resources to execute in real time. In contrast, area correlation methods appear to be more analytical, providing dense results and more computationally efficient, due to the lack of extracting features.

Block Matching Algorithm One of the most known stereo correspondence algorithm, based on area correlation approach, was developed by K.Konolige [8] in 1998. Ever since this algorithm has been added in OpenCV open libraries and used as the basic stereo matching algorithm. The basic parts of this method consist on,

i. Stereo camera's undistortion, calibration and rectification.

ii. Image transformation. Basically, express every pixel area with a certain characteristic-form.

iii. Area correlation. Each area is compared with other areas, in a fixed search window.s

iv. Extrema extraction. The disparity map extraction, based on the extreme values of the correlation at each pixel.

v. Post-filtering. Remove any noises in disparity map.

Area correlation methods uses transformations on pixel intensities to correlate different areas. These can be the *normalized intensities*, *Laplacian of Gaussian* correlations

and even more *nonparametric* transforms. Konolige in his approach uses *LOG* transform with $L1$ norm, on measuring the *absolute difference* correlation. This correlation measurement method is less sensitive in outliers appearance, than others methods (like the *square difference*). Furthermore, at the last step of post-filtering, he uses a combination of *Left/Right Check* and *Interest Operator* to filter any discontinuities and noises in the resulted disparity map. The outline of his algorithm is,

- *LOG* transform with absolute difference correlation.
- Variable disparity search (16,24 or 32 pixels).
- Post-filtering with a combination of *Left/Right Check* and *Interest Operator* filtering methods.
- x4 range interpolation.

Below, Figure 2.18 shows, the result (disparity map) of the stereo block matching algorithm, to stereo snapshot taken on a calibrated and uncalibrated stereo rig. On the uncalibrated rig, the algorithm concludes with less valid matches, because of the small vertical offsets of the image planes (not horizontally aligned/rectified stereo system). On the other hand, the resulted disparity image of the calibrated rig is much more clear and precise, despite of some blank regions in which the image was poor in textures.

Figure 2.18: Calibrated and Uncalibrated disparity image. (*Image from K.Konolige*)

In conclusion, Stereo Block Matching algorithm is ideal for real-time scenarios, where a disparity map must be generated of the viewed scene, to spatially inform the robot about environment. Although, to achieve a more accurate disparity images and improve the 3D world representation, a global method will be introduced in *Our Approach* Chapter [4], which is used in our current implementation combined with a disparity filtering method.

Chapter 3

Problem Statement

3.1 Cooperative Robot Systems in Exploration and Rescue Scenarios

Search and Rescue operations are performed every day in emergency situations, which are occurred by natural disasters outbreaks or are due to human error and sometimes (unfortunately) on human intention. It is important and essential for the rescuing teams to be prepared for such disasters in order to minimize the number of casualties. For this reason, special manufactured for Search and Rescue mobile robots have been developed and used by rescuing organizations, in such scenarios, to enhance their rescuing actions

Robot localization in an exploring area is critical for its navigation. Particularly, in most cases the rescuing robots have to maneuver on obstructed environments, in which global positioning methods may be limited or inapplicable for the robot localization. Also given the area map we can not ensure its reliability, as the exploring area may have been changed to the proposed one. So, mobile robots need to have the ability to construct the environment map in real time through their embedded sensors and also to determine their poses in it, in order to proceed with their navigation and rescuing actions.

In addition, multi-robot approaches are often suggested as they have several advantages compared with the single robot systems, especially in Search and Rescue scenarios. Through their cooperation, they can manage and assign each requested task according to the current cooperation status, they can merge overlapping informations, and thus accomplish single tasks faster, as a single robot system would do. However, these approaches require careful fine tunning and sufficient problem awareness, in order to create

an optimal cooperation solution. Furthermore, task assignment should be done according to robot's capabilities and also the robots in need (*i.e* sensor fault) should be assisted by others in help to accomplish their tasks. To approach such situations, a mature, well-balanced and effective cooperation approach must be suggested.

Finally, rescuing robot's must be capable of perceiving the surrounding world and be able to recognize objects of interest. In Search and Rescue scenarios, the primary task is to locate people in need, so the robots must be able to perform this task. However, due to the need of performing those assignments fast and accurately, efficient processing ways must be implemented in order to achieve that.

3.2 Related Work

In recent years, autonomous navigation systems have been an area of increasing research interest. Most of these approaches ([9, 10, 11]), focus on systems for outdoor or urban search operations, which require the support from the satellites to get the GPS location. However, in indoor navigation scenarios, satellite signals are attenuated and scattered by building materials and the building morphology, resulting inaccurate location approximations. In some cases, the occurred metric error could be even in the size of the exploring area, as a result making this localization method totally impractical.

In contrast to satellite-based systems, that are the de-facto standard in outdoor navigation scenarios, in indoor positioning there are many approaches characterized by dedicated local infrastructure and customized mobile robot units, depending on the scenario, to solve the localization problem. These approaches can be classified in two categories, the wireless technologies and the non-radio technologies.

In the first category, the robot location estimation is done by extracting information from the existing wireless infrastructure that is installed. In specific, there are approaches that use signal strength measurements from WiFi access points [12, 13] or from Bluetooth low energy beacons [14], in an organized network and thus with the combination of robot's odometry data, they can estimate its location in the exploring area. In addition, most recently, there are publications on cheaper indoor positioning solutions, like the use of radio frequency identification (RFID) technology [15], in which a known arranged grid of passive tags is set up in the exploring area and with a RFID reader attached on the robot frame the localization is done through the received signal strength (RSS) information. Those implementations, however, require some pre-work on setting up the

wireless infrastructure and preparing the area accordingly, to proceed to the mapping process and deploy the mobile robot. Consequently,, those approaches are time consuming and thus inapplicable, specifically in an emergency situation in which an immediate Search and Rescue scenario must be planned and performed. As it can be clear, my approach in this diploma thesis belongs to the second category, in which the mobile robot will be equipped with all the necessary sensors and perform the localization itself, without any interfering external supporting infrastructure.

Specifically on the other hand, non-radio approaches, use information from robot sensory data to estimate its location in the environment. Recently, Xing *et al.* [16] implemented a marker-based multi-sensor indoor navigation algorithm, ideal for micro aerial vehicles (MAVs), which uses the fusion of the inertial sensor raw data and ultrasonic sensor measurements along with the visual information of the proposed markers, which are placed randomly on the ground, to build a map automatically. Also, an indoor positioning approach, which published last year [17], shows the capability of a mobile robot to be able to navigate indoors, by using the information from its odometry, fused with the orientation data from an attached electronic compass. Lastly, there are approaches that combine methods from the above categories, for a more accurate location estimation, like combining WiFi network RSS measurements with inertia measurement unit raw data [12] and the odometry data [18]. Although, methods based mostly on odometry may introduce errors to the localization process, due to sensor error models, so there is the need for a more robust visual pose extraction method to be introduced, without any visual markers intervention.

SLAM algorithms (as described in Section [2.3.3]) are capable of building a map of the surrounding environment (indoor or outdoor) and simultaneously use it to localize the robot in it. In particular, during the recent years SLAM problem have received a lot of attention, due to wide adaptability to robotic exploring scenarios, with different approaches being demonstrated. In RoboCup Rescue competition, Visual SLAM implementations have been showed [19] , with promising results, but they are limited due to cameras specifications and lens distortions, as they are affected by light and imaging conditions and they may require high computational resources. A widely known open-source SLAM algorithm, is the Gmapping approach [20], which is based on Rao-Blackwellized particle filtering. Although, this method depends on sufficiently and accurate odometry data and is affected of any roll and pitch motions (made for planar environments), which makes it unusable for unmanned aerial vehicles (UAVs). In contrast to this, there are

publications for Quadcopters, like [21], that solve the SLAM problem with the use of a LiDAR system, but they do not provide full 6DoF pose estimate. In publication [22], Team Hector developed an open-source flexible SLAM system, that was specific designed for Urban Search and Rescue (USAR) scenarios in RoboCup Rescue. This system relies on fast LiDAR data scan-matching at full update rate and inertial data, providing 6DoF pose information about the navigated robot. Since then, Hector SLAM is one of the de-facto used SLAM systems, that solves the *online* SLAM (2.5) problem, and it is used in a plethora of robotic exploration applications. To sum up, Hector SLAM is used in my approach onboard, on the quadcopter through its ROS node[1], to perform the localization and mapping.

Finally, a research area that has made important steps in past few years, is the cooperative robotics. Specifically in Search and Rescue scenarios, multi-robot systems could aid and cover a vast area more efficiently and quicker, than a single robot system. Many works have been implemented in multi-robot cooperation [23, 24, 25] for SAR scenarios, but they were based on ground mobile robots. In addition, publications like Cai *et al.* [26], uses a MAV to incorporate with a ground-robot team, to perform localization and mapping algorithms through a low-cost range finder sensor, transmitting all the sensor information to a portable PC for processing. Furthermore, there are publications that use dedicated middleware for SAR scenarios, like CINeMA [27], to ensure network connectivity and cooperative localization. In my approach, ROS [1] middleware is used as the network infrastructure, due to its open-source philosophy and its high modularity on package construction and deployment.

[1] https://github.com/tu-darmstadt-ros-pkg/hector_slam

Chapter 4

Our Approach

4.1 Network and ROS Setup

First of all, to achieve the aerial-ground robot system's collaboration, an optimal and adequate connection is needed to be established for their communication. In specific, due to the variety of environment conditions and morphologies, where a Search and Rescue scenario need to be approached (i.e urban environments, outdoor situations and more), we proposed a 2.4GHz WiFi network setup, which was sufficient in most of the tested cases, under certain mapping distances (20m-30m range).

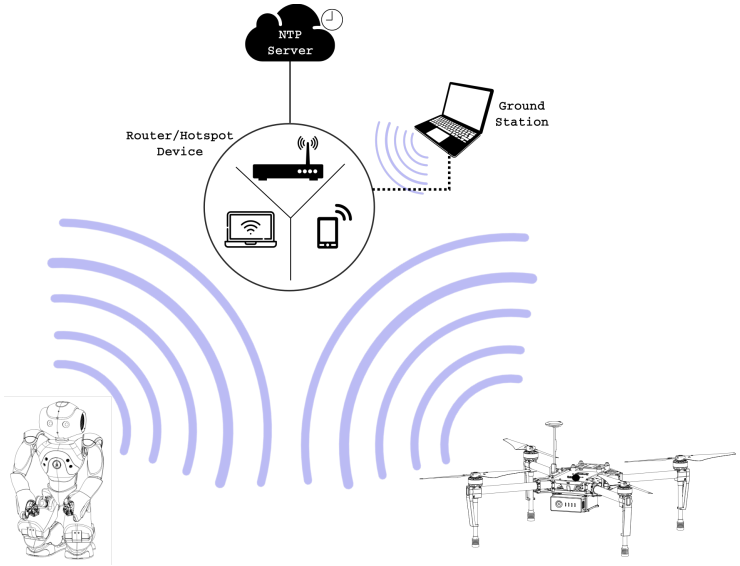


Figure 4.1: Network Structure

The 2.4GHz instead of the 5GHz frequency band was selected to achieve wider range coverage, as the latter frequency signals may be absorbed or obstructed in such urban or high-vegetation environment conditions. Despite the selected band's smaller network speed, the ROS network is adjusted optimally to work for our approach, and thus, for the mobile robot team to be able to cover bigger areas.

Each mobile robot was assigned to a known static IP address, under its connection to the router device. The ground station (a laptop in our case) is also connected in the same network, either through wireless or wired connection. Also, the ground station is registered as the ROS master device, which role is to enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer. Through the ground station we will be able to monitor the robot collaboration process, for the mapping and the navigation, and also provide target positions in the created map for the the ground robot to go and search for individuals. An illustration of the aforementioned network structure is shown in Figure 4.1.

Not to mention, as ROS uses the systems' clock as a time source for applying timestamps on streaming data, there may be time differences as both systems are started up. For this reason, each time we boot systems up, we set both robot systems' clocks to a common NTP server's clock. In this way, any generated ROS message, from each robot, will be almost synchronized (synchronization differences up to $10ms$) to the same clock, and thus can be processed with respect to time. Also, when Internet connection was unavailable, we use the *chrony* UNIX daemon, which synchronizes the robot clock system according to the ground station system clock.

Matrice and Nao ROS Nodes

For both the *Matrice* and the Nao robot system, we used the ROS SDK packages as provided in open-source ROS stacks. In *Matrice* ROS package, we added a new node, which included the publication of all stereo camera's image raw data and ultrasonic data, captured from *Guidance* system. An overview of ROS topic publications, in systems startup is shown on the below figures.

The dji_sdk ROS Node topics:

geometry_msgs/QuaternionStamped
dji_sdk/attitude
sensor_msgs/BatteryState
dji_sdk/battery_status
sensor_msgs/Imu
dji_sdk/imu
std_msgs/UInt8
dji_sdk/flight_status
std_msgs/Float32
dji_sdk/height_above_takeoff
geometry_msgs/Vector3Stamped
dji_sdk/velocity

The allCamerasAndSonars_M100 ROS Node topics:

sensor_msgs/Image
/guidance/*/left_camera/image_raw
sensor_msgs/Image
/guidance/*/right_camera/image_raw
sensor_msgs/Range
/guidance/*/ultrasonic

where (*) = front,left,right, down and rear.

The naoqi_driver ROS Node topics:

naoqi_bridge_msgs/AudioBuffer
/naoqi_driver/audio
sensor_msgs/CameraInfo
/naoqi_driver/camera/bottom/camera_info
sensor_msgs/Image
/naoqi_driver/camera/bottom/image_raw
sensor_msgs/CameraInfo
/naoqi_driver/camera/front/camera_info
sensor_msgs/Image
/naoqi_driver/camera/front/image_raw
sensor_msgs/Imu
/naoqi_driver/imu/torso
naoqi_bridge_msgs/StringStamped
/naoqi_driver/info
nav_msgs/Odometry
/naoqi_driver/odom
sensor_msgs/Range
/naoqi_driver/sonar/left
sensor_msgs/Range
/naoqi_driver/sonar/right

4.2 Total Coordinate System setup and *tf* Library

Generally, robots are complex systems as they consist on moving parts, sensors and other complex mechanisms. Furthermore, there may be scenarios that include many robots like

this (like our scenario), in which robots coexist in the same area and need to cooperate to accomplish a certain task. This fact leads to a need for representing the positions and orientations of robots, including their sub-system parts, in same coordinate system to proceed accordingly and solve the scenario.

For this reason, in robotics field it is assumed that each interacting-existing robot part is described by its coordinate system and its origin, which is given by a position and a rotation vector in reference to another an frame. Especially, assuming a point P in the coordinate frame $\{A\}$, it can be represented as a 3x1 position vector (like a point in space), as ${}^{A}P = [p_x\ p_y\ p_z]^T$. To describe the orientation of a body, we attach a coordinate system to the body and then we give a description of this coordinate system relative to the reference system. So for a body-attached coordinate system $\{B\}$ on a coordinate system $\{A\}$, the description of $\{B\}$ can be written by expressing the unit vector if its three principal axis in reference to $\{A\}$.

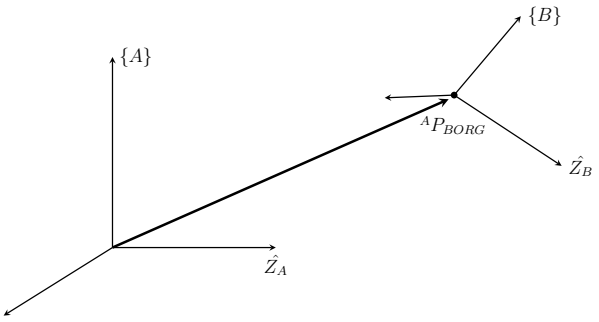


Figure 4.2: The relative position of $\{A\}$ and $\{B\}$ coordinate systems.

As shown in Figure 4.2, we denote the unit vector giving the principal directions of coordinate system B as the $\hat{X}_B$, $\hat{Y}_B$ and $\hat{Z}_B$. Also, the principal direction vectors can are denoted in coordinate system $\{A\}$, as ${}^{A}\hat{X}_B$, ${}^{A}\hat{Y}_B$ and ${}^{A}\hat{Z}_B$. Thus, we define the matrix of the expressed $\{B\}$ principal axis in $\{A\}$ coordinate system as,

$$ {}^{A}_{B}R = [{}^{A}\hat{X}_B\ {}^{A}\hat{Y}_B\ {}^{A}\hat{Z}_B] = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{23} & r_{33} \end{bmatrix} \tag{4.1} $$

which is known as the rotation matrix as it describes the frame $\{B\}$ relative to $\{A\}$.

Also 4.1 can be expanded as,

$$ {}^{A}_{B}R = [{}^{A}\hat{X}_B \; {}^{A}\hat{Y}_B \; {}^{A}\hat{Z}_B] = \begin{bmatrix} {}^{B}\hat{X}^T_A \\ {}^{B}\hat{Y}^T_A \\ {}^{B}\hat{Z}^T_A \end{bmatrix} = {}^{B}_{A}R^T \tag{4.2} $$

Hence, ${}^{A}_{B}R$ the description of the frame $\{B\}$ relative to $\{A\}$ is given by the transpose ${}^{B}_{A}R$, which is also the inverse of the rotation matrix. So,

$$ {}^{A}_{B}R = {}^{B}_{A}R^{-1} = {}^{B}_{A}R^T \tag{4.3} $$

Thus, the frame of $\{B\}$ is described relative to $\{A\}$, as follows,

$$ \{B\} = \{{}^{A}_{B}R, {}^{A}P_{BORG}\} \tag{4.4} $$

Finally, the mapping/transformation of a point P in coordinate system $\{B\}$ to $\{A\}$, is denoted by ${}^{A}_{B}T$ and forms as,

$$ {}^{A}P = {}^{A}_{B}T \cdot {}^{B}P \text{ , where } {}^{A}_{B}T = \left[\begin{array}{ccc|c} & {}^{A}_{B}R & & {}^{A}P_{BORG} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{4.5} $$

However, many complex robot systems are consisted with parts that are moved with respect to the base frame, *i.e* the hands of a humanoid robot in respect to its torso frame. This means that the transformation matrix between two specific moving parts (frames) is differentiated from time to time. Thus, it's necessary to store the timestamp information when any of those transformation is calculated, despite if it's static or not. The ROS communication infrastructure, discussed in Section [2.2], is based on timestamps and frames to express measurements, robot poses and more, as they were posted at a given time, in the corresponding coordinate system. The ROS ecosystem has an integrated transform library, the *tf* [3], which follows the aforementioned "frame philosophy", supports fully the previous transformation methods and is mainly used as primary way to keep track of positional information. Therefore, it is extensively used in our approach as we need constantly information about robot's relative poses and their positions in the generated map frame.

Thus, we created a dedicated ROS node to provide the quadcopter static transforms, including the LiDAR frame and the *Guidance* cameras' and sonars' frames. We precisely measured the sensor positions in reference to a common quadcopter body point (assumed

the $base\_frame$ origin) and included the resulted real world positions into the tf tree. By this way, the tf tree contains frame transformations described in real world units, and thus, any processed measurement and extracted result will be spatially described the same way. The quadcopter tf tree node is illustrated in Figure 4.3 and its structure visualization in the RViZ plugin, shown in 4.5a.

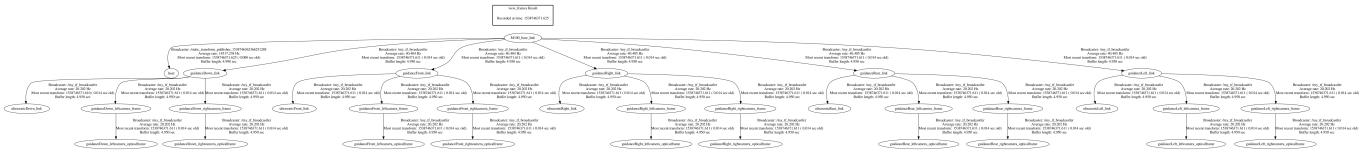


Figure 4.3: Quadcopter tf tree

On the other hand, the Nao tf tree expansion and visualization is shown in 4.4 4.5b, respectively, and it's used as provided from the ROS driver module for NAOqi OS. It publishes all sensor and actuator data as well as basic diagnostics messages, like battery and temperature.

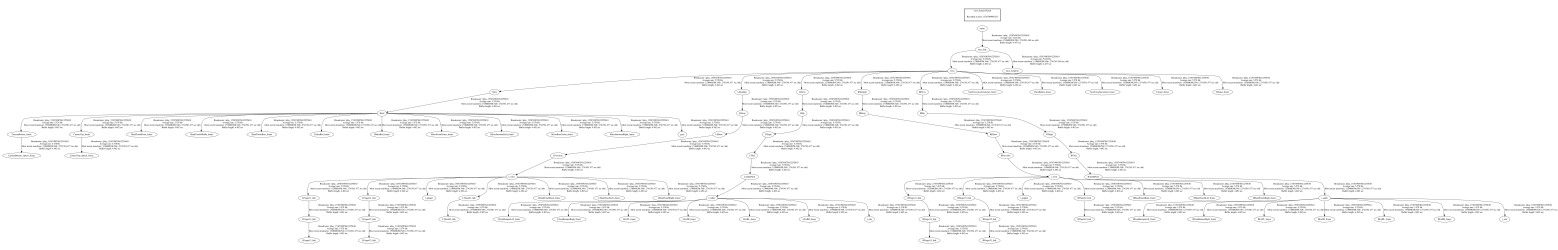
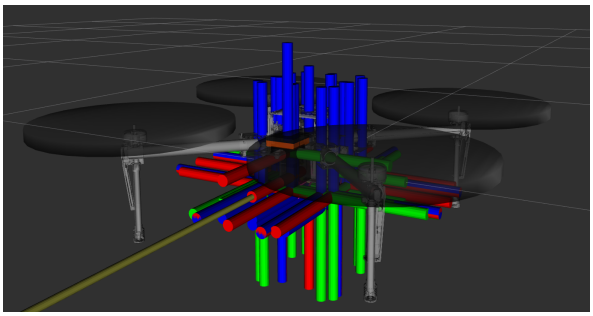
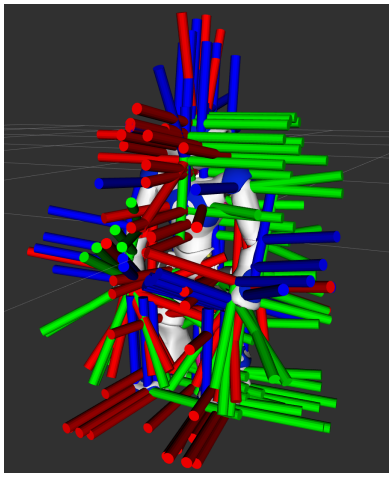


Figure 4.4: Quadcopter tf tree



(a) The view of the standard tf trees of the *Matrice* system.



(b) The view of the standard tf trees of the Nao system.

Figure 4.5: Views of tf trees in RViZ plugin.

4.3 Guidance system Stereo-Cameras Calibration

Camera calibration is the process of finding the true parameters that describe a camera system, as discussed in Subsection [2.4.2]. The purpose of calibration procedure is critical for lens distortion correction, estimation of the camera's geometry (internal calibration), as well as, for extracting its position in relation to the scene (external calibration). Notably, the stereo cameras calibration process provides additional information about cameras' relative position in the 3D space and thus we can calculate the reprojection matrix of transforming the 2D image plane pixes to 3D world units (eq. 2.40). Consequently, in our approach every guidance stereo camera system is calibrated before applying any image processing algorithm, especially needed for the following stereo matching process.

For the cameras' calibration step, Bouget's *Camera Calibration Toolbox* [7] for *MATLAB*® is used for each *Guidance* stereo camera system, to extract cameras' information. In specific, the stereo calibration procedure calculates the *Camera* and *Distortion* Matrices, as well as the *Rotation* and *Translation* matrix that can transform a point described in right camera's coordinate system to the left camera's (as described in Subsection [2.4.3]).

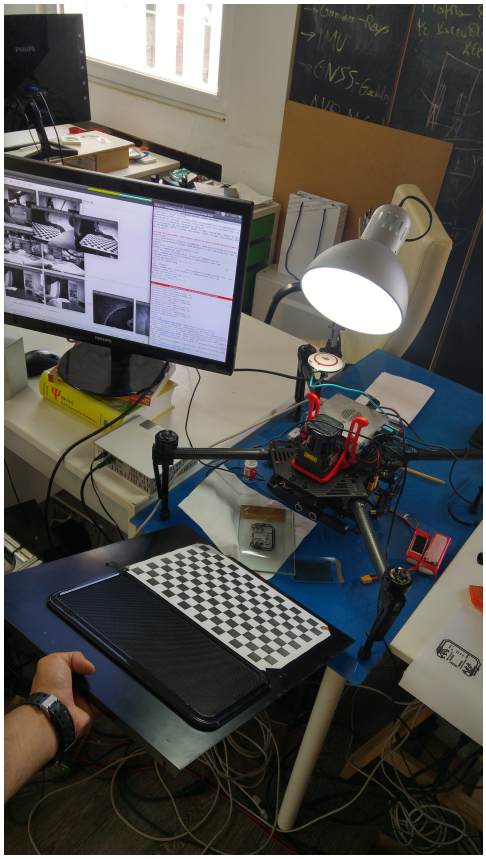


Figure 4.6: *Guidance* stereo camera calibration process.

Thus, for each *Guidance* stereo camera system we captured a number of 20-30 images that show a 16x11 planar chessboard pattern in a variety of orientations and positions (Figure 4.6). The selected calibration pattern had checker squares of 17mm per side, precisely. Given that, the *Camera Calibration Toolbox* program executed the calibration procedure and estimated the camera system parameters.

Once the calibration is done, to evaluate the accuracy of the estimated parameters we

considered two *Camera Calibration Toolbox* features. Firstly, assuming a detected pattern keypoint in the image plane, the toolbox calculated the error distance of reprojecting the corresponding world point on the image plane, in pixel units (Figure 4.7). Also, the *Camera Calibration Toolbox*, given the physical characteristics of the calibration pattern, could estimate the positions of the camera system in relation to the calibration pattern views, in metric units (as illustrated in Figure 4.8a). It's noteworthy that the estimated stereo baseline distance is given by the T_x value of the estimated Translation matrix, as our camera systems are aligned parallel on the same base axis (shown also by the panoramic view of the estimated stereo camera allocation Figure 4.8b). Henceforth, in respect to the reprojection errors and the estimated stereo camera baseline, re-calibrations were made after additions and removals of stereo image pairs from the calibration dataset, until smaller than 0.08 overall mean reprojection error in pixel units is achieved and less than 0.02% baseline distance error is occurred in metric units, as we know that the real one is 15mm.

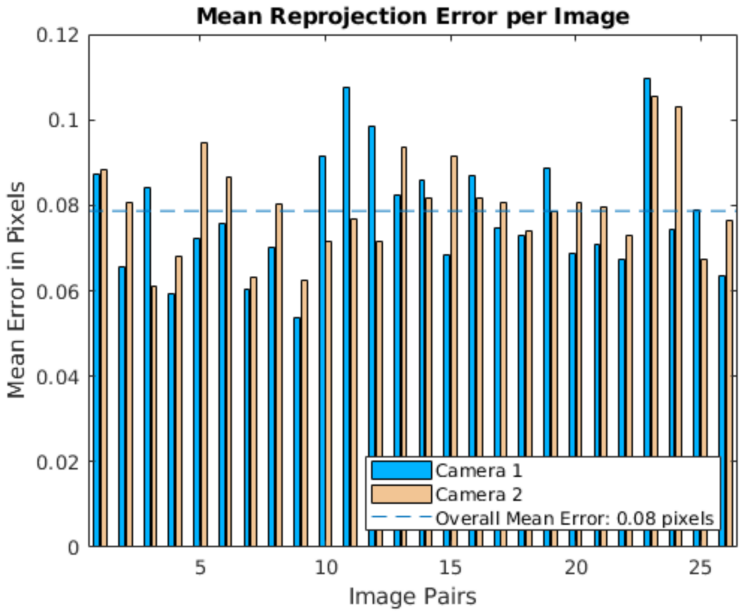
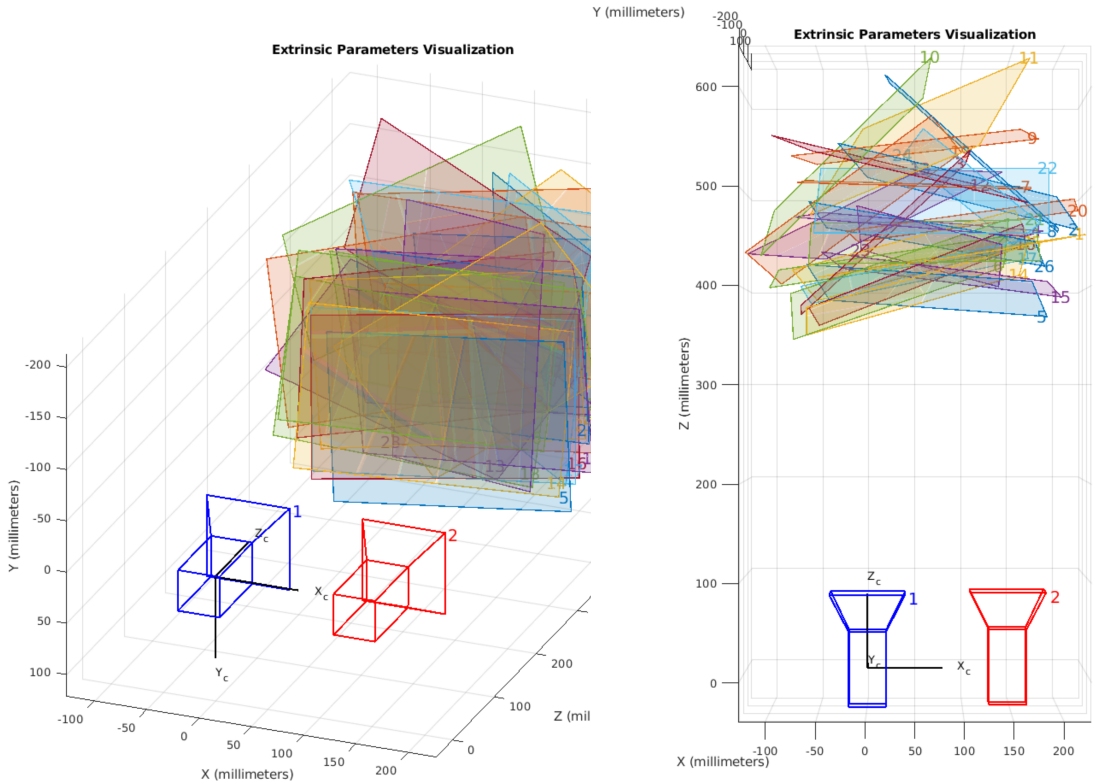


Figure 4.7: Mean Reprojection Error per Image pair in *Guidance* Front Stereo camera

As the corresponding stereo camera's geometrical model is estimated, the following step is to calculate the rectification transforms for each calibrated camera's image plane. The purpose of stereo rectification process is to transform the camera's image planes to become coplanar and every epipolar line will be parallel to one of the image axis (usually



(a) Side view

(b) Panoramic view

Figure 4.8: The spatial configuration of the two cameras and the captured calibration planes in 3D world space.

the horizontal axis), and thus simplify stereo matching process. Therefore, a node is implemented to calculate the *Rectification* matrices for each camera, which will align their image planes to an ideal one so that epipolar lines in both stereo images will be parallel. Also, the *Projection* matrices are computed, which represent the *Intrinsics* matrices of the processed/rectified images. Not to mention, the *Reprojection* matrix, namely Q, is computed as described in Subsection [2.4.3]. The previously referred computations have been made with OpenCV *calib3d* class functions.

The total stereo cameras' description parameters are stored in XML files (in $yaml-cpp$ format) locally on *Matrice* system and there is a specific ROS node implemented, namely $camerainfo\_publisher$, that imports these camera information matrices from the XML files, for each of the five cameras, and publishes them in $sensor\_msgs/CameraInfo$

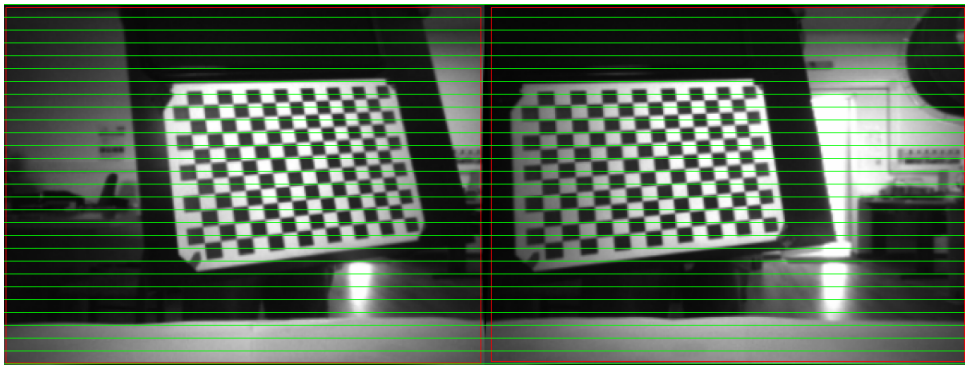
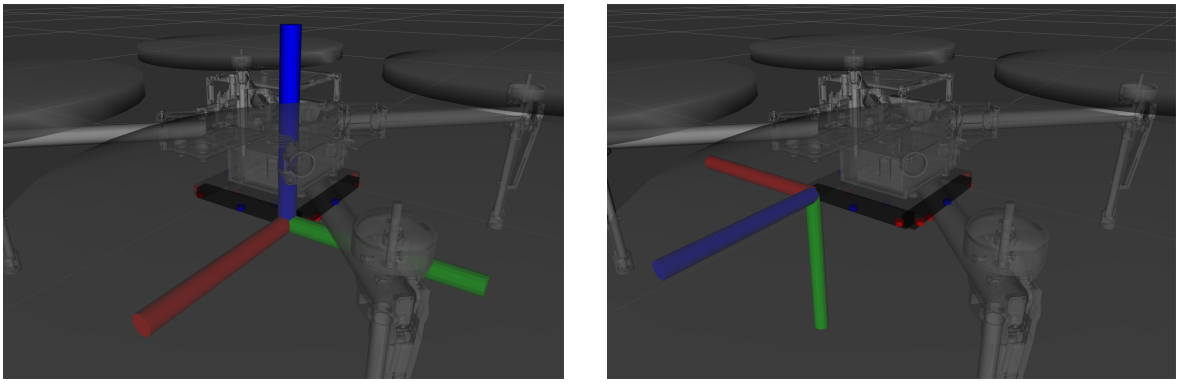


Figure 4.9: Undistorted and Rectified *Guidance* Front Stereo Camera Image Planes.

ROS message format, to be available during the *Guidance* cameras' raw data processing. In addition to that, a separate node is created for rectifying the captured image raw data from each camera system, by applying the aforementioned rectifying methods (discussed in Subsection [2.4.3]), given the camera information matrices from their dedicated $sensor\_msgs/CameraInfo$ topic. Figure 4.9 shows the result of front stereo camera's image planes rectification, in which despite the small distortion effect the rectification process selects a slightly smaller region of interest on the image plane and applies the remapping effect to rectify each stereo image plane accordingly.



(a) Camera frame (b) Optical frame

Figure 4.10: Visualized (a) left camera's frame and (b) right camera's optical frame axes in RViZ.

Finally, optical camera frames are added in the total TF tree, in respect of ROS

REP 103[1] and OpenCV *Mat* coordination system setup, by which the top-left corner is considered as the image frame origin (Z-axis: forward, X-axis: right and Y-axis: down). The Figures 4.10a and 4.10b show the physical camera and its optical frames, respectively.

4.4 Stereo Processing with a Global Block matcher and a Disparity Post-Filtering method

Stereo Correspondence problem is vital for accurate 3D world reconstructions. For enhancing the stereo matching outcome by keeping the process computation in similar complexity levels, we use a global block matching method known as *Semi-Global Matching* (*SGM*) [28]. The *SGM* method, as opposed to the *BM* method (explained in paragraph 2.4.3), performs pixel-level matching as approximates a global cost function by pathwise optimizations from all directions through the image. In particular, as a semi-global method the disparity map computation is done by selecting the disparity values for each pixel individually that corresponds to the minimum cost (local optimal), as enforces smoothness constraints to penalize changes of neighboring disparities. The matching cost is based on the radiometric differences, described by the Mutual Information [29] of the input images.

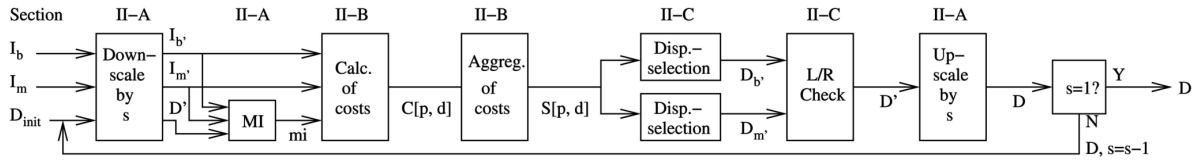


Figure 4.11: *SGM* total processing steps. (Image from H.Hirschmüller)

In specific, the *SGM* method execution consists in three parts, the Matching Cost Calculation, the Cost Aggregation and the Disparity Computation (as it is shown with symbols II-A, II-B and II-C in Figure 4.11, respectively). By assuming the left image as the base image I_b and the right as the match image I_m, we define a pixel $\mathbf{p}$ in base image plane and its correspondence pixel $\mathbf{q}$ in match image, where,

- $I_{b\mathbf{p}}$ is the $\mathbf{p}$ pixel intensity in the base image.

[1] http://www.ros.org/reps/rep-0103.html

- $I_{m\mathbf{q}}$ is the $\mathbf{p}$ pixel suspected correspondence with pixel $\mathbf{q}$ of the match image.
- $\mathbf{q} = e_{bm}(\mathbf{p}, d)$, where e_{bm} is the epipolar line in the match image for the base pixel $\mathbf{p}$ and d a line parameter. In our case, as the image planes are rectified, the $\mathbf{q}$ is expressed by $e_{bm}(p_x - d, p_y)$, where d is the disparity value.

The matching cost calculation is based on the Mutual Information (MI) of the two images, which is described by their entropy (separate information content) and their joint entropy. The total MI matching cost is calculated as,

$$C_{MI}(\mathbf{p}, d) = -mi_{Ib,f_D(I_m)}(I_{b\mathbf{p}}, I_{m\mathbf{q}}) \quad \text{, where} \quad mi_{I_1,I_2} = h_{I_1}(i) + h_{I_2}(k) - h_{I_1,I_2}(i,k) \tag{4.6}$$

The terms h_{I_1}, h_{I_2} and h_{I_1,I_2} are calculated from the probability distributions of corresponding intensities P_{I_1}, P_{I_2} and P_{I_1,I_2}, respectively, through $Parzen$ estimation method. The function f_D represents the warping of I_m to the disparity image D. This cost is then used for matching both images and calculating the new disparity, which is used in the next iteration. The starting disparity D_{init} is randomly initialized, as it's used only to estimate the probability distribution of the corresponding intensities P and the matching costs of the C_{MI} of the higher resolution level. Then through a recursive way, this method uses the up-scaled disparity to calculate hierarchically the MI costs. In our approach we use the simpler sub-pixel metric of $Birchfield - Tomasi$ [30], proposed by *H.Hirschmüller* in [28], that provides a dissimilarity measure to find pixel correspondences and enforce the consistency constraints with dissimilarities occlusion.

To avoid noise and ambiguity inclusion in cost calculation result, a smoothness constraint is enforced between the neighboring pixels disparities, to penalize high-volume and unexpected changes. The pixelwise cost and the smoothness constraints are expressed by the energy function $E()$ of the disparity image D, as follows,

$$E(D) = \sum_{\mathbf{p}} \Big(C(\mathbf{p}, D_{\mathbf{p}}) + \sum_{q \in N_{\mathbf{p}}} P_1 T\big[|D_{\mathbf{p}} - D_{\mathbf{q}}| = x\big] + \sum_{q \in N_{\mathbf{p}}} P_2 T\big[|D_{\mathbf{p}} - D_{\mathbf{q}}| > x\big] \Big) \tag{4.7}$$

where the operator $T[]$ expresses a boolean function and $N_{\mathbf{p}}$ the neighborhood of pixel $\mathbf{p}$.

In energy equation 4.7, the first term is the sum of all pixel matching costs for the disparities of D. The second and third term control the disparity smoothness by adding

constant penalties to the energy function, depending on the neighboring pixel disparity changes. Particularly, a small constant penalty $P1$ is added for all the pixels $\mathbf{q}$ in neighborhood of $\mathbf{p}$ where disparity changes by a small value (x pixels), to preserve slanted surfaces. A larger constant penalty $P2$ is added for larger disparity changes (appeared as depth discontinuities) which are caused by pixel intensity changes. To obtain best results, we tunned $P1$ and $P2$ parameters experimentally, by re-playing different data captures of indoor and outdoor environments.

The problem of stereo matching is to find the disparity image D that minimizes the energy $E(D)$. The aggregated cost of a pixel $\mathbf{p}$ and disparity d is calculated by summing the costs of all 1D minimum cost paths (as shown in Figure 4.12), that end in pixel $\mathbf{p}$ and disparity d.

The aggregated (smoothed) cost is given by,

$$S(\mathbf{p}, d) = \sum_{\mathbf{r}} L_{\mathbf{r}}(\mathbf{p}, d) \qquad (4.8)$$

where L_r is the cost along the path in the direction $\mathbf{r}$ of the pixel $\mathbf{p}$ at disparity d.

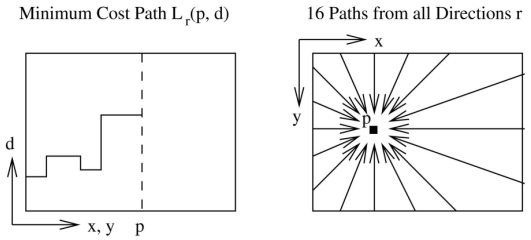
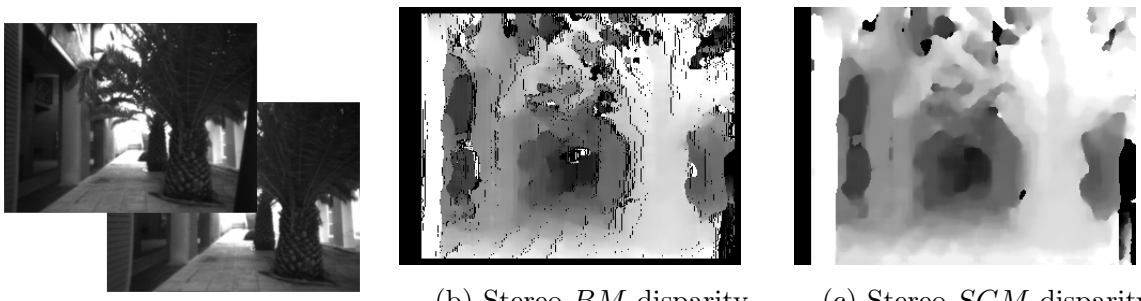


Figure 4.12: Costs aggregation in disparity space. (Image from H.Hirschmüller)



(a) Input Stereo Images

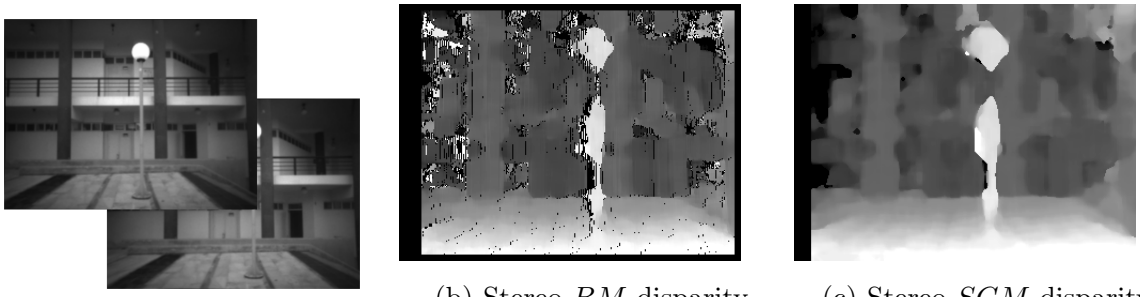
(b) Stereo BM disparity image

(c) Stereo SGM disparity image

Figure 4.13: Disparity images calculated with Stereo BM and Stereo SGM methods.

As a result, the final disparity image D_b that corresponds to the base image is derived by selecting the disparity that correspond to the minimum cost of $S[\mathbf{p}, d]$ for every pixel $\mathbf{p}$. Similarly, the disparity image D_m is calculated from same costs by traversing the

epipolar line that corresponds to the pixel $\mathbf{q}$ of the match image and by selecting the disparities that suffices the minimal cost of $S[e_{mb}(\mathbf{q}, d), d]$. Finally, consistency check is applied on disparity images D_b and D_m to exclude false matches.



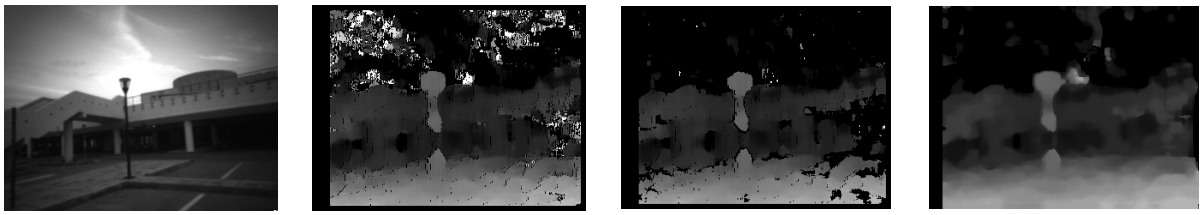
(a) Input Stereo Images

(b) Stereo BM disparity image

(c) Stereo SGM disparity image

Figure 4.14: Disparity images calculated with Stereo BM and Stereo SGM methods.

The Figures 4.13b and 4.14b show the result of Stereo Block matching algorithm, given the images in Figures 4.13a and 4.14a as an input. This resulted disparity image was produced using 16 disparity maps/depth resolution and by matching blocks of 13 pixels instead of one, as proposed by K.Konolige in [8]. Also, none of the pre or post-processing methods are applied (i.e *Sobel* pre-filtering, speckle filtering or else). In most cases, Stereo BM produces disparity discontinuities (as shown in Figure 4.15b) at low-texture detailed image regions, due to algorithm's local-cost behavior. The *uniqueness ratio* parameter may resolve some discontinuities (as Figure 4.15c shows) by adjusting the winner takes all cost function margin in disparity computation, but it can not overcome SGM application results.



(a) Optical Image (b) Stereo BM (c) Filtered Stereo BM (d) Stereo SGM

Figure 4.15: Disparity images calculated with Stereo BM and Stereo SGM methods.

In addition, Z.Farbman *et al.* [31] presented a multi-scale edge-preserving decomposition method for images, to extract and retain multi-scale detail characteristics. Particu-

larly, this edge-preserving operator is based on weighted least squares (WLS) framework and it's provably ideal for progressive imagery coarsening as well as detail extraction in various spatial scales. In 2014, D.Min *et al.* [32] used the WLS decomposition operator to implement an inhomogeneous global edge-preserving smoothing method, namely, the *fast global smoother* (*FGS*). The *FGS* technique allows fast global objective function approximation, by expressing the linear system solution as an inhomogeneous Laplacian matrix defined over a *d*-dimensional spatial domain. This approach permits the application of tridiagonal matrix algorithms in a cascaded fashion iteratively.

As a global smoother algorithm it has been tested in a variety of computer vision applications, including depth upsampling tasks. The *FGS* algorithm can upsample low-resolution depth map through a sparse data interpolation method, in which the initially captured images are defined as the guide images and the computed depth map as the sparse input. By solving the resulted linear system the depth map is adaptively propagated by considering the structure of the guidance image, and increases its spatial resolution.

A dedicated ROS node, namely *sgbm_w_wls*, have been implemented to compute the disparity image with SGM method combined with *FGS* WLS-based filter application, in a specific manner as described below, in order to enhance its spatial information.

(a) Optical Image (b) Stereo *SGM* (c) Stereo *SGM* with *FGS*

Figure 4.16: Disparity map enhancement with WLS-filter application.

In specific, combined with the SGM method, the *FGS* filter make use of the left-right-consistency-based confidence to refine the results in half-occlusions and uniform disparity areas. We assume that the left grayscaled stereo-pair view is the *FGS*'s guidance image and we set it as the main matcher. Then, the WLS-filter node initializes the right matcher, in respect to the left/main matcher, to be able to compute the right-view disparity map. As the disparity computations are done through the SGM method, from each stereo view

individually, we apply the WLS-filter by giving the main view and the two calculated disparities (left-to-right and reverse) as an input. The output disparity image is the upscaled WLS-filtered disparity image, which has selectively preserved the sharp edges, by fading the smaller features in the magnitude and thus been spatial enhanced over low-textured areas. Figures in 4.16 and 4.17 show some stereo captures and the disparity images after WLS-filter application, compared with the aforementioned stereo matching processes. It is noteworthy that the WLS-filter defines an 1D smoothing parameter λ, which defines the amount of regularization during the filtering iterations, and permits variance tuning for setting filtering sensitiveness to guidance image adaptation. In our experiments, we used a slightly higher value for λ of the proposed one (9000 instead of 8000), to preserve extreme situations in which the SGM method gives a uniform disparity map and thus more source image information must be adhered in the final filtered disparity map.

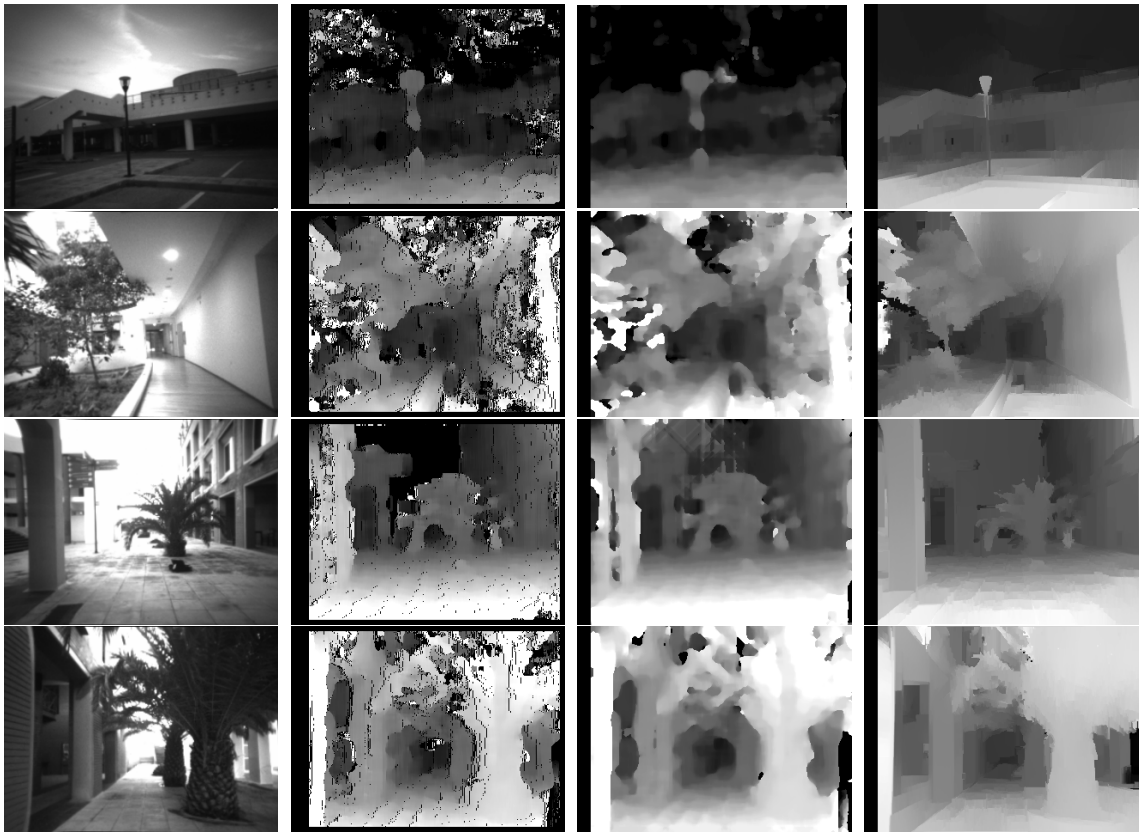


Figure 4.17: Examples of upsampling disparity images with *FGS* post-filtering.

Moreover, this node implements disparity map to 3D world coordinates conversion with the proposed approach in Subsection [2.4.3], by using the reprojection matrix Q (eq. 2.40). Stereo matching procedure is applied in each *Guidance* system's stereo camera and the derived disparity maps are transformed to depth maps. The generated depth maps are expressed in respect to left view of the stereo-cameras' coordinate system.

The front *Guidance* stereo camera system is sufficient for generating a 3D map estimation, hence its mainly used in our scenarios. Although, a node that uses all *Guidance* cameras image raw data has been implemented in order to use the spatial information generated from all the camera sensors, and thus to generate more accurate 3D mappings. More information will be provided in the following Section [4.5].

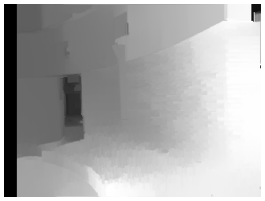
4.5 Point Cloud Library and 3D Mapping with Octrees

The Point Cloud Library [4] (PCL) is an open source C++ library, specially designed for 3D point cloud processing. Due to its implementation in respect to ROS architecture and its integration with it, it was the ideal library to manipulate the stereo matching computed 3D depth maps. Specifically, we express the depth maps in point clouds message forms, with the use of *Boost* shared pointers[1] to be able to manage the memory consumption due to their big sizes and achieve fast computations.

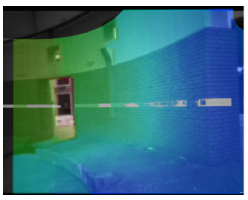
The Figure in 4.18 shows the transformation of an estimated depth map in point cloud form, in RViZ plugin. Also, the illustrated white line(spheres) represent the LiDAR scanned endpoints. The fusion of the left camera's scene view and the point cloud, is shown in Figure (4.18c). The point cloud data is appeared aligned to the optical camera's data, because they are expressed in left camera's coordinate system. Figures (4.18d) and (4.18e) show the point cloud data in third person perceptive.

[1] https://www.boost.org/doc/libs/1_62_0/libs/smart_ptr/shared_ptr.htm

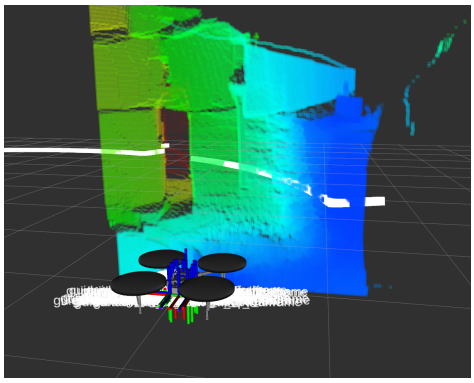
(a) Optical Image



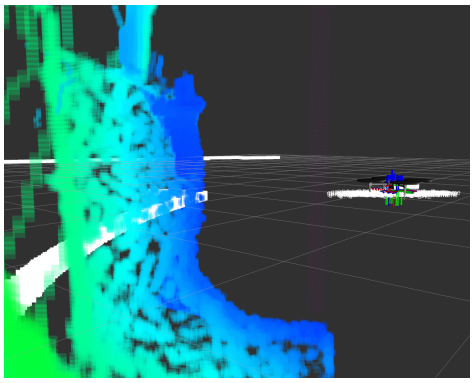
(b) Stereo SGM with FGS



(c) Optical Fused with PCL



(d) Third person view of point cloud in RViZ plugin



(e) Third person side view of point cloud in RViZ plugin

Figure 4.18: Point cloud generation from stereo matching estimated depth m.

Despite the variety of PCL library in 3D processing algorithms, which are dedicated for point cloud data, there were processing limitations due to *Matrice* system specifications by keeping the computations in real-time. Nevertheless, offline experiments have been made on generated point cloud data, and thus we created an offline filtering node, which can exclude non-ground data measurements from the total point cloud. The suggested node pipeline starts with the statistical outlier removal, for removing sparse outliers from the input point cloud, proceeding with a resampling method of moving least squares to reduce data irregularities (small distance measurement errors), that where unable to be removed by the previous statistic analysis, and finally by applying a progressive morphological filtering method, to separate ground and non-ground point data.

However, as we mentioned before this method is not applied in our current real-time system, but instead a lower in computational requirements octree-based method is proposed and used, both for point cloud filtering and 3D mapping.

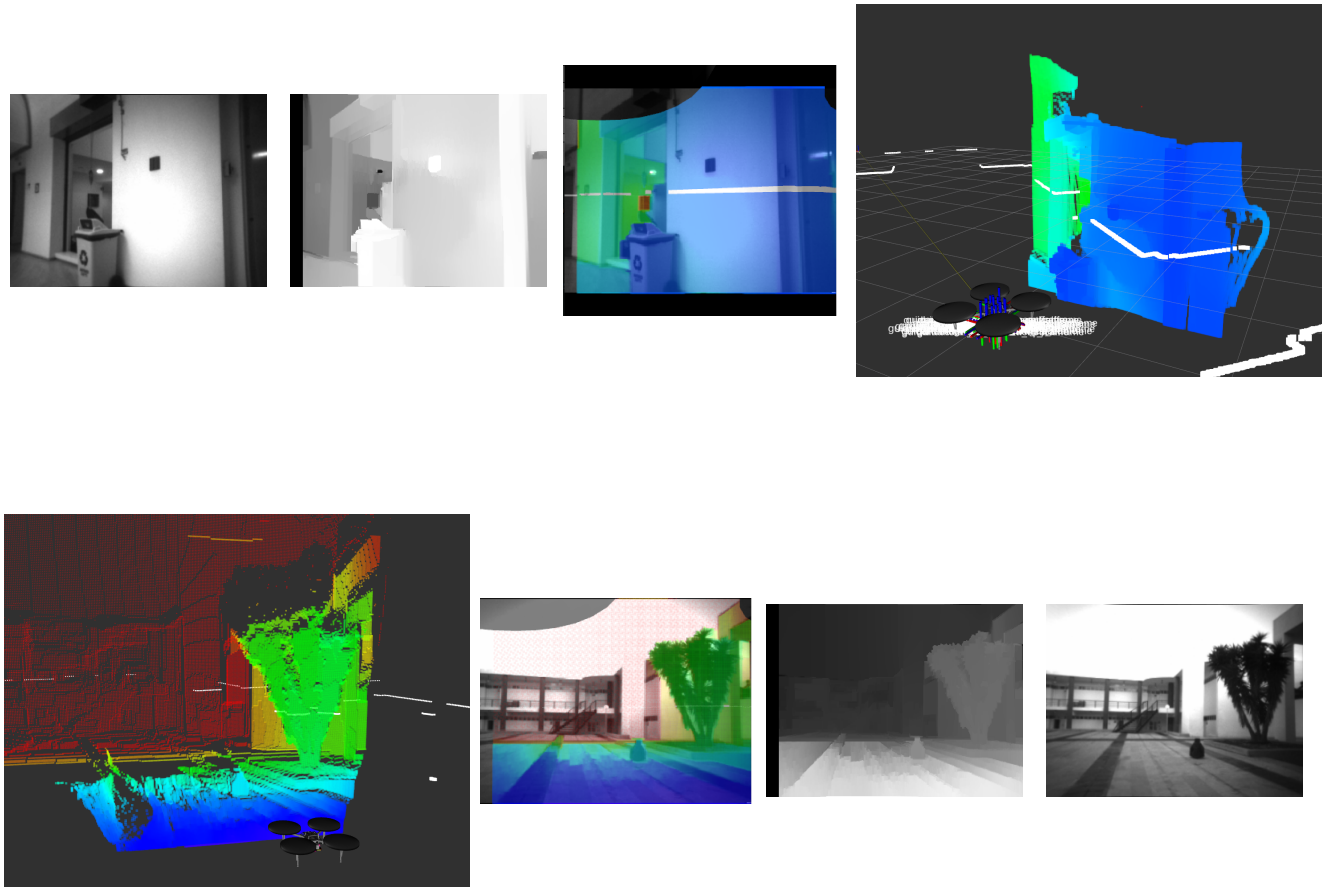


Figure 4.20: Indoor and Outdoor point cloud generation.

Since we obtain spatial information in metric units from the quadcopter's surrounding 3D world, there was a need of applying a 3D mapping algorithm, to construct a 3D map and estimate the optimal navigation path for solving the search and rescue scenario. However, the point clouds store large amounts of measurement points and hence are not memory efficient, especially for an on-board solution. Thus, the OctoMap framework [33] was the ideal choice for the 3D map construction, as it uses a probabilistic approach in point cloud data processing with respect to runtime and memory usage. Basically, it performs probabilistic occupancy updates in for every 3D point measurement, as it estimates the occupied and unoccupied space through ray casting along the sensor origin to the end point of a 3D measurement.

The occupied, as well the unoccupied, space is spatially expressed with the octree data structure (shown in Figure 4.21). Each node in an octree represents the space contained in a cubic volume, namely, the voxel. The occupation probability of a voxel n, given a sensor measurement $z_{1:t}$, is estimated through as a binary Bayes filter as follows,

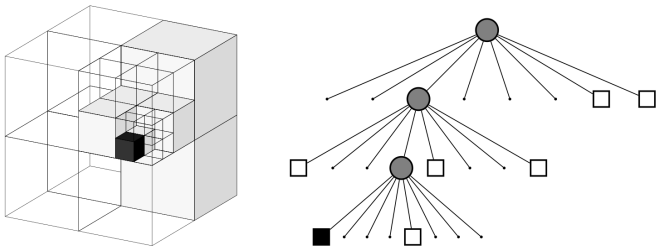


Figure 4.21: The volumetric and tree representation of a octree, that stores free (shaded white) and occupied (black) cells.

$$P(n|z_{1:t}) = \left[1 + \frac{1 - P(n|z_t)}{P(n|z_t)} \cdot \frac{1 - P(n|z_{1:t-1})}{P(n|z_{1:t-1})} \cdot \frac{P(n)}{1 - P(n)}\right]^{-1} \tag{4.9}$$

where $P(n)$ is the prior probability, $P(n|z_{1:t})$ the previous estimate and z_t the current measurement. The update rule (4.9), by assuming uniform prior probability, can be written in log-odds formulation as,

$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + L(n|z_t) \tag{4.10}$$

This formulation allows faster updates due to multiplication replacement with additions. Thus, the information of log-odds values, that are assigned to voxels, are saved and their probabilities can be extracted with,

$$L(n) = log\left[\frac{P(n)}{1 - P(n)}\right] \tag{4.11}$$

Furthermore, in dynamic environments during a navigation process, OctoMap applies a clamping update policy on equation 4.10, by which achieves high adaptability to changes during the mapping by keeping the confidence of the map bounded. So, the occupancy estimates are updated according to,

$$L(n|z_{1:t}) = max\Big(min\big(L(n|z_{1:t-1}) + L(n|z_t), l_{max}\big), l_{min}\Big) \text{ , where } L(n) \in [l_{min}, l_{max}] \tag{4.12}$$

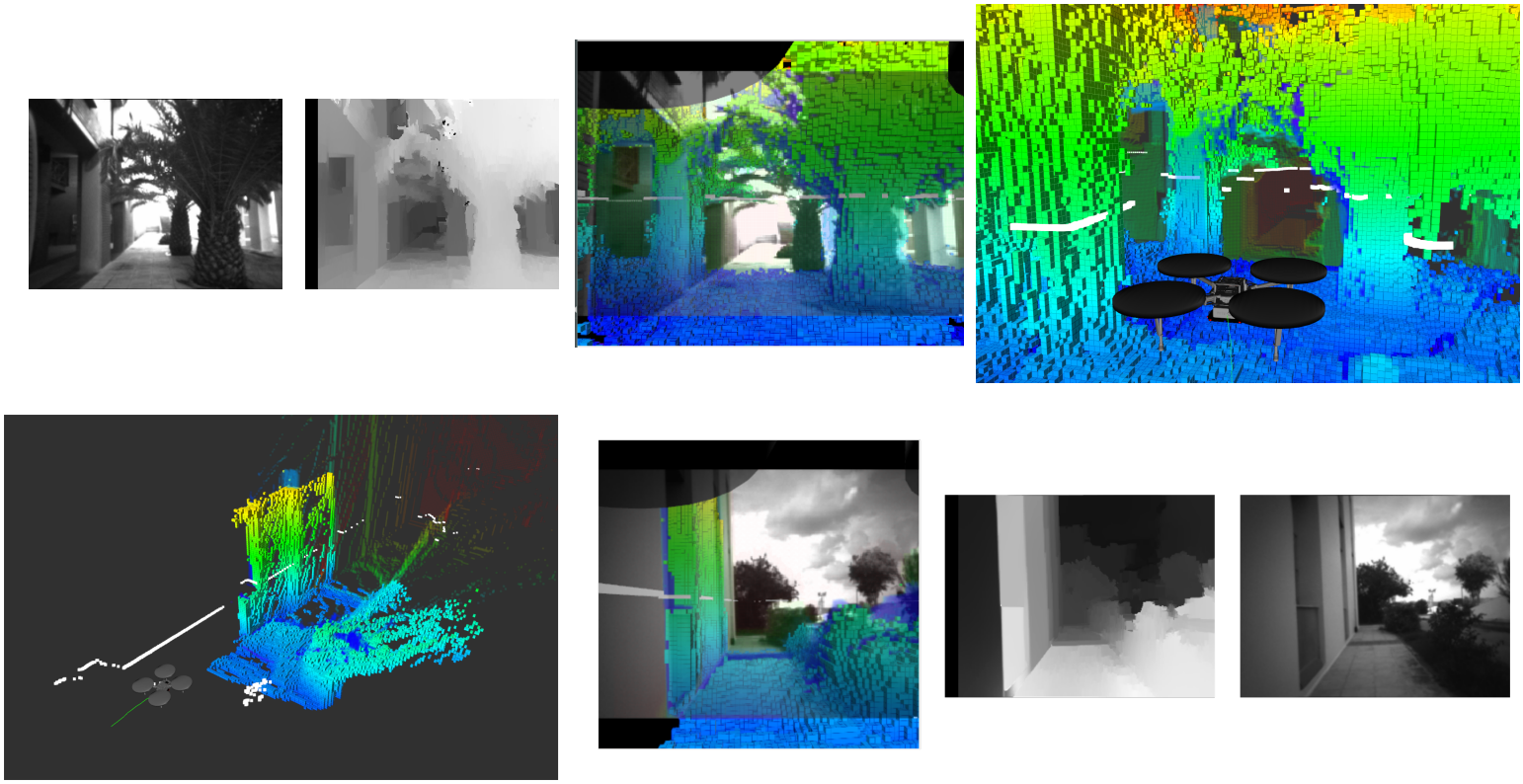


Figure 4.23: Optical feed to point cloud to octree mapping pipeline. The height is visualized by a color coding and the white line (spheres) illustrates the LiDAR scan endpoints.

Henceforth, the OctoMap framework was integrated to the total ROS system and tested under various indoor and outdoor mapping scenarios. Since OctoMap can be used with any kind of distance sensor, as long as the inverse sensor model is available, we employed a beam-based inverse sensor model based on the front stereo camera system. Consequently, we assume that the left camera of the front stereo camera system with its extracted point cloud (expressed in its optical frame) is considered as an 3D LiDAR sensor with its 3D laser scan point cloud. Specifically, given the extracted point cloud from the front cameras' stereo matching process, we assume that each calculated 3D point measurement corresponds to an obstacle surface and the line of sight between the sensor origin (left camera optical frame origin) and the endpoint does not contain any obstacles. As a result, we used the proposed 3D variant of the Bresenham algorithm for formulating the volume raycasting process and approximate the beam. As the volumes

are updated along the beam according to 4.10, we use the following inverse sensor model,

$$L(n|z_t) = \begin{cases} l_{occupied} & \text{if beam is reflected during the volume raycasting} \\ l_{free} & \text{if beam traversed volume} \end{cases} \quad (4.13)$$

The log-odds values of $l_{occupied}$ and l_{free} are set as proposed by default, to 0.85 and -0.4 respectively, implying $P(occupied) = 0.7$ and $P(free) = 0.4$. Instead, the clamping thresholds are specifically adjusted to achieve better map compression in the trade-off map confidence, for keeping runtime to real-time levels. Since these thresholds provide a lower and upper bound for the occupancy probability, information close to P=0 and P=1 is lost compared to the full map without no clamping. Armin H. *et al.*, concluded to the clamping range of $[l_{min} : l_{max}] = [-2 : 3.5]$ for occupancy updates in equation 4.12, by using a 3D LiDAR sensor and performing offline the octree mapping. On the contrary, throughout our experiments we chose to work with the probabilities clamping range of $[0.25 : 0.75]$, i.e $([l_{min} : l_{max}] = [-1.1 : 1.1])$, as stronger compression was achieved without endangering losing map confidence, and thus inserting valuable non-ground information to the final map.

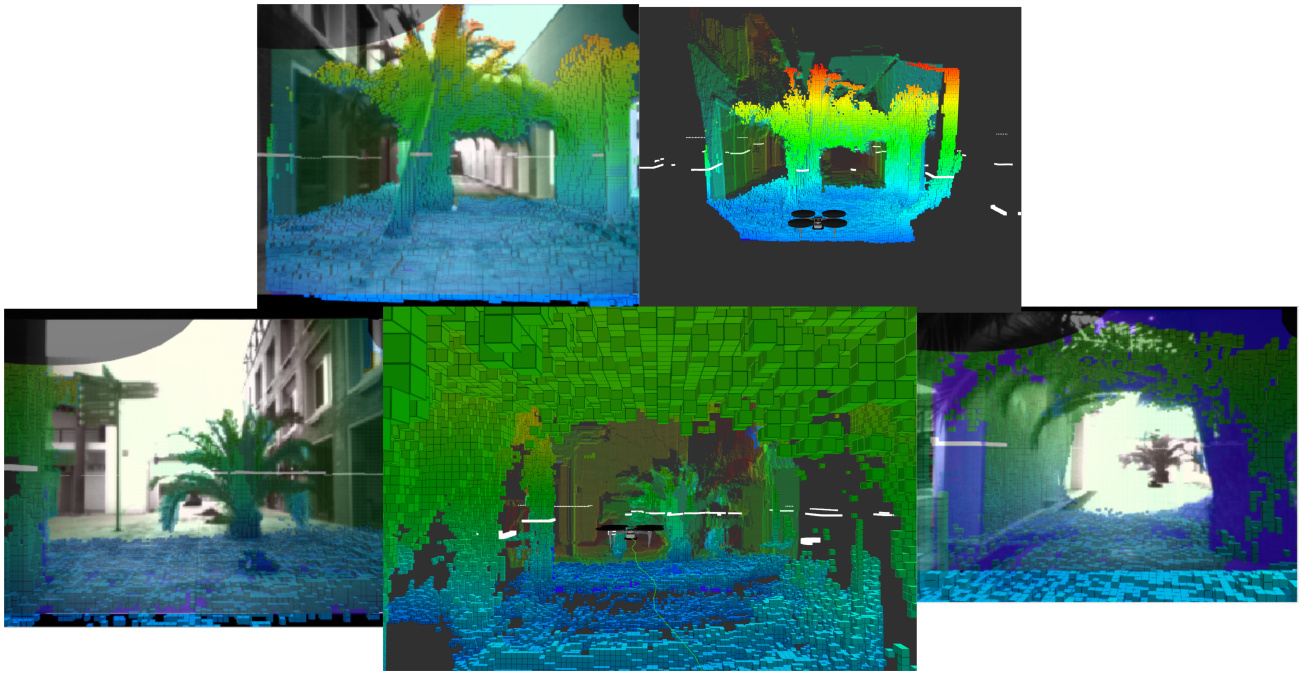


Figure 4.24: More outdoor octree mapping scenarios.

Although, OctoMap framework does not contain an integrated localization approach, so it needs registered sensors poses in the map frame, to construct the total 3D area model. Thus, we use a robust SLAM approach to provide the quadcopter's poses in the map coordinate system and also construct a 2D map, based on LiDAR sensor scan data. Those resulted maps, the 3D based on the stereo camera's depth point clouds and the 2D based on LiDAR scans, are going to be combined to produce a 2.5D map that contains the SLAM outcome map belief fused with possible obstacle positions, in fixed height, as resulted from OctoMap process. More information will be given in the following section.

Figures in 4.23, 4.24, 4.26 and **??** show re-played quadcopter flights to illustrate the corresponding 3D octree model, at the current time. As it clear, the quadcopter obtain a satisfactory 3D world representation belief and can combine it the concurrent generated laser-scan map, that is generated from the SLAM process. As long the OctoMap subscribes to the registered map poses of the quadcopter, generated by the SLAM system, the final maps are aligned and can be fused to extract the dedicated 2.5D map for the humanoid robot.

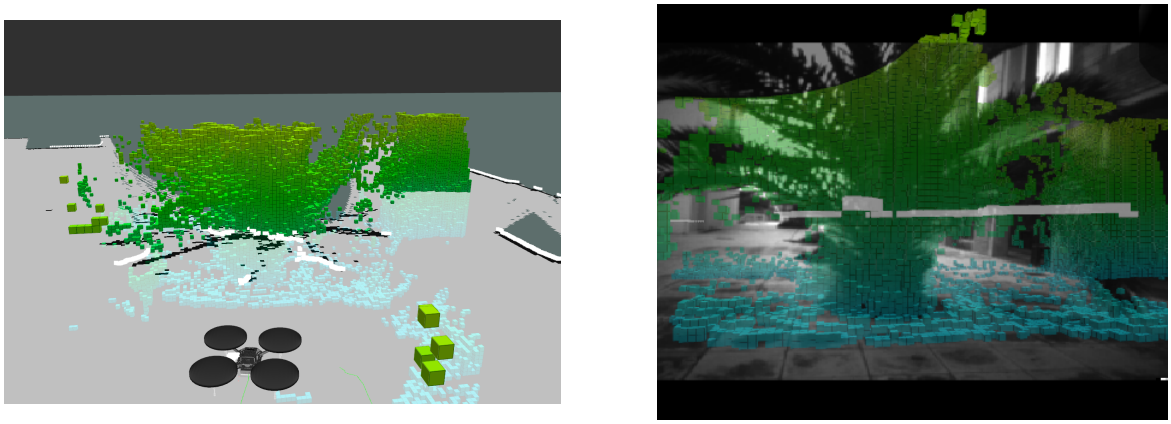


Figure 4.25: Fusing both 3D octree map and 2D SLAM map.

As mentioned before, the quadcopter applies map compression through the clamping update policy defined at 4.12. Whenever the log-odds value of a voxel reaches either l_{min} or l_{max} values, the node is considered as stable. Furthermore, a node pruning method is also implemented in the OctoMap framework, in which if all children of an inner tree node are stable leaf nodes with the same occupancy state, then they are pruned. On the other hand, if a new measurement contradicts with the state of the corresponding inner node, then its children are regenerated and updated accordingly. By combining

the two compression methods, of octree pruning and the clamping update policy, the quadcopter achieve sufficient map compression and less memory requirements (i.e for an area of 10x10x5 the quadcopter produces a 5MB map model,at 5cm map resolution).

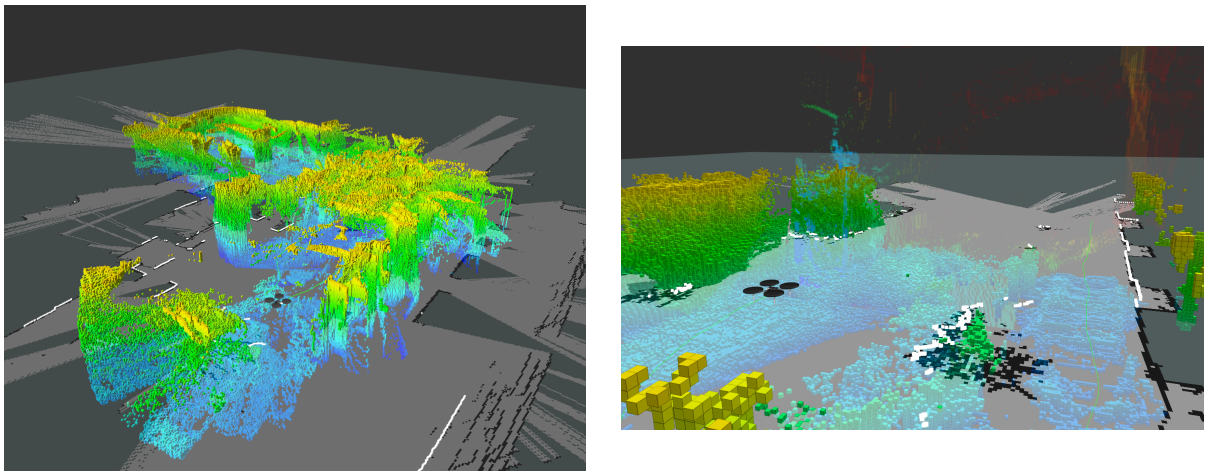


Figure 4.26: Fusing both 3D octree map and 2D SLAM map.

Lastly, for further memory consumption reduction the lossy maximum-likelihood compression was tested, as it converts each node to be either completely free or occupied, but the spatial information levels were not sufficient.

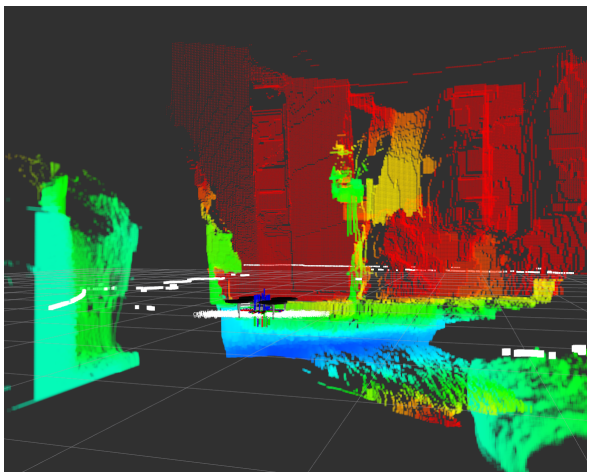


Figure 4.27: Point cloud depth data extracted from front, left an right *Guidance* stereo cameras.

Apart from the aforementioned, a dedicated ROS node was implemented to concatenate point clouds computed from the front, left and right stereo camera system, into

a single one, relatively to the left front camera's coordinate system (Figure 4.27). The purpose of creating this node was to insert multi-point cloud information in the OctoMap server to construct a more detailed 3D model, by using information from all the front facing cameras simultaneously, however it also raised the computational needs and memory consumption. Hence, to achieve the real-time 3D octree mapping, only the front stereo camera's point cloud data are used.

4.6 Quadcopter SLAM and Fusion with OctoMap Framework

The abilities of real-time self-localization and map generation of a mobile robot in an unknown area, are crucial to be able to operate in real world unstructed environments. Additionally, in Search and Rescue scenarios, time is an important fact to consider, as individuals could be in danger and need immediate assistance. Thus, in our approach, an aerial vehicle is used (the *Matrice* quadcopter) as it can move fast compared to ground robots, to percept and construct a 3D world representation and pass it to a ground robot to make it start the exploring and searching for victims.

As mentioned in the previous section, we use the OctoMap 3D mapping method to percept the surrounding 3D world environment. Although, OctoMap, as a mapping method, needs the robot's pose information to insert new 3D measurements into the total 3D occupancy map. Hence, we use a SLAM approach to provide the proper pose information in the constructed 2D map, informing the OctoMap node with the robot's current state in the map frame.

Specifically, we use the Hector SLAM system [22] implementation, which provides full 6DoF robot pose estimation by also solving the Online SLAM problem (2.5) reliably. The Hector system combines two subsystems, a 2D SLAM system based on the integration of laser scans in a planar map and an integrated 3D navigation system based on inertial measurement unit (system overview in Figure 4.28). Both

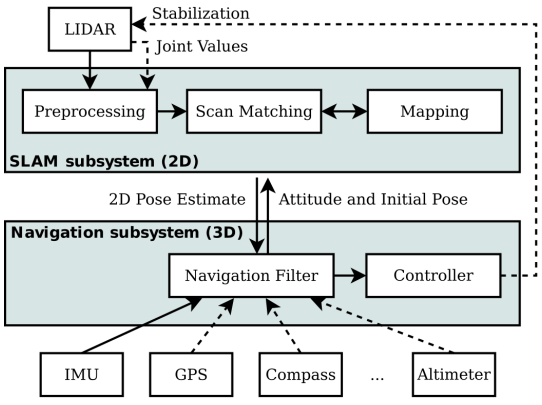


Figure 4.28: Hector SLAM system overview.

systems are updated individually and are loosely coupled so that they remain synchronized over time.

By defining the navigation coordinate system as a right-handed system by the rule of thumb, the robot full 3D state is expressed by its 3D position and orientation, as defined in (2.1), and its velocity vector $v = (v_x, v_y, v_z)^T$, as,

$$\mathtt{x} = (\theta^T x^T v^T) \tag{4.14}$$

The inertial measurements are expressed by the body-fixed angular rates vector $\omega = (\omega_x, \omega_y, \omega_z)^T$ and the accelerations vector $a = (a_x, a_y, a_z)^T$. In total, they are written as,

$$\mathtt{u} = (\omega^T a^T) \tag{4.15}$$

So, the motion model is described by the following nonlinear differential equation system,

$$\dot{\Theta} = E_\Omega \cdot \omega \tag{4.16}$$

$$\dot{x} = \mathtt{v} \tag{4.17}$$

$$\dot{v} = R_\Omega \cdot a + g \tag{4.18}$$

where g is the gravity vector, R_Ω is the rotation matrix that transforms a vector of the body frame to the map coordinate system, and E_Ω maps the body fixed angular rates to the derivative of the Euler angles.

2D SLAM

During the navigation, the LiDAR system might exhibit 6DoF motion, so its laser scans are transformed through quadcopter pose estimation and the tf tree information, into the quadcopter's stabilized base frame coordinate system. Every laser scan is converted into a point cloud of scan endpoints, expressed in the base frame coordinate system. The implemented LiDAR (Figure 2.8a) data node transforms laser scans data into point cloud form, hence, we filter the outliers to enhance SLAM procedure.

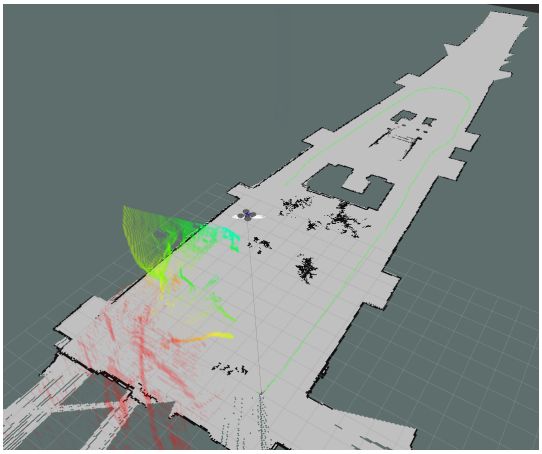


Figure 4.29: Indoor SLAM at South side of ECE Department in TUC.

In addition, Hector SLAM uses an occupancy grid map (explained in Subsection [2.3.2]) to represent the mapping environment. Due to occupancy grid map discrete nature, sub-grid cell approximation can not be performed, as there is limitation to the map resolution parameter. For this reason, an occupancy value and gradient approximation is applied by using the information from the four closest point positions, to achieve accurate estimations of occupancy probabilities and derivatives.

For scan matching procedure, Hector SLAM approach is based on the optimization of scanned endpoints point cloud alignment, on the map frame learnt so far. The approach philosophy is that if the matching is aligned to the existing map, thus the point cloud is aligned with the preceding scans. A Gauss-Newton approach is implemented, which seeks for the rigid transformation $\xi = (x, y, \psi)$ that minimizes,

$$\xi^* = \underset{\xi}{argmin} \sum_{i=1}^{n} \left[1 - M\big(S_i(\xi)\big)\right]^2 \tag{4.19}$$

which finds the transformation of the best laser scan alignment on the map. The term $S_i(\xi)$ gives the coordinates of the laser scan endpoint i expressed in the world coordinate system, taken from pose ξ (as expressed in 2.8). Assuming that the function $M(S_i(\xi))$ gives the map occupancy of the coordinates of $S_i(\xi)$ and given some starting estimate of ξ, to find the best rigid transformation 4.19 we estimate $\Delta\xi$ which optimizes the error

measure according to,

$$\sum_{i=1}^{n} \left[1 - M\big(S(\xi + \Delta\xi)\big)\right]^2 \rightarrow 0 \tag{4.20}$$

Solving for $\Delta\xi$ yields the Gauss-Newton equation for the minimization problem,

$$\Delta\xi = H^{-1} \sum_{i=1}^{n} \left[\nabla(M\big(S_i(\xi)\big)\frac{\partial S_i(\xi)}{\partial \xi}\right]^T \left[1 - M\big(S_i(\xi)\big)\right] \tag{4.21}$$

with,

$$H = \left[\nabla(M\big(S_i(\xi)\big)\frac{\partial S_i(\xi)}{\partial \xi}\right]^T \left[\nabla(M\big(S_i(\xi)\big)\frac{\partial S_i(\xi)}{\partial \xi}\right] \tag{4.22}$$

By using the map gradient $\nabla M(S_i(\xi))$ and $\frac{\partial S_i(\xi)}{\partial \xi}$, the Gauss-Newton 4.21 equation can be evaluated.

Finally, a Gaussian approximation of the match uncertainty is applied, by covariance matrix estimation through Hessian matrix approximation. In specific, the covariance matrix is approximated by,

$$R = Var\{\xi\} = \sigma^2 \cdot H^{-1} \tag{4.23}$$

where σ is a scaling factor dependent on laser scanner device properties.

3D State Estimation

As mentioned before, the 6DoF pose estimation is performed with an Extended Kalman Filter, using the motion model described by the equations system in 4.18. Also, the two Hector subsystems (shown in Figure 4.28) are executing in parallel and exchange information to each other. Denoting the Kalman estimate as at the time of the scan with covariance P and the SLAM pose (ξ^*, R), defined in 4.19 and 4.25, the covariance intersection is written as,

$$P^+ = P - (1-\omega)^{-1} \cdot KCP \quad \text{and} \quad \hat{\mathrm{x}}^+ = \hat{\mathrm{x}} + K(\xi^* - C\hat{\mathrm{x}}) \tag{4.24}$$

where,

$$K = PC^T \left(\frac{1-\omega}{\omega} \cdot R + C^T PC\right)^{-1} \tag{4.25}$$

The Hector SLAM system was tested and evaluated on *Matrice* quadcopter, in a variety of indoor and outdoor environments, as some resulted maps are illustrated on Figures 4.29 and 4.31). The resulted map accuracies through the low computational requirements and real-time runtimes, made Hector SLAM system the ideal Online SLAM approach for our scenario.

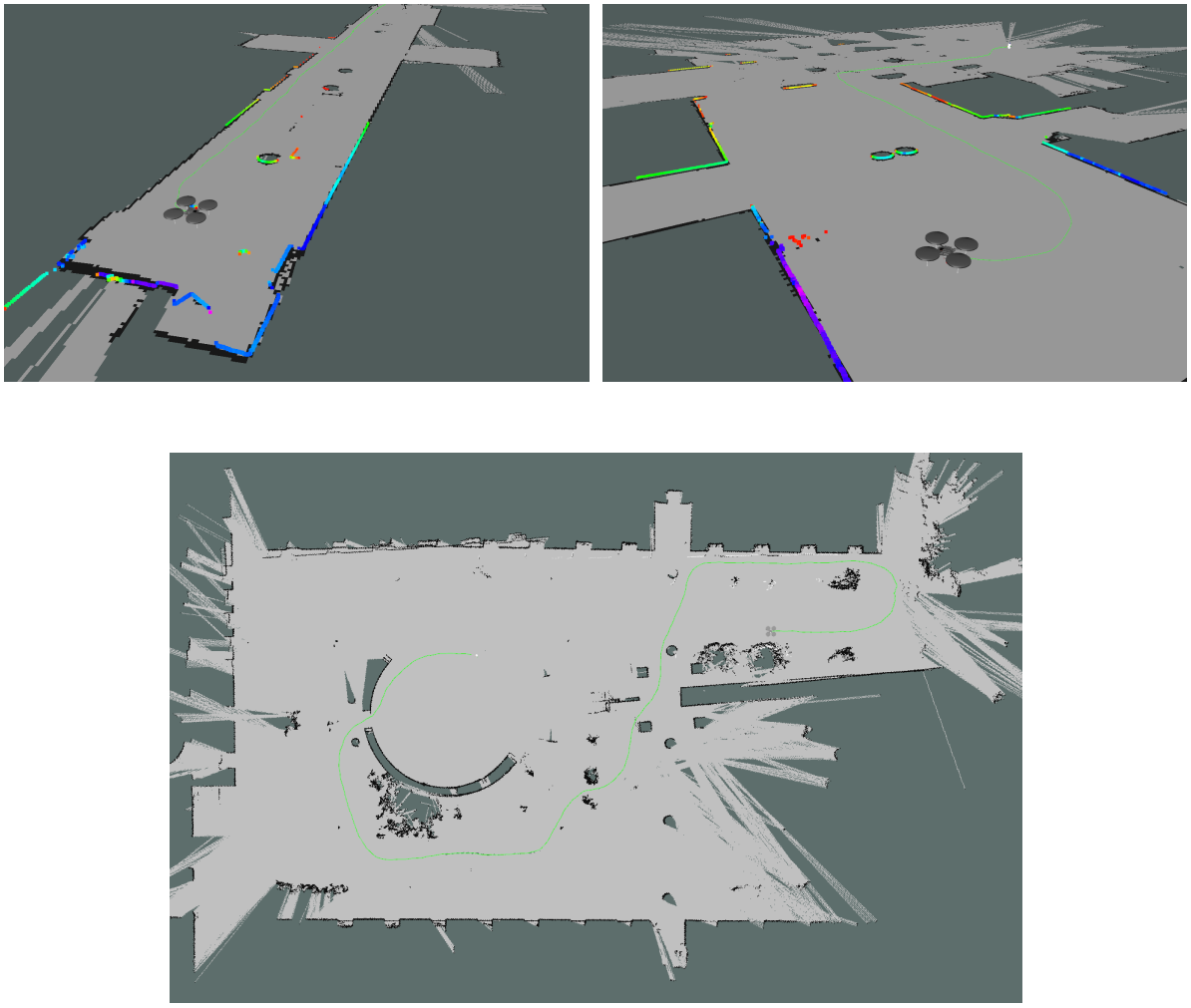


Figure 4.31: Maps learned using Hector SLAM system onboard on *Matrice* quadcopter

4.7 Ground Robot Detection and Initial Localization

During the quadcopter navigation in the unknown environment, while the mapping process is performed, it also searches for for the ground robot in the area. The detection

is done by using an ALVAR[1] AR marker which is placed on a custom made hat on the robot's head (Figure 4.32). An implemented ROS node subscribes on the optical footage of the pointing-down *Guidance* camera system and the performs marker detection to locate the humanoid robot.

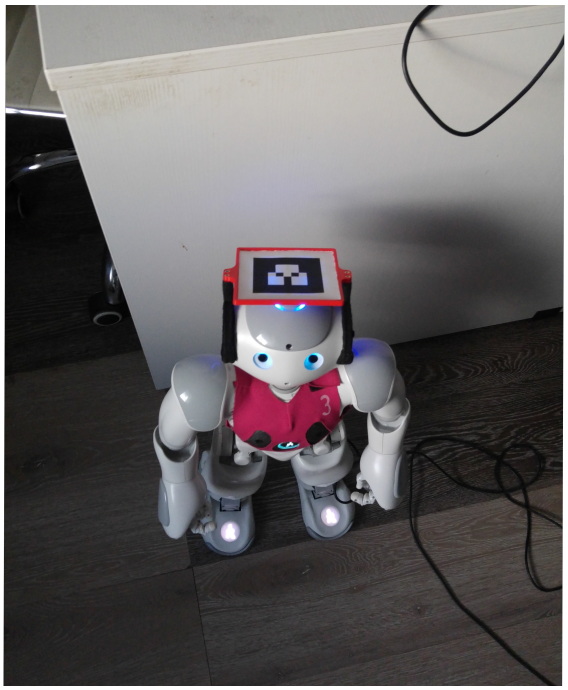


Figure 4.32: Humanoid robot Nao equipped with an ALVAR marker.

The ALVAR library is an open source Augmented Reality (AR) tag tracking library, which is mostly used in AR applications, as it provides an efficient and robust marker tracking implementation. It uses image filters and feature extraction algorithms to locate the marker in the image plane and performs Kalman filtering to estimate its pose relative to the camera frame.

In specific, the marker detector converts the image to bitonal (black and white) format using an adaptive threshold and searches for the image edges to extract lines. Then it detects all the occurred quadrangle shaped areas and checks if they represent a known marker.

Location and orientation approximation is done by applying homography transformations and Kalman estimation, and can provide centimeter-level coordinates. As a 320x240

[1] http://virtual.vtt.fi/virtual/proj2/multimedia/index.html

resolution camera is used, a 10cm/size square marker can be successfully recognized in the upper limit distance of 2m and viewing angle of 20-30 degrees or more. Also, according to the ARTootKit accuracy measurements in [34] and also from our experiments, the maximum error values in position estimation are $\sim \pm 20mm$ taken from target distance of $\sim 2m$, which is more than satisfactory for our initial robot localization purpose.

Particularly, the estimated marker pose given from the ALVAR ROS node is expressed in camera optical frame coordinate system by which is detected from. Also, the marker is placed on the robot head, thus its pose is matched with the robot's head frame pose. At the same time as the quadcopter maps the area it localizes itself in it, so, when the humanoid robot is located its pose in map coordinate system can be determined. In specific, a point P in detected camera optical frame can be transformed/expressed to map frame, as follows,

$$
\begin{aligned}
{}^{map}P = {}^{map}_{M100\_base}T \cdot {}^{M100\_base}_{stereo\_camera}T \cdot {}^{stereo\_camera}_{left\_camera}T \cdot {}^{left\_camera}_{left\_optical}T \cdot {}^{left\_optical}_{marker}T \cdot {}^{marker}P \\
\Leftrightarrow {}^{map}P = {}^{map}_{marker}T \cdot {}^{marker}P
\end{aligned}
$$

As long as, the *marker* frame is aligned with the humanoid robot's *head* frame (the marker is on its head), we can assume the robot's head pose in relation to the quadcopter's body. Thus, by extracting the transformation of robot's *head* frame to its *base_link* frame (${}^{base\_link}_{head}T$), we can finally approximate humanoid body pose in relation to quadcopter's pose.

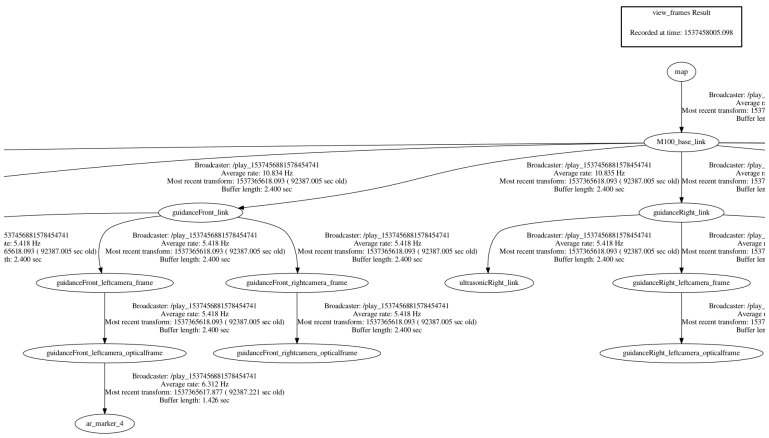


Figure 4.33: Detected ALVAR marker in front left quadcopter's camera optical frame.

As a result, the quadcopter publishes the viewed pose of the humanoid robot, relative to its body frame, in map coordinates by applying the inverse transformation of ${}^{base\_link}_{head}T$ and then the ${}^{map}_{marker}T$. By this transformation, we have the robot pose expressed in map frame. Although, since we are projecting the robot pose on the planar map coordinate system, we acquire only its x , y and θ values. Figure 4.34 show the marker's pose projection on the map's planar frame. Not to mention, the marker pose is rotated 90^o around Z-axis, to be aligned with the robot heading direction.

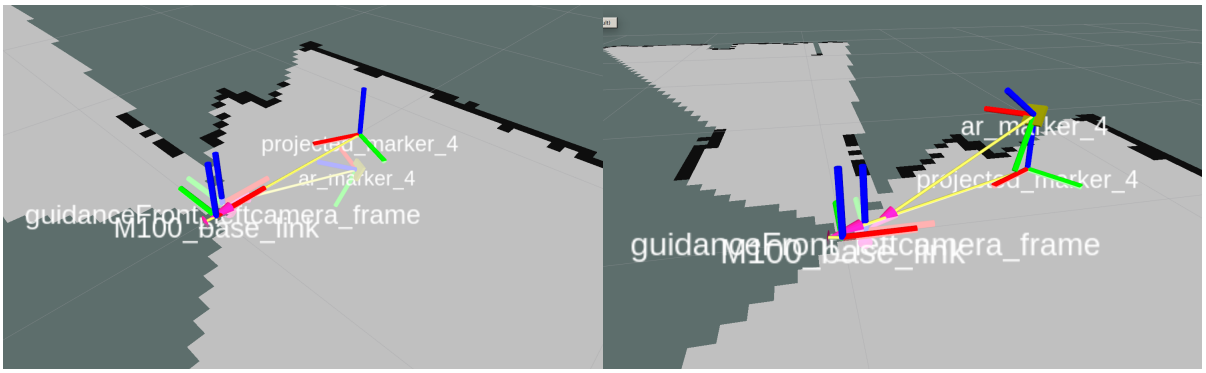


Figure 4.34: *AR_Marker_4* pose projection on the map frame.

The humanoid robot's initial pose is published as a *geometry_msgs/PoseWithCovariance Stamped* ROS message, consisted by the current timestamp, the estimated robot pose (position:(x, y, z) and orientation in quaternion form:(x, y, z, w)) in the map coordinate system and a 6x6 covariance matrix initialized by a Gaussian distribution. Figure 4.35 shows the steps initializing Nao pose in the map coordinate system, as discussed above.

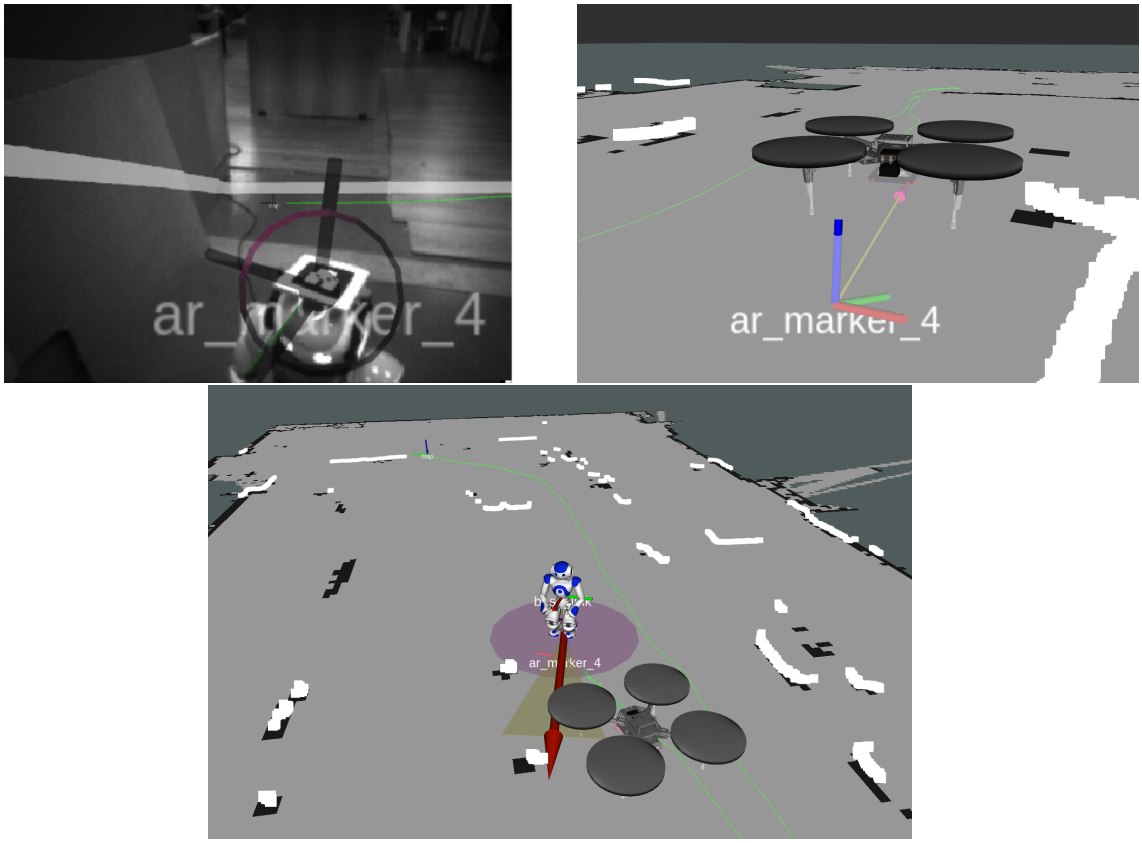


Figure 4.35: Locating and initializing Nao pose in the map frame.

4.8 Laser-scan Transformation and adaptive Monte Carlo Localization on the Ground Robot

Since we have considered that the ground robot is not equipped with any perception sensor, despite its odometry data there was a need for aerial assistance through the quadcopter vision to help it with the localization. During the humanoid robot navigation under complicated paths, translational and rotational errors may accrue, and thus its final pose estimation distribution is spreaded due to high uncertainty. Consequently, the odometry data aren't sufficient for accurate localizing the robot in the given map.

For this reason, a dedicated ROS node was implemented to correct the ground robot's estimated position during its navigation, by informing the localization procedure for its pose, as perceived from the air (quadcopter), in the current map. The used localization

approach for the ground robot is the adaptive Monte Carlo Localization (KLD-sampling) approach, which uses a particle filter to track the pose of a robot against a known map. Hence, this ROS node transforms the quadcopter laser scans to humanoid robot *base_link* frame, to be included in MCL posterior estimates. Given the relative poses of the two robots through the tf tree, the quadcopter's laser scan endpoints are spatial transformed from the *laser* frame to the humanoid robot *base_link* frame, through detected AR *marker* frame.

Specifically, as long as the LiDAR scans are generated in respect to quadcopter's *laser* frame, they are performed in a circular scan around the sensor origin point (as shown in Figure 2.9). By transforming those scan endpoints in another frame in the same plane (basically defining a new sensor position and orientation) we must sample those points as they are scanned from the new pose. So, a dedicated node was implemented for this procedure. The Figure 4.37 shows the transformation of the transfered laser scan range data, depending the Nao head's pose in respect to the *Matrice* body frame.

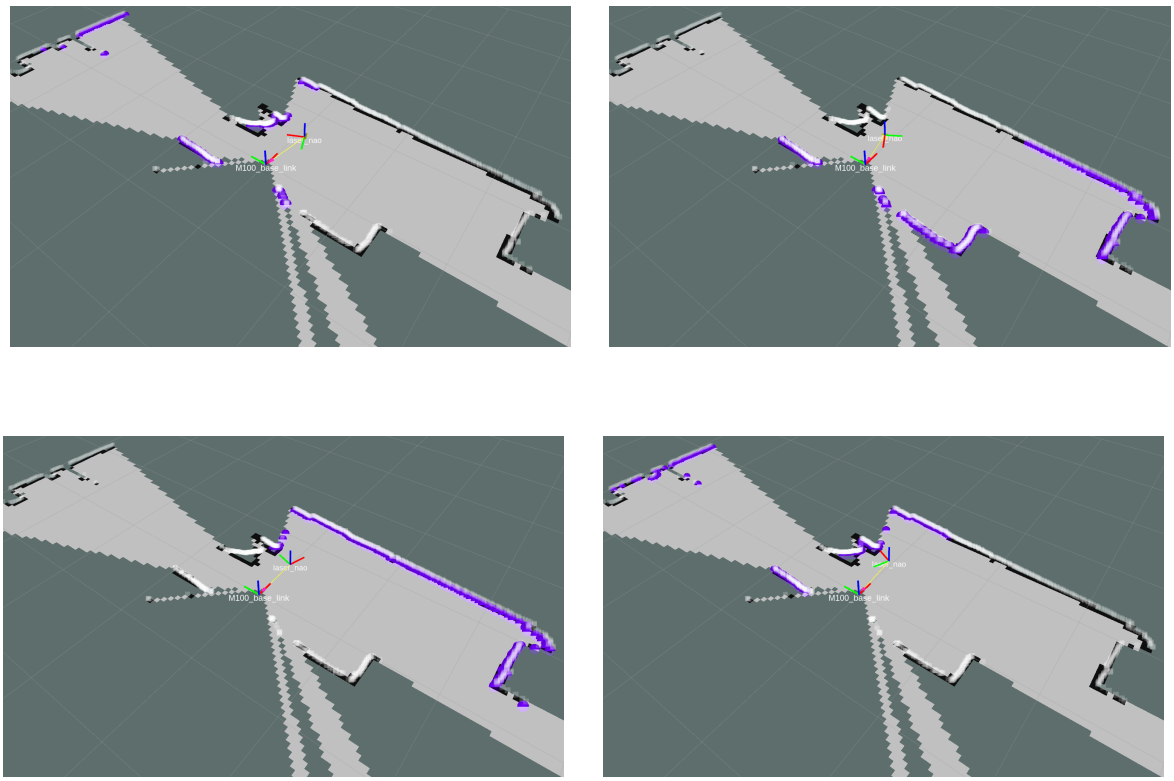


Figure 4.37: Nao transfered laser scan endpoints, depending on its pose relative to *Matrice*.

Particularly, this node express those scan endpoints to point cloud form and then transforms them to humanoid robot's *base_link* frame to obtain their positions relative to its body. After that, by assuming the LiDAR sensor (2.8a) angular resolution and measurement step, samples on 360^o detection angle and filters out any overlaid points, relative to the new robot pose origin. Hence, the resulted filtered point cloud is expressed back in $sensor\_msgs/LaserScan$ ROS message form and successfully adapted on the humanoid robot system. Given that, the humanoid robot could integrate the dedicated transformed laser scans to the MCL algorithm and enhance its localization process.

Adaptive (KLD-sampling) Monte Carlo Localization

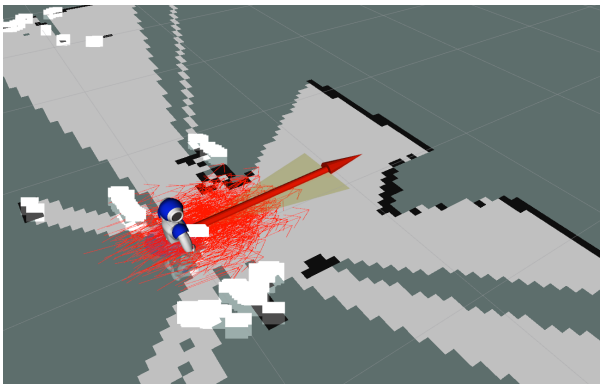


Figure 4.38: Nao localization with KLD-sampling MCL approach.

As mentioned before, the used localization approach is the KLD-sampling [35] which is a variant of Monte Carlo particle filtering localization method. Basically, KLD-sampling adapts the number of particles, depending on a statistical bound on the sample-based approximation quality. In particular, at each iteration of particle filter, KLD-sampling determines the number of samples such that, with probability $1 - \delta$, the error between the true posterior and the sample-based approximation is less than ϵ. This adaptive method has been tested in many mobile robot localization scenarios and yielded drastic improvements compared to particle filters with fixed sample set sizes, and thus is selected for our humanoid localization approach.

Firstly, in each particle filter iteration the KLD-sampling approach (2) defines and initializes a histogram grid. This histogram represents the volume of the state space that is covered by particles, and thus determines the statistical bound to make the KLD-sampling stop generating particles. As a result, the algorithm adapts the number of particles depending on the pose current uncertainty, by increasing the bound if more particles are needed. This happens mostly on the early stages of the localization, as the particles are widespread all over the map. Then, as the particles are concentrated on a smaller number of different bins, the KLD-sampling generates less samples. So the KLD-sampling algorithm adaptation to particle production, both outperforms MCL fixed sample set sizes approaches and offers a lower in computational complexity approach.

Algorithm 2

KLD_Sampling_MCL(X_{t-1},u_t,z_t,m,ϵ,δ)

1: $X_t = \emptyset$
2: $M = 0, M_X = \infty, k = 0$
3: **for** all k in H **do**
4: $\quad b = empty$
5: **end for**
6: **repeat**
7: $\quad$ draw i with probability $\propto w_{t-1}^i$
8: $\quad x_t^{[M]} =$ **sample_motion_model**$(u_t, x_{t-1}^{[i]})$
9: $\quad w_t^{[M]} =$ **measurement_model**$(z_t, x_t^{[M]}, m)$
10: $\quad X_t = X_t + \left\langle x_t^{[M]}, w_t^{[M]} \right\rangle$
11: $\quad$ **if** $x_t^{[M]}$ falls into empty bin b **then**
12: $\quad\quad k = k + 1$
13: $\quad\quad b = non - empty$
14: $\quad\quad$ **if** $k > 1$ **then**
15: $\quad\quad\quad M_x := \frac{k-1}{2\epsilon}\left\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-2}}z_{1-\delta}\right\}^3$
16: $\quad\quad$ **end if**
17: $\quad$ **end if**
18: $\quad M = M + 1$
19: **until** $M < M_X$
20: **return** X_t

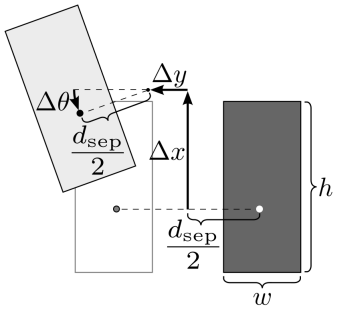
Hence, we have integrated the adaptive MCL algorithm to our approach and tested it various situations (Figures in 11). The error bounds parameters of $1-\delta$ and ϵ, as defined before, are set as proposed in [36] to 0.99 and 0.01, respectively.

4.9 Path Planning and Ground Robot Navigation with a Footstep Planner

As long as we have obtained a sufficient representation of the surround environment on a occupancy grid map and also localized the humanoid robot in it, next step is the exploration and the search for humans. In our scenario, as we considered that the ground robot can't take spatial measurements to localize itself, likewise, we assume that the quadcopter can't perform human detection. Thus, we perform human detection method

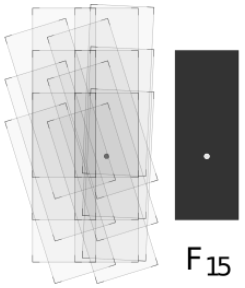
(discussed in more detail in following chapter [4.10]), through the humanoid robot's camera footage, during its exploration in the mapped area.

For this reason, as the ground robot pose is determined in the map frame, we need a path planning method to estimate an optimal path for it to follow and start searching for individuals. Hence, we used the footstep planner [37] of the humanoid ROS navigation stack, which provides an anytime search-based path planning approach for humanoid robots (including parameterization for Nao robot), to estimate the ideal footstep path given the robot's current position. In specific, this path planning approach implements the R^* search algorithm, which is a A^* variant, with an Euclidean heuristic function to extract optimal paths. The R^* algorithm combines weighted $A*$ searches in a local deterministic search with a randomized component, and thus provides speed-up to the search procedure.



Parameter	Value / Range
d_{sep}	9.5 cm
w	9.0 cm
h	16.0 cm
Δx	$[-4.0, 4.0]$ cm
Δy	$[-1.0, 4.0]$ cm
$\Delta\theta$	$[-2.9, 20.0]^\circ$

(a) The footstep model and the parameterization for the humanoid robot Nao.



(b) Nao footsteps variety relative to initial (right) foot position.

Figure 4.39: Footstep planner ROS node parameterization for the humanoid robot Nao.

The Nao's framework, namely NAOqi, provides a walking engine that uses a linear inverted pendulum model to generate stable gaits. The walking engine, despite the direct omnidirectional velocity control for commanding the Nao to walk to a target position, it features also a *FootStep* command functionality. The *FootStep* action is parameterized by a displacement of the moving foot relative to the stance foot, defined by the vector $a = (\Delta x, \Delta y, \Delta\theta)$ (as shown in Figure 4.38a). Thus, assuming that we have established the connection between the NAOqi and the ROS master process through the *ros-naoqi* package, we implemented a ROS node that keeps the Nao robot in rest mode until its initial pose is estimated and published by the quadcopter and initializes the walking

engine *FootStep* ROS service, to be able to listen requested *FootStep* messages (in form of *humanoid_nav_msgs/StepTarget* message) from the planner node.

So, given the AMCL pose of the humanoid robot and the current map status, the footstep planner node estimates the next step state. By expanding a current foot pose state s, the planner determines all successor states s' by applying all the transitions of all available actions (as pictured in Figure 4.38b). By denoting an step transition action as a and s, s' the current and next footstep state, respectively, the estimated transition costs are given by,

A humanoid_nav_msgs /StepTarget ROS Message

uint8 $right = 0$
uint8 $left = 1$

geometry_msgs/Pose2D *pose*
  uint32 x
  time y
  string *theta*
uint8 *leg*

$$c(s, s') = ||(x, y), (x', y')|| + k \tag{4.26}$$

where $s = (x, y, \theta)$, $s' = (x', y', \theta')$ and k the constant costs of making a step transitions. Thus, the iterative construction of states (x, y, θ) connected by footsteps builds a lattice graph, where the single footstep actions correspond to the motion primitives used in search-based planning. On this lattice graph, search methods like $A*$ or its variants can be applied, to give the optimal path. Briefly, the $A*$ search expands states according to the following evaluation function,

$$f(s) = g(s) + h(s), \text{ where } \forall s : h(s) \leq c(s, goal) \tag{4.27}$$

The function $g(s)$ gives the costs of the best path from the starting state to the current state s and the function $h(s)$ is the heuristic function which provides the estimated costs to the *goal* from the state s. Also, each estimated cost to the *goal* from a current state s, must be smaller or equal that the true cost of traversing from s to *goal*. On the other hand, the weighted $A*$ approach multiplies the heuristic function with a weight factor and achieves a trade-off of optimality and speed.

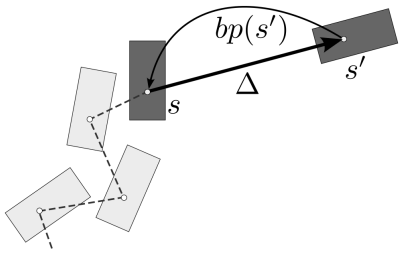
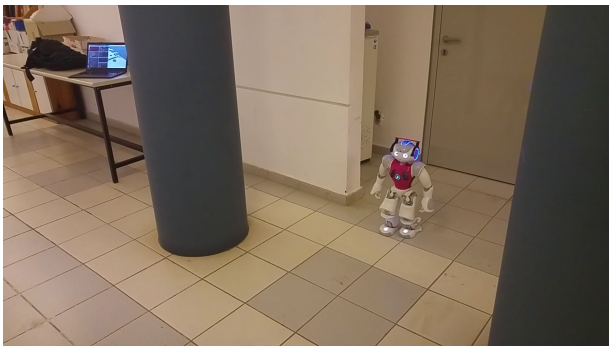
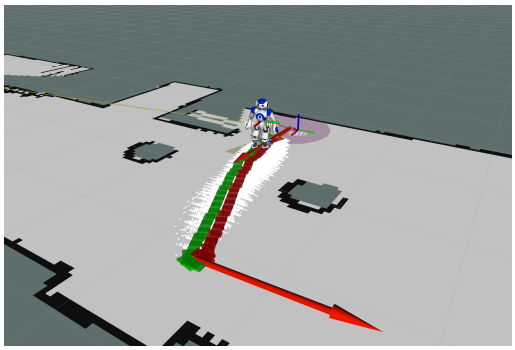


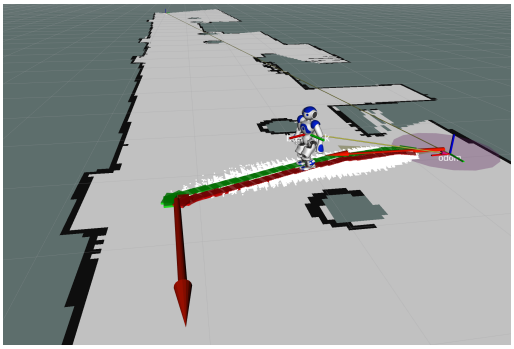
Figure 4.40: $R*$ state expansion.

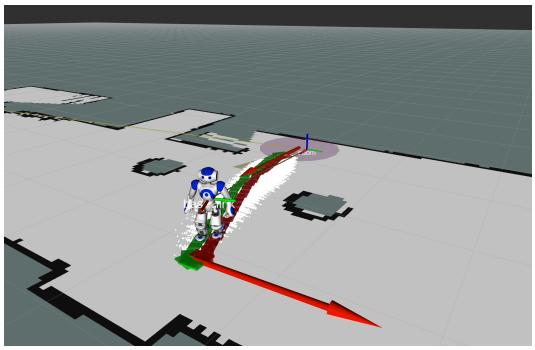
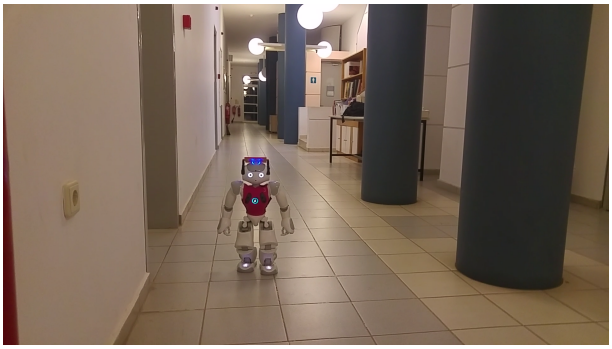
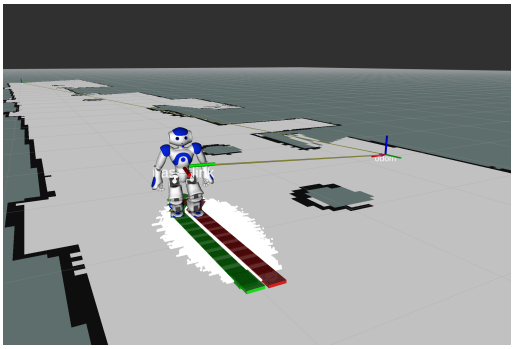
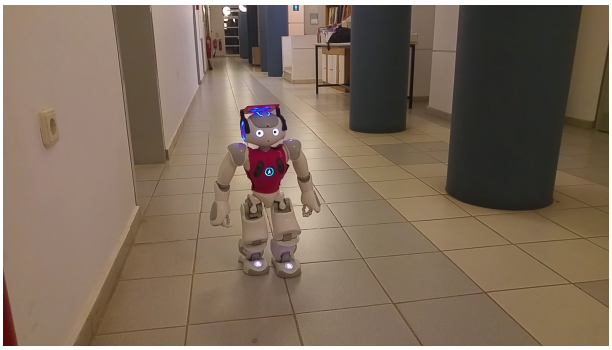
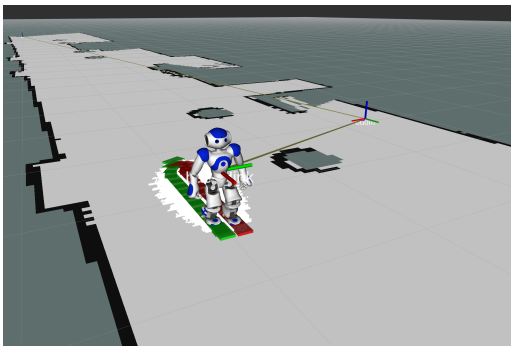
Nevertheless, the used search approach of randomized $A*$ ($R*$), depend less on the quality of the heuristic function. Compared with the $A*$ approach , this search avoids local minima situations, as it runs a series of short-range weighted $A*$ searches towards randomly chosen sub-goals. Particularly, the $R*$ constructs a sparse state graph Γ in the same way as the lattice graph is expanded.

At each iteration, k-successors are determined in a random direction at distance Δ (Figure 4.39). If the successor state s' is feasible (collision-free) it is added to Γ graph with the edge of $s \rightarrow s'$ with the pointer of its predecessor $bp(s') = s$. Each edge in Γ corresponds to a path in the original lattice graph that is determined with by a local search with weighted $A*$. This approach yields that the R* will avoid local minima situations and will provide fast solutions even with the Euclidean heuristic. Therefore, it was tested through the footstep planner node in different world situations and provided efficient paths for the humanoid robot to follow. Below on Figure 4.45, an example of Nao's footstep navigation procedure is shown in parallel to the real world state. As the map was known, the humanoid robot was prepared to receive goal positions and estimate optimal paths through the footstep planner node to begin with the navigation.





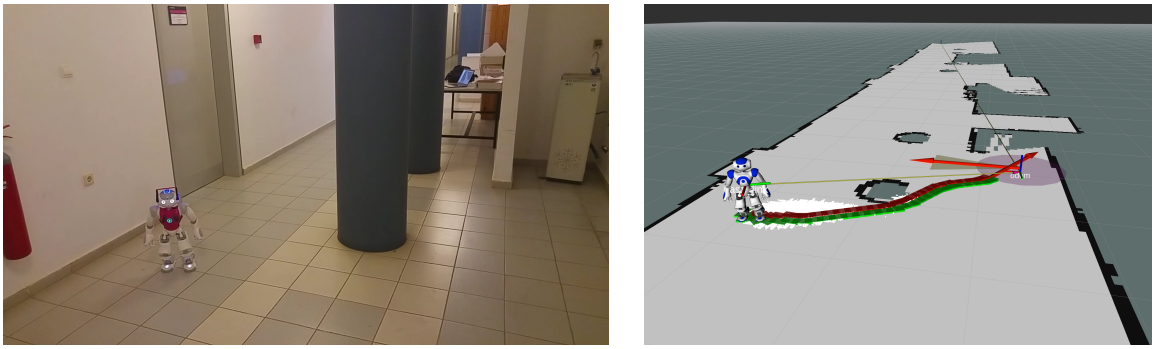


Figure 4.46: Nao navigation in known environment, using the footstep planner ROS node

4.10 YOLO Object Detector and Human Detection

Nowadays, new neural networks techniques and forms are emerging, due to the rapid growing of the machine learning research field, and they are applied in every day applications, thanks to their wide learning adaptability, rapidness and efficiency. Specifically, in robotic applications there are plenty examples and implementations been made that use neural networks, like in robot motion, task coordination, robot vision and more.

Object detection and recognition algorithms deal with the detection and the localization of an appeared object in the camera frame and its classification in one of the known classes-labels. Current detection systems generally start by extracting a set o features from the input images and then use classifiers or localizers to identify the objects in the feature space. These systems are detecting either in sliding window fashion all over the image or in specific selected regions. To deal with the human detection problem, we are using a unified object detector system, the YOLO [38], which outperforms most of the aforementioned algorithms, as it has extensively applied in real-time robotic vision applications since 2015 [39].

The YOLO system is based on a single convolutional neural network, developed with the Darknet open-source framework [40], that predicts the bounding boxes of detected objects in the image frame and the class probabilities for those boxes, simultaneously. Unlike to most object detection systems that use classifiers for the detection, the YOLO frames the detection problem through a regression approach. By design, this system uses the entire image information while training and testing, so it learns a variety of object representations in observed camera scene, as each image have different contextual information.

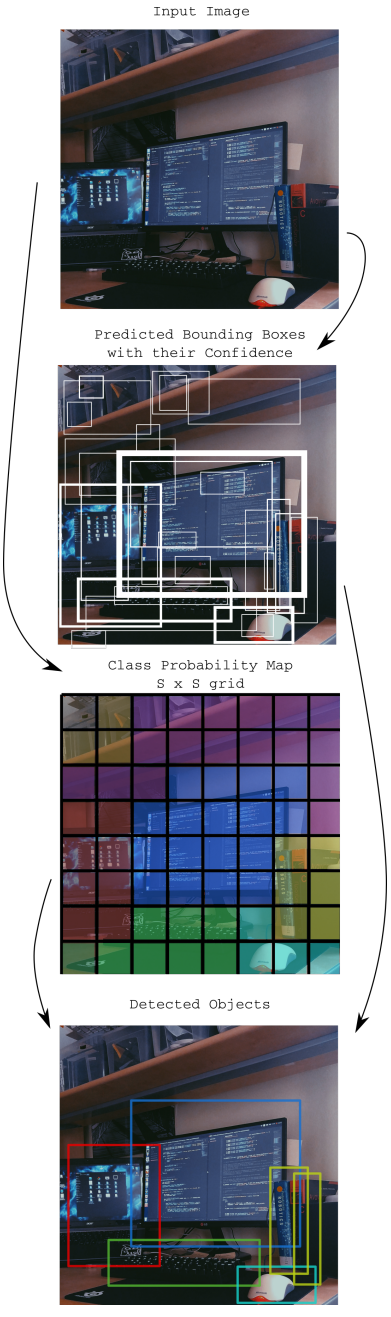


Figure 4.47: YOLO model

This network predicts the bounding boxes across all classes in the image, by using features from the entire image.

To be more technically specific, the YOLO detector divides the image in $S \times S$ grid and for each cell predicts the B bounding boxes with their confidence and the C conditional class probability. As the system was firstly evaluated on the PASCAL VOC detection dataset [41], it uses a 7×7 grid image segmentation and 2 bounding boxes prediction for each grid cell. The Figure 4.47 shows the network structure as it was implemented for the previously mentioned contest, by having a prediction tensor of $7 \times 7 \times 30 = \Big(S \times S \times (B*5+C)\Big)$. The initial convolutional layers are responsible for the feature extraction and the two fully-connected in the end are for the output probabilities and coordinates prediction. Also, the output is optimized for sum-squared error, in order to weight equally the localization and classification errors.

A humanoid_nav_msgs /StepTarget ROS Message

uint8 *right* = 0
uint8 *left* = 1

geometry_msgs/Pose2D *pose*
  uint32 *x*
  time *y*
  string *theta*
uint8 *leg*

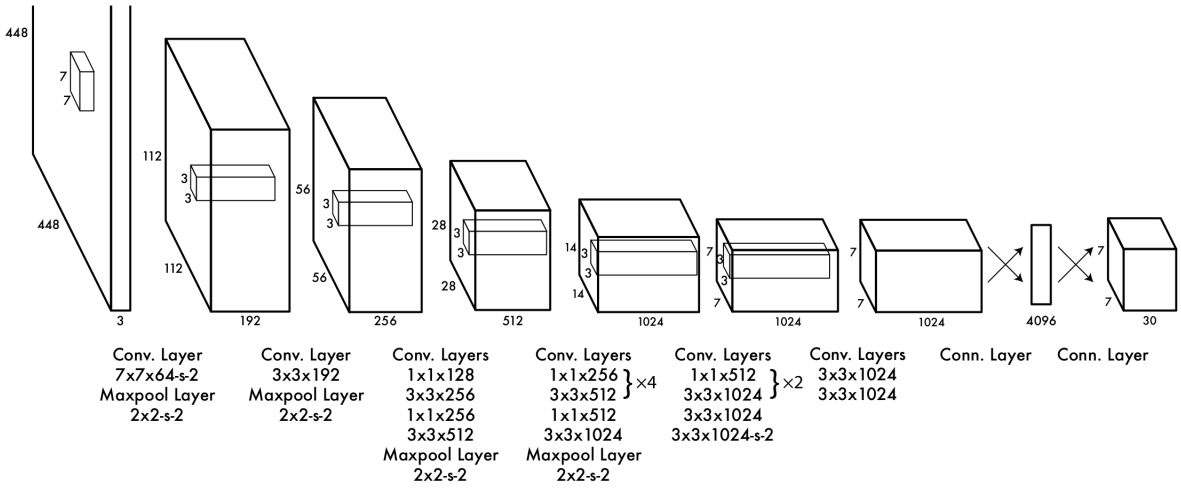


Figure 4.48: YOLO network structure

Hence, we used the Darknet open source framework to train a new YOLO neural network, based on the given human imagery dataset. Since, YOLO was initially created on PASCAL VOC images, we downloaded VOC2012 dataset and we extracted all

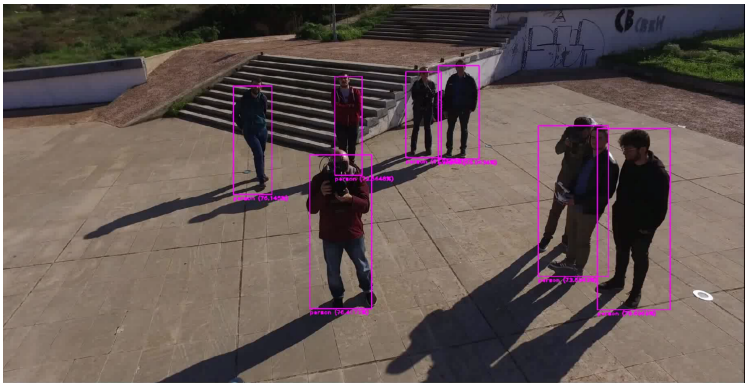


Figure 4.49: Detecting humans as appeared in groups.

photos showing humans in the frame. At the same time, we created a .text file containing the region of interest rectangle of the appeared humans, in Darknet required form $(x0\_box\_ratio, y0\_box\_ratio, height\_ratio, width\_ratio)$ for the training. Also, pre-trained convolutional weights were used[1], that were trained on ImageNet[2].

The resulted weights after the training was tested on two different network structures, the one as the default appeared in Figure 4.47 and the other as a "tiny" version based on Darknet's reference model[3], which consisted on 6 convolutional layers without any the large fully connected layers in the end. The neural network have been tested on Nao's camera footage (and on typical mobile camera footage) and sufficiently detected individuals, despite if they appeared in groups (see Figure 4.48), and worked pretty well under low-light conditions (see Figure 4.49).

[1] https://pjreddie.com/darknet/imagenet/#darknet53
[2] http://image-net.org/
[3] Config file at https://github.com/pjreddie/darknet/blob/master/cfg/darknet.cfg.

Figure 4.50: Darknet neural network use under low-light environment conditions.

Chapter 5

Conclusion

5.1 Conclusion

This thesis describes a ground-aerial robot collaboration approach, in which a quadcopter and a humanoid robot cooperate to solve a real-world Search and Rescue scenario. Assuming that the humanoid robot can't take any range measurements to localize itself in the world, we use the quadcopter to perform both the environment mapping and the humanoid robot localization. The quadcopter uses a combination of SLAM and OctoMapping approaches to extract a 2.5D occupancy grid map, specifically made for the humanoid robot navigation. As the humanoid robot is initially localized in the map frame, it is assigned with a goal position to reach and a parameterized footstep planner performs the path planning. An adaptive MCL approach is used for the humanoid robot localization, given its odometry data and the particularly spatial transformed quadcopter's LiDAR range data. Finally, a pre-trained convolutional Darknet neural network is used to perform human detection, on the way to its target position. Each system part is presented in detail and the entire system is validated with real-world experiments. The entire project has been implemented within the Robot Operating System (ROS) and is available as an open source package[1].

[1] https://github.com/jimcha21

5.2 Future Work

5.2.1 Advanced 3D Perception Approach

The use of a stereo camera compared with a 3D LiDAR system, in 3D world perception, could be a cheaper solution but requires fine tunning for satisfactory results. The proposed stereo matching to 3D map reconstruction pipeline, was specifically selected and adjusted to work with the given *Guidance* system and to able to perform onboard in real-time situations. Although, in our approach all processes are executed on the $Manifold$'s quad-core ARM processor and there is no use of the embedded NVIDIA graphics processor, which supports the CUDA parallel computing platform. Nowadays, especially in machine vision applications, algorithms are implemented in respect to parallelism to be able to be executed on graphic processor units and take advantage of their multi-core specifications Thus, importing and adjusting the used stereo matching and post-filter algorithms to work with $Manifold$'s GPU will yield better onboard runtimes through lesser memory consumption.

Also, it's noteworthy that an upgraded *Guidance* system could also aid to obtain better 3D world reconstructions, as the stereo matching algorithms tend to perform better on higher image resolutions.

5.2.2 Fully Autonomous System

In our approach, despite the semi-autonomous humanoid robot behavior (awaits for our target position to proceed with the exploration) the quadcopter is guided from us, as we monitor the current map status along with the sensor measurements to ensure sufficient area coverage through a safe flight. So, an ideal scenario would implement an autonomous quadcopter that could decide to follow specific paths in the unknown environment to maximize its knowledge about the external world, and then inform the ground robot accordingly.

Therefore, given the implemented SLAM approach, a fastSLAM exploration method could be applied, which computes future estimates of estimated control sequences and trades off path uncertainty versus map uncertainty when evaluating such control sequences. FastSLAM exploration techniques are tested in wide-area robotic exploration scenarios, by which the robots tend to visit unknown terrain to enrich the map state, and other times lead back to previously mapped areas to improve their pose estimates. By

combining also the located position of the humanoid robot in the map frame, a diverse of control policies and strategies could be designed and implemented.

References

[1] Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA Workshop on Open Source Software. (2009) 10, 44

[2] Kam, H., Lee, S.H., Park, T., Kim, C.H.: Rviz: a toolkit for real domain data visualization. **60** (10 2015) 1–9 12

[3] Foote, T.: tf: The transform library. In: Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on. Open-Source Software workshop (April 2013) 1–6 12, 49

[4] Rusu, R.B., Cousins, S.: 3D is here: Point Cloud Library (PCL). In: IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China (May 9-13 2011) 20, 62

[5] Heikkila, J., Silven, O.: A four-step camera calibration procedure with implicit image correction. In: Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (June 1997) 1106–1112 25

[6] Zhang, Z.: A flexible new technique for camera calibration. IEEE Transactions on Pattern Analysis and Machine Intelligence **22**(11) (Nov 2000) 1330–1334 29, 31

[7] Bouguet, J.Y.: The calibration toolbox for matlab, example 5: Stereo rectification algorithm 35, 52

[8] Konolige, K.: Small vision systems: Hardware and implementation. In Shirai, Y., Hirose, S., eds.: Robotics Research, London, Springer London (1998) 203–212 39, 59

[9] Loose, H., Zasepa, A.M., Pierzchala, P., Ritter, R.: Gps based navigation of an autonomous mobile vehicle 42

[10] Hamid, M.H.A., Adom, A.H., Rahim, N.A., Rahiman, M.H.F.: Navigation of mobile robot using global positioning system (gps) and obstacle avoidance system with commanded loop daisy chaining application method. In: 2009 5th International Colloquium on Signal Processing Its Applications. (March 2009) 176–181 42

[11] Loose, H.: Autonomous mobile robots for outdoor tasks 42

[12] Malyavej, V., Kumkeaw, W., Aorpimai, M.: Indoor robot localization by rssi/imu sensor fusion. In: 2013 10th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology. (May 2013) 1–6 42, 43

[13] Biswas, J., Veloso, M.: Wifi localization and navigation for autonomous indoor mobile robots. In: 2010 IEEE International Conference on Robotics and Automation. (May 2010) 4379–4384 42

[14] Hou, X., Arslan, T.: Monte carlo localization algorithm for indoor positioning using bluetooth low energy devices. In: 2017 International Conference on Localization and GNSS (ICL-GNSS). (June 2017) 1–6 42

[15] Liu, C., Cheng, Z., Zhang, Y., Wang, G.: An indoor positioning system based on rfid with rotating antenna and passive tags. In: 2017 2nd International Conference on Robotics and Automation Engineering (ICRAE). (Dec 2017) 455–459 42

[16] Xing, B., Zhu, Q., Pan, F., Feng, X.: Marker-based multi-sensor fusion indoor localization system for micro air vehicles. In: Sensors. (2018) 43

[17] Chen, W., Zhang, T.: An indoor mobile robot navigation technique using odometry and electronic compass. International Journal of Advanced Robotic Systems **14**(3) (2017) 1729881417711643 43

[18] Chen, C., Chai, W., Nasir, A.K., Roth, H.: Low cost imu based indoor mobile robot navigation with the assist of odometry and wi-fi using dynamic constraints. In: Proceedings of the 2012 IEEE/ION Position, Location and Navigation Symposium. (April 2012) 1274–1279 43

[19] Huang, A.S., Bachrach, A., Henry, P., Krainin, M., Maturana, D., Fox, D., Roy, N.: Visual odometry and mapping for autonomous flight using an rgb-d camera. In: ISRR. (2011) 43

[20] Grzonka, S., Plagemann, C., Grisetti, G., Burgard, W.: Look-ahead proposals for robust grid-based slam with rao-blackwellized particle filters. In: International Journal of Robotics Research (IJRR). (February 2009) 191–200 43

[21] Grzonka, S., Grisetti, G., Burgard, W.: Towards a navigation system for autonomous indoor flying. In: 2009 IEEE International Conference on Robotics and Automation. (May 2009) 2878–2883 44

[22] Kohlbrecher, S., von Stryk, O., Meyer, J., Klingauf, U.: A flexible and scalable slam system with full 3d motion estimation. In: 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics. (Nov 2011) 155–160 44, 70

[23] Ruangpayoongsak, N., Roth, H., Chudoba, J.: Mobile robots for search and rescue. In: IEEE International Safety, Security and Rescue Rototics, Workshop, 2005. (June 2005) 212–217 44

[24] Guarnieri, M., Kurazume, R., Masuda, H., Inoh, T., Takita, K., Debenest, P., Hodoshima, R., Fukushima, E., Hirose, S.: Helios system: A team of tracked robots for special urban search and rescue operations. In: 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems. (Oct 2009) 2795–2800 44

[25] Reid, R., Cann, A., Meiklejohn, C., Poli, L., Boeing, A., Braunl, T.: Cooperative multi-robot navigation, exploration, mapping and object detection with ros. In: 2013 IEEE Intelligent Vehicles Symposium (IV). (June 2013) 1083–1088 44

[26] Luo, C., Espinosa, A.P., Pranantha, D., Gloria, A.D.: Multi-robot search and rescue team. In: 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics. (Nov 2011) 296–301 44

[27] Lee, C.E., Im, H.J., Lim, J.M., Cho, Y.J., Sung, T.K.: Seamless routing and cooperative localization of multiple mobile robots for search and rescue application. ETRI Journal **37**(2) 262–272 44

[28] Hirschmuller, H.: Stereo processing by semiglobal matching and mutual information. IEEE Transactions on Pattern Analysis and Machine Intelligence **30**(2) (Feb 2008) 328–341 56, 57

[29] Viola, P., Wells, W.M.: Alignment by maximization of mutual information. In: Proceedings of IEEE International Conference on Computer Vision. (June 1995) 16–23 56

[30] Birchfield, S., Tomasi, C.: A pixel dissimilarity measure that is insensitive to image sampling. IEEE Transactions on Pattern Analysis and Machine Intelligence **20**(4) (April 1998) 401–406 57

[31] Farbman, Z., Fattal, R., Lischinski, D., Szeliski, R.: Edge-preserving decompositions for multi-scale tone and detail manipulation. In: ACM SIGGRAPH 2008 Papers. SIGGRAPH '08, New York, NY, USA, ACM (2008) 67:1–67:10 59

[32] Min, D., Choi, S., Lu, J., Ham, B., Sohn, K., Do, M.: Fast global image smoothing based on weighted least squares. IEEE Transactions on Image Processing **23**(12) (12 2014) 5638–5653 60

[33] Hornung, A., Wurm, K., Bennewitz, M., Stachniss, C., Burgard, W.: Octomap: An efficient probabilistic 3d mapping framework based on octrees. **34** (04 2013) 64

[34] Malbezin, P., Piekarski, W., Thomas, B.H.: Measuring artootkit accuracy in long distance tracking experiments. In: The First IEEE International Workshop Agumented Reality Toolkit,. (Sept 2002) 2 pp.– 76

[35] Fox, D.: Kld-sampling: Adaptive particle filters and mobile robot localization. (10 2001) 80

[36] Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics (Intelligent Robotics and Autonomous Agents). The MIT Press (2005) 81

[37] Hornung, A., Dornbush, A., Likhachev, M., Bennewitz, M.: Anytime search-based footstep planning with suboptimality bounds. In: 2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012). (Nov 2012) 674–679 82

[38] Redmon, J., Farhadi, A.: Yolov3: An incremental improvement. arXiv (2018) 87

[39] Joseph Redmon, Santosh Divvala, R.G.A.F.: You only look once: Unified, real-time object detection. arXiv:1506.02640 (2015) 87

[40] Redmon, J.: Darknet: Open source neural networks in c. http://pjreddie.com/darknet/ (2013–2016) 87

[41] Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html 88