ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΠΑΡΑΓΩΓΗΣ
http://www.eee.uniwa.gr
http://www.idpe.uniwa.gr
Θηβών 250, Αθήνα-Αιγάλεω 12241
Τηλ: +30 210 538-1614
Διατμηματικό Πρόγραμμα Μεταπτυχιακών Σπουδών
Τεχνητή Νοημοσύνη και Βαθιά Μάθηση
https://aidl.uniwa.gr/

UNIVERSITY OF WEST ATTICA
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING
DEPARTMENT OF INDUSTRIAL DESIGN AND PRODUCTION ENGINEERING
http://www.eee.uniwa.gr
http://www.idpe.uniwa.gr
250, Thivon Str., Athens, GR-12241, Greece
Tel: +30 210 538-1614
Master of Science in
Artificial Intelligence and Deep Learning
https://aidl.uniwa.gr/

Master of Science Thesis

Study of simulation and modelling tools for autonomous vehicle driving based on stereoscopy

Student: Apostolou, Nikolaos
Registration Number: AIDL-0002

MSc Thesis Supervisor

Piromalis, Dimitrios
Associate Professor

ATHENS-EGALEO, February 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΠΑΡΑΓΩΓΗΣ
http://www.eee.uniwa.gr
http://www.idpe.uniwa.gr
Θηβών 250, Αθήνα-Αιγάλεω 12241
Τηλ: +30 210 538-1614
Διατμηματικό Πρόγραμμα Μεταπτυχιακών Σπουδών
Τεχνητή Νοημοσύνη και Βαθιά Μάθηση
https://aidl.uniwa.gr/

UNIVERSITY OF WEST ATTICA
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING
DEPARTMENT OF INDUSTRIAL DESIGN AND PRODUCTION ENGINEERING
http://www.eee.uniwa.gr
http://www.idpe.uniwa.gr
250, Thivon Str., Athens, GR-12241, Greece
Tel: +30 210 538-1614
Master of Science in
Artificial Intelligence and Deep Learning
https://aidl.uniwa.gr/

Μεταπτυχιακή Διπλωματική Εργασία

Μελέτη εργαλείων προσομοίωσης και μοντελοποίησης για την καθοδήγηση αυτόνομων οχημάτων με βάση στερεοσκοπική διάταξη

Φοιτητής: Αποστόλου Νικόλαος
ΑΜ: AIDL-0002

Επιβλέπων Καθηγητής
Πυρομάλης Δημήτριος
Αναπληρωτής Καθηγητής

ΑΘΗΝΑ-ΑΙΓΑΛΕΩ, Φεβρουάριος 2024

This MSc Thesis has been accepted, evaluated and graded by the following committee:

Supervisor	Member	Member
Dimitrios Pyromalis Digitally signed by Dimitrios Pyromalis Date: 2024.04.21 00:19:31 +03'00'	Panagiotis Papageorgas Digitally signed by Panagiotis Papageorgas Date: 2024.04.21 10:55:00 +03'00'	Dimitrios Kantzos Digitally signed by Dimitrios Kantzos Date: 2024.04.22 09:56:10 +03'00'
Piromalis, Dimitrios	Papageorgas, Panagiotis	Cantzos, Demetrios
Associate Professor	Professor	Professor
Electrical and Electronics Engineering Department	Electrical and Electronics Engineering Department	Industrial Design and Production Engineering Department
University of West Attica	University of West Attica	University of West Attica

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Αποστόλου Νικόλαος του Γεωργίου, με αριθμό μητρώου AIDL-0002 μεταπτυχιακός φοιτητής του ΔΠΜΣ «Τεχνητή Νοημοσύνη και Βαθιά Μάθηση» του Τμήματος Ηλεκτρολόγων και Ηλεκτρονικών Μηχανικών και του Τμήματος Μηχανικών Βιομηχανικής Σχεδίασης και Παραγωγής, της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής,

δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της μεταπτυχιακής διπλωματικής εργασίας και κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος. Η εργασία δεν έχει κατατεθεί στο πλαίσιο των απαιτήσεων για τη λήψη άλλου τίτλου σπουδών ή επαγγελματικής πιστοποίησης πλην του παρόντος.
Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του διπλώματός μου.
Επιθυμώ την απαγόρευση πρόσβασης στο πλήρες κείμενο της εργασίας μου μέχρι 31/02/2025 και έπειτα από αίτησή μου στη Βιβλιοθήκη και έγκριση του επιβλέποντος καθηγητή.»

Ο Δηλών
Αποστόλου Νικόλαος του Γεωργίου

[illegible]

(Υπογραφή φοιτητή/ήτριας)

Declaration of the author of this MSc thesis

I, Nikolaos Georgios Apostolou with the following student registration number: AIDL-0002, postgraduate student of the MSc programme in "Artificial Intelligence and Deep Learning", which is organized by the Department of Electrical and Electronic Engineering and the Department of Industrial Design and Production Engineering of the Faculty of Engineering of the University of West Attica, hereby declare that:
I am the author of this MSc thesis and any help I may have received is clearly mentioned in the thesis. Additionally, all the sources I have used (e.g., to extract data, ideas, words or phrases) are cited with full reference to the corresponding authors, the publishing house or the journal; this also applies to the Internet sources that I have used. I also confirm that I have personally written this thesis and the intellectual property rights belong to myself and to the University of West Attica. This work has not been submitted for any other degree or professional qualification except as specified in it.
Any violations of my academic responsibilities, as stated above, constitutes substantial reason for the cancellation of the conferred MSc degree.
I wish to deny access to the full text of my MSc thesis until 31/02/2025, following my application to the Library of UNIWA and the approval from my supervisor.

The author
Nikolaos Apostolou

[illegible]
(Signature)

I dedicate this thesis to my beloved father who passed away while I was writing it.

For the preparation of this thesis, as part of my studies in the Master's Program "Artificial Intelligence and Deep Learning" of the Department of Electrical and Electronic Engineering, I would like to especially thank the supervising professors Papageorgas Panagiotis and Pyromalis Dimitrios for giving me the opportunity to work on such an interesting subject, and for the valuable help and guidance they have given me in its writing and completion.
I would like also to express my sincere gratitude to Professor Panagiotis Papageorgas for his patience, and his encouragement. Without his generous support this journey would not be possible.

Abstract

The subject of this thesis is the analysis and study of the tools available today that assist engineers in the modeling and development of autonomous vehicles with automatic navigation. A comprehensive explanation of the basic concepts will be made to familiarize the reader with the specific technology.

The individual systems of autonomous vehicles will be analyzed from the hardware and software perspective, and the tools that can be used to simulate autonomous vehicle operations and their underlying algorithms will be explored. In addition, the application of stereoscopic vision in these functions will be analyzed.

Then a summary review of the utilization of artificial intelligence in combination with current technology review of stereoscopic vision using cameras will be presented along with references to literature.

Finally, a review of a specific stereoscopic camera will take place (Intel RealSense D435), and a simple example of integrating the camera with the MATLAB platform will be given. Following that, MATLAB will be used to calibrate the camera. In addition some measurements will be performed to test the accuracy of the specific camera.

Keywords

Autonomous vehicles, Robots, Modeling and Simulation, Stereo Vision, RealSense

Περίληψη

Το αντικείμενο της εργασίας είναι η μελέτη των εργαλείων που διατίθενται σήμερα και βοηθούν τους μηχανικούς στην μοντελοποίηση και ανάπτυξη αυτόνομων οχημάτων με αυτόματη πλοήγηση.

Θα γίνει μια περιεκτική εξήγηση των βασικών εννοιών ώστε να εξοικειωθεί ο αναγνώστης με την συγκεκριμένη τεχνολογία.

Θα γίνει ανάλυση των επιμέρους συστημάτων των αυτόνομων οχημάτων από την σκοπιά του υλικού καθώς και του λογισμικού και θα διερευνηθούν τα εργαλεία τα οποία μπορεί να χρησιμοποιηθούν για προσομοίωση των λειτουργιών των αυτόνομων οχημάτων καθώς και των αλγορίθμων που διέπουν αυτές τις λειτουργίες. Ειδικότερα θα αναλυθεί και η χρήση της στερεοσκοπικής όρασης στις λειτουργίες αυτές.

Στη συνέχεια θα παρουσιαστεί μια σύνοψη της χρήσης της τεχνικής νοημοσύνης στις παραπάνω λειτουργίες και σε συνδυασμό με διερεύνηση της τρέχουσας τεχνολογίας στερεοσκοπικής όρασης με χρήση καμερών συνοδευόμενη από αναφορές σε βιβλιογραφία.

Τέλος θα γίνει παρουσίαση συγκεκριμένου μοντέλου στερεοσκοπικής κάμερας (Intel RealSense D435) με ανάλυση των δυνατοτήτων της και θα δοθεί ένα απλό παράδειγμα διασύνδεσης της εν λόγω κάμερας με πλατφόρμα MATLAB με διερεύνηση δυνατότητας βαθμονόμησής της μέσω της πλατφόρμας καθώς και μέτρηση της ακρίβειάς της.

Λέξεις – κλειδιά

Αυτόνομα οχήματα, Προσομοίωση και μοντελοποίηση, Στερεοσκοπική Όραση.

Table of Contents

List of Tables

List of Figures

Acronym Index

ADAS:	“Advanced Driver Assistance System”
AI:	“Artificial Intelligence”
AV:	“Autonomous Vehicle”
CNN:	“Convolutional Neural Networks”
FoV:	“Field of View”
GAN:	“Generative Adversarial Networks”
GPS:	“Global Positioning System”
IEEE:	“The Institute for Electrical and Electronics Engineers”
IMU:	“Inertial Measurement Unit”
ROS:	“Robot Operating System”
SAE:	“Society of Automotive Engineers”
SLAM:	“Simultaneous Localization and Mapping”
SRP:	“Scriptable Render Pipeline”
YOLO:	“You Only Look Once”

INTRODUCTION

The subject of this thesis

The subject of this thesis is to present a broad overview of current stereoscopy technology and the tools available to model and simulate the functions and algorithms for navigating autonomous vehicles.

Simulation is an important tool in the development process since it enables the developers to test and evaluate methods and algorithms without the need to perform expensive and potentially dangerous testing in real situations. Furthermore, it would be extremely difficult to create specific testing scenarios in a populated city environment.

In recent years a plethora of 3D sensors have become available thus enabling 3D vision applications. These sensors are mostly based on active 3D mapping technologies which include two different technological families that both use additional source of light: the Time-of-Flight (ToF) and the Structured Light families. Besides active technologies, however, small and compact 3D cameras have also made their appearance, using passive technologies such as processing stereo image pairs, and being small and lightweight with low power consumption can further increase the range of applications on which they can be utilized.

Methodology

The review of the related technology was achieved through a literature search in IEEExplore[1], Researchgate[2] and Arxiv[3] for related keywords and the study of relevant publications and identified key studies and the corresponding results.

[1] https://ieeexplore.ieee.org/Xplore/home.jsp

[2] https://www.researchgate.net/

[3] https://arxiv.org/

1 Autonomous Vehicles

An autonomous vehicle (AV) is a car that is able to move and navigate by itself without the assistance and intervention of a human driver. An AV consists of two major components [1]: Hardware (sensors etc.) and Software (functional groups) which will be reviewed in the next sections. These two components are enabling the vehicle to make decisions and navigate in real time. The cameras and sensors transmit information to the onboard computer that processes the data and forwards instructions to actuators, which in turn perform the vehicle control, i.e. steering, as well as braking and acceleration.

Computer vision (CV), Artificial intelligence (AI), and machine learning algorithms (ML) are some of the scientific fields that contribute to the technology of self-driving automobiles.

One of the most significant scientific fields in the concept of autonomous vehicles is Computer Vision. In order for autonomous vehicles (AVs) to navigate successfully, they need to recognize objects in addition to other vehicles in their surroundings in real-time, such as road signs, obstacles, and pedestrians. To do this, AVs integrate advanced cameras and sensors with object detection algorithms.
The public's acceptance of autonomous vehicles (AVs) and their commercial availability have increased significantly due to the swift advancements in car cameras and artificial intelligence in vision which brought them even closer to meeting safety requirements.

A scale for vehicle automation that goes from 0% to 100% automation has been established by the Society of Automotive Engineers (SAE) [2].
At **Level 0**, automated warnings and temporary assistance like lane departure alerts and emergency braking are provided.
At **Level 1**, the difference from the previous level is that it provides support for either steering or braking and acceleration.
At **Level 2**, support is provided to both steering and braking/acceleration. This support is achieved through lane centring and adaptive cruise control.
The first three levels provide features that support the driver who actually is the one who performs the vehicle control. The next three enable true autonomous driving and in this case the driver is not required to have vehicle control or supervision.
At **Levels 3 and 4**, features are provided that enable autonomous vehicle operation under particular circumstances. Level 3 features in specific situations might necessitate the driver to intervene and take control of the vehicle.
Level 5 provides the same features with the previous level, the only difference being that in this level those features are capable of operating the vehicle autonomously in any road conditions.
Today automated driving systems are mainly at levels 2 and 3.

1.1 Hardware

There are three major categories of hardware components that comprise autonomous vehicle systems. These are sensors, controllers, and actuators.

1.1.1 Vehicle Sensors

Sensors play a key role since they enable the AV to collect information from its environment. This information is then used by the system to navigate through the surroundings and perform various functions, including object detection and course planning.

Sensors can be categorized as passive and active. Active are the ones that emit some kind of energy and measure its reflection. Passive sensors do not emitting energy but instead detect emitted radiation from the environment and they do not interacting with it.

The most commonly used car sensors include:

1. Cameras, inertial measurement units (IMU) such as gyroscopes and accelerometers are examples of passive sensors.
2. LIDARs, Radars and ultrasonic transceivers. These fall under the category of active sensors.
3. Sensors that depend on external devices, such as Global Positioning Systems (GPS), which functions together with a number of satellites.

Different factors need to be considered when selecting sensors, such as cost, range, field of view (FoV), overall system complexity, sampling rate, accuracy, etc.

Next, the functionality of the most important AV sensors will be examined.

Cameras capture two-dimensional images, by recording the light reflected off of three-dimensional objects. The geometry of the three-dimensional scene may then be reconstructed using images that were taken from various angles. The majority of tasks related to autonomous car perception, like object detection and recognition or dividing the image view into regions assigning semantic labels (semantic segmentation), are designed for visual sensor systems.

Lidar is a sensor that measures the time that an emitted laser signal takes until it gets reflected back to the sensor. It also measures intensity of the signals it receives back. Lidar sensors provide high-resolution and high-precision 3D point clouds of the environment, which can be used for localization and mapping, but it is expensive, sensitive, and power-consuming. Because of their precision LIDARs are used for depth perception. LIDARs form the basis of most industrial autonomous car localization and mapping systems now in use.

Radar is a type of sensor that uses radio waves to measure the reflected signals' phase difference and frequency shift. Although Radar has poor resolution and angular accuracy, it has long range and can reliably identify objects and obstacles in the environment, features which can be utilized for mapping and localization. Radars are widely used in the automobile industry's Advanced Driver Assistance Systems (ADAS) to help drivers and enable autonomous car features like adaptive cruise control and emergency braking.

An ultrasonic sensor is a sound-emitting device that detects the amplitude and time-of-flight of reflected signals. These sensors are low-cost devices that are in general able to function in short ranges and detect objects and obstacles in the surrounding environment. They have similar to Radars low resolution and angular accuracy, but they can be successfully utilized for mapping and localization.

GPS is a satellite-based technology that provides worldwide and absolute positioning information for the vehicle. GPS is widely used for localization and mapping, but it has limitations such as low accuracy, signal loss, and spoofing attacks. Such problems appear for example in buildings and mountains.

In these cases, Inertial Measurement Units (IMUs) are employed. IMU is a sensor that measures the vehicle's linear and angular velocities, accelerations, and orientations. IMU is often used in combination with GPS to improve localization accuracy and robustness, but it suffers from drift and noise errors over time. Typical IMU sensors are Gyroscopes, Magnetometers or Accelerometers.

Wheel encoders are another type of sensors which are mostly utilized to determine the Autonomous Vehicle position and measure its speed and direction by monitoring electronic signals generated by wheel motion.

1.1.2 Controllers

Hardware controllers in autonomous vehicles are devices that control the physical components of the vehicle, such as the engine, brakes, steering, and sensors. Hardware controllers receive commands from the software layer, such as the perception, planning, and control algorithms, and execute them by sending signals to the actuators. Hardware controllers also monitor the status and feedback of the vehicle components and sensors and report them to the software layer.

Hardware controllers are essential for ensuring the safety, performance, and reliability of autonomous vehicles, as they are responsible for the interaction between the software and the hardware. Hardware controllers need to meet the requirements of high speed, low latency, high accuracy, and high robustness, as well as comply with the standards and regulations of the automotive industry.

The electronic throttle, the torque steering motor, electronic brake booster, gear shifter, and electronic parking brake are a few examples of hardware controllers.

1.1.3 Actuators

Actuators are those devices in autonomous vehicles that translate commands from the controllers into a mechanical movement that performs the necessary operation. Some examples of actuators are throttle, steering or brake actuators.

Throttle actuators adjust the throttle to control vehicle acceleration.

Steering actuators control precisely the steering angle of the vehicle.

Brake actuators are responsible for applying or releasing the brakes thus controlling the stopping of the vehicle.

1.2 Software

Software components of autonomous vehicles are categorized according to the functions that enable them to perceive the environment and navigate.

1.2.1 Perception

Perception module is a software component that enables an autonomous vehicle to recognize and understand its environment and make decisions accordingly and is resembles the human visual cognition process. Perception software uses AV sensors, i.e., cameras, lidars, radars, and ultrasonic sensors, to detect and classify objects, such as pedestrians, other vehicles, road lanes, traffic signs, and traffic lights, in the vehicle's surroundings. Perception software also performs tasks such as tracking, and calibration to provide accurate and reliable information for the vehicle's planning and control systems.

There are two major categories of Perception technologies. One is based on computer vision and the other on Artificial Intelligence.

In the first category visual perception problems are modeled with the help of projective geometry and the best solution is selected with optimization techniques.

In the second category, the best solution is selected using data-driven techniques. One example of such a technique is classification models based on Convolutional Neural Networks (CNNs).

1.2.2 Localization and Mapping

Localization and mapping for autonomous vehicles are two interrelated tasks that enable the vehicle to estimate its precise location and orientation in the environment and to create and update a map of its surroundings. Based on these two tasks autonomous vehicles manage to navigate safely and efficiently in real situations facing complex and dynamic scenarios.

There are different methods and techniques for localization and mapping for autonomous vehicles, depending on the type and number of sensors, the accuracy and reliability requirements, and the computational and memory constraints.

Some of the common methods and techniques for combining and processing the sensor data for localisation and mapping are:

Simultaneous Localization and Mapping: Simultaneous Localization and Mapping, or SLAM, is a technique that addresses both localization and mapping issues at the same time. It does this by iteratively updating the map and the vehicle's pose based on sensor measurements and information. Depending on the kind and representation of the map, there are various forms of SLAM, including feature-based, direct, dense, and semantic SLAM.

Kalman Filter: This method estimates the system's state and uncertainty by utilizing sensor measurements and a mathematical model of the system and enables AVs to localize themselves about known landmarks. SLAM, lidar odometry, GPS/IMU fusion, and other localization and mapping issues can use the Kalman filter method to reduce the uncertainty provided by different sources of information and increase the accuracy of positioning systems.

Particle Filter: Particle filtering is a method that represents the system's state (e.g. AV position and orientation) and probability distribution via a collection of random samples, or particles. As the AV moves the particles are updated with sensor measurements to approximate AV's real state. Updating particles continuously results in converging their values in the true AV state.

1.2.3 Prediction

A prediction module for autonomous vehicles is a software component that predicts the future motions and behaviors of other traffic agents, such as vehicles, cyclists as well as pedestrians and animals, in the vicinity of the autonomous vehicle. Prediction module outputs are used by the vehicle to plan safe and efficient navigation routes, avoid collisions, and adapt to dynamic and uncertain situations.

Deep learning has become popular for prediction in autonomous driving, as it can handle high-dimensional and noisy sensor data, capture complex and nonlinear patterns, and generate long-term and multimodal predictions. Some examples of deep learning-based prediction methods are recurrent neural networks (RNNs) -used for sequential data, convolutional neural networks (CNNs) -used for image related tasks, generative adversarial networks (GANs) -used for generating realistic data, and attention mechanisms -used in processing sequential data.

1.2.4 Planning

An autonomous vehicle's planning module is a software element that determines the best and most optimal paths for the car to take using data from its perception and prediction modules, its current state and capabilities, and the rules and regulations governing traffic. Planning methods fall into one of the following categories:

Methods Based on Optimization: Optimization-based approaches approach the planning problem similarly to an optimization problem, aiming to maximize or reduce a cost or utility function while taking into account certain limitations like the dynamics of the vehicle, the geometry of the route, and the avoidance of collisions.

Sampling-Based Methods: Sampling-based methods are methods that generate a collection of possible trajectories by randomly sampling the state space of the vehicle, and then selecting the best one according to some criteria, such as the distance to the goal, the collision risk, and the comfort level. Sampling-based methods can handle complex and uncertain scenarios and can be easily parallelized and distributed, but they are not guaranteed to find the optimal or feasible solution.

1.2.5 Control

A control module for autonomous vehicles is a software component that executes the trajectories produced by the planning module and transmits commands to the hardware controllers to control the vehicle’s actuators, such as the throttle, brake, and steering.

Some of the common methods and techniques for autonomous vehicle control are:

PID Control: This is a simple and widely used control method. PID control adjusts the control input based on three components of the error between intended and actual outputs. These are the proportional, integral, and derivative components. PID control can provide fast and stable responses, but it requires careful tuning of the parameters and may not be robust to disturbances and uncertainties.

Model-Based Control: This control method designs the control input by using a mathematical model of the dynamics and kinematics of the vehicle. Model-based control can provide optimal and smooth control, but it requires accurate and reliable models and may not be adaptive to model errors and parameter variations. Model Predictive Control (MPC), Linear Quadratic

Gaussian (LQG), and Linear Quadratic Regulator (LQR) are a few types of model-based control techniques.

2 Stereo Vision

The development of computer vision is based on human vision. Combining the images captured by each of our eyes and utilizing the difference (also called disparity) between these images, allows human beings to acquire a perception of depth. The design and implementation of efficient algorithms that imitate our ability to carry out this depth perception activity is the main focus of computer stereo vision and is known as stereopsis. Efficient computer algorithms for stereoscopic perception are of great importance in several contexts. Cartography (creating accurate topographic maps), robot navigation, aerial reconnaissance (analyzing landscapes), and close-range photogrammetry (measuring distances and dimensions from photographs) are some of them. Additionally, they play a major role in processes like (a) building three-dimensional scene models which is used in computer graphics applications or (b) segmenting images for object recognition.

Specifically, when used in autonomous vehicles Stereo Vision can build a three-dimensional visualization of the environment, which can be analyzed by the autonomous vehicle's algorithms to make decisions about steering, acceleration, and braking.

2.1 Processes

Stereo vision aims to reconstruct the target scene within the computer. According to [3] the entire stereo vision system that carries out this task can be separated into six distinct functional blocks, to complete the stereovision task.

- Camera Calibration
- Image Acquisition
- Feature Extraction
- Stereo Matching
- 3-D Information Recovery
- Post-Processing

Camera calibration: The task of establishing the internal (also called intrinsic) and external (also called extrinsic) characteristics of a camera, is known as camera calibration. The whole process is based on an imaging model. An example case of an imaging model is the linear or Pinhole model where the camera is represented as a simple dark box with a small hole or aperture on one side. These parameters obtained by the calibration process define a relationship that maps a specific point of the 3-D coordinate system with its respective 2-D point on the image plane. Several cameras are frequently used in stereo vision, and each camera is calibrated independently.

Image acquisition: A 2-D picture is obtained from a 3-D physical scene using general imaging methods. While the information on the plane located perpendicular to the camera's optical axis is often retained in the picture, the depth information along the camera's optical axis is lost.

To perform 3-D computer vision, it is frequently required to acquire three-dimensional data from the target environment or a higher-dimensional form comprehensive information. The acquisition of 3-D images is therefore necessary. This covers both the collection of 3-D images explicitly and the implicit collection of images that contain 3-D information, with the following processing necessary to extract the 3-D information from the images.

In particular, there are numerous ways to acquire (or recover) depth information. These include the use of specialized tools and equipment to directly collect distance data, the layer-by-layer movement of the focus plane to obtain three-dimensional information, and stereo vision technology, which imitates the human binocular vision system to perceive the environment.

Feature extraction: Feature extraction is the method of acquiring relevant and distinctive information from stereo images, such as pixel values, colors, edges, corners, textures, and descriptors. Feature extraction is an important step for stereo vision, as it enables the matching of corresponding points in the "stereo pair", which is essential for computing the disparity and reconstructing the 3D scene. Different methods for feature extraction include:

Pixel-Based Methods: Pixel-based methods are methods that use pixel intensity or color values as the features for stereo matching. Pixel-based methods are simple and fast, but they are sensitive to noise, illumination, and occlusion, and they may not be distinctive enough for complex scenes.

Edge-Based Methods: Edge-based methods are methods that use the edges or contours as the features for stereo matching. Edge-based methods are robust to noise and illumination, and they can preserve the shape and structure of the objects, but they may not be reliable for regions without texture or discontinuous regions, and they may suffer from edge fragmentation and ambiguity.

Corner-Based Methods: Corner-based methods are methods that use the corners or interest points as the features for stereo matching. Corner-based methods are invariant to rotation and scale, and they can reduce the computational cost by using sparse features, but they may not be stable for noisy or blurred images, and they may miss some important information in smooth regions.

Texture-Based Methods: Texture-based methods are methods that use the texture or local patterns as the features for stereo matching. Texture-based methods can capture the fine details and variations of the images, and they can handle textureless or repetitive regions, but they may not be robust to noise and distortion, and they may require large memory and computation resources.

Descriptor-Based Methods: Descriptor-based methods are methods that use descriptors or feature vectors as the features for stereo matching. Descriptor-based methods can combine multiple types of features, such as pixel, edge, corner, and texture, and encode them into compact and discriminative representations, but they may require complex and time-consuming algorithms, and they may depend on the quality and selection of the features.

Stereo matching is the process of finding the corresponding points in a set of stereo images (i.e., images captured from a slightly different viewpoint), which are aligned such that the corresponding points lie on the same horizontal line. Stereo matching is an important step for stereo vision, as it enables the computation of the disparity, which is the difference in horizontal coordinates between the corresponding points. The disparity is inversely proportional to the depth of the point in the 3D scene and can be used to reconstruct the 3D structure of the scene.

Stereo matching is the most significant but also the most difficult part of stereovision.

Currently, stereo-matching technologies are divided into the following categories:

Local Methods: Local methods are methods that compare small windows or patches around each pixel in the reference image with the corresponding windows or patches in the other image, and select the best match based on some similarity or dissimilarity measures, such as a sum of absolute differences (SAD), normalized cross-correlation (NCC), sum of squared differences (SSD), or census transform. Local methods are fast and simple, but they may not be robust to noise, occlusion, and textureless regions, and they may produce noisy and inconsistent disparity maps.

Learning-Based Methods: Learning-based methods are methods that use machine learning techniques, such as artificial neural networks, support vector machines, random forests, and deep learning, to learn the stereo-matching function from data. Learning-based methods can handle complex and nonlinear systems and can be adaptive and data-driven, but they require large amounts of data and may not be interpretable or verifiable. A review of deep learning techniques for stereo matching is given in [4].

Feature matching is a method based on feature points. Features represent points with unique or special properties in the image as matching points. The features used for this purpose might be an edge line segment, the coordinates of the contour changing point or corner point of the image, the outline of the shape of the object, etc.,

3D information recovery: 3D information recovery in stereo vision is the process of reconstructing the 3D structure of a scene from a pair of rectified stereo images, which are aligned such that the corresponding points lie on the same horizontal line. 3D information recovery is an important application of stereo vision, as it enables the estimation of the depth and shape of the objects in the scene, which can be used for tasks such as autonomous driving, robot navigation, and 3D reconstruction.

The factors that affect the accuracy of distance measurement mainly include digital quantization effects, camera calibration errors, exactness of feature detection, as well as matching and positioning.

Post-Processing: The postprocessing process in stereo vision is the process of improving the quality and reliability of the disparity map and the 3D reconstruction results obtained from the stereo-matching process. The postprocessing process is important for stereo vision, as it can reduce the noise, outliers, and errors in the disparity map, fill the holes, smooth the edges, and enhance the details in the 3D reconstruction.

The main types of commonly used post-processing are:

Hole Filling: Hole filling is a technique that fills the holes or gaps in the disparity map or the 3D reconstruction, by using interpolation, inpainting, or propagation methods. Hole filling can improve the completeness and smoothness of the disparity map and the 3D reconstruction, but it may also introduce artefacts or distortions in the results.

Variance Check: Variance check is a technique that measures the variance or confidence of the disparity values in a local window and filters out the pixels with high variance or low confidence, which are likely to be noisy or ambiguous. Variance checks can improve the reliability and consistency of the disparity map, but it may also remove some valid or fine details in the disparity map.

Weighted Median Filter: Weighted median filter is a technique that applies a median filter to the disparity map, but assigns different weights to the pixels according to their similarity or distance to the center pixel. Weighted median filter can reduce the noise and outliers in the disparity map, and preserve the edges and details in the disparity map, but it may also smooth out some valid or fine details in the disparity map.

2.2 Stereo Vision Technology Review

Stereo Vision has been studied extensively in recent years and reviewed across different industries. Especially in computer vision, autonomous vehicles, and robotics.

A review of stereo vision algorithms based on their real-time applicability has been published in the Journal of Real-time Image Processing, 2016 [5]. Algorithms were categorized into three groups: the first group includes algorithms performing in real-time on standard processors, the second group the ones that have shown real-time performances in specialized hardware such as FPGA, ASIX, GPU, etc., and the last group the ones that have not demonstrated real-time performance. The algorithms were compared in terms of accuracy (percentage of pixels with incorrect disparity) and performance (in Millions of disparity evaluations per second). There is always a tradeoff between accuracy and performance. Algorithms with real-time performance on standard CPUs achieved accuracy between 85% and 96%. Algorithms in specialized HW achieved the same level of accuracy but x200 faster.

Simultaneous Localization and Mapping (SLAM) as it was described in 1.2.2 is a very important task for autonomous vehicles working in unknown environments since it provides localization information and enables the vehicle to know its position related to the environment. Various sensors are used to enable SLAM such as GPS, Wireless Sensor Networks, Lidar etc.

In addition to those sensors, stereo vision sensors also enable autonomous vehicles to perform SLAM and they are relatively cheap compared to other sensors. Vision-based SLAM methods are called VSLAM and are very active in current research. There are many research publications as mentioned in [6] that have shown VSLAM methods that can perform better than traditional SLAM which rely on a single sensor such as LIDAR. VSLAM methods have evolved significantly from monocular systems to recent ones such as ORB-SLAM3 which exploit parallel execution achieving real-time results.

In theory, single camera sensors are not able to provide depth information, even though there are papers describing how depth information can be gained. For example, in [7] a depth estimation model for monocular vision is presented, using unsupervised deep learning. Other deep learning methods have also been published but they require powerful CPUs so these methods are not a good fit for autonomous vehicles.

Stereo cameras providing dual view can provide depth information more efficiently as already described in 2.1.

Another type of camera providing depth information is RGB-D. These cameras also provide color information (RGB) and use active measurements (infrared or laser) thus offering richer information which makes them ideal for SLAM applications. Examples of this type of camera is Microsoft Kinect or Intel RealSense. RealSense will be reviewed later in section 5.2.

VSLAM methods are either direct or indirect. In indirect methods, features of every frame pair are acquired and then matched to establish depth information or trajectory. Direct methods on the other hand use illumination information of Regions of Interest to align images.

ORB-SLAM2 [9] is an open-source framework able to handle stereo, monocular and RGB-D cameras. It was developed by the Computer Vision and Robotics Group at the University of Zaragoza in Spain.

According to [10] the ORB-SLAM2 framework (Oriented FAST and Rotated BRIEF - Simultaneous Localization and Mapping) is the best indirect framework in terms of accuracy although is slower than the rest.

A more recent version of the ORB-SLAM framework has been recently published named ORB-SLAM3 [11]. It introduces relocalization capability, i.e. it can relocalize itself when the track is lost, and also introduces a virtual-inertial SLAM which combines IMU data with visual information.

Another review of VSLAM methods [12] indicated that although remarkable results have been achieved there are still some challenges and open issues. VSLAM techniques face some limitations when variations in illuminations or reflective surfaces and untextured objects are involved.

Combining Stereo Vision with additional algorithms enables autonomous vehicles to perform more advanced tasks. In [13] for example, Stereo Vision is combined with an object detection algorithm making the vision system capable of providing information about objects such as distance and dimensions. A CNN is used to detect the object and then with triangulation calculates the distance height and width of the object. Their experiment showed an accuracy of 0.3% in dimensions and 2.8% in distance.

In the same direction, pedestrian detection is a very important task and is part of Advance Driver Assistant Systems (ADAS). In [14] stereo vision is used to acquire a depth map and detect objects. Intensity-Hue-Saturation transform is used to fuse the images, restrict the area of interest around the object detected and then send this object image to the next step for pedestrian detection achieving detection times less than 6ms. Usually, today’s ADAS systems use data fusion from other sensors besides vision such as LIDAR or Radar to enhance detection time to acceptable levels.

Another task where stereo vision has been applied is speed estimation of objects which can be utilized in autonomous vehicles to estimate the speed of other vehicles. In [15] a method is proposed in which at first objects are detected in subsequent frames using YOLO (CNN-based algorithm), and then these objects are matched based on several features. Then the centroids of the matched objects are found and using the disparity maps from the stereo image pairs their depth is calculated. From the depths of the centroids and the speed of the video stream in frames per second, the object’s speed can be calculated.

3 Simulation and Modelling tools

Simulation and modeling tools for autonomous vehicle driving involve using computer programs to create virtual environments in which autonomous vehicles can be tested and trained to navigate roads and traffic safely and efficiently.
Validation and testing of autonomous vehicles in a real scenario involving pedestrians and cars would be unrealistic in terms of cost. Furthermore, because of the uncertainty of the real world as new functionalities are developed, new problems appear which is a potential cause of accidents.
Because of the problems described above, testing autonomous vehicles in a simulated environment becomes an important part of the development process. Computer simulation can model a system including its static and its dynamic characteristics.
By applying simulation in different levels several testing setups can be created such as Hardware in the loop or Software in the loop. Hardware in the loop incorporates for example real sensors in the simulation setup to validate the design. Software in the loop on the other hand enables testing of the algorithms used in the various models.

Some popular simulation and modeling tools for autonomous vehicle driving based on stereoscopy include:

CARLA: Users can create virtual landscapes with realistic graphics and physics, including a variety of weather and lighting conditions, using this open-source simulator for study on autonomous driving.

Unity: A quite popular game engine that can be used to create virtual worlds for testing autonomous vehicle algorithms. By combining Unity with machine learning techniques, users can create specific scenarios with realistic physics and train autonomous vehicles.

NVIDIA DRIVE: NVIDIA DRIVE is a platform that provides tools for developing and testing autonomous driving algorithms, including simulation tools that support stereoscopic cameras. It includes a range of pre-built environments and scenarios for testing autonomous vehicles.

MATLAB Simulink: MATLAB Simulink is a widely used tool for modeling and simulating complex systems, including autonomous vehicles. It includes support for stereoscopic cameras and can be used to model the behaviour of an autonomous vehicle in a variety of scenarios.

Gazebo: Gazebo is a simulation environment that allows to design, test, and evaluate robots and other dynamic systems in realistic scenarios.

3.1 ROS Framework

The most commonly used platform for building robots is the Robot Operating System (ROS), which can be used directly embedded in hardware or simulation. A vast array of preconfigured, modularized, and configurable software packages is included with ROS and can be utilized as robot plugins. This enables the robot to carry out several tasks, including mapping, localization,

navigation, reading data from sensors, and regulating actuator movements. Additionally, ROS offers a way for the different software components to communicate effectively with one another. The majority of these prebuilt software packages are regularly updated, put through extensive testing, and are actively used by the global robotics community. Because of all these capabilities, the robot developer may concentrate more on establishing higher-level functionalities rather than having to start from scratch and write basic software packages for each robot.
ROS provides a generic software framework addressing several prototyping issues in robot development such as software reuse, testing, debugging, portability etc. A software component created for one robot can easily ported to be used in a different one. Also by allowing the developers to test the behavior of a robot before creating the real physical one, reduces time, effort and cost.

3.1.1 ROS Architecture

Three components make up the architecture of ROS:

- File system;
- Computation graph
- Community (forums, resources, and information sharing and collaboration with the global ROS community)
-

File System

ROS uses a concept called the ROS file system, which is a way of organizing and finding files and resources within a ROS workspace.

In ROS, a workspace is a directory where you organize and build your robotic software. The workspace contains packages, which are the basic units of organization in ROS. Each package can have its directory structure with various files, such as source code, configuration files, launch files, and more.

Here are some key components related to the ROS file system:

Package Path (ROS_PACKAGE_PATH): This environment variable is a colon-separated list of directories. When you run a ROS command, it searches for packages in these directories. Each package in the path is checked for the presence of specific directories like src (source code), launch (launch files), and others.

Workspace: A ROS workspace is a directory that contains one or more ROS packages. The workspace is where you organize, build, and develop your robotic software.

Package: A ROS package is the basic unit of software in ROS. It contains the code, configuration files, launch files, and other resources necessary for a specific functionality.

Computational Graph

A conceptual depiction of the communication structure between the different nodes in a ROS system is the ROS computational graph. A robotic system in ROS is usually made up of several software modules, or nodes, that each carry out a particular function. The interactions between these nodes through topics, services, and actions are visualized by the computational graph.

Key components of the ROS computational graph include:

Nodes: Nodes are discrete software components that carry out particular functions. Sensor drivers, control algorithms, and user interfaces are a few types of nodes. Nodes can interact with one another through action-based communication, posting or subscribing to topics, and providing or utilizing services. The master is a central node that allows other nodes to discover each other to communicate.

Topics: these are named communication channels that enable nodes to send and receive messages are called topics. A topic is a collection of messages that nodes can publish, and other nodes can subscribe to receive the messages. Asynchronous communication between nodes is facilitated via topics.

Services: A synchronous request-response communication pattern is provided by services. When a node provides a service, other nodes can call it to make a specific task request. The outcome is then returned by the service-providing node.

Actions: Compared to services, actions offer a more complicated type of communication. They permit behavior that is goal-oriented and includes elements of feedback and outcome. Nodes can send action goals, receive updates on how the goal is doing, and eventually get a result.

ROS Community

The ROS community refers to the global network of developers, researchers, engineers, roboticists, and enthusiasts who use, contribute to, and support the development of ROS. ROS is an open-source middleware framework designed to help developers create complex and robust robot software.

Key aspects of the ROS community include:

Collaboration: The ROS community emphasizes collaboration and knowledge-sharing. Members contribute to the development of the ROS core, create and maintain packages (collections of related ROS nodes), and share their expertise through forums, mailing lists, and other communication channels.

Open Source: ROS is an open-source project, which means that its source code is freely available to the public. This openness encourages community members to contribute improvements, fixes, and new features, fostering a collaborative environment.

Forums and Mailing Lists: By using forums and mailing lists, ROS users and developers can interact with the community. A well-known Q&A site where people may post queries, look for assistance, and share their expertise is called ROS Answers. For more general debates and announcements, mailing groups like ros-users and ros-devel are also utilized.

Documentation: The community values documentation highly. A wealth of documentation, including tutorials, manuals, and API references, is offered by ROS. The documentation is frequently updated and improved by community members to make it more user-friendly for users with different levels of experience.

Conferences and Events: ROS-related conferences and events provide opportunities for community members to meet in person, share their work, and learn from each other. Examples include ROSCon, which is an annual conference focused on ROS, and various robotics and AI conferences where ROS-related research is presented.

Code Repositories: ROS code is hosted on public repositories like GitHub, allowing developers to contribute to the core framework and share their own ROS packages. This centralized platform facilitates collaboration and version control.

Members of the ROS community can take advantage of an extensive network of resources, expertise, and information. By facilitating cooperative efforts to overcome robotics and automation difficulties, it also promotes creativity and speeds up the development of robotic applications.

3.2 CARLA

CARLA [16] (Car Learning to Act) is an open-source simulator based on Unreal Engine that gives users the ability to control and modify several simulation elements. CARLA was introduced in [17] and has been developed from the beginning to support the development, training, and validation of autonomous driving systems. In addition to open-source code and protocols, CARLA provides free-to-use digital assets (buildings, vehicles, and urban layouts) that were specifically developed for this purpose. The simulation platform can be adapted with sensor suites, and environmental conditions, and enables complete control of all static and dynamic actors, creation of maps and much more.

Provides several features such as:

Flexible API: CARLA provides a robust API that lets users manage any aspect of the simulation, including weather, sensors, traffic generation, pedestrian behavior, and much more.

Sensor suite for autonomous driving: users can set up a variety of sensor suites, including as GPS, depth sensors, LIDARs, and multiple cameras.

Maps generation: using tools like RoadRunner, users can easily create their maps while being consistent with the OpenDrive standard.

Fast simulation for planning and control: when graphics are not required (e.g., in traffic simulation or road behaviors) there is a fast simulation mode which disables rendering achieving fast execution.

Traffic scenarios simulation: The ScenarioRunner engine allows users to define and execute a variety of traffic situations based on modular behaviors.

ROS integration: with ROS-Bridge CARLA can be integrated with ROS

Autonomous Driving baselines: Autonomous Driving baselines are provided as runnable agents in CARLA, including an AutoWare agent (allowing integration with AutoWare) and a Conditional Imitation Learning agent (allowing learning by human demonstrations).

An example from [17] is given below [Figure 1] for a street perspective in different weather conditions.

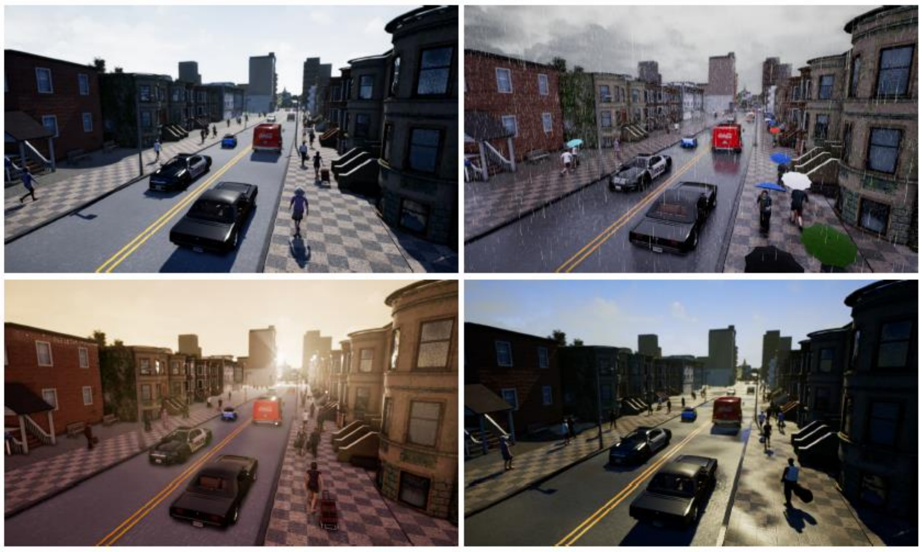


Figure 1: A city street in different weather conditions[4]

3.3 Unity

Unity is a real-time 3D rendering platform developed by Unity Technologies for three-dimensional and two-dimensional games and other interactive content development as well as simulations. It’s a tried-and-tested, full-featured platform that powers millions of multi-platform games and applications [18].

It was first released in 2005 as an OS X-exclusive game engine and is now available on more than 25 platforms.

A game engine is a common alternative to a simulator for robot and AV simulation. It offers representations of physical laws, collision detection, sound, animation, and 3D graphics, and some of them also offer Virtual Reality simulation support.

[4] Source: [17] Dosovitskiy, Alexey & Ros, German & Codevilla, Felipe & López, Antonio & Koltun, Vladlen, CARLA: An Open Urban Driving Simulator, 1st Conference on Robot Learning (CoRL 2017)

Unity has additional advantages not commonly found in other platforms. These include the Asset Store (see below), and it's extremely large community of cross-industry developers and creators.

The features and tools needed to create a simulation environment in Unity are the same ones used in creating other types of rich interactive content: these tools include lighting and physics, particle and weather systems, animation, and machine learning.

Some of the Features of Unity are described below:

Scripting Flexibility: scripting is supported via C# and UnityScript (similar to JavaScript). External libraries can also be integrated through languages such as C++, Python, etc. Unity also provides support for plugins allowing critical parts of the code to be written in C or C++ enabling developers to optimize these parts.

Fast prototyping: Unity provides an editor with a user-friendly interface enabling fast prototyping. It offers an editor-play mode in which the developer can preview the final result. The simulation can be played and paused at any point and the developer can change values, properties, and other attributes, and instantly see the outcome. Easy debugging is also possible by stepping through the simulation frame by frame.

Full interactivity: Unity provides a set of classes and methods exposing a powerful API that gives access to the Unity Engine and its systems and components including physics, rendering, animation, and communications.

High-end graphics: Unity accommodates a powerful rendering technology called Scriptable Render Pipeline or SRP, which allows the user to customize the rendering process in C#, thus giving much more granularity on the construction of each scene allowing to make it content-specific.

There are two SRPs available: the High-Definition Render Pipeline (HDRP) offers HD quality graphics on supporting high-performance hardware, and the Universal Render Pipeline (URP) scales easily across all Unity-supported platforms.

Advanced artist and designer tools: Unity contains a range of 3D visual design tools, lighting and special effects, audio systems, etc., which allows designers to create immersive environments.

Machine Learning and AI capabilities: Unity provides an ML-Agents toolkit which is a trainable agent through reinforcement learning or other machine learning techniques. It is a framework which enables researchers to study intelligent agents in complex environments and provides developers with the tools to implement AI in simulations with the latest machine learning technologies.

The Asset Store: The Asset Store is a marketplace which provides developers with productivity tools and off-the-shelf assets, with many options for environment creation, reducing development time.

Unity has been successfully used to evaluate stereo vision in autonomous vehicles. In [19] a simulation environment has been created with several city objects such as roads, pedestrians, buildings, crossroads, traffic signs, etc. and the motion of a single vehicle equipped with a single camera has been simulated. In the following picture [Figure 2], an example environment generated with Unity is presented as shown in [19].

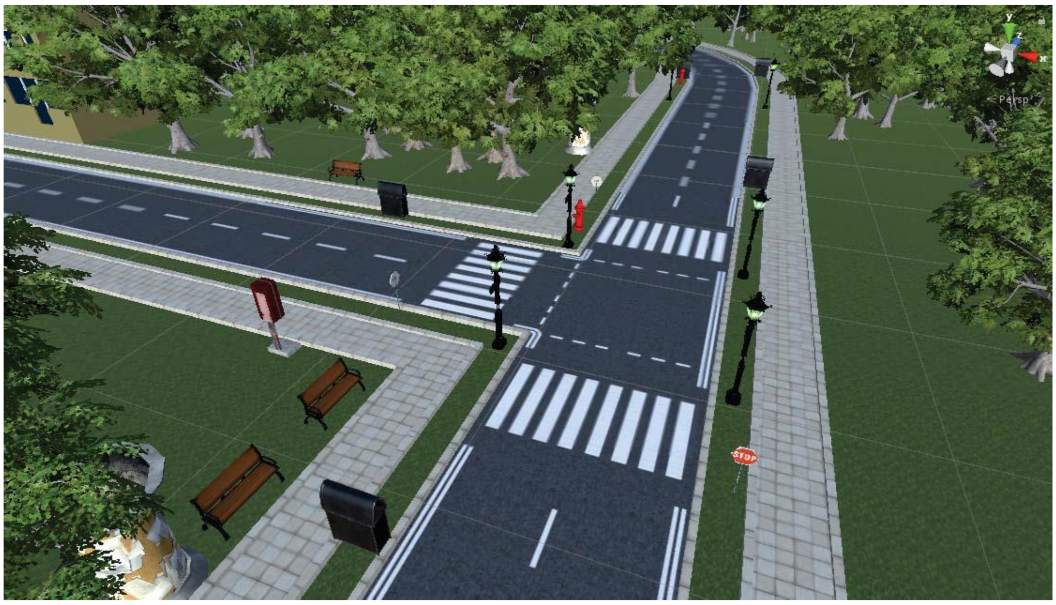


Figure 2: Unity generated environment example[5]

A simulated vehicle was created with sensors including a stereo camera. The Vehicle could drive along a predefined path and collect data. It has been shown how virtual images can captured and used to compute disparity data.

3.4 NVIDIA Drive

NVIDIA Drive is a scalable, open platform that enables the development and deployment of software-defined autonomous vehicles. It consists of hardware, software, and cloud components that work together to create a comprehensive solution for self-driving cars, trucks, robots, and more. NVIDIA Drive enables developers to leverage the power of NVIDIA GPUs, deep learning, computer vision, and sensor processing to create intelligent and safe mobility applications.

Some of the features of NVIDIA Drive include:

NVIDIA Drive AGx: a family of high-performance, energy-efficient hardware platforms that can deliver the computing power needed for autonomous driving by integrating multiple NVIDIA GPUs and NVIDIA Xavier system-on-chips (SoCs).

NVIDIA DriveWorks: a software development kit (SDK) providing a wide range of tools, libraries, and modules for autonomous vehicle software components such as planning, mapping, perception, as well as driver monitoring (through DMS which is a Driver Monitoring System for driver distraction and/or drowsiness).

[5] Source: [19] M. Vukić, B. Grgić, D. Dinčir, L. Kostelac and I. Marković, "Unity based Urban Environment Simulation for Autonomous Vehicle Stereo Vision Evaluation," 2019

NVIDIA Drive OS: is the real-time operating system (RTOS) that powers the NVIDIA DriveWorks SDK and is installed on the NVIDIA Drive AG hardware platform.

NVIDIA Drive AV: a cloud-based service that enables over-the-air (OTA) updates, fleet management, data collection, and remote assistance for autonomous vehicles.

Leading automakers, suppliers, startups, and academic institutions use NVIDIA Drive to speed up the development and implementation of driverless vehicles globally. Examples of partners and/or customers of NVIDIA Drive are Mercedes-Benz, Volvo, Tesla, Toyota, Hyundai, Honda, Audi, and more.

Nvidia Drive supports stereo vision through its DriveWorks SDK, which provides a stereo algorithm that can be executed on GPU or a combination of multiple hardware engines on Drive AGX boards. Stereoscopic rendering is also supported by Nvidia graphics cards, and for programs that don't support or generate stereo rendering natively, the driver can synthesize left and right eye images and extract depth information.

Furthermore, NVIDIA Drive Sim [20] is an end-to-end simulation platform that supports autonomous vehicle development and validation from concept to deployment. It is built on NVIDIA Omniverse, a scalable development platform based on Universal Scene Description (OpenUSD). NVIDIA DRIVE Sim is designed to run large-scale, physically based multi-sensor simulations that are open, scalable, and modular. It can generate immersive 3D environments with physically accurate simulations to support the creation of virtual test beds for AV performance testing.

NVIDIA DRIVE Sim is an open platform that maintains a rich partner community that create and deploys simulation models for developers and other ecosystem members enabling them to accelerate AV development and testing at scale.

NVIDIA DRIVE Constellation, [21] is a cloud-based vehicle simulation platform, which runs NVIDIA DRIVE Sim and generates sensor output from the simulated car and then sends these data to the target vehicle hardware in another DRIVE Constellation computer, which in turn sends back driving directions back to the simulator.

3.5 MATLAB/Simulink

MATLAB and Simulink are software products that enable engineers to perform numerical computation, data analysis, visualization, programming, simulation, and model-based design. MATLAB is a textual programming language that supports matrix operations, functions, and scripts. Simulink is a graphical programming environment that can be used to create block diagrams to model and simulate dynamic systems. MATLAB and Simulink can be used together to combine the power of textual and graphical programming in one environment.

3.5.1 Block Diagrams

As mentioned above, in Simulink, a graphical modeling and simulation environment for dynamic systems, Blocks can be used to represent different sections of a system organized in block diagrams. A block can model an actual part, a smaller system, or a whole function. A block is fully characterized by its input/output relationship.

Only after the inputs and outputs of a block are specified is its definition complete; this work is related to the model's objective.

Block libraries, which are collections of blocks arranged according to functionality, are offered by Simulink.

A component's input frequently has a delayed impact on its output. This impact might not be immediate but instead, it serves as a differential equation's input. A component state's history could also play a role. Simulink uses memory and numerical solvers when simulation calls for solving a differential or difference equation to evaluate the state for the next step in time.

Simulink handles data in three separate categories:

- Signals - During simulation, block inputs and outputs are calculated.
- States - Internal values calculated throughout the simulation that reflect the block's dynamics
- Parameters - User-controlled values that adjust a block's functionality

Simulink computes new values for states and signals at each time step.

3.5.2 Simulink and stereo vision

Simulink can be used to construct and simulate models of stereo-vision systems. The technique of retrieving three-dimensional information from two or more two-dimensional images of a scene is called stereo vision, and it has applications in autonomous driving, robot navigation, and three-dimensional reconstruction. Block diagrams can be created using Simulink, a graphical programming environment, to model and simulate dynamic systems like stereo vision algorithms.

To model stereo vision with Simulink, one can follow these general steps:

- Acquire stereo images from a pair of cameras or a video file.
- Rectify the stereo images to align the corresponding points in the same row.
- Estimate the disparity map by matching the corresponding points in the stereo pair.
- Reconstruct the 3D scene by triangulating the corresponding points using the camera parameters.

You can use the Computer Vision Toolbox blocks and functions to perform these steps in Simulink. For example, you can use the Stereo Camera Calibrator block to calibrate your stereo camera, the Rectify Stereo Images block to rectify your stereo images, the Disparity block to compute the disparity map, and the Point Cloud block to generate the 3D point cloud.

Stereo Visual Simultaneous Localization and Mapping: This is a documentation page that explains how to use Simulink to implement a stereo visual SLAM system, which can estimate the pose and map of a stereo camera in an unknown environment.

3-D Scene Reconstruction with Stereo Vision: This is an example that shows you how to use Simulink to perform stereo rectification, disparity estimation, and 3D reconstruction from a pair of stereo images.

3.5.3 Code Generation

The process of turning a Simulink model into executable code that can execute on a target hardware platform is known as "code generation from Simulink." There are several uses for code generation, including:

- Acceleration of simulation: Your model can be used to generate code that can be executed in a quicker simulation mode, like SIL (Software-in-the-Loop) or Rapid Accelerator.
- Rapid prototyping: To test your idea in real-time, you may create code from your model and install it on a prototype board like an Arduino or Raspberry Pi.
- Hardware-in-the-loop testing: You can generate code from your model and execute it on a hardware device that interfaces with a physical plant or system, such as a motor or a vehicle.
- Embedded deployment: You can generate code from your model and embed it in a production device, such as a microcontroller or an FPGA, to implement your application in the field.

To generate code from Simulink, you need to use either Simulink Coder or Embedded Coder. Simulink Coder generates standalone C and C++ code from Simulink models, Stateflow charts, and MATLAB functions. Embedded Coder extends Simulink Coder with additional features, such as code optimization, code verification, code traceability, and code interface customization.

3.6 Gazebo

Gazebo [22] is a simulation environment that offers the ability to design, test, and evaluate robots and other dynamic systems in realistic scenarios. Gazebo provides high-quality graphics, physics, sensors, and cloud services to make simulation easy and efficient. Gazebo can be used to simulate various applications, such as autonomous driving, robot navigation, 3D reconstruction, and more.

Gazebo runs on Linux machines or Linux virtual machines and uses a plugin package to communicate with MATLAB and Simulink. One can use MATLAB and Simulink to create and control models in Gazebo and perform data analysis and visualization. Code can also be generated code from Simulink models and can be deployed on simulated or real robots.

3.6.1 Gazebo and ROS integration

An extensively used middleware for robotics applications is ROS (Robot Operating System). Through the integration of Gazebo and ROS, it is possible to control and communicate with simulated robots and sensors in Gazebo using ROS messages, services, and dynamic reconfiguration.

The method of integrating Gazebo with ROS varies based on the ROS version being utilized. The ‘gazebo_ros_pkgs’ package collection, which offers wrappers around the standalone Gazebo, is required for ROS 1. On the other hand, for ROS 2 a collection of packages called ‘gazebo_ros2_pkgs’ must be utilized, as they offer comparable functionality although they have a few differences in the API and implementation.

Some of the benefits of integrating Gazebo and ROS are:

- The same code and tools for both simulation and real-world deployment can be used.
- Testing and debugging robotics algorithms in a realistic and safe environment.
- One can leverage the existing ROS ecosystem of packages and libraries for your simulation needs.

3.6.2 Stereo Vision with Gazebo

To use stereo vision with Gazebo first a camera model must be created in URDF or SDF format and attached it to the robot model.

Then utilizing the ‘gazebo_ros_pkgs’ or ‘gazebo_ros2_pkgs packages’ integrating Gazebo and ROS, and using ROS messages, services, and dynamic reconfigure to control and communicate with the simulated stereo camera.

In addition, through the use of blocks and functions in Simulink, stereo vision algorithms can be implemented and simulated, and code can be generated automatically for deployment on the real or simulated robot.

Recent references exist in the literature utilizing the Gazebo platform and Stereo Vision. In [23] a ROS-based stairs detection was implemented on a crawler robot Servosila Engineer and performed experiments with Gazebo virtual environment and a stereo camera [Figure 3: Robot inspects the car in gazebo virtual environment].

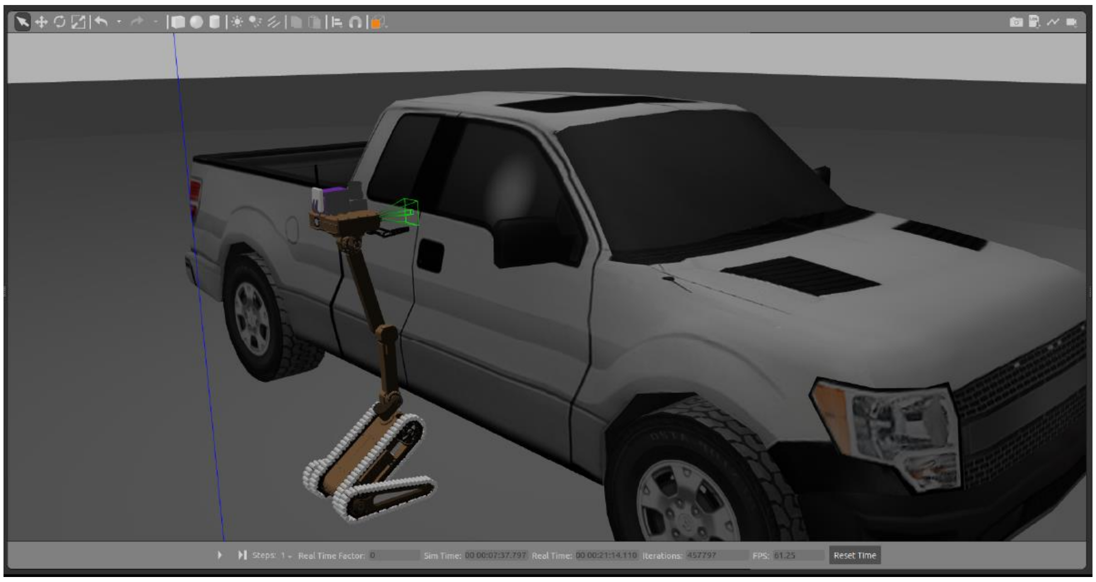


Figure 3: Robot inspects the car in gazebo virtual environment[6]

The simulation of the robot and a stereo camera is started in Gazebo with an environment that contains stairs. The robot cameras publish captured pictures to ROS topics. Then the ROS node receives both images and calculates a depth map.

In [24] a newly proposed stereo vision sparse 3D reconstruction algorithm was simulated in a gazebo environment carrying out comparison experiments.

4 Artificial Intelligence in Autonomous Vehicles

Artificial Intelligence (AI) has played a crucial role in advancing autonomous vehicles (AVs) by providing the necessary capabilities for perception, decision-making, and control.

The development of autonomous vehicles (AVs) has been greatly improved by artificial intelligence, which offers additional methods and tools needed for perception, decision-making, and control. However as mentioned in [25], a certain amount of uncertainty has also been introduced. Areas in which AI has contributed to the development and enhancement of autonomous vehicles are Perception and Sensor fusion, Object Detection, Path Planning, Localization and Mapping, etc. Specifically, Deep Learning methods are used to solve several issues in autonomous vehicles such as Path planning [26].

4.1 Machine Learning

Machine learning (ML) is a branch of artificial intelligence (AI) that focuses on the development of algorithms trained on data sets and creates models that enable computers to learn from and make predictions or decisions based on these data. The main objective of machine learning is to develop systems that can automatically perform better over time without being explicitly programmed for each task.

[6] Source: [23] M. Mustafin, T. Tsoy, E. A. Martínez-García, R. Meshcheryakov and E. Magid, "Modelling mobile robot navigation in 3D environments: camera-based stairs recognition in Gazebo," 2022

It is a technology that has many applications in the real world, such as self-driving cars, fraud detection, and speech recognition.

Machine learning algorithms require data for training. This data serves as experiences that the algorithm can learn from. The quality and quantity of the training data are extremely important since the performance of the machine learning model is dependent upon them.

The engagement of machine learning in autonomous vehicles enables the AV to learn from data and make predictions about the environment around it.

Machine Learning can be categorized into the following types:

- Supervised Learning: The algorithm is trained on a labelled dataset where each data point has a corresponding label or output value, and it learns to make predictions or decisions on new data based on input features. The algorithm learns to associate input features with output labels. For example, in a classification task, the features might be characteristics of objects, and the labels would be the categories those objects belong to.
- Unsupervised Learning: The algorithm is given unlabeled data and must find patterns or structures within it without explicit guidance. That means that data does not have any pre-existing labels. Unsupervised learning must discover patterns in the data without any previous help.
- Reinforcement Learning: The algorithm learns by interacting with an environment. It receives feedback in the form of rewards or penalties based on its actions and adjusts its behavior to maximize cumulative rewards.

Neural networks

Neural networks are a subset of machine learning algorithms that try to emulate the human brain by combining computer science and statistics. They are computational models that consist of a series of nodes representing artificial neurons, that are connected. Each node receives input from other nodes and produces an output, which is then passed on to other nodes in the network. The output of each neuron depends also on two other internal parameters, its weight and threshold and uses activation functions to calculate the output. Neural networks are used for a variety of tasks, including speech recognition, image recognition, and natural language processing.

Deep learning is a subset of machine learning that uses neural networks with multiple layers to learn from large amounts of data. The term “deep” refers to the use of multiple layers in the network.

Convolutional Neural Network (CNN)

A specific type of artificial neural network that is mainly used for image recognition and classification is the Convolutional Neural Network (CNN). CNNs effectively process data with a grid-like structure, such as images, and are especially efficient at pinpointing patterns and features within images.

CNNs are sparsely connected multilayer networks of nodes, with each node performing a specific function in the process of image recognition. A typical architecture of a CNN is a series of convolutional and pooling layers connected in sequence. The convolutional layer extracts features from the input images such as shapes, corners or edges by applying a set of filters and is usually the first layer. Convolutional layers may repeat multiple times. Then a pooling layer follows which produces data with lower dimensions by performing a function called down sampling. Repeating this sequence multiple times enables each subsequent layer to learn more complex features than preceding ones.

4.2 Artificial Intelligence and Computer Vision

Vision-based control in autonomous vehicles utilizing Artificial Intelligence methods has been reviewed in [27]. Vision-based control enables Autonomous Vehicles to identify and avoid obstacles as well as estimate their position.

Measuring distance to objects is also a critical functionality in Autonomous Vehicles. A survey of learning-based methods advancing depth estimation using stereo vision is given in [28].

An algorithm to estimate Stereo matching cost using CNNs is described in [29]. Computing the stereo-matching cost is one of the steps of a stereo-matching algorithm.

Perception and environment analysis are two additional areas where AI methods are involved. In [30], obstacle detection and classification using pre-trained Convolutional Neural Networks are employed and gives the ability to identify the free space for Autonomous vehicle movement.

5 Stereo cameras

As it mentioned in section 2.2 Stereo Cameras are either active or passive depending on what technology they are using to measure depth.

Passive cameras perform the depth measuring task by comparing the left and right image and matching pixel correspondence and to achieve that are using image features. However, these image features are affected by several parameters such as light conditions, the texture of the surfaces in the scene, the complexity of the scene, etc. and this can increase the error probability of the feature matching technique.

Active cameras are using additional projectors that project a light pattern to the scene. In general, this type of cameras is using IR projectors and sensors because they are less susceptible in interference from light sources in the scene thus, they are more reliable. Active stereo matching techniques are however affected by the attributes of the generated pattern. In [31] an evaluation of the performance relationship between the generated pattern and the matching cost of the left and right frames is presented.

5.1 Active Stereo Cameras

Because of substantial active research in this field, many commercial cameras of this type have emerged in the market.

Indicative examples of such cameras (RGB-D) are:

Microsoft Azure Kinect [32], Orbbec 3D Astra series and Femto series [33], and Intel Realsense D400 and L515 series (end of life) [34].

Microsoft Kinect Azure uses Time of Flight Technique to calculate depth, i.e., it emits a signal and measures the time it takes for the signal to be reflected back to the camera. It also includes IMU sensors (gyroscope and accelerometer) for motion sensor.

Orbbec 3D Cameras use structure light technique which project a pattern and estimate depth by comparing the original pattern with the deformed one reflected from the target objects.

Intel RealSense L515 series cameras are utilizing Lidar (also Time of Flight technique) to make depth estimations.

Intel RealSense D400 series use another active stereo method to provide depth measurements. They emit an unstructured static light pattern (dots) and they are comparing the corresponding dot positions in the left and right frame captured by the two Infrared cameras.

In the next section the Intel RealSense Camera is examined in more detail.

5.2 Intel RealSense

In this section, a description of the camera will be given along with the technology behind it as well as the individual components of the camera.

The RealSense D435 is a depth-sensing camera developed by Intel [34] as part of their RealSense product line. It is designed for applications such as 3D scanning, augmented reality, robotics, and computer vision. The technology behind the RealSense D435 involves a combination of hardware and software components for capturing depth information and RGB (color) data simultaneously. It has a wide field of view making it ideal for applications where capturing large portions of the scene is crucial, such as applications involving autonomous vehicles.

All D400 series cameras provided by Intel are active stereo cameras but they have different specifications.

The next figure [Figure 4] shows an inside view of the D435 camera and the different lenses that it employs[7].

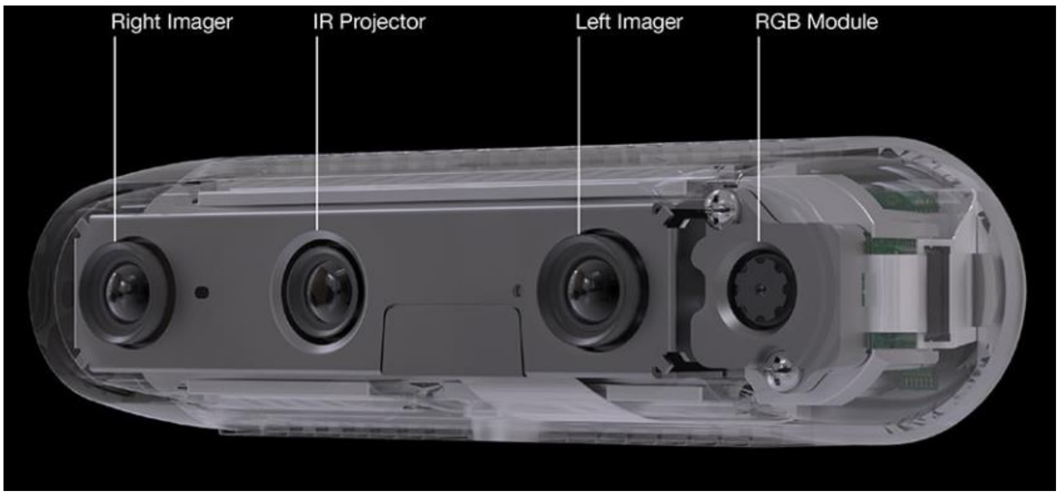


Figure 4: RealSense Internal view[7]

RealSense camera comes with a robust Software Development Kit which enables integration with several platforms such as Nvidia Jetson and Raspberry Pi 3 supporting ROS, MATLAB, Unity and programming languages such as C++, Python and C#.

5.3 How the camera works

The camera uses stereo vision for depth estimation[8]. To achieve that it uses two cameras, i.e. a left and right imager (1280x800 active pixels), and an IR projector to illuminate the object for which depth data are to be collected. According to the camera datasheet the IR projector projects a non-visible static pattern to improve depth accuracy in scenes with low texture. The depth imaging (vision) processor receives data from the left and right images that capture the scene. The processor correlates the points in each image pair to determine the depth value of each pixel. The depth pixel values are processed to generate a depth frame and a sequence of depth frames creates a depth video stream.

Specifically, the D435 model has an additional RGB color sensor (1920x1080 active pixels), providing color image and texture information.

The camera system also employs a Vision Processor D4 to perform stereo vision processing and is located on the Vision Processor D4 board.

[7] Source: https://www.intelrealsense.com/depth-camera-d435/

[8] Source: https://dev.intelrealsense.com/docs/intel-realsense-d400-series-product-family-datasheet

The following picture [Figure 5: D435 block diagram] provides a block diagram of the D435 camera system with all the components[9].

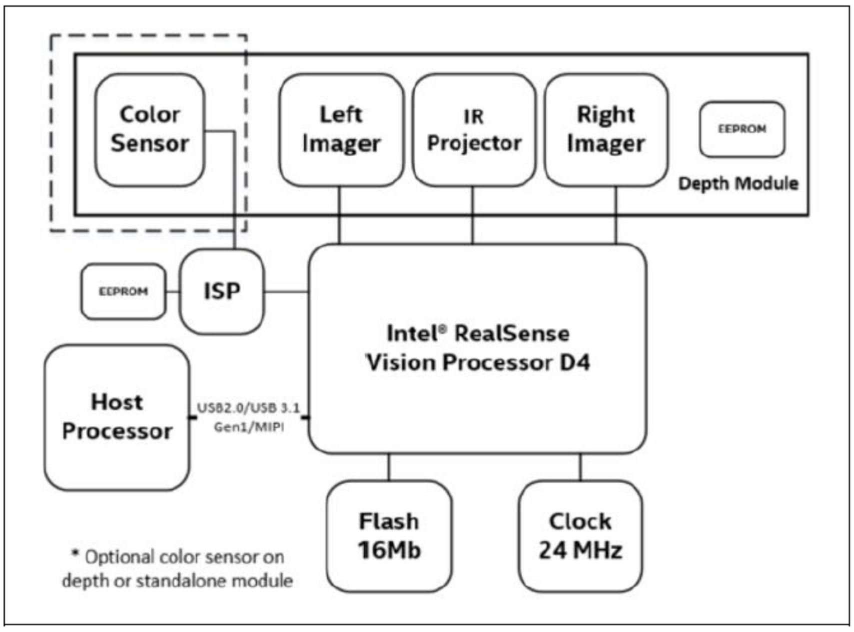


Figure 5: D435 block diagram[9]

The camera communicates with the host through a USB 2.0/USB 3.1 Gen 1 connection.

The two main components of the D435 RealSense camera are shown in the following picture [Figure 6]. These are the D430 depth module with the left and right imager the IR projector, and the D4 board where the Vision processor D4 lives. The RGB sensor is a separate module.

[9] Source: Intel RealSense D400 Series Product Family Datasheet

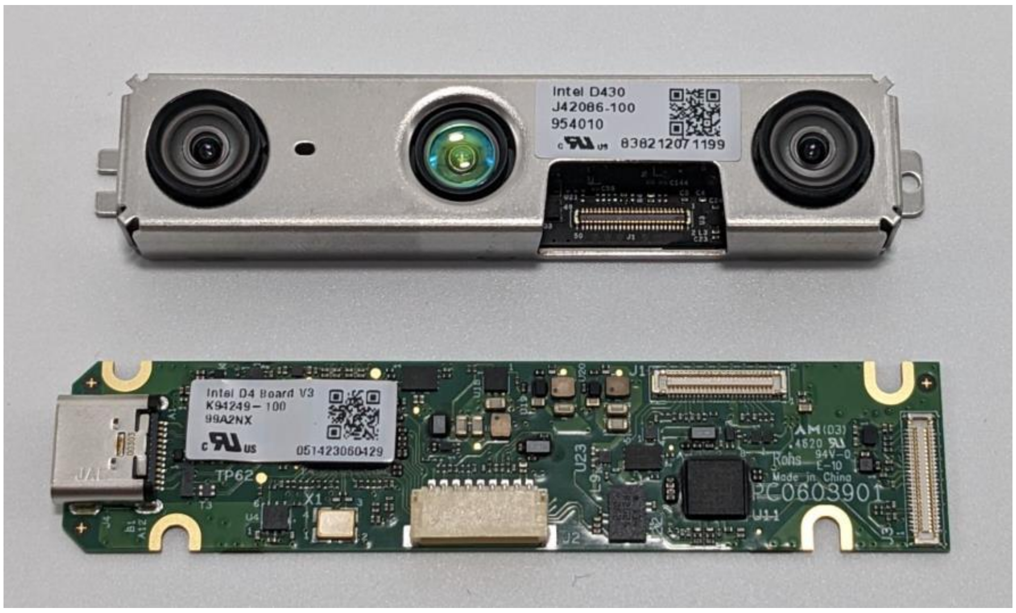


Figure 6: Main components of D435 RealSense[10]

The two components are interconnected with an S-type flat cable (Flex Interposer) through a 50-pin connector [Figure 7][11].

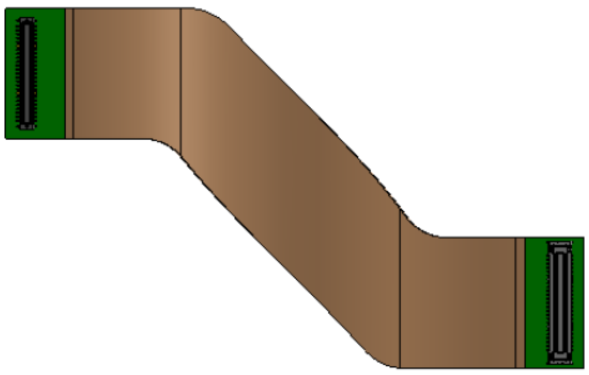


Figure 7: S-type connector cable[11]

In a comparison with other members of the D400 series cameras the following table [**Error! Reference source not found.**][11] shows the specifications of their depth modules.

Table 1: D400 series depth module specifications [11]

Stereo Module	Depth Module D410	Depth Module D415	Depth Module D430	Depth Module D450
Baseline	55 mm	55 mm	50 mm	95 mm

[10] Source: https://store.intelrealsense.com/buy-intel-realsense-depth-module-d430-and-v3-d4-board-bundle.html

[11] Source: Intel RealSense D400 Series Product Family Datasheet

Left/Right Imagers Type	Standard	Standard	Wide	Wide
Depth FOV HD (16:9) (degrees)	H:65 / V:40 / D:72	H:65 / V:40 / D:72	H:87 / V:58 / D:95	H:87 / V:58 / D:95
Depth FOV VGA (4:3) (degrees)	H:50 / V:40 / D:61	H:50 / V:40 / D:61	H:75 / V:62 / D:89	H:75 / V:62 / D:89
IR Projector	Standard	Standard	Wide	Wide
IR Projector FOV	H:67 / V:41 / D:75	H:67 / V:41 / D:75	H:90 / V:63 / D:99	H:90 / V:63 / D:99

It can be observed that D430 and D450 have a wide Field of View (FoV) but it has a smaller baseline than the other models thus making it less accurate.

The IR projector used for the D435 camera is the AMS Princeton Optronics projector [35] which projects around 5000 light dots.

5.4 Accuracy measurements

In this section, we will present a quick attempt to measure the accuracy of the RealSense camera. RealSense Software Development Kit provides a Depth Quality tool[12] which can be used to evaluate some metrics and estimate the quality of the information provided by the camera.

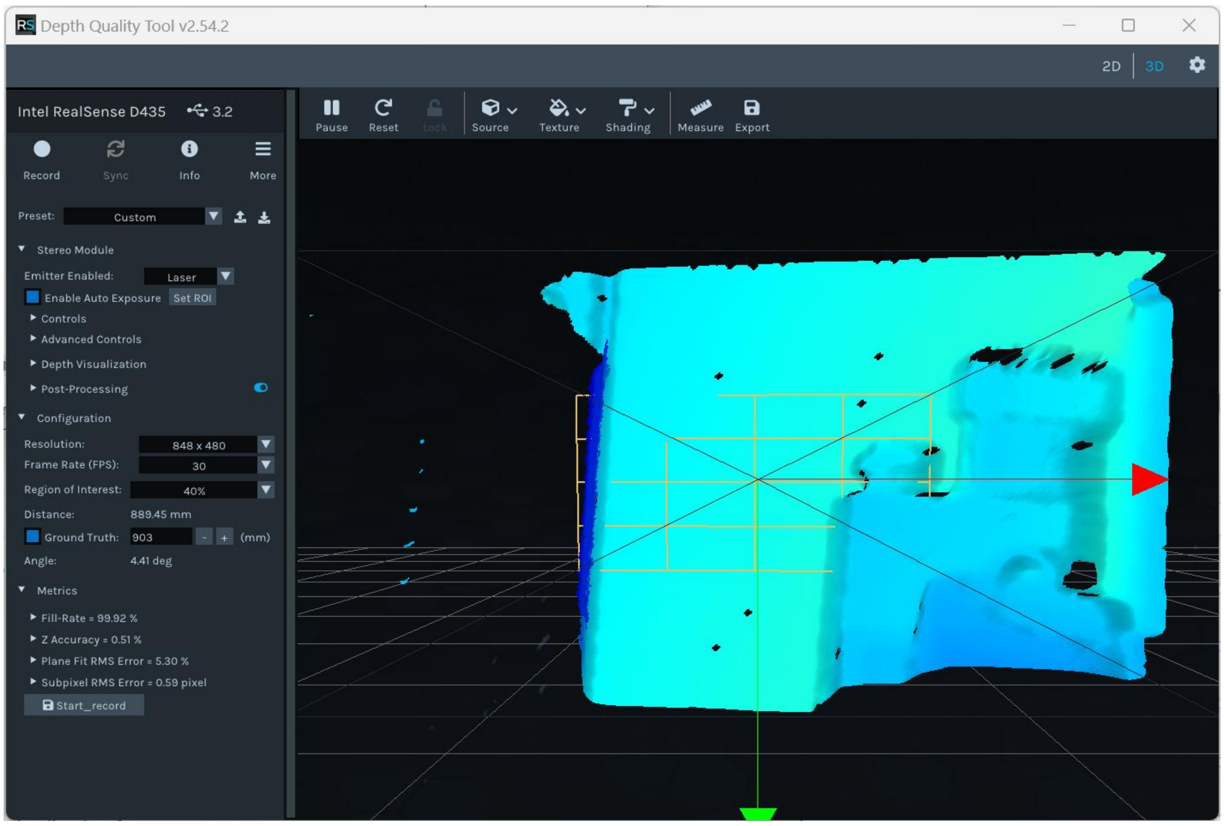
There are several types of metrics that can be evaluated with Depth Quality Tool as explained in a white paper by Intel [36]. Some of them are:

Z-Accuracy: It measures the difference between the depth values that are evaluated from the camera and the actual distances per pixel. A flat surface can be used such as a wall.

Fill Rate: this is the percentage of the depth image that contains valid depth values.

RMS Error: It measures the spatial noise which is the RMS deviation of a best-fit plane and the actual measurements. In the following picture [Figure 8] a tool measurement is shown using as target a white wall. The actual distance between the camera and the wall was 0.903m.

[12] Source: https://github.com/IntelRealSense/librealsense/releases/download/v2.49.0/Depth.Quality.Tool.exe



Intel RealSense D435 3.2

Preset: Custom

Stereo Module

Emitter Enabled: Laser

Enable Auto Exposure Set ROI

Controls

Advanced Controls

Depth Visualization

Post-Processing

Configuration

Resolution: 848 x 480

Frame Rate (FPS): 30

Region of Interest: 40%

Distance: 889.45 mm

Ground Truth: 903 (mm)

Angle: 4.41 deg

Metrics

Fill-Rate = 99.92 %

Z Accuracy = 0.51 %

Plane Fit RMS Error = 5.30 %

Subpixel RMS Error = 0.59 pixel

Figure 8: Depth Quality Tool example

As it can be seen from the Depth Quality Tool report the Fill rate is 99.92%, Z-accuracy 0.51% error, and plane-Fit RMS error is 5.3%.

Now few z-accuracy measurements will be performed without using the provided tool. To achieve the measurements the following setup was used:

An A4 size white paper was placed in several different positions in front of the camera.

The distance from the camera was measured using the **Bosch PLR 50** laser distance measuring tool [Figure 9].

Figure 9: Bosch PLR 50

According to the tool instruction manual, its accuracy is ± 2.0 mm in the operating temperature range of -10 °C to +50 °C. Its measuring distance range is 0.05m to 50m.

In each placement three different distance measurements were taken:

- The actual Distance using the PLR 50
- The reported distance from the camera depth frame in a luminous scene
- The reported distance from the camera depth frame in a dark scene

In each of the above cases, several measurements were taken, and the average measurement was recorded as the final measured distance.

Other cameras and environmental specific parameters are given in the table below.

Table 2: Measurement setup parameters

Setup parameters	
Room temperature (Celsius)	18
Object:	White A4 paper
Camera SW version	2.531.4623
Camera Preset Mode	High Accuracy

A sample snapshot from the IR imager [Figure 10]and the camera depth frame [Figure 11] follows:

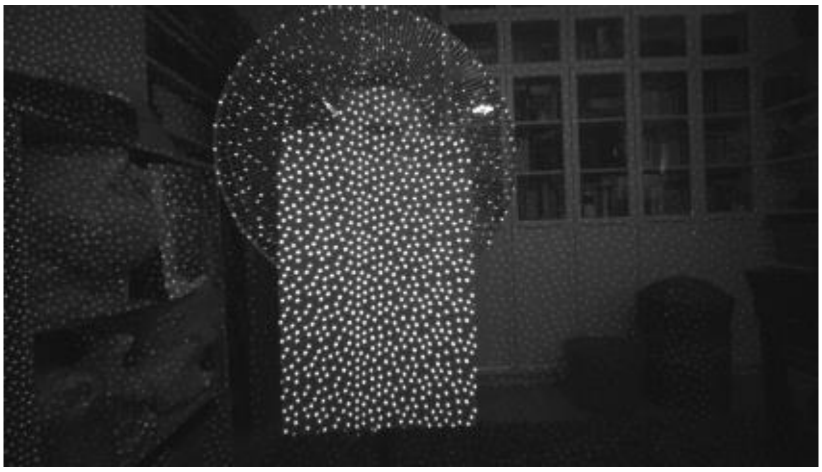


Figure 10: Sample IR frame from RealSense camera

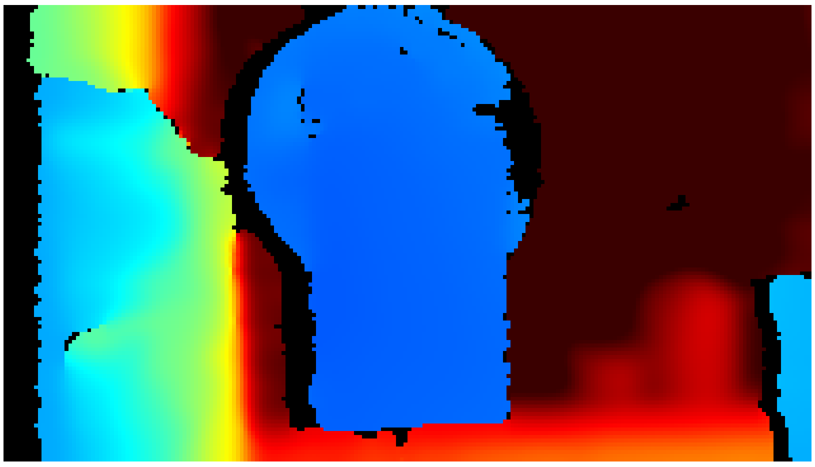


Figure 11: Sample depth frame from RealSense camera

The target A4 white paper is in the centre of the picture frame. In the sample IR frame, the static IR laser pattern projected by the IR projector appears as bright dots. It is used to improve the accuracy of the measurements.

According to the RealSence datasheet, the D435 operating range is 0.2 m to over 3m depending on lighting conditions, so the measurements were taken within the range of 0.2m to 3m.

The measurements recorded are presented in the following table [Table 3]:

Table 3: RealSense distance measurements

Real Distance (m)	**Camera measured distance (luminous scene)**	**Camera measured distance (dark scene)**
0,322	0,304	0,302
0,386	0,367	0,365
0,557	0,534	0,533
0,867	0,869	0,869
1,249	1,291	1,29
1,706	1,801	1,798
2,136	2,304	2,361

2,412	2,618	2,63

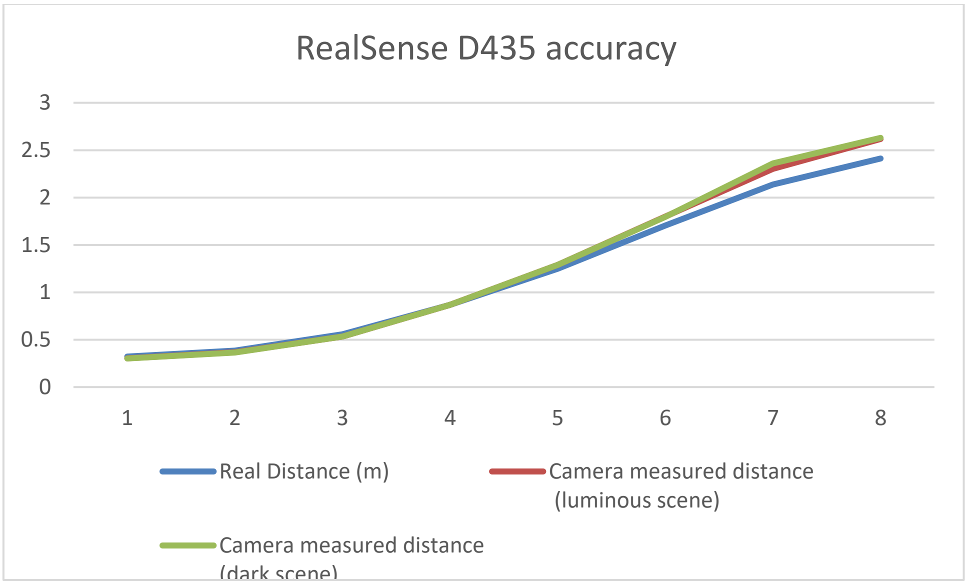


Figure 12: RealSense D435 accuracy

Figure 12 depicts the depth accuracy graph of the D435 camera spanning the 0.2m – 3m region. It is evident that camera is less accurate for distances above 1.5m. The error percentage varies from 0% to 9% and it appears to be the same in both lighting conditions.

5.5 MATLAB integration with Stereo camera

In this section, a simple example will be described on how to utilize a stereo camera with MATLAB.

The camera that was used is the Intel RealSense model 435 [Figure 13] which is an RGB-D type as mentioned in the previous sections.

Figure 13: Intel RealSense model 435

The RealSense camera can provide depth information about the environment in real-time, which can be used in Simulink to model and simulate a variety of dynamic systems, such as robotic arms or drones.

To use a RealSense camera with Simulink, one first needs to install the RealSense SDK and the MATLAB support package for RealSense cameras. Once these packages, have been installed a Simulink model can be created that uses the RealSense camera as a source of input data.

The Simulink model can then be connected to a hardware setup that includes the RealSense camera, along with other devices such as motors or actuators. One can use Simulink to simulate the behavior of the hardware setup and test ones design in real-time.

Camera Calibration with MATLAB

The camera must be calibrated to make available to MATLAB specific camera parameters.

The parameters of a lens, an image sensor, or a video camera are estimated by camera calibration as mentioned in 2.1. Lens distortion can then be eliminated and structures in a scene estimated using these characteristics. Among the camera's parameters are:

Intrinsics: These refer to a camera's internal properties, like the skew coefficient, focal length, and optical center, sometimes referred to as the primary point.

Extrinsics: These explain where the camera is in the three-dimensional scene, including its direction and position.

To achieve this calibration a specific image pattern can be used in order to get references from 3D world points and their corresponding 2D image points. For stereo camera the pattern should not have 180 degree ambiguity. The selected pattern in our case was the checkerboard pattern shown below [Figure 14]:

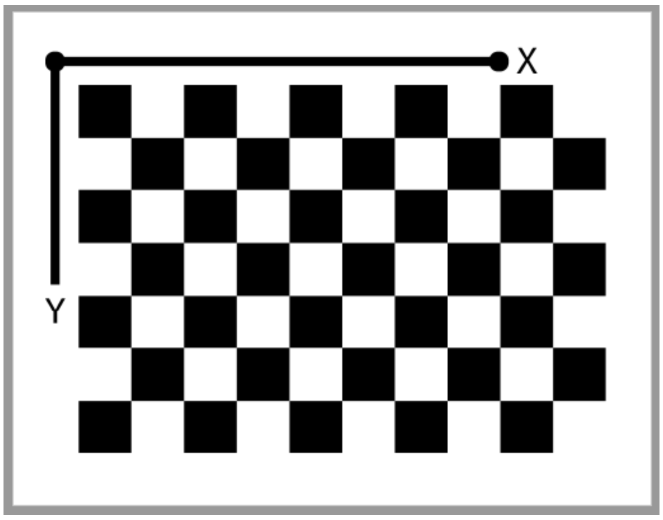


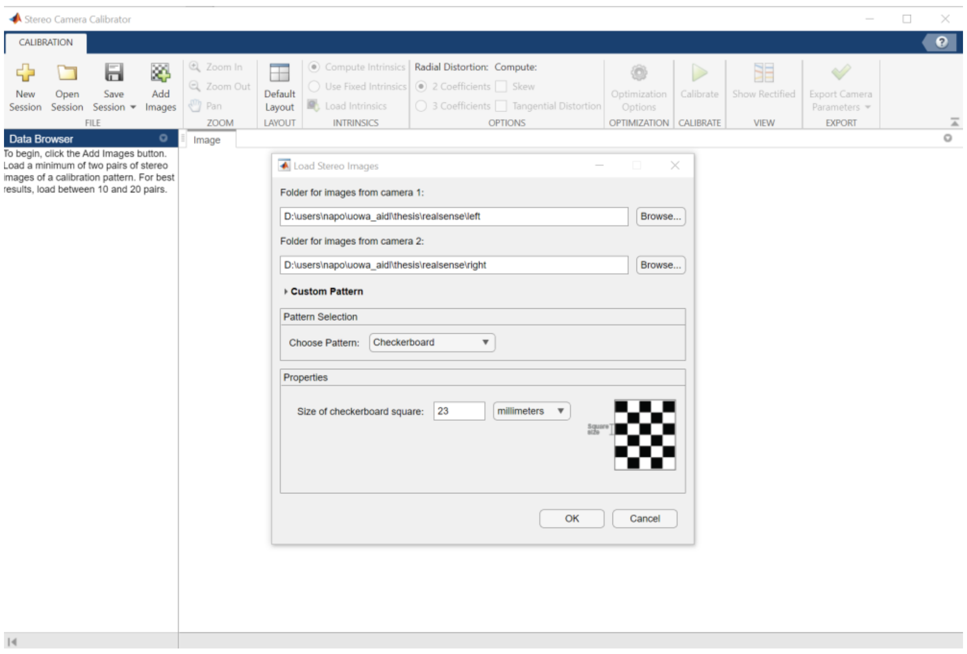
Figure 14: Camera calibration pattern[13]

[13] Source: https://www.mathworks.com/help/vision/ug/calibration-patterns.html

The printed pattern was placed in a flat surface to avoid distortions and several images were captured each one with different orientation of the pattern.

In Appendix A the MATLAB code that was used to capture several frames from left and right camera at the same time is listed.

Then the Computer Vision Toolbox stereoCameraCalibrator app was used to calibrate the camera.



To begin, click the Add Images button. Load a minimum of two pairs of stereo images of a calibration pattern. For best results, load between 10 and 20 pairs.

Folder for images from camera 1: D:\users\napo\uowa_aidl\thesis\realsense\left

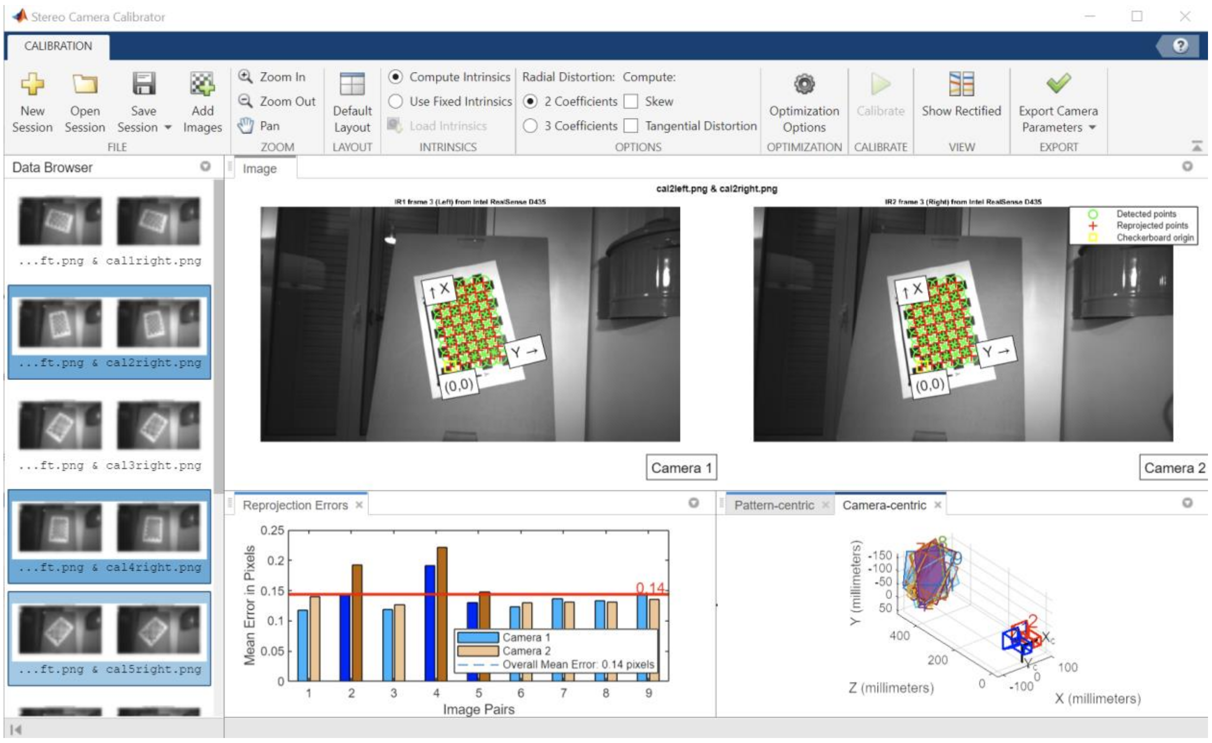
Folder for images from camera 2: D:\users\napo\uowa_aidl\thesis\realsense\right

Choose Pattern: Checkerboard

Size of checkerboard square: 23 millimeters

Figure 15: Loading Patterns in StereoCameraCalibration app

After loading the calibration images the calibration parameters were calculated.



[● Compute Intrinsics] ○ Use Fixed Intrinsics, Load Intrinsics

Radial Distortion: [● 2 Coefficients] ○ 3 Coefficients

Compute: ☐ Skew ☐ Tangential Distortion

Overall Mean Error: 0.14 pixels

Figure 16: MATLAB StereoCameraCalibrator app

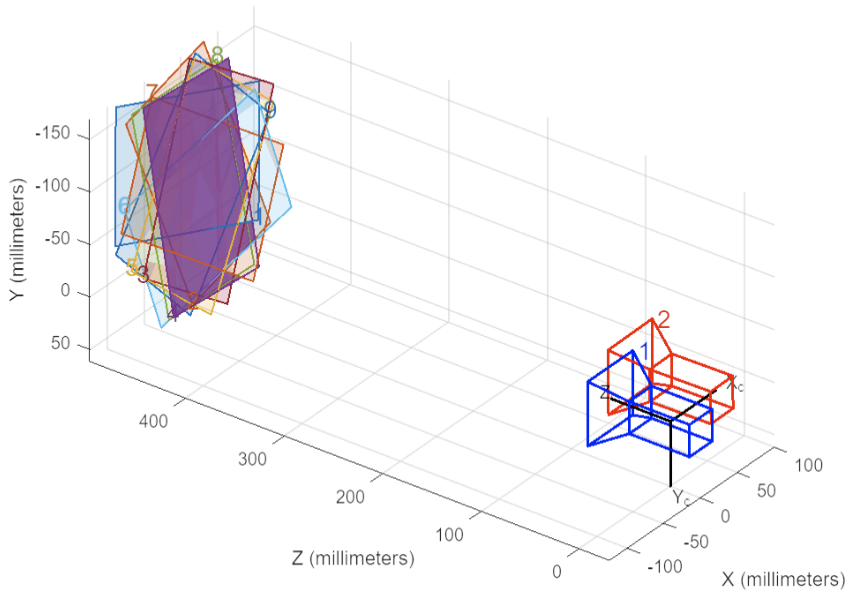


Figure 17: Visualization of Patterns and Camera

Appendix B is a listing of a MATLAB program that was used to provides a video stream from the left or right IR sensor of the camera. The sensor shutters are synchronized so both frames are taken at the same time.

The video stream can then be used to achieve other functions such as object avoidance or track following etc.

Acquiring depth frame with MATLAB

Now using the depth stream provided by the camera we can show an example of capturing the depth information with MATLAB and displaying it as a colorized depth frame.

The result is shown in the picture below [Figure 18]:

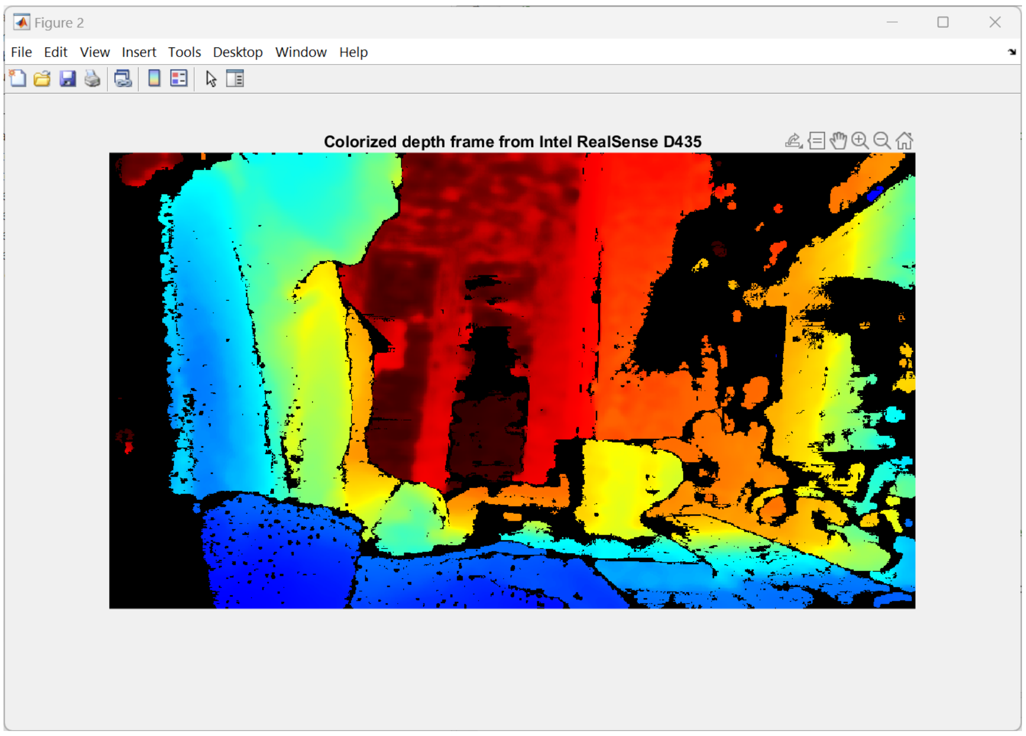


Figure 2
File Edit View Insert Tools Desktop Window Help
Colorized depth frame from Intel RealSense D435

Figure 18: Capturing a depth frame with MATLAB

Appendix C lists the code used to extract the specific depth frame and display it as depicted above.

6 CONCLUSIONS

Given the current development status of the Stereo Vision and the autonomous vehicle problems in the research this thesis reviews some simulation and modelling platforms and their importance in testing new algorithms given their ability to simulate stereo vision sensors. Simulation and modelling platforms provide also urban environment scene construction with dynamic objects and traffic generation reducing risks and costs of testing and evaluation of AVs.

Stereo vision provides results with adequate accuracy for a number of problems including object and obstacle detection, SLAM methods, speed detection of other objects as well as pedestrian detection. The cost of these simple sensors is lower and their size is smaller than LIDARS and RADARS performing similar tasks but vision sensors need improvement in low light and textureless conditions.

Finally, a review of an active RGBD camera was given along with tools provided and few test were performed by integrating the camera into MATLAB environment and testing its performance in bright and dark environments showing same accuracy in both environments.

Bibliography – References – Online sources

1. *R. Fan, J. Jiao, H. Ye, Y. Yu, I. Pitas, and M. Liu, "Key ingredients of self-driving cars," Conference: 27th European Signal Processing Conference (EUSIPCO) 2019.*
2. *SAE levels of driving automation [https://www.sae.org/blog/sae-j3016-update], accessed Dec 2023.*
3. *Yu-Jin Zhang, "3-D Computer Vision,", Springer, 2023.*
4. *Kun Zhou, Xiangxi Meng, Bo Cheng, "Review of Stereo Matching Algorithms Based on Deep Learning", Computational Intelligence and Neuroscience, vol. 2020, Article ID 8562323, 12 pages, 2020. doi: 10.1155/2020/8562323*
5. *Tippetts, B., Lee, D.J., Lillywhite, K. et al. Review of stereo vision algorithms and their suitability for resource-limited systems. J Real-Time Image Proc 11, 5–25 (2016), doi: 10.1007/s11554-012-0313-2*
6. *Ali Tourani, Hriday Bavle, Jose Luis Sanchez-Lopez, Holger Voos, "Visual SLAM: What are the Current Trends and What to Expect?", Oct 2022, arXiv:2210.10491*
7. *X. Jiang and M. Ding, "Unsupervised Monocular Depth Estimation with Scale Unification," 2019 12th International Symposium on Computational Intelligence and Design (ISCID), Hangzhou, China, 2019, pp. 284-287, doi: 10.1109/ISCID.2019.00072.*
8. *Ali Tourani, Hriday Bavle, Jose Luis Sanchez-Lopez, Holger Voos, "Visual SLAM: What are the Current Trends and What to Expect?", Oct 2022, arXiv:2210.10491*
9. *R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," in IEEE Transactions on Robotics, vol. 33, no. 5, pp. 1255-1262, Oct. 2017, doi: 10.1109/TRO.2017.2705103*
10. *B. Gao, H. Lang and J. Ren, "Stereo Visual SLAM for Autonomous Vehicles: A Review," 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Toronto, ON, Canada, 2020, pp. 1316-1322, doi: 10.1109/SMC42975.2020.9283161*
11. *C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel and J. D. Tardós, "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM," in IEEE Transactions on Robotics, vol. 37, no. 6, pp. 1874-1890, Dec. 2021, doi: 10.1109/TRO.2021.3075644*
12. *J. K. Makhubela, T. Zuva and O. Y. Agunbiade, "A Review on Vision Simultaneous Localization and Mapping (VSLAM)," 2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC), Mon Tresor, Mauritius, 2018, pp. 1-5, doi: 10.1109/ICONIC.2018.8601227*
13. *O. Real-Moreno, J. C. Rodríguez-Quiñonez, O. Sergiyenko, W. Flores-Fuentes, P. Mercorelli and L. R. Ramírez-Hernández, "Obtaining Object Information from Stereo Vision System for Autonomous Vehicles," 2021 IEEE 30th International Symposium on Industrial Electronics (ISIE), Kyoto, Japan, 2021, pp. 1-6, doi: 10.1109/ISIE45552.2021.9576262*
14. *A. shakeri, B. Moshiri and H. G. Garakani, "Pedestrian Detection Using Image Fusion and Stereo Vision in Autonomous Vehicles," 2018 9th International Symposium on Telecommunications (IST), Tehran, Iran, 2018, pp. 592-596, doi: 10.1109/ISTEL.2018.8661069.*
15. *U. S., M. N. M., A. Z. Joseph, N. G. N. V. V. S. S. Srinath, C. L. Priyanka and P. Sankaran, "Stereo Vision Based Speed Estimation for Autonomous Driving," 2019 International*

Conference on Information Technology (ICIT), Bhubaneswar, India, 2019, pp. 201-205, doi: 10.1109/ICIT48102.2019.00042
16. *CARLA [https://carla.org/] – the official site of CARLA, accessed Dec 2023*
17. *Dosovitskiy, Alexey & Ros, German & Codevilla, Felipe & López, Antonio & Koltun, Vladlen. (2017). CARLA: An Open Urban Driving Simulator, 1st Conference on Robot Learning (CoRL 2017)*
18. *Unity engine official site [https://unity.com/products/unity-engine], accessed Jan 2024.*
19. *M. Vukić, B. Grgić, D. Dinčir, L. Kostelac and I. Marković, "Unity based Urban Environment Simulation for Autonomous Vehicle Stereo Vision Evaluation," 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 2019, pp. 949-954, doi: 10.23919/MIPRO.2019.8756805*
20. *NVIDIA Drive sim [https://developer.nvidia.com/drive/simulation], accessed Dec 2023.*
21. *NVIDIA DRIVE Constellation [https://www.nvidia.com/content/dam/en-zz/Solutions/self-driving-cars/drive-constellation/nvidia-drive-constellation-datasheet-2019-oct.pdf], accessed Dec 2023*
22. *Gazebo (gazebosim.org)[https://gazebosim.org/home] - the official website of Gazebo*
23. *M. Mustafin, T. Tsoy, E. A. Martínez-García, R. Meshcheryakov and E. Magid, "Modelling mobile robot navigation in 3D environments: camera-based stairs recognition in Gazebo," 2022 Moscow Workshop on Electronic and Networking Technologies (MWENT), Moscow, Russian Federation, 2022, pp. 1-6, doi: 10.1109/MWENT55238.2022.9802368*
24. *Z. Liu, B. Song, Y. Guo and H. Xu, "Improved Template Matching Based Stereo Vision Sparse 3D Reconstruction Algorithm," 2020 Chinese Control And Decision Conference (CCDC), Hefei, China, 2020, pp. 4305-4310, doi: 10.1109/CCDC49329.2020.9164629*
25. *Osório, A., & Pinto, A. (2019). Information, uncertainty and the manipulability of artificial intelligence autonomous vehicles systems. International Journal of Human-Computer Studies. doi:10.1016/j.ijhcs.2019.05.003*
26. *A. Khanum, C. -Y. Lee and C. -S. Yang, "Involvement of Deep Learning for Vision Sensor-Based Autonomous Driving Control: A Review," in IEEE Sensors Journal, vol. 23, no. 14, pp. 15321-15341, 15 July15, 2023, doi: 10.1109/JSEN.2023.3280959*
27. *S. K. Mishra and S. Das, "A Review on Vision Based Control of Autonomous Vehicles Using Artificial Intelligence Techniques," 2019 International Conference on Information Technology (ICIT), Bhubaneswar, India, 2019, pp. 500-504, doi: 10.1109/ICIT48102.2019.00094*
28. *M. Poggi, F. Tosi, K. Batsos, P. Mordohai and S. Mattoccia, "On the Synergies Between Machine Learning and Binocular Stereo for Depth Estimation From Images: A Survey," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 44, no. 9, pp. 5314-5334, 1 Sept. 2022, doi: 10.1109/TPAMI.2021.3070917*
29. *Zbontar, J., & LeCun, Y. (2015). Computing the stereo matching cost with a convolutional neural network. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi:10.1109/cvpr.2015.7298767*
30. *A. Burlacu et al., "Stereo vision based environment analysis and perception for autonomous driving applications," 2018 IEEE 14th International Conference on*

Intelligent Computer Communication and Processing (ICCP), Cluj-Napoca, Romania, 2018, pp. 281-286, doi: 10.1109/ICCP.2018.8516434

31. *Jang, Mingyu & Yoon, Hyunse & Lee, Seongmin & Kang, Jiwoo & Lee, Sanghoon. (2022). A Comparison and Evaluation of Stereo Matching on Active Stereo Images. Sensors. 22. 3332. 10.3390/s22093332.*
32. *Microsoft Azure Kinect DK official site, [https://azure.microsoft.com/en-us/products/kinect-dk], accessed Nov 2023.*
33. *Orbbec cameras official site, [https://shop.orbbec3d.com/camera], accessed Nov 2023.*
34. *Intel Realsense product description, [https://www.intelrealsense.com/depth-camera-d435/], accessed Nov 2023*
35. *A. Grunnet-Jepsen, J. N. Sweetser, P. Winer, A. Takagi and J. Woodfill, "Projectors for Intel RealSense D4xx Series Depth Cameras,", [https://www.intelrealsense.com/wp-content/uploads/2019/03/WhitePaper_on_Projectors_for_RealSense_D4xx_1.0.pdf], accessed Nov 2023.*
36. *RealSense Camera Depth Testing Methodology v1.2, [https://dev.intelrealsense.com/docs/camera-depth-testing-methodology], accessed Nov 2023.*
37. *R. Hamilton, H. Seager, K. P. Divakarla, A. Emadi and S. Razavi, "Modeling and Simulation of an Autonomous-capable Electrified Vehicle: A Review," 2018 IEEE Electrical Power and Energy Conference (EPEC), Toronto, ON, Canada, 2018, pp. 1-7, doi: 10.1109/EPEC.2018.8598294.*

Appendix A

MATLAB code for capturing imager frames from RealSense Camera.

```
%view infrared frames
%https://www.intel.com/content/www/us/en/support/articles/000031703/emerging-technologies/intel-
realsense-technology.html


function capture_frames()
    % Make Pipeline object to manage streaming
    pipe = realsense.pipeline();
    % Make Colorizer object to prettify depth output
    colorizer = realsense.colorizer();


    align_to = realsense.stream.depth; % try to align color frame
    align = realsense.align(align_to);

    % enable infrared stream
    cfg = realsense.config();
    cfg.enable_stream(realsense.stream.color,1280,720);
    cfg.enable_stream(realsense.stream.depth,1280,720);
    cfg.enable_stream(realsense.stream.infrared,1,1280,720);
    cfg.enable_stream(realsense.stream.infrared,2,1280,720);

    % cfg.enable_all_streams();

  % Start streaming on an arbitrary camera with default settings
  profile = pipe.start(cfg);

  % Get streaming device's name
  dev = profile.get_device();
  name = dev.get_info(realsense.camera_info.name);


  sens = dev.first('depth_sensor');
  sens.set_option(realsense.option.emitter_enabled, 0); % or 1 for laser dots

  % Get frames. We discard the first couple to allow
  % the camera time to settle
  for i = 1:5
      fs = pipe.wait_for_frames();
  end

  % Stop streaming
  pipe.stop();
```

```
    aligned_frames = align.process(fs);
    % Select infrared frames
    ir1 = fs.get_infrared_frame(1);
    ir2 = fs.get_infrared_frame(2);
    frame_no = fs.get_frame_number();
    irdata1 = ir1.get_data();
    irdata2 = ir2.get_data();
    ir_img1 = reshape(irdata1, ir1.get_width(), []);
    ir_img2 = reshape(irdata2, ir2.get_width(), []);
    figure, imshow(ir_img1');
    title(sprintf("IR1 frame %d (Left) from %s", frame_no, name));
    figure, imshow(ir_img2');
    title(sprintf("IR2 frame %d (Right) from %s", frame_no, name));


    % Select Color frame
    RGBimage = fs.get_color_frame();
    %RGBimage = aligned_frames.get_color_frame();
    figure,
  imshow(permute(reshape(RGBimage.get_data()',[3,RGBimage.get_width(),RGBimage.get_height()]),[
  3 2 1]));

    % Select depth frame
    depth = fs.get_depth_frame();
    % Colorize depth frame
    color = colorizer.colorize(depth);

    % Get actual data and convert into a format imshow can use
    % (Color data arrives as [R, G, B, R, G, B, ...] vector)

    data = color.get_data();
    img = permute(reshape(data',[3,color.get_width(),color.get_height()]),[3 2 1]);

    % Display image
    figure, imshow(img);
    title(sprintf("Colorized depth frame %d from %s", frame_no, name));
  end
```

Appendix B

The following program was used to get a video stream from the left and right IR camera.

```
% example of streaming video from realsense infrared sensor left and right

% Make Pipeline object to manage streaming
pipe = realsense.pipeline();


% Make Colorizer object to prettify depth output
colorizer = realsense.colorizer();

% define point cloud object
pcl_obj = realsense.pointcloud();

% enable infrared stream
cfg = realsense.config();
%cfg.enable_stream(realsense.stream.color,1280,720);
%cfg.enable_stream(realsense.stream.depth,1280,720);
cfg.enable_stream(realsense.stream.infrared,1,1280,720);
cfg.enable_stream(realsense.stream.infrared,2,1280,720);

% Get streaming device's name
dev = profile.get_device();
name = dev.get_info(realsense.camera_info.name);

% disable (or enable) laser dots
sens = dev.first('depth_sensor');
sens.set_option(realsense.option.emitter_enabled, 0); %or 1 for laser dots

% Start streaming with above defined settings
profile = pipe.start(cfg);

align_to = realsense.stream.color;
alignedFs = realsense.align(align_to);

% Get streaming device's name
dev = profile.get_device();
name = dev.get_info(realsense.camera_info.name);

% Get frames. We discard the first couple to allow
% the camera time to settle

for i = 1:5
    fs = pipe.wait_for_frames();
end

%create a point cloud player
%player1 = pcplayer([-1 1],[-1 1],[-1 1]);

frameCount = 0;
Ifig1 = figure;

while ishandle(Ifig1) && frameCount < 500
```

```
    frameCount = frameCount+1;
    fs = pipe.wait_for_frames();

    %align the depth frames to the color stream

    aligned_frames = alignedFs.process(fs);
    depth = aligned_frames.get_depth_frame();


    %color = fs.get_color_frame();

    % Select infrared frames
    ir1 = fs.get_infrared_frame(1);
    ir2 = fs.get_infrared_frame(2);
    frame_no = fs.get_frame_number();
    irdata1 = ir1.get_data();
    irdata2 = ir2.get_data();
    ir_img1 = reshape(irdata1, ir1.get_width(), []);
    ir_img2 = reshape(irdata2, ir2.get_width(), []);
    set(Ifig1, 'NumberTitle', 'off', 'Name', sprintf("IR1 frame %d (Left) from %s", frame_no, name));

    imshow(ir_img1');
    %title(sprintf("IR1 frame %d (Left) from %s", frame_no, name));

    %get the points cloud based on the aligned depth stream

    %pnts = pcl_obj.calculate(depth);
    %pcl_obj.map_to(color);
    %colordata = color.get_data();

    %colordatavector = [colordata(1:3:end)',colordata(2:3:end)',colordata(3:3:end)'];
    %vertices = pnts.get_vertices();

    %view(player1,vertices,colordatavector)

    %imshow
  end

  disp ("closing")

  % Stop streaming
  pipe.stop();
```

Appendix C

The following MATLAB code was used to capture depth frame information and displaying it as a colorized frame.

```
function depth_D435()
   % Make Pipeline object to manage streaming
   pipe = realsense.pipeline();
   % Make Colorizer object to prettify depth output
   colorizer = realsense.colorizer();

   % Start streaming on an arbitrary camera with default settings
   profile = pipe.start();

   % Get streaming device's name
   dev = profile.get_device();
   name = dev.get_info(realsense.camera_info.name);

   % Get frames. We discard the first couple to allow
   % the camera time to settle
   for i = 1:5
       fs = pipe.wait_for_frames();
   end

   % Stop streaming
   pipe.stop();

   % Select depth frame
   depth = fs.get_depth_frame();
   % Colorize depth frame
   color = colorizer.colorize(depth);

   % Get actual data and convert into a format imshow can use
   % (Color data arrives as [R, G, B, R, G, B, ...] vector)
   data = color.get_data();
   img = permute(reshape(data',[3,color.get_width(),color.get_height()]),[3 2 1]);

   % Display image
   imshow(img);
   title(sprintf("Colorized depth frame from %s", name));
end
```