

**École polytechnique de Louvain**

# **LiDAR - Camera fusion for SLAM of autonomous robots or drones in indoor environments**

Authors : **Bruno NICOLETTA, Matteo FERRETTI DI CASTELFERRETTO**  
Supervisors : **Julien HENDRICKX, François WIELANT**  
Readers : **François HEREMANS, Renaud DETRY**  
Academic year 2020–2021  
Master [120] in Electro-mechanical Engineering





# **LiDAR - Camera fusion for SLAM of autonomous robots or drones in indoor environments**

Bruno Nicoletta and Matteo Ferretti di Castelferretto

August 2021

Promoter and advisor : Julien Hendrickx  
Co-promoter and advisor : François Wielant

École polytechnique de Louvain  
UCLouvain, EPL



# Acknowledgements

We would like to thank our promoter, professor Julien Hendrickx, for giving us the opportunity to work on this subject, supported us during this academic year and providing us with the adequate resources.

We would also like to express our special thanks to Mr. François Wielant who has been present throughout our work. By his indispensable presence and his wise advice, Mr. Wielant will have been a decisive asset in the success of this thesis.

Finally, we would like to thank the Polytechnic School of Louvain for providing essential equipment and know-how, without which this work would not have been possible.



# Abstract

This master's thesis is part of a long-term academic project, launched by the Polytechnic School of Louvain, with the aim to develop technologies that would enable low-cost UAVs to navigate autonomously in indoor environments.

In the previous years, several master theses tried to bring a robust solution for localisation and mapping of autonomous drones based on the AR.Drone 2.0 from Parrot, equipped with a set of sensors including a monocular camera, an inertial measurement unit and various localised range finders. Our master's thesis differs from previous these and work on homemade modular and portable platform to develop a sensor system for the SLAM without the constraints from the equipment of the (outdated) AR.Drone 2.0. In this master's thesis, we will the study the potential of a system combining a monocular camera with a LiDAR sensor. The main goal is to map out the overall possibilities and potential implementation with the addition of a LiDAR to the sensor system.

After a review of the State-of-the-art regarding SLAM and relevant sensors, we propose a first implementation of a sensor system based on monocular camera with a 2D scanning LiDAR sensor. This system combines the depth information from the LiDAR and the visual information from the camera to create 3D point clouds and depth images which can be fed to state-of-the-art visual SLAM algorithms.

The sensor system shows interesting results and is first step toward a LiDAR based SLAM method. The solution works but not perfectly. One of the main issue is the computational time to establish the depth images. Therefore, we will also address the shortcomings and discuss interesting directions going forward.



# Contents

|                                                |           |
|------------------------------------------------|-----------|
| <b>Acknowledgements</b>                        | <b>i</b>  |
| <b>Abstract</b>                                | <b>ii</b> |
| <b>1 Introduction</b>                          | <b>1</b>  |
| 1.1 Context                                    | 1         |
| 1.2 SLAM and autonomous drones                 | 2         |
| 1.3 Previous Workd                             | 2         |
| 1.4 Solution                                   | 3         |
| 1.4.1 Goals of the project                     | 3         |
| 1.5 Outlines                                   | 3         |
| <b>2 State-of-the-Art</b>                      | <b>5</b>  |
| 2.1 Drones                                     | 5         |
| 2.1.1 Drones geometry                          | 5         |
| 2.2 Sensors                                    | 7         |
| 2.2.1 RGB Camera                               | 7         |
| 2.2.2 LiDAR                                    | 8         |
| 2.2.3 Ultrasonic Sensors                       | 10        |
| 2.2.4 RGB-D Camera                             | 10        |
| 2.2.5 Inertial Measurement Unit                | 11        |
| 2.3 Sensor fusion                              | 12        |
| 2.3.1 LiDAR - RGB Camera Fusion                | 12        |
| 2.4 Simultaneous Localisation and Mapping      | 13        |
| 2.4.1 LiDAR SLAM                               | 13        |
| 2.4.2 Visual SLAM                              | 15        |
| 2.4.3 Sensor Fusion SLAM                       | 18        |
| <b>3 LiDAR Sensors</b>                         | <b>20</b> |
| 3.1 Evolution of LiDAR sensors                 | 20        |
| 3.2 Working principles                         | 21        |
| 3.2.1 Time of flight LiDAR                     | 22        |
| 3.2.2 Phase ranging LiDAR                      | 22        |
| 3.2.3 Triangular ranging LiDAR                 | 23        |
| 3.3 Selection of our LiDAR sensor : RPLIDAR A3 | 24        |
| 3.3.1 Our selection                            | 24        |
| 3.3.2 Specifications                           | 28        |
| 3.3.3 Communication interface                  | 30        |

|          |                                                                       |           |
|----------|-----------------------------------------------------------------------|-----------|
| 3.3.4    | Sensor output . . . . .                                               | 30        |
| 3.4      | Limitations . . . . .                                                 | 31        |
| 3.4.1    | Interface for visualisation . . . . .                                 | 31        |
| 3.4.2    | Inherent problems . . . . .                                           | 32        |
| 3.4.3    | Limit Testing . . . . .                                               | 33        |
| <b>4</b> | <b>Summary of the first part</b>                                      | <b>38</b> |
| <b>5</b> | <b>Setup description</b>                                              | <b>40</b> |
| 5.1      | Physical setup . . . . .                                              | 40        |
| 5.1.1    | Hardware . . . . .                                                    | 40        |
| 5.1.2    | Mechanical aspect . . . . .                                           | 41        |
| 5.2      | Software . . . . .                                                    | 41        |
| 5.2.1    | Framework Decision . . . . .                                          | 41        |
| 5.2.2    | ROS, working principle and test bench . . . . .                       | 43        |
| <b>6</b> | <b>Fusion of LiDAR and camera data</b>                                | <b>47</b> |
| 6.1      | Calibration of the camera . . . . .                                   | 47        |
| 6.2      | Transposing Lidar information to the camera reference frame . . . . . | 50        |
| 6.2.1    | Theoretical approach . . . . .                                        | 50        |
| 6.2.2    | Implementation and results . . . . .                                  | 52        |
| 6.3      | Upadte on the progression . . . . .                                   | 53        |
| <b>7</b> | <b>Estimated depth from Convolutional Neural Networks</b>             | <b>56</b> |
| 7.1      | DenseDepth, Encoder-decoder architecture . . . . .                    | 56        |
| 7.1.1    | Working principle . . . . .                                           | 56        |
| 7.1.2    | Implementation . . . . .                                              | 57        |
| 7.2      | Adabin, depth estimation using adaptive bins . . . . .                | 57        |
| 7.2.1    | Working principle . . . . .                                           | 57        |
| 7.2.2    | Implementation and results . . . . .                                  | 58        |
| 7.3      | Comparison and best candidate . . . . .                               | 59        |
| 7.3.1    | Performance as a ROS node . . . . .                                   | 61        |
| <b>8</b> | <b>Depth Image Correction</b>                                         | <b>62</b> |
| 8.1      | Correction process . . . . .                                          | 63        |
| 8.2      | Results . . . . .                                                     | 66        |
| 8.2.1    | Scenario 1 . . . . .                                                  | 67        |
| 8.2.2    | Scenario 2 . . . . .                                                  | 70        |
| 8.2.3    | Scenario 3 . . . . .                                                  | 73        |
| 8.2.4    | 3D visualisation . . . . .                                            | 76        |
| 8.3      | Other algorithms . . . . .                                            | 77        |
| <b>9</b> | <b>LiDAR, additional gains regarding drone technology</b>             | <b>80</b> |
| 9.1      | Obstacle avoidance . . . . .                                          | 80        |



|                                                                          |            |
|--------------------------------------------------------------------------|------------|
| 9.2 Odometry . . . . .                                                   | 80         |
| 9.2.1 ICP odometry . . . . .                                             | 80         |
| 9.2.2 Gauss-Newton Optimization odometry . . . . .                       | 81         |
| <b>10 LiDAR-Camera fusion applied in SLAM</b>                            | <b>83</b>  |
| 10.1 ROS environment . . . . .                                           | 83         |
| 10.2 RTAB-Map . . . . .                                                  | 83         |
| 10.2.1 Working principle and implementation . . . . .                    | 83         |
| 10.2.2 Results and discussion . . . . .                                  | 85         |
| 10.3 ORB-SLAM2 . . . . .                                                 | 88         |
| 10.3.1 What is ORB, Oriented FAST and Rotated BRIEF ? . . . . .          | 88         |
| 10.3.2 Implementation . . . . .                                          | 90         |
| 10.3.3 Results and discussion . . . . .                                  | 91         |
| <b>11 Going forward : Lines of investigation and future developments</b> | <b>99</b>  |
| 11.1 Implementation based on 2D scanning LiDAR technology . . . . .      | 99         |
| 11.1.1 Mechanical/hardware improvements . . . . .                        | 99         |
| 11.1.2 Software based improvements . . . . .                             | 100        |
| 11.2 Improving the system with other sensors . . . . .                   | 102        |
| 11.2.1 3D LiDAR . . . . .                                                | 102        |
| 11.2.2 RGB-D camera . . . . .                                            | 102        |
| 11.2.3 Flash solid-state LiDAR . . . . .                                 | 102        |
| 11.3 Comments and conclusion . . . . .                                   | 103        |
| <b>12 Conclusion</b>                                                     | <b>105</b> |



# List of Figures

|      |                                                                        |    |
|------|------------------------------------------------------------------------|----|
| 2.1  | Two main drone geometry                                                | 5  |
| 2.2  | Under actuated propeller configuration                                 | 6  |
| 2.3  | Summary of typical drone geometry                                      | 7  |
| 2.4  | Example of non-coplanar geometry                                       | 8  |
| 2.5  | Scan of a single plane 2D LiDAR                                        | 9  |
| 2.6  | Optical Phased Array. Image courtesy of Analog Photonics               | 10 |
| 2.7  | Different LiDAR types                                                  | 10 |
| 2.8  | Different IR light pattern                                             | 11 |
| 2.9  | Stereo IR camera D435 from Intel                                       | 11 |
| 2.10 | 3D scene reconstruction form [54]                                      | 12 |
| 2.11 | Different SLAM generated maps form [32]                                | 14 |
| 2.12 | Loop closure problem explained in [72]                                 | 14 |
| 2.13 | Scale space representation [93]                                        | 16 |
| 2.14 | result of the computation of $D(x, y, \sigma)$ [93]                    | 16 |
| 2.15 | SURF descriptor representation, credits to: [41]                       | 17 |
| 2.16 | Example of photometric consistency                                     | 18 |
| 3.1  | <i>Hughes Aircraft</i>                                                 | 20 |
| 3.2  | Point Cloud created by Velodyne LiDAR's <i>Alpha Prime sensor</i> [55] | 21 |
| 3.3  | Time-of-flight                                                         | 22 |
| 3.4  | Phase shift                                                            | 23 |
| 3.5  | Phase shift                                                            | 23 |
| 3.6  | Triangulation                                                          | 24 |
| 3.7  | Triangulation                                                          | 24 |
| 3.8  |                                                                        | 24 |
| 3.9  | List and comparison of the <i>Slamtec</i> LiDAR sensors [97]           | 26 |
| 3.10 | RPLIDAR A3                                                             | 28 |
| 3.11 | Schematic of the RPLIDAR A3                                            | 28 |
| 3.12 | RPLIDAR A3M                                                            | 30 |
| 3.13 | Coordinate system                                                      | 30 |
| 3.14 | RoboStudio                                                             | 31 |
| 3.15 | Rviz interface and corresponding real world setup                      | 32 |
| 3.16 | RPLiDAR A3 scanned data for the situation pictured in 3.15b            | 32 |
| 3.17 | Classic case                                                           | 33 |
| 3.18 | Case with erroneous data                                               | 33 |
| 3.19 | <i>Fine wire mesh example</i>                                          | 34 |
| 3.20 | Testing : Tupperware                                                   | 35 |

## List of Figures

|                                                                                                                                                                               |    |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.21 Testing : Plastic foil                                                                                                                                                   | 35 |
| 3.22 Testing : Glass cup                                                                                                                                                      | 35 |
| 3.23 Testing : Black PLA                                                                                                                                                      | 35 |
| 3.24 Testing : Black styrofoam                                                                                                                                                | 35 |
| 3.25 Testing : Black styrofoam                                                                                                                                                | 35 |
| 3.26 Testing : Back of the screen                                                                                                                                             | 36 |
| 3.27 Testing : Back of the screen                                                                                                                                             | 36 |
| 3.28 Testing : Bright display                                                                                                                                                 | 36 |
| 3.29 Testing : Bright display                                                                                                                                                 | 36 |
| 3.30 Back of the blackboard                                                                                                                                                   | 37 |
| 3.31 Back of the blackboard                                                                                                                                                   | 37 |
| 3.32 Testing :PlexiGlass                                                                                                                                                      | 37 |
| 3.33 Testing : Cooking pot                                                                                                                                                    | 37 |
| 5.1 Semantic schematic of the mobile platform                                                                                                                                 | 41 |
| 5.2 Setup                                                                                                                                                                     | 42 |
| 5.3 ROS working principle                                                                                                                                                     | 43 |
| 5.4 Transform tree of the system                                                                                                                                              | 45 |
| 5.5 ROS test bench                                                                                                                                                            | 46 |
| 6.1 Remapping from the 3D points in the real world to the 2D image plan of the camera [63]                                                                                    | 47 |
| 6.2 Schematic of the pinhole camera model [1]                                                                                                                                 | 48 |
| 6.3 LiDAR reference frame                                                                                                                                                     | 50 |
| 6.4 Demonstration setup                                                                                                                                                       | 53 |
| 6.5 General setup: Projections of Lidar data into camera image                                                                                                                | 54 |
| 6.6 Stools setup: Projections of Lidar data into camera image                                                                                                                 | 55 |
| 7.1 General schematic of the Network [5]                                                                                                                                      | 57 |
| 7.2 AdaBind block diagram                                                                                                                                                     | 58 |
| 7.3 FFlow chart of the ROS environment with Monodepth and Monodepth <sub>adabinnodesimplemented</sub>                                                                         | 59 |
| 7.4 Example scene where AdaBins is superior. The four corners of the box are originally at 1.26m from the camera and the four corners of the wall are measured to be at 1.99m | 60 |
| 7.5 Example scene where AdaBins and DenseDepth are comparable. Each box sits at 1.27m from the camera while the wall's corners are measured at 1.99m                          | 60 |
| 8.1 Schematic of the LiDAR and camera information localisation                                                                                                                | 62 |
| 8.2 LiDAR reference frame                                                                                                                                                     | 63 |
| 8.3 Pictures of the demonstration setup in Scenario 1                                                                                                                         | 67 |
| 8.4 View captured by the monocular camera in Scenario 1                                                                                                                       | 67 |
| 8.5 Comparison between the estimated depth by AdaBins and Method 1                                                                                                            | 68 |
| 8.6 Comparison between the estimated depth by AdaBins and Method 1                                                                                                            | 68 |
| 8.7 Comparison between the estimated depth by AdaBins and Method 1                                                                                                            | 68 |

## List of Figures

|       |                                                                                              |    |
|-------|----------------------------------------------------------------------------------------------|----|
| 8.8   | Comparison between the estimated depth by AdaBins and Method 2 . . .                         | 69 |
| 8.9   | Comparison between the estimated depth by AdaBins and Method 2 . . .                         | 69 |
| 8.10  | Comparison between the estimated depth by AdaBins and Method 2 . . .                         | 69 |
| 8.11  | Pictures of the demonstration setup in Scenario 2 . . . . .                                  | 70 |
| 8.12  | View captured by the monocular camera in Scenario 2 . . . . .                                | 70 |
| 8.13  | Comparison between the estimated depth by AdaBins and Method 1 . . .                         | 71 |
| 8.14  | Comparison between the estimated depth by AdaBins and Method 1 . . .                         | 71 |
| 8.15  | Comparison between the estimated depth by AdaBins and Method 1 . . .                         | 71 |
| 8.16  | Comparison between the estimated depth by AdaBins and Method 2 . . .                         | 72 |
| 8.17  | Comparison between the estimated depth by AdaBins and Method 2 . . .                         | 72 |
| 8.18  | Comparison between the estimated depth by AdaBins and Method 2 . . .                         | 72 |
| 8.19  | Pictures of the demonstration setup in Scenario 3 . . . . .                                  | 73 |
| 8.20  | View captured by the monocular camera in Scenario 3 . . . . .                                | 73 |
| 8.21  | Comparison between the estimated depth by AdaBins and Method 1 . . .                         | 74 |
| 8.22  | Comparison between the estimated depth by AdaBins and Method 1 . . .                         | 74 |
| 8.23  | Comparison between the estimated depth by AdaBins and Method 1 . . .                         | 74 |
| 8.24  | Comparison between the estimated depth by AdaBins and Method 2 . . .                         | 75 |
| 8.25  | Comparison between the estimated depth by AdaBins and Method 2 . . .                         | 75 |
| 8.26  | Comparison between the estimated depth by AdaBins and Method 2 . . .                         | 75 |
| 8.27  | Comparison between the estimated depth by AdaBins and Method 2 . . .                         | 76 |
| 8.28  | Comparison between the estimated depth by AdaBins and Method 2 . . .                         | 76 |
| 8.29  | Comparison between the estimated depth by AdaBins and Method 2 . . .                         | 76 |
| 8.30  | YOLO output examples . . . . .                                                               | 77 |
| 8.31  | Yolo results with the setup of scenario 3 . . . . .                                          | 78 |
| 8.32  | YOLO output examples . . . . .                                                               | 79 |
|       |                                                                                              |    |
| 9.1   | ICP after five iterations . . . . .                                                          | 81 |
|       |                                                                                              |    |
| 10.1  | Available messages for SLAM implementation . . . . .                                         | 84 |
| 10.2  | Bag of words concept . . . . .                                                               | 84 |
| 10.3  | Global view of the ROS implementation of RTAB-Map [51] . . . . .                             | 85 |
| 10.4  | Global view of the ROS implementation with our topics connected . . . .                      | 85 |
| 10.5  | Complete transform tree generated by RTABMap . . . . .                                       | 86 |
| 10.6  | Map generated by RTABMap and the resulting passes with the original<br>environment . . . . . | 87 |
| 10.7  | FAST comparison representation. Credits to [92] . . . . .                                    | 88 |
| 10.8  | Multi scale representation. Credits to [92] . . . . .                                        | 89 |
| 10.9  | Flow chart of ORB-SALM2 [69] . . . . .                                                       | 90 |
| 10.10 | ROS flow chart of ORB-SLAM2 monocular . . . . .                                              | 91 |
| 10.11 | ROS flow chart of ORB-SLAM2 stereo . . . . .                                                 | 92 |
| 10.12 | Generated map by ORB-SLAM2 after 3 passes with fixed camera angle . .                        | 93 |
| 10.13 | Generated map by ORB-SLAM2 after 3 passes with changing camera angle                         | 94 |
| 10.14 | Detail highlight of the map from figure 10.13 . . . . .                                      | 95 |

*List of Figures*

|       |                                                                     |     |
|-------|---------------------------------------------------------------------|-----|
| 10.15 | Generated map by ORB-SLAM2 after 3 passes with the depth correction |     |
|       | node . . . . .                                                      | 96  |
| 10.16 | Locations in generated map with stereo ORB-SLAM2                    | 97  |
| 10.17 | comparison of the two maps, ORB-SLAM mono left, ORB-SLAM stereo     |     |
|       | right. . . . .                                                      | 97  |
| 10.18 | Highlight of lost information between mono and stereo ORB-SLAM2.    | 98  |
| 11.1  | Examples of                                                         | 100 |
| 11.2  | Overview of the principle used by Y. Lu et al. in [2]               | 101 |

# List of Tables

|      |                                                               |    |
|------|---------------------------------------------------------------|----|
| 3.1  | Size and weight [16] and [15]                                 | 29 |
| 3.2  | Battery/electricity requirement [16] and [15]                 | 29 |
| 3.3  | Key performance information [16] and [15]                     | 29 |
| 3.4  | Key communication information [15]                            | 30 |
| 3.5  | Output information [15]                                       | 31 |
| 7.1  | Upsampling block                                              | 57 |
| 7.2  | Summary of the timings, averaged on 20 frames                 | 61 |
| 10.1 | Results of RTABMap                                            | 87 |
| 10.2 | Results of ORB-SLAM2 monocular with fixed camera direction    | 92 |
| 10.3 | Results of ORB-SLAM2 monocular with oriented camera direction | 93 |
| 10.4 | Results of ORB-SLAM2 monocular with oriented camera direction | 94 |





# 1 Introduction

## 1.1 Context

To meet the needs of our constantly expanding society, man found himself obliged to replace more and more tedious tasks by robotic applications. The first intensive uses of robotics were mainly in production lines where the work is heavy and repetitive. But with the changing needs of industries and consumers, robotics can no longer be satisfied with being repetitive, it must now and for the years to come, become autonomous. This is a paradigm shift that we have already observed for a few years with the arrival, already dating, of long-range military drones, mowers and intelligent vacuum cleaners and in the near future, the arrival of driverless cars on our roads.

The field where this technology is approaching maturity is that of mobile and semi-autonomous robots. These are robots that already know their environment or have a global tracking system. Conversely, where maturity is lacking, is in the field of completely autonomous robots with the only point of reference, the environment. This paradigm imposes new constraints and gave birth to a concept known as SLAM (Simultaneous Location and Mapping). In other words, the art of creating a map of the environment while using it for pose estimation.

In parallel, unmanned aerial vehicles (UAV), also commonly termed drones, play a significant role in various sectors and their demand is in constant increase. According to a report published by *MarketsandMarkets* in June 2021, the UAV market is estimated to be USD 27.4 billion in 2021 and is projected to reach USD 58.4 billion by 2026 [62]. This growth can be mainly attributed by their expanding use in the military sector but also in various commercial application such as monitoring, surveying, mapping, precision agriculture, aerial remote sensing, product delivery etc.

Autonomous UAV is an active field of research. Indeed, integration of new technologies such as artificial intelligence, sense and avoid systems, cloud computing, and the possibility to make autonomous drones to carry specific tasks could have great scientific and industrial/commercial benefits. Of particular interest is the implementation of SLAM algorithms for drones to make them capable of moving in unknown environments. It is in this line of thought that our master's thesis is embedded.

## 1.2 SLAM and autonomous drones

SLAM is divided into two categories, indoor and outdoor. The indoor SLAM is particularly interesting because it could, once applied on a suitable mobile robot, facilitate the exploration of industrial sites, map tunnels and other confined environments or even explore damaged buildings in search of survivors and all, this in a completely autonomous way. Multi-rotor drones as mobile robots (quadcopters for example) shine particularly for those tasks because of the following reasons:

- Not limited by the ground geometry as this is a flying machine.
- The multi-rotor geometry gives those devices the ability to fly in any directions. Hence an outstanding manoeuvrability.
- Multiple motor configurations give the possibility for those machines to carry heavy payloads.
- The reduced size of their body makes them well suited for small hard to reach space.

Because of the mentioned advantages, developers are actively working on fully autonomous drones and this will be the center subject of our work.

## 1.3 Previous Workd

Realizing the potential interest of fully autonomous drones, the Polytechnic School of Louvain launched a long-term academic project with the aim to develop technologies that would enable low-cost UAVs to navigate autonomously in indoor environments.

In the previous years, several master theses tried to bring a robust solution for localisation and mapping of autonomous drones. From 2012 to 2015, the master's theses were focused on visual SLAM to build 2D map based on keypoints and inter-drone communications. From 2015 to 2018, students worked on the adaptation of previous work on ROS and extended the SLAM algorithm for 3D mapping. They were all based on the same machine namely AR.Drone 2.0 from Parrot. Those drones are equipped with a set of sensors including a monocular camera, an inertial measurement unit and various localised range finders. Here is a summary of the last two theses in this long-term academic project:

**Boris Dehem 2016-2017** In 2016-2017, Boris Dehem in addition to the previous works before him, implemented a visual SLAM based on the SIFT/SURF detector and descriptor. This algorithm gathers key-features from its environment and places them into a 3D map giving the robot awareness of its position. The placing is made by a bundle adjustment based triangulation and gave interesting results. However the code suffered

from important lags due to the heavy computation needed and leads to a map whose interpretation is complex, especially to the human eye.

**Louai Kechtban 2017-2018** In 2017-2018, Louai Kechtban based his work on the previous thesis but replaced SIFT/SURF by another detector/descriptor namely ORB. In addition to this, he had the goal to explore new ways of improving the effectiveness of SIFT/SURF and to speed up the code with methods like "Divide and Conquer" or migrate from openCV2 to openCV3. The performance did improve, however, the tests did not give the expected results and the author recommend to stay with SIFT/SURF.

### 1.4 Solution

The previous works were all based solely on the use of a camera for map construction. It showed the limitation of the AR.Drone 2.0 platform in the context of visual SLAM. The conclusion of Louai Kechtban gave the incentive that another paradigm has to be explored and the addition of new sensors could be a good starting point.

In this master's thesis, we will study the potential of a system combining a monocular camera with a LiDAR sensor. The main goal is to map out the overall picture and possibilities of such sensor system.

#### 1.4.1 Goals of the project

This work aims to explore the addition of an alternative sensor, the 2D LiDAR. The objectives are:

- To chose and buy a suited LiDAR sensor for a possible implementation on drones.
- To build a new mobile platform computationally more capable than the hardware of the AR.Drone 2.0.
- To fuse the data of the LiDAR sensor and the feed of a monocular camera in order to bring depth information to the image feed.
- To test the gains of having a LiDAR in the context of a well known SLAM algorithm.
- Dress the possible implementation/improvements going forward.

### 1.5 Outlines

This work will cover the following subjects:

- We are going to give a quick summary of the state of the art concerning drones, the different sensors currently used as well as the different SLAM approaches available today.

## *1 Introduction*

- We will then explain why a particular LiDAR sensor has been chosen for this project and give technical information about our ROS implementation and the mobile platform.
- Explain in 3 consecutive chapters how we proceeded with the fusion of LiDAR measurements and camera information with the help of a convolutional neural networks.
- We will highlight some interesting inherent advantages of having an on-board LiDAR for an autonomous drone.
- And end with a performance comparison of our LiDAR-camera fusion method combined with a well known visual SLAM and a monocular SLAM only.
- We will discuss the possible implementation/improvements going forward.

## 2 State-of-the-Art

### 2.1 Drones

In this section we will briefly describe what the state of the art is in terms of drones and UAV technology. We will cover the pros and cons of two different general aircraft architecture and give a short example about one interesting but yet to be implemented drone architecture.

#### 2.1.1 Drones geometry

Drone or UAV [\[95\]](#) ( unmanned aerial vehicle ) in general, is an aircraft without any human pilot. In today's standards, drones can be divided into two categories when comparing the most common flight methods. Wing based and Co-planar Multi-rotor.

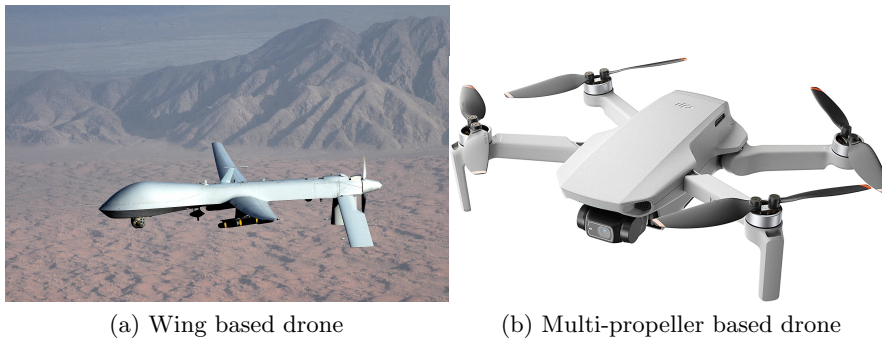


Figure 2.1: Two main drone geometry

#### Wing based Drone

This type of drone is based around the classical geometry of a regular aircraft and thus uses the lift effect of a wing to stay above the ground. This method of displacement is very energy efficient as the drone can benefit from its speed to stay in flight and is thus well suited for long range missions such as large land cartography ( thousands of square kilometers), military missions and so on. In this paper we will not go further about this geometry as our Master thesis is mainly oriented towards indoor optimized drone.

### Co-planar Multi-rotor Drone

An omnidirectional drone is very similar to a helicopter as it has no wings and uses multiple propellers to stay above the ground. The only difference between the two, is the fact that one has a perpendicular propeller to counteract the angular momentum of the main one and the other has a pair number of propellers rotating in opposite direction in order to keep the angular momentum to zero. This kind of drones are more energy hungry than the wing based drones but benefits from a much more precise flight behaviour. This makes those drones well suited for indoor flight as the lack of space creates the need for fine movements and precise speed profiles. They are used for small landscape cartography [6], agricultural surveillance and pesticides dispensing [24] and indoor industrial facility check [39] for example.

### Propeller configuration

The concept of multirotor configuration has many advantages and no two rotor configuration are created equal. Having more than two rotors enables a very accurate 3D displacement of the machine without any moving parts on the propellers, which is a big advantage compared to the single rotor configuration ( Helicopter ). Having no moving parts improves drastically the simplicity of the mechanic and thus reduces the number of possible failure points. One trade off of this technology is the linear increase of energy consumption linked to the increase of the number of rotors. More rotors means more thrust but also a reduced battery life.

The most common architecture for multi rotor drones are 4, 6 and 8 rotors with all motors on the same plane. While a 3D object in free space has 6 degrees of freedom: 3 for rotation and 3 for translation, the drone can't control them all at the same time. When the drone has to go forward, it has to tilt in the desired direction by increasing the thrust of one pair of propeller regarding the other. One can then conclude that not all degrees of freedom are independent and it said that the system is under-actuated. One representation of the system can be found in [2.2].

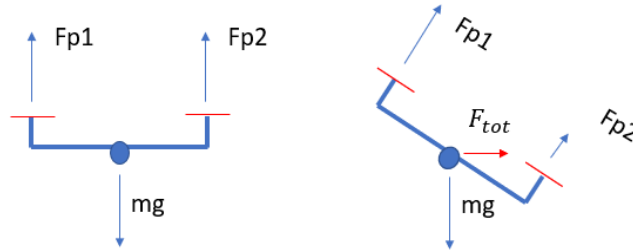


Figure 2.2: Under actuated propeller configuration

There is a lot of possible propeller configurations and each of them are suited for specific tasks. The most common setup is the "quadcopter" configuration [2.1b]. This setup is composed of one central unit with four equally spaced propellers rotating in opposite directions. This contraption is a good balance between performance and cost

and was the architecture used by the students in the previous related works at UCLouvain

Although in this paper the chosen geometry consist of four motors, depending on the task ( payload, speed and autonomy ), it is not necessary the best choice. And in order to design a new drone system from the ground up a complete study needs to be done as [74] states in their research paper.

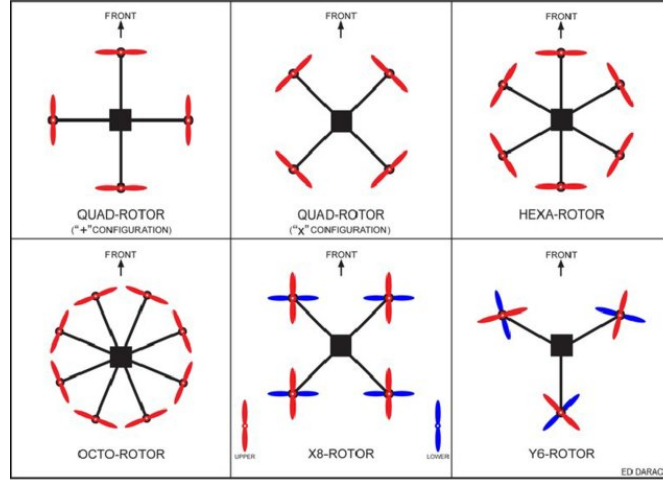


Figure 2.3: Summary of typical drone geometry

### Non-Coplanar multi-rotor drone, concept for the future

Another and more exotic concept is the non co-planar rotor arrangement as studied in [60] from Mahmoud Hamandi et al. Where a complete control algorithm and simulations were made to evaluate the viability of such devices and their practical use. An example of non co-planar multi-rotor drone can be found in [2.4].

## 2.2 Sensors

Now that we have a good overview of how drones are constructed and what their purpose are on a mechanical point of view we should not forget that this is only a small part of what the complete system is. Nowadays, drones have to be autonomous and by definition, aware of their environment. To fulfill this, they are equipped with a variety of sensors connected to a main computing unit and this section aims to describe the latest sensor technology used for autonomous drones.

### 2.2.1 RGB Camera

A cheap way of giving a drone perception about its environment is by coupling it to a camera sensor. With today's technology and with improving manufactures techniques in

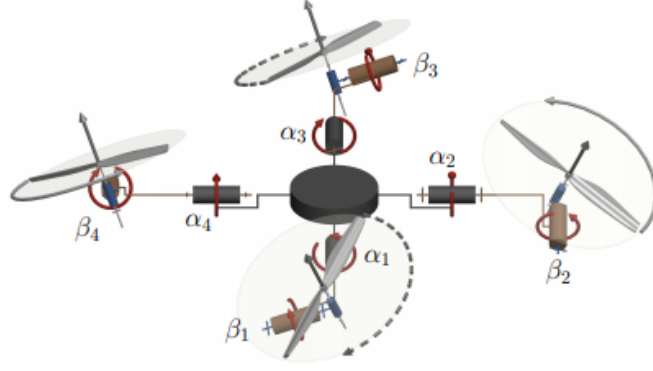


Figure 2.4: Example of non-coplanar geometry

the IC world, even a sub 30 euros camera module [33] is capable of outputting visual data with high frame rate ( 30 fps ) and with high definition image quality ( 1920x1080p). This sensor can give a human like representation of what it sees and gather a massive amount of information at the same time.

This characteristic has its advantages and its downsides. The advantage is that this sensor can accurately capture the environment as it is without any processing required by the module and thus results in a high refresh rate information stream. However, the raw nature of the information being published by the sensor requires a lot of heavy calculations from the MCU of the robot.

Even when taking into account the heavy computational cost of image data, most of nowadays's consumers drones have a built in camera. This can be explained by the increase of the computational capability of micro-controllers and on-board computers such as the ESP32 from Espressif [31], Raspberry Pi 4 [34] and its new built in GPU cores or the immensely popular Jetson Nano board [73] from NVIDIA which has 128 CUDA cores dedicated to computer vision and neural networks.

### 2.2.2 LiDAR

"LiDAR" stands for "Light Detection and Ranging" and works by flashing a beam of light on a surface and, depending on the technology (the different working principles will be discussed in Chapter 3), computes the depth of the surroundings. Some of the commercially available LiDAR sensors are mounted on a rotating base and can thus sample the environment over 360° other are fixed and flash the environment as would an camera. There are 4 main categories of LiDAR on the market today and we are going to give a brief explanation and some examples for each of them in the following subsections.



### Single beam LiDAR

A single beam LiDAR sensor is a single light emitting diode coupled with a very sensitive light sensor. The diode sends a light impulse to a targeted area and calculates the time of flight of the beam when bouncing back to the light sensor. Using the light's speed constant and fast timers, the device is capable of evaluating distances from several hundred meters down to the precision of a centimeter.

### 2D plane LiDAR

A 2D single plane LiDAR is a single beam sensor mounted on a rotating axle. It achieves the same task as the single beamed one but extends the measurements on a 2D plane 360 °around the carrier. An scan example of such a sensor can be found in figure [2.5](#). This sensor gives quite a big amount of data to process for the MCU ( Refresh rate around 20 fps for mid-range sensors ) and is often coupled with a preprocessing unit who its task is to filter out the outliers from the scans.

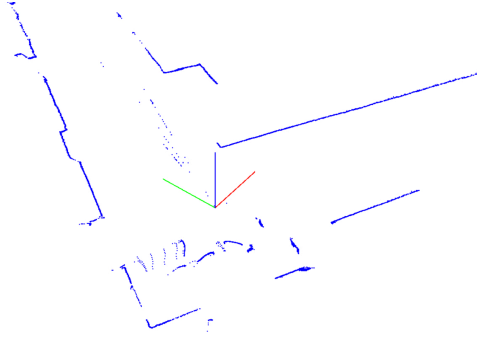


Figure 2.5: Scan of a single plane 2D LiDAR

### 3D multi plane LiDAR

The 3D multiplane LiDAR is the result of the addition of a third plane of measurements when sampling ranges. One can compare it to multiple 2D LiDARs stacked together and is a relatively new technology. The main draw back of this sensor is that despite its great precision and high refresh-rate, the big amount of data outputted requires a powerful CPU in addition to a high cost ( speaking of several thousands of dollars).

### Solid-state LiDAR

The Solid-state LiDAR is an array of light emitting diodes that send out bursts of photons in specific patterns and phases to create directional emission, of which the focus and size can be adjusted [\[28\]](#). This device is compact and robust to vibrations. Their refresh rate can be very high (more than 10 scans per second). They are more precise than rotating LiDARs as there are no moving parts subject to uncertainties. One drawback of this technology is that the beam is directional and does not cover a 360 °field of view.

They became quite popular with their integration by Apple in the iPhone 12 Pro as help sensor for augmented reality apps.

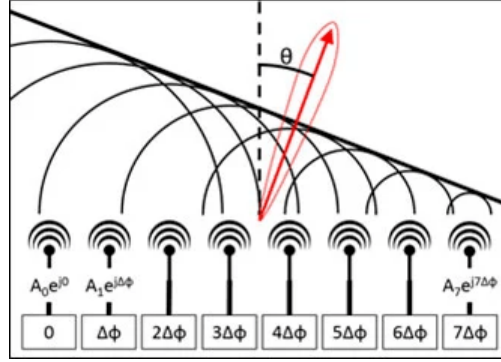


Figure 2.6: Optical Phased Array. Image courtesy of Analog Photonics

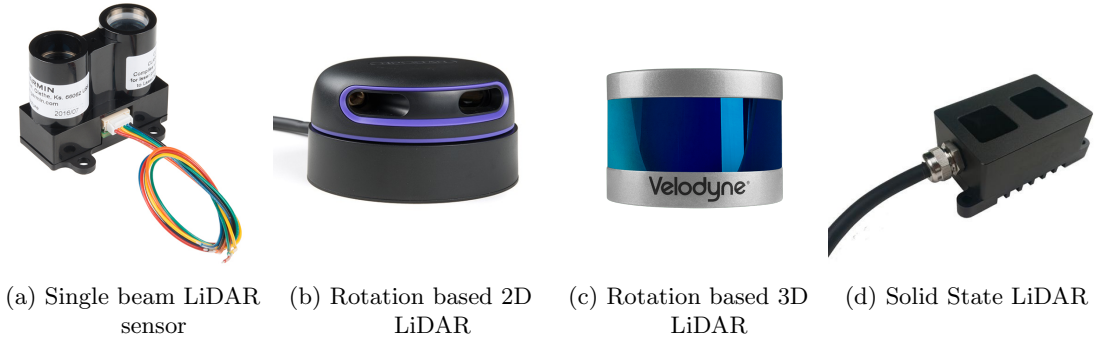


Figure 2.7: Different LiDAR types

### 2.2.3 Ultrasonic Sensors

The principle of this sensor is similar to the one from the single beam LiDAR but uses sound waves instead of light. It can be very accurate (  $[mm]$  precision ) with ranges from  $30[cm]$  to about  $5[m]$  for the high-end ones. The drawbacks are first, is as it uses sound to measure its environment. Second, it cannot work under vacuum. And third, sensing accuracy is affected by soft materials, temperature changes and if multiple sensors are used at the same time the measurement suffers from sound wave interactions.

### 2.2.4 RGB-D Camera

#### Coded light cameras

Coded light RGB-D cameras were first introduced in 2009 by Microsoft with the Kinect sensor. The concept was to project a pattern of multiple IR points on a surface

and then with an IR sensor and a RGB camera compute a depth measurement for each RGB pixel giving as a result an RGB-D image. As the technology of the RGB-D camera evolve, new IR light patterns are used and they all have their weakness and strength. The intel RealSense SR300 uses vertical IR lines instead of points for depth sensing. This kind of pattern is very accurate in short range conditions ( sub meter range ) [40] while the dot pattern is more suited for medium range depth sensing ( one to 3 meters ).

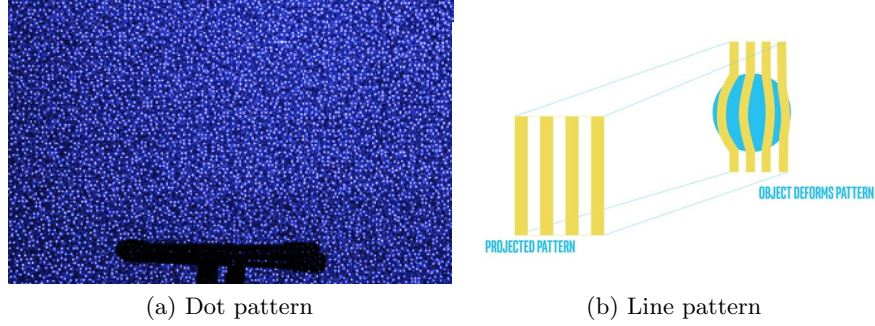


Figure 2.8: Different IR light pattern

### Stereo IR camera

IR light coded cameras are a great solution for short range depth sensing but are easily disturbed by the change of light intensities and parasitic sources of IR light. That's why manufacturers came with another concept, the stereo IR depth camera. It uses two stereo camera separated by a known distance in order to measure depth. This contraption is insensible to IR disturbance as any IR light can be used a depth information.

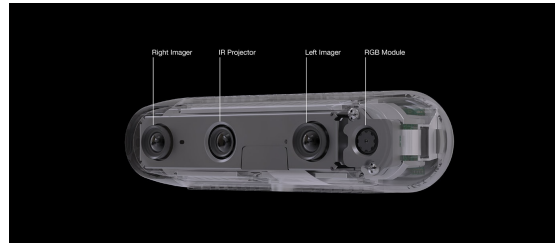


Figure 2.9: Stereo IR camera D435 from Intel

### 2.2.5 Inertial Measurement Unit

One useful sensor for drone application is the Inertial measurement unit. This sensor can measure acceleration in the 3 spatial dimensions and is often coupled to a MEMS gyroscope. The whole sensor setup can be contained inside a single IC the size of rice grain and makes it suited for very light weight drone application. The inertial measurements

provided by the IMU are often used to predict future positions of the robot in space and can then reduce the overall uncertainty of its location ( Kalman filter).

## 2.3 Sensor fusion

Sometimes one sensor is not enough for a drone to gather information about its environment. Each sensor is capable of providing his own set of data and no sensor is perfect. For comparison, the human body is packed with sensors and they all works together to create an accurate map of the environment. The human body is creating what we call "Sensor fusion". The concept is reused in robotics with the goal to improve the measurement capability by increasing the overall sensor accuracy or by creating a whole new type of more capable data. As the main sensor used in this paper is the 2D LiDAR, this section aims to describe the latest concepts and fusion techniques around it.

### 2.3.1 LiDAR - RGB Camera Fusion

One limitation of the 2D LiDAR is that a lot of information from the environment can't be sampled by the laser beam. This is explained by the fact that the laser only rotates on one axle and only samples points that are on the same plane. This would not be an issue for a wheeled mobile robot as his environment is mostly 2D but when talking about a flying robot things becomes a lot more complicated. The robot has to gather data in 3 dimension and the need for a 3D sensor becomes essential. To overcome this problem, Jieyu Li et al. [54] used a combination of 2D LiDAR point clustering and RGB vanishing point detection to reconstruct a 3D scene form a 2D image.

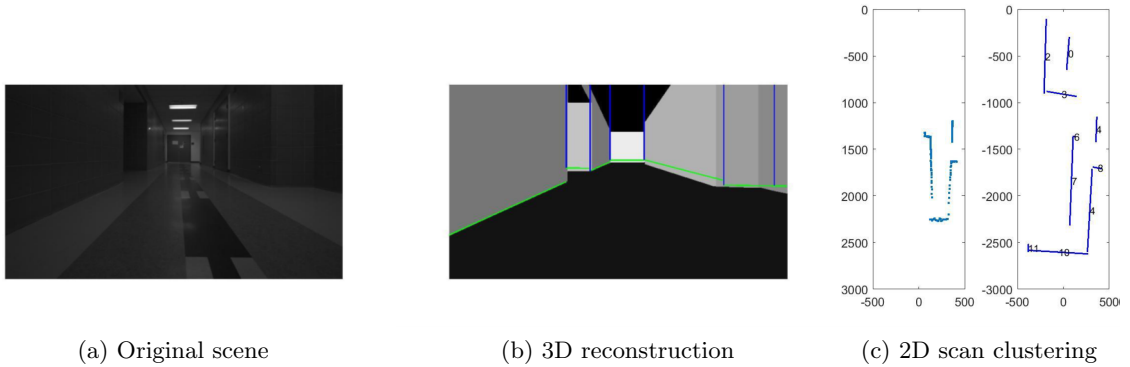


Figure 2.10: 3D scene reconstruction from [54]

Instead of combining directly two sensors to create one unique enhanced data stream, one can also use two sensors independently and then combine the relevant information that they provide for localisation or obstacle avoidance. That is what Chen et al. [13] did with their interesting approach of using one sensor at a time depending on the environment context. Two localisation algorithms where running simultaneously using

the LiDAR for the first one and the RGB camera for the second one. When one could not provide enough information about the surroundings, the other one took the lead and its results were stitched to the measurements of previous algorithm using a least squared method. This method overcomes the localisation inaccuracies of a 2D LiDAR linked to the crossing of ramps and stairs as the device will be tilted and generate incoherent data between two scans.

One last interesting approach proposed by Keisuke Tateno et al. [87] that will be one of the main interest of our Master thesis, would be the use of convolutional Neural Network on RGB images to get depth prediction. This solution gave interesting results but was not accurate enough for real world usage. One possible improvement would be the combination of their semantic segmentation with LiDAR data in order to reconstruct more accurately indoor scenes.

## 2.4 Simultaneous Localisation and Mapping

Now one has a good overview of what sensors we have to our disposal and what are, on a non exhaustive point of view, the techniques to combine them for increased measurement accuracy. The next important section to cover would be how to use those sensors for a simultaneous localisation and environment mapping. This section aims to describe and classify the most popular and promising SLAM algorithms for a better understanding of what has still to be developed. This section will be based on the excellent papers of Maksim Filipenko et al. [32] about various SLAM algorithms for mobile robot and César Debeunne et al. [26] who focuses his study on visual based SLAM only.

### 2.4.1 LiDAR SLAM

LiDAR sensors are a first choice for performing SLAM in an unknown environment. The simplicity of their data stream and the high refresh rate that they provide pushed researchers to develop a lot of different algorithms. The most popular LiDAR based SLAM algorithm is GMapping developed by Giorgio Grisetti et al. [36] in 2007 and uses the concept of Rao-Blackwellized Particle Filters to construct a map of possible positions of the robot. This map then decreases in possible positions over time as the robot scans its environment collapsing the uncertainties of its pose.

While Gmapping does not work in 3D environment and does not use the fast refresh rate of the modern LiDAR sensors, a new approach was proposed by Stefan Kohlbrecher et al. [48], Hector SLAM. This new implementation of a LiDAR based SLAM is advertised as very low demanding in computational resources and can compensate for unwanted moves of the LiDAR into the 3 dimensions of space. To achieve this, they coupled the scanner to multiple sensors such as a Inertial Measurement Unit, a GPS or even a compass and applied an Extended Kalman Filter to estimate a 6DoF pose inside the 2D map. The algorithm uses a special implementation of the scan matching concept and tries to stitch together multiple preceding scans by applying a Gauss-Newton approach.

Hector SLAM has many advantages over Gmapping but also shares one big flaw. It cannot close the map accurately on itself and it can be seen in [2.12](#)

It is important to remember that LiDARs are not perfect and that measurement errors can accumulate over time. It is especially true when the subject travels multiple times at the same location but through different paths. This problem is called "the loop closure problem" and was popularised by Wolfgang Hess et al. [\[37\]](#) leading to the creation of the "Cartographer" algorithm. It is based on the same principle as Hector slam but instead of matching the scans with the preceding ones, it matches them with all acquired scans when the variance of measurements falls under a particular threshold. This leads to the closure of the map and corrects any measurements drifts accumulated over time.

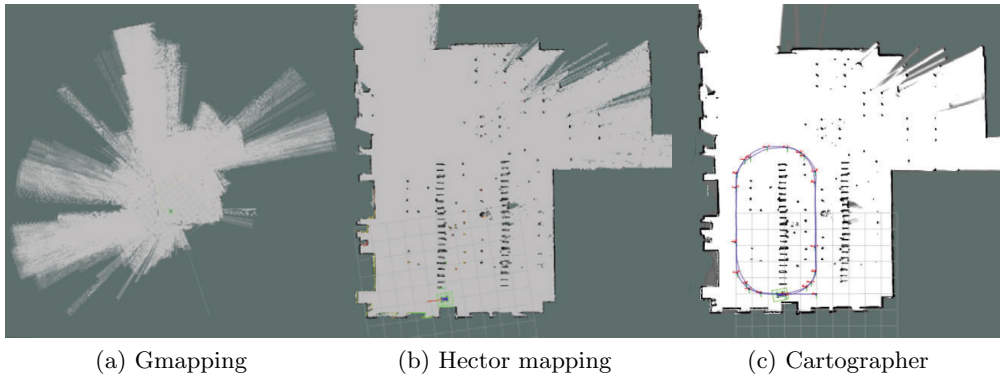


Figure 2.11: Different SLAM generated maps from [\[32\]](#)

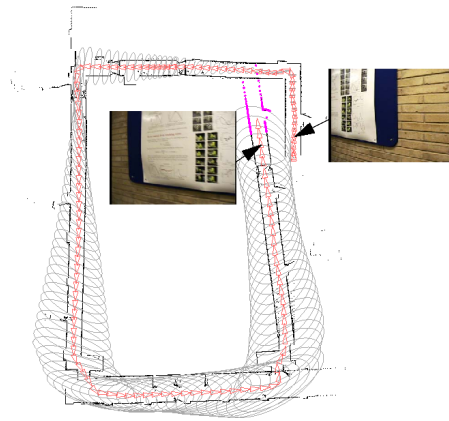


Figure 2.12: Loop closure problem explained in [\[72\]](#)

### 2.4.2 Visual SLAM

Visual SLAM methods are very popular nowadays because visual sensors are inexpensive, have a high refresh rate and can gather a large amount of information about the environment. The objective of visual SLAM is to estimate the relative movement of the sensor (the robot) by analysing the movement of the pixels of the camera between each frame. There are multiple ways to perceive movement such as the detection of important pixels in the image and tracking their movement down. This method is said to be "feature based visual SLAM". Other methods use the complete set of pixels inside the image for mapping and this approach is called "direct visual SLAM". Some methods use in addition to the RGB image information from a depth or an event camera.

#### Feature based SLAM

Feature based SLAM are a group of algorithms that relies on key features embedded in images. The algorithm passes each image from the video stream into what is called a "feature detector" and a feature descriptor in order to extract relevant invariant information about the surroundings. The feature detector, selects the interesting key-points and the feature descriptor uses those key-points to compute their light weight description for further use. As the environment moves around the robot, the gathered features are tracked between multiple images and this creates a cloud that will be used to build a map. There are various features detectors such as SIFT (Scale-Invariant Feature Transform) [58], FAST (Features from Accelerated Segment Test) [90], AGAST (Adaptive and Generic Accelerated Segment Test) [61], ORB (Oriented FAST and rotated BRIEF) [79] or SURF (Speeded Up Robust Features) [7]. And they can be coupled with various feature descriptors such as the descriptor part of SURF (Speeded Up Robust Features), DAISY [89] and HOG (Histograms of oriented gradients) [23].

In order to give the reader an example of what a feature detector and a feature descriptor is, we will briefly cover the theory behind SIFT (detector) and SURF (descriptor).

**SIFT** What makes SIFT a useful detector, is the fact that it is scale invariant, insensitive to rotations, illumination and robust against changing view point. The first step of the method consists of the detection of scale-space extrema. The algorithm makes multiple down-sampled copies of the original image and passes them through a Gaussian filter  $G(x, y, \sigma)$  of linearly increasing standard deviation. This creates what the author calls a scale space. In each down-sampled image all details that are bigger than the corresponding standard deviation  $k\sigma$  are smoothed out. A representative example can be found on figure 2.13.

Once the space is constructed, the goal is now to compute the local extrema in the difference  $D(x, y, \sigma)$  of each filtered image  $L(x, y, \sigma)$  following this equation.



$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \quad (2.1)$$

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \quad (2.2)$$

$$= L(x, y, k\sigma) - L(x, y, \sigma) \quad (2.3)$$

A representation of the result can be seen on 2.14. Each sample point is compared to its eight neighbors in the current image and nine neighbors in the scale above and below. It is selected only if it is larger than all of these neighbors or smaller than all of them.

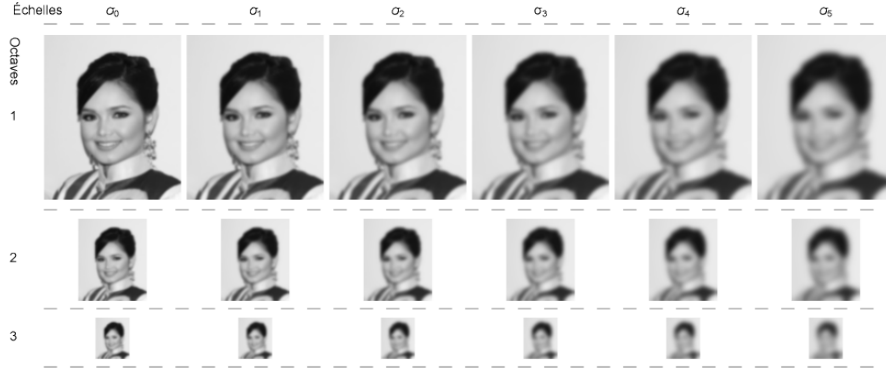


Figure 2.13: Scale space representation [93]

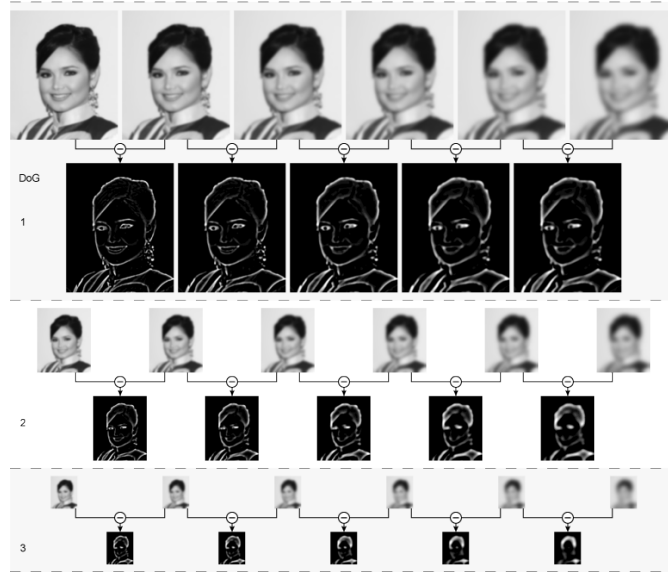


Figure 2.14: result of the computation of  $D(x, y, \sigma)$  [93]

**SURF** SURF is the speed up version of SIFT, it constructs a square region around the key-point of 20 pixels squared. Then the region is split in 4X4 regions containing 25



pixels each. The descriptor applies a Haar wavelet response into the x and y direction called dx and dy. It then stores  $\sum dx$ ,  $\sum dy$ ,  $\sum |dx|$  and  $\sum |dy|$  for each of the 16 regions giving a 64 vector as key-point descriptor. See figure 2.15

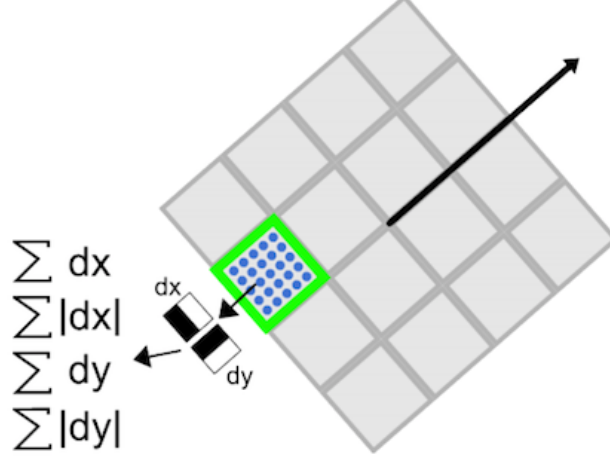


Figure 2.15: SURF descriptor representation, credits to: [41]

Feature based SLAM can be divided into two groups on how to handle the detected key-points, filter based and bundle adjustment. We will not talk about the filter based group as this technology is greatly surpassed by the bundle adjustment techniques. The first bundle adjustment attempt to create a SLAM algorithm that only uses a monocular camera is the one proposed by Georg Klein et al. [47] using FAST and gave interesting results but performance is not yet good enough for an arbitrarily picked environment. After some refinement of the method, one way of improvement was to add a way to perform loop closing on the generated map in addition to a better method of collecting key features namely ORB and this gave birth to ORB-SLAM.

ORB-SLAM(2), originally developed by Raul Mur-Artal et al. [68] is the most popular visual SLAM algorithm available today and is based around the feature descriptor ORB to gather its key features. It works with multiple sensors such as RGB-D camera or stereo camera and features a loop closure method making from it a solid SLAM option to chose from the visual SLAM space.

### Direct SLAM

Direct SLAM is a type of SLAM that uses directly the image without any features detector and descriptor. It is often based on photometric consistency as seen on figure 2.16 from [75]. In this category we have some popular algorithms such as DPPTAM [71], LSD-SLAM [30] or DSO [29].

One other way of using direct images is the use a convolutional neural network to guess the important features such as semantic segmentation for depth prediction. CNN-SLAM

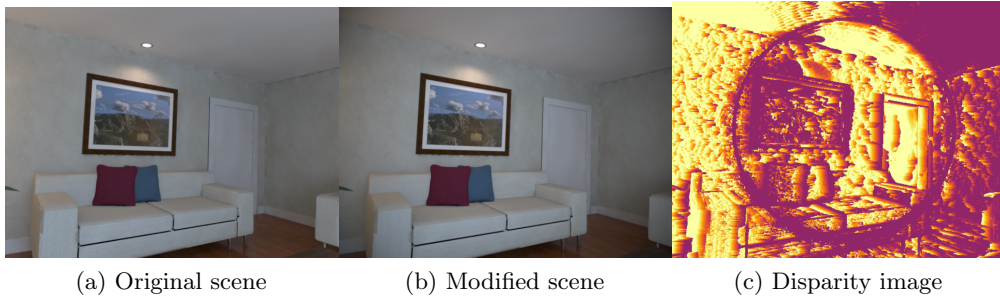


Figure 2.16: Example of photometric consistency

was developed by Keisuke Tateno et al. [87] and proved to be useful in high photometric disparity environment. Those algorithms on the other hand are not very user friendly as they often need a GPU to make them work but gives an opening to a possible monocular dense depth estimation neural network.

## RGB-D SLAM

RGB-D Cameras can give a tremendous amount of information about the environment but have limited range. This makes them suited for indoor environment and 3D map construction. A lot of previously mentioned algorithms already support the use of RGB-D sensors but are mostly based on feature detection in 3D space. One interesting approach was proposed by Renato F. Salas-Moreno et al. [80] and focused on plane recognition of a dense scene to reconstruct accurately a 3D map using an occupancy approach with on-line compression.

### 2.4.3 Sensor Fusion SLAM

In many cases, relying on only one type of sensor to perform SLAM is not enough. We are aware that LiDAR can accumulate measurement errors over time and that RGB-D cameras are very sensitive to changing light conditions and environment textures. It is to take the best of both worlds that new techniques have emerged to fuse the benefits of LiDAR and RGB-D camera measurements into one more robust algorithm. Multiple approaches were presented and we are going to focus on the most popular and established ones.

## Extended Kalman Filter SLAM

One way of improving localisation of a SLAM algorithm is to feed multiple sources of pose information into a switching mechanism who determines the best candidate to use for mapping. An EKF is often used to fulfill this goal. One interesting example of implementation of this technique was proposed by Yinglei Xu et al. [80] where they combined a LiDAR EKF SLAM with an RGB-D camera in order to tackle some unsuccessful visual tracking of the RGB-D camera.

The best example about this association of sensors was implemented by Elena López et al. [57] where they used on-board sensors of a drone to estimate the z-position correction factor and a 2D LiDAR to evaluate the x and y-position correction factors of different visual SLAM based on an implementation of an EKF.

### Improved Visual SLAM

Visual SLAM have proven to be very effective and that is why some people chose to improve this framework instead of rethinking the whole concept. When it comes to monocular SLAM, one weakness is that the algorithm has to guess the relative depth of each pixel in order to locate the founded key-features on the map. An expensive but interesting way of counteracting this problem would be the addition of a 3D LiDAR to the mix and stitching his depth measurements to the pixels of the monocular image. As the LiDAR has a lower resolution than the RGB image, a lot pixels do not get a depth measurement. In the work proposed by Varuna De Silva et al. [25], a Gaussian process regression was used to recover missing depth values in the RGB image. This leads to the emulation of an RGB-D camera type of SLAM.

### Improved LiDAR SLAM

Often when using LiDAR based SLAM in combination with a visual sensor, is the fact that the LiDAR is used for scan matching while the visual sensor is used for features detection. The most frequent problem with this type of SLAM, is the loop closure. One way of simplifying this process is to collect features with a camera and then making guesses to help the algorithm close the loop of the map. The implementation of this method can be found in the work of Zulun Zhu et al. [102].

### Concurrent LiDAR-Visual SLAM

The last method covered by this section is the concurrent SLAM algorithms. By running two different SLAM algorithms at the same time, using two different sensors, one can then merge the two maps created to obtain a more precise version of it. Youngwoo Seo et al. [83] implemented this mentioned method.

In terms of pure sensor fusion, Guolai Jiang et al. [42] implemented the most advanced fusion technique by combining the laser points cloud data and image feature points data as constraints, a graph optimization method is used to optimize the pose of the robot. At the same time, the BoW based on visual features is used for loop closure detection, and then the grid map built by laser is further optimized.

## 3 LiDAR Sensors

The integration of Light Detection and Ranging (LiDAR) sensor being a novelty for this year Master's thesis, it is important to go a bit more into depth regarding the sensor. This section has two main objectives : first, to provide the reader with a summary of LiDAR technology as such and, second, to outline the major characteristics of the LiDAR sensor used in our system.

More precisely, in this section, we will discuss the LiDAR sensor in itself ; firstly addressing its history, the technology behind it, the main types of LiDAR and some of the most relevant applications of such sensors. We will then address the selection of the LiDAR sensor employed in our set-up and its specifications. Finally, we will tackle the possible issues and limitations of the LiDAR.

### 3.1 Evolution of LiDAR sensors

In 1960, Theodore Maiman and his fellow scientists at Hughes Research Laboratory built and demonstrated the first naturally pulsed ruby lasers. Their invention of Ruby Laser caught the attention of the U.S. military who began research into military laser devices. The *Hughes Aircraft Company* introduced a first LiDAR-like system in 1961. Originally called Colidar (an acronym for "COherent Light Detecting And Ranging"), the system combined laser-focused imaging with the ability to calculate distances by measuring the time for a signal to return in the context of satellite tracking.

The following year (in May 1962), Louis Smullin and Giorgio Fiocco of the Massachusetts Institute of Technology (MIT) developed a method based on Maiman's technology to precisely measure the distance between the Earth and the Moon and, in parallel, Guy Goyer and Robert Watson of the National Center for Atmospheric Research (NCAR) applied those early lasers to the measurement of cloud heights. The first practical terrestrial application of a colidar system was the "Colidar Mark II" which is illustrated in Figure 3.1. Produced in 1963, this rifle-like laser rangefinder had a range of 7 miles and an accuracy of 15 feet.

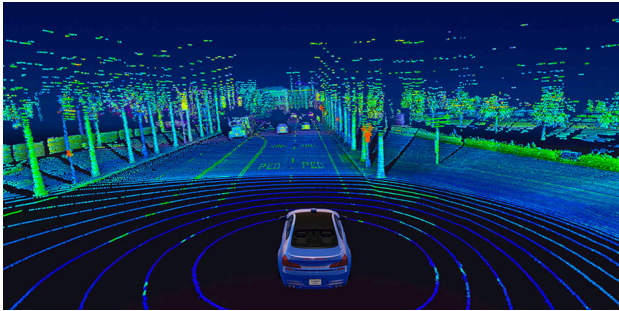


Figure 3.1: *Hughes Aircraft Company* engineer, Rod Smith, aiming the Colidar mark II, the world's first commercial LiDAR [70]

### 3 LiDAR Sensors

In 1971, the astronauts of the Apollo 15 mission used a Laser altimeter to map the surface of the moon, exhibiting to the general public the accuracy and usefulness of such technology. However, due to the high price associated with such instruments, it was mainly reserved for military and scientific research. With improvements in optical systems and signal processing in the 1980s, more mature products began to appear in the 1990s. From there, their popularity went on a steady increase. First, it became a standard for geographical mapping. In 2005, LiDAR systems made the headlines as the eyes behind the self driving cars in the US military’s DARPA Grand Challenge and again in the DARPA Urban Challenge in 2007. Gradually gaining the market, a series of laser radar manufacturers risen in the past two decades. With the active development in field such as large-scale surveying, UAVs, autonomous robots and self-driving cars, LiDAR has become a mainstream technology for surface data acquisition. As reported in some research, the market is supposed to reach 5.2 billion by 2022.

According to some experts, LiDAR technology solutions are still in the early stages of the Moore’s Law curve and the expected improvements in cost and performance will continue to strengthen and expand its applications. Nowadays, LiDAR plays a major role in various fields such as topography (geomorphology, altimetry and bathymetry), geosciences (oceanography, seismic risk, meteorology, atmospheric physics) and agriculture and environmental sciences (air pollution studies, agronomy and forestry), but also in archaeology, wind prospecting, air traffic control, automatic guidance of land or space vehicles, road safety, military and defense.



Among these different applications, geographical and atmospheric mapping are still the most common. However, from the roof of self-driving cars to the back of the latest smartphone, LiDAR is set to be a key sensor for many future technologies.

Figure 3.2: Point Cloud created by Velodyne LiDAR’s *Alpha Prime* sensor [55]

This brief overview of the evolution of LiDAR sensors over time is a synthesized compilation of various papers, primarily [4], [65], [96], [94], [8], and [70]. If the reader is interested in learning more about the history of LiDAR, we particularly recommend [4], [65], [96], [94].

## 3.2 Working principles

The main different types of LiDAR sensors have been introduced in the previous section. The major difference between them was which points of the environments are taken into

account. In this section, we will not focus on which points are data collected but rather *how* those data are collected by briefly explaining the working principles behind this technology. There are three main implementation methods regarding LiDAR :

- Time of flight LiDAR
- Phase ranging LiDAR
- Triangular ranging LiDAR

Inspired by [45], [56], [59], [91] and [82], we will summarize those three working principles.

#### 3.2.1 Time of flight LiDAR

The first category, Time of Flight (TOF), is the predominant method for application which implies large range and high accuracy. The concept behind time of flight is simple. A series of laser pulses are emitted, reflected by an object and then received back by the LiDAR, as depicted in Figure 3.3.

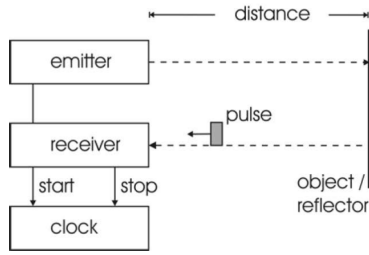


Figure 3.3: Time-of-flight distance measurement principle [45]

Knowing the speed of light, measuring the travel time allows to know the distance traveled. To reduce noise in the measurement, the times of flight of several pulses sent in quick succession are often averaged. The accuracy of the measured distance is directly linked to the accuracy on the time measurements. Indeed, the speed of light being very fast, in order to achieve high accuracy, it is necessary to rely on high level of electronics (especially high precision clocks and short light pulse) which generally result to more expensive systems.

#### 3.2.2 Phase ranging LiDAR

Another method is to measure the phase shift. In this case, a continuous laser wave is projected into the surrounding, modulating the amplitude of the laser emission signal. With the phase shift between the emitted and received sinusoidal modulated light, it is possible to determine the distance to the reflecting object. However, the discriminated phase difference can only be between 0 and  $2\pi$ , and therefore distance ambiguity can occur. Moreover, the modulation frequency of the transmitted signal is proportional to the radar ranging accuracy, and inversely proportional to the maximum unambiguous distance detected. To solve this problem, manufactured phase ranging LiDARs usually use multiple modulation frequencies to transmit at the same time. Higher modulation frequencies enhance the ranging accuracy while lower modulation frequencies improve the maximum unambiguity distance of the system. Figure 3.4 and 3.5 illustrate the phase shift measurement set-up and expected waveforms.

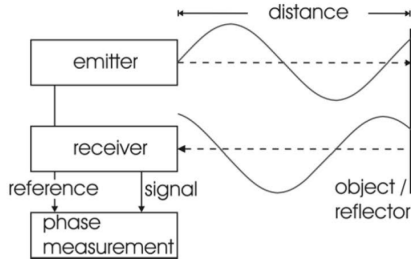


Figure 3.4: Phase shift distance measurement principle [45]

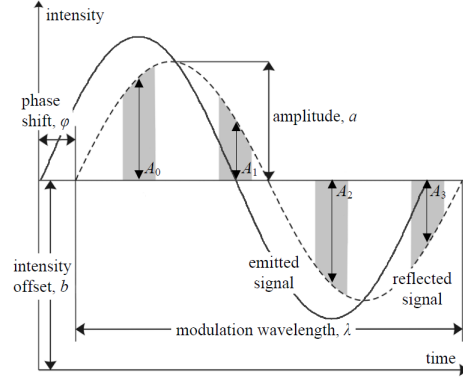


Figure 3.5: Phase shift between the emitted and reflected waveforms [45]

### 3.2.3 Triangular ranging LiDAR

Triangular ranging is a method that takes advantage of trigonometric formula. The light emitted by the laser after irradiating an object, is reflected back to a detector area of the device. Knowing the distance between the laser emitting the light and the detector and using geometric formula, the distance of the detected object can be derived.

Compared to the two first methods, triangulation has a shorter working range. This can be expected from Figures 3.6 and 3.7: the further away the object measured is, the smaller the position difference. Since the calculation of the distance directly depends on the angle, a smaller angle difference leads inevitably to a loss in accuracy (which is not the case with previous methods). Triangulation is also not able to get a signal-to-noise ratio as high as other methods and also lower sampling and frame rates. So why continuing to work with triangular ranging LiDAR? First, they are easier to implement and therefore, the cost of such sensors are significantly lower than with other working principles. Secondly, the accuracy of the triangulation method is very high at close range. Those different characteristics made them better suited for some application. By example, for applications such as drone-assisted topography or self-driving vehicles, the first two methods will be generally preferred. On the other side, triangular ranging method will usually be found, among other things, in various indoor robots such the popular robot vacuum cleaners. This triangulation principle is the basis of the RPLIDAR A3, the LiDAR sensor chosen for this year's thesis as depicted in Figure 3.8.



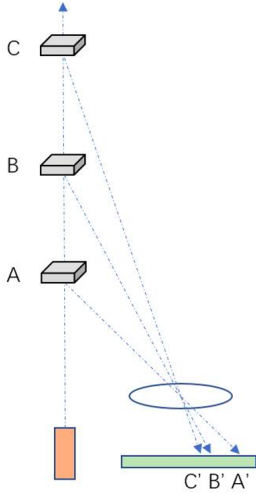


Figure 3.6: Triangulation distance measurement principle [91]

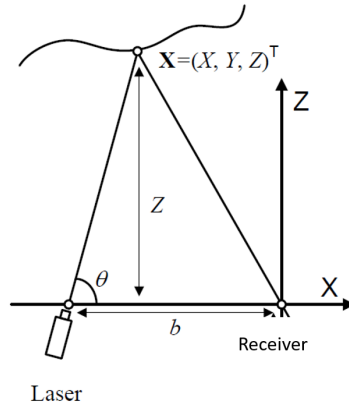


Figure 3.7: Triangulation distance measurement principle [45]

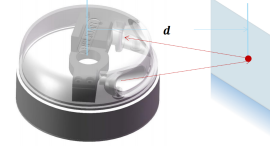


Figure 3.8  
RPLIDAR A3 operating by the triangulation method [15]

### 3.3 Selection of our LiDAR sensor : RPLIDAR A3

This section aims to provide the reader with an explanation of the LiDAR sensor selected for this Master's thesis, as well as its main specifications and features.

#### 3.3.1 Our selection

##### Why LiDAR?

LiDARs have become indispensable sensors for some established technologies or many still under research. In particular, many ground robots, self-driving cars and autonomous drones are putting forward the interest of these sensors. (For the anecdote term "LiDAR" returns more than 1.65 million results on Google Scholar while other sensors such as IMU returns 280 thousands results and RGB-D camera less than 50 thousands)

LiDARs feature many advantages that explain their frequent mention in these research areas. LiDAR sensors have a high update rate and high ranging accuracy, which is crucial for either a self-driving car or an autonomous UAV. In addition, it has other interesting possibilities, such as not being influenced as much by the light of the environment as other sensors, having the possibility of scanning a 360° view of the environment (which is interesting for detecting obstacles), mapping the environment and even being used in part for odometry. (But we will expand further on those different advantages and possibilities of LiDAR in subsequent chapters, mainly in chapter 9)

Finally, as students, from an academic point of view, we were genuinely interested in the study and implementation of such a sensor.



#### Why a 2D-LiDAR model?

First, we have to address the choice of selecting a 2D LiDAR. The main concerns in selecting a sensor in order to be later implemented on UAVs to perform SLAM in indoor environment (and so GPS-denied environment) :

- Price : the sensor ideally should be as low as possible and cost-effective enough to be included on a drone
- Weight and volume : in terms of specification, to be integrated in or on a drone, the sensors have to be light enough and not take too much room
- Accuracy and information update rate : the performance of the drone to precisely localize itself and potential obstacles as well to be able to move in its environment will directly be linked to the performance of its sensors

With these following criteria in mind, we can discuss what kind of LiDAR sensor is a better fit. (As a reminder, the different types of LiDAR sensor were summarized in the State-of-the-Art section). As the objective is to have information about the 3D environment, the first sensor that comes to mind is probably a 3D LiDAR sensor. Indeed, a 3D LiDAR is the best in terms of giving information about the surrounding 3D environment and already utilized in many applications (mainly self-driving cars) to create accurate 3D point clouds. With by far the best performances, its main drawbacks are its high price and heavier weight. While those drawbacks are less detrimental when it comes to cars, regarding UAVs, those cannot be disregarded.

This leaves us with 2 viable options : 2D scanning LiDAR and the solid-state LiDAR. (For obvious reasons, a single beam laser would not be the best fit for such application). Moreover, solid-state LiDAR and 2D scanning LiDAR serve very different purposes. One is not better than the other and a solution could integrate the two as their information can be complementary. As we wanted to have a 360° information of the environment (which could be of interest for loop closure and odometry) we decided to prefer a 2D scanning LiDAR. In addition, the main goal of this Master's thesis is to explore the possibilities to implement cost effective and accurate hardware + software (sets of sensors and needed algorithms) solutions for accurate SLAM of indoor drones. According to us, a 2D scanning is a better fit and could more easily replace a 3D LiDAR that would have been ideal for this kind of application if weight and cost were not limiting factors. This affirmation is reinforced by the fact that many similar projects to implement drones with 2D or 3D scanning LiDAR and, in parallel, many studies are looking at different solutions to explore cost-effective alternatives for 3D scanning LiDAR based on 2D scanning LiDAR to create accurate 3D point clouds.

Those different investigated solutions will be later discussed in subsequent chapters. As mentioned in the introduction, this is one of the main challenges: **being able to create a usable 3D point cloud based, in part, on a 2D scanning LiDAR for use downstream by a SLAM algorithm.**

#### Why the RPLIDAR A3?

Various 2D scanning LiDARs are available on the market with different price range and specifications. One company that seems to be very popular in similar projects is *Hokuyo Automatic*, a global manufacturer of sensor and automation technology which is particularly known for its 2D and 3D scanning laser range finders for use in automated guided vehicle (AGV), UAV, and mobile robot applications. Nevertheless, Hokuyo's LiDARs are much more expensive than LiDARs from some other manufacturers and this often even for specs - on paper at least - equivalent.

Looking for cheaper alternatives, we stumbled upon *Slamtec* which, which focus on the development of the core sensors regarding Slam and the related software. They specialized themselves in manufacturing low-cost LiDAR sensors. Figure 3.9 establishes a comparative of the different LiDAR sensors produced by *Slamtec*.

| Item                | Measurement Performance |                          |                    |                  |                     |
|---------------------|-------------------------|--------------------------|--------------------|------------------|---------------------|
|                     | Distance Range          | Measurement Resolution   | Angular Resolution | Sample Rate      | Working Environment |
| RPLIDAR A1M8        | 0.15m – 12m             | <0.5 mm                  | ≤1°                | 2000 – 8000 Hz   | Indoor              |
|                     |                         | <1% of actual distance * |                    |                  |                     |
| RPLIDAR A2M6        | 0.2m – 18m              | <0.5 mm                  | ≤0.9°              | 2000 – 8000 Hz   | Indoor              |
|                     |                         | <1% of actual distance * |                    |                  |                     |
| RPLIDAR A2M8        | 0.15m – 12m             | <0.5 mm                  | ≤0.9°              | 2000 – 8000 Hz   | Indoor              |
|                     |                         | <1% of actual distance * |                    |                  |                     |
| RPLIDAR A3M1        | White object: 25 meters |                          | 0.225°             | 16 kHz or 10 kHz | Indoor & Outdoor    |
|                     | Dark object: 10 meters  |                          |                    |                  |                     |
| RPLIDAR S1          | White Objects: 40m      | 3cm                      | 0.313°-0.587°      | 9200 Hz          | Indoor& Outdoor     |
|                     | Black Object: 10m       |                          |                    |                  |                     |
| SLAMTEC MAPPER M1M1 | 20m                     | 5cm                      |                    | 7000 Hz          | Indoor & Outdoor    |

Figure 3.9: List and comparison of the *Slamtec* LiDAR sensors 97

### 3 LiDAR Sensors

RPLidar S1 and Slamtec mapper M1M1 are both on their category respectively. First, the S1 model uses TOF ranging method, which can be interesting for some application (mainly where large range needs to be taken into account) but at the expense of a higher price (and some trade-off on other specifications such as a lower sampling rate) [17]. Slamtec mapper, on the other hand, is a line of LiDAR sensors with built-in functions of simultaneous localization and mapping (SLAM) such as actively detecting and correcting closed-loop, and achieving 100,000 square meters high-precision map and pose. There exist two models in this line : the M1M1 and M2M1. The first is quite limited in term of performance and is to considered more like a developer kit, not suitable for our application [19]. The M2M2 expands on the S1 model with the built-in SLAM functions [20]. However, these built in functions are conceived for 2D SLAM, paying a higher price and adding more weight for built-in functions and the related data processing unit is useless are those functions would be much better integrated via external designed ROS nodes.

All the other *Slamtec* LiDAR sensors use the triangular ranging method. And so for similar prices, they offer better performances, especially for lower distance range. Among those different RPLidar models, we decided to go with the A3M1 model as it offers the best specifications and the price increase was more than reasonable. Better angular resolution, longer ranging distance but above all, the most important benefit is the possibility to double the sample rate. (This higher sampling rate will relevant for latter discussion). Overall, it is one of the best low-cost LiDARs and the specifications make it relevant and reusable for other projects if needed.

#### Why not use a RGB-D camera?

This question, and therefore the reasoning, is very similar to the question : "why not use Solid-state LiDAR?". Indeed, as stated for the Solid-state Lidar, RGB-D camera and 2D scanning LiDAR serve very different purposes. One is not better than the other and a system could integrate the two as their information can be complementary. Many projects such as [52] utilized both. According to us, both LiDAR and RGB-D camera were interesting sensors and could have gone used as a starting point. We decided to go with the LiDAR for a few reasons :

- LiDAR sensors development is increasing rapidly and so is its use in research and technological projects
- LiDAR gives data in 360° circular 2D plane (and updated at high frequency) which could be interesting for some aspects of later implementation (cfr Chapter 9)
- The RGB-D camera gives data in narrower window, without a shorter maximum distance range and the RGB-D camera system is usually larger.
- The RGB-D camera already gives a depth image (more on that later) and the implementation from the creation of the depth image to the SLAM algorithms are already well (which is both an upside and a downside) but will limit our contribution

### 3 LiDAR Sensors

- The use of LiDAR + monocular camera allows to utilize a sensor that is already often included in many drones : monocular camera
- We believe that the combination of Laser + monocular camera could lead to some interesting algorithm that would result into more interesting depth images and particularly with the possibility to use potential machine learning algorithms (more on that in later chapters)
- The number of research papers on LiDAR sensors solutions for similar projects are clear indicators of the potential interest of including LiDAR
- From an academic point of view, we were interesting to implement a solution with this sensor
- Finally, in the long run, a RGB-D camera can still be added to the system (to replace the prediction of the depth image by AdaBins (Chapter 7) but more on those topics in later chapters) and the current work would still stand.

#### 3.3.2 Specifications

The RPLIDAR A3 (Model: A3M1, illustrated in Figures 3.10 and 3.11) is a LiDAR sensor produced by *Slamtec* (Shanghai Slamtec Co.) as a "Low Cost 360 Degree Laser Range Scanner" as they market it. The system can perform 2D 360-degree scan within a 25-meter range taking up to 16000 samples of laser ranging per second with high rotation speed. In the subsection, we will summarize the main specifications of this sensor. For further information, please refer to the dedicated section on the official *Slamtec* website 16 or directly to the datasheet 15 and/or the user manual 14.



Figure 3.10: RPLIDAR A3 (Model: A3M1) 16

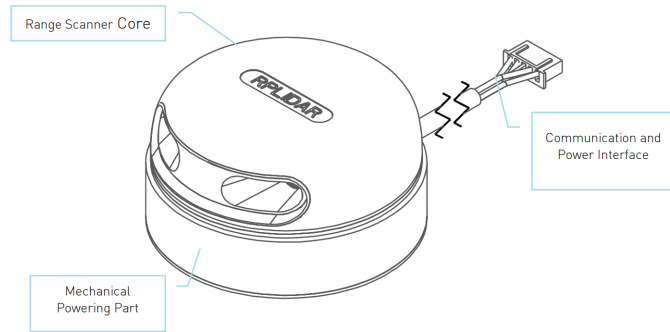


Figure 3.11: Schematic of the RPLIDAR A3 (Model: A3M1) 15

### 3 LiDAR Sensors

The dimension, key performance information and battery/electricity requirement are summarized in Tables 3.1, 3.2 and 3.3, respectively. More information can be found in [15] and a rigorous schematic of all the dimension provided by *Slamtec* can be found in the appendices.

|                  |                 |                  |
|------------------|-----------------|------------------|
| Height : 41 [mm] | Width : 76 [mm] | Weight : 190 [g] |
|------------------|-----------------|------------------|

Table 3.1: Size and weight [16] and [15]

|                                  |                        |
|----------------------------------|------------------------|
| Power consumption : 2.25 - 3 [W] | System voltage : 5 [V] |
|----------------------------------|------------------------|

Table 3.2: Battery/electricity requirement [16] and [15]

| Possible standard modes | <b>Enhanced mode</b><br>(ideal for indoor environments with maximum ranging distance and sampling frequency) | <b>Outdoor Mode</b><br>(ideal for both indoor and outdoor environments with reliable resistance to other light sources such as daylight) |
|-------------------------|--------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| Distance Range          | White object : 25 [m]<br>Black object : 10 [m]                                                               | White object : 25 [m]<br>Black object : TBD [m]                                                                                          |
| Minimum operating range | 0.2 [m]                                                                                                      | 0.2 [m]                                                                                                                                  |
| Sample Rate             | 16kHz                                                                                                        | 10 kHz                                                                                                                                   |
| Scan Rate               | Adjustable between 5-15 [Hz]                                                                                 | Adjustable between 5-15 [Hz]                                                                                                             |
| Angular resolution      | 0.225°                                                                                                       | 0.225°                                                                                                                                   |

Table 3.3: Key performance information [16] and [15]

### 3.3.3 Communication interface

The most relevant information concerning the communication interface are summarized in Table 3.4

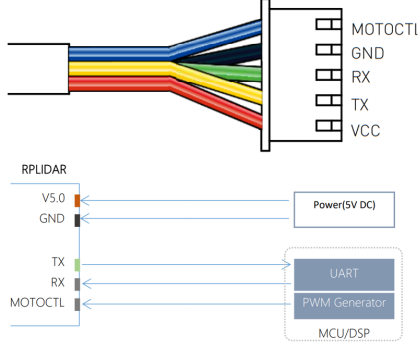


Figure 3.12: RPLIDAR A3M Communication interface [15]

|                                        |                              |
|----------------------------------------|------------------------------|
| <b>Communication interface</b>         | TTL UART                     |
| <b>Communication speed / Band rate</b> | 256 000 bps                  |
| <b>Compatibility</b>                   | Support former SDK protocols |

Table 3.4: Key communication information [15]

### 3.3.4 Sensor output

For each sample point, The RPLIDAR's data output will contain the information summarized in Table 3.5. The coordinate system for the output data considers the rotating center of the range scanner core as the origin and the rotation angle increases as rotating clockwise. Figure 3.13 illustrates this coordinate system and the information send by the sensor to the system. In our implementation, these data are recovered by one of the ROS nodes but this part will be explained in subsequent sections.

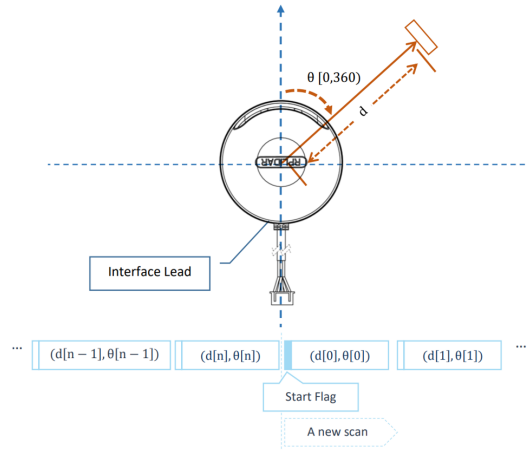


Figure 3.13: Coordinate system and form of the the output data [15]

### 3 LiDAR Sensors

| Data type  | Unit    | Description                                                                                     |
|------------|---------|-------------------------------------------------------------------------------------------------|
| Distance   | [mm]    | Current measured distance value between the rotating core of the RPLIDAR and the sampling point |
| Heading    | degree  | Current heading angle of the measurement                                                        |
| Start Flag | Boolean | Flag of a new scan                                                                              |

Table 3.5: Output information [15]

## 3.4 Limitations

This section explores the proper functioning of the RPLidar A3 and examines some limit tests, potential problem situations and other subtleties of the LiDAR sensor operation.

### 3.4.1 Interface for visualisation

*Slamtec* develops and provides its own extendable application : RoboStudio [21]. With its user-friendly interface, the application allow developers can build communication with the robot and get the robot's sensor data, pose, state, map information and other data. Figure 3.14 shows the RoboStudio interface. However, for now, the application is available for windows and android operating systems only. Moreover, even if it delivers an easy to use and polished way to visualize the data, it is not the most appropriate tool with our environment (running various ROS node on Linux).

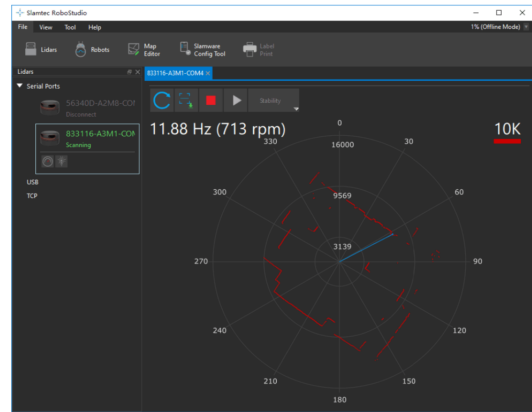


Figure 3.14: RoboStudio interface for RPlidars [15]

Indeed, we are working with the Robot Operating System (ROS) framework (for more information, please refer to chapter 5 discussing the setup). AS the ROS framework is very popular in robotic implementations, *Slamtec* also provides a ROS support through the RPlidar\_ros ROS node [18]. This node can be used to set different parameters as discussed earlier (such as the operating mode or the sampling rate) and returns a message containing the information described in the subsection 3.3.4. To visualize the data scanned by the LiDAR, we use rviz, a 3D visualizer designed for the ROS framework.

### 3 LiDAR Sensors

Figure 3.15a illustrates with the corresponding scene in Figure 3.15b.

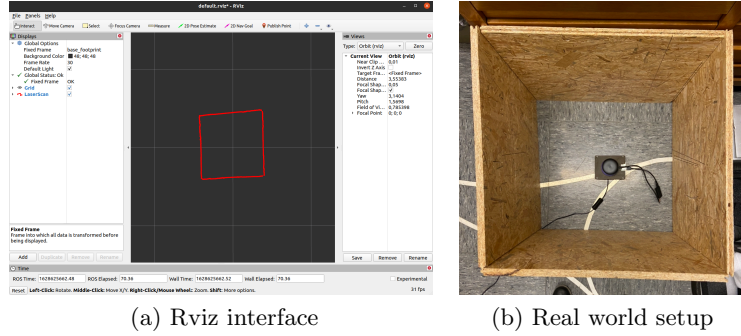


Figure 3.15: Rviz interface and corresponding real world setup

#### 3.4.2 Inherent problems

Before testing more challenging surfaces, it is important to comment on some on the observed behavior of the RPLidar in normal situation. Figure 3.16a outlines the point scanned by the LiDAR and received from the RPLidar\_ros node. Figure 3.16b illustrates a sample of the received array containing the information with first, the range in meters and second, the heading angle in radians. Looking at those data, we receive 1440 points with an applied 16 kHz sampling rate which match with a frame rate a bit lower than 10 kHz. By calculation or by looking at the printed value, one can observe that every scanned point is spaced by  $4.3610^{-3}$  radians, in other words,  $0.25^\circ$ . However, as one can see, a small window of data is missing. Indeed, as outline in the green rectangle, 4 points are actually missing (which equals to only  $1^\circ$  of blind spot). Moreover, as depicted in 3.16b, the last point, right after the  $2\pi$  mark (and so at the end of a given scan) is always infinite. The corresponding point is highlighted by the green circle in Figure 3.16a.

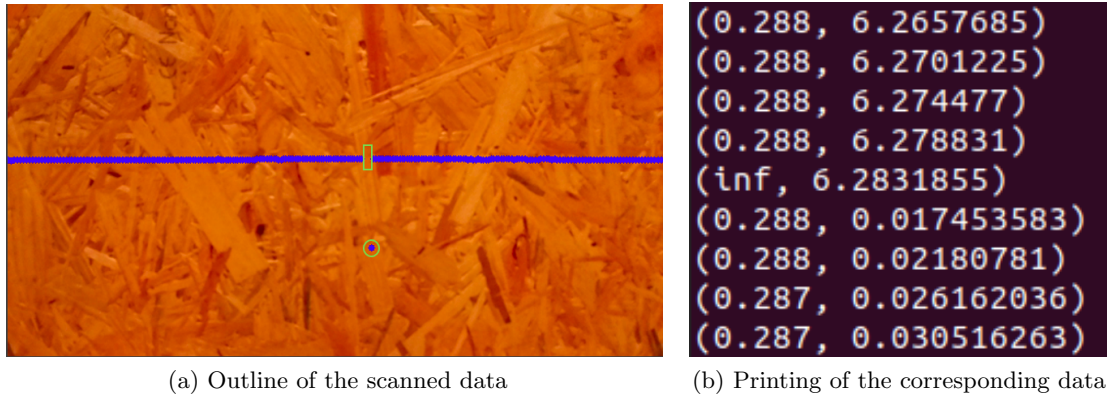


Figure 3.16: RPLiDAR A3 scanned data for the situation pictured in 3.15b

Another behaviour that could raise some issues is the non differentiation between



undetected objects (object non reflection, malfunctioning of the LiDAR etc.) and objects out of range. Indeed, all of these situation will lead to an "infinite" value in the received range array.

#### 3.4.3 Limit Testing

As LiDARs need the reflected light to estimate the distance between the reflecting surface and the sensor, LiDAR will not work properly in some situations. This section is not intended to be an exhaustive list of all the surfaces, material types, colors, and orientations that can be problematic but, more humbly, is intended to give some typical examples to give the reader a feel of the situations and elements that might prevent consistent LiDAR ranging data.

Throughout this section, we will use pictures to illustrate the discussed situations. In the figures of this section, we transposed the data received from the LiDAR into the camera's reference frame (more on this in Chapter 6) and highlighted points with coherent ranging distance in green and erroneous data (data points set to infinity by the LiDAR, as explained above) in red. More pictures regarding this section can be found in the appendices (note that some of them are a bit older and so the accuracy of the point highlighted can be slightly below of the ones featured in this section as the method developed in chapter 6 was still later improved).

As we will discussed mainly problematic situations leading to non-exploitable data from the LiDAR, Figure 3.17 and 3.18 illustrate a regular case without any erroneous data and one with some non-reflecting objects.

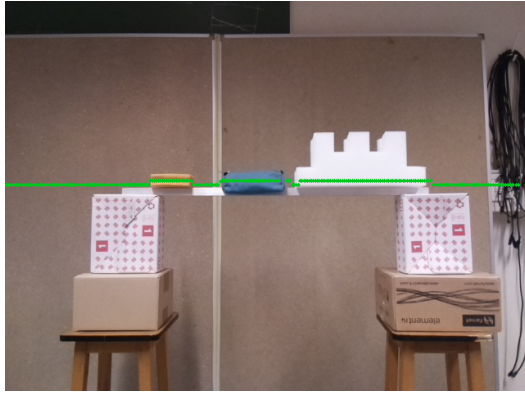


Figure 3.17: Classic case

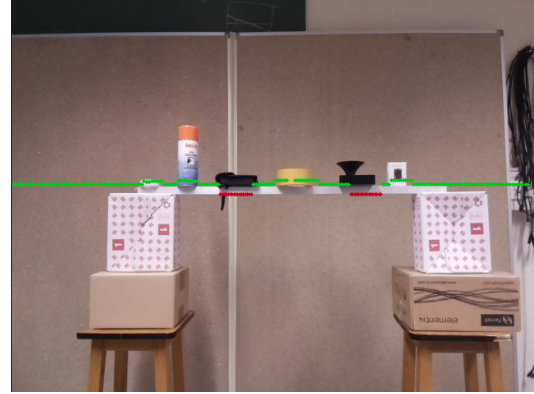


Figure 3.18: Case with erroneous data

#### Black objects and non reflecting surfaces

Some objects by their texture and colors does not reflect correctly the laser light. Those objects interact with the laser light but will not return decent reflected light for distance

### 3 LiDAR Sensors

ranging, leading to an undetermined "infinite" value for the corresponding scanned points. To illustrate some common encountered cases, we can cite the case of opaque plastic Tupperware (Figure 3.20), plastic foil (Figure 3.21), a glass cup (Figure 3.22), an object printed in polylactic acid 3D Printer Filament (Figure 3.23 and 3.18) or black styrofoam (Figure 3.24 and 3.25).

Some comments need to be made. Regarding the plastic foil, let us note that this is only the case for thick layers. In the case of one thin layer, the LiDAR laser just goes through the layer and is not influenced by the layer (and so no indication of the presence of the layer can be retrieved). (Object not detected at all forms another category that will address next). Concerning the glass, and some other glass materials lead to an undetermined "infinite" value, other glass surfaces such as windows, are "transparent" for the LiDAR in the sense that, as the one layer plastic foil, the LiDAR is not altered by the window and therefore the presence of a window can not be retrieved from the LiDAR data.

#### Glass and fine wire mesh

As mentioned in the previous paragraph, some objects let the light of the laser pass through without altering significantly the received light, thus making them undetectable by the LiDAR. This is by example the case for glass surfaces, the one layer plastic foil as mentionned above or other materials "finely fragmented" such as fine wire mesh.



Figure 3.19: *Fine wire mesh example*

### 3 LiDAR Sensors

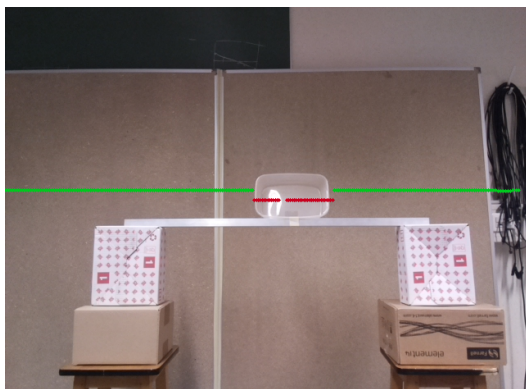


Figure 3.20: Testing : Tupperware

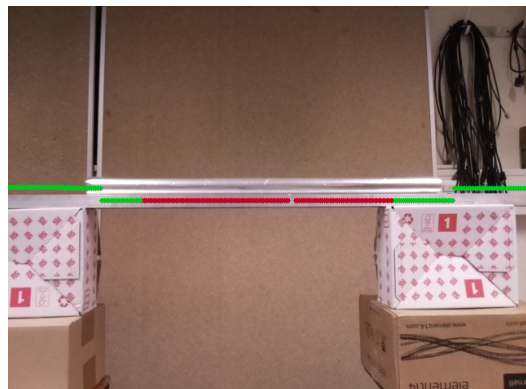


Figure 3.21: Testing : Plastic foil

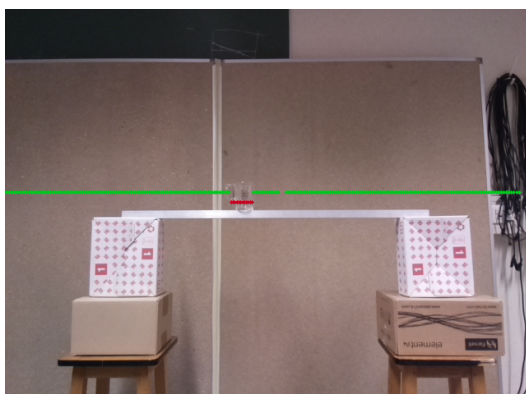


Figure 3.22: Testing : Glass cup

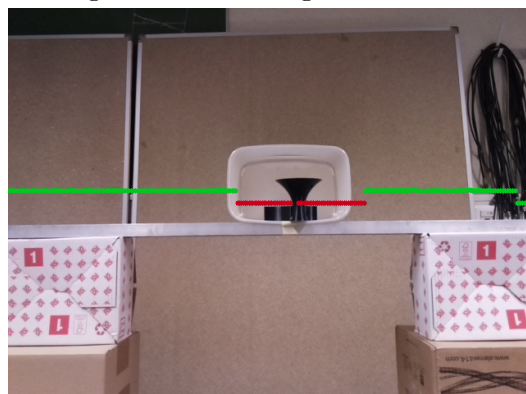


Figure 3.23: Testing : Black PLA

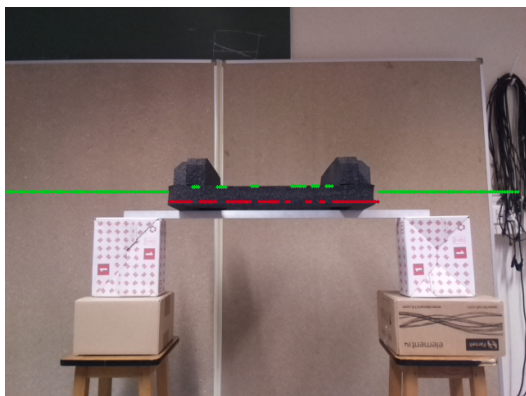


Figure 3.24: Testing : Black styrofoam

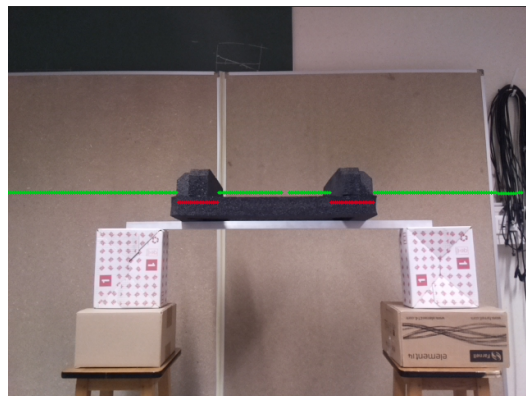


Figure 3.25: Testing : Black styrofoam



#### Relevance of the relative orientation of the surface

For some surfaces, the detection can be highly impacted by the relative orientation of the LiDAR and the designated surface. As one can see in Figure 3.26 and 3.27, depending on its orientation, the LiDAR will be able or not to determine the ranging distance of the back of the screen (computer on the left). This is also the case for the bright screen in Figure 3.28 and 3.29 or some "mirror-like" surfaces such as the back of the blackboard in Figure 3.30 and 3.31, a plexiglass surface in Figure 3.32 or even on the sides of a well polished cooking pot in Figure 3.33. In those instances, the light is mainly deviated, preventing the LiDAR to measure ranging distances. It is important to note that the orientation plays a major role. For example for the black back of the screen, the back of the blackboard or the pexiglass surface, when perpendicular to the LiDAR, the LiDAR can determine its distance but when the same surface is obliquely sensed, the LiDAR cannot retrieve the information.

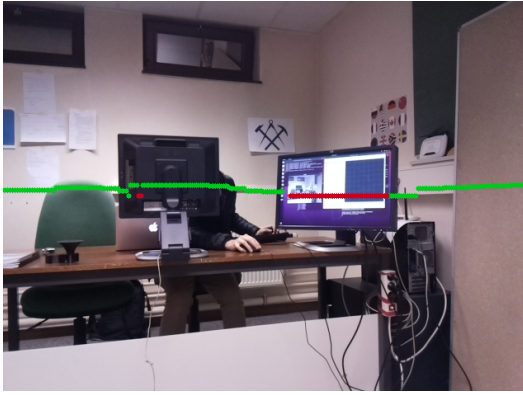


Figure 3.26: Testing : Back of the screen

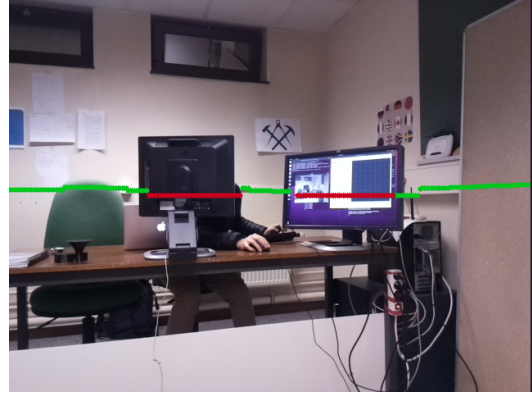


Figure 3.27: Testing : Back of the screen

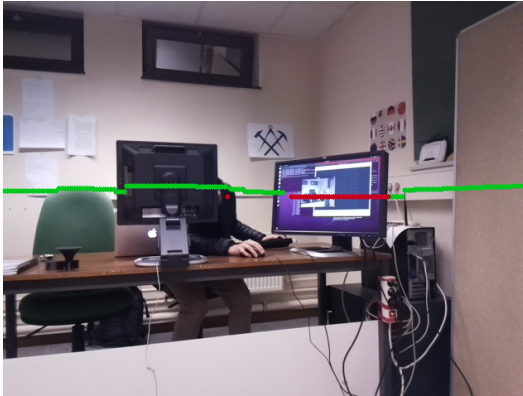


Figure 3.28: Testing : Bright display

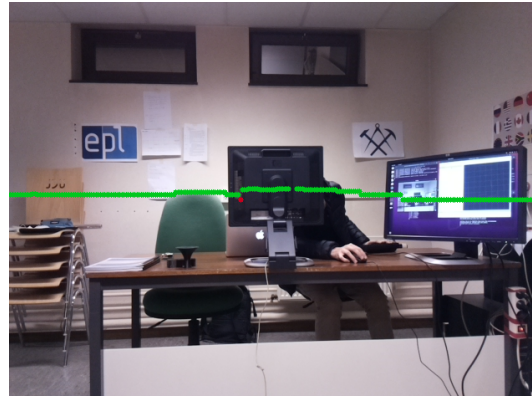


Figure 3.29: Testing : Bright display

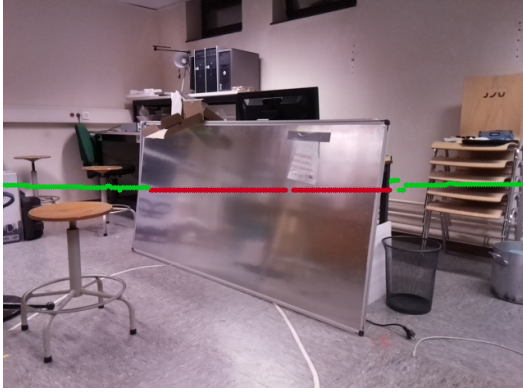


Figure 3.30: Back of the blackboard

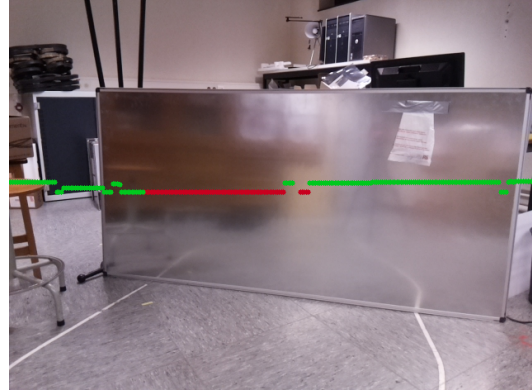


Figure 3.31: Back of the blackboard

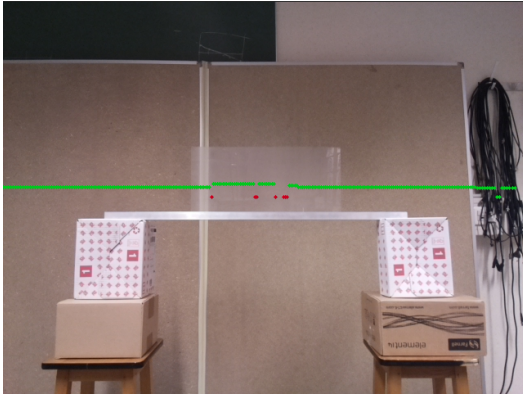


Figure 3.32: Testing :PlexiGlass

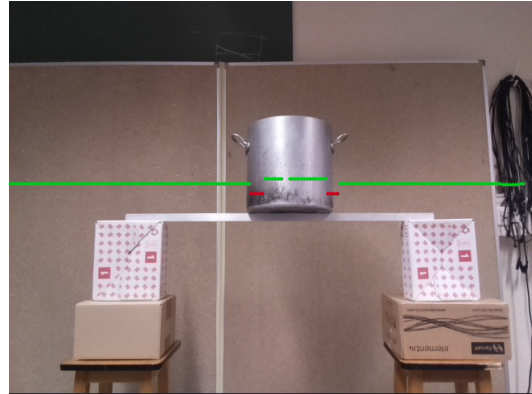


Figure 3.33: Testing : Cooking pot

#### Irregularities

Let us also note that sometimes, even though most of an object is correctly detected by the LiDAR, some parts of the object can be inconsistent and cause some issues, either for some challenging colors, textures or without any clear noticeable reason. (When printing data, it is not rare to see a few random "infinite value" appearing). In our implementation, when the LiDAR detected 'infinite value', those values are set to the previous coherent data. This is a simple way to eliminate outliers. However for longer sequences of 'infinite value', this method is not the best as it will repeat the same value for the length of the erroneous sequence. Another way to implement it would be to look for the next coherent value and apply an interpolation for the erroneous data between those two. Unfortunately, it would impact the response time of the system as the sampling and frame rate of the LiDAR are high and browsing a matrix several times each time a lidar scan is received, would take a lot of computational time.

## 4 Summary of the first part

The first part of this Master's thesis composed of the first two chapters was more research based. It encloses the summary of the preliminary study that was conducted upstream before ordering the necessary equipment (in particular the LiDAR sensor) and starting the implementation of the system downstream.

As this first part comes to an end, it seems appropriate to take the opportunity to review and summarize the key points that have been mentioned, as well as to restate the objectives in order to ensure the reader's understanding of the situation and relevance of the objectives of the next chapters.

To not lose track of the objectives, the goal is to investigate the possibilities, establish the premises and start a first implementation in order to realize in the long term a sensor+algorithm solution to allow drones to simultaneously localize itself and map unknown indoor environments.

But as we introduced more concepts as this point, we will give on this page an overview of the tasks tackled (and at the same time, a summary of the content of the following chapters).

As explained in the introduction, previous Master's thesis were based on the AR.Drone 2.0 and (mainly) on its hardware. To establish their SLAM algorithm, they implemented a key feature based method. In our approach, we do not work with the AR.Drone 2.0.

As we have seen in the State-of-the-Art section, there are different ways to implement SLAM algorithms. Unlike previous Master's thesis, we wanted to try another approach with direct SLAM methods. This type of slam takes different kind of inputs ; one of them being depth images, array with the typical information of a RGB camera but with a value for the depth.

We discussed the technology regarding sensors in chapters 2 and 3, and decided to implement a solution based on a 2D-scanning LiDAR and a monocular camera. After building our experiment setup (this part will be covered in Chapter 5), the first big objective will be to create the depth image mentioned above.

- To do so, we first need to merge the depth from the LiDAR and the visual information from the camera into one common reference framework. (this part will be covered in Chapter 6)

#### 4 Summary of the first part

- We will then need a way to approximate the depths of the points for which we do not have depth information from the LiDAR. With that in mind, we will discuss two pre-trained convolutional neural networks (CNN) which aim at estimating the depth using only the monocular camera : DenseDepth and AdaBins (this part will be covered in Chapter 7)
- To provide better depth images, we will then try to enhance the depth image received by the pre-trained CNN with the depth received by the LiDAR and discuss various ways to improve both the first estimation of the depth image and the correction of this depth image with the data provided by the LiDAR. (this part will be covered in Chapter 8)
- We will then discuss other upsides of using LiDAR to perform certain tasks (this part will be covered in Chapter 9) and explore together the various possibilities going forward (this part will be covered in Chapter 11)
- Finally, after having created depth images upstream, we need to address how to utilize those depth images. Therefore the last chapter will be dedicated to direct SLAM algorithm method based on depth images (this part will be covered in Chapter 10)

## 5 Setup description

In this chapter we cover the main physical and software aspects of our testing setup.

### 5.1 Physical setup

Knowing that the main goal was to integrate our setup on a mobile robot or a drone, we needed to design a platform that could be modular and portable. This section aims to describe the complete hardware and software aspect of our experimental setup.

#### 5.1.1 Hardware

Our setup is composed of three main hardware groups. The workstation, the mobile platform and the network router.

The workstation is a desktop computer running an Intel Core i7-3770 @ 3.4GHz x 8 coupled to 16 gigabyte of RAM. It uses Ubuntu 20.04 LTS with ROS Neotic.

The mobile platform is based around a Raspberry Pi 4 8GB of RAM running the quad core BCM2711 SoC from Broadcom clocked at 1.5 GHz and is connected via a wireless connection to the workstation. The Raspberry pi single board computer serves as a hub to connect all the necessary sensors to the ROS environment. It also uses a mobile version of Ubuntu 20.04 LTS especially compiled for ARM architecture.

We used multiple sensors for our implementation beginning with the RPLidar A3 from SLAMTEC for 2D scan acquisition. Coupled with the LiDAR, we used a relatively cheap camera module namely the Raspberry Pi Camera V2 capable of full HD video recording at 30 fps. The complete specification of the camera are the following:

| Camera Module v2  |                                             |
|-------------------|---------------------------------------------|
| Size              | Around $25 \times 24 \times 9$ mm           |
| Weight            | 3g                                          |
| Still resolution  | 8 Megapixels                                |
| Video modes       | 1080p30, 720p60 and $640 \times 480$ p60/90 |
| Sensor            | Sony IMX219                                 |
| Sensor resolution | $3280 \times 2464$ pixels                   |
| Pixel size        | $1.12 \mu\text{m} \times 1.12 \mu\text{m}$  |

The camera is mounted on a hinge and shares its location with an inertial measurement unit attached on its back. The chip is a MPU-6050 communicating via I2C with the



Raspberry Pi 4. The refresh rate is 50Hz and the full documentation is available in [66]. A semantic schematic of the mobile platform is available on figure 5.1.

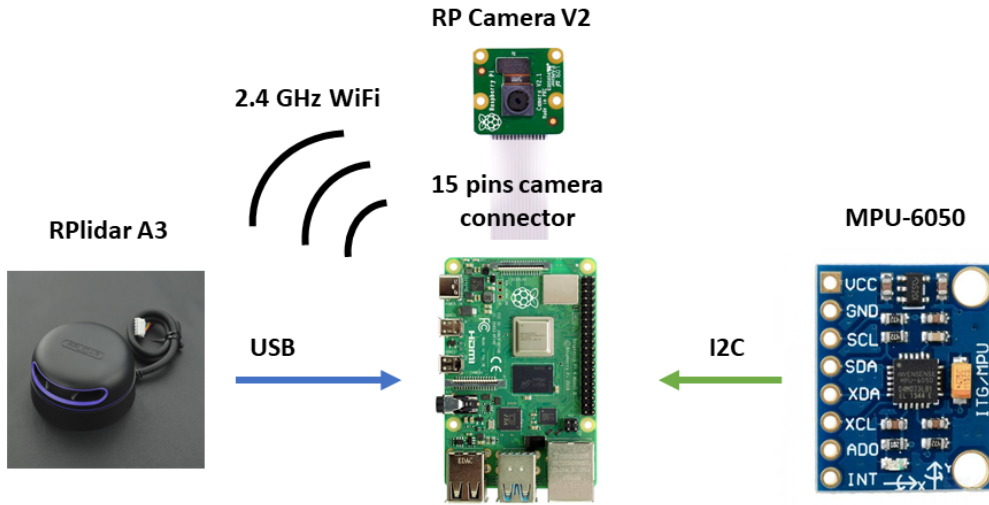


Figure 5.1: Semantic schematic of the mobile platform

### 5.1.2 Mechanical aspect

To accommodate on a easy and modular way all of our components, we decided to create our own mechanical setup. The mobile base was designed in such a way that all of the electronic components are easily accessible and can be dismounted without hassle. The model was designed in Solidworks and 3D printed in PLA. The complete assembled CAD can be found on figure 5.2b. The contraption is composed of one base plate and one top plate. The base plate is designed to accommodate the Raspberry Pi 4 and the top plate supports the RPLidar A3. The Raspberry Pi Camera V2 lies on a mobile plate mounted on a hinge, giving her one degree of freedom needed later for manual adjustment. The IMU board is screwed just behind the camera module giving him exactly the same orientation. The hinge has the goal to set the camera field of view as parallel to the LiDaR beam as possible. The complete assembly is then screwed on a tripod with a threaded metal piece. Giving the mobile platform 2 degrees of freedom namely the pitch and yaw.

## 5.2 Software

### 5.2.1 Framework Decision

As we had many sensors to our disposal we were looking for a clean and relatively easy to use programming environment to experiment with. Reading the previous work of students from UCL [35] and after an extensive research, we decided to use the ROS

## 5 Setup description

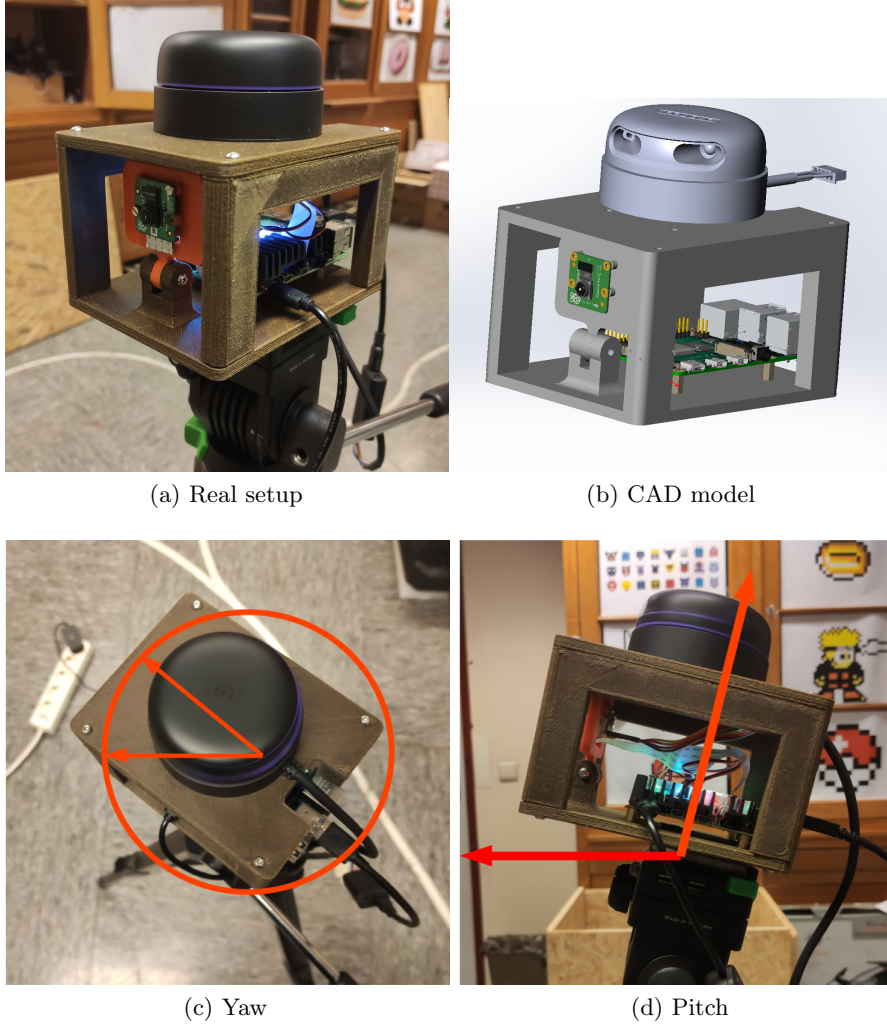


Figure 5.2: Setup

(Robot Operating System) framework. The reasons to favor ROS over another solution are multiple and stands as follow:

- ROS is an open source environment with a very active community. This is a key feature as many bugs and programming errors can be rapidly solved by browsing forums and the dedicated Wikipedia.
- It support multiple programming languages such as C++ and Python3.
- It is compiled for multiple CPU architectures such as X86 or ARM.
- The software architecture can be decentralized and run on multiple network connected machines.

## 5 Setup description

- Each piece of code can be run independently as a node and if a coding error occurs, only the concerned node would be affected. Giving the system a high level of robustness.
- A lot of nodes are already build and available to handle a variety of sensors. Those nodes are open source and the code can be easily reviewed if any modification is needed.

ROS is a powerful tool that is versatile, community driven and flexible. It is for all these reasons that we have decided to make ROS our main working tool throughout the duration of this end-of-study work.

### 5.2.2 ROS, working principle and test bench

In general, ROS consists of code and tools that help your project's code run and do the required job, including the infrastructure for running it, like messages passing between processes.

ROS is designed to be a loosely coupled system where a process is called a node and every node should be responsible for one task. Nodes communicate with each other using messages passing via logical channels called topics. Each node can send or get data from the other node using the publish/subscribe model. To synchronize all the messages between the nodes, ROS needs a master node running on a "host" machine called Roscore. An illustration of the working principle can be found on figure [5.3](#).

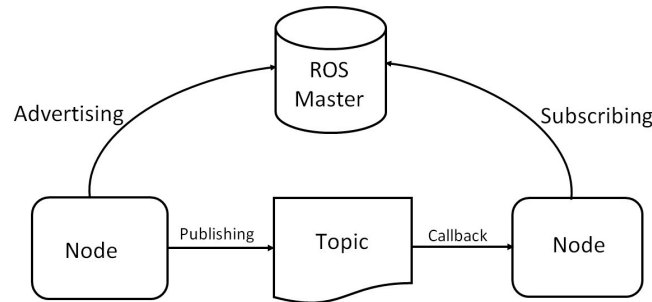


Figure 5.3: ROS working principle

#### Working principle

#### Test bench

For this master thesis, we had multiple objectives to fulfill beginning with experimenting with whether or not the Lidar-Camera combination would be a good contender for SLAM of autonomous robot. To be able to rapidly test the new software that was implemented, we decided to divide our ROS implementation as follows.

- **Mobile platform**, The mobile platform contains all the necessary nodes for sensor readings and is composed as following:

## 5 Setup description

- **Raspicam\_node** is the node dedicated to the readings of video stream from the Raspberry Pi Camera V2 and publishes tree types of messages. This node was made by Ubiquity Robotics and can be found here [\[76\]](#).  
"/image" which is an uncompressed video stream message easier to process but due to its uncompressed nature, can rapidly saturate the bandwidth of the wireless connection.  
"/image/compressed" is a compressed video stream message that takes less bandwidth but needs to be processed in order to be usable by another subscriber node.  
"/camera.info" is a message that contains the complete set of parameters used for correcting the distortion of the camera module.
  - **RPLidar\_ros** Node provided by [\[76\]](#) and is dedicated to the communication between the LiDaR and the Raspberry Pi 4. This node publishes only one message.  
"/scan" is the only message published by this node and contains the scan data of the LiDaR.
  - **MPU\_6050\_driver** Custom node created to gather the gyro and accelerometer data and pack them into one message of type Imu and is called "/imu/data\_raw"
  - **IMU\_complementary\_filter** Filter which fuses angular velocities, accelerations, and (optionally) magnetic readings from a generic IMU device into a quaternion. This filter publishes its data into a "/imu/data" message.
  - **Frame Transforms** This part of our ROS implementation serves a crucial role in making the published data migrate from one frame of existence to another. In fact, each sensor is related to its own frame of measurements and this may cause problems when making calculations with two different data sources. As those two streams were not sampled at the same place in 3D space, converting back the data to the frame of one sensor or another can cause many errors if not handled properly. ROS handles that in a very clever way and asks the user to define a frame for each sensor. When the user has to convert some data from one frame to another, a package called /tf can be used to retrieve the translation and rotation matrix necessary to the conversion to the desired data frame. A schematic of the complete transform tree can be found on figure [5.4](#).  
"tf\_broadcast\_imu" is the only tf package that takes an argument. When all tf have fixed values, this node broadcast a transform according to the orientation values of the IMU.
- **Workstation**, The most important module of ROS is what we call ROSCORE. This node has the objective of synchronizing all the messages running across the nodes and the system cannot work without it. We decided to run it on our

## 5 Setup description

workstation in order to shorten the path between our experimental nodes and the potentially published messages. As we had two distinct objectives, we decided to divide our setup into two different groups. One dedicated to the sensors message handling and one having the task to experiment with multiple SLAM algorithms.

- **Sensor test bench** This part of the workstation environment is dedicated to the handling of the messages sent by the sensors nodes. It receives the messages, couples them with the corresponding tf and after computation handles the result to the SLAM part of the setup.
- **SLAM test bench** The SLAM test bench is dedicated to the implementation of SLAM algorithms. It receives the modified messages from the Sensors test bench and handles them to feed them to the multiple SLAM algorithms implemented.

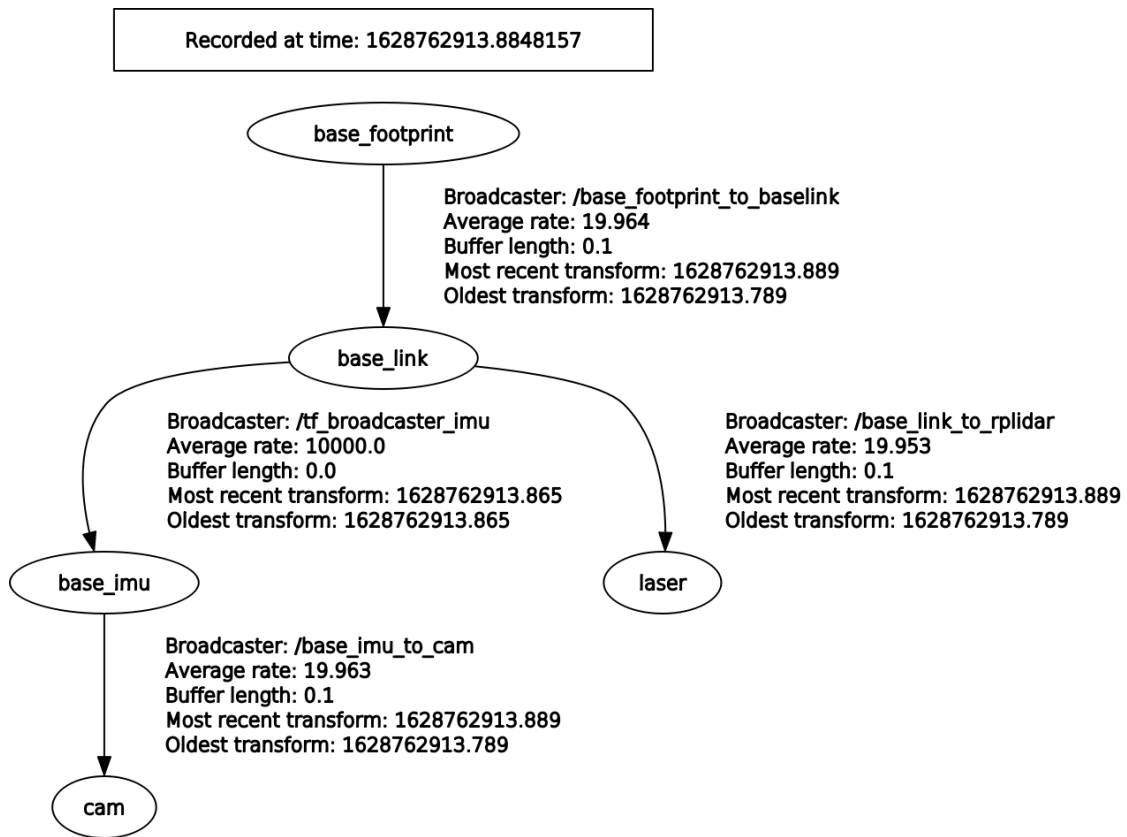


Figure 5.4: Transform tree of the system

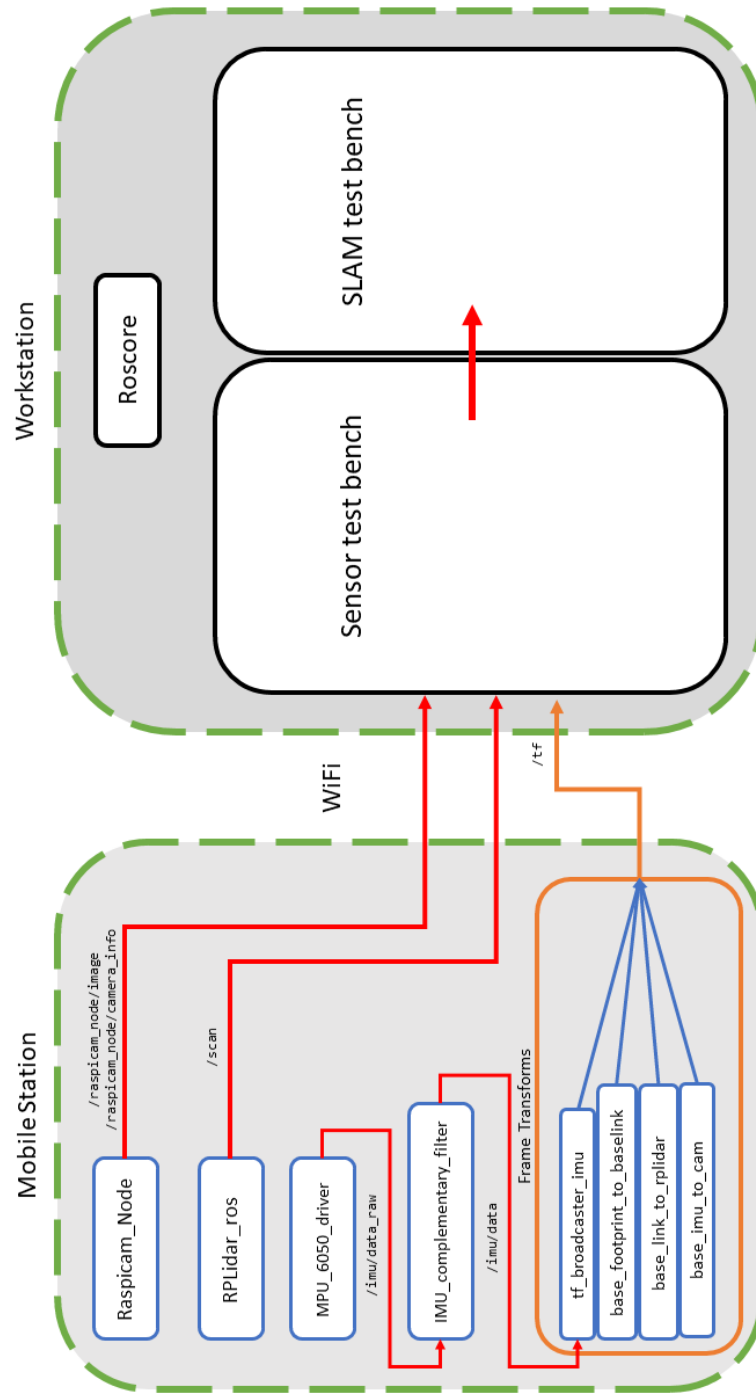


Figure 5.5: ROS test bench

## 6 Fusion of LiDAR and camera data

The first task is to correctly merge the Lidar data with the camera data. This section will first introduce the concept and explain the method from a theoretical point of view. Then, we will proceed with an explanation of the dedicated ROS node and, finally, some results will be provided.

### 6.1 Calibration of the camera

Camera calibration consists of estimating a function that will allow to convert the 3D coordinates in the 3D world into 2D coordinates in the 2D image plan of the camera. To do that, models are generally used. Calibrating the camera is determining the parameters of the chosen model for a given camera (depending on its specific characteristics) in order to establish the correspondence between information in the 3D world and their placements in 2D plan of the camera; in other words, the unknowns in the function that maps the 3D points to the 2D image plane.

This is a crucial step ensuring the pictures gathered from the camera are not distorted and respect the proportion of the real 3D world. To effectuate the calibration process we use pre-made functions from the OpenCV library, more precisely `calib3d` [1]. In almost every previous master's theses, they use these same tools and, also almost every year, the preceding master's theses repeatedly cover the subject with details. We will not abrogate to the rule as the pinhole camera model used for the calibration is closely intertwined with the matching of the LiDAR data points with the camera information.

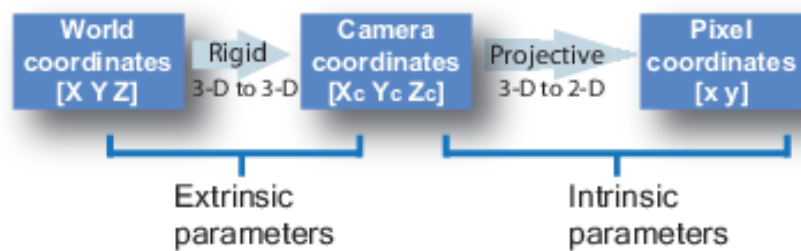


Figure 6.1: Remapping from the 3D points in the real world to the 2D image plan of the camera [63]

As illustrated in Figure 6.1, the process can be divided in two steps :

- Determining the extrinsic parameters of the model which maps a point expressed in the world coordinates ( $P_W = X_W, Y_W, Z_W$ ) to the camera coordinates ( $P_C = X_C, Y_C, Z_C$ ).
- Determining the intrinsic parameters of the model which maps the point expressed in the camera coordinates ( $P_C = X_C, Y_C, Z_C$ ) to the corresponding pixel in the 2D image plan ( $u, v$ ).

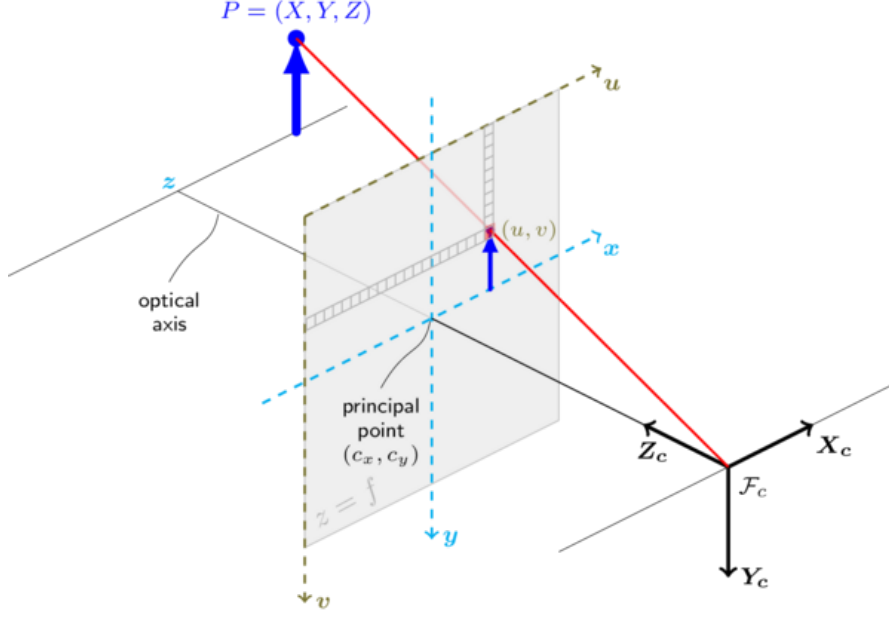


Figure 6.2: Schematic of the pinhole camera model [1]

### Extrinsic parameters

$$P_{C,i} = \begin{bmatrix} X_{C,i} \\ Y_{C,i} \\ Z_{C,i} \end{bmatrix} = R \cdot P_{W,i} + t \quad (6.1)$$

Where  $R \in \mathbb{R}^{3 \times 3}$  is the rotation matrix between the two coordinate systems and  $t \in \mathbb{R}^{3 \times 1}$  is the origin of the reference frame expressed in the camera coordinate system. Equation 6.1 using homogeneous coordinates can be rewritten as follows :

$$P_{C,i} = \begin{bmatrix} X_{C,i} \\ Y_{C,i} \\ Z_{C,i} \end{bmatrix} = [R \mid t] \begin{bmatrix} X_{P,i} \\ Y_{P,i} \\ Z_{P,i} \\ 1 \end{bmatrix} + t \quad (6.2)$$



### Intrinsic parameters

$$z_C \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} \quad (6.3)$$

Where  $(u,v)$  are the coordinates of the projection point in pixels,  $f_x$  and  $f_y$  are the horizontal and vertical focal lengths respectively,  $c_x$  and  $c_y$  are the coordinates of the principal point (usually the center) of the image.

### Global equation

Gathering everything together, we can write the global equation of the pinhole model as expressed in Equation 6.4 mapping the points of the real world into the image plane by projecting 3D points using a perspective transformation.

$$z_C \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} [R \mid t] \begin{bmatrix} X_{P,i} \\ Y_{P,i} \\ Z_{P,i} \\ 1 \end{bmatrix} + t \quad (6.4)$$

The parameter  $z_C$  is usually call the scaling factor as everything is known up to this factor which represents the depth and the dimension that is lost during the remapping process.

### Lens Distortion

The pinhole model is really good and simple but, as every model, not perfect either. More complex models building on this initial model are numerous, with various level of accuracy and complexity. In the OpenCV library `calib3d` mentioned above [1], the lens distortion model is used. Indeed real lenses usually have some distortion, mostly radial distortion and slight tangential distortion. In this model are considered two types of distortion: radial and tangential distortion. Those distortions can be modeled using the Brown-Conrady distortion model (using a third order approximation for the radial distortion, and a second order approximation for the tangential distortion):

$$x_d = x_u(1 + k_1r^2 + k_2r^4 + k_3r^6) + 2p_1x_u y_u + p_2(r^2 + 2x_u) \quad (6.5)$$

$$y_d = y_u(1 + k_1r^2 + k_2r^4 + k_3r^6) + 2p_1x_u y_u + p_1(r^2 + 2y_u) \quad (6.6)$$

Where  $k_i$  and  $p_i$  are the radial and tangential distortion coefficients respectively,  $(x_d, y_d)$  is the distorted image point (where it is actually observed),  $(x_u, y_u)$  is the undistorted image point (where it would be observed if there was no distortion), and  $r$  is the distance between the undistorted point, and the image center  $(x_c, y_c)$ . The equations 6.5 and 6.6 go up to the 3<sup>rd</sup> order but in the OpenCV library `calib3d`, the model is the same but

goes up to the 6th order coefficients. By determining those coefficients  $k_i$  and  $p_i$ , we can get undistorted pixel coordinates.

Note that this section about camera calibration is a summary of the concept behind the used functions from the OpenCV library `calib3d` [1]. Moreover if the reader is eager to see more detailed information on the topic, we recommend to check previous master theses (particularly the master's theses written by L. Kechtban (2018) and B. Dehem (2017)).

## 6.2 Transposing Lidar information to the camera reference frame

Remapping the information received from the LiDAR to the corresponding pixels in the 2D image plan of the camera is a process very similar to the calibration process. In this section, we will introduce the concept from a theoretical point of view and then proceed with some visual demonstration of the proper functioning of this section.

### 6.2.1 Theoretical approach

To match the data perceived by the LiDAR and the camera, we need to establish the function mapping one point in LiDAR coordinate frame to the corresponding point in the camera coordinate frame and then to the corresponding pixel. The first transformation, from one reference frame to the other, can be formulated by Equation 6.7 in which  $P_C$  is a 3D point expressed in the camera coordinate system ( $P_C = X_C, Y_C, Z_C$ ) and  $P_L$  a 3D point expressed in the LiDAR coordinate system ( $P_L = X_L, Y_L, Z_L$ ), and finally  $R$  and  $t$ , respectively the rotation and the translation between the two sensors.

$$P_C = R.P_L + t \quad (6.7)$$

Let us start with the point expressed in the LiDAR coordinate system  $P_L = X_L, Y_L, Z_L$ . As depicted in Figure 8.2, this point can be expressed as follows :

$$P_{L,i} = \begin{bmatrix} X_{L,i} \\ Y_{L,i} \\ Z_{L,i} \end{bmatrix} = \begin{bmatrix} d_i \sin \theta_i \\ 0 \\ d_i \cos \theta_i \end{bmatrix} \quad (6.8)$$

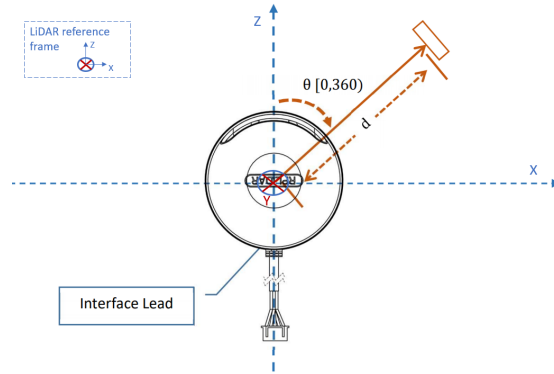


Figure 6.3: LiDAR reference frame

To transpose this point from the LiDAR reference frame to the point expressed in the camera frame, we need to apply a rotation and a translation if the frames do not coincide. Equation 6.10 is the matrix of rotation from the LiDAR frame to the camera frame. In our implemented code, we only use a rotation around the Y axis ( $R_Y(\omega_y)$ ). Indeed, we observed a slight horizontal shift between the LiDAR and camera center, which we adjusted for by trial and error. It is likely that the z and x axes are also not perfectly aligned but if so, the shift seems so unnoticeable that it can be negligible.

$$R = R_Z(\omega_z)R_Y(\omega_y)R_X(\omega_x) \quad (6.9)$$

$$= \begin{bmatrix} \cos\omega_z & -\sin(\omega_z) & 0 \\ \sin\omega_z & \cos(\omega_z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\omega_y) & 0 & \sin(\omega_y) \\ 0 & 1 & 0 \\ -\sin(\omega_y) & 0 & \cos(\omega_y) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\omega_x & -\sin(\omega_x) \\ 0 & \sin\omega_x & \cos(\omega_x) \end{bmatrix} \quad (6.10)$$

Equation 6.11 is the matrix of rotation from the LiDAR frame to the camera frame in which h is the the distance along the Y axis (the difference of height) between the camera and the LiDAR and b the distance along the Z axis. (The distance along the X axis is negligible in our system).

$$t = \begin{bmatrix} 0 \\ h \\ b \end{bmatrix} \quad (6.11)$$

Putting everything together, we get the equation 6.12 :

$$P_{C,i} = \begin{bmatrix} X_{C,i} \\ Y_{C,i} \\ Z_{C,i} \end{bmatrix} = R_Z(\omega_z)R_Y(\omega_y)R_X(\omega_x) \begin{bmatrix} d_i \sin \theta_i \\ 0 \\ d_i \cos \theta_i \end{bmatrix} + \begin{bmatrix} 0 \\ h \\ b \end{bmatrix} \quad (6.12)$$

Note that in our system, we implemented this model and fine-tuned, by trial and error, some of the parameters. For the translation and rotation, we could also use Frame Transforms introduced in Chapter 5. It is particularly interesting if the camera has to move independently from the LiDAR. In that case, with data from the IMU, the system can automatically change the different values of the rotation and translation matrices. Although both methods were correctly implemented, we mainly use the first implementation without the Frame Transforms as the system seems to give more accurate matching and the Frame Transform implementation would also benefit from some by trial and error tuning. Moreover, moving in flight the camera independently from the LiDAR does not seem to be a desired objective. (But the option is there, if one wishes : uncomment the concerned lines in the ROS node and comment the equivalent lines of the other method).

Now that we have the LiDAR ranges expressed in the camera reference frame, we can map those points to the corresponding pixels. This step is very similar to the calibration of the camera, as the process and objective is the same : mapping a 3D located point

from the camera center to the corresponding pixels in the 2D image plan. Indeed we can use the equations 6.13 and 6.14, taken from the pinhole model, where  $\alpha_x$  and  $\alpha_y$  are the focal lengths in pixels,  $\mu_0$  and  $\nu_0$  are the coordinates of image center in pixels and  $s$  is the skew parameter.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \alpha_x & s & \mu_0 \\ 0 & \alpha_y & \nu_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} \quad (6.13)$$

As the camera is calibrated and we use fonctions upstream to undistord images (6.1), the image plan in which we place those points are also already undistorted. Therefore we do not need to take into account other deformation such as lens distortion or even the skew factor (featured in Equation 6.13 but set to zero) as they have already been taken care of in preceding nodes.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x/z \\ y/z \end{bmatrix} \quad (6.14)$$

### 6.2.2 Implementation and results

Based on the theory explained above, a dedicated ROS, `rp_cyclop`, was written to transpose the LiDAR data into the image and add some visuals for the demonstration.

This ROS node takes as input the image from the Raspberry Pi camera sent by the `Raspicam_Node` and the array of the distances measured by the LiDAR sent by the `RPLidar_ros` node. The node "corrects" the LiDAR's data as discussed in the end of chapter 3. Secondly, applying the concepts described above, the node converts the data from the LiDAR expressed in the LiDAR reference frame to the 2D image plan of the camera, associating the calculated depth to the corresponding pixels.

Moreover, for demonstration, the node can be set to return the received image with the pixels with depth information from the LiDAR highlighted in colors, as depicted in the Figure 3.16a in chapter 3. Colors can be adjusted to highlight points for which the depth received by the LiDAR is corrupted/non-conclusive (and so set to infinite by the `RPLidar_ros`) like in the section 3.4.3 regarding limit testing. Colors can also be tuned to emphasize the difference in depth perceived by the LiDAR, the color gradually changing from blue to red as the detected object gets closer. This is in fact the method used in the different demonstration Figures of this section.

Indeed, to visually show the correct transposition of the Lidar depth to the corresponding pixels in the camera image plan, we assembled a small experimental setup as depicted in Figure 6.4. The objective of the demonstration is to put different object with various shape and depth and visually show that the LiDAR corresponds to the detected

objects. To do so, we have scanned the setup while tilting up and down the system composed of the different sensors and the Raspberry Pi and display the detected depth by the LiDAR at various heights. Two videos demonstrating the proper functioning of the node are accessible on the following link : [GoogleDrive \(link\)](#) or [Github \(link\)](#). In addition, Figures [6.5](#) illustrates the results for different tilt angles of the system.

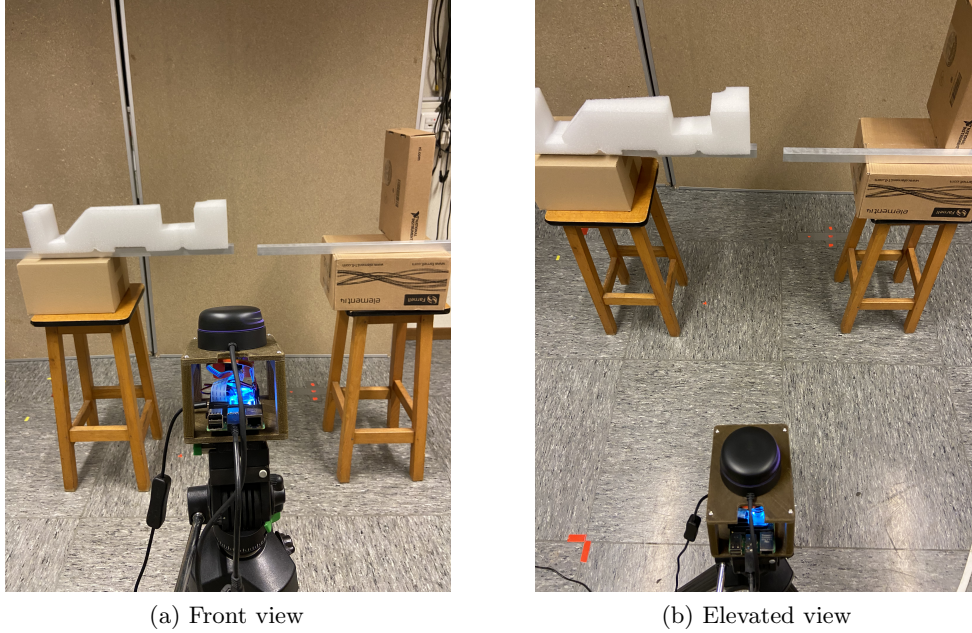
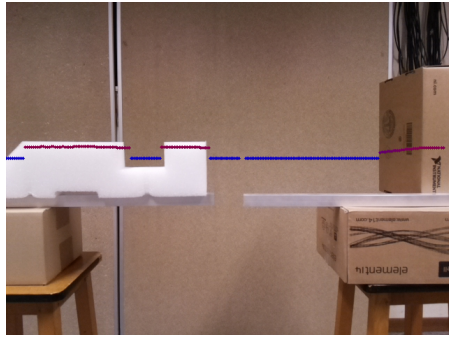


Figure 6.4: Demonstration setup

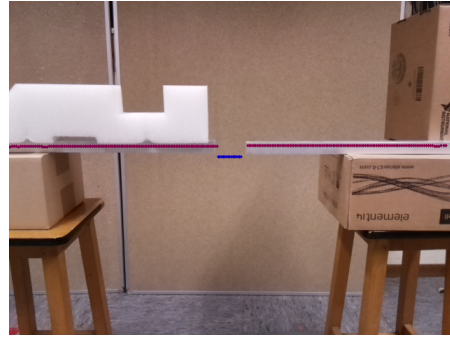
In terms of performance, as one can see, the matching of data is very accurate, especially for the y coordinates (the height of the image) as particularly noticeable in Figure [6.5b](#). The correspondence for the x coordinate (the width of the image) can sometimes be lightly shifted. By example, for the demonstration set-up a 1cm horizontal displacement of the data can sometimes be observed for a object placed at 0.9 m away from the LiDAR. The ratio of the horizontal mismatch over the distance is therefore slightly more than 1% which seems negligible in the context of SLAM implementation of UAVs.

### 6.3 Upadte on the progression

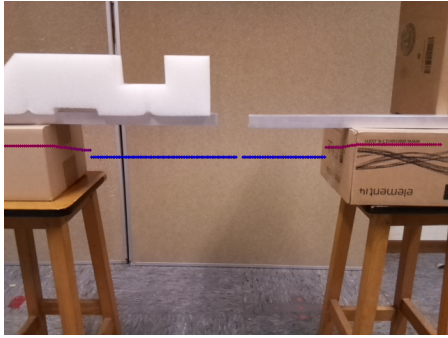
Maybe one can ask himself why transposing data from the LiDAR reference frame to the camera reference frame. What we want to do is create depth images which contain the same information of a regular image captured by the camera but with one additional element ; the depth of each pixel. Indeed those depth images are particularly useful as they can be "fed" to a SLAM algorithm (well-known open source ones or homemade) to estimate the pose of the UAV and and to create a visual map of the environment



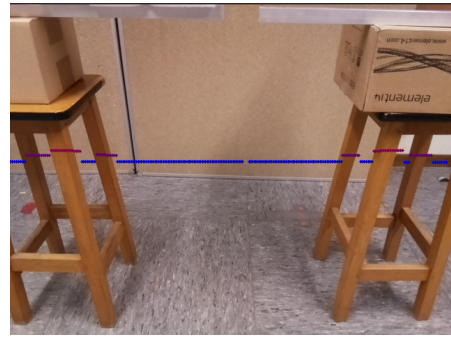
(a) First scan



(b) Second scan



(c) Third scan



(d) Fourth scan

Figure 6.5: General setup: Projections of Lidar data into camera image  
(Results of the demonstration of LiDAR data into the camera frame)

(more on that in chapter [10](#)). Now that we have the image captured by the camera and the depth sensed by the LiDAR in the respected format, we can use those to create our depth image. The next challenge is then to assign coherent depth values of all the other pixels of the image for which we don't have any information yet.



## 6 Fusion of LiDAR and camera data



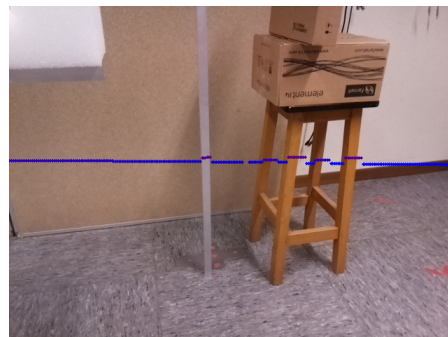
(a) Setup aiming on left stool



(b) Setup aiming on right stool



(c) Result for left tool



(d) Result for left tool

Figure 6.6: Stools setup: Projections of Lidar data into camera image  
(Results of the demonstration of LiDAR data into the camera frame)

## 7 Estimated depth from Convolutional Neural Networks

Looking at the 2D characteristic of the LiDAR's scans and knowing that a drone would evolve in a 3D environment, we knew that the measurement's dimension were not enough for indoor mapping. We wanted a way of getting a 3D measurements of 2 2D sensors (a camera and a LiDAR for example). The goal was to create a 3D pointcloud or a depth map from the fusion of the scans and the video stream. In a previous section, we showed how we made a correspondence between the scans and the camera's pixels (a example of the situation can be seen on figure 8.1). A first step to make the image data usable for a possible fusion with the LiDAR would be to derive a first guess of what kind of depth we could expect in each images from the video stream. In this section we describe how we managed to make a semantic segmentation of the image based on monocular depth estimation with the use of convolutional networks. We will give an overview of the two most promising types of networks used along side with a performance comparison.

Two different convolutional neural networks (CNN) where tested from the same author namely Ibraheem Alhashim et al. in [5] and [9]. The complete LiDAR-Camera correction algorithms will be explained in the next section of this paper.

### 7.1 DenseDepth, Encoder-decoder architecture

#### 7.1.1 Working principle

This network is what the author calls a transferred learned architecture. It is divided into two main components, the encoder and the decoder as illustrated in 7.1. The encoder is based on a pre-trained network called DenseNet-169. This network serves as a feature maps creator outputting a [15, 20, 1664] tensor from a [480, 640, 3] BGR image and has been trained on the ImageNet database [50]. The author has then truncated the network by removing the classification layers linked to the ImageNet database and added its own network. The custom decoder network is composed of 4 up-sampling blocks them-self composed of 4 different types of layers. The block structure can be seen in table 7.1

Each block is composed of a  $2 \times$  bilinear upsampling followed by two  $3 \times 3$  convolutional layers with output filters set to half the number of inputs filters, and were the first convolutional layer of the two is applied on the concatenation of the output of the previous layer and the pooling layer from the encoder having the same spatial dimension. Each upsampling block, except for the last one, is followed by a leaky ReLU activation function with parameter  $\alpha = 0.2$ .

The output of this second block of the network gives an output tensor of shape



## 7 Estimated depth from Convolutional Neural Networks

| Layer    | Function          |
|----------|-------------------|
| UP       | Upsample 2x2      |
| CONCAT   | Concatenate POOL3 |
| UP-CONVA | Convolution 3x3   |
| UP-CONVB | Convolution 3x3   |

Table 7.1: Upsampling block

$[240, 320, 1]$  which is clipped to the range of  $[0.4, 10]$  which correspond to the depth of the scene in meters.

The whole network has been trained on the NYU Depth V2 indoor data set and the author provides the trained weights for the user to experiment with on its own implementation of the code.

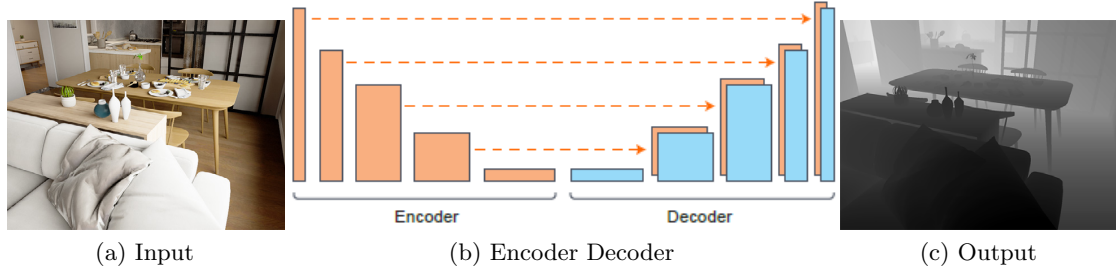


Figure 7.1: General schematic of the Network 5

### 7.1.2 Implementation

After some researches, we found an ROS implementation of the network which can be viewed in [\[88\]](#). The node uses Python2 and as we were using Python3, we had to convert the code and make some minor changes without any modifications of the core components. The resulting node is "monodepth" and subscribes to the "/raspicam\_node/image" and the "/raspicam\_node/camera.info" topic coming from the "raspicam\_node". It computes for each received frames a depth map and a pointcloud that are republished immediately as "/monodepth/pointcloud" and "/monodepth/image\_depth".

## 7.2 Adabin, depth estimation using adaptive bins

### 7.2.1 Working principle

This network is based on the same "Encoder-Decoder" architecture but replaces the previous encoder DenseNet-169 by a more lightweight and pretrained one called "EfficientNet-B5". The output of the decoder is no more a tensor of  $[240, 320, 1]$  but a  $[240, 320, C]$  with  $C$  representing the features. This tensor is then fed into what they call

a mViT module based on Transformer networks. This module dynamically computes a set of subdivisions "b" of the maximum range ( $d_{min}$  and  $d_{max}$ ) allowed by user. Those subdivisions are called bins and their width varies from one frame to another depending on the scene. The module computes also a set of convolutional kernels "R" called "Range Attention Maps". R and b are then used together in a "Hybrid Regression" module to recompute the final depth map by linear combination of the bins. This linear combination " $c(b_i)$ " can be seen in equation [7.1](#).

$$c(b_i) = d_{min} + (d_{max} - d_{min}) \left( \frac{b_i}{c} + \sum_{j=1}^{i-j} b_j \right) \quad (7.1)$$

Where i is the bin number given by the mViT module. And at each pixel the depth is retrieved with the following equation:

$$\tilde{d} = \sum_{k=1}^N c(b_k) p_k \quad (7.2)$$

With  $p_k$  the kth softmax score at each pixel interpreted as a probability over the N depth-bin-centers  $c(b_i)$ . The complete block diagram of the network can be found on figure [7.2](#).

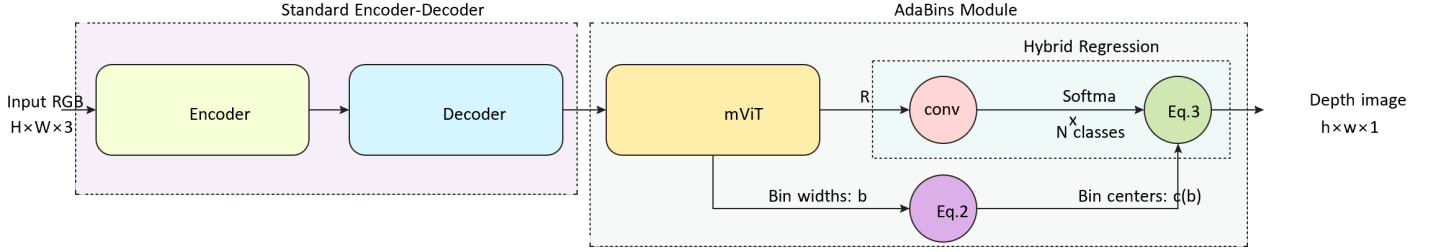


Figure 7.2: AdaBind block diagram

### 7.2.2 Implementation and results

As this network is relatively new (2020), no ROS implementations of the code was available and we, consequently, created our own custom node. The code is available on our GitHub and is called "monodepth\_adabin" [\[64\]](#). This node takes the same inputs as "monodepth" in addition to "/scan". The node republish a 480X640 depth map in meters at the same time as the received camera.info and scan under the topic's name "/monodepth\_adabin/depth" "/monodepth\_adabin/camera.info" and "/monodepth\_adabin/scan". The goal here is to publish the three messages at the same time to emulate the implementation of an RGB-D camera node. Having synced messages is often needed by various SLAM algorithms or nodes that require the data from various sensors. The corresponding flowchart can be found on figure [7.3](#)

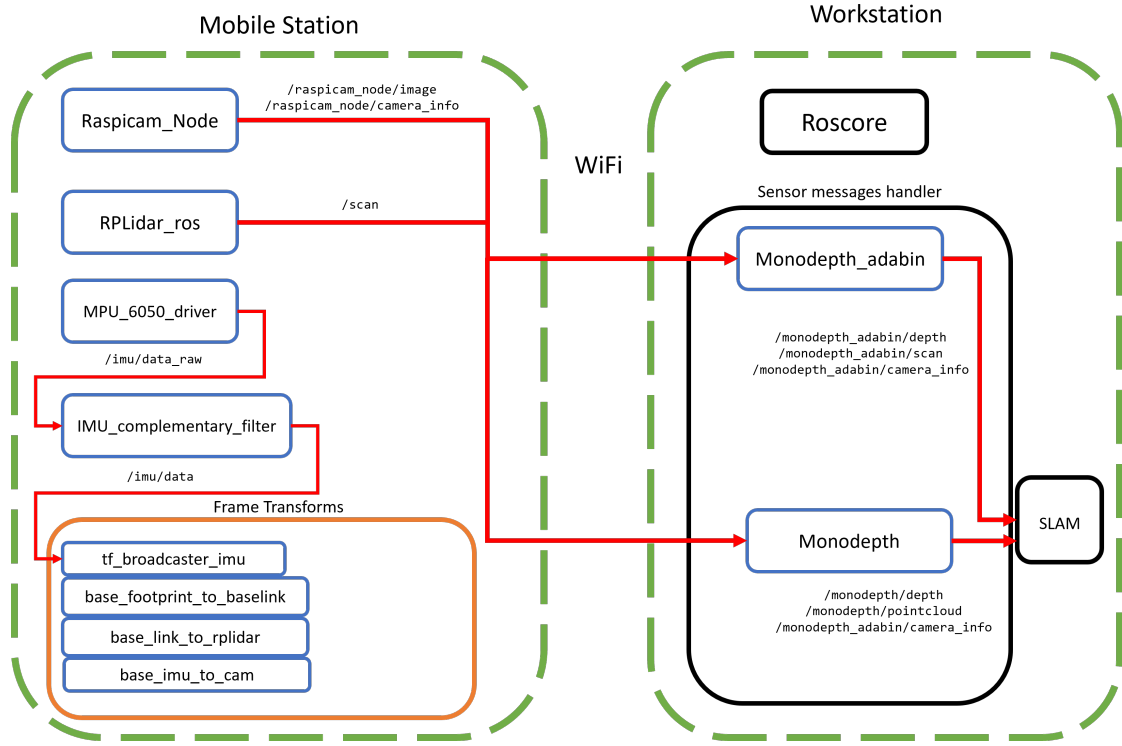


Figure 7.3: FFlow chart of the ROS environment with Monodepth and Monodepth<sub>adabin</sub> nodes implemented

### 7.3 Comparison and best candidate

To make the comparison between the two algorithms, we took into consideration two main criteria, execution speed and spatial segmentation. Two representative scenes were fed to the neural networks and the results are available on the following figures. As we did not have the ground truth available for making a complete and accurate comparison, we measured the 4 corners of each carton boxes in addition to the 4 corners of the image itself. We then compared those measurements to the one calculated by the networks in order to find the one that makes the best separation between the background and the subject (carton boxes).

Each code was run on a Ryzen 7 1800x @ 3.6 GHz coupled to a nvidia GTX1080 with 32GB of ram. This setup was used in order to compare the pure performance of the networks and not be limited by any hardware bottleneck.

Looking at figure 7.4 we can see that AdaBins does a much better job of separating the foreground from the background and it becomes more obvious on figure 7.5. In addition to the qualitative performance, AdaBins runs in 0.23 sec and DenseDepth runs in 1.07 sec per images. With all those factors taken into account, AdaBins will, de facto,

be favored.

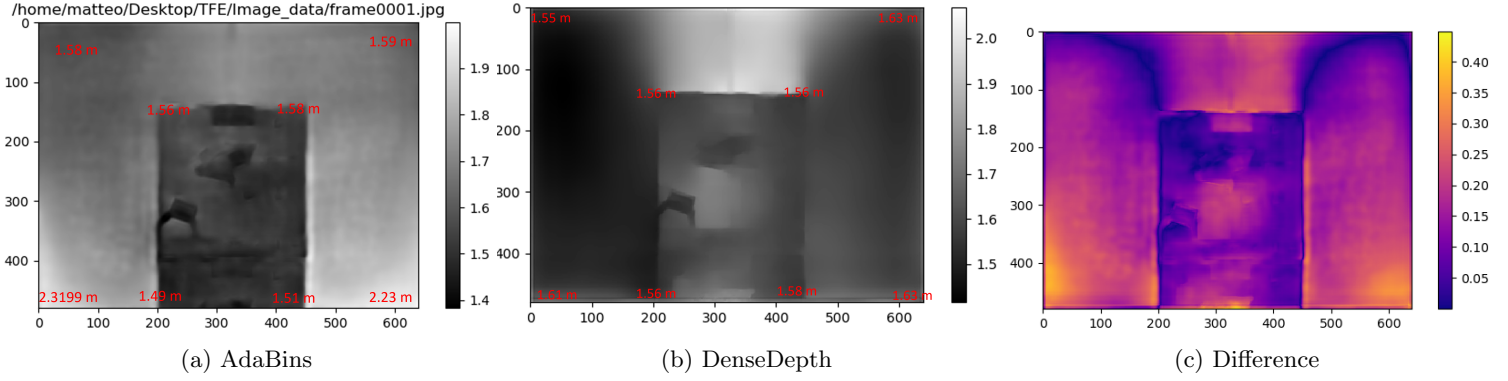


Figure 7.4: Example scene where AdaBins is superior. The four corners of the box are originally at 1.26m from the camera and the four corners of the wall are measured to be at 1.99m

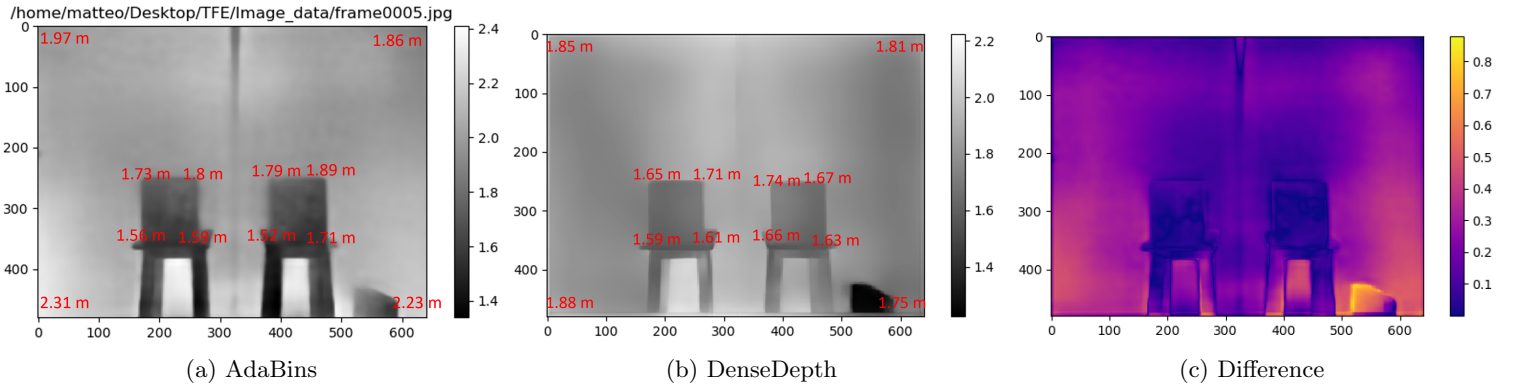


Figure 7.5: Example scene where AdaBins and DenseDepth are comparable. Each box sits at 1.27m from the camera while the wall's corners are measured at 1.99m

### 7.3.1 Performance as a ROS node

We then implemented the networks in ROS as described in figure 7.3 and measured the real world time intervals. The results were not expected as seen on 7.2. Many factors can explain such poor performance starting with the resources that the ROS environment allows to each node. Or for example the fact that the networks work best on GPUs than CPUs only. The workstation from the lab only has one CPU which may be the cause of the degraded performance regarding the original measurements. Those results will have an impact on the working behavior of the studied SLAM algorithm.

| Vanilla setup |              |
|---------------|--------------|
| DenseDepth    | 1.07 [s/img] |
| AdaBins       | 0.23 [s/img] |
| ROS setup     |              |
| DenseDepth    | 18.7 [s/img] |
| AdaBins       | 15.4 [s/img] |

Table 7.2: Summary of the timings, averaged on 20 frames

## 8 Depth Image Correction

Our original subgoal was to create exploitable depth images to later be used by State-of-the-Art (or homemade versions) of SLAM algorithms. In chapter 6, we have established the fusion of the depth information sensed by the LiDAR with the visual information from the monocular camera. Thus we have a rather accurate value of the depth of the pixels corresponding to the points scanned by the LiDAR. However, we do not have any information on the depth of the other pixels of the image (as illustrated in Figure 8.1). In Chapter 7, we have discussed some algorithms taking as only input, the images received from the monocular camera and, passing this information through a trained convolutional neural network, can give estimation for the depth. This algorithm allocates a depth value for each pixel, returning a depth image. However the accuracy of those algorithms are limited and the estimated depth can sometimes be very different from the truth depth of the real world.

In this chapter, the goal is to enhance the depth image created by AdaBins (one of the algorithm introduced in chapter 7) by taking into consideration the depth perceived by the LiDAR.

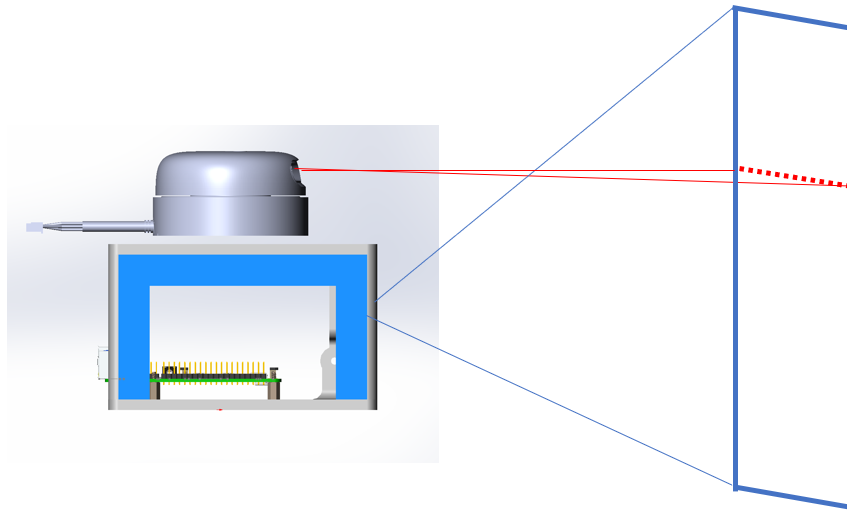


Figure 8.1: Schematic of the LiDAR and camera information localisation

## 8.1 Correction process

Building on the code needed for Chapter 6, we wrote a ROS node whose purpose is to adjust the depth image received from the AdaBin ROS node by taking into account the depth perceived by the LiDAR. More precisely, this node called *Depth\_correction* receives messages from the Monodepth\_adabin with the estimated depth of the image and the scan of the LiDAR containing the information outlined in Chapter 3. The node returns different messages with, among them, a corrected depth image and the corresponding 3D point cloud which can be later used by SLAM algorithms.

The related code can be found on our Github ([Github \(link\)](#)). In this section we will explain the logic behind it.

To summarize what the code does, we need to introduce a few notation.

Let  $u_i$  and  $u_{(i-\Delta u)}$  be the horizontal coordinates in pixel of two consecutive points scanned by the LiDAR (and therefore for which we have accurate depth information if LiDAR measurements does not encounter problematic objects/surfaces as mentioned in chapter 3). Let also note the corresponding vertical pixel coordinates as  $v_i$  and  $v_{i-\Delta v}$ .

Let  $D_L[x, y]$  and  $D_A[x, y]$  be , respectively, the depth perceived by the LiDAR and the depth estimated by AdaBins at the pixel coordinates  $(x, y)$ .

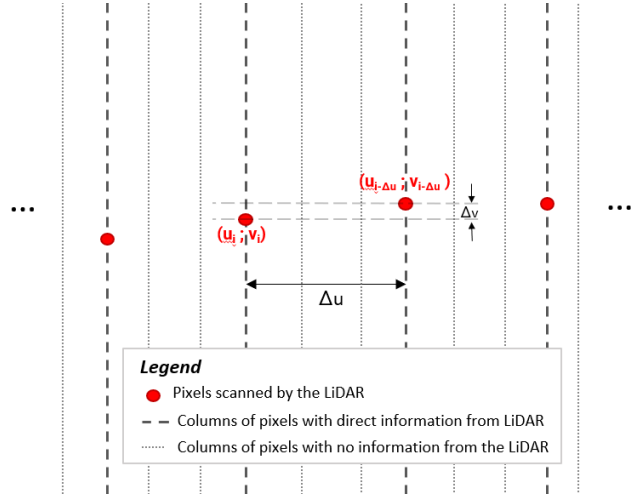


Figure 8.2: LiDAR reference frame

With those notations, we can write Equations [8.1](#) and [8.2](#)

$$\Delta u = (u_i - u_{(i-\Delta u)}) \quad (8.1)$$

$$\Delta d = (D_L[u_i, v_i] - D_L[u_{(i-\Delta u)}, v_{(i-\Delta v)}]) \quad (8.2)$$

In Equation [8.1](#),  $\Delta u$  expresses the horizontal distance in pixel between two consecutive points scanned by the LiDAR ; in other words, it expresses the horizontal pixel spacing between the Lidar's data (which can vary).

In Equation [8.2](#),  $\Delta d$  is the difference of depth between those two points ; in other words, the depth step between two consecutive LiDAR measurements.

Let  $D_C[x, y]$  be the corrected estimated depth at the pixel coordinates  $(x, y)$ . Also  $x$  is the increment of horizontal pixel, an integer such that  $\{x \in \mathbb{N} \mid 0 \leq x \leq \Delta u\}$  and  $h$

## 8 Depth Image Correction

is the increment of vertical pixel, an integer such that  $\{h \in \mathbb{N} \mid 0 \leq h \leq H\}$  with  $H$  the notation for the maximum vertical pixel, in other words, the height of image expressed in pixel. With those notations, the corrected depth can be written as :

$$D_C[u_{(i-\Delta u)} + x, h] = (D_A[u_{(i-\Delta u)} + x, h] + \text{correction}) \quad (8.3)$$

We thought of different ways to apply such correction. We will go over the implementation process, mentioning different approaches.

As a reminder, we only have LiDAR information for some specific points, one every few column of pixels. The first thing we do is to extrapolate those values to have one good estimate of the depth for one point on each column of pixel. Then we calculate the difference of depth between the depth perceived by the Lidar (for the horizontal pixel coordinates for which we have direct information from the LiDAR) or the extrapolated depth based on the depth perceived by the Lidar (for the horizontal pixel coordinates that we have no direct information of the LiDAR) and the depth estimated by AdaBins at those points.

In the first method, we apply this difference to each pixel of the column. Naturally, the depth estimated at the height of the LiDAR can be totally unrelated to other pixels of the same column (pixels located on the same horizontal coordinate but different vertical coordinate). Therefore our first implementation applied a scaling coefficient to the correction term, so the further the pixel is located from the height of the Lidar data point, the lesser is the influence of the correction on this given pixel. Such implementation can be written as described in Equation 8.4 - 8.5 :

$$\text{Depth error} = D_L[u_{(i-\Delta u)} + x, \frac{v_i + v_{(i-\Delta u)}}{2}] - D_A[u_{(i-\Delta u)} + x, \frac{v_i + v_{(i-\Delta u)}}{2}] \quad (8.4)$$

$$D_C[u_{(i-\Delta u)} + x, h] = D_A[u_{(i-\Delta u)} + x, h] + \text{Depth error} \left( \frac{H - \left| \frac{v_i + v_{(i-\Delta u)}}{2} - h \right|}{H} \right) \quad (8.5)$$

But even with this coefficient taking the vertical distance into account, the correction is rather poor. We could also limit the correction to only a window of pixels above and below the scanned point by the LiDAR for which we consider that the correction has a good probability to be relevant. It would still be applying the correction arbitrary to the pixels only taking into consideration their location on the image but with zero consideration for what that pixel might represent. All in all, this first method, applying the correction on each pixel with a decreasing ratio the further away the pixel is, is quite limited. Indeed, it takes no consideration of the elements composing the environment, the depth even five pixels below can be very different from the depth scanned by the LiDAR.



## 8 Depth Image Correction

Instead of applying this correction with no sense for what these pixels represent, we could try another approach. This second method would consist of applying the correction to only the pixels that are "related" to the pixels for which we have depth information from the LiDAR. The question remains how to establish this relation between pixels. What comes directly in mind is using the color of the pixels. However the color can greatly vary depending on the lighting. Moreover, different objects with different depths can have similar colors.

Another approach to establish this relation could be to use the estimated depth by AdaBins. Even though the depths estimated by AdaBins can be quite off, it can be used to group the pixels being part of same plane. Indeed, AdaBins often places the pixels belonging to a given "plane" at relatively close depths (even if the depth of these "planes" may be far from accurate). With that in mind, our second method consists of applying the correction to only the pixels whose estimated depths is sufficiently close to the one at the point for which we have information from the LiDAR (directly or indirectly). In concrete terms, we look at the difference defined in Equation [8.6](#):

$$D_A[u_{(i-\Delta u)} + x, \frac{v_i + v_{(i-\Delta u)}}{2}] - D_A[u_{(i-\Delta u)} + x, h] \leq \text{Threshold} \quad (8.6)$$

If this difference is below a chosen threshold, we consider that the pixels at  $(u_{(i-\Delta u)} + x, h)$  are in the same plane as the point for which we have (direct or indirect) information from the LiDAR and, therefore, it is relevant to apply on them the correction based in the LiDAR information. This correction can be described by Equation [8.7](#) with "Depth error" such as defined in Equation [8.4](#).

$$D_C[u_{(i-\Delta u)} + x, h] = (D_A[u_{(i-\Delta u)} + x, h] + \text{Depth error}) \quad (8.7)$$

This second method works well. However, we remarked that we can simply assign the perceived depth value at the LiDAR height to the set of points that respects Equation [8.6](#). The corrected depth in this last method can be described as follows :

$$D_C[u_{(i-\Delta u)} + x, h] = (D_L[u_{(i-\Delta u)} + x, h] + \frac{x}{\Delta u} \Delta d) \quad (8.8)$$

With this last method we can lose some information about the slight variance of depth for different point of the same column belonging - according to Adabin - to the same plane. Nevertheless, this variance of depth can also be an error in the estimation made by Adadin. Applying this third method, described in Equation [8.8](#), straighten the depth image. The resulting depth image is much more regular which, in the case of indoor environments where encountered objects have usually straight shapes, is not a problem. According to us, this last method is the better choice

## 8.2 Results

In this section we will present and comment the results obtained using these algorithms and highlight the applied corrections. We will only focus on the last two methods mentioned. In this section, method 1 will refer to the second method introduced in the previous equation and summarized by Equation 8.6 and method 2 to the third and last method introduced expressed by equation 8.7. All the numbers of the Figures featured in this section are expressed in meters.

To highlight the correction, we will look at the results of 3 scenarios

- **Scenario 1** : two boxes placed at the same distance in front of a background.
- **Scenario 2** : two boxes placed at different distances in front of a background.
- **Scenario 3** : two boxes obliquely positioned at different distances in front of a background.

From Figures 8.3 to 8.26, as one can see the estimation from AdaBins is incorrect ; the depth is wrong and even the relative position of the objects (e.g. the relative positions of the boxes in scenario 2 and 3). With method 1 and method 2, the correction adjust those depth values leading to a much more accurate representation of the real position of the objects.

Comparing method 1 and 2, as explained in the previous section, method 1 keep the estimated depth from AdaBins and apply a "correction term" to this estimated depth, leading to a more "blurry" representation. Method 2 allocates the depth sensed by the LiDAR (directly or indirectly) to all the pixels respecting the condition expressed by 8.6. Therefore this second method leads to a more "straight" representation. Those behaviours are apparent in the Figures of this section. As already stated, due to the inconsistencies/errors of the estimation by AdaBins and the straightened rendering/appearance well suited for most indoor environments, the second method is preferred.

For each scenario, we indicate the distances of the boxes and the background from the camera. We have tried to be rigorous when measuring these distances but with the setup, perfect accuracy is difficult to obtain. Very slight differences in the measured depths and the ones displayed on the different figures can also come from small inaccuracy on the objects placement measurements.

Finally, to better visualize the different results of the AdaBins, Method 1 and Method 2, we end this section with some 3D visualisations of the 3D point clouds created by the ROS node for the different scenarios and displayed with Rviz (Figures 11.1 to 8.29).

### 8.2.1 Scenario 1

The first demonstration consists of two boxes located at the same depth and a background behind them. The boxes are located at a perpendicular distance of 1.4[m] away from the camera and the background is located at 2.5[m]. The following pictures display the demonstration setup and then a comparison of the depth images created by only AdaBins, Method 1 and Method 2 for scenario 1.



(a) Demonstration setup



(b) Demonstration setup

Figure 8.3: Pictures of the demonstration setup in Scenario 1



Figure 8.4: View captured by the monocular camera in Scenario 1

## 8 Depth Image Correction

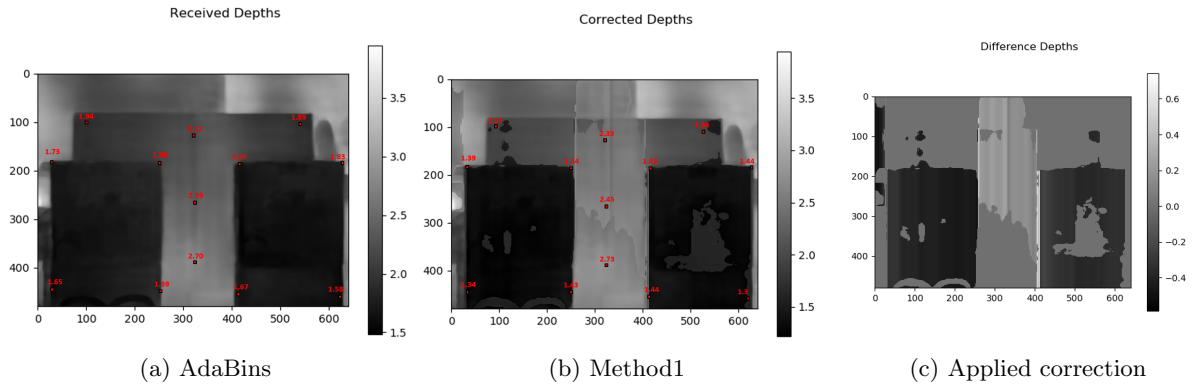


Figure 8.5: Comparison between the estimated depth by AdaBins and Method 1

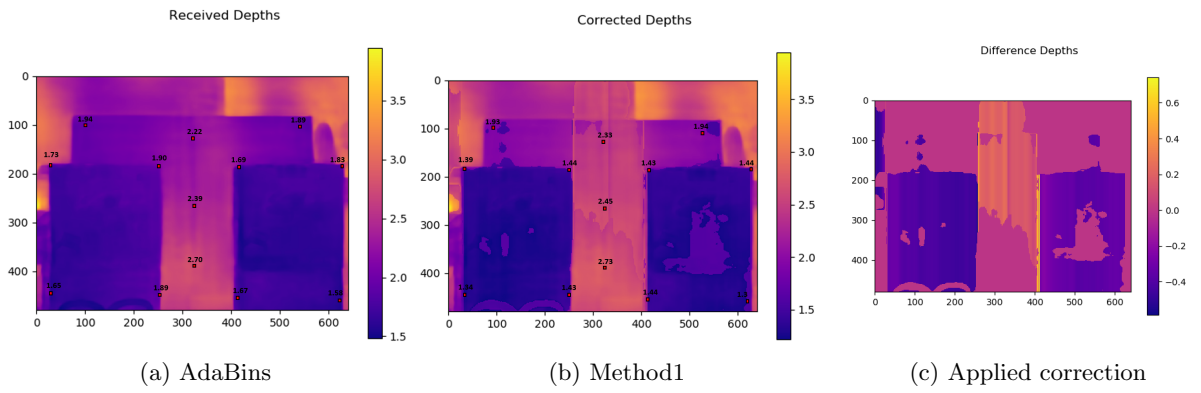


Figure 8.6: Comparison between the estimated depth by AdaBins and Method 1

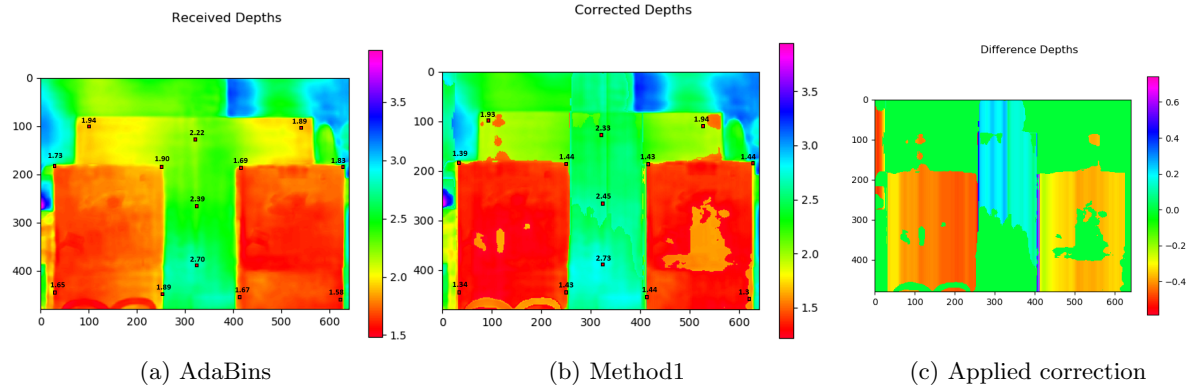


Figure 8.7: Comparison between the estimated depth by AdaBins and Method 1

## 8 Depth Image Correction

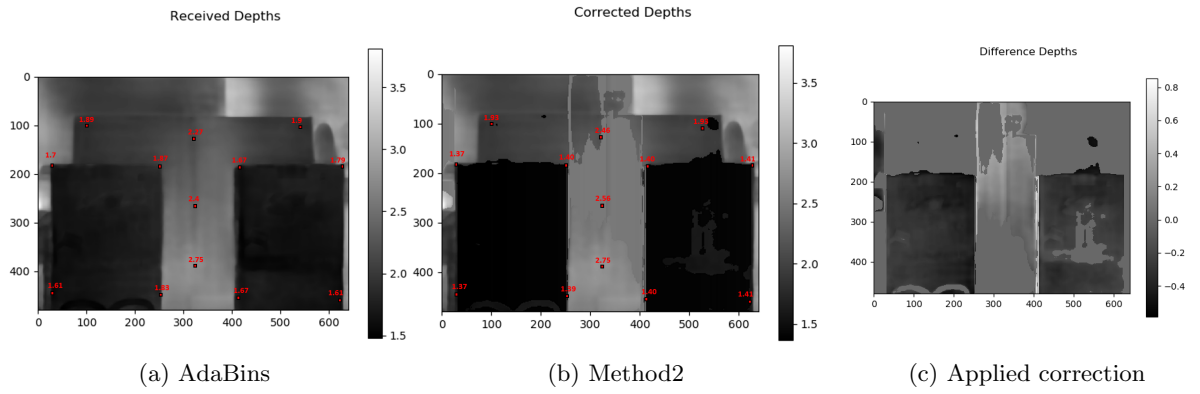


Figure 8.8: Comparison between the estimated depth by AdaBins and Method 2

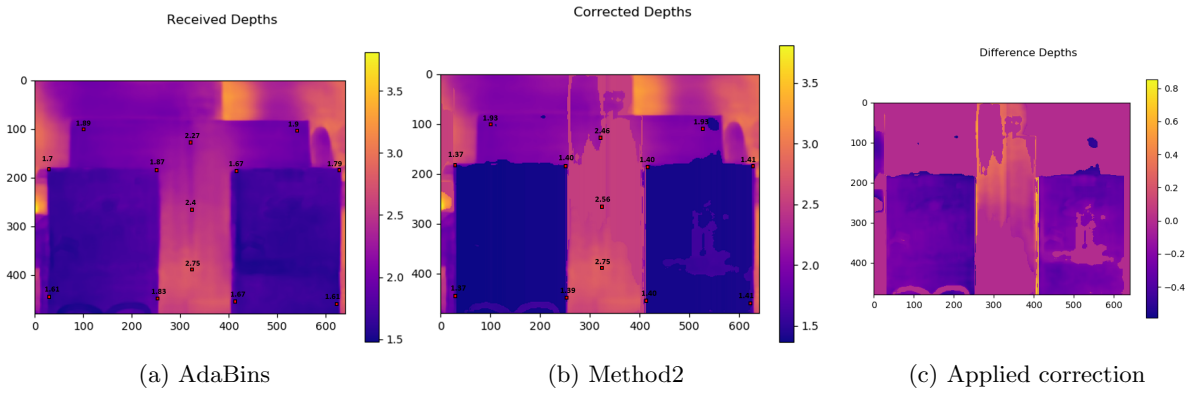


Figure 8.9: Comparison between the estimated depth by AdaBins and Method 2

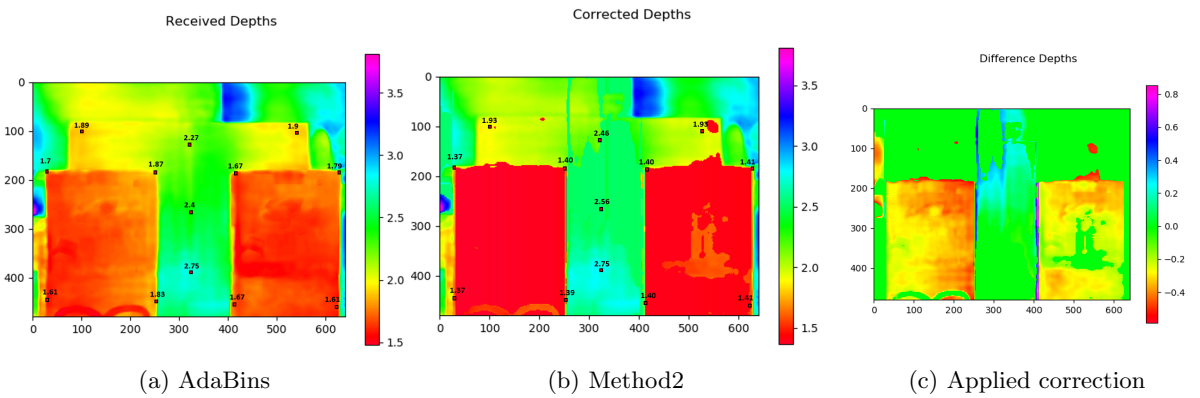
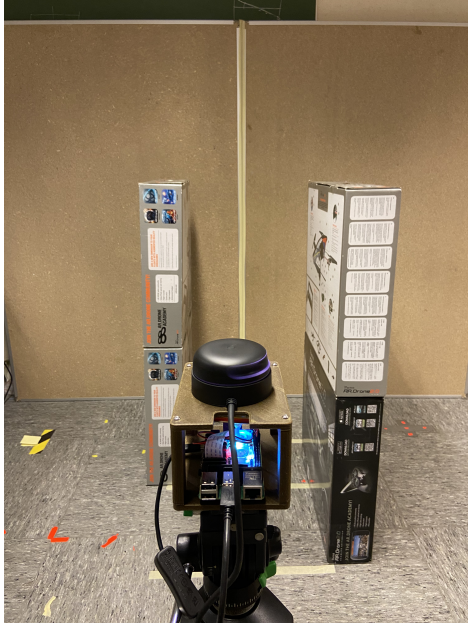


Figure 8.10: Comparison between the estimated depth by AdaBins and Method 2

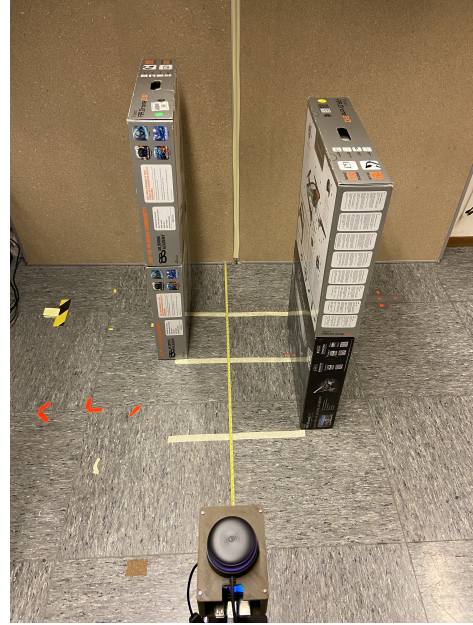


### 8.2.2 Scenario 2

In the second scenario, two boxes are placed at different depths. The visible face of the box on the left is placed at 1.18[m] away from the camera and, for the box on the right, at 0.79[m]. The background is at a 1.9[m] distance from the camera. The following pictures display the demonstration setup and then a comparison of the depth images created by only AdaBins, Method 1 and Method 2 for scenario 2.



(a) Demonstration setup



(b) Demonstration setup

Figure 8.11: Pictures of the demonstration setup in Scenario 2



Figure 8.12: View captured by the monocular camera in Scenario 2

## 8 Depth Image Correction

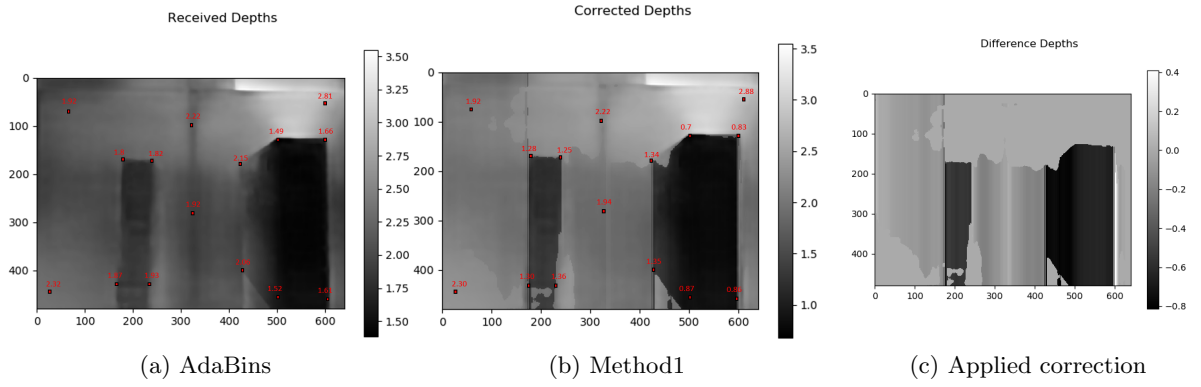


Figure 8.13: Comparison between the estimated depth by AdaBins and Method 1

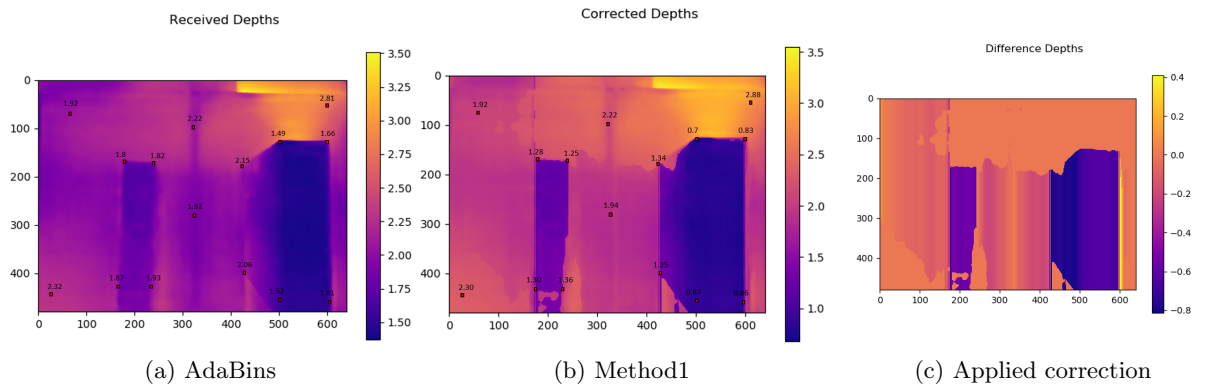


Figure 8.14: Comparison between the estimated depth by AdaBins and Method 1

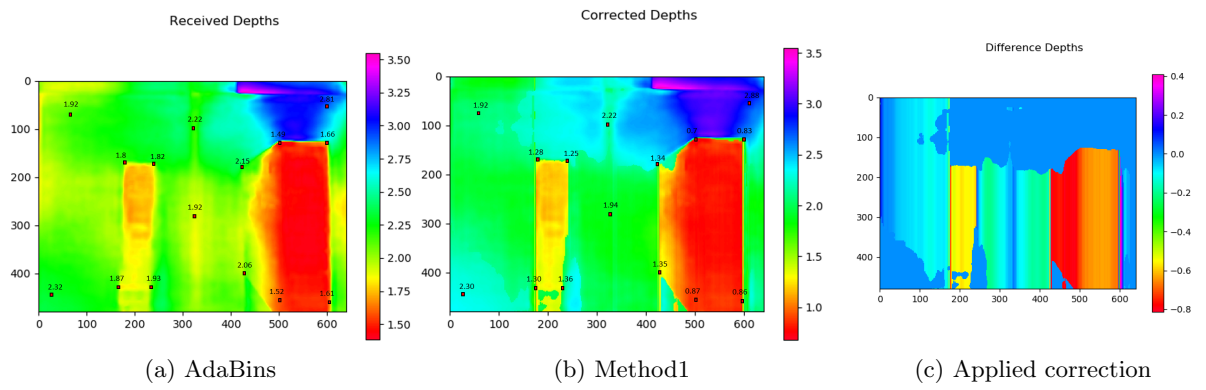


Figure 8.15: Comparison between the estimated depth by AdaBins and Method 1

## 8 Depth Image Correction

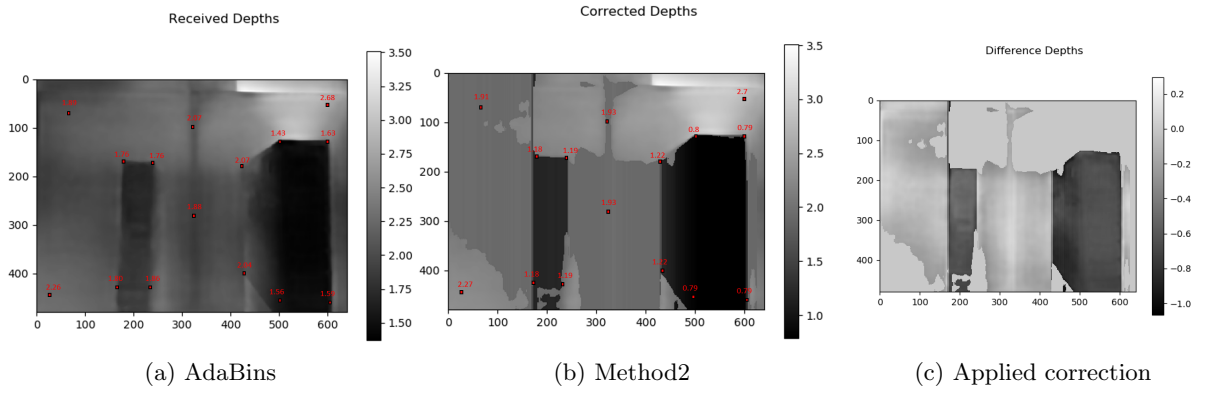


Figure 8.16: Comparison between the estimated depth by AdaBins and Method 2

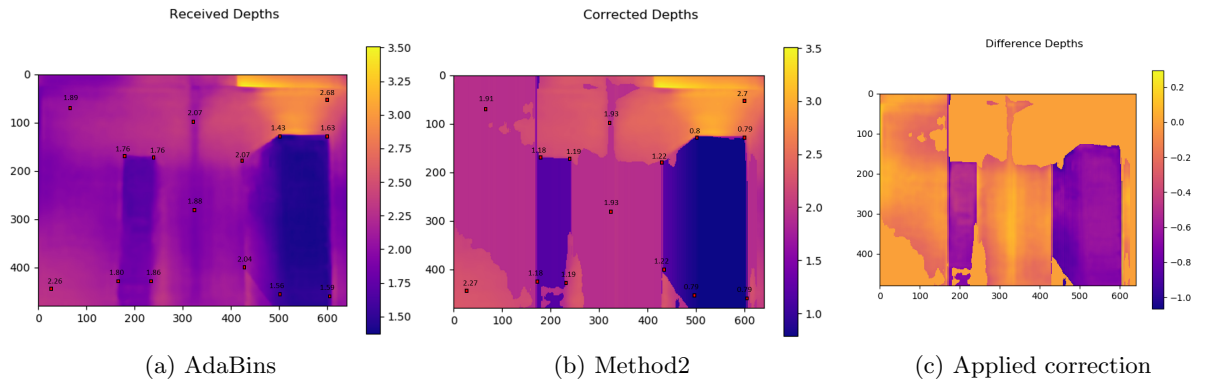


Figure 8.17: Comparison between the estimated depth by AdaBins and Method 2

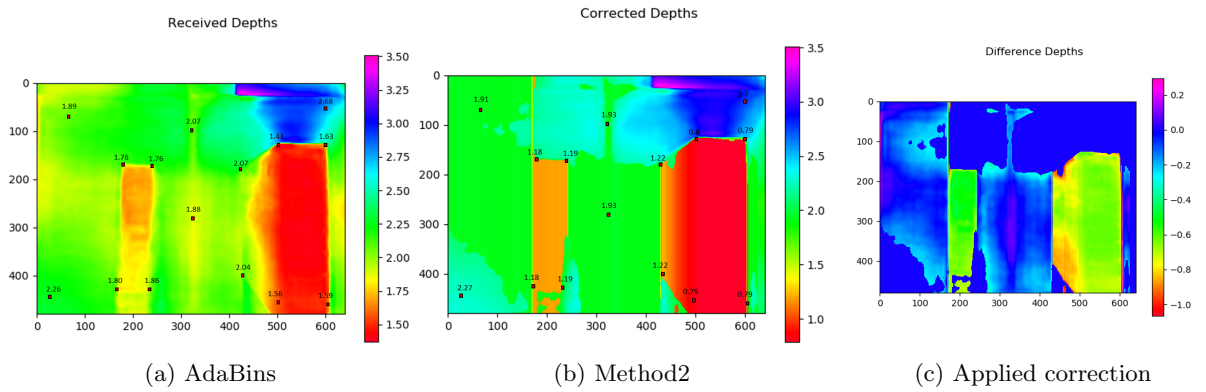


Figure 8.18: Comparison between the estimated depth by AdaBins and Method 2



### 8.2.3 Scenario 3

The third scenario consists of two boxes located at different depths and placed obliquely in the setup. The box on the left is positioned between 1.26[m] (for the side of the box closer to the camera) and 1.64[m] (for the side of the box closer to the background) away from the camera while the box on the right is positioned between at a distance from the camera ranging from 0.98[m] to 0.83[m]. (More precisely the box on the right is positioned at a depth of 0.98[m] and 0.64[m] but the box is cut on the picture, so we mention the distance (0.83[m]) as cropped in the image.) The background is at a 2.08[m] distance from the camera. The following pictures display the demonstration setup and then a comparison of the depth images created by only AdaBins, Method 1 and Method 2 for scenario 3.



(a) Demonstration setup



(b) Demonstration setup

Figure 8.19: Pictures of the demonstration setup in Scenario 3



Figure 8.20: View captured by the monocular camera in Scenario 3

## 8 Depth Image Correction

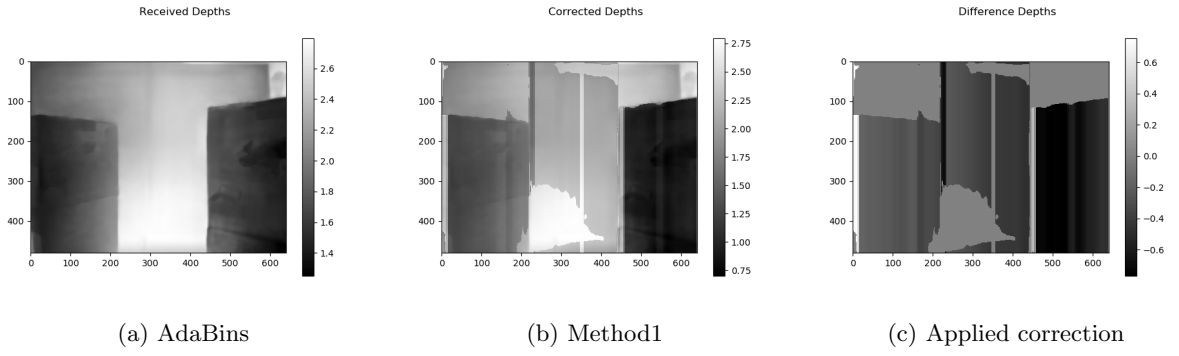


Figure 8.21: Comparison between the estimated depth by AdaBins and Method 1

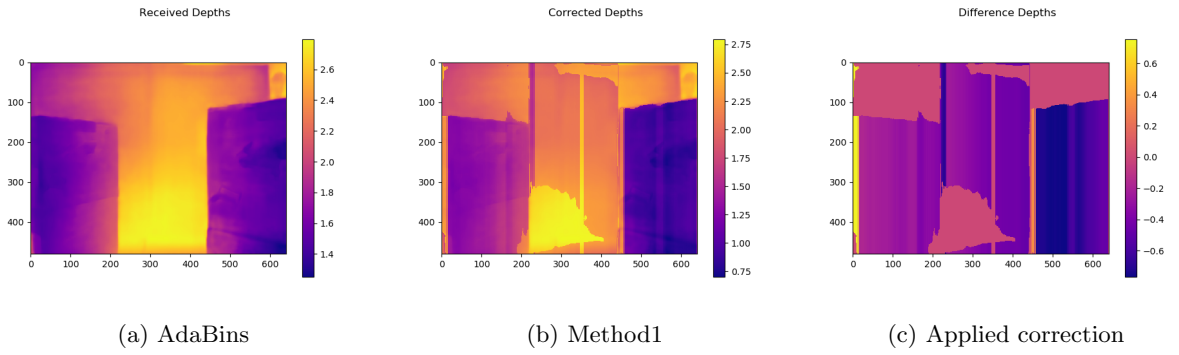


Figure 8.22: Comparison between the estimated depth by AdaBins and Method 1

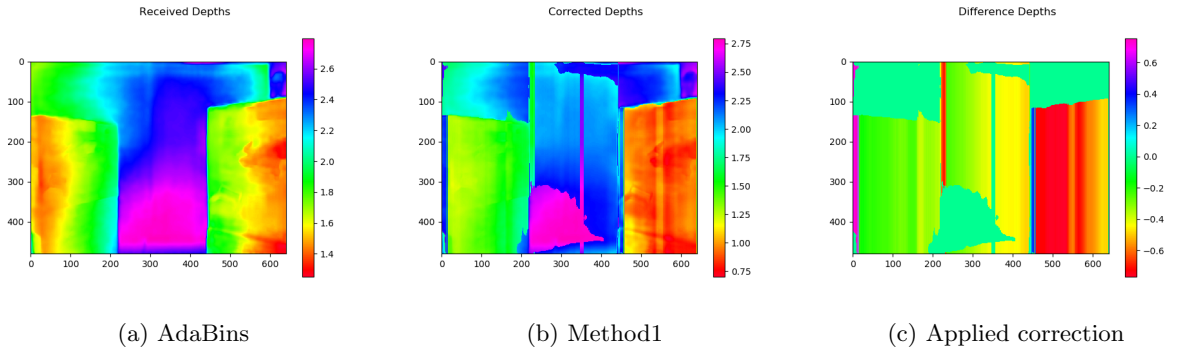


Figure 8.23: Comparison between the estimated depth by AdaBins and Method 1

## 8 Depth Image Correction

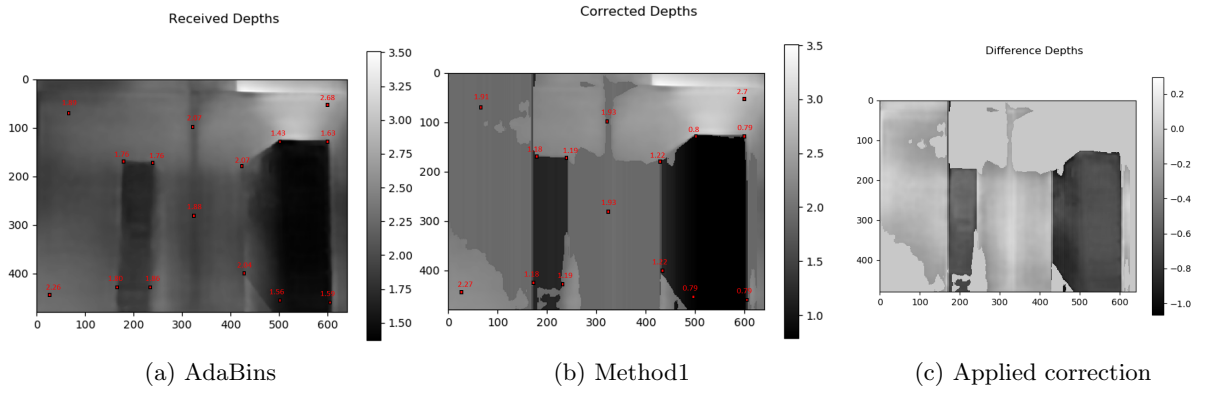


Figure 8.24: Comparison between the estimated depth by AdaBins and Method 2

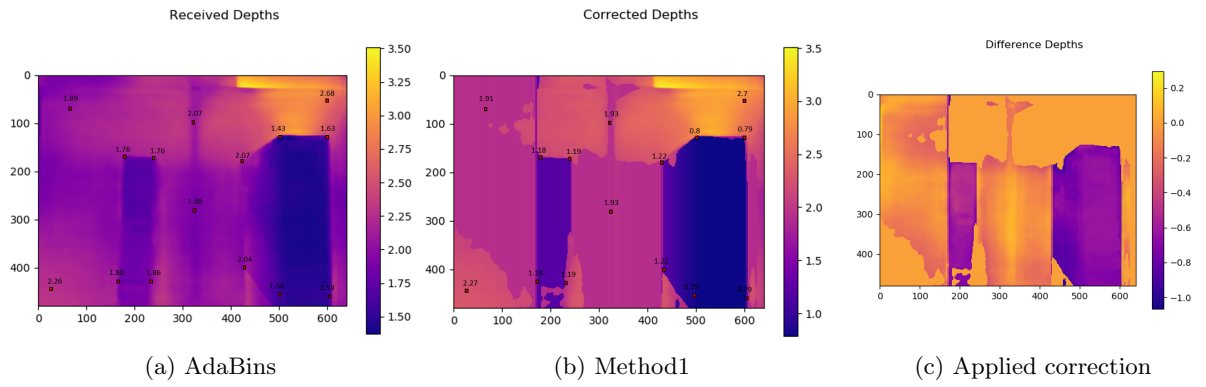


Figure 8.25: Comparison between the estimated depth by AdaBins and Method 2

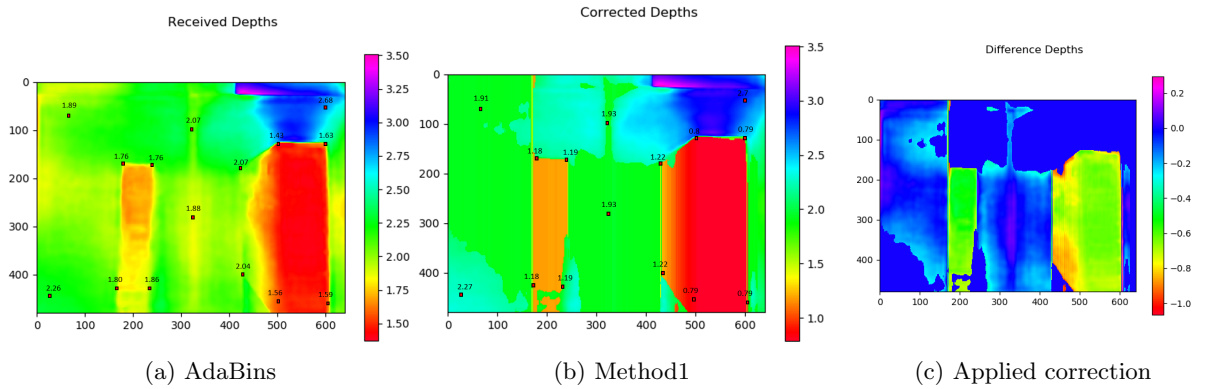


Figure 8.26: Comparison between the estimated depth by AdaBins and Method 2

### 8.2.4 3D visualisation

#### Scenario 1

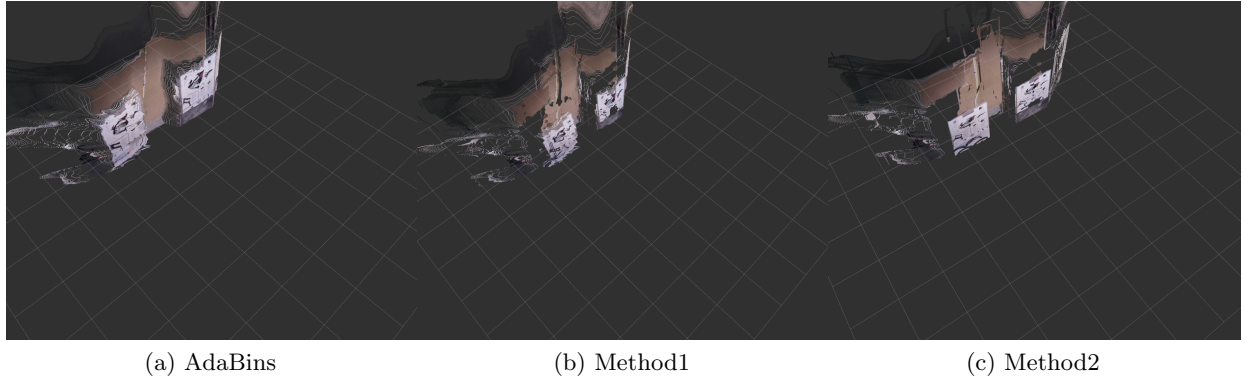


Figure 8.27: Comparison between the estimated depth by AdaBins and Method 2

#### Scenario 2

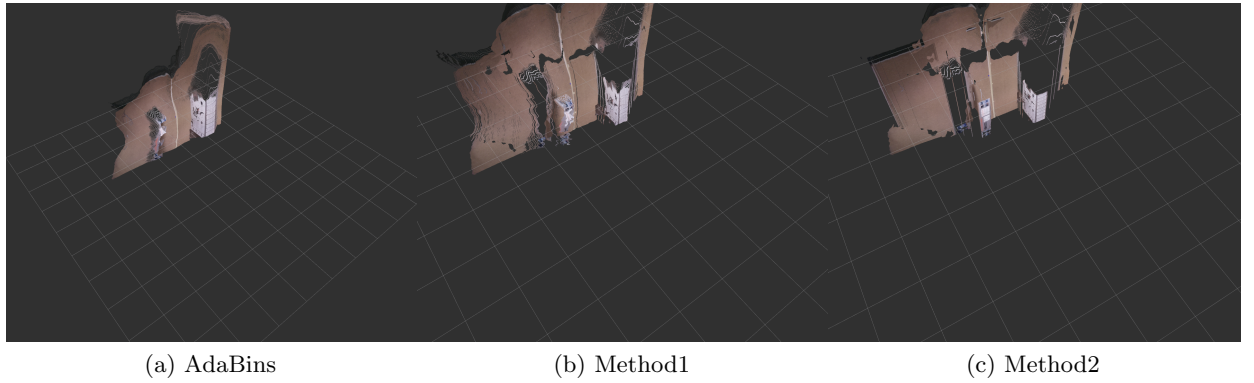


Figure 8.28: Comparison between the estimated depth by AdaBins and Method 2

#### Scenario 3

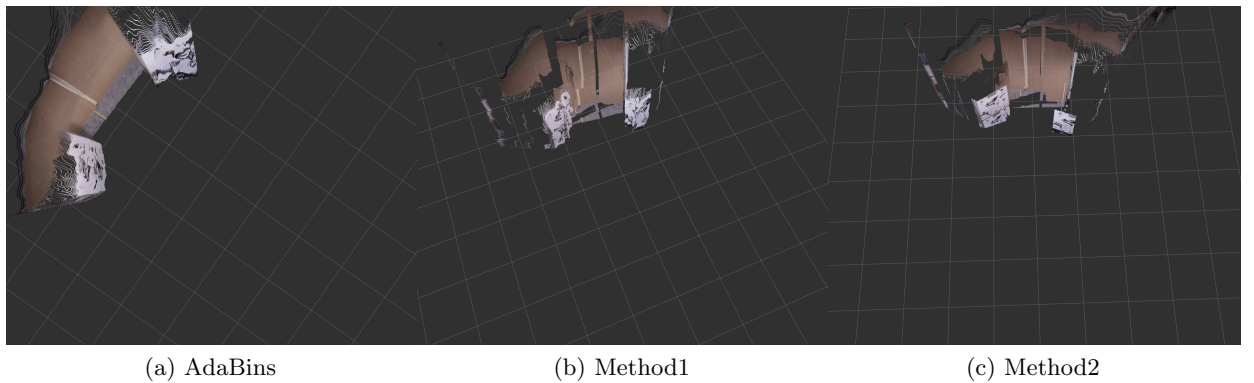


Figure 8.29: Comparison between the estimated depth by AdaBins and Method 2

### 8.3 Other algorithms

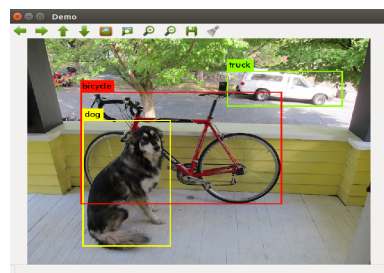
In our correction, we use AdaBins to get a first estimation of the depth of each pixel and then we adjust those depths with the LiDAR information, using the estimated depth as a tool for segmenting the image (this is what does Equation 8.6).

We then thought about other ways to implement the correction. Indeed, there exist lots of well-known algorithms for image segmentation. While we use the estimated depth by AdaBins to segment pixels we could have used other characteristics such as the colors of the pixels, as mentioned in the section titled *Correction process*. Although colors would have given poor results for such environment, other methods could be utilized.

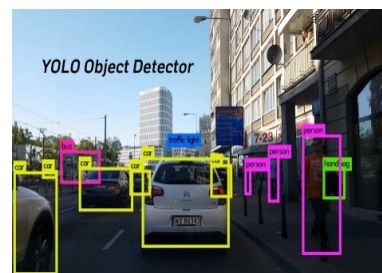
Image segmentation consists of collecting similar pixels together and separating out dissimilar pixels. We will not cover in details all the possible algorithms for image segmentation as it is not the goal of this master's thesis. The objective of this section is to nourish a reflection of the different methods that could be utilized for such application.

Among the various well-known categories of method, we can list : Threshold methods, Edge Based Segmentation, Region Based Segmentation, Clustering Based Segmentation, Watershed Based Method and Artificial/Convolutional Neural Network Based Segmentation. Instead of using Adabins to determine if the point scanned by the LiDAR is "related" to other pixels in the image, we could use various algorithms from these different categories.

Segmentation is closely related to object detection. One particularly interesting algorithm, at least according to us, is the YOLO algorithm. YOLO, which stands for "You Only Look Once", is a very well known state-of-the-art algorithm for object detection and widely used for numerous projects. This algorithm is based on a convolutional neural network that takes as input an image and detects and recognizes various objects in the picture (in real-time). It determines set of pixels belonging to a object and label those objects. Figure 8.30 is a example of YOLO utilization.



(a) YOLO Example 1



(b) YOLO Example 2

Figure 8.30: YOLO output examples



YOLO is particularly known for its speed of execution, its high accuracy and its learning capabilities. YOLO applications can be found in autonomous driving (to detect objects around cars such as vehicles, people, and parking signals), in wildlife observations (to detect and identify various types of animals in forests) or even in security applications (security systems to enforce security in an area, detect intruders or identify problematic situations).

In the context of autonomous UAVs, the main advantage of using an algorithm detecting and labelling the objects, is the possibility for the drone to better understand its surrounding. Not only mapping the environment but actually have a sense for what the objects constituting this environment are.

The problem is that the pre-trained YOLO versions do not take many of the commonly encountered objects in indoor settings. However it can still detect many of them but with erroneous labelling. To illustrate, Figure 8.31 display the YOLO output for the third scenario explored in the last section. As one can see, the algorithm detects the two boxes but wrongly labels them as "refrigerator".



Figure 8.31: Yolo results with the setup of scenario 3

The upside is that YOLO could be trained to identify traditional indoor objects. It could even identify to better interact with objects such as doors and even glass windows (which causes difficulty for the LiDAR sensing, therefore recognize and adequately handle more challenging situations).

We could even implement a method that does not need the help of AdaBins for a first depth estimation but directly identify objects and attributes depth to the object scanned by the LiDAR and "undetermined" depth values for pixels we do not have any information yet. Indeed, in method 2, the only real purpose of the estimated depth by AdaBins is to

segment the pixels (to better allocate the depth sensed by the LiDAR to the corresponding groups/clusters of pixels). We could even implement the drone in a way that makes it scan the surrounding at different height in order to gather depth information of the objects constituting the scene. Figures 8.32 illustrates this point for a case with a homogeneous background. With this method, we could neglect AdaBins and therefore gain considerable computational time.

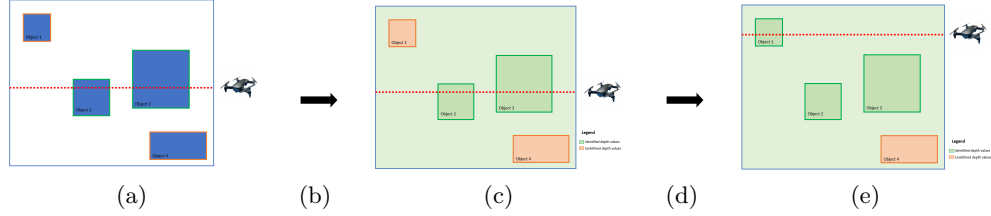


Figure 8.32: YOLO output examples

## 9 LiDAR, additional gains regarding drone technology

In this short section, we will cover some interesting advantages that an on-board LiDAR can bring to a 3-dimensional mobile robot (drone).

### 9.1 Obstacle avoidance

One of the first requirements for drone is obstacle avoidance. Indeed, when flying into an indoor environment, the drone is sensibly more prone to encounter near by objects or narrow doorways. Having LiDAR data at disposal is a key advantage comparing to RGB-D cameras or proximity sensors. Each scan has a minimum of 360 degrees of field of view which makes this sensor, in terms of obstacle detection, far superior to a RGB-D camera or directional range sensor. That is why we decided to implement a simple ROS node that constantly monitors the smallest value of each scans coming from the LiDAR and publishes a warning when values are below an arbitrary threshold. The node is called "emergency\_stop" and publishes a message containing the scan data with a boolean value set to true when an obstacle is detected.

### 9.2 Odometry

LiDARs are also useful for movement estimation. The nature of their measurements gives the possibility to track moving objects trough multiple scans and by reverse deduction allows to guess the position of the robot in space. In this section, we will describe two of the most known odometry algorithms based on LiDAR sensors.

#### 9.2.1 ICP odometry

The most common method to accomplish this task, is called "Iterative Closest Point Odometry". This technique stitches one scan (the source) to the previous one (the target) by iteratively searching for the best rigid spatial transform between the two. With the best matching transformation matrix, the robot is capable to deduce the corresponding translation and rotation of its position between two scans.

To find the best transformation matrix, the algorithm is relatively simple. It begins by associating the nearest neighbour between the target and the source and then selects the best rigid transformation that minimises the mean-squared distance of each pair of points. An example of the minimizing equation can be found in equation [9.1](#).



$$\tilde{T} = \underset{T}{\operatorname{argmin}} \frac{1}{N} \sum_i^N \|t_i - Ts_i\|^2 \quad (9.1)$$

With  $T$  the rigid transformation candidate and  $t_i$  and  $s_i$  the target and source points respectively.

Once done, the algorithm computes the alignment error and repeats the process until the error is considered acceptable or the maximum iterations number is reached. An example can be found on figure [9.1](#).

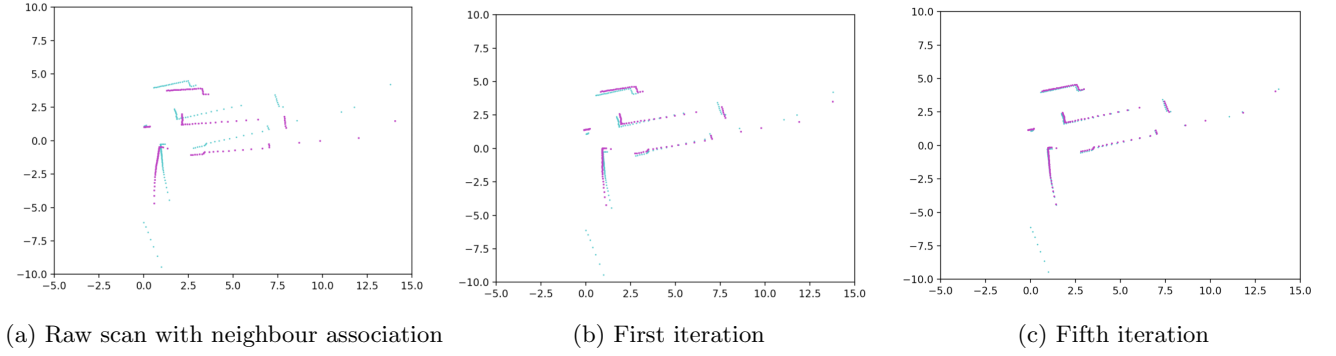


Figure 9.1: ICP after five iterations

### 9.2.2 Gauss-Newton Optimization odometry

This method needs the construction of a map in parallel to the odometry calculation to work. The map is constructed as a grid of finite elements of fixed size and each cell contains a probability of occupation between 0 and 1. The map can be seen as a probability function with a 2D or 3D pose as input and a probability of occupation as output. Analogously to an ICP method, the goal of the method is to find the rigid transformation that will minimise the error between the scan and the map. In other words, the equation to minimise is:

$$\xi^* = \underset{\xi}{\operatorname{argmin}} \sum_{i=1}^n [1 - M(S_i(\xi))]^2 \quad (9.2)$$

After a first order Taylor approximation, the equation to minimize becomes:

$$\sum_{i=1}^n \left[ 1 - M(S_i(\xi)) - \nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi} \Delta \xi \right]^2 \rightarrow 0 \quad (9.3)$$

With  $S_i(\xi)$  the world coordinates of the scan points,  $M(\dots)$  the map's probability output for each world coordinates and  $\Delta \xi$  the variation of the rigid transformation leading to the minimization. The method is fully described in [\[48\]](#) for more informations.

## *9 LiDAR, additional gains regarding drone technology*

In comparison to ICP, this method does not need heavy computation but requires instead more memory to work as the map needs to be stored progressively. In our work, we decided to experiment with the ICP method as it was easier to implement. This will be part of a SLAM ROS node called RtabMap and will be covered in chapter [chapter 10](#).

# 10 LiDAR-Camera fusion applied in SLAM

This chapter aims to describe the two main SLAM algorithms used during our research. We tried to evaluate the relevance of our method in real SLAM situations and eventually show a potential benefit of its usage. This section also explains the choices made while highlighting the failures and successes in the implementation of two slam algorithms aimed at evaluating the method approached in this work.

## 10.1 ROS environment

Starting from previous chapters, our ROS environment has now multiple interconnected nodes and topics available as seen on figure [10.1](#). This list of topics gives the possibility to implement various SLAM algorithm. With the `depth_correction` node giving a better depth measurement than the initial messages provided by `monodepth_adabin` we were curious to see if this was enough for a slam algorithm to work reliably. Two SLAM algorithms were tested namely, OBRS-SLAM2 and RTAB-Map. The selection of those two SLAMs was a weighted sum of their ability to work with 3D point cloud or depth maps in conjunction with other sensors taking into account their popularity among the ROS community.

## 10.2 RTAB-Map

### 10.2.1 Working principle and implementation

RTAB-Map was an interesting candidate as it combines the measurements of multiple sensors at the same time. It was designed to work primarily with a combination of visual sensors and LiDAR and that is why it seems, at first glance, well suited for our setup. RTAB-Map (Real-Time Appearance-Based Mapping) is an RGB-D SLAM approach based on a loop closure detector. The loop closure detector uses a bag-of-words approach in order to determinate if a new image detected by an RGB-D sensor is from a new location or from a location that has been already visited. An example of a bag of words consist of an histogram of the occurrence of the key-features detected in each scene, an illustration of the concept is on figure [10.2](#). To work, RTAB-Map has to have access to a minimum of two types of information, odometry and 2D or 3D measurements. With our setup we can feed, without real problems, all of our generated topics to the algorithm. Looking at figure [10.3](#), we will replace the RGB-D camera with the output of our corrected neural network, the laser scan with the output of the RPLidar A3 and the odometry node with the output of a node called `ICP_odometry`. This gives the corresponding flowchart [10.4](#) where the connected topics are highlighted in red.

## 10 LiDAR-Camera fusion applied in SLAM

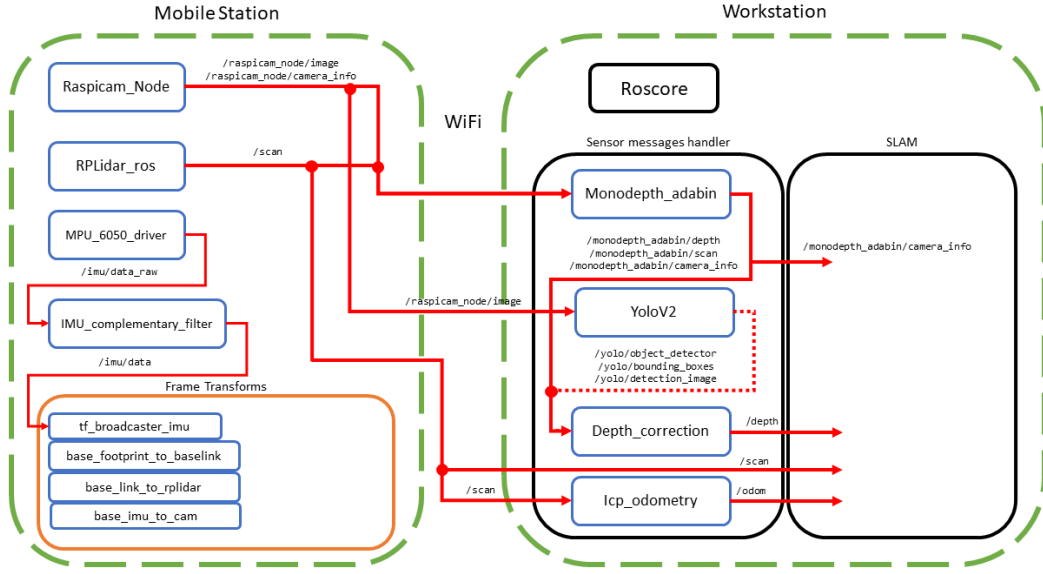


Figure 10.1: Available messages for SLAM implementation

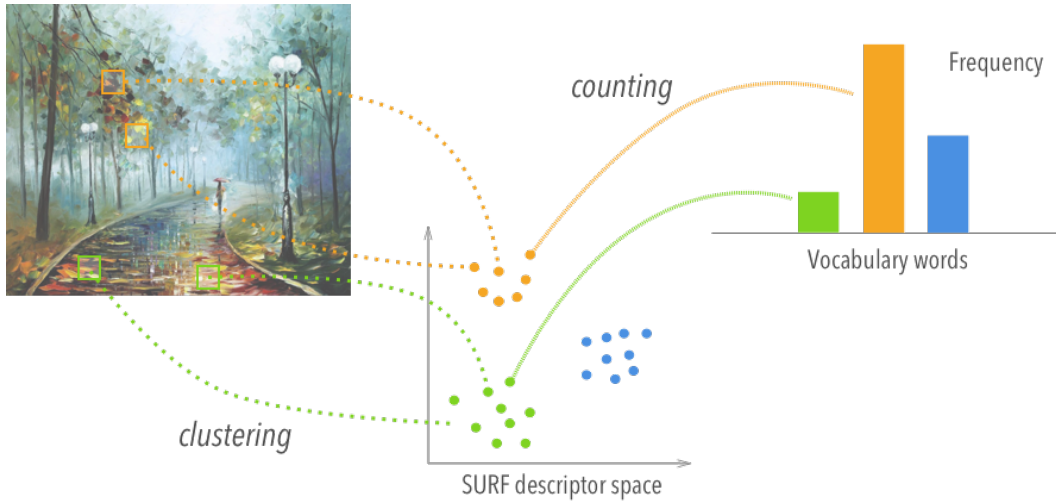


Figure 10.2: Bag of words concept

ICP\_odometry uses the technique of iterative closest point to generate an odometry value from the scans of the RPLidar A3. With this value the node also generates what is called a transform named `"/odom"` between the map that RTAB-Map is supposed to create and the `"base_footprint"` of the robot on which all of the transforms are connected. An overview of the Transforms is available on figure [10.5](#) for clarification. To understand what this graph is, you have to treat it like a real tree. The stem (`"/world"`) is fixed in space and `"/odom"` gives a position relative to the `"/world"`. The `"/base_footprint"` is directly

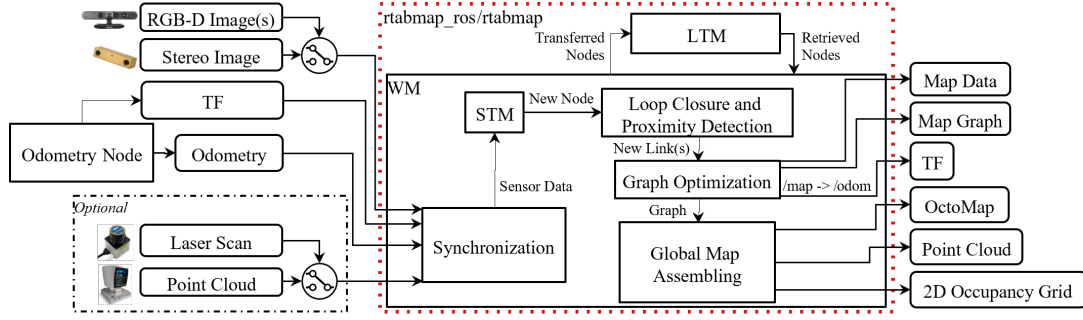


Figure 10.3: Global view of the ROS implementation of RTAB-Map [51]

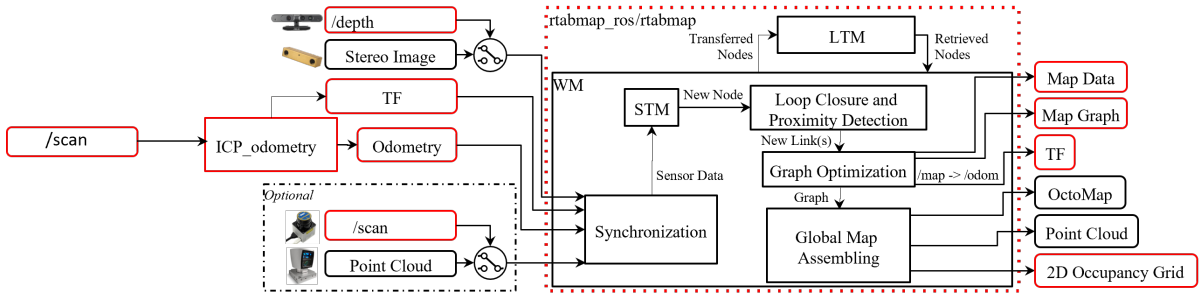


Figure 10.4: Global view of the ROS implementation with our topics connected

connected to `/odom` and thus shares its attributes (the position regarding the `/world`), it is said to be static. Each link of the graph represents a fixed rotation and/or translation of a frame compared to the previous one.

ICP is a relatively common technique and lots of ROS packages already exist. Our implementation is a prebuilt node from ROS and came embedded in the package containing RTAB-Map.

### 10.2.2 Results and discussion

We tinkered a lot to implement this node on our setup. But we were not able to launch it with the output of `depth_correction`. In fact, the depth topics that we were sending did not have a high enough frame-rate to make a match with the odometry data. The ICP\_node updates its position at approximately 1Hz while the depth values of `Depth_correction` are send at a rate of one image per 15 seconds. RTABMap expects a higher frame rate of the camera compared to the odometry update rate. The node is consequently not able to sync the two messages and build a corresponding map.

To still be able to experiment with the node and have some data to work on, we decided to disconnect the depth topic from RTABMap and replace it with the scans of the LiDAR. This converts the node to a 2D SLAM and we decided to test the algorithm accuracy compared to a known path.

## 10 LiDAR-Camera fusion applied in SLAM

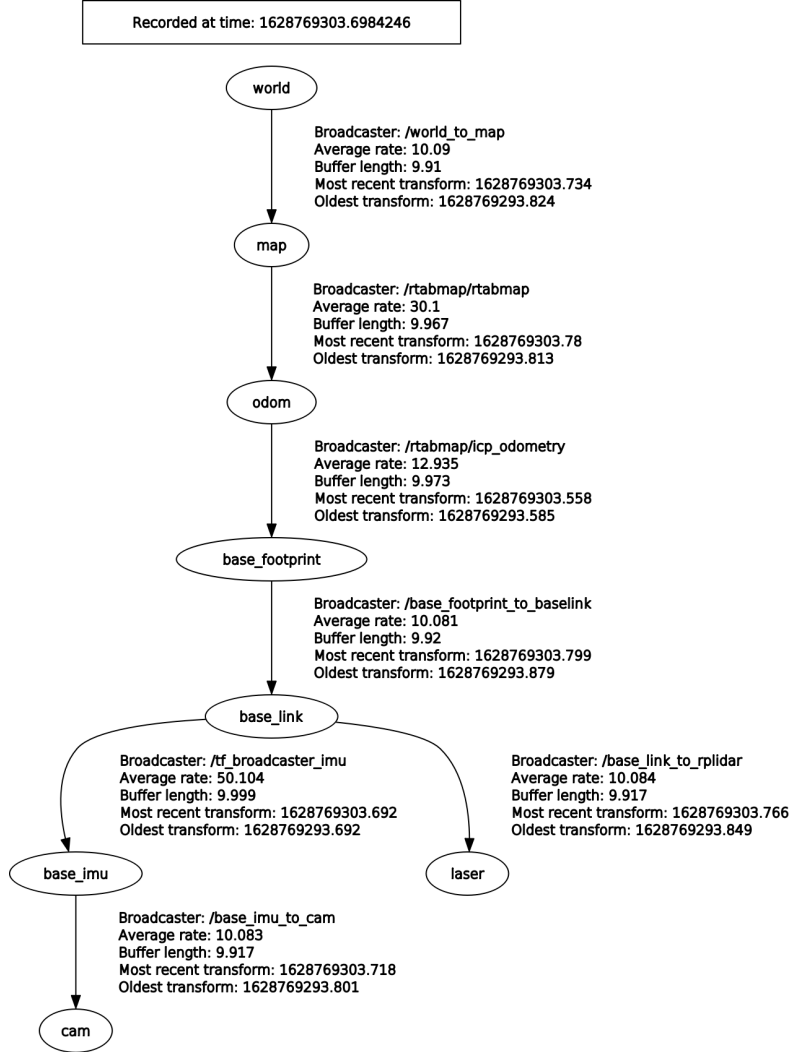


Figure 10.5: Complete transform tree generated by RTABMap

The setup consist of a known path of four way-points. The path is covered 3 times and at each way-point a measurement of the position is made. The dimension of the experimental room are  $5 * 7.3 \text{ m}^2$ . The door of the room was open during the first pass and closed for the next 2. The results are available in table [10.1](#) and on figure [10.6](#).

This short test shows that RTABMap, combined with the RPLidar A3, is precise to the tenth of a meter with very consistent results. There is little to no drift in the measurement over the 18.06 meters of path.

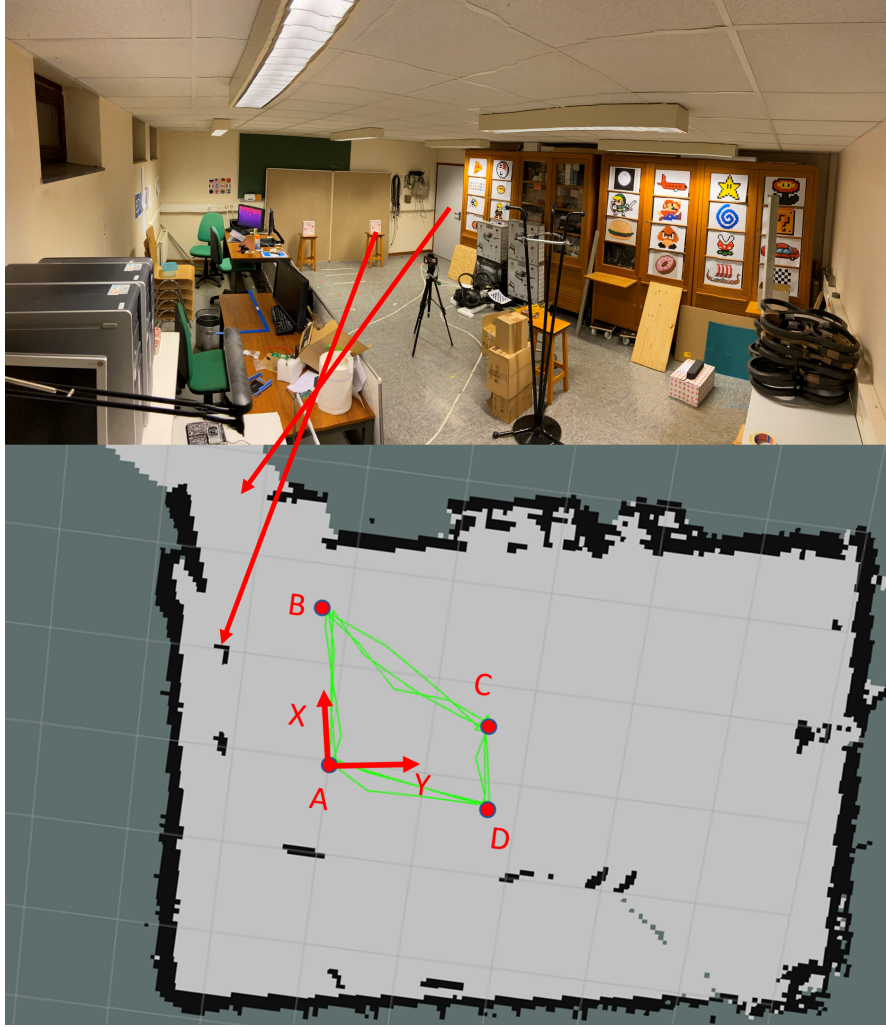


Figure 10.6: Map generated by RTABMap and the resulting passes with the original environment

| Way-point        | A             | B             | C              | D               |
|------------------|---------------|---------------|----------------|-----------------|
| Truth (x, y) [m] | (0, 0)        | (1.5, 0)      | (0.411, 1.677) | (-0.389, 1.677) |
| #1               | (0, 0)        | (1.52, -0.2)  | (0.56, 1.56)   | (-0.255, 1.63)  |
| #2               | (-0.4, 0.016) | (1.52, -0.18) | (0.58, 1.53)   | (-0.228, 1.68)  |
| #3               | (0.0)         | (1.5, -0.218) | (0.582, 1.55)  | (-0.25, 1.66)   |
| mean error [m]   | 0.13          | 0.2           | 0.21           | 0.147           |

Table 10.1: Results of RTABMap

## 10.3 ORB-SLAM2

After the implementation failure of RTABMap, we decided to turn our attention to a second candidate, ORB-SLAM2. This SLAM algorithm is based around a feature detector and descriptor called ORB [69]. The Master thesis from Ketchban (2018) [43] mentioned how this feature detector could be used for 3D monocular mapping with encouraging results. Knowing that ORB-SLAM2 could directly handle RGB-D images, we decided to test our Depth\_correction node on this algorithm for a performance comparison with the monocular version.

### 10.3.1 What is ORB, Oriented FAST and Rotated BRIEF ?

ORB is a response to SIFT and SURF. Unlike ORB, SIFT and SURF are not open source. When implemented for a commercial use, the user has to pay some fees. Reading the literature [92], it is said that ORB is two orders of magnitude faster than SIFT while having the same performance in feature detection. ORB detector is based on FAST with a fast and accurate orientation component and ORB descriptor is built on oriented BRIEF with variance and correlation analysis.

#### FAST with rotation and scale invariance

Fast compares each pixel's brightness of the image with 16 of its neighbours. If more than 8 pixels are brighter or darker than the concerned pixel, it is selected as a keypoint.

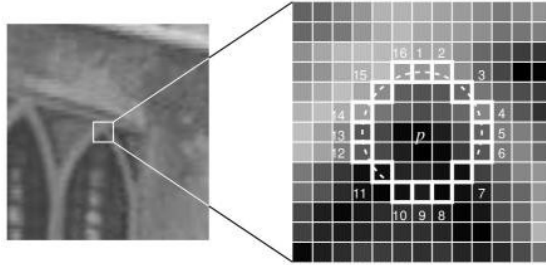
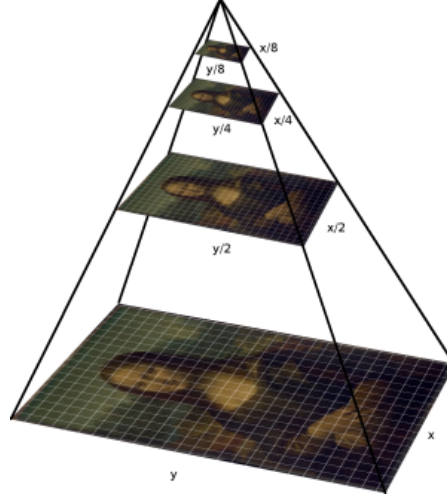


Figure 10.7: FAST comparison representation. Credits to [92]

To add an scale invariance component to the key-point detection, ORB computes a set of different scales of the image and recomparates all the pixels on each scale. Making it robust against scale changes.

ORB then computes a vector of each key feature representing the intensity centroid. Following those equations:



Figure 10.8: Multi scale representation. Credits to [92](#)

$$m_{pq} = \sum_{x,y} x^p y^q I(x, y) \quad (10.1)$$

$$C = \left( \frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \quad (10.2)$$

$$\theta = \text{atan2}(m_{01}, m_{10}) \quad (10.3)$$

$$(10.4)$$

$C$  is the vector's amplitude and  $\theta$  is the orientation giving ORB a relative robustness against rotations.

## BRIEF

BRIEF starts by smoothing the image with a gaussian kernel and then selects a random pair of pixels around the key-point. The first pixel in the random pair is drawn from a Gaussian distribution centered around the key-point with a standard deviation or spread of sigma. The second pixel in the random pair is drawn from a Gaussian distribution centered around the first pixel with a standard deviation or spread of sigma by two. Now if the first pixel is brighter than the second, it assigns the value of 1 to corresponding bit else 0.

BRIEF repeats the process 128 times for each key-point and creates a vector containing all the values.

### OBS-SLAM2 working principle

ORB-SLAM2 works on three steps, tracking, local matching and loop closing. A complete flow chart of the algorithm is available on figure [10.9](#).

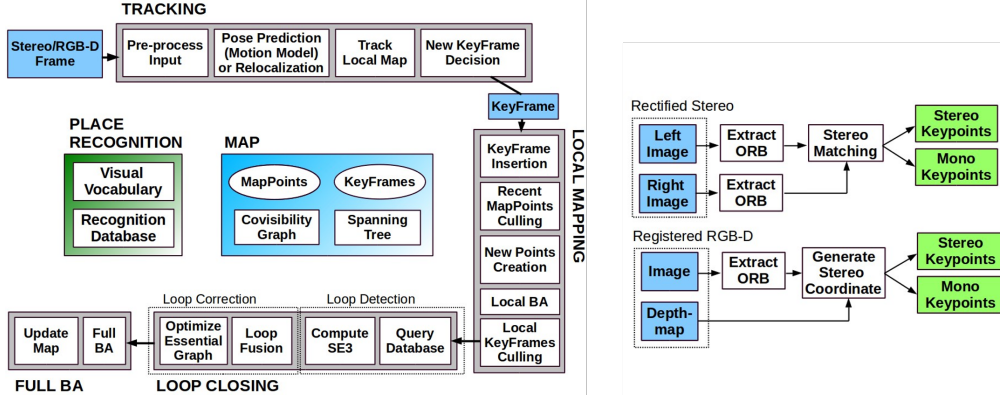


Figure 10.9: Flow chart of ORB-SALM2 [\[69\]](#)

The tracking consists on frame to frame feature matching on a local map to determine on a reduced part of the map the real time location of the camera. It performs a motion-only bundle adjustment for placing the feature in the correct position with minimal errors. Then it makes local maps and optimizes them using algorithms like ICP (Iterative Closest Point) and performs a local Bundle Adjustment so as to compute the most probable position of the camera. Finally it performs a loop closure by graph optimization to correct accumulated drifts.

It gathers its visual features with a 3D variant of the ORB features detector and descriptor. Those features are called stereo key-points and are firstly detected on the RGB image coming from the depth sensor. If it is a stereo camera their location is stored as  $(u_L, v_L, u_R)$  being  $(u_L, v_L)$  the coordinates on the left image of the stereo camera and  $u_R$  the horizontal coordinate of the right image. On the other hand, if it is a RGB-D camera, the features  $(u_L, v_L)$  are detected on the single RGB image and  $u_R$  is computed as such:

$$u_R = u_L - \frac{f_x b}{d} \quad (10.5)$$

With  $f_x$  the horizontal focal length and  $b$  the baseline between the structured light projector and the infrared camera. In our case, we chose an averaged arbitrary value for  $b$  as this parameter is unknown (typical value is 8 cm). The depth associated to  $u_R$  is used to triangulate the key-point across multiple frames and gives scale to the scene. The descriptor can then take care of the storage of the key-points.

#### 10.3.2 Implementation

The ROS implementation of ORB-SLAM2 has two work mode, monocular and stereo-  
RGB-D mode. We implemented the two modes to make a relevant comparison between

our depth\_correction and the vanilla setting of ORB-SLAM2. The corresponding ROS environment for monocular implementation is the following [10.10](#).

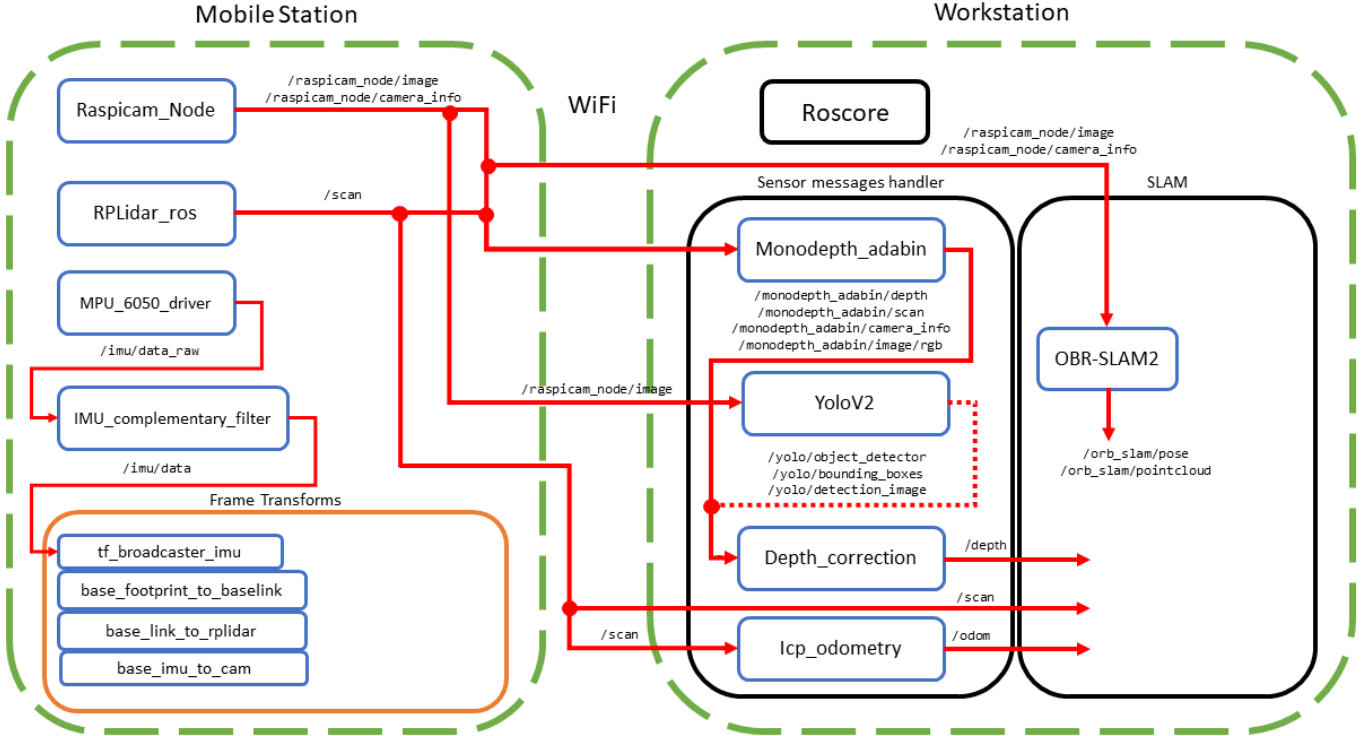


Figure 10.10: ROS flow chart of ORB-SLAM2 monocular

And the stereo implementation for our depth correction is on in figure [10.11](#).

The output of the two setups are the same, a pose that tells the robot its position in space and a pointcloud constructed with the detected key-features.

### 10.3.3 Results and discussion

The modus operandi is exactly the same as explained in RTABMap but in addition, we made two sets of 3 passes for the monocular setup. One set with the camera always oriented in the same direction and one set with the camera pointing to the next step direction. This gives a realistic situation of what a robot behavior could be. Indeed, when testing the algorithm, we noticed that for the monocular setup, rotation of the camera (on the x and y plane) could easily lead to the lost of tracking while a rotation free camera movement did not suffer from this problem.

#### ORB-SLAM2 monocular

The result of the first pass with a fixed camera is available in table [10.2](#).

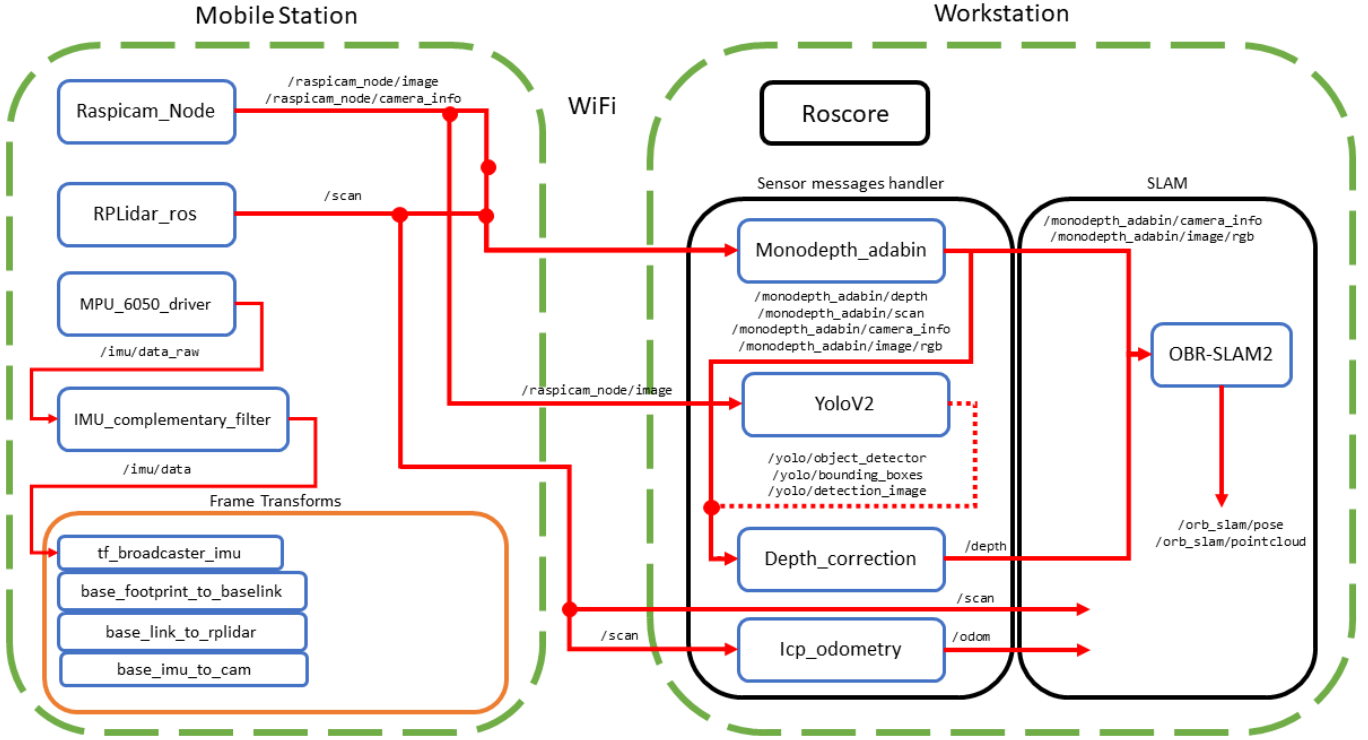


Figure 10.11: ROS flow chart of ORB-SLAM2 stereo

| Way-point        | A              | B              | C              | D               |
|------------------|----------------|----------------|----------------|-----------------|
| Truth (x, y) [m] | (0, 0)         | (1.5, 0)       | (0.411, 1.677) | (-0.389, 1.677) |
| #1               | (0, 0)         | (1.598, -0.14) | (0.49, 1.68)   | (-0.36, 1.75)   |
| #2               | (-0.0, 0.085)  | (1.43, 0.065)  | (0.48, 1.668)  | (-0.35, 1.72)   |
| #3               | (0.015, 0.082) | (1.435, -0.07) | (0.45, 1.67)   | (-0.35, 1.72)   |
| mean error [m]   | 0.056          | 0.121          | 0.062          | 0.065           |

Table 10.2: Results of ORB-SLAM2 monocular with fixed camera direction

And the generated map after the 3 passes is available on figure 10.12.

Results with a directional camera behavior are on table 10.3.

The resulting 3D map seen from above is to be seen on figure 10.13. The system effectively reconstruct a map but is not precise enough to show a realistic representation of the back of the room (shown by the question mark on the picture). A comparison will be made in figure 10.18.

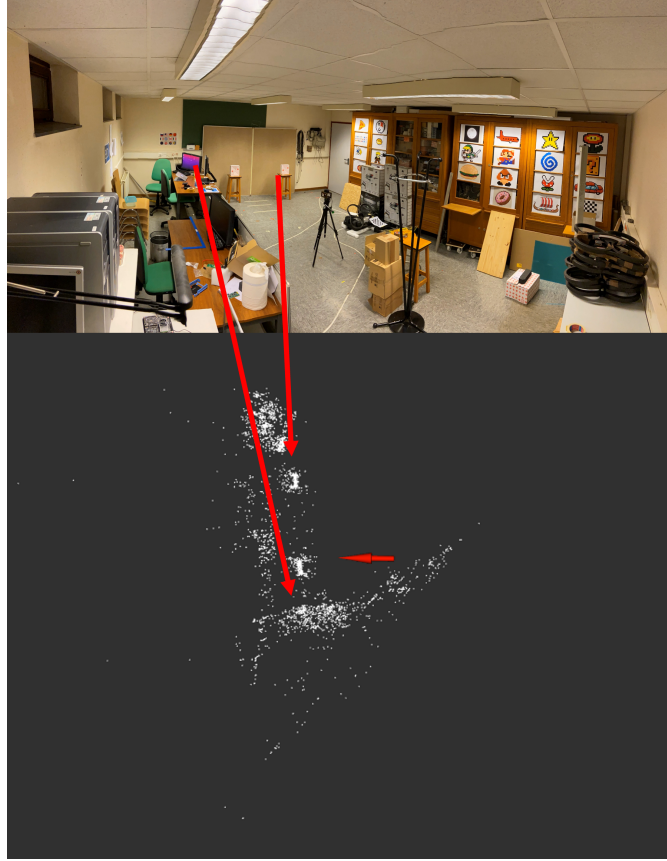


Figure 10.12: Generated map by ORB-SLAM2 after 3 passes with fixed camera angle

| Way-point        | A              | B             | C              | D               |
|------------------|----------------|---------------|----------------|-----------------|
| Truth (x, y) [m] | (0, 0)         | (1.5, 0)      | (0.411, 1.677) | (-0.389, 1.677) |
| #1               | (0, 0)         | (1.3, -0.21)  | (0.96, 1.85)   | (0.2, 2.24)     |
| #2               | (-0.0, 0.07)   | (1.56, -0.21) | (0.98, 1.86)   | (-0.4, 1.69)    |
| #3               | (-0.016, 0.07) | (1.53, -0.02) | (1.00, 1.86)   | (-0.44, 1.68)   |
| mean error [m]   | 0.047          | 0.23          | 0.6            | 0.3             |

Table 10.3: Results of ORB-SLAM2 monocular with oriented camera direction

### ORB-SLAM2 stereo

Here comes the results where we connected our RGB-D-like topics to ORB-SLAM2. Results are available on table [10.4](#).

And the generated map after the 3 passes is available on figure [10.15](#).

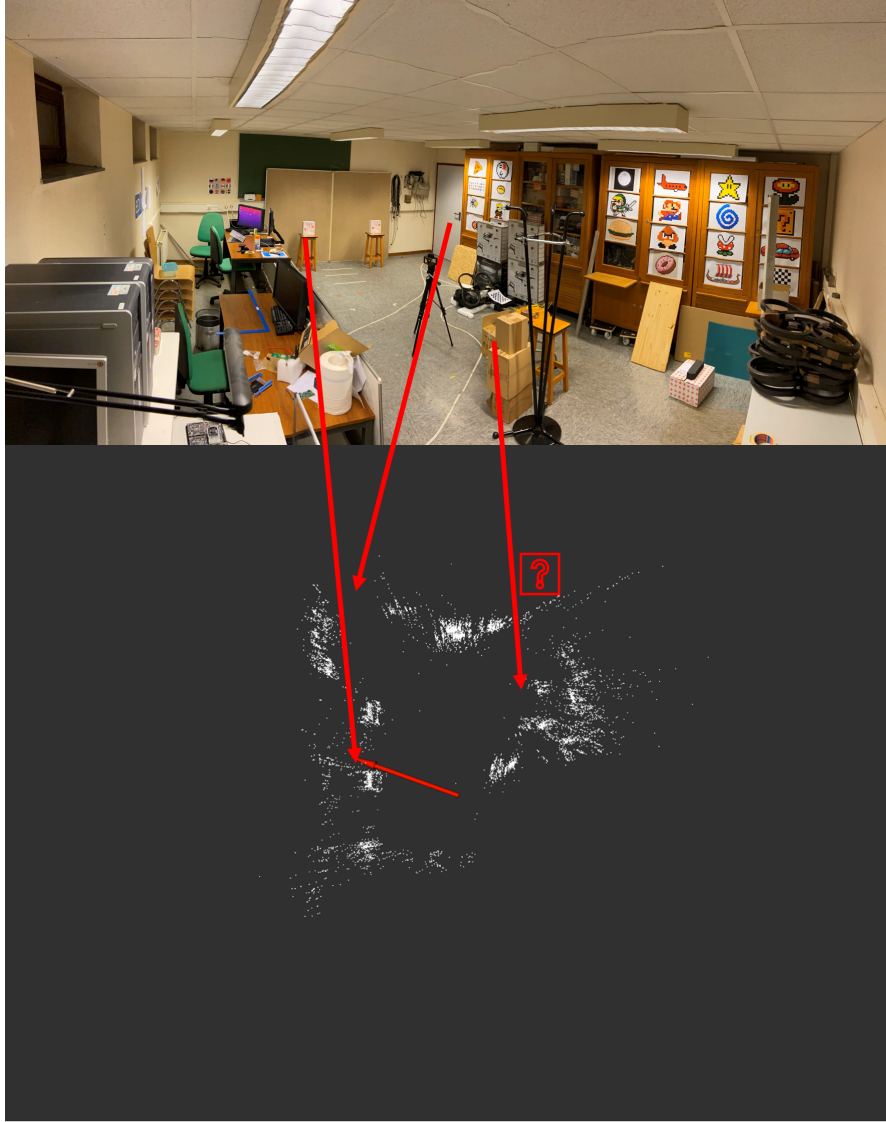
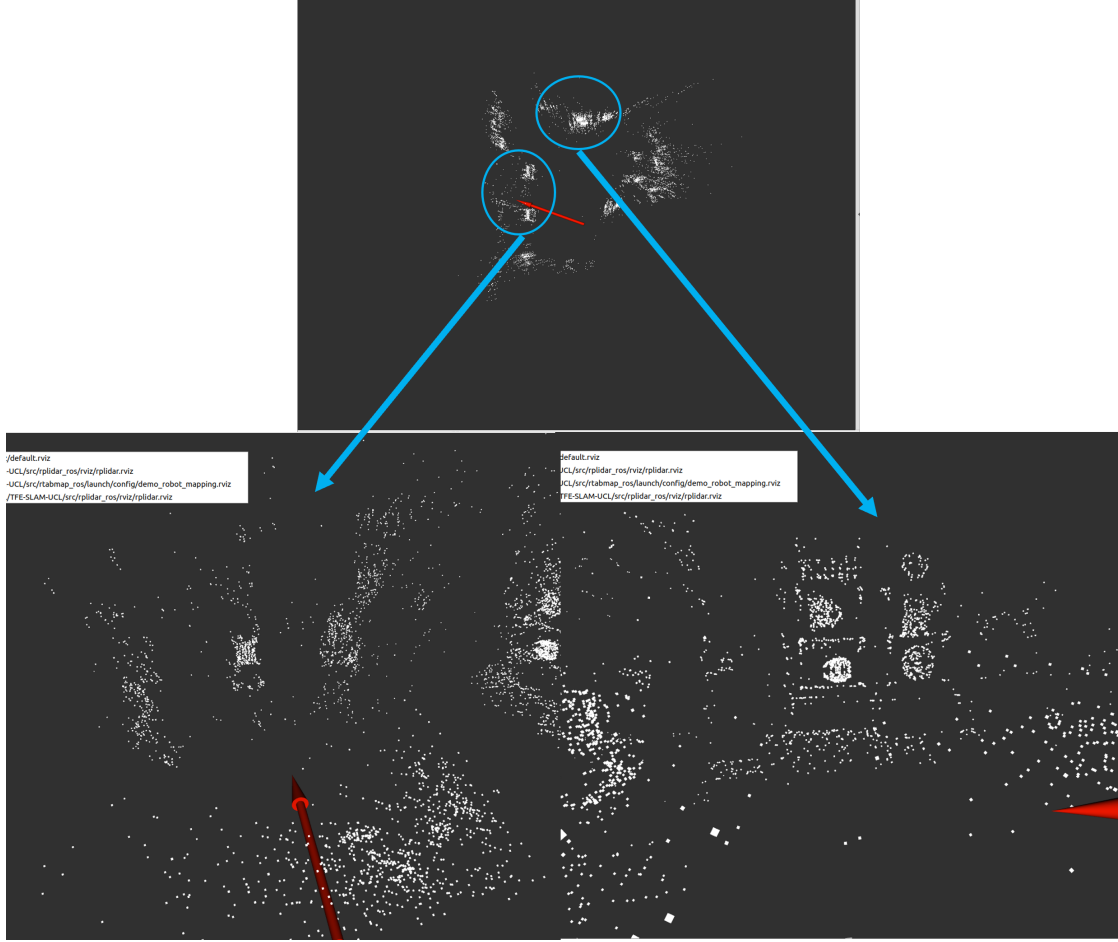


Figure 10.13: Generated map by ORB-SLAM2 after 3 passes with changing camera angle

| Way-point        | A             | B              | C              | D               |
|------------------|---------------|----------------|----------------|-----------------|
| Truth (x, y) [m] | (0, 0)        | (1.5, 0)       | (0.411, 1.677) | (-0.389, 1.677) |
| #1               | (0.24, 0.028) | (0.883, -0.95) | (1.13, 1.14)   | (0.65, 1.66)    |
| #2               | (-0.0, 0.00)  | (0.86, -1.0)   | (1.13, 1.38)   | (0.67, 1.66)    |
| #3               | (-0.0, 0.01)  | (0.84, -0.88)  | (1.13, 1.25)   | (0.64, 1.59)    |
| mean error [m]   | 0.08          | 0.14           | 0.84           | 1.04            |

Table 10.4: Results of ORB-SLAM2 monocular with oriented camera direction

Figure 10.14: Detail highlight of the map from figure [10.13](#)

## Discussion

Looking at the raw results, the mean errors of the positions generated by ORB-SLAM2 stereo are quite far from the real positions. In fact, when the maximum mean error of ORB-SLAM2 is 0.6 meters, the maximum error produced by ORB-SAM2 stereo is 1.04 meters. This shows that our solution is not ready yet for accurate mapping of the environment. One interesting observation when comparing ORB-SLAM2 mono with rotation and ORB-SLAM2 stereo is that the data from ORB-SLAM2 stereo are way more consistent between each other. The repeatability of the position measurements has thus been increased. This effect becomes obvious when looking at figure [10.17](#). We can see that the features from ORB-SLAM2 stereo are more segmented than those from ORB-SLAM2 mono. The algorithm gets the ability to place the features on a more relatively accurate way and it is especially true for far away points [69](#). The effect is visible on the right side of each 3D map. For the monocular version, far away points are roughly grouped together losing important information about the geometry of the room



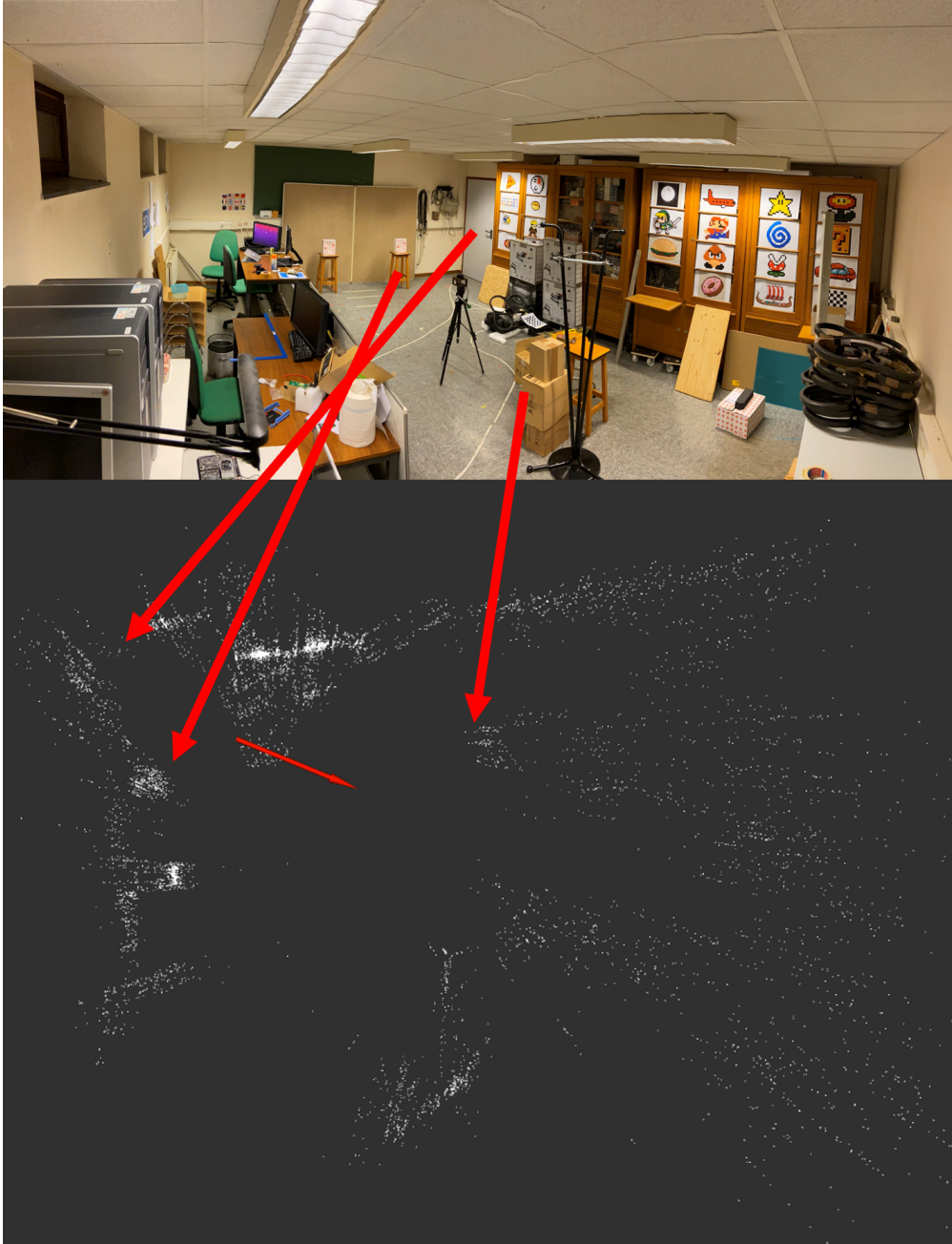


Figure 10.15: Generated map by ORB-SLAM2 after 3 passes with the depth correction node

when for the stereo version, we can clearly see on figure [10.18](#) two available paths for the robot to go.



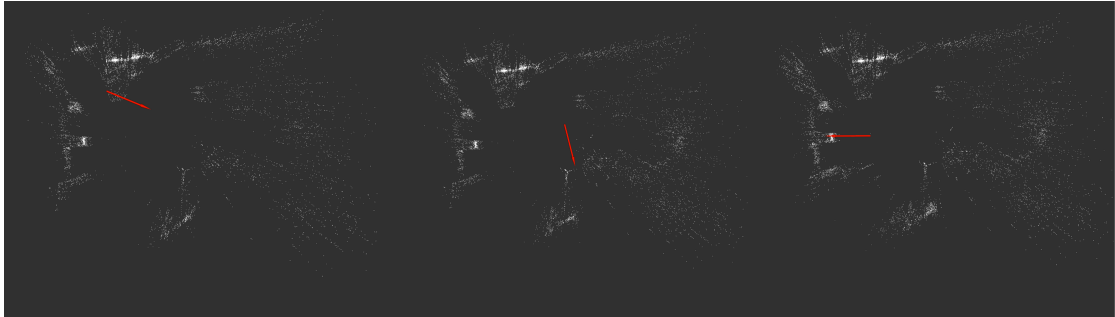


Figure 10.16: Locations in generated map with stereo ORB-SLAM2

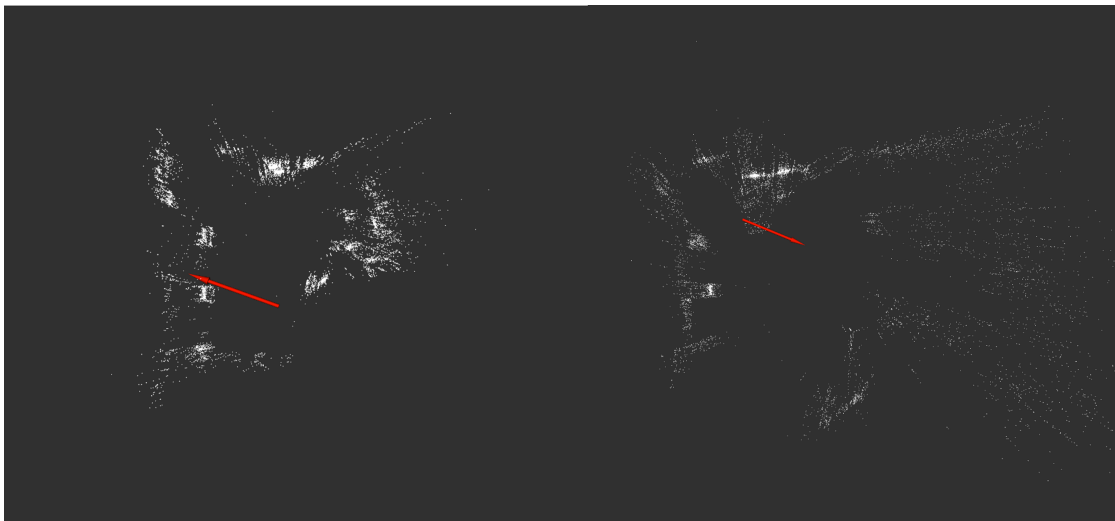


Figure 10.17: comparison of the two maps, ORB-SLAM mono left, ORB-SLAM stereo right.

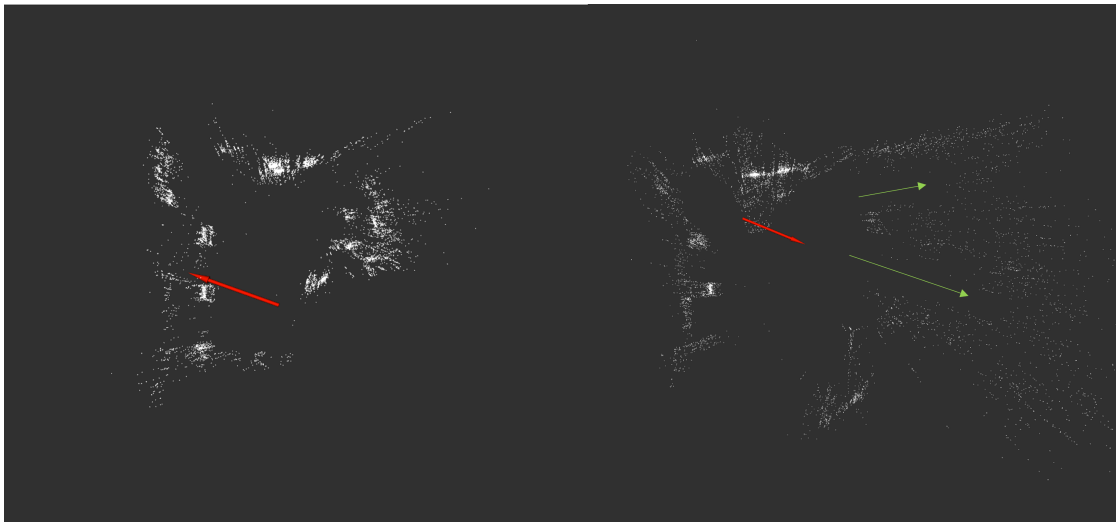


Figure 10.18: Highlight of lost information between mono and stereo ORB-SLAM2.

# 11 Going forward : Lines of investigation and future developments

In this chapter, we want to provide the reader with an overview of possible ways to build on this project and alternative implementation solutions. We will first discuss possibilities to improve the depth image creation and/or correction only based on the 2D scanning LiDAR. Secondly we will expose solutions making good use of additional sensors. Finally we will comment on complete/hybrid solutions and give our thoughts for interesting ways going forward.

## 11.1 Implementation based on 2D scanning LiDAR technology

Different projects across the globe investigate solutions to get 3D point clouds from 2D scanning LiDAR. Indeed, 2D LiDAR is advantageous in many applications due to its lighter weight, lower computational cost and sensor cost. Those characteristics are particularly important in many applications like the one this master's thesis is about: autonomous UAVs. In this section, we will explore a few of those solutions to sense information for low-cost LiDAR 3D point clouds.

### 11.1.1 Mechanical/hardware improvements

A first approach in order to get a 3D point clouds from a 2D scanning LiDAR relies on mechanical solutions. Some projects rely on placing the LiDAR sensor on a unidirectional rotating platform to give it a more degree of freedom (DOF) and scan 3D environment. Those solutions are usually referred as "tilting 2D Laser Range Finder (LRF)". The complexity of such a solution (mainly due the extrinsic calibration) is however far from trivial. In 2020, A. Khurana and K.S. Nagla published a paper regarding a 3D laser measurement system based on a 2D LRF (the Hokuyo UTM-30LX-EW) and a rotating platform (the PTU-E46) [44]. Their system is illustrated in Figure 11.1a.

In the same category, other research projects are based on a 2D LiDAR mounted onto a 1-DOF linear stage that moves along the vertical direction. As a result, the system can also perform vertical scanning within the motion range of the linear stage. An example of such implementation can be found in the paper written by Yang et al. in 2018 titled *A robust and accurate SLAM algorithm for omni-directional mobile robots based on a novel 2.5D lidar device* [99]. Their system is illustrated in Figure 11.1b.

Finally, another original perspective was the solution designed by M.A.Cooper in 2018. The solution explained in his paper, *Converting a 2D Scanning LiDAR to a 3D System for Use on Quad-Rotor UAVs in Autonomous Navigation*, does not rely on moving the LiDAR sensor itself but rather moving a specially designed platforms composed of various mirrors around the LiDAR. Those mirrors having different tilt angles can be used to get 3D information of the environment. However this solution is complex to replicate and the system is quite voluminous [22]. Their system is illustrated in Figure 11.1c.

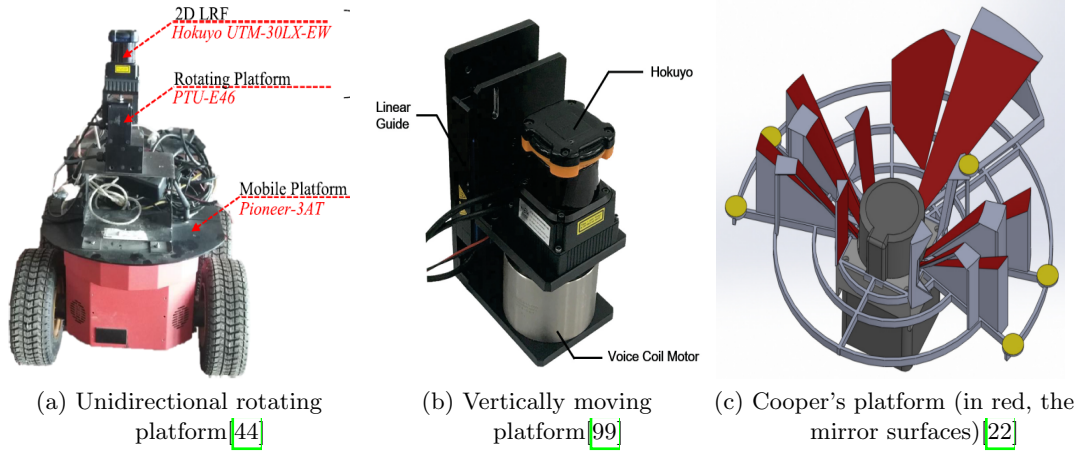


Figure 11.1: Examples of

### 11.1.2 Software based improvements

A second approach for solutions based on monocular camera and 2D scanning LiDAR is to rely on software to recover depth information about the 3D environment. The proposed system in this master's thesis is an example of this category of solutions. In our case we use an algorithm (the CNN network AdaBins) to get a first depth image from the monocular camera and then correct it with the LiDAR data. Another interesting way to proceed would be to use a CNN network that directly takes as input the monocular camera and LiDAR information to directly give a better depth image.

With the aim to provide better low-cost LiDAR for self-driving vehicles, a group of researchers from the Rochester Institute of Technology proposes a solution that uses a 2D LiDAR sensor and a camera to extend single beam LiDAR to full resolution. With this solution, companies using automated mobile robots can use a less cost prohibitive solution that also provides a comparatively higher resolution at a much faster rate, with the ability to determine per-pixel depth. In order to obtain information similar to that of a 3D LiDAR, but at a much-reduced cost, they implemented a Deep-Learning-based technique taking 2D LiDAR point array and stereo images as training inputs. After training, only one single image and one LiDAR beam can produce high-precision depth

maps in real applications [2]. Figure 11.2 illustrates the employed method.

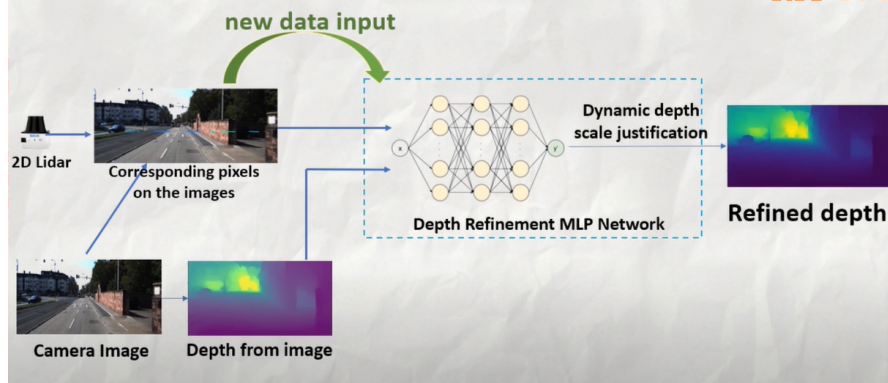


Figure 11.2: Overview of the principle used by Y. Lu et al. in [2]

Several convolutional neural networks based on those principles already have been implemented but none in open source, at least to our knowledge. As the reader could notice, our implementation was partially inspired by this type of implementation. However, as developing a deep learning algorithm to correct the depth image with the LiDAR data (combined with all of the others parts of this master’s thesis) was not feasible in that time frame.

We could also utilize object detectors, image segmentation or even 3D point clouds segmentation to improve as we have introduced in Chapter 8. However this is in itself a large topic and we will not cover all the possibilities in this section. Moreover, they are often used in combination with other methods to enhance the result those methods but not as the main solution. To give an example for a task that was previously executed by a LiDAR placed on a rotating platform like in [46], in 2020, D. Cha and W. Chung published a paper on 2D LiDAR solution for human-leg detection in 3D feature space using LiDAR scan segmentation, support data description and others detection classifier to compare [12].

We can also cite the work of Chen et al. [13] for their *Indoor Layout Estimation by 2D LiDAR and Camera Fusion* in which they use semantic segmentation, 2D similarity transformation estimation and solved by the recursive random sample consensus (R-RANSAC) algorithm. Their work was already introduced in the State-of-the-Art section and illustrated in Figure 2.10.

## 11.2 Improving the system with other sensors

Another way to improve on the system is by the use of other sensors. In this section, we will discuss either alternative or additional sensors.

### 11.2.1 3D LiDAR

Naturally, LiDAR technology is a fast evolving field. Depending on the development on the technology related to LiDAR sensors, some of the characteristics preventing 3D LiDAR sensors to be less relevant for UAVs applications could diminish : above all the price but also the weight and volume of the system. Indeed lots of implementation and research shows interesting results, especially in the field of self-driving vehicles [100] [84] [101] [38] [11]. If this is the case, 3D LiDAR could be a better starting point than the 2D scanning LiDAR. However such a shift should not happen in the very near future.

### 11.2.2 RGB-D camera

Another approach could consist of using a RGB-D camera instead of either the monocular camera or both the monocular camera and LiDAR. Indeed many research projects rely on RGB-D cameras for SLAM implementation. To cite a few examples, we can point to [3], [27], [86], [85].

However, it is important to remind a few of the disadvantages of RGB-D cameras : shorter distance range, a higher computational cost (compared to 2D LiDAR), narrow field of view, sensible to light conditions. Those characteristics can be limiting for indoor applications prevent the eventual extension to outdoor applications.

As introduced in Chapter 2, some research projects work on an adequate fusion of both RGB-D camera and 2D LiDAR sensor (like in 2021 [77], in 2020 with [67] or in 2019 with [98]) which are also implementations worth looking into.

### 11.2.3 Flash solid-state LiDAR

From their use in self-driving vehicles to their integration to the latest Iphone, Flash solid-state LiDAR sensors have become increasingly popular these few last years. Indeed the major problem of 2D scanning LiDAR (especially for high-end applications such as self-driving vehicles) is their moving mechanism which could cause problems with mass production of highly automated vehicles and the perspective of time synchronization of data and durability. More recent LiDAR sensors with non-moving part, solid-state LiDARs, could therefore replace the traditional scanning LiDAR. The technology is however not as developed yet as scanning LiDAR.

The cost of 3D flash Lidar are still too high to consider them for real self-driving vehicles, let alone autonomous drones. On the other hand, low-cost 2D flash LiDAR

are now available and could be use to directly have depth information for the pixels of the depth image (very similarly of a RGB-D camera). Nonetheless, replacing the 2D scanning LiDAR for a 2D flash LiDAR would deprive the system of some advantages such as the ones mentioned in chapter 9.

If the reader wants to know more about flash solid-state LiDAR, we recommend [53], [56] or [10]. Please note we do not recommend this solution but we believe that it was still worth mentioning.

### 11.3 Comments and conclusion

The objective of this chapter was not to exhaustively list all the possibilities for implementation or improvement on the current system. The aim was to layout an overview of the main paths going forward to provide the reader and next year students, a perspective on the possible way forward.

We discussed several existing approaches that could be interesting in the context of this long-term academic project. Naturally those solutions are not mutually exclusive. Most the interesting solutions combined some of the software improvements that we mentioned above combined with various mechanical/hardware fusion.

Our work was a first step, a foot in the door, of a sensor system which aims to be used for SLAM algorithms for autonomous UAVs. There is not one unique good solution and various research projects tries various implementations and combination of sensors.

According to us, one of the most promising solutions to investigate is enhancing the 3D point cloud / depth image creation thought adequate software. More precisely, either via a image segmentation method (as mentionned at the end of chapter 8) or via a deep learning algorithm that would compute, from the LiDAR scan and the camera image, a direct enhanced and coherent depth image. Such implementations could be highly cost effective and should easily be editable in the case of technology advances (such as cheaper 3D LiDAR, addition of RGB-D cameras or other sensors etc.).

In our system, the computational time of AdaBins prevents the system to be really efficient. We have already mentioned other methods to bypass AdaBins in Chapter 8. Another approach could be the use of a RGB-D camera to replace the estimation made by AdaBins. However using RGB-D camera would increase the overall cost of the sensor system and possibly limit its application (due to the inherent disadvantages of RGB-D cameras mentioned in the previous section). As in many of those proposed directions to go forward, it is sometimes a question on what trade off to do and what we prioritize : cost, accuracy, ranging distance, weight/volume, etc.

Also, algorithms that could recognize particular objects such as walls, doors and windows

could be of great use and improve both the creation of the depth image and/or the usability of the created map. Indeed many recent projects regarding autonomous vehicles are highly based on 3D object detection. We highly recommend the reader to look at the following papers : [49], [81] and [78]. Even-though those systems are designed for 3D scanning LiDAR and for self-driving cars, the underlying principles are still applicable to applications like ours.

Finally, developing a mechanism for the 2D scanning LiDAR to sense more points at different heights in the depth image could also be interesting to enhance the corrected depth image.

With all this said and in order to summarize, the main limitation of this work is the computational time of the AdaBins Node. We can highlight two main ways to resolve the issue :

- A first approach (cfr ending of chapter 8)
  - Only take the depths sensed by the LiDAR to create the image depth
  - Use an segmentation image or object detection algorithm to set areas of pixels where those depths perceived by the LiDAR can be applicable
  - Make the drone move at different height to gather the unknown depths or create a mechanism for the LiDAR to sense at different heights.
- A second approach
  - Implementing a deep learning algorithm that directly takes as input the monocular camera and LiDAR information and return a depth image



## 12 Conclusion

This thesis aimed to change the paradigm of drone sensors used by the students of previous years by adding a LiDAR. Then to see if this could be applied to a SLAM algorithm and to study its potential advantages.

The process was not easy and the learning curve was steep. We eventually succeeded in some of our objectives and we can clearly say that we did reached the following targets:

- The selection of an adequate LiDAR sensor for our application
- The creation of a mobile platform dedicated to sensor testing that improves the modularity of the whole system
- The mastering of ROS environment by linking our platform to a ROS network
- Practicing image segmentation for monocular depth recovery
- The fusion of the camera feed and the LiDAR scans
- Establishing depth images and 3D point clouds based on the monocular camera and 2D scanning LiDAR
- Mentioning directions for later implementations/improvements

We also had mitigated results in some of our goals which are the following:

- Wrapping the "depth image creation" algorithm into an efficient ROS node, making it fast enough to be able to work in real-time.
- Comparing the performance of multiple SLAM algorithms with our implementation of the sensor fusion.

## Future relevant directions

Going forward two main areas has to be improved : the creation of the 3D point cloud/ the image depth and the implementation of the visual SLAM algorithm (if one wants to use a homemade version/different from the available state-of-the-art algorithms).

Regarding the possible improvements on the 3D point clouds, we dedicated Chapter [11](#) to dress an overview of the interesting directions to pursue.

Regarding the practical implementation of an adequate SLAM algorithm, a more in depth study of the application of our method needs to be done. Indeed, we only had unsuccessful attempts while experimenting with a SLAM algorithm that takes both 2D scans and dense depth information as input (RTABMap), particularly due to the computational time of the AdaBins node (as already mentioned). Leading to a lack of relevant measurements with this SLAM method. On the other hand, we showed that a simple RGB-D SLAM (ORB-SLAM2) could already benefit from our setup, utilizing the obtained depth images from our sensor system.

Finally, please note that a dedicated GitHub with appropriate read.me files is available at : [Github](https://github.com/miniferretti/TFE-SLAM-UCL.git) (<https://github.com/miniferretti/TFE-SLAM-UCL.git>)



# Bibliography

- [1] O. 2. documentation : calib3d. (). "Camera calibration and 3d reconstruction," [Online]. Available: [https://docs.opencv.org/2.4/modules/calib3d/doc/camera\\_calibration\\_and\\_3d\\_reconstruction.html?fbclid=IwAR1YvyL7iVcSEw7W0g\\_qPRj8cJ7f7omBFvp1Ff1iwFNU-hV74ZFF-yysdyA#camera-calibration-and-3d-reconstruction](https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?fbclid=IwAR1YvyL7iVcSEw7W0g_qPRj8cJ7f7omBFvp1Ff1iwFNU-hV74ZFF-yysdyA#camera-calibration-and-3d-reconstruction). (accessed: 08.04.2021).
- [2] S. (USA). (Jul. 2020). "Using 2d lidar with a camera in 3d lidar applications," [Online]. Available: <https://sickusablog.com/using-2d-lidar-camera-3d-lidar-applications/>. (accessed: 08.10.2021).
- [3] W. Aguilar, G. Rodríguez, L. Álvarez, S. Sandoval, F. Quisaguano Paredes and A. Limaico, "Visual slam with a rgb-d camera on a quadrotor uav using on-board processing," May 2017, pp. 596–606, ISBN: 978-3-319-59146-9. DOI: [10.1007/978-3-319-59147-6\\_51](https://doi.org/10.1007/978-3-319-59147-6_51).
- [4] X. W. et al, "Iop conference series: Earth and environmental science : The evolution of lidar and its application in high precision measurement," *Earth and Environmental Science*, vol. Sci. 502 012008, pp. 1–12, 2020. [Online]. Available: <https://medium.com/analytics-vidhya/how-does-a-self-driving-vehicle-see-using-lidar-49f569ededf2>.
- [5] I. Alhashim and P. Wonka, *High quality monocular depth estimation via transfer learning*, 2019. arXiv: [1812.11941 \[cs.CV\]](https://arxiv.org/abs/1812.11941).
- [6] A. Bava. (). "Drone mapping saves time and reduces costs for landscape architects," [Online]. Available: <https://medium.com/aerial-acuity/drone-mapping-saves-time-and-reduces-costs-for-landscape-architects-ed337c204a69>. (accessed: 027.07.2021).
- [7] H. Bay, T. Tuytelaars and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof and A. Pinz, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417, ISBN: 978-3-540-33833-8.
- [8] R. Bhadani, "How does a self-driving vehicle see using lidar?," Dec. 2020. [Online]. Available: <https://medium.com/analytics-vidhya/how-does-a-self-driving-vehicle-see-using-lidar-49f569ededf2>, (accessed: 07.29.2021).
- [9] S. F. Bhat, I. Alhashim and P. Wonka, *Adabins: Depth estimation using adaptive bins*, 2020. arXiv: [2011.14141 \[cs.CV\]](https://arxiv.org/abs/2011.14141).

## Bibliography

- [10] K. C. J. M. S. C. H. Jang C. S. Kim, “Design factor optimization of 3D flash lidar sensor based on geometrical model for automated vehicle and advanced driver assistance system applications,” *International Journal of Automotive Technology* volume, vol. 18, 2017, pp. 147–156. [Online]. Available: <https://doi.org/10.1007/s12239-017-0015-7>.
- [11] I. Caminal, J. R. Casas and S. Royo, “Slam-based 3d outdoor reconstructions from lidar data,” in *2018 International Conference on 3D Immersion (IC3D)*, 2018, pp. 1–8. DOI: [10.1109/IC3D.2018.8657869](https://doi.org/10.1109/IC3D.2018.8657869).
- [12] D. Cha and W. Chung, “Human-leg detection in 3d feature space for a person-following mobile robot using 2d lidars,” *International Journal of Precision Engineering and Manufacturing*, vol. 21, Apr. 2020. DOI: [10.1007/s12541-020-00343-7](https://doi.org/10.1007/s12541-020-00343-7).
- [13] X. Chen, H. Zhang, H. Lu, J. Xiao, Q. Qiu and Y. Li, “Robust slam system based on monocular vision and lidar for robotic urban search and rescue,” in *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, 2017, pp. 41–47. DOI: [10.1109/SSRR.2017.8088138](https://doi.org/10.1109/SSRR.2017.8088138).
- [14] S. ( S. Co), “Low cost 360 degree laser range scanner : Development kit user manual (rev.1.3),” Apr. 2021, pp. 1–17.
- [15] —, *Low cost 360 degree laser range scanner : Introduction and datasheet (rev.1.9)*, Apr. 2021.
- [16] —, (). “Rplidar a3,” [Online]. Available: <https://www.slamtec.com/en/Lidar/A3Spec>. (accessed: 07.28.2021).
- [17] —, “Rplidar s1 : Introduction and data sheet,” Mar. 2021, pp. 1–17.
- [18] —, (). “Rplidar<sub>ros</sub> : Rosnodefortherplidar,” [Online]. Available: [https://github.com/slamtec/rplidar\\_ros](https://github.com/slamtec/rplidar_ros). (accessed: 07.29.2021).
- [19] —, “Slamtec mapper : Introduction and datasheet (laser mapping sensor : Model: M1m1),” Nov. 2019, pp. 1–16.
- [20] —, “Slamtec mapper : Introduction and datasheet (laser mapping sensor : Model: M2m2),” Jul. 2021, pp. 1–15.
- [21] —, “Slamware robostudio : User manual,” Nov. 2017, pp. 1–37.
- [22] M. Cooper, J. Raquet and R. Patton, “Algorithm on converting a 2d scanning lidar to 3d for use in autonomous indoor navigation,” *Advances in Robotics and Automation*, vol. 07, Jan. 2018. DOI: [10.4172/2168-9695.1000184](https://doi.org/10.4172/2168-9695.1000184).
- [23] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, vol. 1, 2005, 886–893 vol. 1. DOI: [10.1109/CVPR.2005.1177](https://doi.org/10.1109/CVPR.2005.1177).
- [24] P. Daponte, “A review on the use of drones for precision agriculture,” *IOP Conference Series: Earth and Environmental Science*, 2019. DOI: <https://iopscience.iop.org/article/10.1088/1755-1315/275/1/012022/pdf>.

## Bibliography

- [25] D. De Silva, J. Roche and A. Kondo, “Fusion of lidar and camera sensor data for environment sensing in driverless vehicles,” Oct. 2017.
- [26] C. Debeunne and D. Vivet, “A review of visual-lidar fusion based simultaneous localization and mapping,” *Sensors*, vol. 20, p. 2068, Apr. 2020. DOI: [10.3390/s20072068](https://doi.org/10.3390/s20072068).
- [27] A. Deris, I. Trigonis, A. Aravanis and E. Stathopoulou, “Depth cameras on uavs: A first approach,” *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLII-2/W3, pp. 231–236, Feb. 2017. DOI: [10.5194/isprs-archives-XLII-2-W3-231-2017](https://doi.org/10.5194/isprs-archives-XLII-2-W3-231-2017).
- [28] C. Dubois. (). “What is solid state lidar and is it faster, cheaper, better?” [Online]. Available: <https://www.allaboutcircuits.com/news/solid-state-lidar-faster-cheaper-better/>. (accessed: 027.07.2021).
- [29] J. Engel, V. Koltun and D. Cremers, *Direct sparse odometry*, 2016. arXiv: [1607.02565](https://arxiv.org/abs/1607.02565) [cs.CV].
- [30] J. Engel, T. Schöps and D. Cremers, “Lsd-slam: Large-scale direct monocular slam,” in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele and T. Tuytelaars, Eds., Cham: Springer International Publishing, 2014, pp. 834–849, ISBN: 978-3-319-10605-2.
- [31] Espressif. (). “Esp32,” [Online]. Available: <https://www.espressif.com/en/products/socs/esp32>. (accessed: 07.27.2021).
- [32] M. Filipenko and I. Afanasyev, “Comparison of various slam systems for mobile robot in an indoor environment,” in *2018 International Conference on Intelligent Systems (IS)*, 2018, pp. 400–407. DOI: [10.1109/IS.2018.8710464](https://doi.org/10.1109/IS.2018.8710464).
- [33] R. P. foundation. (). “Camera module 2,” [Online]. Available: <https://www.raspberrypi.org/products/camera-module-v2/>. (accessed: 07.27.2021).
- [34] —, (). “Raspberry pi 4-b,” [Online]. Available: <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/>. (accessed: 07.27.2021).
- [35] F. [Gérardy Julien [UCL] Schiltz, “Suivi d’une cible mobile par des drones autonomes,” 2016. [Online]. Available: <https://dial.uclouvain.be/memoire/ucl/object/thesis:6743>.
- [36] G. Grisetti, C. Stachniss and W. Burgard, “Improved techniques for grid mapping with rao-blackwellized particle filters,” *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007. DOI: [10.1109/TR0.2006.889486](https://doi.org/10.1109/TR0.2006.889486).
- [37] W. Hess, D. Kohler, H. Rapp and D. Andor, “Real-time loop closure in 2d lidar slam,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1271–1278. DOI: [10.1109/ICRA.2016.7487258](https://doi.org/10.1109/ICRA.2016.7487258).
- [38] M. Himmelsbach, A. Müller, T. Luettel and H.-J. Wuensche, “Lidar-based 3d object perception,” Oct. 2008.
- [39] <https://www.flyability.com/elios-2>. (). “Elios 2 intuitive indoor inspection,” [Online]. Available: <https://www.flyability.com/elios-2>. (accessed: 07.27.2021).

## Bibliography

- [40] Intel. (). “Beginners guide to depth,” [Online]. Available: <https://www.intelrealsense.com/beginners-guide-to-depth/>. (accessed: 027.07.2021).
- [41] *Introduction to surf (speeded-up robust features)*, <https://medium.com/data-breach/introduction-to-surf-speeded-up-robust-features-c7396d6e7c4e>, Accessed: 2021-08-11.
- [42] G. Jiang, L. Yin, S. Jin, C. Tian, X. Ma and Y. Ou, “A simultaneous localization and mapping (slam) framework for 2.5d map building based on low-cost lidar and vision fusion,” *Applied Sciences*, vol. 9, no. 10, 2019, ISSN: 2076-3417. DOI: [10.3390/app9102105](https://doi.org/10.3390/app9102105). [Online]. Available: <https://www.mdpi.com/2076-3417/9/10/2105>.
- [43] L. Kechtban, “Toward an optimized 3d monocular slam,” Anglais, Ph.D. dissertation, UCL - Ecole polytechnique de Louvain, 2018. [Online]. Available: <http://hdl.handle.net/2078.1/thesis:17191>.
- [44] A. Khurana and K. Nagla, “An improved method for extrinsic calibration of tilting 2d lrf,” *Journal of Intelligent Robotic Systems*, vol. 99, Sep. 2020. DOI: [10.1007/s10846-020-01147-7](https://doi.org/10.1007/s10846-020-01147-7).
- [45] D. Kim and S. Lee, “Advances in 3d camera: Time-of-flight vs. active triangulation,” in *Intelligent Autonomous Systems 12: Volume 1 Proceedings of the 12th International Conference IAS-12, held June 26-29, 2012, Jeju Island, Korea*, S. Lee, H. Cho, K.-J. Yoon and J. Lee, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 301–309, ISBN: 978-3-642-33926-4. DOI: [10.1007/978-3-642-33926-4\\_28](https://doi.org/10.1007/978-3-642-33926-4_28). [Online]. Available: [https://doi.org/10.1007/978-3-642-33926-4\\_28](https://doi.org/10.1007/978-3-642-33926-4_28).
- [46] J. Kim, H. Jeong and D. Lee, “Single 2d lidar based follow-me of mobile robot on hilly terrains,” *Journal of Mechanical Science and Technology*, vol. 34, pp. 1–10, Sep. 2020. DOI: [10.1007/s12206-020-0835-7](https://doi.org/10.1007/s12206-020-0835-7).
- [47] G. Klein and D. Murray, “Parallel tracking and mapping for small ar workspaces,” in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2007, pp. 225–234. DOI: [10.1109/ISMAR.2007.4538852](https://doi.org/10.1109/ISMAR.2007.4538852).
- [48] S. Kohlbrecher, J. Meyer, O. von Stryk and U. Klingauf, “A flexible and scalable slam system with full 3d motion estimation,” in *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, IEEE, Nov. 2011.
- [49] G. A. Kumar, J. H. Lee, J. Hwang, J. Park, S. H. Youn and S. Kwon, “Lidar and camera fusion approach for object distance estimation in self-driving vehicles,” *Symmetry*, vol. 12, no. 2, 2020, ISSN: 2073-8994. DOI: [10.3390/sym12020324](https://doi.org/10.3390/sym12020324). [Online]. Available: <https://www.mdpi.com/2073-8994/12/2/324>.
- [50] S. V. lab. (2020). “Imagenet database,” [Online]. Available: <https://www.image-net.org/>.

- [51] M. Labbé and F. Michaud, “Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation,” *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019. DOI: <https://doi.org/10.1002/rob.21831>, eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21831>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21831>.
- [52] S. Lee, D. Har and D. Kum, “Drone-assisted disaster management: Finding victims via infrared camera and lidar sensor fusion,” in *2016 3rd Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE)*, 2016, pp. 84–89. DOI: [10.1109/APWC-on-CSE.2016.025](https://doi.org/10.1109/APWC-on-CSE.2016.025).
- [53] J. Lemmetti, N. Sorri, I. Kallioniemi, P. Melanen and P. Uusimaa, “Long-range all-solid-state flash LiDAR sensor for autonomous driving,” in *High-Power Diode Laser Technology XIX*, M. S. Zediker, Ed., International Society for Optics and Photonics, vol. 11668, SPIE, 2021, pp. 99–105. [Online]. Available: <https://doi.org/10.1117/12.2578769>.
- [54] J. Li and R. L. Stevenson, *Indoor layout estimation by 2d lidar and camera fusion*, 2020. arXiv: [2001.05422 \[cs.CV\]](https://arxiv.org/abs/2001.05422).
- [55] V. Lidar, “A guide to lidar wavelengths for autonomous vehicles and driver assistance,” *Velodyne Lidar*, Oct. 2018. [Online]. Available: [https://velodynelidar.com/blog/guide-to-lidar-wavelengths/?utm\\_source=rss&utm\\_medium=rss&utm\\_campaign=guide-to-lidar-wavelengths](https://velodynelidar.com/blog/guide-to-lidar-wavelengths/?utm_source=rss&utm_medium=rss&utm_campaign=guide-to-lidar-wavelengths), accessed: 07.29.2021.
- [56] J. Liu, Q. Sun, Z. Fan and Y. Jia, “Tof lidar development in autonomous vehicle,” in *2018 IEEE 3rd Optoelectronics Global Conference (OGC)*, 2018, pp. 185–190. DOI: [10.1109/OGC.2018.8529992](https://doi.org/10.1109/OGC.2018.8529992).
- [57] E. López, S. García, R. Barea, L. M. Bergasa, E. J. Molinos, R. Arroyo, E. Romera and S. Pardo, “A multi-sensorial simultaneous localization and mapping (slam) system for low-cost micro aerial vehicles in gps-denied environments,” *Sensors*, vol. 17, no. 4, 2017, ISSN: 1424-8220. DOI: [10.3390/s17040802](https://doi.org/10.3390/s17040802). [Online]. Available: <https://www.mdpi.com/1424-8220/17/4/802>.
- [58] D. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol. 60, pp. 91–, Nov. 2004. DOI: [10.1023/B:VISI.0000029664.99615.94](https://doi.org/10.1023/B:VISI.0000029664.99615.94).
- [59] N. MADALI, “Light detection and ranging (lidar),” Jul. 2020. [Online]. Available: <https://towardsdatascience.com/light-detection-and-ranging-lidar-8f22971eeaea>, accessed: 07.29.2021.
- [60] Q. S. Mahmoud Hamandi Federico Usai, “Design of multirotor aerial vehicles: A taxonomy based on input allocation,” *Archives-Ouvertes*, 2021. DOI: <https://hal.archives-ouvertes.fr/hal-02433405/document>.
- [61] E. Mair, G. Hager, D. Burschka, M. Suppa and G. Hirzinger, “Adaptive and generic corner detection based on the accelerated segment test,” Sep. 2010, ISBN: 978-3-642-15551-2. DOI: [10.1007/978-3-642-15552-9\\_14](https://doi.org/10.1007/978-3-642-15552-9_14).



## Bibliography

- [62] MarketsandMarkets. (). “Unmanned aerial vehicle (uav) market by point of sale, systems, platform (civil commercial, and defense government), function, end use, application, type, mode of operation, mtow, range, and region - global forecast to 2026,” [Online]. Available: <https://www.marketsandmarkets.com/Market-Reports/unmanned-aerial-vehicles-uav-market-662.html>. (accessed: 08.07.2021).
- [63] MathWorks. (). “What is lidar camera calibration?” [Online]. Available: <https://www.mathworks.com/help/lidar/ug/lidar-camera-calibration.html>. (accessed: 08.04.2021).
- [64] B. N. Matteo Ferretti, *Tfe-slam-ucl, lidar-camera fusion*, <https://github.com/miniferretti/TFE-SLAM-UCL>, 2021.
- [65] P. F. McManamon, *LiDAR Technologies and Systems*. Bellingham, Washington: SPIE, 2019.
- [66] *Mpu-6050 six-axis (gyro + accelerometer) mems motiontracking™ devices*, MPU-6050, TDK Invensense, 2015. [Online]. Available: <https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/>.
- [67] L. Mu, P. Yao, Y. Zheng, K. Chen, F. Wang and N. Qi, “Research on slam algorithm of mobile robot based on the fusion of 2d lidar and depth camera,” *IEEE Access*, vol. 8, pp. 157 628–157 642, 2020. DOI: [10.1109/ACCESS.2020.3019659](https://doi.org/10.1109/ACCESS.2020.3019659).
- [68] R. Mur-Artal and J. D. Tardos, “Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras,” *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017, ISSN: 1941-0468. DOI: [10.1109/tro.2017.2705103](https://doi.org/10.1109/tro.2017.2705103). [Online]. Available: <http://dx.doi.org/10.1109/TR0.2017.2705103>.
- [69] R. Mur-Artal and J. D. Tardós, “ORB-SLAM2: An open-source SLAM system for monocular, stereo and RGB-D cameras,” *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017. DOI: [10.1109/TR0.2017.2705103](https://doi.org/10.1109/TR0.2017.2705103).
- [70] T. Neff. (). “The laser that’s changing the worldextras: The laser that’s changing the worldphotos and video: The laser that’s changing...,” [Online]. Available: <https://todddneff.com/books/lidarhistory/extras/photos-and-video-the-laser-thats-changing-the-world/ancient-history/>. (accessed: 07.29.2021).
- [71] R. A. Newcombe, S. J. Lovegrove and A. J. Davison, “Dtam: Dense tracking and mapping in real-time,” in *2011 International Conference on Computer Vision*, 2011, pp. 2320–2327. DOI: [10.1109/ICCV.2011.6126513](https://doi.org/10.1109/ICCV.2011.6126513).
- [72] P. Newman and K. Ho, “Slam-loop closing with visually salient features,” *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 635–642, 2005.
- [73] Nvidia. (). “Raspberry pi 4-b,” [Online]. Available: <https://www.nvidia.com/fr-fr/autonomous-machines/embedded-systems/jetson-nano/>. (accessed: 07.27.2021).

## Bibliography

- [74] D. K. Petar Piljek and M. Krznar, “Method for characterization of a multirotor uav electric propulsion system,” *Applied Sciences*, 2020.
- [75] J. Quenzel, R. A. Rosu, T. Läbe, C. Stachniss and S. Behnke, *Beyond photometric consistency: Gradient-based dissimilarity for improving visual odometry and stereo matching*, 2020. arXiv: [2004.04090 \[cs.CV\]](https://arxiv.org/abs/2004.04090).
- [76] *Rascpicam<sub>n</sub>ode*, [https://github.com/Slamtec/rplidar\\_ros](https://github.com/Slamtec/rplidar_ros), Accessed: 2021-08-4.
- [77] V. A. Rosas-Cervantes, Q.-D. Hoang, S.-G. Lee and J.-H. Choi, “Multi-robot 2.5d localization and mapping using a monte carlo algorithm on a multi-level surface,” *Sensors*, vol. 21, no. 13, 2021, ISSN: 1424-8220. DOI: [10.3390/s21134588](https://doi.org/10.3390/s21134588). [Online]. Available: <https://www.mdpi.com/1424-8220/21/13/4588>.
- [78] A. Rövid, V. Remeli and Z. Szalay, “Raw fusion of camera and sparse lidar for detecting distant objects:” *at - Automatisierungstechnik*, vol. 68, no. 5, pp. 337–346, 2020. DOI: [doi : 10.1515/auto-2019-0086](https://doi.org/10.1515/auto-2019-0086). [Online]. Available: <https://doi.org/10.1515/auto-2019-0086>.
- [79] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, “Orb: An efficient alternative to sift or surf,” in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571. DOI: [10.1109/ICCV.2011.6126544](https://doi.org/10.1109/ICCV.2011.6126544).
- [80] R. F. Salas-Moreno, B. Glocken, P. H. J. Kelly and A. J. Davison, “Dense planar slam,” in *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2014, pp. 157–164. DOI: [10.1109/ISMAR.2014.6948422](https://doi.org/10.1109/ISMAR.2014.6948422).
- [81] M. Sanatkar. (Jun. 2020). “Lidar 3d object detection methods,” [Online]. Available: <https://towardsdatascience.com/lidar-3d-object-detection-methods-f34cf3227aea>. (accessed: 08.10.2021).
- [82] B. SCHWEBER, “Lidar and time of flight,” Dec. 2019. [Online]. Available: <https://www.microcontrollertips.com/lidar-and-time-of-flight-part-2-operation/>, accessed: 07.29.2021.
- [83] Y. Seo and C.-C. Chou, “A tight coupling of vision-lidar measurements for an effective odometry,” in *2019 IEEE Intelligent Vehicles Symposium (IV)*, 2019, pp. 1118–1123. DOI: [10.1109/IVS.2019.8814164](https://doi.org/10.1109/IVS.2019.8814164).
- [84] Y.-S. Shin, Y. Park and A. Kim, “Direct visual slam using sparse depth for camera-lidar system,” May 2018, pp. 1–8. DOI: [10.1109/ICRA.2018.8461102](https://doi.org/10.1109/ICRA.2018.8461102).
- [85] P. Stegagno, M. Basile, H. H. Bühlhoff and A. Franchi, “Vision-based autonomous control of a quadrotor uav using an onboard rgb-d camera and its application to haptic teleoperation,” *IFAC Proceedings Volumes*, vol. 46, no. 30, pp. 87–92, 2013, 2nd IFAC Workshop on Research, Education and Development of Unmanned Aerial Systems, ISSN: 1474-6670. DOI: <https://doi.org/10.3182/20131120-3-FR-4045.00016>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1474667015402769>.

- [86] R. TaoZhang, “Uav 3d mapping with rgb-d camera,” in *2017 Chinese Automation Congress (CAC)*, 2017, pp. 2727–2731. DOI: [10.1109/CAC.2017.8243239](https://doi.org/10.1109/CAC.2017.8243239).
- [87] K. Tateno, F. Tombari, I. Laina and N. Navab, *Cnn-slam: Real-time dense monocular slam with learned depth prediction*, 2017. arXiv: [1704.03489 \[cs.CV\]](https://arxiv.org/abs/1704.03489).
- [88] Tentone. (2020). “Monodepth,” [Online]. Available: <https://github.com/tentone/monodepth>.
- [89] E. Tola, V. Lepetit and P. Fua, “Daisy: An efficient dense descriptor applied to wide-baseline stereo,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 5, pp. 815–830, 2010. DOI: [10.1109/TPAMI.2009.77](https://doi.org/10.1109/TPAMI.2009.77).
- [90] M. Trajkovic and M. Hedley, “Fast corner detection,” *Image and Vision Computing*, vol. 16, pp. 75–87, Feb. 1998. DOI: [10.1016/S0262-8856\(97\)00056-5](https://doi.org/10.1016/S0262-8856(97)00056-5).
- [91] “Triangular ranging radar and tof lidar,” *Programmer Sought*, [Online]. Available: <https://www.programmersought.com/article/31885267576/>, accessed: 07.29.2021.
- [92] D. Tyagi. (2019). “Introduction to orb (oriented fast and rotated brief),” [Online]. Available: <https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf> (visited on 13/08/2021).
- [93] Wikipédia, *Scale-invariant feature transform — wikipédia, l’encyclopédie libre*, [En ligne; Page disponible le 19-décembre-2020], 2020. [Online]. Available: [http://fr.wikipedia.org/w/index.php?title=Scale-invariant\\_feature\\_transform&oldid=177813849](http://fr.wikipedia.org/w/index.php?title=Scale-invariant_feature_transform&oldid=177813849).
- [94] Wikipedia. (). “Lidar,” [Online]. Available: [https://en.wikipedia.org/wiki/Unmanned\\_aerial\\_vehicle](https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle), (accessed: 07.27.2021).
- [95] —, (). “Unmanned aerial vehicle,” [Online]. Available: [https://en.wikipedia.org/wiki/Unmanned\\_aerial\\_vehicle](https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle), (accessed: 027.07.2021).
- [96] C. Woodford, “Lidar,” *ExplainThatStuff*, vol. Communication, pp. 1–12, Last updated: July 10, 2021. [Online]. Available: <https://www.explainthatstuff.com/lidar.html>, accessed: 07.29.2021.
- [97] E. Wu, “All you need to know about slamtec rplidar, mapper and slamware,” *SeedStudio*, 2019. [Online]. Available: <https://www.seeedstudio.com/blog/2019/08/05/all-you-need-to-know-about-slamtec-rplidar-mapper-and-slamware/>, last accessed: 08.02.2021.
- [98] Y. Xu, Y. Ou and T. Xu, “Slam of robot based on the fusion of vision and lidar,” in *2018 IEEE International Conference on Cyborg and Bionic Systems (CBS)*, 2018, pp. 121–126. DOI: [10.1109/CBS.2018.8612212](https://doi.org/10.1109/CBS.2018.8612212).
- [99] Y. Yang, G. Yang, Y. Tian, T. Zheng, L. Li and Z. Wang, “A robust and accurate slam algorithm for omni-directional mobile robots based on a novel 2.5d lidar device,” in *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2018, pp. 2123–2127. DOI: [10.1109/ICIEA.2018.8398060](https://doi.org/10.1109/ICIEA.2018.8398060).

## Bibliography

- [100] K. Yoneda, R. Yanase, M. Aldibaja, N. Suganuma and K. Sato, “Mono-camera based vehicle localization using lidar intensity map for automated driving,” *Artificial Life and Robotics*, vol. 24, Oct. 2018. DOI: [10.1007/s10015-018-0502-6](https://doi.org/10.1007/s10015-018-0502-6).
- [101] B. Zhou, Y. He, K. Qian, X. Ma and X. Li, “S4-slam: A real-time 3d lidar slam system for ground/watersurface multi-scene outdoor applications,” *Autonomous Robots*, vol. 45, Jan. 2021. DOI: [10.1007/s10514-020-09948-3](https://doi.org/10.1007/s10514-020-09948-3).
- [102] Z. Zhu, S. Yang, H. Dai and F. Li, “Loop detection and correction of 3d laser-based slam with visual information,” May 2018, pp. 53–58. DOI: [10.1145/3205326.3205357](https://doi.org/10.1145/3205326.3205357).



UNIVERSITÉ CATHOLIQUE DE LOUVAIN  
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | [www.uclouvain.be/epl](http://www.uclouvain.be/epl)