ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΗΜΑΤΩΝ, ΕΛΕΓΧΟΥ ΚΑΙ ΡΟΜΠΟΤΙΚΗΣ

Development of a Framework for 3D Reconstruction and Inspection of Vineyards

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννης Χ. Ζάρρας

Επιβλέπων : Κωνσταντίνος Τζαφέστας
Αναπληρωτής Καθηγητής, Ε.Μ.Π.

Αθήνα, Μάρτιος 2024

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΗΜΑΤΩΝ, ΕΛΕΓΧΟΥ ΚΑΙ ΡΟΜΠΟΤΙΚΗΣ

Development of a Framework for 3D Reconstruction and Inspection of Vineyards

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννης Χ. Ζάρρας

Επιβλέπων : Κωνσταντίνος Τζαφέστας
Αναπληρωτής Καθηγητής, Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 27η Μαρτίου 2024.

.....	.....	.....
Κωνσταντίνος Τζαφέστας Αναπληρωτής Καθηγητής, Ε.Μ.Π.	Ευάγγελος Παπαδόπουλος Καθηγητής, Ε.Μ.Π.	Πέτρος Μαραγκός Καθηγητής, Ε.Μ.Π.

Αθήνα, Μάρτιος 2024

.....
Ιωάννης Χ. Ζάρρας

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Στην οικογένειά μου

Περίληψη

Η τρισδιάστατη απεικόνιση χώρου είναι ένα ισχυρό εργαλείο που φέρνει ήδη ριζικές αλλαγές στον τομέα της γεωργίας. Χρησιμοποιείται για την απομακρυσμένη παρακολούθηση της ανάπτυξης των καλλιεργειών, την ταυτοποίηση παρασίτων και ασθενειών, ακόμα και την αυτοματοποίηση των εργασιών στον αγρό. Ωστόσο, αποτελεί πρόκληση η διασφάλιση της απαραίτητης οπτικής ποιότητας της απεικόνισης ώστε να μπορεί ο αγρότης να διακρίνει εύκολα λεπτομέρειες στα φυτά.

Η παρούσα διπλωματική εργασία επικεντρώνεται στο σχεδιασμό και την υλοποίηση ενός αποδοτικού συστήματος αντίληψης για τετράποδα ρομπότ, ικανού να ανακατασκευάσει με ακρίβεια ένα αμπέλι παράγοντας σαφές οπτικό περιεχόμενο για τον χρήστη. Το αντικείμενο της παρούσας εργασίας είναι όχι μόνο η έρευνα και η ανάπτυξη του απαραίτητου λογισμικού για την επίτευξη του παραπάνω στόχου, αλλά και η ανασκόπηση του διαθέσιμου υλικού για αυτόν τον σκοπό και τελικά η εκτέλεση ενός πειράματος σε προσομοίωση αλλά και με πραγματικό ρομπότ σε πραγματικές συνθήκες.

Αρχικά εξετάζονται υπάρχοντα πακέτα λογισμικού για τρισδιάστατη ανακατασκευή και χωρική χαρτογράφηση. Η σύγκριση παράγει πολύτιμες πληροφορίες σχετικά με τα οφέλη και τα μειονεκτήματα πρωτοποριακών μεθόδων ανακατασκευής χώρου. Τα πακέτα δοκιμάζονται στο ROS noetic και εφαρμόζεται επιτάχυνση υλικού CUDA, όταν αυτό είναι εφικτό. Επιλέγεται τελικά το εργαλείο χωρικής χαρτογράφησης ZED με υπολογισμό βάθους με νευρωνικά δίκτυα, για να συμπληρώσει τον προσαρμοσμένο αλγόριθμο τρισδιάστατης ανακατασκευής που σχεδιάζεται στο τέλος αυτής της εργασίας.

Στο τρίτο κεφάλαιο της παρούσας διπλωματικής εργασίας μελετώνται αλγόριθμοι πλοήγησης και σχεδιασμού τροχιάς. Ο αλγόριθμος πλοήγησης του ρομπότ που εν τέλει εφαρμόστηκε βασίζεται σε προηγούμενες μελέτες για οπτική οδομετρία με δύο κάμερες που πραγματοποιήθηκαν από την Ομάδα Τετράποδων Ρομπότ του Εργαστηρίου Συστημάτων Ελέγχου του ΕΜΠ, σε έναν πρωτότυπο αλγόριθμο αποφυγής εμποδίων που αποσυνθέτει τον χώρο σε πολύγωνα, καθώς και σε έναν PID ελεγκτή παρακολούθησης τροχιάς.

Στη συνέχεια εξετάζεται η επιλογή των υλικοτεχνικών στοιχείων που θα χρησιμοποιηθούν για την υλοποίηση του συστήματος αντίληψης. Μια σύγκριση αισθητήρων και μονάδων επεξεργασίας, καθώς και μια εξερεύνηση των τρόπων με τους οποίους αυτά μπορούν να συνδυαστούν και να τοποθετηθούν σε ένα τετράποδο ρομπότ, οδήγησε σε καλά τεκμηριωμένες αποφάσεις σχετικά με την τελική δομή του συστήματος. Για την επίτευξη των συγκρίσεων αναπτύχθηκε ένα εργαλείο τρισδιάστατης απεικόνισης πεδίου θέασης και ανάλυσης lidar (FoVaLiRa) στην πλατφόρμα ανάπτυξης Unity.

Το τελευταίο κεφάλαιο επικεντρώνεται στην ανάπτυξη του αλγορίθμου Vinymap για αντικειμενική αξιολόγηση ποιότητας point cloud, τρισδιάστατη απεικόνιση αμπελιών και αξιολόγηση πυκνότητας φυλλώματος. Το Vinymap είναι μια καινοτόμος διεργασία και δοκιμάστηκε σε κινητό ρομπότ που φέρει το σύστημα αντίληψης που δημιουργήθηκε στα πλαίσια αυτής της εργασίας. Είναι γραμμένο σε python3, με χρήση δύο γνωστών και καθιερωμένων βιβλιοθηκών ανοιχτού κώδικα, των Open3D και OpenCV.

Λέξεις Κλειδιά – Γεωργικά Ρομπότ, Ρομποτική Αντίληψη, Τρισδιάστατη Ανακατασκευή Χώρου, Πλοήγηση και Έλεγχος Κινούμενων Ρομπότ, Σχεδιασμός Τροχιάς

Abstract

3D spatial mapping is a powerful tool that is revolutionizing the field of agriculture. By creating detailed digital representations of fields, farmers can gain insights into their land that were previously unattainable. This technology can be used to remotely monitor crop growth, identify pests and diseases, optimize irrigation and fertilizer applications, and even automate field operations. However, it is a challenge to ensure the necessary optical quality so that the farmer can easily distinguish details in the plants.

The present thesis focuses on the design and implementation of an effective perception system capable of accurately reconstructing a vineyard while producing clear and comprehensive visual content for the user. The subject of the present work is not only the research and development of the necessary software to achieve the above goal, but also a review of the available hardware for that purpose and ultimately the execution of an experiment in simulation, as well as with a real robot in realistic conditions.

First, the available software on 3D reconstruction and spatial mapping is reviewed. Comparing four different packages produces valuable insight regarding the benefits and drawbacks of various state-of-the-art reconstruction methods. The packages are tested on ROS noetic and CUDA hardware acceleration is enabled to speed up the process. The ZED spatial mapping tool with neural depth capabilities is chosen to complement the custom 3D reconstruction algorithm that is ultimately implemented in this work.

Navigation and path planning algorithms are studied in the third chapter of the present thesis. The final design of the high-level planner which is utilized in both the simulation and the real-life experiments is based on prior studies on dual camera visual odometry conducted by the Legged Robots Team of the Control Systems Lab in NTUA, as well as on a novel obstacle avoidance algorithm which leverages polytopic decomposition and a stable trajectory tracking PID controller.

Deciding which hardware components to utilize for the implementation of the perception system is next addressed in this work. A comparison of sensors and processing units, as well as an exploration of the ways in which they can be combined and placed on a quadruped robot, led to well-argued decisions about the final structure of the perception system. To justify and visualize these comparisons, a three-dimensional field of view visualization and lidar resolution analysis tool (FoVaLiRa) was developed in the Unity Development platform.

The final chapter focuses on the development of the Vinymap Objective Quality Assessment, Canopy Inspection and 3D Reconstruction Algorithm. Vinymap is a novel approach to reconstructing a real vineyard while maintaining the visual features of the leaves, the grapes, and the trunk intact. It also assesses the vineyard’s canopy density and provides valuable quantitative indexes to the farmers. Vinymap was tested on a mobile robotic platform with the perception system developed in this work. It is written in python3 and utilizes open3D and openCV, two well-known and well-established open-source libraries.

Keywords – Agricultural Robots, Robotic Perception, 3D Reconstruction, Robotic Navigation and Control, Path Planning

Acknowledgements

First and foremost, I would like to express my gratitude to God, for staying with me through the thick and thin.
I would like to express my deep gratitude to Professor Evangelos Papadopoulos for opening the door to the world of robotics. His guidance, expertise, and insightful advice were instrumental in shaping this thesis. I am also incredibly grateful to PhD candidate Athanasios Mastrogeorgiou for sharing his knowledge right from the outset and being a mentor throughout this journey. My sincere thanks to PhD candidates Konstantinos Koutsoukis and Konstantinos Machairas for their encouragement and support. Their ability to foster a welcoming lab environment made this experience even more enriching. I would like to acknowledge the significant contributions of Christos Kokkas. His valuable work played a crucial role in this research. I would like to thank Dimitrios Zarras, my brother and colleague, for being a constant source of guidance and support as I navigated unfamiliar territory. I am deeply grateful to Nikoletta Papageorgiou. Her selfless assistance and insightful remarks improved the quality of my work, while her encouragement made it easier.
Finally, I would like to thank my family. Their encouragement, love and support will always be unique and irreplaceable.

Ioannis Zarras
March 2024

Table of Contents

Λίστα Εικόνων

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Definition
API	Application Programming Interface
BPA	Ball Pivoting Algorithm
CDI	Canopy Density Index
CPU	Central Processing Unit
CSL	Control Systems Laboratory
CPP	Coverage Path Planning
CUDA	Compute Unified Device Architecture
CVPR	Computer Vision and Pattern Recognition
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
FLOPS	Floating-point Operations per Second
FOV	Field Of View
GMSL2	Gigabit Multimedia Serial Link 2
GPS	Global Positioning System
GPU	Graphics Processing Unit
HDMI	High-Definition Multimedia Interface
ICP	Iterative Closest Point
IEEE	Institute of Electrical and Electronics Engineers
IMU	Inertial Measurement Unit
LSTM	Long Short-Term Memory Model
MATLAB	Matrix Laboratory
MAV	Micro air vehicle
MCD	Maximum Correspondence Distance
MEMS	Microelectromechanical Systems
MPC	Model Predictive Controller
NTUA	National Technical University of Athens
NUC	Next Unit of Computing
OS	Operating System

PCQA	Point Cloud Quality Assessment Algorithm
PID	Proportional Integral Derivative (Controller)
QR	Quick Response (Code)
RAM	Random Access Memory
RGB	Red, Green, Blue
ROS	Robot Operating System
RP	Robotic Platform
RRT	Rapidly exploring Random Tree
RTAB-map	Real-Time Appearance-Based Mapping
SBC	Single Board Computer
SLAM	Simultaneous Localization and Mapping
SOPCQA	Simple Objective Point Cloud Quality Assessment
SOTA	State of the Art
TFLOP	Trillion Floating-point Operations per Second
TOPS	Trillions of Operations per Second
TPU	Tensor Processing Unit
UAV	Unmanned Aerial Vehicle
USB	Universal Serial Bus
UV	Ultra-Violet
VRAM	Video Random Access Memory
VSLAM	Visual Simultaneous Localization and Mapping

1 Εκτεταμένη Ελληνική Περίληψη

1.1 Εισαγωγή

Ο αυξανόμενος πληθυσμός της Γης και η συνεχής κλιματική αλλαγή αποτελούν σημαντική πρόκληση για τη διαθεσιμότητα τροφίμων παγκοσμίως. Οι πρακτικές της γεωργίας ακριβείας (Precision Agriculture) και η αυτοματοποιημένη επιθεώρηση των καλλιεργειών έχουν αναδειχθεί ως βασικές στρατηγικές για να καταστεί η γεωργία πιο αποδοτική και βιώσιμη. Ένας κρίσιμος τομέας είναι οι αμπελώνες, λόγω της δομημένης διάταξής τους και της υψηλής αξίας των παραγόμενων προϊόντων [1].

Ωστόσο, πρέπει να αντιμετωπιστούν αρκετές προκλήσεις πριν μπορέσουν τα ρομπότ να ενσωματωθούν πλήρως στη διαχείριση των αμπελώνων. Ένα σημαντικό εμπόδιο είναι το ασθενές δορυφορικό σήμα στις ορεινές περιοχές με αμπελώνες. Αυτό καθιστά αναξιόπιστα τα Παγκόσμια Συστήματα Στιγματοθέτησης (GPS), ένα ουσιαστικό στοιχείο για την πλοήγηση των ρομπότ. Πολλές υπάρχουσες λύσεις βασίζονται σε μεγάλο βαθμό στο GPS [2], καθιστώντας τις αναποτελεσματικές σε περιβάλλοντα χωρίς ευκρινές σήμα [3].

Μια επιπλέον πρόκληση δημιουργεί το ανώμαλο έδαφος μέσα στους αμπελώνες αλλά και τα εμπόδια όπως βράχοι, εξοπλισμός άρδευσης και εργαζόμενοι. Αυτά καθιστούν αναγκαία την ανάπτυξη αποδοτικών και ευέλικτων λύσεων σχεδιασμού διαδρομών. Επιπλέον, η αποδοτικότητα κόστους παραμένει κρίσιμο ζήτημα όσον αφορά την ευρεία υιοθέτηση τέτοιων συστημάτων από γεωργούς.

Πολλές υπάρχουσες λύσεις αντιμετωπίζουν συγκεκριμένες πτυχές του αυτοματισμού, όπως ο ψεκασμός [4], η καταμέτρηση σταφυλιών [5] ή η ανίχνευση ασθενειών [6], στερούμενες μιας ολιστικής προσέγγισης. Οι τελευταίες δεν προσφέρουν μια ολοκληρωμένη τρισδιάστατη απεικόνιση των αμπελιών, η οποία είναι πολύτιμη για τον έλεγχο των καλλιεργειών. Επίσης, η έρευνα συχνά παραμελεί τα κρίσιμα ζητήματα της αυτόνομης πλοήγησης και της αποφυγής εμποδίων μέσα στον αμπελώνα [7], [8]. Αυτός ο κατακερματισμός περιορίζει την αποτελεσματικότητά τους στο να παρέχουν μια πλήρη εικόνα στον αγρότη.

Πρόσφατη έρευνα καταδεικνύει τη δυνατότητα δημιουργίας φωτο-ρεαλιστικών τρισδιάστατων χαρτών που χρησιμοποιούν τρισδιάστατες Γκαουσσιανές [9]. Ενώ είναι πολύτιμες, τέτοιες λύσεις απαιτούν συνήθως τεράστια υπολογιστική ισχύ, καθιστώντας τες απαγορευτικά ακριβές για πολλούς καλλιεργητές και εμποδίζοντας την ικανότητα εκτέλεσης σε πραγματικό χρόνο, η οποία είναι ουσιαστική καθώς η γρήγορη λήψη αποφάσεων είναι ζωτικής σημασίας για τους αγρότες προκειμένου να αντιδράσουν σε ξαφνικά καιρικά φαινόμενα ή καταστροφές [10].

Δεδομένων αυτών των περιορισμών, η παρούσα διπλωματική παρουσιάζει μια ολιστική λύση για το πρόβλημα της επιθεώρησης αμπελώνων με κινητά ρομπότ. Στα πλαίσιά της αναπτύχθηκε λογισμικό και υλικό και εφαρμόστηκε σε προσομοίωση αλλά και σε πραγματικές συνθήκες με την τροποποίηση και χρήση μιας ρομποτικής πλατφόρμας.

Αρχικά πραγματοποιήθηκε εις βάθος έρευνα και αξιολόγηση υφιστάμενων πακέτων λογισμικού, με έμφαση στις δυνατότητες τρισδιάστατης ανακατασκευής και χωρικής χαρτογράφησης. Η σύγκριση εστίασε στην αποσαφήνιση των πλεονεκτημάτων και

μειονεκτημάτων κάθε μεθόδου, λαμβάνοντας υπόψη κριτήρια όπως η ακρίβεια, η ταχύτητα, η ευκολία χρήσης και η συμβατότητα με το σύστημα ROS.

Επιπρόσθετα, αξιοποιήθηκε η τεχνολογία CUDA για την επιτάχυνση της επεξεργασίας, όπου αυτό ήταν εφικτό. Η λεπτομερής αξιολόγηση κατέληξε στην επιλογή του εργαλείου ZED spatial mapping για χωρική χαρτογράφηση και υπολογισμό βάθους με νευρωνικά δίκτυα. Ο αλγόριθμος αυτός κρίθηκε ως η πλέον κατάλληλη λύση για την ενίσχυση του προσαρμοσμένου αλγορίθμου τρισδιάστατης ανακατασκευής που αναπτύχθηκε στα πλαίσια της παρούσας εργασίας.

Για την πλοήγηση της ρομποτικής πλατφόρμας στον χώρο χρησιμοποιείται ένα σύστημα οπτικής οδομετρίας με διπλή κάμερα, σχεδιασμένο ειδικά για περιβάλλοντα χωρίς σήμα GPS αλλά και περιβάλλοντα με οπτική ομοιομορφία που σημαίνει την έλλειψη πληθώρας διαθέσιμων οπτικών χαρακτηριστικών. Η οπτική οδομετρία με διπλή κάμερα εξασφαλίζει τη συνεχή λειτουργία της πλοήγησης ακόμα και όταν η μία κάμερα τυφλώνεται από το ηλιακό φως ή ένα εμπόδιο κρύβει το οπτικό της πεδίο. Η αποτελεσματικότητα του αλγορίθμου βελτιώθηκε με τροποποίηση του κλεισίματος βρόχου με την ανίχνευση ετικετών April Tag, προσθέτοντας έναν αλγόριθμο οπτικού σερβο-ελέγχου που εξασφαλίζει την βέλτιστη ευθυγράμμιση του ρομπότ με την ετικέτα April Tag πριν αυτό ξεκινήσει τις διορθώσεις κλεισίματος βρόχου. Επιπλέον, μελετήθηκε και αξιοποιήθηκε ένας αλγόριθμος σχεδιασμού διαδρομών που χρησιμοποιεί αποσύνθεση του χώρου σε πολύγωνα για να εντοπίσει διαδρόμους χωρίς εμπόδια. Αυτή η προσέγγιση, που εφαρμόζεται για πρώτη φορά σε πραγματικό ρομπότ, επωφελείται από τη διάταξη του αμπελώνα που εκ φύσεως σχηματίζει παράλληλους διαδρόμους.

Εν συνεχεία σχεδιάστηκε το σύστημα αντίληψης με στόχο να φιλοξενήσει τους επιλεγμένους αλγορίθμους τρισδιάστατης χαρτογράφησης, πλοήγησης και σχεδιασμού τροχιάς. Η υλοποίηση του συστήματος αντίληψης προϋποθέτει την επιλογή κατάλληλων υλικοτεχνικών στοιχείων. Πραγματοποιήθηκε λεπτομερής σύγκριση αισθητήρων (κάμερες, LiDAR, GPS) και μονάδων επεξεργασίας, λαμβάνοντας υπόψη κριτήρια όπως η ακρίβεια, η εμβέλεια, η ανάλυση, το κόστος, η ισχύς επεξεργασίας και η κατανάλωση ενέργειας. Εξετάστηκαν διάφορες διατάξεις τοποθέτησης των αισθητήρων σε τετράποδα ρομπότ, λαμβάνοντας υπόψη παράγοντες όπως το πεδίο θέασης, το βάρος και την προστασία από κρούσεις.

Για την υποστήριξη της λήψης τεκμηριωμένων αποφάσεων, αναπτύχθηκε το FoVaLiRa (Field of View and LiDAR Analysis), ένα εργαλείο τρισδιάστατης απεικόνισης στην πλατφόρμα Unity. Το FoVaLiRa βοήθησε στην οπτικοποίηση του πεδίου θέασης κάθε αισθητήρα, στην αξιολόγηση της επικάλυψης πεδίων θέασης και στην εξέταση της ανάλυσης LiDAR σε διάφορες αποστάσεις.

Τέλος, αναπτύχηκε το καινοτόμο λογισμικό Vinymap. Το Vinymap παρέχει ολοκληρωμένη επιθεώρηση του αμπελώνα μέσω δύο βασικών αλγορίθμων: 1) Έναν αλγόριθμο φωτορεαλιστικής 3D χαρτογράφησης που δημιουργεί σαφείς οπτικές απεικονίσεις του αμπελώνα. 2) Έναν αλγόριθμο αξιολόγησης της πυκνότητας του φυλλώματος που παρέχει ζωτικής σημασίας πληροφορίες που σχετίζονται με την υγεία και την ποιότητα των σταφυλιών, καθώς το πυκνό φύλλωμα μπορεί να περιορίσει το ηλιακό φως και τον αερισμό γύρω από τα τσαμπιά σταφυλιών [11]. Η αποτελεσματικότητα του Vinymap αξιολογήθηκε αρχικά σε προσομοιώσεις. Στη συνέχεια, κατασκευάσαμε έναν συνθετικό αμπελώνα στο Εργαστήριο Αυτομάτου Ελέγχου της Σχολής Μηχανολόγων Μηχανικών του

Ε.Μ.Π. για να δοκιμάσουμε εκτενώς το πλαίσιο, εφαρμοσμένο πλέον σε μια πραγματική ρομποτική πλατφόρμα υπό ρεαλιστικές συνθήκες.

1.2 Έρευνα και Αξιολόγηση Πακέτων Λογισμικού Χαρτογράφησης και Ανακατασκευής Χώρου

Εξετάστηκαν τέσσερα λογισμικά ανακατασκευής χώρου πρώτο ήταν το RTAB-Map [12] το οποίο εκτελεί τρισδιάστατη απεικόνιση χώρου σε πραγματικό χρόνο εφαρμόζοντας αλγορίθμους SLAM. Το δεύτερο ήταν το Robot-Centric Elevation Mapping [13], ανεπτυγμένο από την εταιρεία Anybotics που εδρεύει στο ETH της Ζυρίχης και το οποίο εκτελεί 2.5D mapping ή αλλιώς ψευδό-τρισδιάστατη απεικόνιση χώρου. Και αυτό το πακέτο λειτουργεί σε πραγματικό χρόνο αλλά δεν αποτελεί μια εφαρμογή SLAM καθώς εκτελεί μόνο χαρτογράφηση του περιβάλλοντος χώρου χωρίς να επιτυγχάνει εντοπισμό ή πλοήγηση. Το τρίτο πακέτο είναι το Gradslam [14] το οποίο πραγματοποιεί τρισδιάστατη απεικόνιση χώρου και εφαρμόζει αλγόριθμους SLAM. Το τέταρτο πακέτο είναι το ZED Spatial Mapping [15] το οποίο επίσης εκτελεί τρισδιάστατη απεικόνιση χώρου σε πραγματικό χρόνο όντας μια εφαρμογή SLAM.

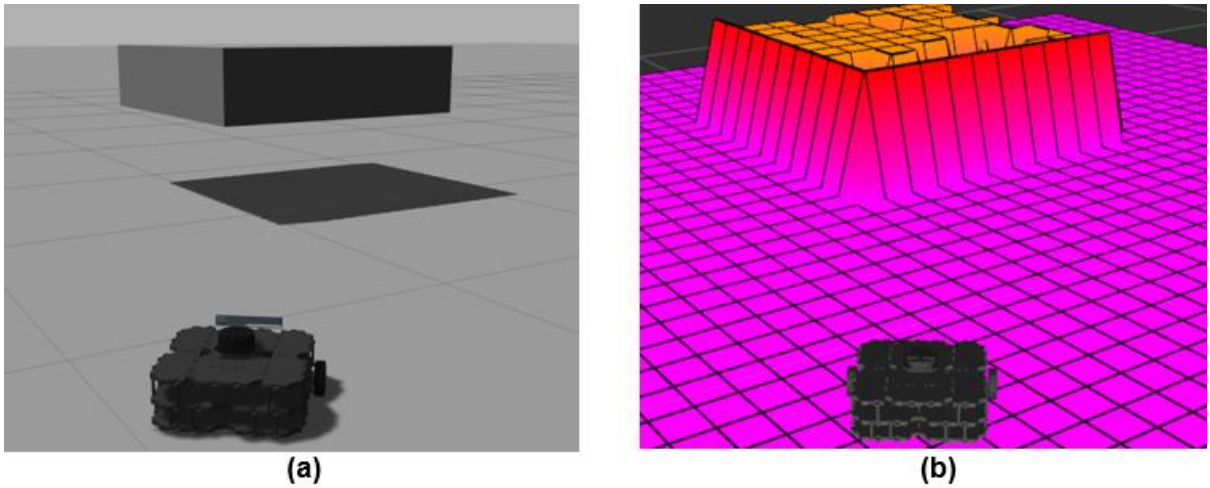
1.2.1 Rtabmap

Το Rtabmap εφαρμόζει οπτική οδομετρία σε πραγματικό χρόνο, δηλαδή λαμβάνει διαδοχικές εικόνες καθώς το ρομπότ που το χρησιμοποιεί κινείται στο χώρο και συγκρίνοντας τα οπτικά χαρακτηριστικά μεταξύ διαδοχικών εικόνων εκτιμάει την κατεύθυνση και το μέτρο της κίνησης του ρομπότ. Αξιοποιώντας τα δεδομένα της οπτικής οδομετρίας, το ρομπότ εκτιμάει τη θέση του στο χώρο και αξιοποιώντας έναν αισθητήρα τρισδιάστατης απεικόνισης που επιλέγει ο χρήστης, καταφέρνει να ανακατασκευάσει τον χώρο γύρω του δίδοντας ως έξοδο ένα τρισδιάστατο νέφος σημείων (3D point cloud). Ιδιαίτερο χαρακτηριστικό του RTAB-Map αποτελεί η προσέγγιση Bag-of-Words για την επίτευξη κλεισίματος βρόχου (loop closure). Αυτή η διαδικασία λειτουργεί ως εξής: ανά προκαθορισμένα χρονικά διαστήματα το ρομπότ αξιοποιεί κάποιες εικόνες και αποθηκεύει τα οπτικά χαρακτηριστικά τους σε μια βάση δεδομένων με τη μορφή μιας τσάντας (bag) για να τα χρησιμοποιήσει μελλοντικά. Καθώς το ρομπότ συλλέγει πληθώρα τέτοιων τσαντών, ελέγχει για κάθε μία από τις τσάντες που βρίσκει αν εμφανίζει έντονη ομοιότητα με κάποια που έχει συλλέξει παλιότερα. Αν πράγματι βρει μεγάλη ομοιότητα μεταξύ της τρέχοντος τσάντας και μιας παλιότερης, τότε θεωρεί ότι βρίσκεται σε ένα σημείο που έχει ξανά επισκεφθεί και με τον τρόπο αυτό αν η εκτίμηση της θέσης του εμφανίζει απόκλιση, τότε την διορθώνει αναδρομικά μειώνοντας έτσι τα σφάλματα των μετρήσεων που συσσωρεύονται κατά την διάρκεια της κίνησής του.

1.2.2 Robot-Centric Elevation Mapping

Το Robot-Centric Elevation Mapping έχει ως σκοπό τη χαρτογράφηση του περιβάλλοντος χώρου γύρω από το ρομπότ. Καθώς το ρομπότ προχωράει στο χώρο, μέρη του χάρτη που έχει επισκεφθεί προηγουμένως διαγράφονται και ο χάρτης ουσιαστικά ακολουθεί το ρομπότ. Η προσέγγιση αυτή είναι γνωστή ως robot-centric mapping και συμβάλλει στην εξοικονόμηση μνήμης του υπολογιστή του ρομπότ, ενώ διευκολύνει την εκτέλεση του αλγορίθμου σε πραγματικό χρόνο.

Το Robot-Centric Elevation Mapping φτιάχνει έναν ψευδό-τρισδιάστατο χάρτη δηλαδή δεν έχει ως έξοδο ένα νέφος με σημεία στις τρεις διαστάσεις, αλλά έναν δισδιάστατο χάρτη όπου για κάθε σημείο αυτού του χάρτη δίνει μια τιμή ύψους (elevation map). Αυτό έχει ως αποτέλεσμα εμπόδια τα οποία βρίσκονται σε κάποιο ύψος και απέχουν από το έδαφος, να καταχωρούνται ως συμπαγή εμπόδια που εκτείνονται από το έδαφος μέχρι το μέγιστο ύψος στο οποίο φτάνουν. Αυτό δεν αποτελεί μια αντικειμενική απεικόνιση του χώρου και δημιουργεί ιδιαίτερο πρόβλημα σε περιβάλλοντα αμπελώνων όπου τα κλαδιά των αμπελιών συχνά βρίσκονται σε ύψος μεγαλύτερο από το ρομπότ οπότε θεωρητικά επιτρέπουν την κίνηση του ρομπότ κάτω από αυτά. Στην περίπτωση του Robot-Centric Elevation Mapping θα καταχωρούνταν ως απροσπέλαστα εμπόδια πράγμα που δεν είναι αληθές.



Εικόνα 1-1: (a) Μια πλατφόρμα που ίπταται, σε περιβάλλον προσομοίωσης Gazebo. (b) Η απεικόνιση της πλατφόρμας και του γύρω χώρου με τη χρήση του Robot-Centric Elevation Mapping.

Ιδιαίτερο χαρακτηριστικό αυτού του πακέτου αποτελεί η δυναμική αναπροσαρμογή του χάρτη. Αν ένα εμπόδιο που έχει καταχωρηθεί στο χάρτη είναι κινούμενο και άλλαξε η θέση του, ο χάρτης αναπροσαρμόζεται καθώς γίνεται χρήση ray casting για να γίνει αντιληπτό ότι η θέση που καταλάμβανε προηγουμένως το εμπόδιο είναι πλέον κενή και το ίδιο έχει κινηθεί σε μια νέα θέση.

1.2.3 Gradslam

Το Gradslam είναι ένα πακέτο το οποίο δημιουργήθηκε με σκοπό να τροποποιήσει γνωστούς και καλά εδραιωμένους αλγορίθμους που εφαρμόζονται στη διαδικασία του SLAM, έτσι ώστε να μπορούν να εισαχθούν σε μοντέλα μηχανικής μάθησης και να γίνει εκπαίδευση αυτών των μοντέλων απευθείας με δεδομένα εφαρμογών SLAM. Όλο το πακέτο λοφισμικού Gradslam αποτελεί ένα διαφορίσιμο υπολογιστικό γράφο. Ωστόσο το πακέτο αυτό απαιτεί μεγάλη υπολογιστική ισχύ (το ελάχιστο 8GB μνήμης RAM) ακόμα και για την ανακατασκευή μικρών χώρων. Επιπλέον η ικανότητα του για εκτέλεση πραγματικό χρόνο είναι περιορισμένη.

1.2.4 ZED Spatial Mapping

Το ZED Spatial Mapping εφαρμόζει οπτική οδομετρία όπως και το RTAB-Map, αλλά βελτιώνει τα αποτελέσματα της αξιοποιώντας τις μετρήσεις αισθητήρων αδράνειας (IMU), τοποθετημένων πάνω στο ρομπότ ή ενσωματωμένων στην κάμερα η οποία εκτελεί την οπτική οδομετρία (visual-inertial odometry).

Το ZED Spatial Mapping απαιτεί αισθητήρα εκτίμησης βάθους για να εκτελεσθεί και συγκεκριμένα εφαρμόζεται σε κάμερες ZED [16], οι οποίες επιτελούν στερεοσκοπική εκτίμηση βάθους. Παρ' όλα αυτά για ακόμα καλύτερα αποτελέσματα εκτίμησης βάθους χρησιμοποιούνται πλέον νευρωνικά δίκτυα και ο τελικός χάρτης βάθους προκύπτει από τη στατιστική ένωση του στερεοσκοπικού βάθους με το βάθος που προέκυψε μέσω των νευρωνικών δικτύων.

1.2.5 Επιλογή του Κατάλληλου Πακέτου Λογισμικού

Για την αρχική φάση των πειραμάτων που αφορούν το πλαίσιο που αναπτύχθηκε στην παρούσα διπλωματική εργασία χρησιμοποιήθηκε το πακέτο ανακατασκευής RTAB-Map. Αυτό έγινε λόγω της απλότητας που το χαρακτηρίζει, της ικανότητας εκτέλεσης σε πραγματικό χρόνο, της ευελιξίας και πολυχρηστικότητας του, καθώς και της ενεργής κοινότητας χρηστών και ερευνητών που διευκολύνουν την ανάπτυξη λογισμικού με τη χρήση του. Ωστόσο στα τελικά στάδια των πειραμάτων χρησιμοποιήθηκε το ZED Spatial Mapping λόγω της εντυπωσιακής ποιότητας ανακατασκευής που προσφέρει αλλά και των καινοτόμων τεχνολογιών που χρησιμοποιεί.

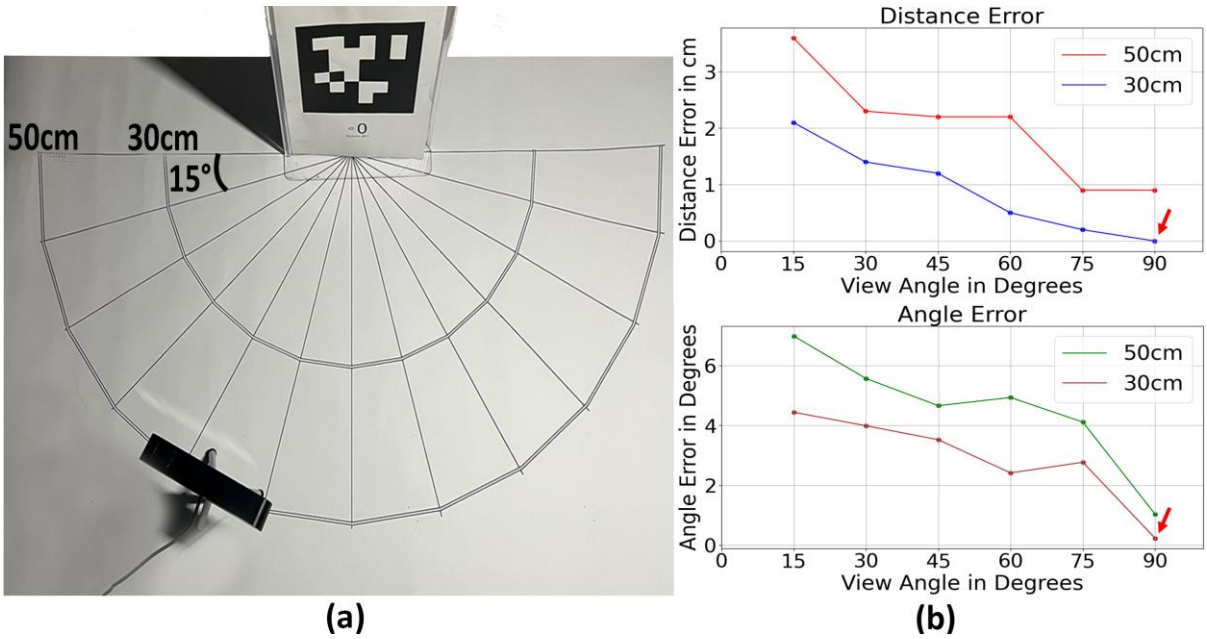
1.3 Πλοήγηση και Σχεδιασμός Διαδρομής

1.3.1 Πλοήγηση

Για να επιτευχθεί πλοήγηση για τη ρομποτική πλατφόρμα η οποία χρησιμοποιήθηκε κατά τη φάση πειραμάτων, αξιοποιήθηκε ένας αλγόριθμος οπτικής οδομετρίας ο οποίος αξιοποιεί δύο κάμερες [17]. Το γεγονός αυτό προσδίδει ευρωστία στο σύστημα καθώς ακόμα και αν μια κάμερα δεν είναι ικανή να εντοπίσει οπτικά χαρακτηριστικά για κάποιον λόγο - όπως για παράδειγμα λόγω τύφλωσης από άμεση ηλιακή ακτινοβολία - τότε η άλλη κάμερα αναλαμβάνει την επίτευξη οπτικής οδομετρίας. Ο αλγόριθμος αυτός αναπτύχθηκε στα πλαίσια του Εργαστηρίου Αυτομάτου Ελέγχου της σχολής των Μηχανολόγων Μηχανικών του Εθνικού Μετσοβίου Πολυτεχνείου. Επιπλέον, ο ίδιος έχει την ιδιαιτερότητα ότι καταφέρνει κλείσιμο βρόχου με τη χρήση ετικέτων April Tags. April Tags είναι σύμβολα εντοπισμού όπως τα γνωστά QR-Codes με τη διαφορά ότι έχουν σχεδιαστεί έτσι ώστε να διευκολύνουν την κάμερα που τα εντοπίζει να αντιλαμβάνεται τον προσανατολισμό τους στον χώρο και επομένως να εκτιμά το δικό της προσανατολισμό σε σχέση με αυτά. Αυτό ακριβώς επιτυγχάνεται και στην περίπτωση του αλγορίθμου που αξιοποιήθηκε. Συγκεκριμένα όταν το ρομπότ εντοπίζει μια ετικέτα, η οποία φέρει ένα μοναδικό αναγνωριστικό, τότε την αποθηκεύει στη μνήμη του. Εάν εντοπίσει ξανά το April Tag με το ίδιο αναγνωριστικό τότε καταλαβαίνει ότι βρίσκεται σε σημείο το οποίο έχει ξαναεπισκεφθεί και αν η εκτίμηση της οδομετρίας του έχει διαφορές από την νέα εκτίμηση με βάση το April Tag τότε αναπροσαρμόζει την τροχιά του και μειώνει το σφάλμα (κλείσιμο βρόχου – loop closure).

Παρατηρήθηκε ότι από πλάγιες γωνίες θέασης και από μεγάλη απόσταση το κλείσιμο βρόχου με χρήση April Tag εισήγαγε σφάλματα στην εκτίμηση οδομετρίας. Για να επιλυθεί το πρόβλημα αυτό κατασκευάστηκε ένας αλγόριθμος οπτικής ευθυγράμμισης της ρομποτικής πλατφόρμας με το April Tag. Η ευθυγράμμιση αυτή λαμβάνει χώρα πριν να ενεργοποιηθεί ο αλγόριθμος κλεισίματος βρόχου και εξασφαλίζει ότι η κάμερα της ρομποτικής πλατφόρμας βλέπει το April Tag από γωνία θέασης 90 μοιρών και απόσταση 30cm. Αφού η ρομποτική πλατφόρμα ευθυγραμμιστεί με το April Tag, τότε και μόνο τότε

ενεργοποιείται ο αλγόριθμος κλεισίματος βρόχου με βάση τον εντοπισμό April Tag. Το σφάλμα από γωνία θέασης 90 μοιρών και απόσταση 30cm βρέθηκε μηδενικό.



Εικόνα 1-2: (a) Η πειραματική διάταξη για την εύρεση της ιδανικής γωνίας θέασης και απόστασης από το April Tag πριν το κλείσιμο βρόχου. (b) Σφάλμα εκτίμησης απόστασης και γωνίας από το April Tag για διάφορες γωνίες και αποστάσεις θέασης.

1.3.2 Σχεδιασμός Διαδρομής

Για την επίτευξη του σχεδιασμού διαδρομής του ρομπότ μέσα στον αμπελώνα εξετάστηκαν τρεις διαφορετικοί αλγόριθμοι: Το Ewok Planner [18], το Safe Corridors Planner [19] και το GB-Planner [20].

Ewok Planner

Το Ewok Planner είναι ένας αλγόριθμος σχεδιασμού διαδρομής ο οποίος εστιάζει στην ταχύτητα εκτέλεσης του σε πραγματικό χρόνο και την ομαλότητα της τροχιάς που υπολογίζει. Χρησιμοποιεί B-Splines για τον υπολογισμό ομαλών τροχιών και αναπαριστά τον χώρο γύρω από το ρομπότ με ένα Voxel Grid για την επίτευξη αποφυγής εμποδίων. Απαιτεί την είσοδο ενός μονοπατιού από την αρχική θέση του ρομπότ έως τη θέση-στόχο και εστιάζει στην ακολούθηση αυτού του μονοπατιού με τον βέλτιστο τρόπο ενώ ταυτοχρόνως επιτυγχάνει αποφυγή εμποδίων.

Safe Corridors Planner

Ο Safe Corridors Planner αξιοποιεί τον αλγόριθμο RRT* για τον υπολογισμό ενός αρχικού μονοπατιού από την αρχική θέση του ρομπότ έως τη θέση-στόχο. Δεν απαιτεί αυτή την πληροφορία σαν είσοδο. Ακολούθως αξιοποιεί το μονοπάτι που υπολογίστηκε αρχικά για να αποσυνθέσει τον χώρο γύρω από το μονοπάτι σε πολύτοπα που αντιπροσωπεύουν ελεύθερο χώρο και δημιουργούν διαδρόμους χωρίς εμπόδια. Ο αλγόριθμος ολοκληρώνεται σε αρκετά μικρό χρονικό διάστημα ώστε να μπορεί να εκτελείται κατ' επανάληψη και επομένως να παρέχει στο ρομπότ δυναμική αποφυγή εμποδίων, δεδομένου ότι ταχύτητα κίνησης του ρομπότ είναι εντός συγκεκριμένων ορίων.

GB-Planner

Το GB-Planner είναι ένας εξερευνητικός αλγόριθμος σχεδιασμού διαδρομής. Αυτό σημαίνει ότι ο αλγόριθμος δεν απαιτεί μια τελική θέση-στόχο ως είσοδο. Αντιθέτως ο σχεδιασμός διαδρομής εκτελείται σε βήματα με σκοπό το ρομπότ να εξερευνήσει τον χώρο γύρω του με μια προεπιλεγμένη στρατηγική. Ο ελεύθερος χώρος γύρω από το ρομπότ αναπαρίσταται σε αυτήν την περίπτωση ως ένας γράφος με κόμβους. Σε κάθε κόμβο αποδίδεται μια εξερευνητική αξία. Το ρομπότ επιλέγει σε κάθε βήμα να επισκεφθεί τον κόμβο με τη μεγαλύτερη εξερευνητική αξία. Στην περίπτωση του GB-Planner, προστίθενται επιπλέον επίπεδα σχεδιασμού που εξασφαλίζουν ότι το ρομπότ θα επισκεφθεί όλους τους κόμβους υψηλού ενδιαφέροντος, καθώς και ότι η εξερεύνηση δεν θα θέσει το ρομπότ εκτός κάποιων στόχων (χρονικών ή εξοικονόμησης ενέργειας).

Επιλογή του Κατάλληλου Πακέτου Σχεδιασμού Διαδρομής

Επιλέχθηκε τελικά η χρήση του Free Corridors Planner, καθώς η τοπολογία των αμπελώνων δημιουργεί εκ φύσεως ελεύθερους διαδρόμους μεταξύ των σειρών του αμπελιού και αυτό διευκολύνει και επιταχύνει την εκτέλεση του αλγορίθμου. Επιπλέον, οι ρομποτικές εργασίες στον χώρο του αμπελιού συνήθως δεν επιβάλλουν υψηλές ταχύτητες κίνησης των ρομπότ με αποτέλεσμα η εκτέλεση σε πραγματικό χρόνο να είναι εφικτή ακόμα και με υψηλές αλγοριθμικές πολυπλοκότητες. Τέλος, το αμπέλι χαρτογραφείται εύκολα και ο εξαγόμενος χάρτης δεν αλλάζει συχνά, πράγμα που καθιστά περιττή την χρήση αλγορίθμων εξερεύνησης, μιας και είναι εύκολο να δοθεί μια θέση-στόχος πριν τον σχεδιασμό διαδρομής με βάση έναν υφιστάμενο χάρτη του αμπελιού.

1.4 Σχεδίαση Συστήματος Αντίληψης

Στα πλαίσια της παρούσας εργασίας δημιουργήθηκε ένα σύστημα αντίληψης πάνω στο οποίο εκτελέστηκαν οι αλγόριθμοι ανακατασκευής χώρου καθώς και οι αλγόριθμοι σχεδιασμού τροχιάς που επιλέχθηκαν. Για το σκοπό αυτό ερευνήθηκε το είδος των αισθητήρων του συστήματος αντίληψης (LiDAR, Depth Cameras, Structured Light Projectors), το πλήθος των αισθητήρων, καθώς και ο τρόπος τοποθέτησής τους πάνω στη ρομποτική πλατφόρμα που χρησιμοποιήθηκε σε πειράματα σε προσομοίωση αλλά και σε πραγματικό χώρο. Εκτός αυτού, το σύστημα αντίληψης περιλαμβάνει και μια μονάδα επεξεργασίας για την επιλογή της οποίας εξετάστηκαν οι εξής κατηγορίες: Μικροελεγκτές, Υπολογιστές μίας πλακέτας (Single Board Computers), Mini PCs και Laptops.

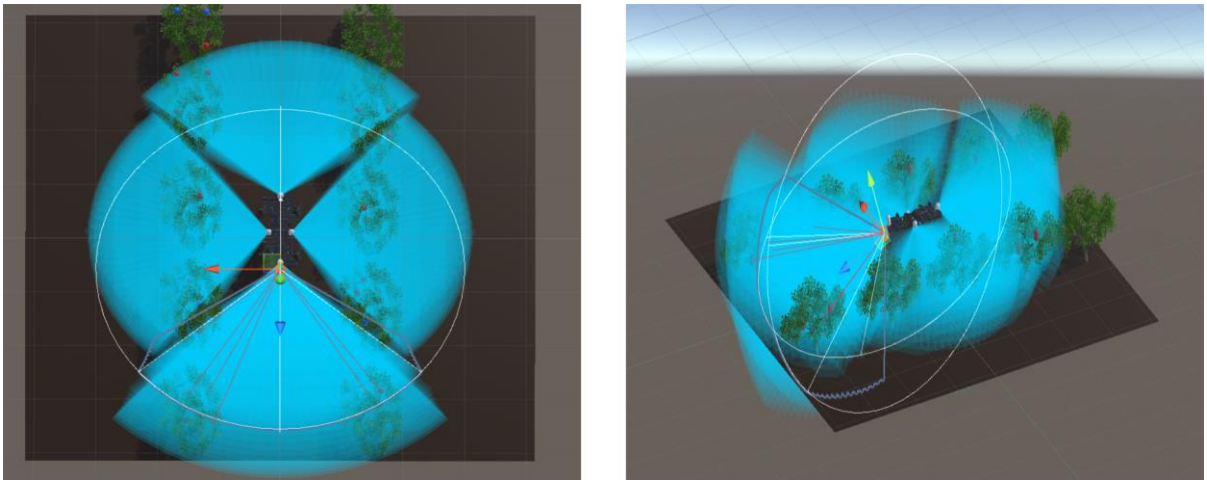
1.4.1 Field of View Visualization and Lidar Resolution Analysis Tool (FoVaLiRa)

Για την διευκόλυνση του προσδιορισμού του είδους αλλά και του τρόπου τοποθέτησης των αισθητήρων του συστήματος αντίληψης πάνω στη ρομποτική πλατφόρμα αναπτύχθηκε το εργαλείο FoVaLiRa με τη βοήθεια του Unity. Το εργαλείο αυτό χρησιμοποιεί ray casting για να προσομοιώσει αισθητήρες λέιζερ αλλά και κάμερες βάθους. Επιτρέπει την εύκολη και γρήγορη σύγκριση πολλών διατάξεων σε διαφορετικές ρομποτικές πλατφόρμες. Με αφορμή τη δημιουργία του εργαλείου επιτεύχθηκε επιπλέον η πρώτη φάση της προσομοίωσης του αμπελιού σε Unity.

Το εργαλείο FoVaLiRa επιτρέπει την εισαγωγή στόχων οι οποίοι στην περίπτωση μας είναι τα τσαμπιά του αμπελιού. Επιτρέπει οπτικοποίηση της επικάλυψης του οπτικού πεδίου

δύο ή περισσοτέρων αισθητήρων πάνω στην ρομποτική πλατφόρμα, καθώς και την εισαγωγή παραμέτρων από .csv αρχείο για την εύκολη παραμετροποίηση της προσομοίωσης.

Η τελική διάταξη που επιλέχθηκε αποτελείται από τέσσερις κάμερες βάθους και έναν αισθητήρα LiDar για επιπλέον ευρωστία. Ο τελευταίος είναι προαιρετικός και το οπτικό του πεδίο επικαλύπτεται με το οπτικό πεδίο της μπροστινής κάμερας του ρομπότ. Η διάταξη αυτή εφαρμόστηκε σε τετράποδο ρομπότ κατά τα πειράματα προσομοίωσης αλλά και στην πραγματική ρομποτική πλατφόρμα κατά τα πειράματα στο εργαστήριο.



Εικόνα 1-3: Το τετράποδο ρομπότ Lealaps II, με την τελική διάταξη αισθητήρων αντίληψης όπως φαίνεται εντός του εργαλείου οπτικοποίησης FoVaLiRa.

1.4.2 Μονάδα Επεξεργασίας

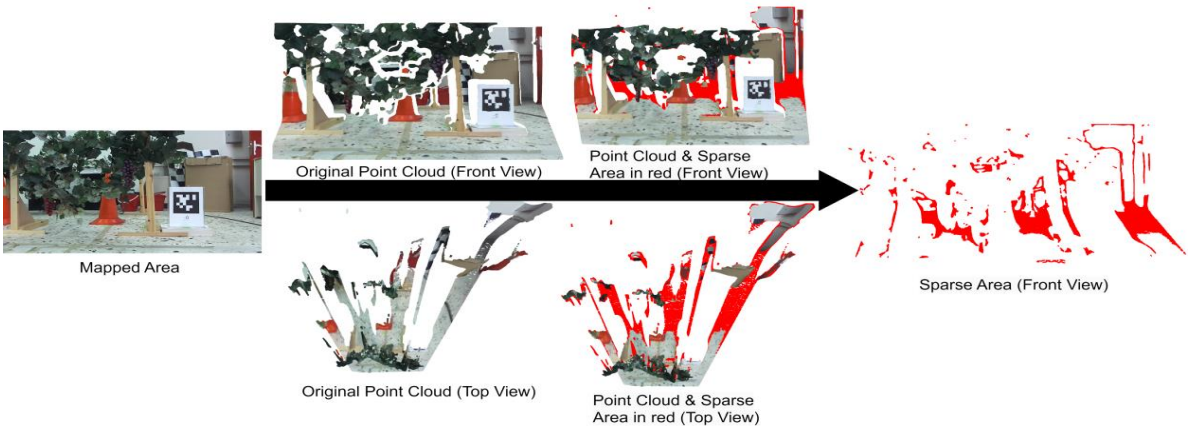
Οι μικροελεγκτές ήταν η κατηγορία που εξετάστηκε πρώτη ως υποψήφια μονάδα επεξεργασίας για το σύστημα αντίληψης που αναπτύχθηκε. Οι μικροελεγκτές προσφέρουν εντυπωσιακή αποδοτικότητα ισχύος αλλά υστερούν σημαντικά σε πολυχρηστικότητα και ευκολία προγραμματισμού. Για το λόγο αυτό εξετάστηκαν έπειτα τα Laptops και τα Mini PCs. Αυτές οι επιλογές διέπονται από σημαντική πολυχρηστικότητα και ευκολία χρήσης αλλά υστερούν σημαντικά σε αποδοτικότητα ισχύος και βάρους. Αυτό οδήγησε στο να εξεταστούν τελικά οι υπολογιστές μιας πλακέτας (Single Board Computers) οι οποίοι όπως αποδείχθηκε προσφέρουν μια ισορροπία μεταξύ πολυχρηστικότητας, βάρους και αποδοτικότητας ισχύος. Αυτά τα χαρακτηριστικά είναι τα ιδανικά για την εφαρμογή που εξετάζουμε στα πλαίσια της παρούσας διπλωματικής εργασίας και έτσι οι υπολογιστές μιας πλακέτας επιλέχθηκαν για τη συνέχεια των πειραμάτων.

1.5 Ανάπτυξη του Vinymap για Βελτιωμένη Ανακατασκευή και Επιθεώρηση

Το Vinymap είναι ένας αλγόριθμος που αναπτύχθηκε για να πετύχει τρεις στόχους: τη βελτίωση της ποιότητας των δεδομένων point cloud που λαμβάνονται από τις κάμερες βάθους, την εκτίμηση της πυκνότητας του φυλλώματος των αμπελιών και την παραγωγή ενός δείκτη ο οποίος ερμηνεύεται εύκολα από τους αγρότες και, τέλος, την φωτο-ρεαλιστική ανακατασκευή του αμπελώνα με σκοπό ο αγρότης να έχει ένα τρισδιάστατο αντίγραφο του αμπελιού του και να το επιθεωρεί σε δεύτερο χρόνο.

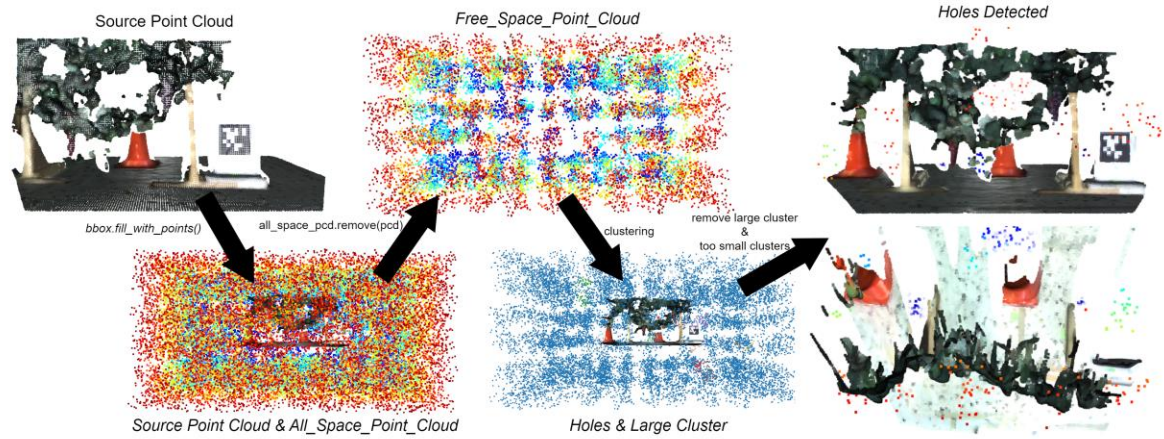
1.5.1 Αξιολόγηση και Βελτίωση Ποιότητας Νέφους Σημείων (point cloud)

Το Vinymap επιτυγχάνει την αξιολόγηση και βελτίωση ποιότητας του point cloud με τρεις τρόπους: μετράει τα points τα οποία γειτνιάζουν με λιγότερα points από τον μέσο όρο. Τα αφαιρεί από το αρχικό point cloud διότι είναι πιθανό να αποτελούν θόρυβο. Έπειτα υπολογίζει τον όγκο που αυτά καταλαμβάνουν και παράγει έναν δείκτη που εκφράζει την αραιότητα του point cloud (sparsity index).



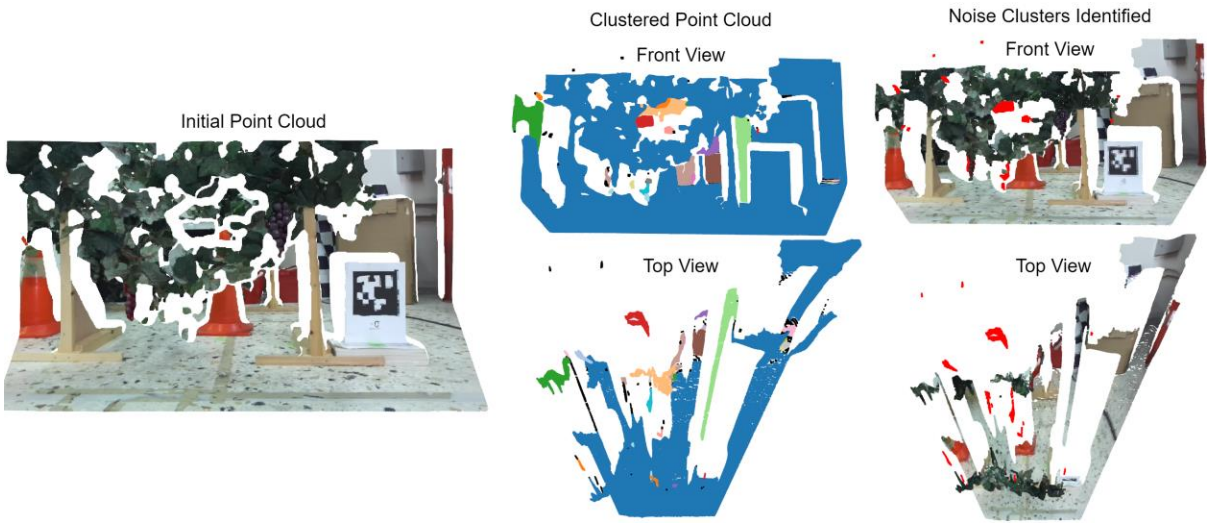
Εικόνα 1-4: Εύρεση περιοχών χαμηλής πυκνότητας στο point cloud.

Κατά δεύτερον εντοπίζει κενά μέσα στο point cloud. Αυτό επιτυγχάνεται γεμίζοντας τον χώρο ομοιόμορφα με points, αφαιρώντας το αρχικό point cloud από την ομοιόμορφη κατανομή points και εφαρμόζοντας opening [21] στο εναπομείναν point cloud. Το αποτέλεσμα είναι η εφάνιση κενών περιοχών του point cloud ως νέα νέφη. Αυτές ερμηνεύονται ως αχαρτογράφητοι χώροι - τρύπες - και μέσω αυτών παράγεται ένας δείκτης ποιότητας σε σχέση με τον αριθμό και τον όγκο που καταλαμβάνουν οι τρύπες στο αρχικό point cloud.



Εικόνα 1-5: Αλγόριθμος εύρεσης κενών περιοχών στο point cloud.

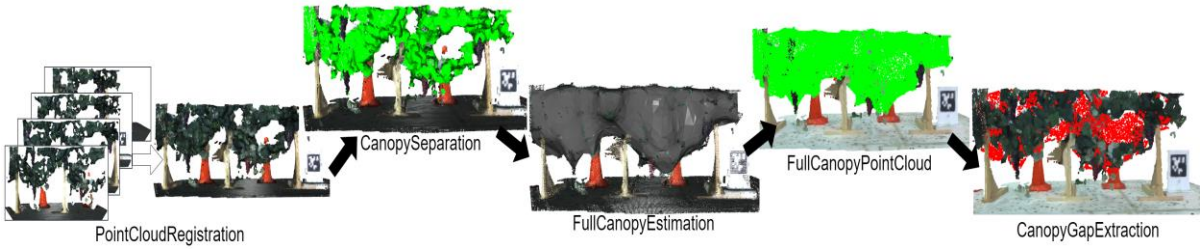
Κατά τρίτον, το Vinymap χωρίζει το αρχικό point cloud σε συστάδες (clusters) και από αυτές αφαιρεί τις απομακρυσμένες και μικρές σε όγκο καθώς αυτές είναι πιθανό να αποτελούν θόρυβο. Παράλληλα μετράει τον όγκο που αυτές καταλαμβάνουν και παράγει έναν ακόμα δείκτη ποιότητας. Συνδυάζοντας τους δείκτες που υπολογίστηκαν σε ένα τελικό δείκτη, το Vinymap δίνει τη δυνατότητα αντικειμενικής αξιολόγησης του point cloud. Παράλληλα, με τις ενέργειες αφαίρεσης θορύβου βελτιώνεται η ποιότητα των αρχικών νεφών σημείων.



Εικόνα 1-6: Εντοπισμός συστάδων που αποτελούν θόρυβο.

1.5.2 Εκτίμηση Πυκνότητας Φυλλώματος Αμπελιών

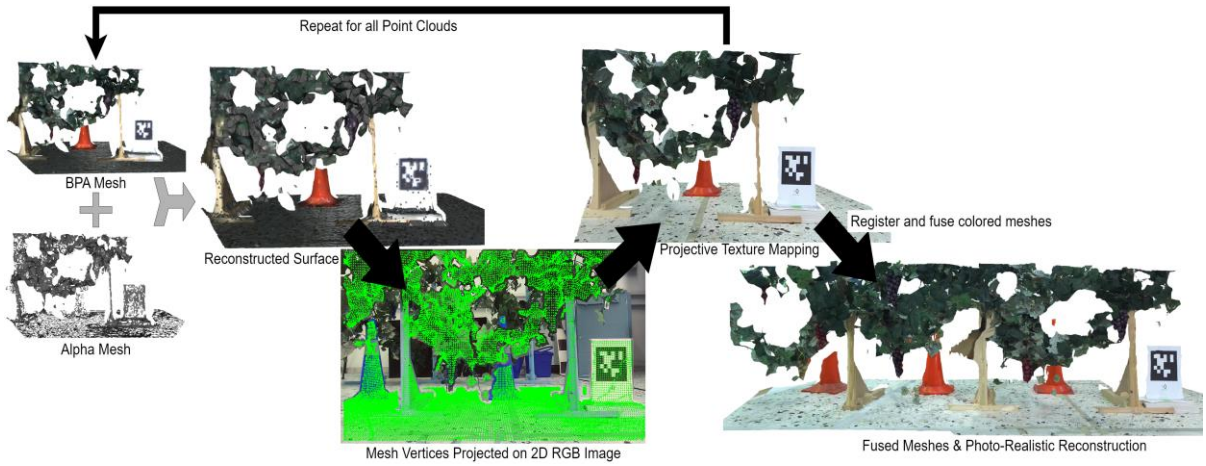
Το Vinymap επιτελεί επιπλέον εκτίμηση της πυκνότητας του φυλλώματος των αμπελιών. Αυτό επιτυγχάνεται με τη συνένωση των διαδοχικών λαμβανομένων point clouds (registration) με τη χρήση του αλγορίθμου KISS-ICP [22]. Ακολουθεί ο διαχωρισμός του φυλλώματος με βάση το χρώμα. Η διαδικασία ολοκληρώνεται με την εκτίμηση του πλήρους φυλλώματος μέσω της χρήσης των τοπολογικών ιδιοτήτων των alpha shapes [23]. Η συσχέτιση του πλήρους, γεμάτου φυλλώματος με το κανονικό φύλλωμα προσδίδει έναν δείκτη της πυκνότητας του φυλλώματος του αμπελιού.



Εικόνα 1-7: Αλγόριθμος εκτίμησης πυκνότητας φυλλώματος αμπελιών.

1.5.3 Φωτο-ρεαλιστική Ανακατασκευή Αμπελώνα

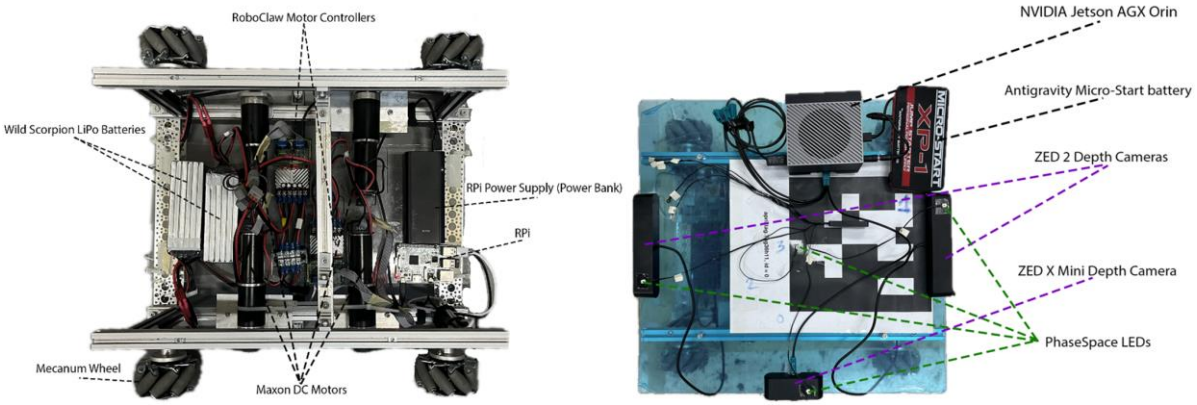
Τέλος το Vinymap επιτυγχάνει φωτο-ρεαλιστική ανακατασκευή αμπελώνα χάρη στη σύγχρονη αποθήκευση έγχρωμων εικόνων μαζί με τα διαδοχικά λαμβανόμενα point cloud. Οι εικόνες αυτές προβάλλονται πάνω σε ένα ανακατασκευασμένο πλέγμα που προκύπτει από την συνένωση των point clouds. Για την επίτευξη της διαδικασίας αυτής απαιτείται η γνώση των εσωτερικών και εξωτερικών παραμέτρων της κάμερας που λαμβάνει τα point cloud αλλά και της έγχρωμες εικόνες. Ο αλγόριθμος αυτός μπορεί να χρησιμοποιηθεί και σε συστήματα που αξιοποιούν ένα LiDAR και μια έγχρωμη κάμερα RGB.



Εικόνα 1-8: Αλγόριθμος φωτό-ρεαλιστικής ανακατασκευής Αμπελώνα.

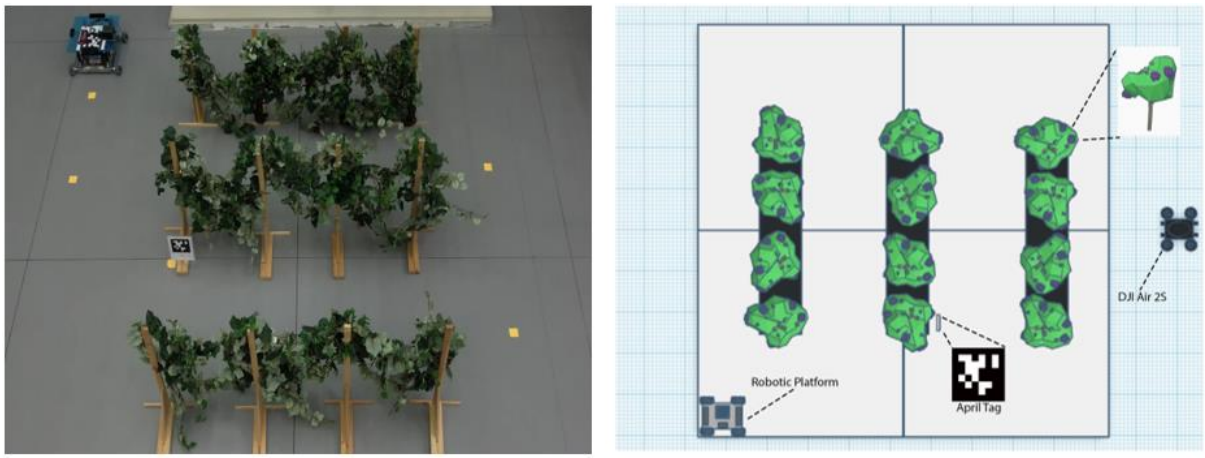
1.6 Πειραματική Διάταξη

Σε πρώτο στάδιο το πλαίσιο λογισμικού αναπτύχθηκε με τη βοήθεια του προσομοιωτή Gazebo [24]. Εκεί κατασκευάστηκε ένα ψηφιακό ανάλογο ενός μικρού αμπελώνα με ιδανικές συνθήκες φωτισμού εσωτερικού χώρου. Στη συνέχεια όλοι οι αλγόριθμοι εφαρμόστηκαν στο ανεπτυγμένο σύστημα αντίληψης, το οποίο αναρτήθηκε πάνω σε μία ρομποτική πλατφόρμα κατασκευασμένη στο Εργαστήριο Αυτόματου Ελέγχου. Ιδιαίτερο χαρακτηριστικό της ρομποτικής πλατφόρμας αυτής αποτελεί ικανότητα της για κάθετη και οριζόντια κίνηση αλλά και αυτο-περιστροφή χάρη στους τροχούς τύπου mecanum [25] που φέρει.



Εικόνα 1-9: Η ρομποτική πλατφόρμα και το σύστημα αντίληψης που αναπτύχθηκαν στο Εργαστήριο Αυτομάτου Ελέγχου.

Η ρομποτική αυτή πλατφόρμα εκτέλεσε πειράματα σε έναν συνθετικό αμπελώνα ο οποίος κατασκευάστηκε στο ίδιο εργαστήριο με σκοπό να αποτελεί ένα πιστό αντίγραφο ενός πραγματικού αμπελιού αναφορικά με τα οπτικά του χαρακτηριστικά αλλά και τις διαστάσεις του.



Εικόνα 1-10: Πειραματική διάταξη συνθετικού αμπελώνα στο Εργαστήριο Αυτομάτου Ελέγχου.

1.7 Αποτελέσματα

1.7.1 Αξιολόγηση και Βελτίωση Ποιότητας Point Cloud

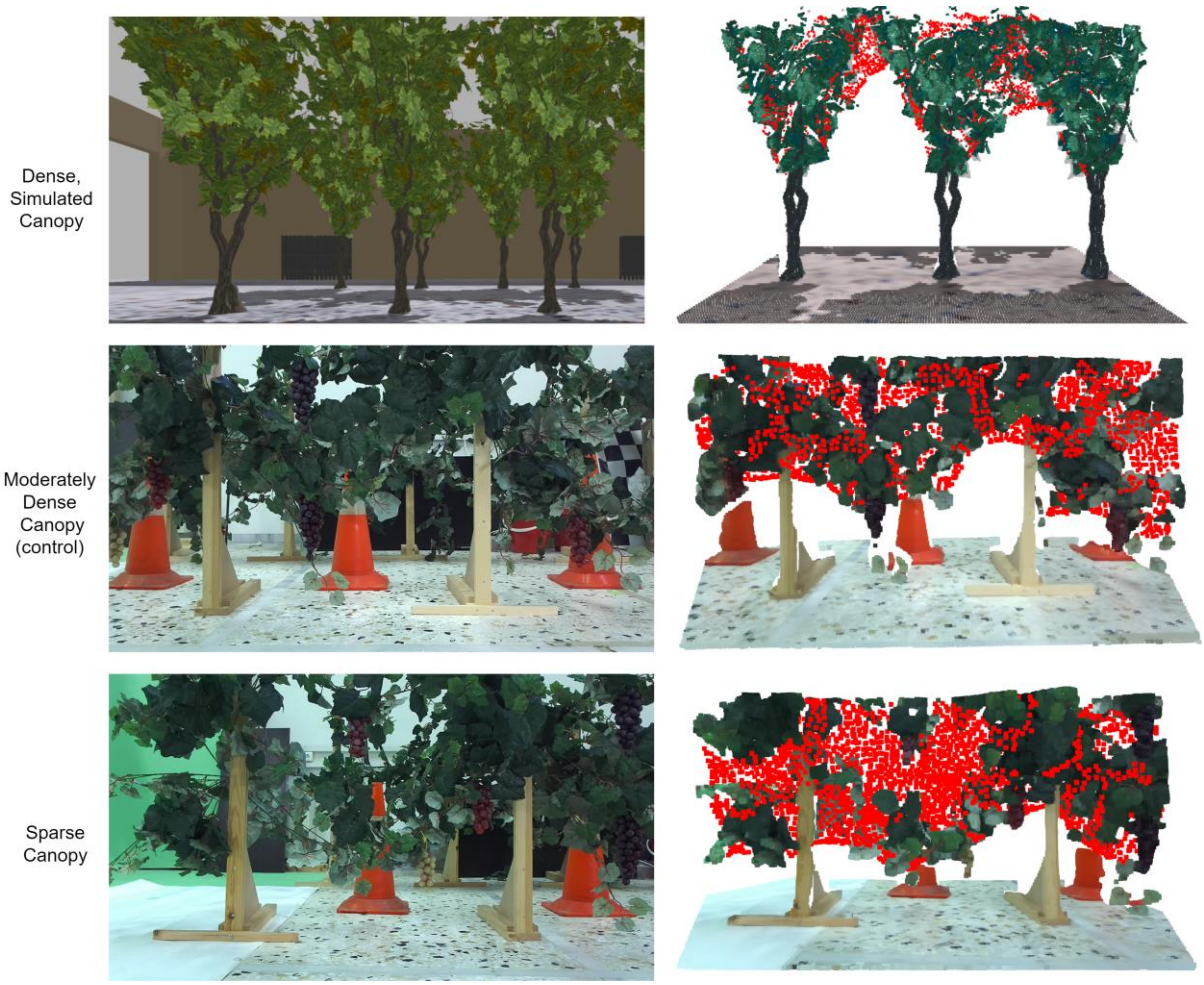
Αρχικά, η αξιολόγηση του αλγορίθμου πραγματοποιήθηκε με υποκειμενική μεθοδολογία. Συγκρίθηκαν τα νέφη σημείων που προέρχονταν απευθείας από την κάμερα με τα αντίστοιχα επεξεργασμένα και βελτιωμένα. Στη συνέχεια, υλοποιήθηκε ποσοτική ανάλυση των αποτελεσμάτων, αξιοποιώντας τους δείκτες αξιολόγησης του Vinymap.

Στα αρχικά νέφη σημείων προστέθηκε τεχνητά θόρυβος, με σκοπό να αξιολογηθεί η ικανότητα του αλγορίθμου Vinymap να τον ανιχνεύει και να τον απομακρύνει. Η ανάλυση επιβεβαίωσε την επιτυχή ανίχνευση και απομάκρυνση του θορύβου

1.7.2 Αξιολόγηση Αλγορίθμου Εκτίμησης Πυκνότητας Φυλλώματος

Με στόχο να αξιολογηθεί η απόδοση του αλγορίθμου μας για την εκτίμηση της πυκνότητας του φυλλώματος των φυτών, διεξήχθησαν πειράματα σε δύο περιβάλλοντα: έναν προσομοιωμένο και συνθετικό αμπελώνα. Η προσομοίωση παρείχε ένα περιβάλλον με υψηλό επίπεδο ελέγχου, σχεδόν χωρίς διακυμάνσεις στην πυκνότητα του φυλλώματος. Το περιβάλλον του συνθετικού αμπελιού ήταν πιο ρεαλιστικό, αλλά και πάλι ελεγχόμενο. Αρχικά δοκιμάσαμε τον αλγόριθμο σε πυκνά, τέλεια φυλλώματα, λαμβάνοντας πολύ υψηλές τιμές στον δείκτη πυκνότητας, όπως αναμενόταν. Έπειτα δοκιμάσαμε δύο διαφορετικά σενάρια: 1) Κανονικό Φύλλωμα: Ο αλγόριθμος επέστρεψε έναν δείκτη πυκνότητας εντός των αναμενόμενων ορίων, σύμφωνα με την οπτική εκτίμηση του φυλλώματος. 2) Αραιό Φύλλωμα: Η παρουσία κενών οδήγησε σε σημαντικά χαμηλότερο δείκτη πυκνότητας, επικυρώνοντας την ικανότητα του αλγορίθμου να ανιχνεύει και να ποσοτικοποιεί τις ανωμαλίες της βλάστησης.

Στο εργαστήριο, επαναλάβαμε τους ελέγχους κάτω από φυσικό και τεχνητό φωτισμό. Όλες οι δοκιμές παρείχαν ακριβή και εύχρηστα αποτελέσματα, αποδεικνύοντας την απλότητα και την αποτελεσματικότητα του αλγορίθμου στην εκτίμηση της πυκνότητας του φυλλώματος των φυτών.



Εικόνα 1-11: Αξιολόγηση πυκνότητας φυλλώματος. Με κόκκινο χρώμα: κενά στο φύλλωμα των αμπελώνων.

1.7.3 Αξιολόγηση Αλγορίθμου Φωτο-ρεαλιστικής Ανακατασκευής Αμπελώνα

Ο αλγόριθμος Φωτο-ρεαλιστικής Ανακατασκευής Αμπελώνα συγκρίθηκε με τα μη επεξεργασμένα δεδομένα νέφους σημείων και τον αλγόριθμο χαρτογράφησης χώρου ZED Spatial Mapping της Stereolabs όσον αφορά την οπτική πιστότητα και τον πλούτο οπτικής πληροφορίας.

Η άμεση σύγκριση με τα μη επεξεργασμένα δεδομένα point cloud αποκαλύπτει τη διαφορά. Ενώ τα μη επεξεργασμένα δεδομένα παρέχουν μια βασική αναπαράσταση, το ανακατασκευασμένο πλέγμα μας προσφέρει μια σημαντικά πιο ευκρινή απεικόνιση του αμπελώνα. Όπως φαίνεται στην Εικόνα 1-12, η ετικέτα April Tag παρουσιάζει βελτιωμένη ευκρίνεια και τα μεμονωμένα σταφύλια μέσα στα τσαμπιά διακρίνονται πιο εύκολα.

Εικόνα 1-12: Σύγκριση ποιότητας ανακατασκευής.

Σε σύγκριση με την υψηλής ποιότητας ανακατασκευή του αλγορίθμου χαρτογράφησης χώρου ZED Spatial Mapping της Stereolabs στις υψηλότερες ρυθμίσεις του, η ανακατασκευή μας αναδεικνύεται ως η ανώτερη λύση. Η προσέγγιση προβολής εικόνας σε πλέγμα δημιουργεί πλέγματα με σημαντικά μεγαλύτερη οπτική λεπτομέρεια, επιτρέποντας στους αγρότες να επιθεωρήσουν με μεγαλύτερη άνεση μεμονωμένα φύλλα και σταφύλια. Αυτή η βελτιωμένη υφή διευκολύνει τη λήψη πιο ενημερωμένων αποφάσεων στην απομακρυσμένη γεωργία ακριβείας.

1.8 Συμπεράσματα και Μελλοντική Εργασία

Η παρούσα διπλωματική εργασία παρουσιάζει ένα νέο πλαίσιο για την τρισδιάστατη ανακατασκευή και ανάλυση αμπελώνων. Εξετάζοντας την προσέγγισή αυτή σε συνθετικό αμπελώνα, επιτεύχθηκε βελτιωμένη πλοήγηση με οπτική οδομετρία μέσω ετικετών AprilTags και οπτικής ευθυγράμμισης πριν την έναρξη βελτιστοποίησης κλεισίματος βρόχου. Προέκυψε πιο ευκρινή τρισδιάστατη ανακατασκευή αμπελώνα σε σύγκριση με την προσέγγιση χαρτογράφησης χώρου ZED (ZED Spatial Mapping). Αναπτύχθηκε επίσης μια ολοκληρωμένη λύση επιθεώρησης που ενσωματώνει προσεγγίσεις σχεδιασμού διαδρομής με βάση την αποσύνθεση πολύτοπων, μια μέθοδο εφαρμοζόμενη για πρώτη φορά σε πραγματική ρομποτική πλατφόρμα. Το σύστημα αντίληψης προσαρμόστηκε ειδικά στην

εφαρμογή μας με τη βοήθεια του FoVaLiRA, ενός εργαλείου ανάλυσης πεδίου όρασης και ανάλυσης LiDar και καμερών βάθους, αναπτυγμένο στο Unity.

Κοιτάζοντας προς το μέλλον, η ενσωμάτωση τεχνολογιών βαθιάς μάθησης και αυτοματοποιημένων τεχνικών ανάλυσης θα μπορούσε να επιτρέψει πολύπλοκες εργασίες επιθεώρησης, όπως η ανίχνευση ασθενειών ή η επισήμανση πιθανών κινδύνων για περαιτέρω εξέταση από τον οινοποιό. Επιπλέον, η επέκταση της αυτόνομης επιθεώρησης θα μπορούσε να οδηγήσει σε πλήρη κάλυψη και παρακολούθηση του αμπελώνα, χωρίς να απαιτούνται προκαθορισμένα σημεία έναρξης και λήξης διαδρομής. Πιστεύουμε ωστόσο ότι αυτό το πλαίσιο αποτελεί ένα ισχυρό πρώτο βήμα προς μια ολοκληρωμένη λύση για την γεωργία ακριβείας σε αμπελώνες.

2 Introduction

2.1 Motivation

The agricultural sector is facing numerous challenges, including climate change, resource scarcity, and the need to increase productivity while minimizing environmental impact. Traditional agricultural practices often rely on manual observation and data collection, which can be time-consuming, labor-intensive, and error prone. This limits farmers' ability to make informed decisions and optimize crop yields. 3D reconstruction technology offers a transformative solution to these challenges; measuring crop height, canopy density, and other phenotypic parameters, as well as detecting weeds and mapping field topography and soil conditions are only some of the possibilities.

The Legged Robots Team of the Control Systems Lab (CSL) in NTUA has designed and manufactured quadruped robots which utilize sophisticated and in-house-developed mechanical and electrical subsystems. Lealaps II and its successor, Argos, are two optimally designed quadruped robots, well suited for agricultural use. They were lacking, however, a robotic perception system which would empower them to perceive, understand, and interact with their surroundings autonomously.

The objective of the present thesis is to design and implement a perception system for agriculture quadruped robots with 3D reconstruction of vineyards as the main focus. The design considers both software and hardware. It features a novel spatial mapping algorithm which preserves the visual detail required for a reconstruction to be useful for crop and field analysis. A navigation and path-planning stack which enables the robot to traverse a real-life vineyard effectively and safely is also proposed in this work. It utilizes a state-of-the-art Model Predictive Control (MPC) planner, an in-house dual-camera visual odometry algorithm and a robust PID trajectory tracking controller. A thorough review of relevant literature is crucial for narrowing down design decisions.

2.2 Literature Review

2.2.1 Agriculture Robots

Human population has grown rapidly in recent years and will continue to do so for several more. The need for food is a global concern for governments and scientists. However, the way to increase food production should not be to expand cultivated land at the expense of forests, but to increase the productivity of the soil and plants that are already established. As a result, the interest in Agriculture Robots has been increasing the last few years.

There are three main factors that heavily influence the design of agriculture robots and differentiate it from the design of other task-specific robots: agriculture-specific navigation, agriculture-specific image processing and Handling Rough Terrain [26]. Focusing on these aspects, researchers have come up with various solutions in the recent past.

The authors in [27] have designed a wheeled robotic platform which utilizes a monocular downward facing RGB camera primarily used for classification of crop and weed plants and for visual odometry. The robot uses machine vision to detect weeds within the crop rows and treats the weeds by high precision drop-on-demand application of herbicide.

Researchers in [28] have developed the VineRobot, a multi-million euro project robot designed to autonomously and non-intrusively traverse vineyards utilizing ultrasonic sensors and RGB camera visual feedback, while providing the farmers with useable real time data regarding the state of the vineyard it explores. The authors in [29] have developed an autonomous pruning robot. Their pruning setup consists of a Universal Robots UR5e robot mounted on a linear axis. The end effector consists of a set of electric bypass pruners along with a RealSense D435 RGBD camera. For field trials, the robot was installed on the back of a remote-controlled utility vehicle and powered with a portable generator.

It is evident that the world of agriculture is complex enough to allow for the development of a large variety of robotic solutions that differ according to the problem that they are called to solve. The present thesis focuses on the mediterranean vineyard, which is often characterized by mountainous and rocky terrain, unsymmetrical rows as well as erratic canopy at the late vegetative state. The mobility advantage and agility that a legged robot presents cannot be replicated by a simple wheel layout. This is one of the reasons why legs are the main locomotion mechanism in nature and the reason why the perception system described in this work is mounted on and designed for a quadruped robot. Finally, the rich and irregular canopy of vine trees calls for innovative machine vision solutions.

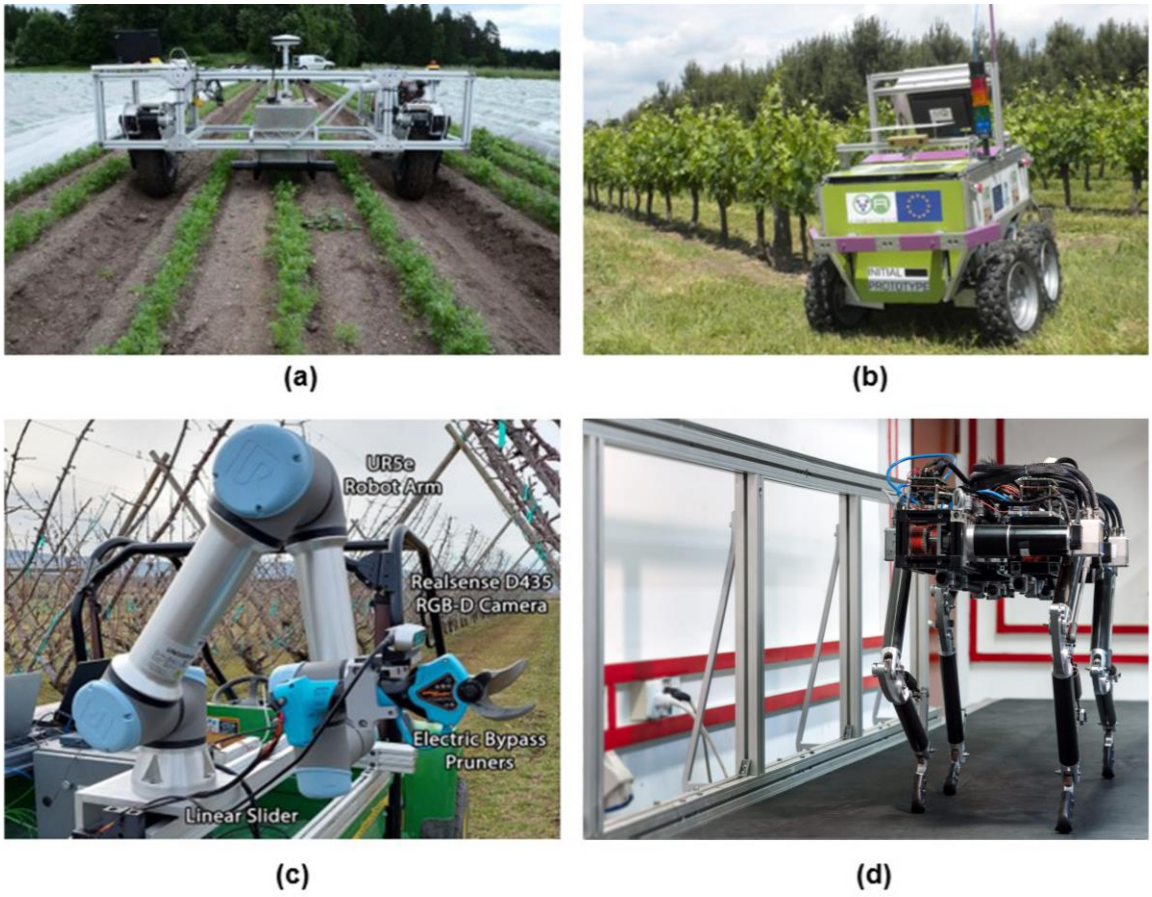


Figure 2-1: Robots designed for various agricultural Tasks.
(a) wheeled weed detection robot designed by the authors in [27]
(b) VineRobot (c) pruning robot designed by the authors in [29]
(d) Lealaps II quadruped robot designed at CSL NTUA.

2.2.2 Path Planning

Although path planning has been studied extensively for applications in indoor environments, there is still much room for development and research for optimization in agricultural environments. In fact, most recent relative research has focused on Coverage Path Planning (CPP) algorithms [30]; these solve the problem of determining a path that passes over all points of an area or volume while avoiding obstacles [31]. Point-to-point path planning i.e.: determining an optimal and collision-free path from a starting point to a destination point, which is more useful in precision agriculture applications, is not as common in the literature. There are, however, several interesting publications.

Recently, researchers have designed a path planning and obstacle avoidance package which enables their custom rover to either move along the middle of a row of a mountainous vineyard or move through a row according a received path line [32]. To achieve this, their software represents three-dimensional space as a two-dimensional occupancy gird map i.e. a grid where cells with obstacles are labeled as occupied. This map is then heuristically subdivided into regions based on the robot's maneuvering capability and fed into a decision-making algorithm. This is a simple and effective approach. The software, however, is closely linked to the robot's kinematic characteristics and cannot be generalized. In a more recent work researchers introduce a new path planning algorithm that utilizes a topological map and extends the A* search based planning algorithm , to ensure a safe path and a maximum distance from the vine trees of a steep slope vineyard [33].

The authors in [34] propose a hybrid Voronoi-based ant colony optimization (V-ACO) path planning algorithm to solve an adaptive ocean sampling problem. Ant Colony Optimization is a metaheuristic algorithm inspired by the foraging behavior of ants. Ants communicate with each other by laying down pheromone trails, which they use to guide others to food sources. ACO algorithms use this same concept to search for solutions to optimization problems.

The algorithm works by having a population of artificial ants, each of which starts at a random location in the search space. The ants then follow a probabilistic path through the search space, using pheromone trails to guide their way. The pheromone trails evaporate over time, so the ants are more likely to follow trails that have been recently laid. If a simulated ant finds a short path to the target, it will execute more routes to and from the target. Thus, the short path will have less evaporated pheromones than other paths and will be preferred as closer to optimal. The Voronoi based scheme utilizes Voronoi partition to highlight high interest areas. The V-ACO algorithm could be altered to suit an agricultural setting, although vineyards have a distinct and special topology of corridors and rows that this scheme does not capitalize on.

Finally, machine learning is often used as a solution to path planning problems that involve multiple agents or take place in complex, dynamic environments. The authors in [35] present deep reinforcement learning as a framework to model the complex interactions and cooperation required by robots that navigate among pedestrians. They utilize an LSTM neural network that enables the algorithm to use observations of an arbitrary number of agents. The algorithm learns collision avoidance among a variety of types of dynamic agents without assuming they follow any particular behavior rules. The authors in [36] propose a Bezier curve based approach for the path planning in a dynamic field using a Modified Genetic Algorithm (MGA). The robot's path is dynamically decided based on the obstacles' locations. With the goal of optimizing the distance between the start point and the target

point, the MGA is employed to search for the most suitable points as the control points of the Bezier curve. Using the chosen control points, the optimum smooth path that minimizes the total distance between the start and the end points is selected.

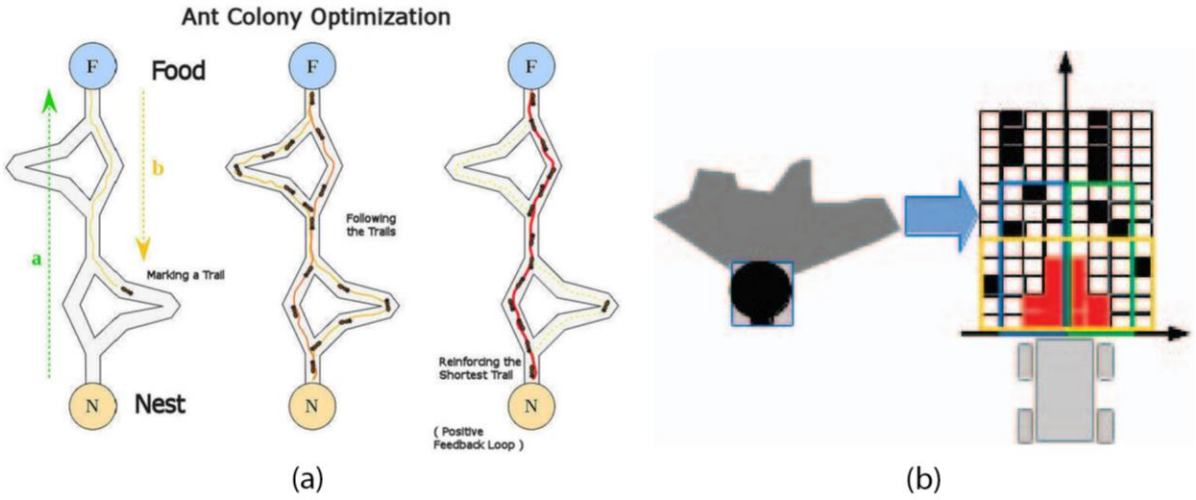


Figure 2-2: Path planning strategies.
(a) Ant Colony Optimization Visualized [34] (b) Local occupation grid map generated in [30]. On the left sensor observation, on the right the produced grid map.

2.2.3 SLAM and 3D reconstruction

Simultaneous Localization and Mapping (SLAM), as the name implies, is the problem of simultaneously estimating a robot's position and the 3D structure of the environment it is traversing. 3D reconstruction, on the other hand, focuses solely on building a detailed 3D representation of the scene, independently of the robot's motion. Most active SLAM approaches provide estimations of the environment in the form of geometric representations (e.g., OG maps, Octomaps). However, when we explore new environments as humans, we are not just interested in the shape of the environment, but also in the textures and materials of the world around us, as well as in semantic elements of the environment (e.g., presence of objects, rooms) [37]. Achieving SLAM while capturing high- and low-level semantics is commonly referred to as Spatial Perception or Spatial Mapping. In this case, 3D space is represented as a 3D mesh or a 3D point cloud. Determining the quality of the representation is of outmost importance, especially for agricultural applications, where the output data must be reliable enough to aid with tasks such as crop inspection.

Point Cloud Quality Assessment

Point cloud quality assessment (PCQA) is a crucial task for evaluating the integrity and usability of point clouds, which are collections of data points that represent three-dimensional objects or environments. PCQA metrics aim to quantify the degradation or distortion introduced during acquisition, processing, compression, transmission, or rendering processes. This information is essential for optimizing point cloud processing algorithms, ensuring data consistency, and maintaining visual quality for applications in various domains. Two main categories of PCQA metrics exist: full-reference (FR) and no-reference (NR).

FR metrics require access to a reference point cloud of the same scene or object to compare with the distorted version. The reference point clouds are usually of very high

quality and together formulate a dataset that is considered a ground truth dataset. The ground truth dataset can be synthetic as in [38] and [39], or include point clouds that are captured with a high quality and expensive scanner as in [40] [41].

NR metrics, on the other hand, operate solely on the distorted point cloud itself, making them more versatile for applications where reliable reference point clouds data may not be readily available [42]. Recent advancements in PCQA have focused on learning-based approaches, which utilize deep neural networks to extract discriminative features from point clouds and predict quality scores. These methods have shown promising results in capturing subtle distortions and generalizing to diverse datasets [43] [44]. Simpler NR quality metrics like point cloud density, point confidence and Local outlier factor (LOF) [45] [46], as well as mathematical methods [47] have also been used successfully throughout literature.

There is yet another common method of assessing 3D point cloud quality: Subjective point cloud quality assessment. Humans visually review and effectively assign quality scores to 3D data. Occlusions, unregistered objects and visible distortion in the reviewed point cloud or mesh are often mentioned [48] [49]. Subjective point cloud quality assessment is often used to evaluate objective quality assessment metrics [50].

Canopy Quality Assessment

The quest to objectively assess canopy quality and predict crop yield has long enthralled precision agriculture researchers. Early efforts relied on laborious and destructive chemical processes like leaf nitrogen analysis [51], offering limited insights and heavily impacting plant health.

As technology evolved, non-destructive alternatives emerged. Hyperspectral imaging [52], while promising, brought its own challenges: complexity, expensive equipment, and specific technical knowledge requirements from the farmers.

Recently, computer vision techniques have gained popularity [53], offering optical assessment methods with potential for real-time insights. Approaches like machine learning-based leaf area estimation are effective [54], but often wrestle with training data requirements, computational demands, and susceptibility to varying lighting and weather conditions, potentially hindering their practical application.

Our approach in Vinymap sought a different path. Driven by the need for simplicity, real-time applicability, and robustness under diverse environmental conditions, we designed a canopy assessment algorithm that is non-destructive, preserving precious vine health for optimal yield, simple, real-time viable and robust, performing reliably across diverse lighting conditions.

Spatial Mapping

Spatial mapping or 3D reconstruction is the ability to create a digital representation of physical space while maintaining semantic information. The three most common methods to achieve 3D reconstruction are Structure from Motion (SfM), Stereopsis and Light Detection and Ranging (LiDar) [55]. The implementation of each method, however, presents great diversity in the literature and depends on the purpose for which spatial mapping is used, but also on the hardware and computational power that is available.

Researchers in [49] leverage 2D thermal images to reconstruct 3D buildings using SfM. They create point clouds from RGB images and thermal images separately and then

successfully align them to create a high-resolution output point cloud. The authors in [48] utilize the COLMAP pipeline [56] [57] and propose a novel view planning method effectively deciding the placement of a set of available RGB cameras in the 3D space in order to optimally reconstruct noisy 3D corn plants using SfM.

Recently, a novel scene representation based on Gaussian splatting [9] has been shown to deliver on-par or even better rendering and reconstruction performance than other state-of-the-art reconstruction methods. Researchers in [58] and the authors in [59] have used this representation to implement monocular camera SLAM frameworks that achieve impressive performance in camera pose estimation, map construction, and novel-view synthesis, while allowing real-time rendering of a high-resolution dense 3D map.

3D reconstruction using Stereo cameras and LiDars is also frequently encountered in the literature and has established its position in this research field in recent years. The authors in [60] propose a fusion of LiDar and Stereo camera data to effectively map 3D space without missing out on texture or volumetric details. The authors in [61] recently proposed a feature-based approach that enables fast and dense mapping of crop fields observed by a vehicle-mounted stereo camera. They leveraged Bayesian inference to tackle the feature matching ambiguity problem that is common in crop field mapping where uncertainties due to repetitive textures and uneven lighting are induced.

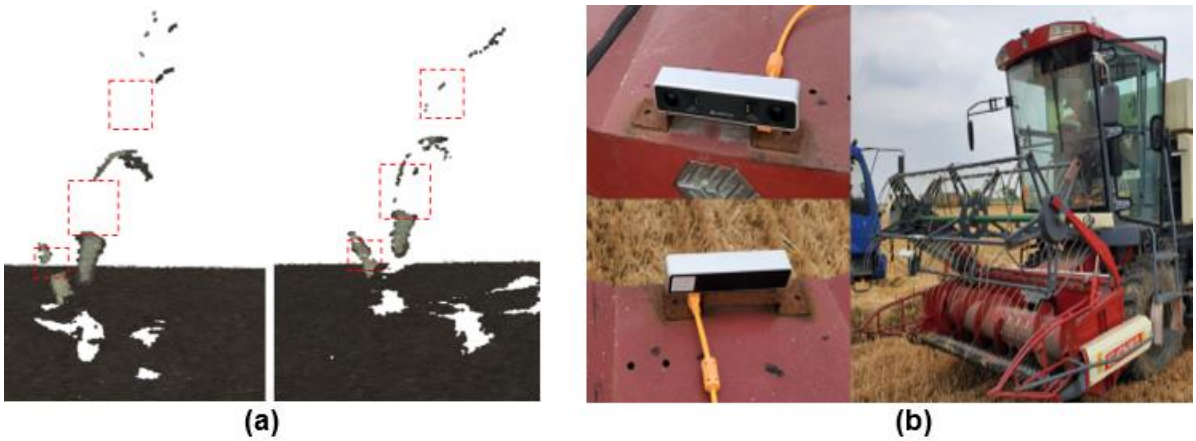


Figure 2-3: (a) Subjective quality assessment of a point cloud of a plant [40]. Left: parts of plant have not been reconstructed. Right: view planning improves reconstruction.
(b) Depth Camera on a combine harvester machine [62].

3 Reconstruction Software

3.1 SLAM

3.1.1 Localization

Robot localization, a fundamental aspect of robotics, refers to the process of determining a robot's position and orientation (pose) within a specified, known environment. With an accurate map of the environment, the robot can utilize sensor measurements (observations) to acquire knowledge about its distance from objects around it (landmarks) and use this knowledge to estimate its position and orientation in the known map. Sensors include cameras, lasers, and inertial measurement units (IMUs) which gather information about the robot's surroundings, while the map serves as a reference frame for interpreting these sensory inputs.

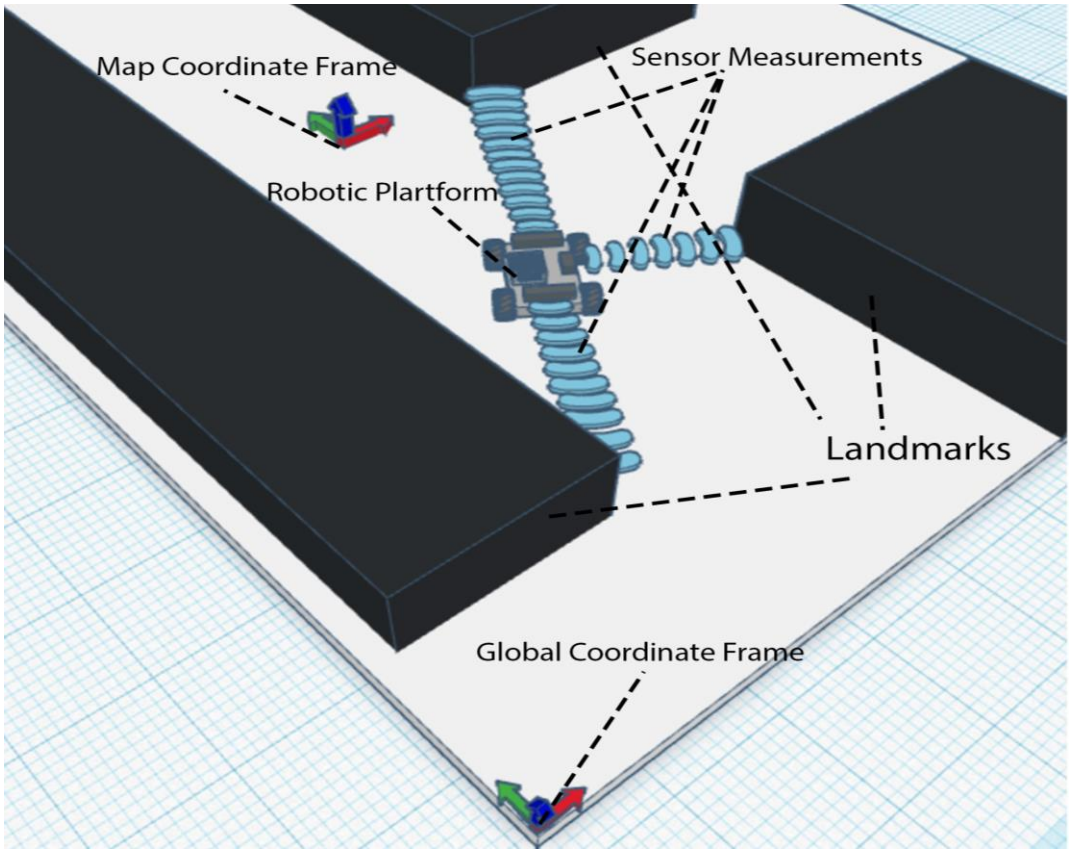


Figure 3-1: Robotic Platform Localization Illustration.

Conventional robot localization techniques often employ a probabilistic approach, where the robot's location is represented by a probability distribution, capturing the uncertainty inherent in sensor measurements. This probabilistic framework allows the robot to continuously refine its estimate of its position as it acquires new sensory data, ensuring robustness against noise and errors. Various localization algorithms have been developed, each tailored to specific sensor modalities and environmental conditions. For instance, visual odometry, relying on camera imagery, excels in indoor environments, while laser-based techniques are well-suited for outdoor scenarios due to range superiority [63].

The success of robot localization hinges on the accuracy and reliability of both sensor data and the underlying map representation. Sensor noise and environmental clutter can introduce significant challenges, requiring robust algorithms and filtering techniques to mitigate these effects. Moreover, the map itself must be adequately detailed and up to date with the environment it represents, in order to provide accurate localization information. As robotics technology advances, the pursuit of more efficient, accurate, and robust localization algorithms remains a critical area of research, paving the way for increasingly sophisticated autonomous robots capable of navigating complex and dynamic environments.

3.1.2 Mapping

In the context of robotics, mapping refers to the process of creating a comprehensive representation of an environment, typically represented as a 2D or 3D map. This representation, often referred to as a reconstruction, captures the spatial layout, objects, and other relevant features of the surroundings. Mapping is a fundamental capability for robots to navigate and operate autonomously in their environment. By constructing a map, a robot can gain a detailed understanding of its surroundings, enabling it to perform tasks such as path planning, object recognition and manipulation and environmental monitoring.

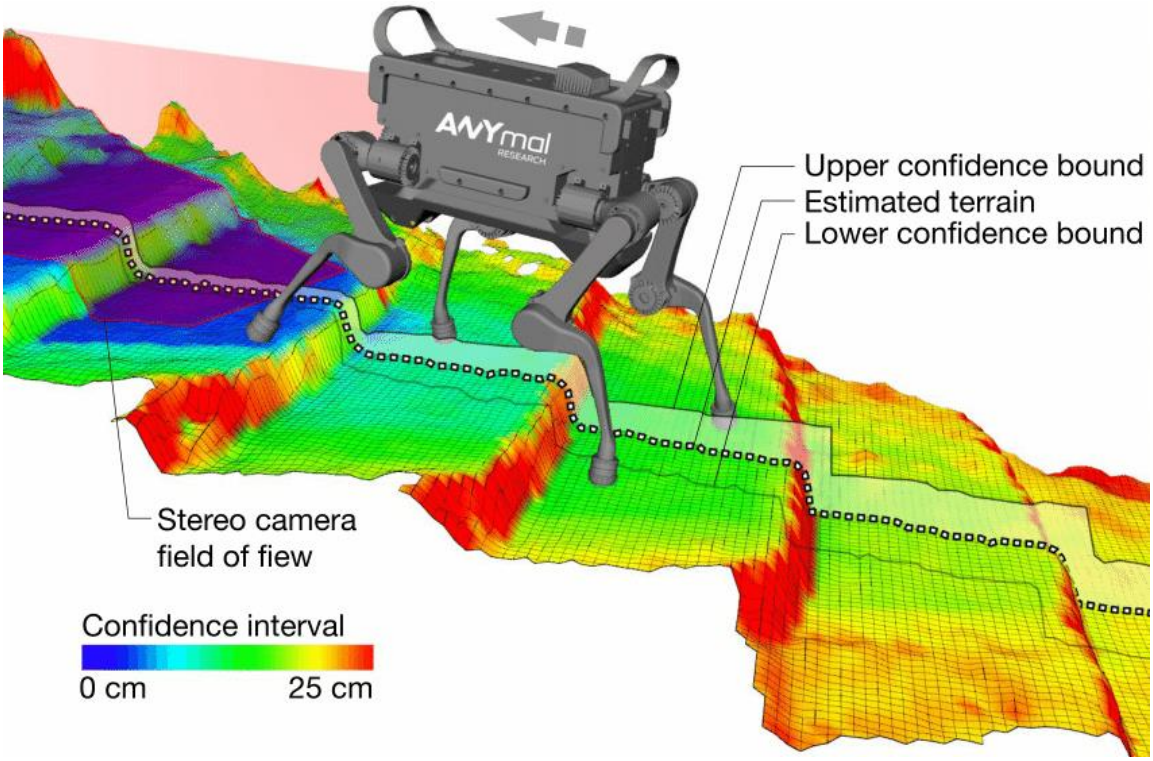


Figure 3-2: ANYmal robot mapping terrain (staircase) using a stereo camera [64].

Making a 2D or 3D reconstruction of the surrounding environment works a lot like connecting pieces of a puzzle; the robot gathers information about its surroundings using its sensors and saves that information in an appropriate file format. It then moves and repeats this process to acquire new data. Provided that it knows its position and orientation in space at any given time, i.e. it always successfully localizes itself in the environment, then it can create a map of its surroundings by appropriately stitching the newly acquired information, at its new pose, with its older representation of the environment.

3.1.3 SLAM

As discussed in sections 2.1.1 and 2.1.1, for a robot to successfully perform localization, an accurate map of the environment must be available. In addition, for a robot to successfully perform mapping of the environment, it must be successfully localized in it. Enabling a robot to perform both of these tasks simultaneously appears at first glance like a chicken-and-egg problem. SLAM stands for Simultaneous Localization and Mapping. It is the name for the solution to this very problem. It is a fundamental technique in robotics that allows a robot to build a map of its surroundings while simultaneously determining its own location within that map. This capability is essential for robots to operate autonomously in unknown environments without the need for a-priori information about the trajectory they are following or a priorly constructed map of their surroundings.

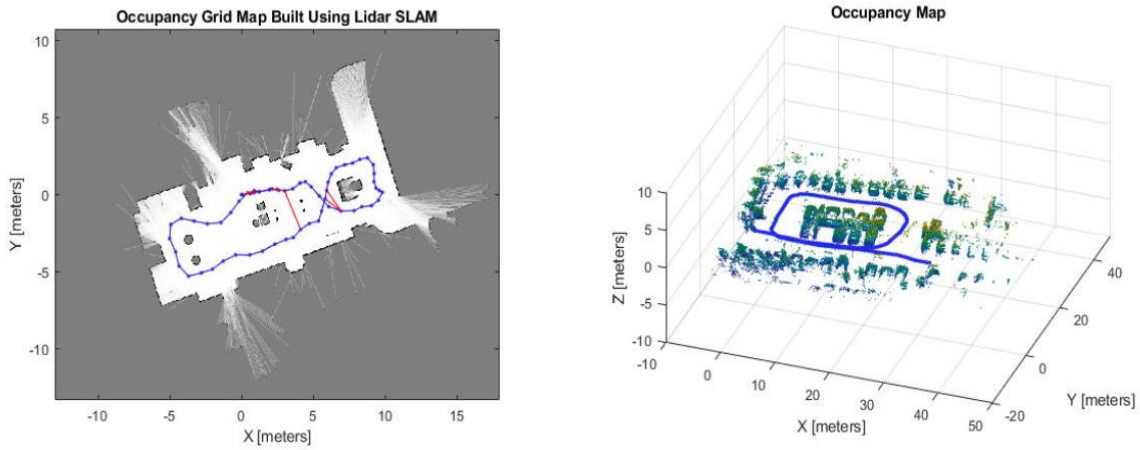


Figure 3-3: 2D and 3D Occupancy Grid Maps built using 2D and 3D Lidar SLAM utilizing MATLAB's Navigation Toolbox [65].

The key to performing SLAM is the knowledge of the relationship between two consecutive poses of a moving robot. This is otherwise known as odometry. The robot gathers information about its surroundings using its sensors and saves that information in an appropriate file format. It then moves and repeats this process to acquire new data. The robot does not know its position and orientation in space at the time of gathering the old or the new data; absolute localization is not a given. It estimates, however, its motion (transformation) from the old to the new pose. Then it can create a map of its surroundings by appropriately stitching the newly acquired information, at its new pose, with the information gathered at its old pose. There are a lot of ways to estimate the transformation from a previous to a new pose (odometry estimation). Odometry estimation is commonly achieved in the following ways:

- Wheel odometry: This method relies on measuring the rotation of the robot's wheels. The robot's position and orientation can then be estimated by tracking the cumulative rotation of each wheel.
- Inertial odometry: This method uses an inertial measurement unit (IMU) to measure the robot's acceleration and angular velocity. The robot's position and orientation can then be estimated by integrating these measurements over time.
- Visual odometry uses a camera to track features in the environment. The robot's position and orientation can then be estimated by matching the features in the camera's images to landmarks in the map.

- Laser odometry uses a laser scanner to measure distances to surrounding objects. The robot's position and orientation can then be estimated by tracking the changes in the laser scans over time.

Wheel odometry is a simple and reliable method that is well-suited for robots that have wheels or tracks. However, it is susceptible to errors due to wheel slippage and non-linearities in the robot's motion. Inertial odometry is a more accurate method that is not affected by wheel slippage. However, it is also prone to errors due to drift, which is the accumulation of IMU measurement errors over time. Visual and Laser odometry accuracy results are largely dependent on the amount of complexity of the environment of the robot, i.e. the amount of features or the shape variability of the surrounding space. To improve the accuracy of odometry estimation the above methods are often combined and statistically reinforce the total odometry accuracy with the usage of Kallman Filters.

3.1.4 VSLAM

Visual simultaneous localization and mapping (vSLAM) is a type of SLAM that utilizes visual data, typically from cameras, to determine the robot's position and orientation within an environment (visual odometry), while simultaneously building a map of that environment. build a map of an environment while simultaneously. VSLAM algorithms typically involve three main components:

- Feature extraction: This involves identifying and extracting distinctive features from the camera images. These features can be points, lines, or other geometric shapes or structures that are relatively invariant to changes in lighting and viewpoint.
- Correspondence matching: This involves matching features from subsequent images to features in the existing map. This is typically done by calculating the distance or angle between features in different images.
- Optical flow and ego-motion estimation: Optical flow, is the pattern of apparent motion of features in a visual scene caused by the relative motion between an observer and a scene. Optical flow can be used to estimate ego-motion. Ego-motion, in the field of computer vision, refers to estimating a camera's motion relative to a rigid scene [66]. An example of ego-motion estimation would be estimating a car's moving position relative to lines on the road or street signs being observed from the car itself.

VSLAM algorithms face several challenges, including:

- Illumination variations. Changes in lighting can make it difficult to extract features and match them across images.
- Occlusions: Objects in the environment can block the view of features, making it difficult to track enough of them to accurately estimate ego-motion.
- Sensor noise: Image sensors are not perfect -especially in low-light conditions- and their measurements can contain noise that can corrupt the localization estimates and accumulate over time.

Loop Closure

Loop closure is a crucial aspect of Simultaneous Localization and Mapping (SLAM) that refers to the ability of a robot to recognize previously visited places and use this knowledge to improve its localization and map estimation. This mechanism plays a critical role in

correcting the accumulated errors in the robot's pose estimation and ensuring the accuracy of the map. Without loop closure, the robot's pose estimation would gradually drift away from its actual position due to the accumulation of odometry errors. With loop closure, the robot identifies a location in space, usually by saving special visual features characteristic to this location. If it detects these features again while it is moving, then it assumes that it is in that location once more. If the odometry data state otherwise, then they are corrected accordingly and recursively along the whole trajectory of the robot. This correction is known as bundle adjustment.

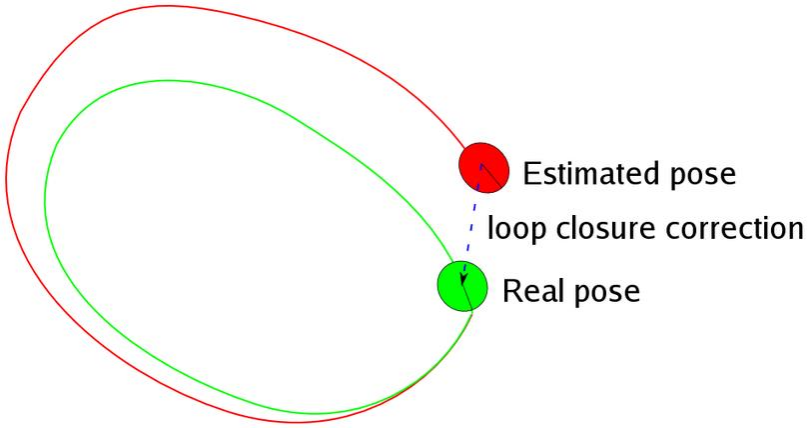


Figure 3-4: Loop Closure Illustration.

Saving visual features for a specific location in the scene and assuming they are unique to this location is not a faultless assumption. The problem that arises is called perceptual aliasing. This occurs when two different places can be perceived as the same. For example, in a texture-wise monotonous and empty building, it is nearly impossible to determine a location solely with the visual information, because all the corridors or rooms may look the same [67]. In that case, a lot of false-positive loop closure detections would disrupt the odometry data.

Multi-Camera Visual SLAM For Vineyard Inspection

A novel approach to enhancing the robustness of vSLAM has been developed in CSL, specifically tailored for vineyard inspection applications [68]. The proposed VSLAM method utilizes multiple cameras which increases the available Field of View (FoV) resulting in more features available to track. Moreover, in situations in which one or more of the cameras is obscured by the sun or leaves, the other cameras will still be able to identify useful features in the environment and thus the visual odometry estimation will not be hindered.

Figure 3-5: (a) One camera's lenses are obstructed resulting in very low number of detected features. (b) A different camera is not obstructed and continues to produce accurate visual odometry [17].

The proposed approach offers yet another novelty to add more robustness to VSLAM: It does not utilize visual features to detect loop closures, as in vineyards, the crop is organized in parallel rows and agricultural robots move in the open corridors between them. There, a purely feature based approach would result in incorrect loop detection due to the high similarity between images (perceptual aliasing). Instead, the developed method leverages AprilTags, which are placed in fixed positions in the vineyard in order to assist with loop closure detection.

AprilTags are a type of fiducial marker, similar to QR codes [69]. AprilTag markers can be easily printed and embedded in any environment. The open-source AprilTag detection software accurately determines the 3D location and orientation of each marker relative to the camera, as well as its unique identifier. The AprilTag library is written in C and requires no external libraries. It is designed to be seamlessly integrated into existing applications and can run efficiently on embedded devices. AprilTags are designed to encode far smaller data payloads than QR codes (between 4 and 12 bits), allowing them to be detected more robustly and from longer ranges. They are also designed for high localization accuracy, unlike QR codes which are mainly designed for saving data in a visual format.

Figure 3-6: April Tags used for research purposes in April Laboratory, University of Michigan [69].

A robot that uses Multi-Camera Visual SLAM For Vineyard Inspection will assume that it has reached a position that it has also visited in the past only if it registers an AprilTag for the second time. Knowing its relative position to the AprilTag using the respective open-source code, while also knowing the fixed positions of all the AprilTags inside the vineyard allows the robot to accurately re-localize and perform bundle adjustment every time it sees an AprilTag, instead of relying on loop closure detection based on visual features.

3.2 SOTA Packages

3.2.1 RTAB-Map

RTAB-Map (Real-Time Appearance-Based Mapping) is a state-of-the-art graph-based Simultaneous Localization and Mapping (SLAM) algorithm that utilizes RGB-D, stereo, and lidar data to construct a comprehensive representation of an environment in real-time [12]. Its core strength lies in its efficient loop closure detection mechanism, which effectively identifies previously visited locations and integrates them into the evolving map.

Loop closure Detector

The loop closure detector in RTAB-Map uses a bag-of-words (BoW) approach to determine how likely it is that a new image comes from a previously visited location. That means that each newly acquired image is dissected into so-called visual words. This is similar to how a text can be dissected into words. For example, in the common classification problem of labeling an email as spam or not spam, each email can be parsed into words. Building a histogram of the frequency of the appearance of each word in the mail and checking if spam-related words appear often can very accurately lead to a characterization of the mail as spam or not spam. For instance, if the word “money” appeared more frequently in the reviewed mail than in an average non-spam email, then the likelihood that the reviewed mail is spam is increased. Similarly, RTAB-Map parses each new acquired image frame into small homogenous pieces known as visual words. It creates a Bag of Words. It then compares the BoW from the newly acquired frame to older BoW which are saved in a constantly updating database of Bags. If the newly acquired BoW consists of similar visual words to a previously saved BoW, then RTAB-Map concludes that the new image comes from a location that has been visited before. This is known as accepting a loop closure hypothesis. When a loop closure hypothesis is accepted, a graph-based optimization is performed.

Graph-based optimization loop closure is a commonly used approach that represents the robot's trajectory as a connected graph, where nodes represent poses and edges represent constraints between poses. This graph-based representation allows for efficient and robust loop closure detection and correction. Pose graph optimization involves refining the poses of all nodes in the graph to minimize the overall error in the odometry data. This process considers the constraints between poses, including odometry measurements and loop closure constraints. Upon accepting a loop closure hypothesis, RTAB-Map adds a new constraint to the odometry data and the map’s graph, then a graph optimizer minimizes the errors in the map.

Performance and Integration

A memory management approach is used to limit the number of locations used for loop closure detection and graph optimization, so that real-time constraints on large-scale environments are always respected. This mainly involves map caching and map pruning [70]. Map caching is the method of storing frequently accessed map elements, such as poses and landmarks, in a separate cache memory. This reduces the need to repeatedly load these elements from disk, which can significantly improve performance. Map pruning involves periodically removing less relevant map elements, such as old poses or landmarks that are no longer considered important. This helps to keep the map size manageable and free up memory for new data.

RTAB-Map's algorithmic pipeline ensures consistent and accurate localization even in complex and dynamic environments. Its versatility extends beyond indoor mapping, as it can effectively handle outdoor environments with varying lighting conditions and challenging terrains. Its open-source nature and modular architecture make it a valuable tool for researchers and developers in robotics, autonomous vehicles, and augmented reality. Recently, a version of the package for iOS devices has been released and produces high quality results paving the way for wider commercial use of the software [12].

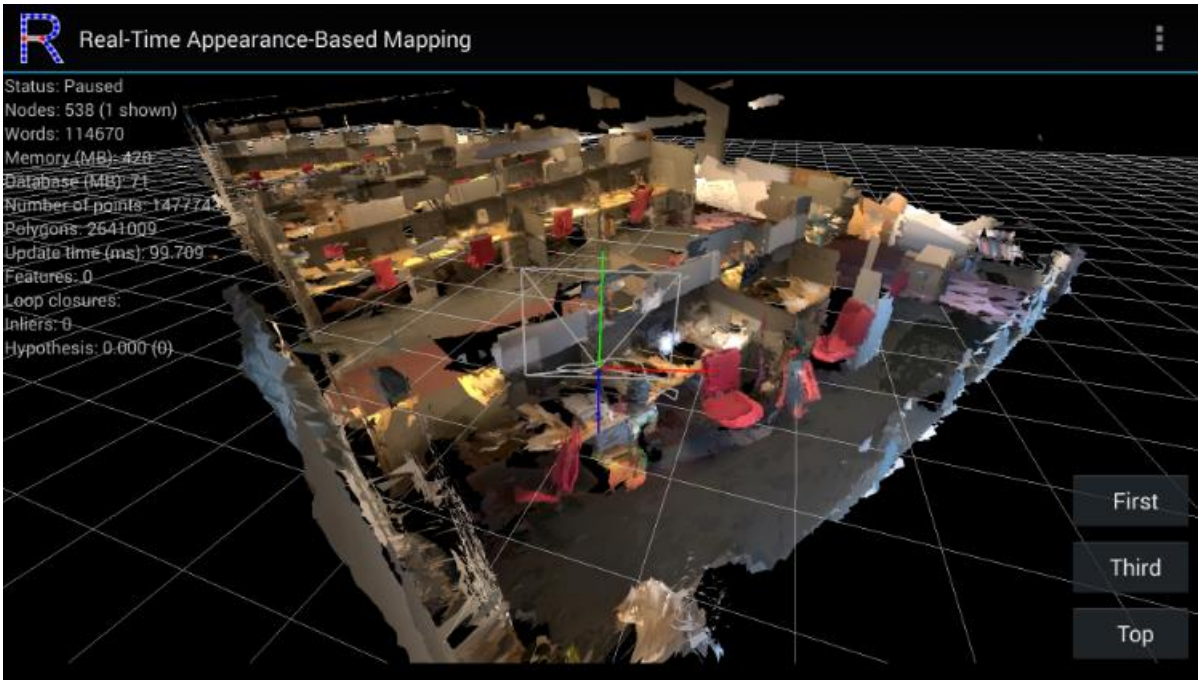


Figure 3-7: A 3D map of an office building constructed with RTAB-Map iOS application [12].

3.2.2 Robot-Centric Elevation Mapping

Robot-Centric Elevation Mapping (RCEMapping) is a ROS package developed by Anybotics as part of ANYmal Research [13]. The mapping process involves fusing range measurements from a sensor, such as a laser or structured light scanner, with the robot's pose estimation, typically obtained from an inertial measurement unit (IMU) and odometry. The integration of range data and pose information enables the construction of a consistent and reliable elevation map of the terrain around the robot.

Robot-Centric Approach

RCEMapping adopts a robot-centric perspective, constructing a local elevation map centered around the robot. This approach aligns with the inherent limitations of onboard sensors, ensuring that the map reflects the robot's direct field of view and the associated pose uncertainty. Namely, at any time, the robot-centric elevation map is a local representation of the surrounding terrain, meaning that the observed regions close to the robot - which have the highest accuracy – are registered in the map, while older, previously seen parts of the map are considered inaccurate and are deleted. Thus, uncertainty that is aggregated through the motion of the robot due to sensor noise does not result in an accumulated drift of the pose of the constructed map.

2.5D Mapping

The core data structure that RCEMapping utilizes is a voxel grid, which partitions the mapping environment into a regular grid of voxels. Each voxel represents a small volume of space, and its occupancy is determined based on the accumulated range measurements from the robot's sensor. To further manage the uncertainty associated with range measurements and pose estimation, RCEMapping employs a probabilistic voxel representation. Each voxel is assigned a probability distribution reflecting the likelihood of different terrain heights at that location. This probabilistic representation enables RCEMapping to provide meaningful estimates of terrain elevation and its associated uncertainty. To make the algorithm more computationally efficient, the OctoMap 3D occupancy mapping library which implements an octree data representation is utilized [71].

Importantly, the map constructed with Anybotics Elevation Mapping is not a 3D map of the environment, but rather a 2.5D map (two-and-a-half dimensional, alternatively pseudo-3D or three-quarter map): For every point in the plane on which the robot moves, the elevation of this point is saved and the result is plotted in 3D space. This simplifies the data but also creates problems with modelling multiple surfaces that are stacked along the z-axis. For example, if an obstacle object is placed 2 meters over a point in the robot’s field of view, then the software will assume that the elevation at that specific area is equal to the height at which this object is. It is however possible that there is enough room under the obstacle for the robot to pass through. Nevertheless, this software will render the area under the obstacle as part of the obstacle itself.

To demonstrate the above issue, a floating horizontal platform was placed in front of a turtlebot robot in a simulated demo of RCEMapping. There was enough space under the platform for the simulated robot to get through as can be seen in figure 20. However, in the elevation map the floating platform was interpreted as ground elevation and therefore as a high obstacle.

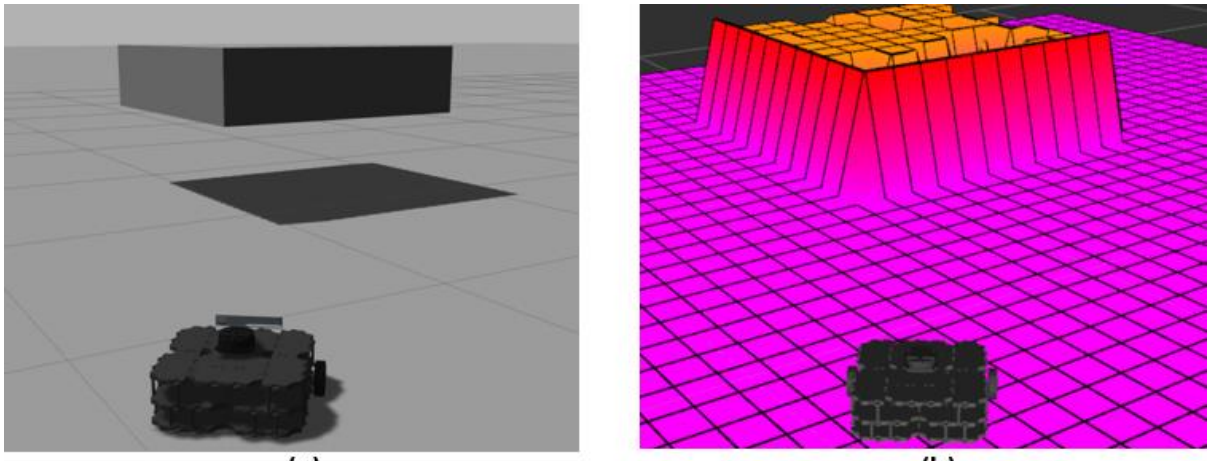


Figure 3-8: (a) A floating platform in Gazebo. (b) The elevation map constructed with RCEMapping.

Dynamic map adjustment

It is worth mentioning that the map can also adapt to dynamic environments. Namely, the map remains consistent even if objects inside the mapped environment are being moved during the mapping process. To achieve this, a visibility check is performed using ray tracing. Virtual rays are casted from the robot's sensors towards the environment. A visibility map is constructed from the points that are collected from the collision of the rays with the surface constructed from the height measurements. This map reflects the maximal height that each cell can have based on the visibility constraint. Namely, if a cell is registered as being high enough to block the sensor's visibility, but the visibility is not being blocked, then that cell violates the visibility constraint and is removed. As this visibility check is computationally intensive, it is only performed at a lower rate (e.g., 1 Hz).

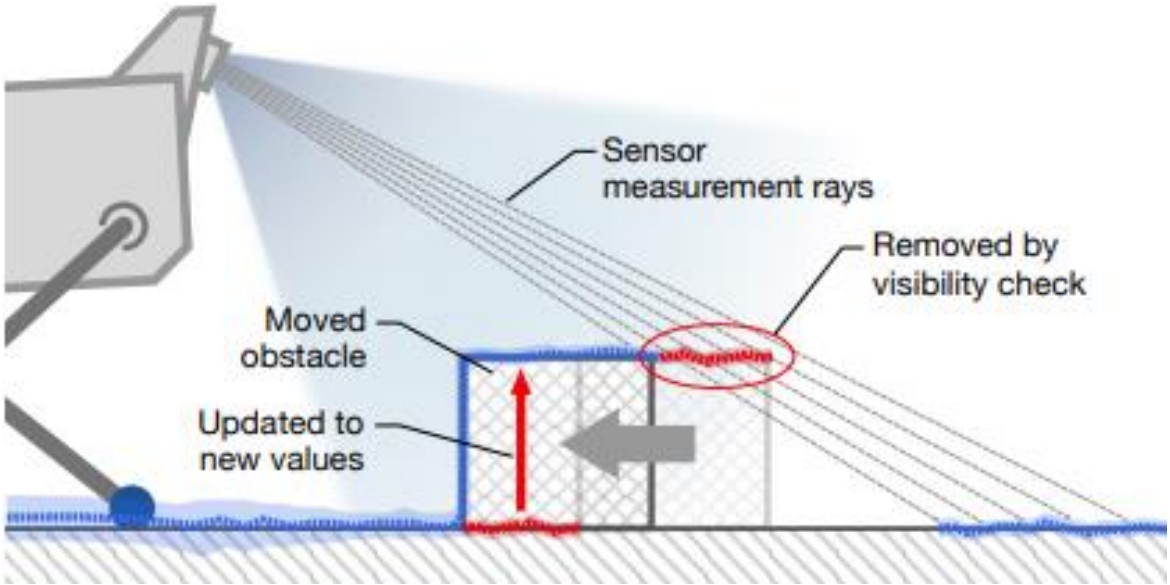


Figure 3-9: A rectangular obstacle was moved from right to left. The visibility is checked with ray tracing and the previous map (red) is accordingly updated resulting in an updated map (blue) [13].

3.2.3 Gradslam

Gradslam emerges as a novel approach to SLAM by employing differentiable optimization techniques, aiming to enable the integration of deep learning into the SLAM process. This change opens up new avenues for improving SLAM performance and adaptability.

In conventional SLAM systems, the mapping and pose estimation processes are typically decoupled, relying on individual algorithms for each task. This separation limits the ability to optimize both processes simultaneously and hinders the integration of deep learning techniques. Gradslam breaks down this barrier by representing the entire SLAM pipeline as a differentiable computational graph. This allows gradients to flow from the outputs of the system (map, trajectory) back to the inputs (raw sensor data, parameters, calibration, etc.), enabling the optimization of both mapping and pose estimation using gradient-based methods [14].

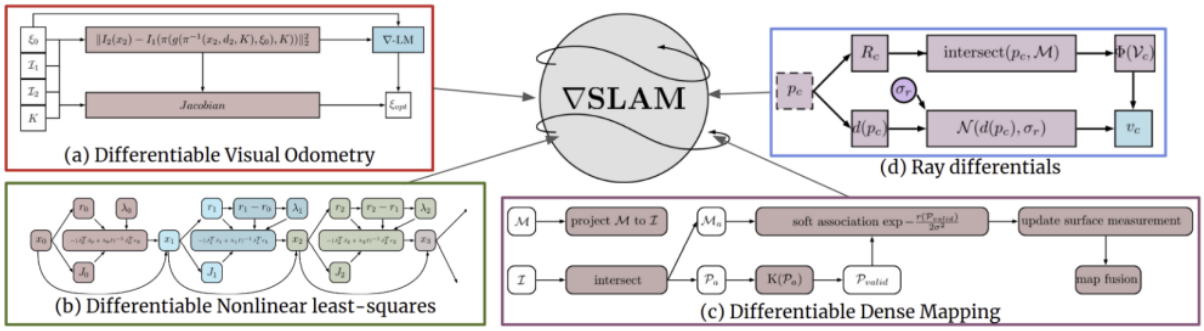


Figure 3-10: Gradslam provides differentiable building blocks for simultaneous localization and mapping (SLAM) systems. The four main blocks it offers are Differentiable Visual Odometry, Differentiable Registration using least-squares, Differentiable Mapping and Ray differentials [14].

Gradslam performs similarly to other state-of-the-art dense mapping algorithms, while offering enhanced flexibility by allowing the integration of various deep learning architectures and loss functions. However, the current implementation of dense SLAM in Gradslam requires a large amount of memory to store the produced computational graph. For instance, running the KinectFusion algorithm with a voxel resolution of 128 × 128 × 128 consumes approximately 6GB of GPU memory. This memory consumption significantly limits the size of scenes that can be reconstructed within this framework. The team that designed Gradslam [14], are currently working on improving the memory efficiency of this implementation. Additionally, they are developing more robust filters for various stages of the pipeline, such as Iterative Closest Point (ICP) registration, photometric warping, and optimization routines.

Figure 3-11: Small office scene reconstructed using Gradslam [14].

3.2.4 Zed Spatial Mapping

ZED cameras, developed by Stereolabs, are renowned for their ability to capture high-quality 3D data and perform real-time spatial mapping. Zed spatial mapping is the dedicated software developed to optimally perform SLAM by fully utilizing the camera hardware. At the heart of Stereolabs' technology lies a combination of stereo vision and inertial measurement unit (IMU) data [72].

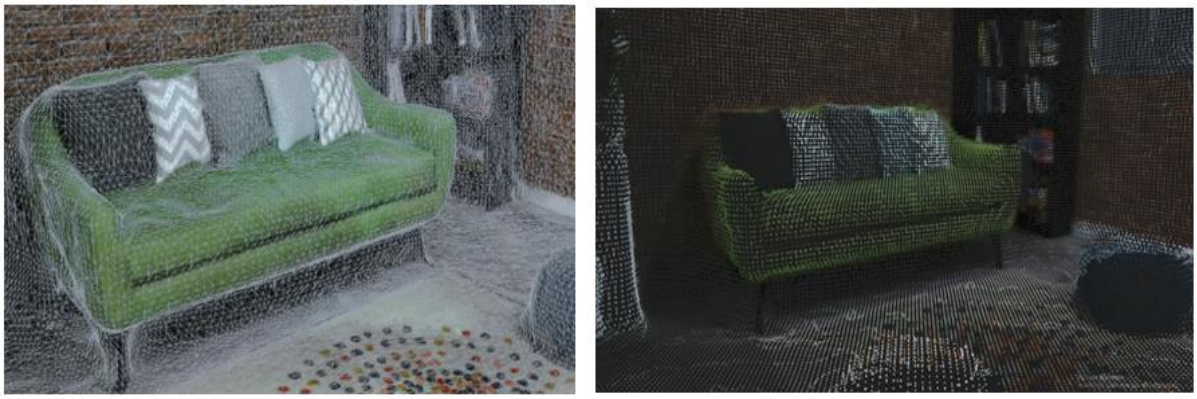


Figure 3-12: Mesh Generation (a) and Point Cloud Generation (b) with ZED Spatial Mapping.

Visual-Inertial Odometry

Visual-inertial odometry (VIO) is a technique for determining the precise position and movement of an agent, such as an aerial or ground robot, without relying on GPS or lidar. It leverages the combined power of cameras and inertial measurement units (IMUs) to achieve exceptional accuracy and robustness [73].

While stereo vision provides depth data and visual odometry, the IMU complements it by tracking the camera's movement and orientation in space. This information is essential for registering subsequent depth maps and maintaining the spatial consistency of the 3D world. The IMU captures acceleration, angular velocity, and orientation data. Those data are statistically combined (usually with the use of extended or unscented Kalman filters [74]) with visual odometry allowing the software to precisely track the camera's trajectory through space.

Neural Depth Sensing

Using deep learning algorithms to estimate depth from a single image or a sequence of images, a technique known as Neural Depth Sensing, is a rapidly growing field with the potential to revolutionize many applications. The main reason is that it allows for monocular depth estimation i.e. estimating depth using a single image. This is a challenging task, but recent advances in deep learning have made it possible to achieve relatively good results [75]. The Zed Spatial Mapping software allows the user to choose between performance-oriented, simple algorithmic stereo depth estimation techniques and neural depth estimation, which is a sophisticated approach that combines stereo and neural depth estimates. In recent software update releases, enabling the neural depth estimation while using Zed Spatial Mapping produced faster and cleaner results.

The Multi-Step Process of Building the Spatial Map

The spatial mapping process encompasses several key steps [15]:

- Feature Detection and Matching: The software identifies distinctive features in each depth map, such as corners or edges. These features are then matched across subsequent images, creating a network of correspondences that stitch together the individual depth maps.
- Planar Segmentation: The software identifies and segments planar surfaces in the scene, such as walls, floors, or ceilings. This process simplifies the representation of the environment and reduces computational complexity.
- Mesh Generation: The software constructs a 3D mesh from the depth data, representing the reconstructed environment as a collection of interconnected triangles. This mesh provides a polygonal approximation of the scene, allowing for efficient visualization and interaction.
- If area memory is enabled, a database that stores information about the environment that the ZED camera has already mapped is created. This information includes a point cloud representation of the environment, as well as information about the camera's pose (position and orientation) at different points in time. When the tracking detects an already-visited area by searching in the database, it will perform a loop closure and compute an updated position estimation that cancels eventual drifts.

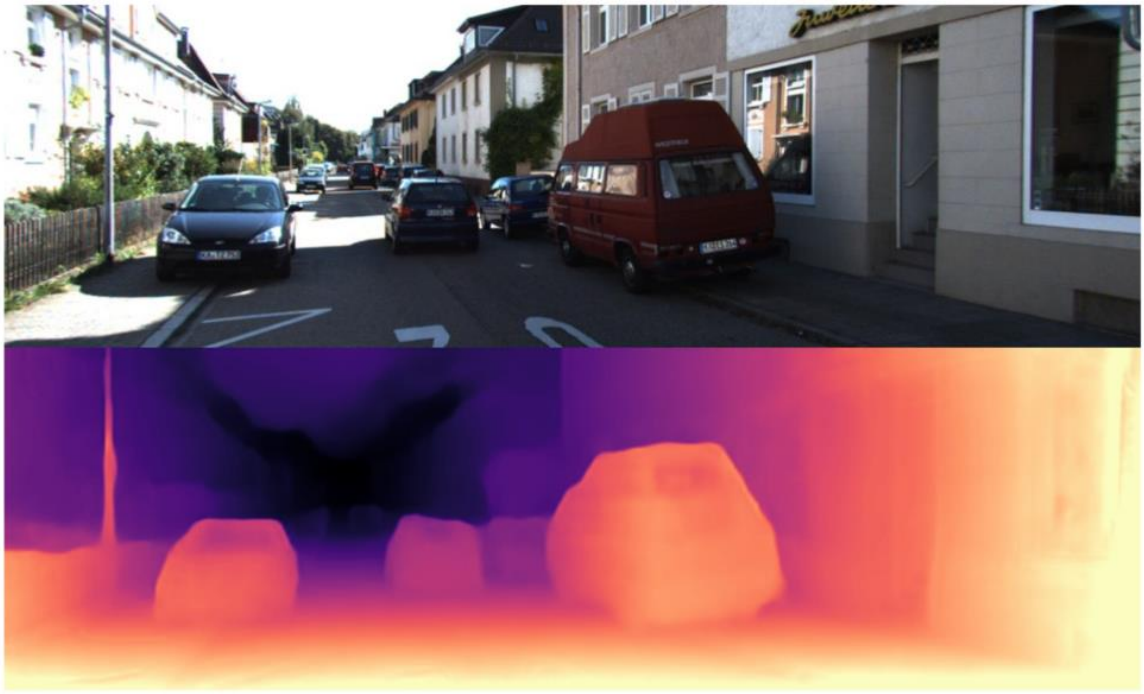


Figure 3-13: Successful Monocular Depth Estimation [75].

3.3 Comparison and Decisions

3.3.1 Comparison

Overall, Rtabmap is a good choice for robotics applications that require high accuracy, robustness, and flexibility. Anybotics Elevation Mapping is well-suited for vineyard mapping due to its efficiency in handling elevation changes which are commonly encountered in mountainous vineyards but fails to capture critical details in the canopy and fruit. Gradslam is a good choice for research and development purposes, especially for machine learning SLAM applications. Nevertheless, it significantly lacks real time performance and memory management, as well as integration and ease of use. Zed Spatial Mapping is an excellent software for prototyping due to its and real-time performance and high-quality Application Programming Interface (API). However, it can only be used with Stereolabs hardware and is not open-source.

Table 3-1 shows a detailed comparison of the four SLAM software packages for vineyard mapping.

Table 3-1: A comprehensive comparison of four SLAM software packages.

Feature	RTAB-map	Robot-Centric Elevation Mapping	Gradslam	ZED Spatial Mapping
Active development	Actively developed, maintained, and upgraded by a large community	Maintained by Anybotics	Maintained by a community of researchers	Actively developed, maintained, with frequent, major updates

Community support	Large and active community with extensive documentation and support forums	Smaller community with good support forums	Community with limited activity in recent years	Dedicated community with extensive documentation and good support forums
Real Time Performance	Performs well in real-time applications. High memory requirements for loop closure	Very efficient for mapping environments with elevation changes	Slow in real-time applications. Very high memory requirements	Very efficient for real-time applications with Zed Stereo cameras
Reconstruction Quality	High quality 3D reconstruction with accurate feature matching and mapping	Can only produce 2.5D, undetailed reconstruction.	High quality reconstruction for simple environments	High quality 3D reconstruction with a variety of output data representation.
Specific Features	Supports a wide range of sensors, including LiDAR and depth cameras	Designed specifically for 2.5D terrain mapping with Octomap representation. Great memory management	Fully differentiable Graph-based SLAM system.	Specifically designed for Zed Stereo cameras. Easy integration.

3.3.2 Decisions

Robot-Centric Elevation Mapping by Anybotics will encounter challenges in vineyards due to the presence of dense vegetation and other dynamic obstacles that cannot be accurately rendered with a 2.5D map representation. It also fails to capture texture and minor details which are crucial for crop inspection applications.

Gradslam's flexible mathematical structure comes at the expense of real-time performance in vineyard mapping applications. Real-time 3D mapping is often crucial in vineyard mapping tasks, such as navigation and obstacle avoidance. Gradslam's potential for lower real-time performance limits its suitability for these applications. It is also the software that enjoys the least active development, maintenance, and support.

In the context of vineyard mapping, Rtabmap emerges as the preferred choice if a ZED Stereo camera is not available. Its versatility, robustness, and accuracy make it well-suited for the challenging conditions often encountered in vineyards. In addition, it comes with extensive documentation, frequent updates and active support forums. Conversely, if a ZED Stereo camera is available, ZED Spatial Mapping provides a compelling alternative. Its

streamlined integration and real-time 3D mapping capabilities make it efficient for rapid prototyping and applications where real-time feedback is desired. ZED Spatial Mapping is actively maintained and supported and it is regularly updated with state-of-the-art algorithmic integration.

4 Path Planning

Path planning stands as a fundamental cornerstone of robotics, enabling autonomous robots to navigate complex and dynamic environments with precision and efficiency. It is the process of determining a collision-free trajectory for a robot to move from a starting point to a designated goal while adhering to various constraints, such as obstacle avoidance, terrain features, and kinematic limitations [76].

Path planning remains a highly active area of research, with ongoing efforts to develop more efficient, robust, and versatile algorithms that can handle increasingly complex and challenging environments. The advancements in path planning algorithms hold immense promise for the future of robotics, paving the way for autonomous robots that can seamlessly integrate into our everyday lives.

4.1 Taxonomy of Planners

4.1.1 Global and Local path planners

Path planning algorithms can be broadly classified into two categories: global and local. Global path planning algorithms, also known as off-line planning algorithms, generate an entire path from the start to the goal before the robot begins its movement. These algorithms excel in open and predictable environments. However, they struggle in dynamic environments with rapidly changing obstacles or unpredictable terrain disturbances.

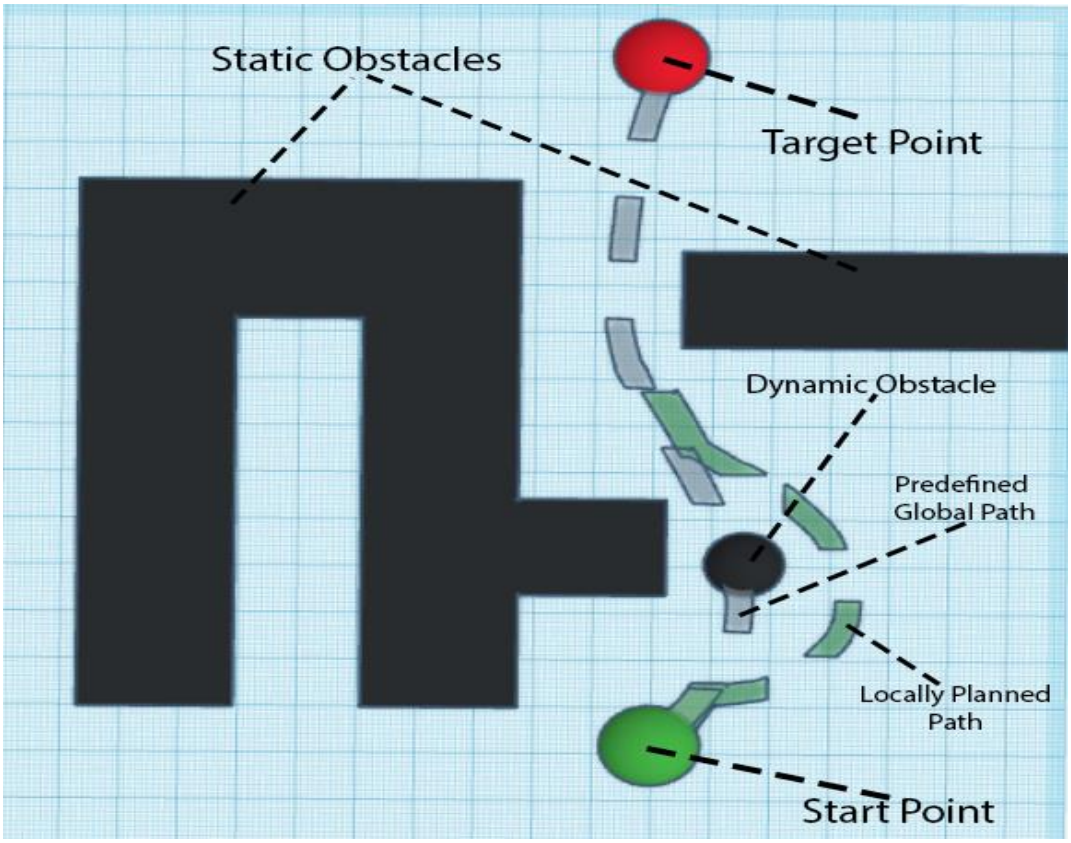


Figure 4-1: Combination of Global and Local Planner Illustrated.

In contrast, local path planning algorithms, also known as on-line planning algorithms, generate paths in real-time as the robot navigates the environment. These algorithms continuously update the map of the environment and adapt the path to avoid obstacles encountered along the way. The predicted path in a local path planner is in principle much shorter than that of a global path planner and concerns the robot's vicinity. They are well-suited for dynamic environments, where obstacles may move or appear unexpectedly and are usually combined with global path planners to enable robots to reach a designated goal avoiding collisions along the way.

4.1.2 Obstacle Representation

Based on how free space and obstacles are mathematically represented, path planning algorithms are further categorized in four main categories:

- Grid-based or Search Based planners: These planners discretize the environment into a grid and use search algorithms like A* or Dijkstra's to find a path through the grid. They are efficient for simple environments but can be computationally expensive for complex environments, as the map on which the robot operates must be extensively subdivided to form a grid which effectively models the operating space [77].

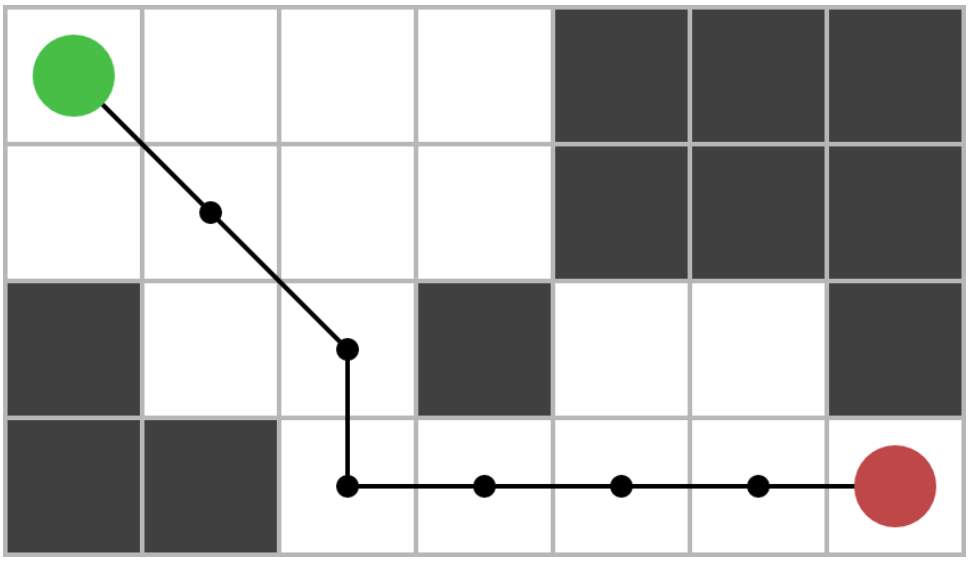


Figure 4-2: Grid-based free space (white squares) and obstacle (dark squares) representation and viable path from starting position (green circle) to target position (red circle).

- Sampling-based planners: These planners build a tree of possible paths in the environment by randomly sampling free space, creating a network that connects the samples (nodes) and checking for collisions in the paths between the samples (edges). They can handle complex environments with obstacles but may not always find the optimal path to the target.

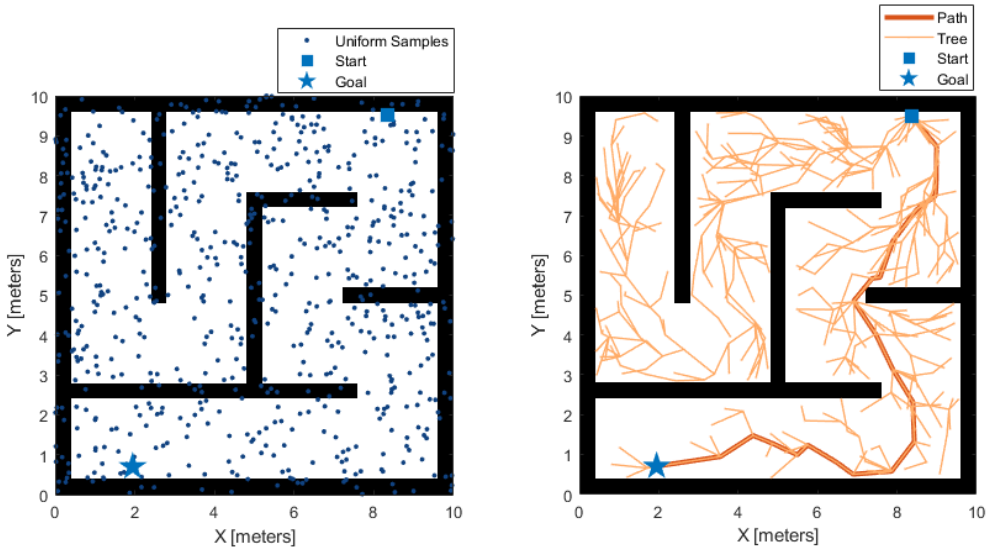


Figure 4-3: Rapidly-exploring Random Tree* (RRT*), a common sampling-based path planner that builds a tree of potential paths by randomly sampling points in the environment and checking for collisions [78] .

- Potential field planners: These planners create a potential field that represents the attractive force towards the goal and the repulsive force from obstacles. The robot moves along the gradient of the potential field to reach the goal. They are simple and efficient but may not be suitable for all environments. Inherent problems to potential field path planners include local minimum traps and unwanted oscillations around obstacles [79].

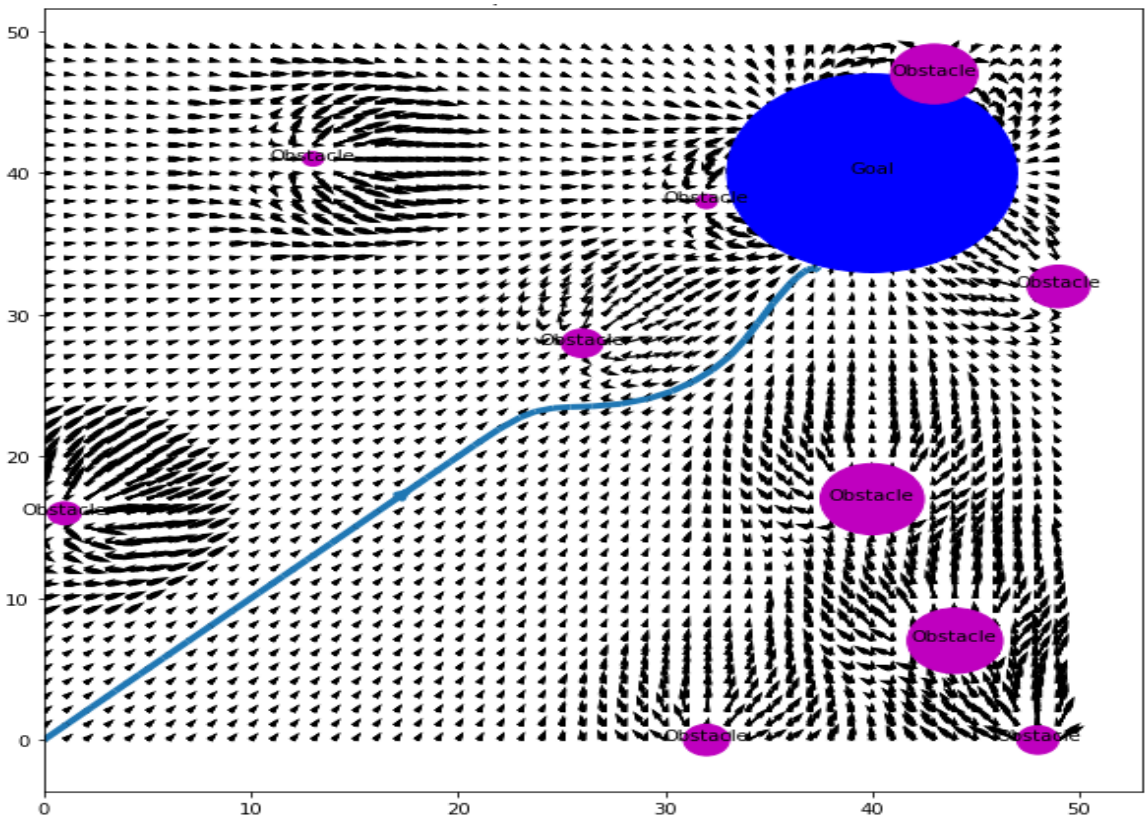


Figure 4-4: Potential Field Planner Visualization. Environment with 10 obstacles [79].

- Hybrid planners: These planners combine elements of the first three planners utilizing sophisticated algorithms to alternate between approaches and achieve considerable flexibility.

4.1.3 Exploratory Path Planners

Exploratory path planning is a branch of path planning that aims to navigate a robot effectively through an unknown or partially known environment to collect information about its surroundings. The goal of an exploratory path planning agent is not to reach a single specific goal efficiently, but rather to maximize the amount of information gathered by exploring the environment. Hence the name, exploratory path planning.

In this context, computing an optimal path before starting the exploration is often not a viable option as there is not enough useful information about the operating environment for a-priori waypoint or goal setting. It is therefore common for explorational path planners to employ complex and automated goal setting schemes. Based on their goal setting scheme, explorational path planners can be categorized as follows:

- Next-best-view (NBV) planners: NBV planners select goals based on the robot's current perception of a specific target in the environment. They identify locations that will provide the robot with more information about that target, such as locations that will provide them with higher visibility or locations that will allow the robot to see new, so far obscured parts of the target. For example, the authors in [80] created an NBV planner to enable a robotic arm to calculate the optimal trajectory around fruits to effectively reconstruct their complete shape. The algorithm predicts fruit shapes prior to mapping them and computes targeted viewpoints to enable the robot to observe yet unobserved parts of the fruits.

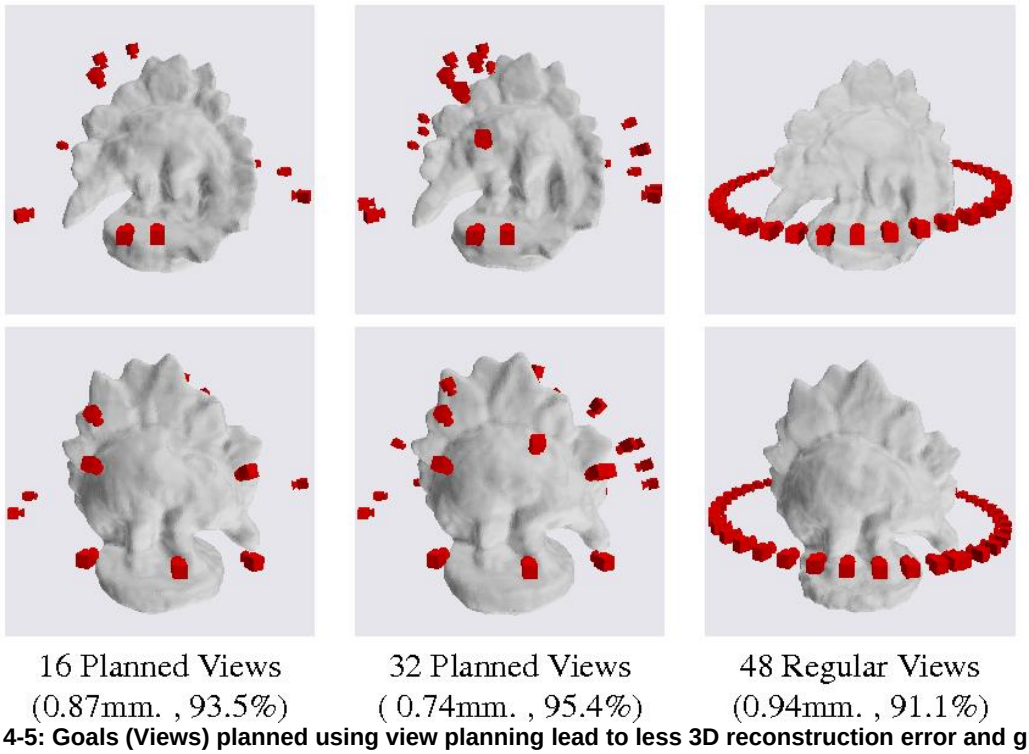


Figure 4-5: Goals (Views) planned using view planning lead to less 3D reconstruction error and greater object completeness percentage than regular views [81].

- Coverage-based planners (CPP): Coverage-based planners aim to maximize the amount of useful area covered by a part of the robot or scanned by a sensor of the robot. It is a particularly important task in the context of agriculture robotics, especially for automated harvesters and watering robots. CPP is however an integral algorithm for a plethora of applications including cleaning robots, underwater vehicles creating image mosaics, demining robots and more [31].

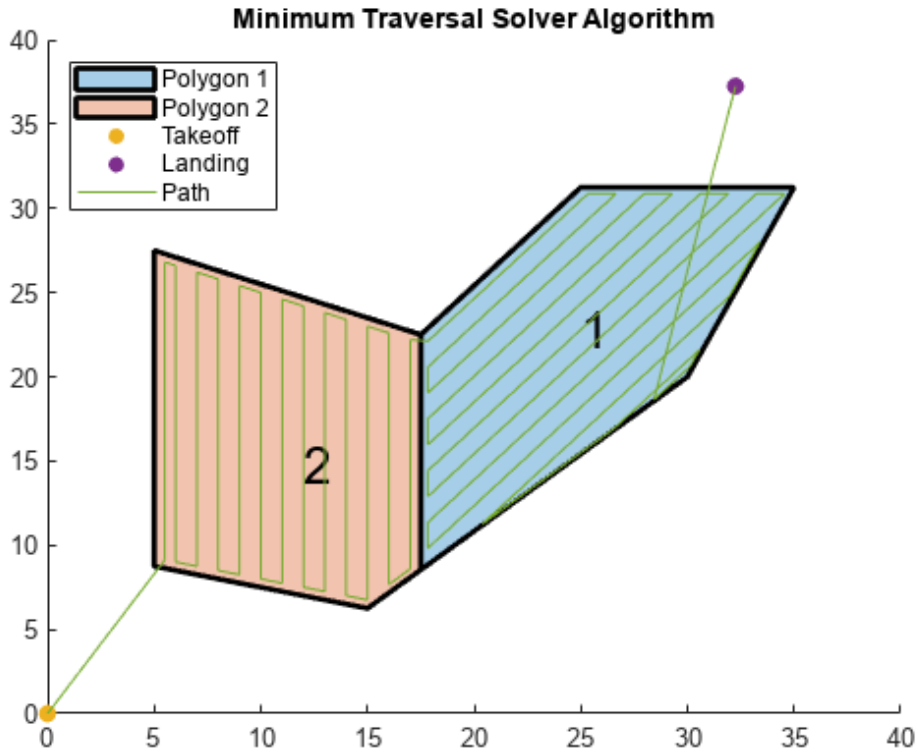


Figure 4-6: Solution for a UAV coverage path planning problem in Matlab calculated using Matlab's UAV Toolbox [82].

- Information-theoretic planners: Information-theoretic planners use information theory to quantify the amount of information gathered by the robot. They plan paths that maximize the expected information gain from each new candidate waypoint, which is the amount of information the robot can expect to gather by visiting it [83]. Waypoints that hold significant exploratory value are often called frontiers. Such planners are commonly used in search and rescue or other time-sensitive applications.

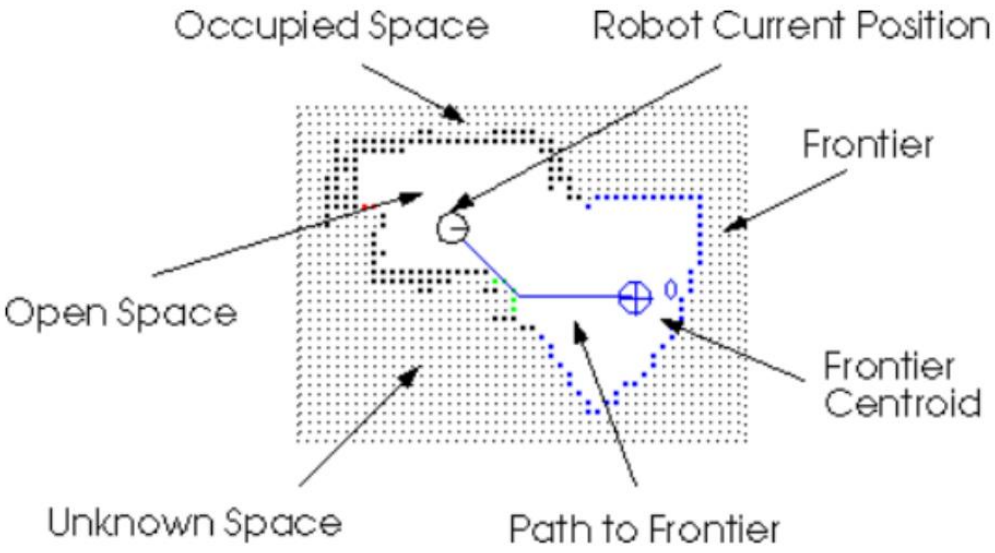


Figure 4-7: Information-Theoretic Exploratory Planner illustration [84].

4.2 SOTA Packages

4.2.1 Ewok Planner

In their work, Ewok: Real-Time Trajectory Replanning for MAVs using Uniform B-splines and 3D Circular Buffer [18], the authors present a real-time approach for local trajectory replanning specifically designed for agile robots such as Micro Aerial Vehicles (MAVs). Unlike traditional methods that assume static environments and prior map knowledge, Ewok thrives in dynamic scenarios, efficiently adapting trajectories on-the-fly to navigate cluttered and unpredictable surroundings. This makes it particularly valuable for applications like search and rescue, autonomous exploration, and spatial mapping in unknown environments.

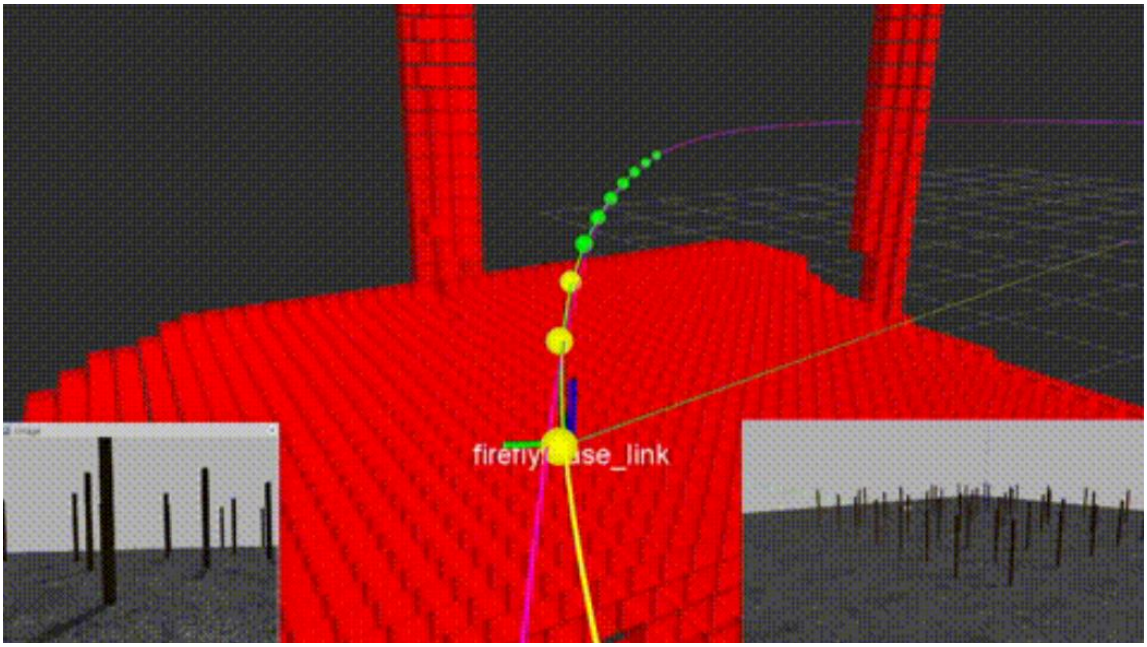


Figure 4-8: MAV dynamically planning its path while moving through a simulated forest using the Ewok Planner [18].

Uniform B-splines

Ewok represents trajectories as smooth and flexible curves using uniform B-splines. Formally, a spline is a piecewise polynomial function defined over a set of intervals [85]. The places where the polynomial pieces meet are known as knots. Splines are a powerful tool when there is a need to represent smooth curves that pass through or interpolate given data points due to the following attributes:

- Continuity: Splines have continuity of at least order k at each knot point, meaning both the function value and its derivatives up to order k match across intervals.
- Locality: Each polynomial piece only affects the curve in its corresponding interval, allowing for local control of the shape.

The degree of the polynomial pieces determines the spline's smoothness and flexibility. For example, a cubic spline (k=2) will have continuous first and second derivatives, creating smooth and visually appealing curves.

B-splines (basis splines) are a specific type of spline representation. B-splines of order n are basis functions for spline functions of the same order defined over the same knots, meaning that all possible spline functions can be built from a linear combination of B-splines, and there is only one unique combination for each spline function. They have several desirable properties:

- Each B-spline is non-zero only over a specific interval defined by its corresponding knots.
- The sum of all B-splines associated with a given knot sequence is always 1.
- B-splines can be defined recursively based on lower-degree B-splines, simplifying calculations.

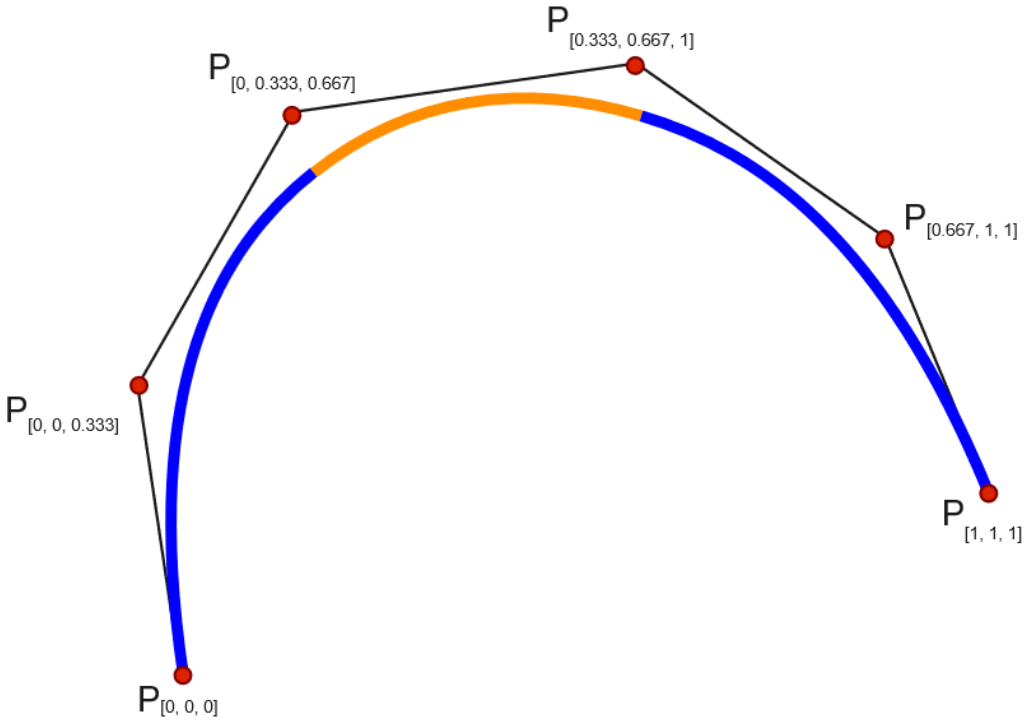


Figure 4-9: A cubic parametric polynomial spline[86]. P denotes control points. The first and third polynomic parts of the curve are painted blue, while the second orange. Single knots at 1/3 and 2/3 of the curve establish a spline of three cubic polynomials meeting with C^2 parametric continuity. Triple knots at both ends of the interval ensure that the curve interpolates the end points.

The value of a B-spline of degree $k - 1$ can be evaluated using the following equation:

$$\boldsymbol{p(t) = \sum_{i=0}^{n} p_i B_{i,k}(t)}\,, \tag{4-1}$$

where $p_i \in R^n$ are control points at times $t_i, i \in [0, \dots, n]$ and $B_{i,k}(t)$ are basis functions that can be computed using the De Boor – Cox recursive formula [87]. Uniform B-splines are a special case where the knots are equally spaced within the time domain. This simplifies calculations and offers predictable control over the curve's shape. In the context of the Ewok planner, uniform B-splines are utilized for calculating an initial global path.

Leveraging splines allows the ewok planner to produce smooth paths, preventing abrupt changes in velocity and acceleration, essential for safe and comfortable agile robot maneuvers. Furthermore, modifying a global trajectory defined with splines is not complicated, as modifications to individual control points affect only a specific segment of the trajectory, allowing for targeted adjustments without impacting the entire path. The authors of the Ewok planner have specifically employed a real-time trajectory modification applied on the global path computed with B-splines. This is achieved through a local planner which solves an optimization problem and is triggered when the robot detects an obstacle in its vicinity and on its computed trajectory. Eq. (4-2) describes the cost function which must be minimized when the local planner is triggered:

$$\boldsymbol{E_{total} = E_{ep} + E_c + E_q + E_l}\,, \tag{4-2}$$

where E_{ep} (endpoint position) is a cost function that expresses position and velocity deviation from the optimal values that are imposed by the global trajectory; E_c (collision) is a cost function that heavily penalizes collision with obstacles; and E_q and E_l are cost functions that ensure continuous and smooth derivatives (cost of integral and limit on the norm, respectively), preventing abrupt changes in velocity and acceleration, essential for safe and comfortable MAV maneuvers.

B-splines guarantee continuous position and its derivatives up to the degree of the spline minus one at any given control point and the basis functions simplify calculations, enabling real-time replanning within onboard computational constraints. However, B-splines do not naturally interpolate control points, hindering imposition of arbitrary boundary conditions. Only static constraints (zero-time derivatives) can be guaranteed by duplicating a control point ($k + 1$ times, where k is the spline degree). Non-zero time derivative constraints necessitate iterative optimization. Other trajectory representations, such as Polynomial Splines allow direct enforcement of boundary conditions, including non-zero time derivatives, at the expense of increased complexity.

When control points originate only from planning algorithms (RRT, PRM) requiring collision-free paths, adhering to these points is crucial. Therefore, representations such as polynomial splines are more suitable. Local replanning employed in Ewok deals with unexpected obstacles not considered in initial planning and changes the control points in real-time. Here, strict adherence to control points becomes less critical, and the inherent smoothness and computational efficiency of B-splines make them preferable.

3D Circular Buffer

To maintain real-time environmental awareness, Ewok employs a 3D circular buffer that moves along with the MAV. This buffer stores occupancy grid information, essentially representing a dynamic map of the surrounding obstacles within a defined range. Sensors like LiDAR or cameras continuously update the buffer, reflecting changes in the environment as the MAV navigates. By querying the buffer during replanning, Ewok identifies potential collisions and modifies the trajectory accordingly. The buffer is implemented by employing the following strategies:

Discretization: The environmental volume is discretized into voxels of size r. This establishes a mapping between points in 3D space p and integer-valued indices x that uniquely identify individual voxels. The inverse operation allows retrieval of the voxel center point given its index. A continuous array of size N represents the 3D environment. An offset index o defines the location of the buffer's coordinate system relative to the MAV. Given the index and offset, the following functions can be defined to check if a voxel lies within the represented volume and determine its address within the stored array:

$$\boldsymbol{insideVolume}(\boldsymbol{x}) = \mathbf{0} \leq \boldsymbol{x} - \boldsymbol{o} < \boldsymbol{N}, \tag{4-3}$$

$$\boldsymbol{address}(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{o})\boldsymbol{mod}\boldsymbol{N}. \tag{4-4}$$

Robotcentric Update: To maintain the buffer centered around the MAV's camera, the offset (o) is simply updated based on the vehicle's movement. The newly incorporated part of the volume is cleared, eliminating the need for large data copies during movement. This way, the size of the array can be restricted to $N = 2^p$ and the above functions can be altered to use cheap bitwise operations instead of divisions:

$$\boldsymbol{insideVolume}(\boldsymbol{x}) = !\left((\boldsymbol{x} - \boldsymbol{o})\&\left(\sim(\mathbf{2}^{\boldsymbol{p}} - \mathbf{1})\right)\right), \tag{4-5}$$

$$\boldsymbol{address}(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{o})\&(\mathbf{2}^{\boldsymbol{p}} - \mathbf{1}), \tag{4-6}$$

where & is a “bitwise and”, ~ is a “bitwise negation” and ! is a “boolean not”.

If a collision is imminent, the local planner identifies the point of conflict and generates alternative segments using B-splines. These segments prioritize obstacle avoidance while adhering to constraints like minimum altitude, maneuverability limits and proximity to the original trajectory. The new segments are seamlessly connected to the existing trajectory, ensuring a smooth and continuous path.

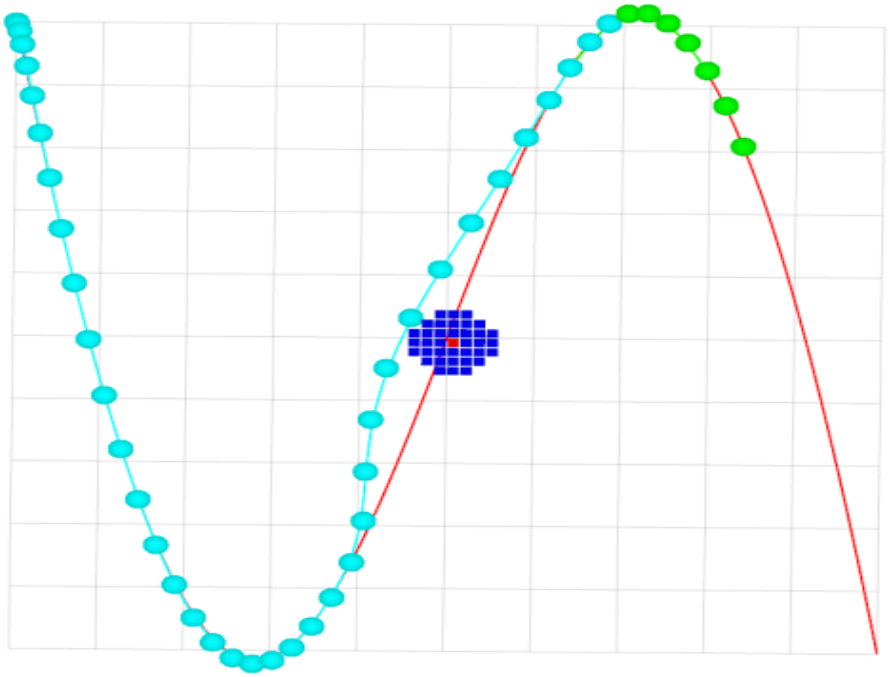


Figure 4-10: Example of online trajectory replanning using the ewok planner [18]. The plot shows a global trajectory computed by fitting a polynomial spline through fixed waypoints (red), voxels within 0.5 m of the obstacle (blue), computed B-spline trajectory with fixed (cyan) and still optimized (green) segments and control points.

Ewok presents a valuable approach for real-time trajectory replanning in dynamic environments for agile robots. One of its limitations is the planning horizon: The 3D circular buffer limits the planning scope to the immediate surroundings. Long-range planning might require additional techniques. In addition, a smooth curved trajectory is especially useful when dealing with fast-moving robots, like MAVs. In the case of a quadruped agriculture robot, a trajectory with angles and sharp turns is perfectly acceptable and can even be optimal.

4.2.2 Sequential MPC Reactive Planning using Safe Corridors

In their work, A Sequential MPC Approach to Reactive Planning for Bipedal Robots [19], the authors delve into a path planning solution that leverages the power of Sequential Model Predictive Control (MPC) and offline polytopic decomposition to enable robust and reactive motion for legged robots in complex and dynamic scenarios.

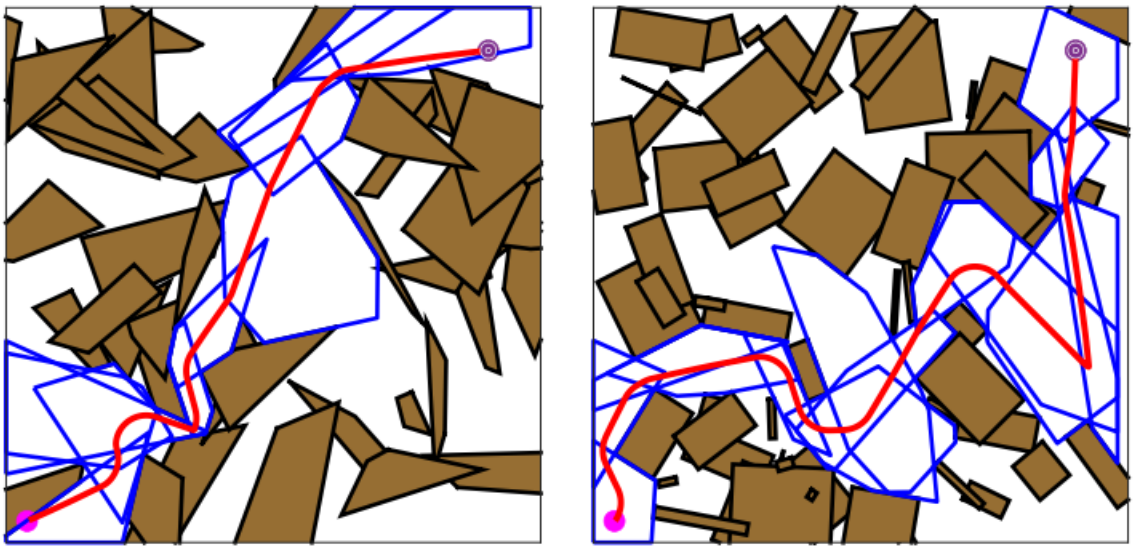


Figure 4-11: Example environments and paths generated by the MPC-safe corridors controller [19]. The successively connected polytopes (blue) represent safe corridors. (a) Polygonal obstacles (b) Rotated rectangular obstacles.

Safe Corridors

The authors leverage RRT* to efficiently explore the environment and identify obstacle-free regions. RRT* iteratively expands a tree-like structure in the configuration space, prioritizing exploration towards the goal while respecting robot constraints. Figure illustrates the logical flow diagram for RRT*, given a set state space X, in which the robot operates, a start point $Xstart$ and a goal point $Xgoal$.

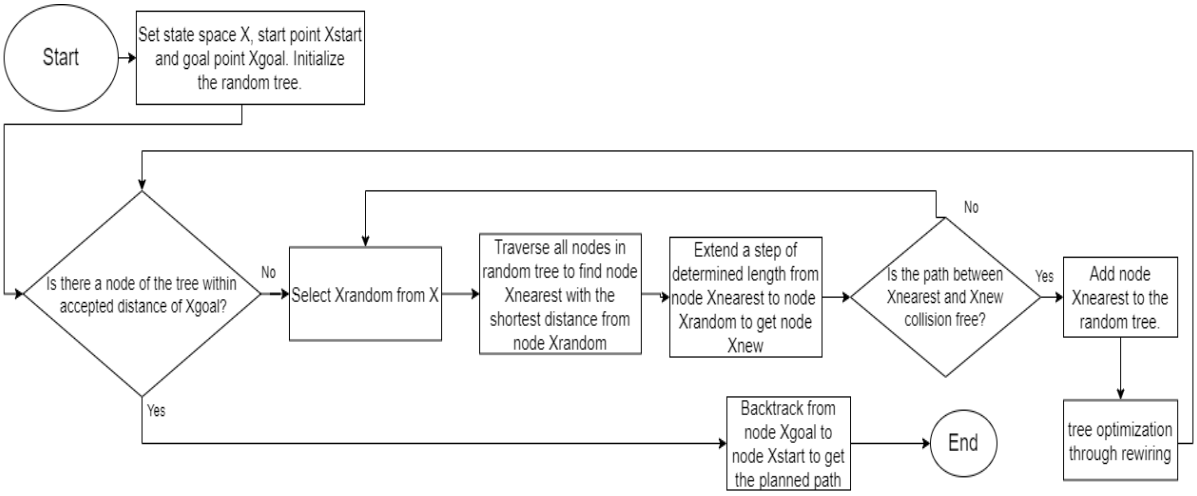


Figure 4-12: RRT* logical flow diagram.

It should be noted that RRT* is the same algorithm as RRT with the added optimization routine after each new node addition. This is a three-step routine that employs the following functions:

Local Search: Starting from the newly created node, its neighboring nodes are checked within a specific radius.

Cost Comparison: For each neighbor, the routine checks if connecting to the new node would create a lower cost path compared to its current connection. Cost can represent distance, time, or any other relevant metric. Algorithms like Dijkstra's or Kruskal's are employed for path cost calculation.

Rewiring: If a lower-cost path is found, the routine rewires the connections of the affected neighbors to connect them through the new node.

The resulting tree is a near-optimal path solution.

Instead of directly using the complex free space identified by RRT*, the authors decompose it into simpler, convex shapes called polytopes. This decomposition leverages computational geometry techniques to ensure safe and tractable representation of free space. The individual polytopes are then carefully connected through a recursive process that uses their Chebyshev centers, to form a continuous and collision-free "corridor" that guides the robot towards its goal. This corridor prioritizes directness while adhering to safety constraints and robot limitations. The algorithm follows the below structure:

Obstacle Inflation: The static workspace, denoted as Ws, is first augmented to account for the physical dimensions of the mobile robot. Obstacles are inflated by a specified radius, assuming the robot can be represented by a disc of that size.

***Convexity Assumption*:** All obstacles in Ws are assumed to be convex. While not universally true, this simplification allows for efficient computation of free space regions for motion planning.

***Sampling-Based Planning*:** A sequence of points, $\Pi = \{p_1, \ldots, p_k\}$, is generated within the free space of the workspace (dynamic and static), denoted by $W \backslash Ws$. This sequence connects the initial location with the desired goal position and is generated using RRT*.

Initial Polytope Generation and Waypoint Extraction: An initial polytope H_0 is created and saved within the free space $W \backslash Ws$ containing the initial robot position y_0. This is done using a convex optimization process [88]. y_0 is saved as the initial waypoint w_0. The algorithm checks if the goal position w_g lies within H_0. If not, the algorithm iterates through the points $p_j \in \Pi$ starting from j = 1. It finds the first point p_{j1} not contained in H_0.

Intersecting Polytopes Guarantee: The algorithm takes the current polytope H_0 and the waypoint p_{j1} as input. It generates a new polytope H_{new} around p_{j1}. It then performs two operations:

Intersection Check: It checks if the current polytope and the polytope H_{new} generated around p_{j1} have a non-empty intersection ($H_0 \cap H_{new} \neq \emptyset$).

Intersection Guarantee: If an intersection exists, the algorithm saves H_{new} as a new polytope H_1 and a waypoint w_1 representing the Chebyshev center of the intersection $H_0 \cap H_{new}$. If there is no intersection, the algorithm generates a sequence of intersecting polytopes and waypoints $\bar{G}$ connecting H_0 and H_{new} by iteratively creating polytopes using points on the straight line that connects y_0 with w_1. The generated polytopes and waypoints are saved. The algorithm repeats this process for each point $p \in \Pi$.

Final Output: The algorithm outputs a sequence of pairwise intersecting free polytopes $\{H_0, H_1, \ldots, H_{M-1}\}$ containing the initial position and a set of waypoints $\{w_1, \ldots, w_M\}$ such that:

$$w_i \in H_{i-1} \cap H_i \quad \text{for} \quad i = 1, \ldots, M-1 \tag{4-7}$$

and $w_M = w_g \in H_{M-1}$. These waypoints and polytopes 0are to be utilized in subsequent Model Predictive Control (MPC) algorithms for robot motion.

Sequential Model Predictive Control (MPC)

Model Predictive Control (MPC) is an advanced control technique widely used in various industries, from chemical plants and refineries to autonomous vehicles. It offers a powerful framework for controlling complex systems while incorporating constraints and optimizing performance goals [89].

At the heart of MPC lies a mathematical model representing the dynamics of the system being controlled. This model can be linear, nonlinear, or a combination, depending on the system's complexity. MPC operates over a finite time window called the receding horizon. At each time step, it predicts the system's future behavior based on the current state and potential control inputs. An optimization problem is formulated with a cost function that penalizes deviations from desired outputs and control effort. By minimizing this cost function, MPC determines the optimal control sequence for the receding horizon. Only the first control input in the sequence is applied to the actual system. At the next time step, the horizon moves forward, incorporating new sensor measurements to update the model and repeat the optimization process. This is what the sequential aspect of this approach implies. In their work [19], the authors leverage MPC to optimize a rough initial trajectory created by the RRT* algorithm. The MPC controller optimizes a cost function that considers factors like trajectory smoothness, control effort, and distance to the goal. This ensures efficient and goal-oriented navigation within the created safe corridors.

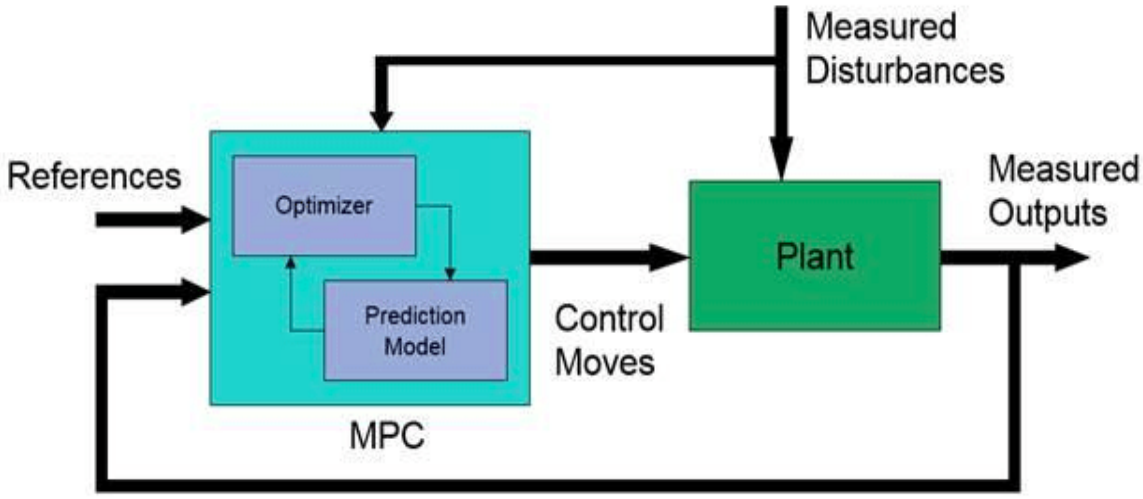


Figure 4-13: Model Predictive Control Schematic.

The core concept behind the planning and control approach in the work by the authors in [19] lies in the following equation for the $i-th$ MPC ($MPC(i)$):

$$\boldsymbol{h_{i,j}(Ax_k + Bu_k) \geq (1-\gamma)h_{i,j}(x_k),\quad j = 1,\dots,l_i\ ,} \tag{4-8}$$

where x_k represents the current state of the robot model (in [19] it is a 3D-LIP robot model) at time step k. The state encompasses its position (x, y), orientation θ, and velocities $(\dot{x},\ \dot{y})$, u_k denotes the control input applied to the robot model at time step k. This input directly influences the acceleration and subsequent trajectory of the robot. $h_{i,j}$ represents a collection of smooth scalar-valued functions. Each function $h_{i,j}$ corresponds to a specific half-space within a polytope C_i. It acts as a measure of how far the robot state x_k is from the

boundary of that half-space. The polyhedra C_i define safe regions within the workspace. The robot's state and trajectory must remain within these polyhedra throughout its motion to guarantee collision-free navigation. Each polytope C_i is formed by the intersection of multiple half-spaces. Each half-space is defined by a linear inequality represented by the function $h_{i,j}$. l_i represents the number of half-spaces that contribute to defining a specific polyhedron C_i.

The core constraint utilizes the functions $h_{i,j}$ associated with a polytope so that if $h_{i,j}(x_k) \geq 0$ for all j, the robot's state lies within the safe region C_i.
The constraint enforces that the control input u_k steers the robot's predicted next state $x_{k+1} = Ax_k + Bu_k$ further into the polytope region or at least keeps it within the boundaries. This ensures collision avoidance with static obstacles, while respecting the robot model's constraints.

Reactive Planning

While the robot follows the current safe corridor segment, the MPC controller continuously replans the subsequent segments in the sequence. This allows for real-time adjustments based on dynamic obstacles or changes in the environment, ensuring reactive collision avoidance and smooth trajectory adaptation. Additionally, the authors propose incorporating a "reactivity layer" within the Safe Corridor generation itself, making the corridors themselves adaptable to moving obstacles without complete replanning, by incorporating sensor input in the corridor formation process. This is because a new polytopic decomposition can be performed in less than a second, even in complex and cluttered environments.

4.2.3 Graph-based exploration planner (GB-planner)

In their work, Graph-based Subterranean Exploration Path Planning using Aerial and Legged Robots [20], the authors propose an approach employing path planning specifically designed for robotic exploration in large-scale, tunnel-like, underground networks, enabling efficient and safe navigation. GBPlanner 2.0 is an extension of their previous work on a Graph-Based Exploration Planner. The new planner presents improved computational performance and better handling of positive and negative obstacles for ground robots. The planner does not require any prior knowledge of the environment other than the general bounds of the volume to be explored.

Graph Representation

Much like in RRT*, graph nodes are placed in random locations of the subterranean environment, in close vicinity to the robot which is performing the exploration. Nodes that are placed on obscured locations or on obstacles are discarded, while the rest of the nodes are connected to form a tree-like structure which depicts potential paths from the robot to each node. Ground traversability is also considered as a constraint during this process. The graph representation simplifies the complex 3D environment into a structured and easily analyzable format.

Exploration Gain Metrics

To guide the robot's decision-making, various metrics are assigned to graph nodes. These metrics quantify the potential information gain associated with exploring a specific node.

Nodes that lie near uncharted territories are ranked higher in terms of explorational value. Metrics reflecting terrain difficulty, narrow passages, or potential hazards are also introduced to influence path selection for safe navigation. The robot iteratively calculates the explorational value of all the nodes in the so-far explored map. It then decides which node to visit next, based on its explorational value and the length of the path it must take to get there.

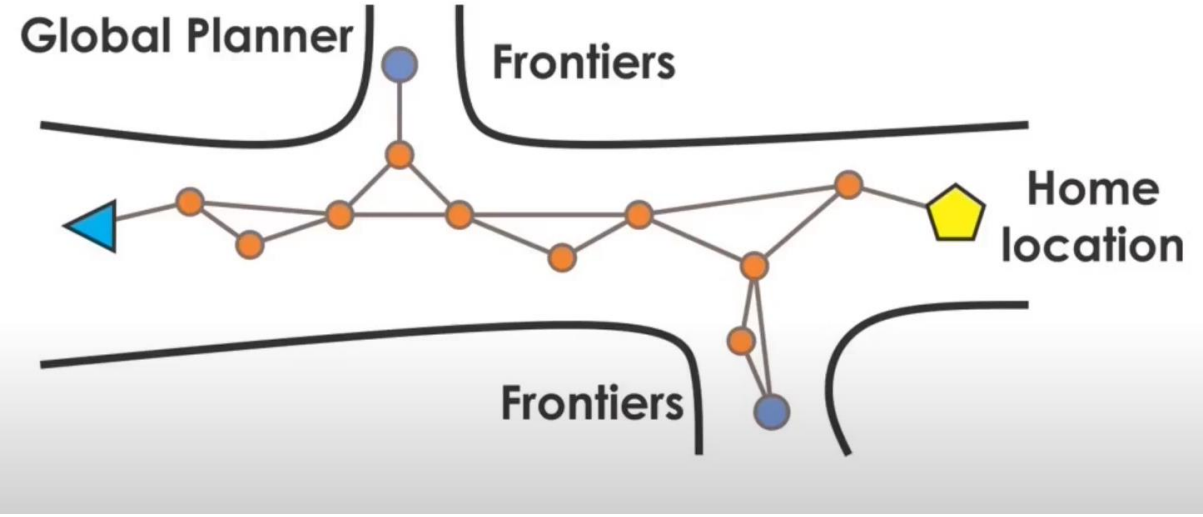


Figure 4-14: A graph representation is used to describe free space [20]. Frontiers and Home Location affect the robot's (blue triangle) decision making process.

Nodes that held great explorational value but have not been visited due to their distance from the robot are saved as frontiers. A frontier in explorational path planning is defined as the boundary between known and unknown space in a map. It can be visualized as the edge of the explored area, where information about the environment is still missing. The robot knows the path from its position to all explored frontiers at any moment. It decides to visit nodes close to a frontier if there are no new nodes in its vicinity with a high enough explorational value.

The authors of GBplanner2.0 specifically employ two separate exploration gain metrics; one for the local planner and one for the global planner. Both approaches utilize a volumetric gain calculation algorithm. The proposed method for calculating volumetric gain leverages ray casting. Given the robot's current pose and a specific 3D sensor model, the algorithm identifies:

Unknown space: The unmapped portion of the environment.

Sensor frustum: This represents the cone-shaped volume which represents the sensor's field of view.

Voxels: The environment is discretized into small, 3D cubes called voxels.

The algorithm then performs the following steps:

Ray Casting: For each direction within the sensor frustum, a ray is cast from the robot's position into the unknown space.

Voxel Intersection: Each ray is checked for intersections with voxels in the unknown space.

Traversable Voxels: The number of intersected voxels is identified, representing the potentially traversable volume within the sensor's field of view.

This approach provides a computational and efficient estimation of the unmapped volume accessible to the robot, aiding in exploration gain estimation. In particular, given a path $\sigma_i \in \Sigma_i, i = 1, \dots, n$, - where Σ_i is the set of all shortest path in the map- , with number of vertices

m_i and a set of vertices $v_j^i \in \sigma_i, j = 1, \dots, m_i$ along the path, the equation that describes the exploration gain estimation process for the local planner is the following:

$$\boldsymbol{ExplorationGain}(\boldsymbol{\sigma_i}) = \boldsymbol{e^{-\gamma_S S(\sigma_i, \sigma_{exp})} \sum_{j=1}^{m_i} VolumetricGain(v_j^i) e^{-\gamma_D D(v_1^i, v_j^i)}}, \quad \textbf{(4-9)}$$

where $S(\sigma_i, \sigma_{exp}), D(v_1^i, v_j^i)$ are weight functions with tunable factors $\gamma_S, \gamma_S > 0$. In addition, $D(v_1^i, v_j^i)$ is the cumulative Euclidean distance from a vertex v_j^i to the root v_1^i along the path σ_i. This promotes efficient exploration by prioritizing paths that require minimal robot travel distance but at the same time maximize the anticipated information gain (e.g., unmapped space coverage) per unit travel distance, leading to a faster exploration rate. Similarly, the equation that describes the exploration gain estimation process for the global exploration planner is:

$$\begin{aligned} &\boldsymbol{GlobalExplorationGain_G}(\boldsymbol{v_{G,i}^F}) \\ &= \boldsymbol{T}(\boldsymbol{v_{G,cur}}, \boldsymbol{v_{G,i}^F}) \boldsymbol{VolumetricGain}(\boldsymbol{v_{G,i}^F}) \boldsymbol{e^{-\varepsilon_D D(v_{G,cur}, v_{G,i}^F)}}, \end{aligned} \quad \textbf{(4-10)}$$

where F symbolizes a frontier, G symbolizes the combination and clustering of all graphs on the map, called the global graph and $\boldsymbol{T}(\boldsymbol{v_{G,cur}}, \boldsymbol{v_{G,i}^F})$ is the estimated remaining exploration time if the planner chooses to go from the current vertex ("cur") to the frontier vertex $\boldsymbol{v_{G,i}^F}$. The T parameter is derived by approximating the robot's Remaining Endurance Time (RET) and subtracting the Estimated Time of Arrival (ETA) required for the two following traversals:

1. Travel from the current vertex to the designated vertex ($\boldsymbol{v_{G,i}^F}$).
2. Return travel from the designated vertex ($\boldsymbol{v_{G,i}^F}$) back to the home location ($v_{G,home}$).

Thus, the above becomes:

$$\boldsymbol{T}(\boldsymbol{v_{G,cur}}, \boldsymbol{v_{G,i}^F}) = \boldsymbol{RET} - \boldsymbol{ETA}(\boldsymbol{v_{G,cur}}, \boldsymbol{v_{G,i}^F}) - \boldsymbol{ETA}(\boldsymbol{v_{G,i}^F}, \boldsymbol{v_{G,home}}) \quad \textbf{(4-11)}$$

Conceptually, this value represents the tentative volumetric gain achievable within the remaining time T if the exploration planner decides to explore frontier $\boldsymbol{v_{G,i}^F}$. This gain is an estimate of the unmapped space the robot can potentially cover during this timeframe, given its energy consumption constraints.

Search Algorithms & Multi-level Planning

Efficient algorithms traverse the graph, selecting paths that optimize a combination of these exploration gain metrics while adhering to robot constraints like battery life, maneuverability, and communication range. To address large-scale environments, hierarchical planning approaches are often employed. A high-level planner creates a coarse roadmap in the formed graph using simplified models, while a lower-level planner refines the path within specific regions based on detailed sensor data for safe and efficient exploration. This technique is known as multi-level planning. In their work, the authors of GB-planner 2.0 utilize multi-level planning not only by employing both a global and a local planner but also

by employing a different exploration gain metric for each planning level, as shown in the previous paragraph.

4.3 Comparison and Decisions

For robot navigation within a vineyard, path planning approaches must address the unique spatial structure and constraints of this environment. A spline-based planner, like the ewok planner, while efficient for simple trajectories, is unsuitable for navigating across multiple vineyard rows due to its limited global planning capabilities. Obtaining a path that allows for multi-row scanning with a mobile robot, while simultaneously avoiding obstacles is a process that demands using both a global and local planner.

An exploration-based planner designed for dynamic environments, such as GB-planner, would work in most environments, complex or simple. However, it becomes redundant in the mostly static context of a vineyard. Pre-mapping the vineyard allows for a more efficient approach, eliminating the need for re-exploring the vineyard every time the robot is tasked with a new inspection procedure. Creating an accurate 2D representation of the vineyard and providing it as input for the utilized planner is the most efficient approach. In addition, finding the optimal path through the entire explored environment can be computationally expensive for large-scale scenarios, when using GB-planner.

Therefore, after evaluating various path planning packages, the Sequential MPC Reactive Planning using Safe Corridors was selected based on its suitability for our specific use case. Unlike the Ewok planner or the GB-planner, this planner combines the ability of planning trajectories that allow the robot to traverse multiple rows, with the addition of efficient optimal path estimation and dynamic obstacle avoidance. The free corridors aspect matches the inherent corridor-like structure of vineyard rows, enabling fast and reliable polytope calculation and free space estimation, leading to safe robot movement.

5 Vision System

For agricultural robots tasked with vineyard inspection, the hardware comprising the perception system is crucial. This system enables the robot to navigate, identify obstacles, and accurately assess vine health. Its importance stems from the inherent complexity of the vineyard environment: dense foliage, varying lighting conditions, and potential presence of pests and diseases demand precise and reliable data acquisition. However, the effectiveness of this system hinges not just on the choice of sensors, but also on their optimal positioning on the robot and their integration with the computing platform. Carefully selecting sensors tailored to the specific inspection needs, strategically placing them for comprehensive coverage, and ensuring compatibility with the processing power of the embedded platform are all essential for a successful vineyard inspection robot. Only through a meticulously designed perception system can an agricultural robot achieve the level of awareness and adaptability required to navigate and analyze the dynamic tapestry of the vineyard.

5.1 Sensing

Standard digital cameras output images as a 2D grid of pixels. Each pixel has values associated with it –Red, Green, and Blue, or RGB. Each attribute has a number from 0 to 255, so black, for example, is (0,0,0) and a pure bright red would be (255,0,0). Thousands to millions of pixels together create the kind of photographs we are all very familiar with. To produce a 3D reconstruction of the surrounding environment, on the other hand, pixels which have a different numerical value associated with them are needed. That number is the distance from the camera, or depth. The three most prominent technologies that are commonly utilized for capturing depth are lidar sensors, depth camera sensors and standard digital cameras utilized for photogrammetry. It was necessary to carefully consider the strengths and weaknesses of each technology to reach an informed decision about the perception system for our robot.

5.1.1 Lidar Sensors

LiDAR, an acronym for Light Detection and Ranging, operates like an echolocation system for light. It emits pulsed laser beams and measures the time it takes for the reflection to return, calculating the distance to objects in its environment. This precise method creates highly detailed 3D point clouds, offering valuable insights for diverse applications. Currently, two main types of LiDAR sensors dominate the scene: mechanical and solid-state.

Mechanical LiDAR sensors

Mechanical LiDARs utilize laser beams guided by rotating mirrors to scan their surroundings, offering high accuracy but require moving parts which can cause potential wear and demand energy to move. In addition, mechanical LiDARs scan the visible scene point by point, gradually building a 3D reconstruction. This can lead to "jitter" if the scene or sensor moves during a single scan. A LiDAR ego-motion correction method must be used to mitigate this error [90].

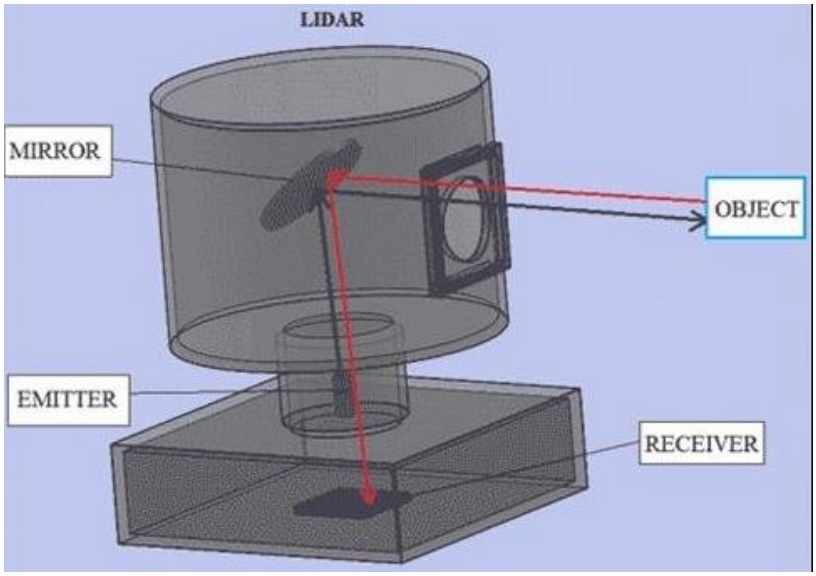


Figure 5-1: 3D arrangement of a typical LiDAR sensor [91].

Solid-state LiDAR sensors

Solid-state LiDARs, on the other hand, eliminate moving parts and promise increased reliability, durability, and energy efficiency. The three most prominent types of Solid-state LiDARs are Flash LiDARs, MEMS-based LiDARS and optical phased array LiDARs.

Flash LiDAR is a technology that illuminates the entire scene in a single pulse by diverging a laser beam, unlike conventional LiDAR's point-by-point scanning. Both technologies use time-of-flight sensors to measure how long the laser takes to bounce back, revealing distances. But while conventional LiDAR uses a single point sensor, Flash LiDAR employs an array of pixels, each recording distance and intensity. The result can be described as taking a 3D photo, not with color, but with distance as the detail. The utilization of an instantaneous pulse (flash) means that these LiDARs can capture high-resolution 3D images smoothly, without jitter, even in dynamic scenes. However, the powerful burst must be eye-safe, limiting wavelengths and driving up costs. Standard image sensors can't easily read these wavelengths, demanding expensive gallium-arsenide alternatives [92].

While not entirely solid-state due to rapidly spinning silica-based mirrors, Microelectromechanical Mirrors (MEMS) LiDARS offer benefits in size and cost that are comparable to Flash LiDARS. They employ a single laser directed at a tiny, rapidly rotating mirror that scans the scene like a high-speed kaleidoscope. However, MEMS scanners primarily operate in one direction (left-to-right). Creating a 2D scan typically requires an additional mirror moving up-down or another laser at a different angle. Also, vibrations and shocks can disrupt the delicate MEMS mirror, potentially requiring recalibration [93].

The third family of solid-state LiDARS, phased array LiDARS, can illuminate any direction by using a microscopic array of individual antennas. Controlling the timing (phase) of each antenna steers a cohesive signal in a specific direction. This technology has been employed in conventional radars since the 1940s. The same technique can be used with light and promises lower cost and higher efficiency than mechanical LiDAR sensors without sacrificing robustness or 3D data quality.

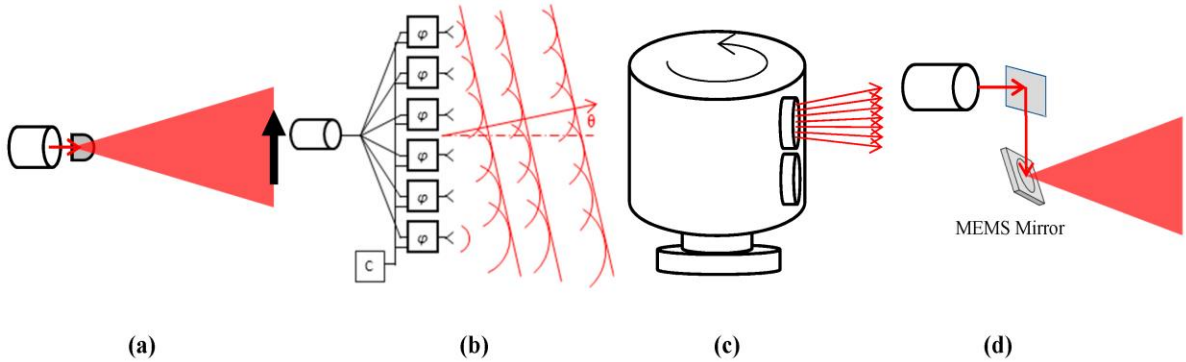


Figure 5-2: (a) A flash LiDAR with diffused light; (b) The principle of an optical phased array (OPA) scanner; (c) A LiDAR motorized spinning scanner; (d) A microelectromechanical mirrors (MEMS) laser scanner [94].

5.1.2 Depth Cameras

Depth cameras play a crucial role in perceiving the 3D world, offering valuable insights for various applications. Two main technologies dominate this field: stereo depth cameras and structured light depth cameras.

Stereo cameras, inspired by human binocular vision, employ two lenses capturing slightly offset images. Sophisticated algorithms analyze these images and detect specified geometric features within them. They calculate depth based on the parallax shift between corresponding features. This process is known as triangulation. Stereo cameras excel in good lighting conditions and provide rich texture information, but accuracy can be sensitive to image quality and object texture.

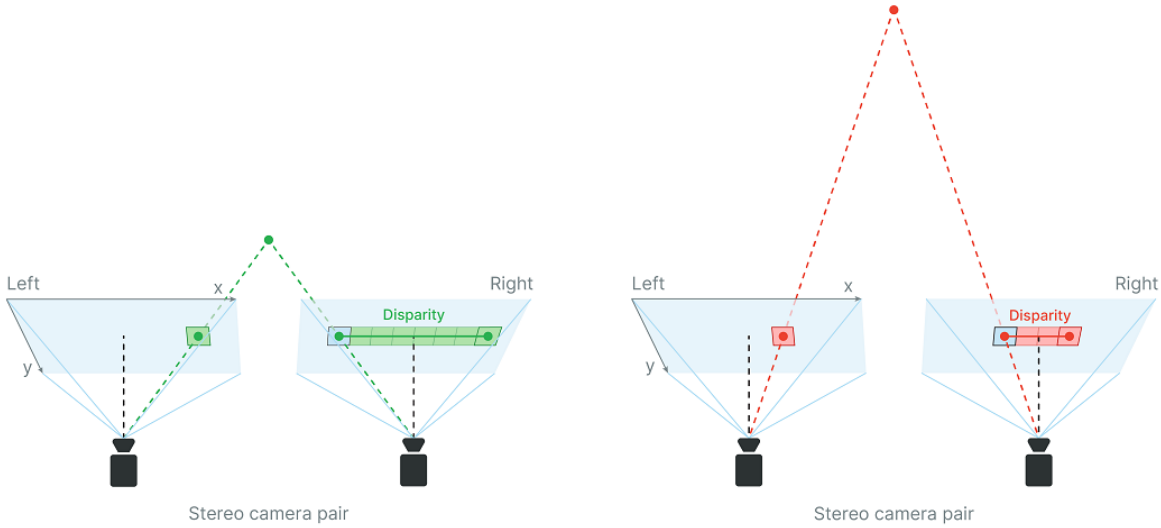


Figure 5-3: Stereo Depth Estimation. Objects further away from the stereo camera pair produce larger disparity [95].

Structured light depth cameras project a known pattern (e.g., infrared dots) onto the scene. The camera then analyzes the deformation of the pattern on objects to calculate depth. This method shines in low-light situations and works reliably on featureless surfaces but struggles with specular reflections and suffers lower overall accuracy compared to high-end stereo depth cameras or LiDAR sensors. Another drawback of structured light cameras is that, contrary to stereo cameras, there are limits to how many of them you can use in a particular space – the cameras might interfere with each other by projecting light into one another's field of view.

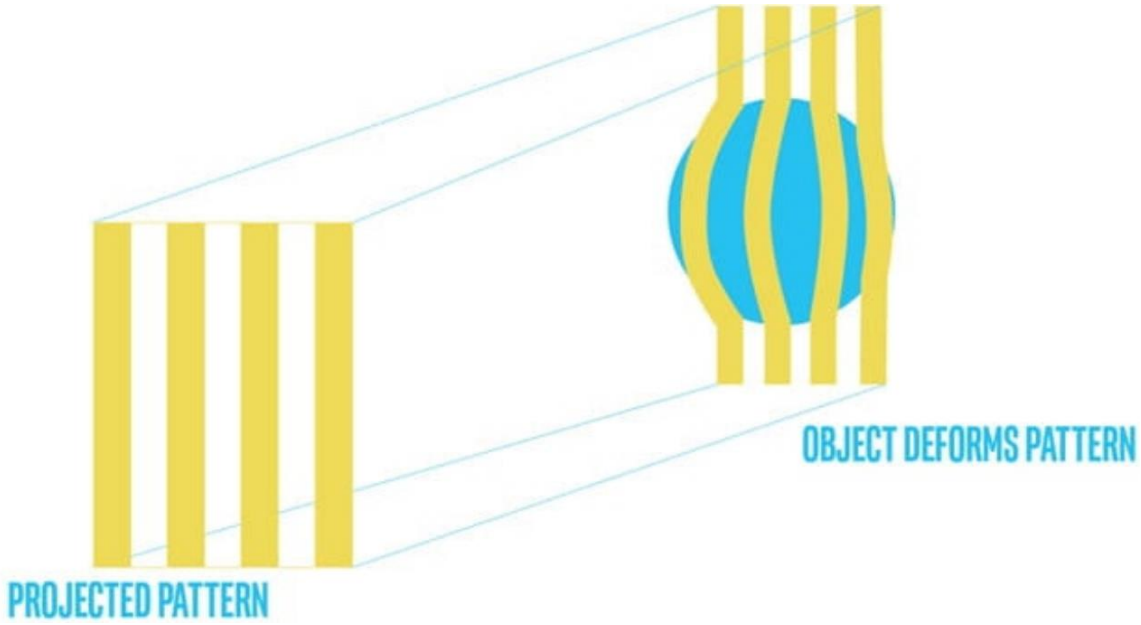


Figure 5-4: Structured Light projected on a sphere [96].

5.1.3 Photogrammetry

Photogrammetry is a well-established 3D reconstruction technique [97]. It utilizes sets of overlapping images to reconstruct 3D models. This process also hinges on the geometric principle of triangulation, where corresponding points in multiple images are identified and used to calculate the 3D position of those points. Unlike stereo depth estimation, however, where the baseline distance between the two camera lenses is known, in photogrammetry a virtual baseline distance regarding two consecutive images is not known, but it is rather estimated based on optical flow computer vision techniques.

Photogrammetry is a broader term encompassing the entire process of creating 3D models from images. It includes not only the geometric reconstruction of the scene but also calibration of the cameras, texture mapping, and other post-processing steps. A term closely related to photogrammetry is Structure from Motion (SfM). SfM focuses specifically on the geometric reconstruction aspect. It relates to the algorithms that receive a set of images as input and automatically estimate the 3D structure of the scene and the camera poses (positions and orientations) without requiring any prior knowledge about the cameras or the scene geometry.

5.1.4 Comparison and Decisions

Each 3D sensing technology boasts unique strengths and weaknesses tailored to specific applications. LiDARS are so far unrivaled in accuracy and range, and they offer precise 3D representation, even over long distances. However, its high cost, high energy demands, as well as the inability to apply texture or color to the 3D models it produces, create significant hurdles for resource-constrained projects that require photo-realistic reconstruction.

Depth Cameras offer a balance between affordability and performance and provide real-time depth information at a significantly lower cost than LiDAR. While accuracy and range might be lower, they excel in compactness and weight, making them ideal for mobile platforms like legged or wheeled robots. They also produce colored point clouds and capture RGB images that can be used to add realism to the reconstructed depth map. Nevertheless,

their performance can be affected by lighting conditions, target surface reflectivity and lack of visual features in the environment.

Photogrammetry leverages conventional cameras and boasts exceptional texture and color capture, enriching 3D models beyond just depth. Additionally, its scalability allows for capturing objects of varying sizes. However, reconstruction accuracy can be lower, and computational demands are significant. Additionally, good lighting and texture variations are crucial for accurate results. Poor lighting and texture conditions can lead to both coloring errors and errors in the reconstructed geometry.

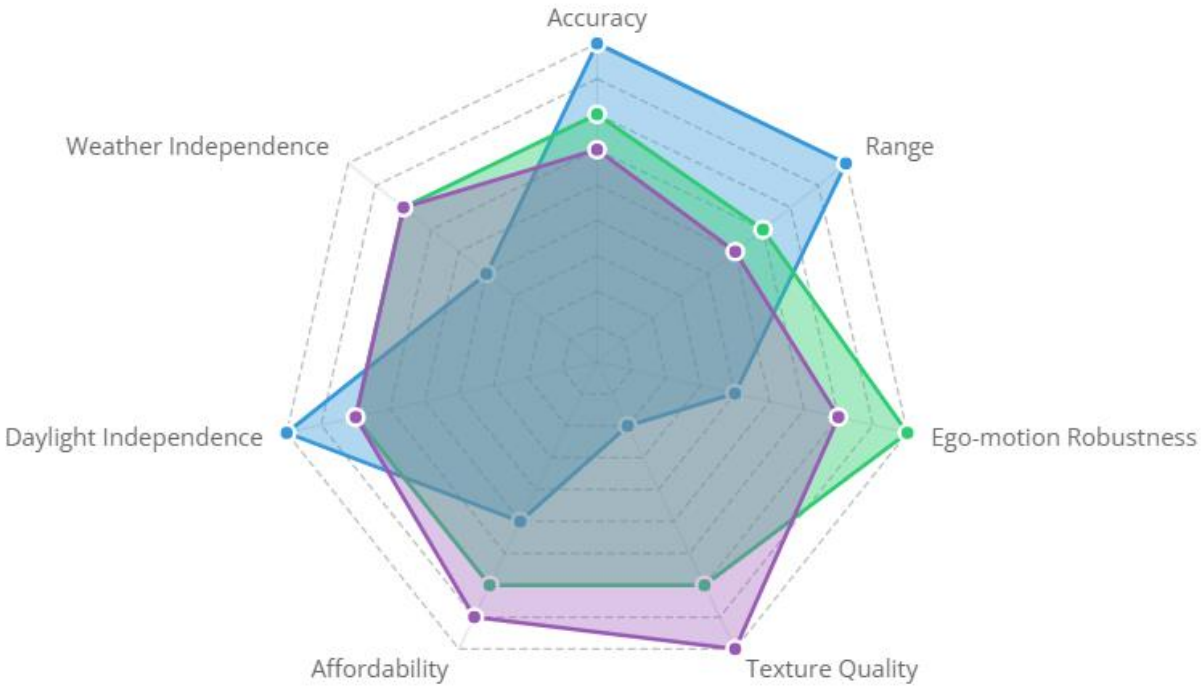


Figure 5-5: Radar Graph illustrating each sensor's strengths and weaknesses.

For our vineyard inspection robot, depth cameras emerged as the optimal choice. Cost effectiveness was an important factor. Depth cameras are more affordable than LiDAR sensors, allowing us to place more sensors on the robot and achieve greater coverage for the same budget. In addition, color information is crucial for analyzing vine and grape health, and depth cameras provide sufficient detail for this purpose. Also, as the robot operates within proximity to the vines, the limited range of depth cameras didn't pose a significant challenge. Lastly, real-time processing requirements, as well as varying light conditions and robotic ego-motion would significantly affect photogrammetry accuracy.

As the project evolves, future iterations may explore integrating other technologies like LiDAR if the need for higher precision 3D models arises. However, for the current stage of vineyard inspection, depth cameras offer a powerful and cost-effective solution.

5.2 Processing Unit

The selection of an appropriate computing platform to host perception systems for agricultural robots presents a critical challenge, as the optimal choice hinges on a delicate balance between various factors. Five distinct categories of platforms have been evaluated for the proposed system: microcontrollers, single-board computers (SBCs), mini computers, and workstations. Each platform has been assessed based on a comprehensive set of

criteria crucial for mobile robot applications: performance, interface, power consumption, weight, and price. This analysis has drawn upon established benchmarks, technical specifications, and relevant research findings to ensure an objective and data-driven evaluation.

5.2.1 Performance Metrics

Evaluating the performance of a computing platform is not a simple task. Computers perform differently across different applications, even under the perception umbrella. It is, however, essential to define specific metrics for performance to employ a scientific lens and culminate in a well-reasoned recommendation for the most suitable solution. In the context of this analysis, the following quantities have been utilized to measure performance:

- CUDA (Compute Unified Device Architecture) Cores: These specialized cores, found in NVIDIA GPUs, excel at parallel processing tasks, making them well-suited for deep learning algorithms employed in perception systems. Their efficiency in handling multiple calculations simultaneously translates to faster processing and real-time performance. CUDA Cores are the Nvidia GPU equivalent of CPU cores [98].
- Tensor Cores: These newer cores, also found in NVIDIA GPUs, specifically enhance the speed of AI training by enabling mixed-precision calculations. This means they can handle lower-precision data formats (e.g., FP16) alongside high-precision data while maintaining accuracy, significantly accelerating the training process for your perception system's neural networks [99].
- FLOPs/TOPS: These metrics directly quantify the processing power of a platform. FLOPs (Floating-point Operations Per Second) measure the number of floating-point calculations, while TOPS (Total Operations Per Second) encompass a wider range including integers and other data types. While FLOPs were traditionally used, the advent of Tensor Cores has shifted the focus to TOPS, as mixed-precision training often benefits from lower-precision operations [100], [101].
- VRAM (Video RAM): This specialized memory holds graphics data crucial for real-time graphics processing applications. In the context of a perception system, it stores information like depth maps, shadow maps, texture maps and other intermediate results generated by the processing units. Having sufficient VRAM ensures smooth and efficient processing of visual data.

  Beyond the Numbers:

5.2.2 MicroControllers

While microcontrollers boast undeniable advantages in terms of power consumption, weight, and price, their suitability for a mobile robot's perception system demands careful evaluation. This subchapter dives into the key characteristics of microcontrollers, highlighting their strengths and limitations in the context of perception tasks.

- **Performance:** Microcontrollers are champions of efficiency, specializing in dedicated, real-time tasks with minimal computational overhead. However, this very strength presents a significant hurdle for perception systems. They usually lack dedicated floating-point math units which renders them incapable of handling the complex computations required for algorithms like convolutional neural networks or spatial mapping. The resulting performance limitations can make them unsuitable for the demanding requirements of perception systems in mobile robots.

- **Memory:** Microcontrollers prioritize compact footprints, resulting in severely limited memory resources. Typically offering only a few megabytes of RAM, they fall far short of the memory requirements for processing visual data and constructing spatial maps. This constraint effectively bars them from consideration for perception applications.
- **Interface:** The miniature size of microcontrollers comes at the cost of standard interfaces. USB ports, often used for camera and LiDAR connections, are scarce, requiring custom solutions using serial communication pins. Additionally, their programming often demands specialized tools and knowledge, deviating from common operating systems used in perception systems.
- **Power Consumption:** Microcontrollers excel in minimizing power draw, consuming mere milliwatts. This makes them ideal for applications where battery life is paramount. However, the computational platform of a mobile robot is often one the least power consuming modules on the robot accounting for only a small portion of its total power usage.
- **Weight:** Microcontrollers reign supreme in the weight category, typically weighing only a few grams. This minimal footprint makes them attractive for robots prioritizing agility and maneuverability, like UAVs or small rovers.
- **Price:** Microcontrollers boast affordability, with most models costing under $100. This price advantage makes them attractive for cost-sensitive projects. However, the trade-off in performance and capabilities must be carefully considered before deciding.

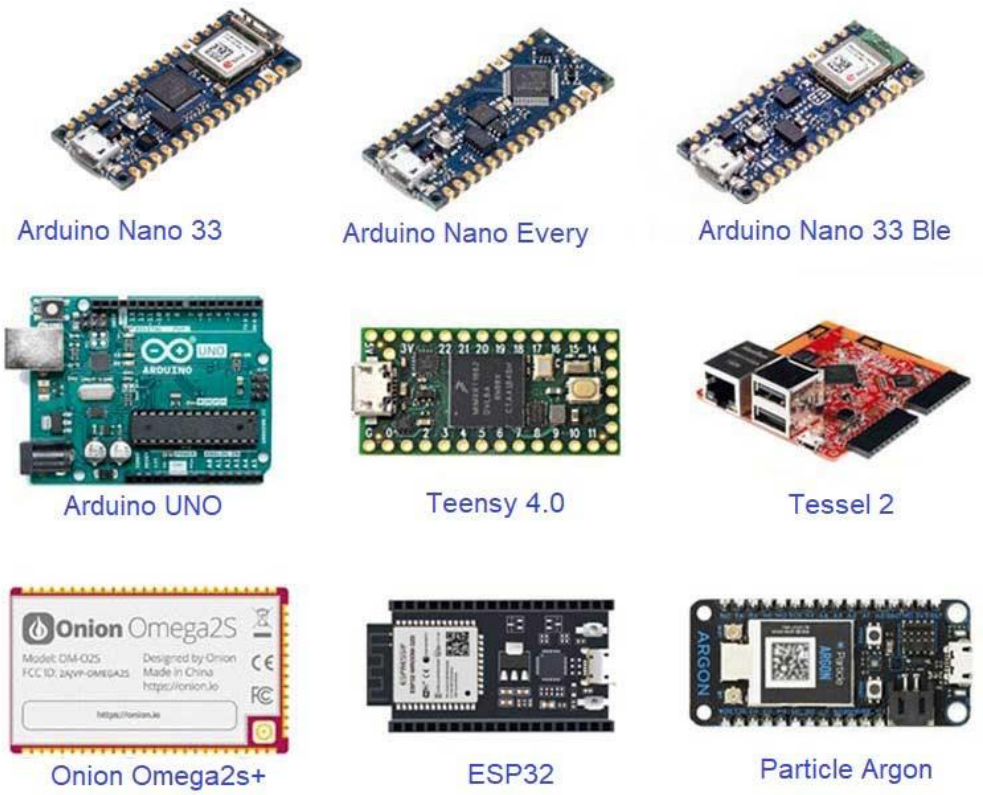


Figure 5-6: Some of the most popular Microcontrollers.

5.2.3 Single-Board Computers (SBCs)

Single-Board Computers (SBCs) offer an intriguing proposition for a mobile robot's perception system, balancing compact size with significant performance gains over microcontrollers. This subchapter delves into their key characteristics.

- **Performance:** The advantage of SBCs lies in their diversity. Performance varies greatly, with some SBCs exceeding microcontrollers by orders of magnitude and others approaching the capabilities of mini-computers. Options capable of processing several TFLOPs (Trillion Floating-point Operations Per Second) are widely available, while high-end SBCs harness Tensor Cores to reach impressive 275 TOPS (Total Operations Per Second) [102].
- **Interface:** Despite their compact size, SBCs are surprisingly well-equipped. They boast multiple USB ports, Ethernet connections, and often HDMI outputs, enabling compatibility with diverse cameras, LiDAR sensors, and displays. Furthermore, their support for common operating systems like Linux and Windows allows for leveraging existing software packages or developing custom solutions with relative ease. While some hardware-specific tweaks might be necessary depending on the chosen SBC architecture, the level of flexibility they offer far surpasses microcontrollers.
- **Power Consumption:** One of the key strengths of SBCs is their power efficiency. Even the most powerful options typically consume only tens of watts, significantly less than mini-computers or workstations. This translates to longer battery life for the agricultural robot, a crucial factor for extended operation. However, higher performance often comes at the cost of slightly increased power draw, so specific workloads must be carefully considered before selecting.
- **Weight:** While not as light as microcontrollers, SBCs remain relatively lightweight, typically ranging from 100 grams to slightly over 1 kilogram. This balance between portability and processing power makes them ideal for robots where maneuverability and computational capabilities are equally important, like legged robots or rovers.
- **Price:** Simple SBCs catering to basic tasks can be found for under $100, making them an attractive option for budget-conscious projects. However, as performance and processing power increase, so does the price tag. SBCs capable of handling AI and vision applications typically range from $100 to $3,000, requiring careful consideration of budget and performance requirements.

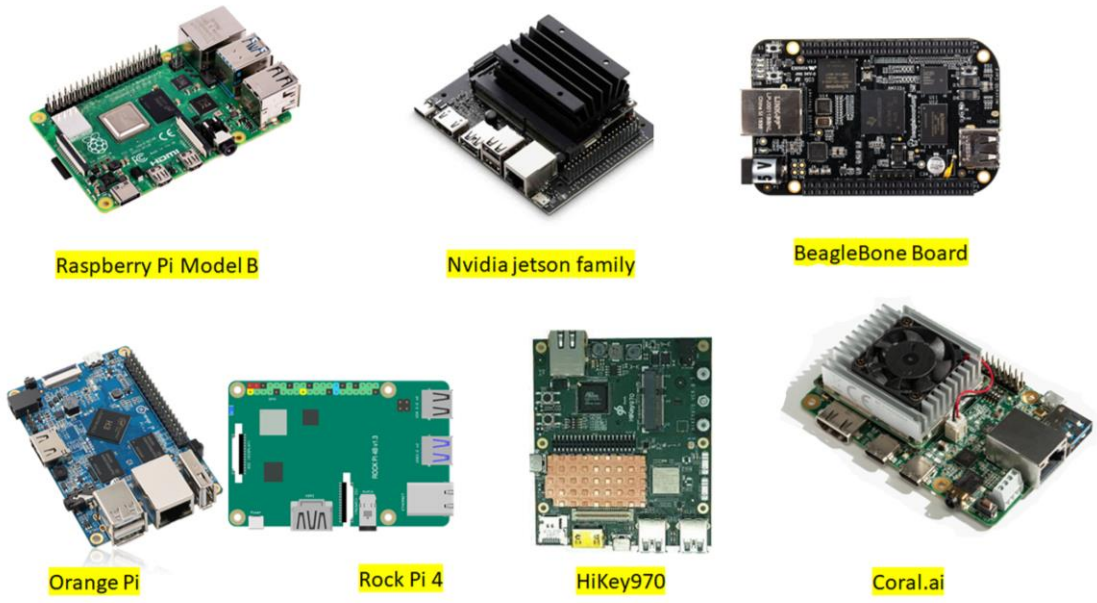


Figure 5-7: Some of the most popular SBCs.

Table 4-1 presents three popular SBCs compared to an Nvidia GeForce RTX 3070 mobile graphics card for intuition purposes, as a standalone graphics card cannot support common operating systems. For that, a motherboard hosting a CPU, RAM and the graphics card must be used.

Table 5-1: Popular SBCs currently in the market.

Specification	**Google Coral Dev Board** (4GB) [103]	**Jetson AGX Xavier** (64GB) [104]	**Jetson AGX Orin** (64 GB) [102]	**Nvidia GeForce RTX 3070** [105]
AI performance	4 TOPS	32 TOPS	275 TOPS	~ 200 TOPS
Cuda Cores	n/a	512	2048	5888
Tensor Cores	n/a (dedicated TPU)	64	64	184
Memory	4 GB RAM	64GB shared	64 GB shared	8GB VRAM
Storage	8 GB eMMC + 64GB expandable	64GBeMMC	64GBeMMC	n/a
Power	5W	10W-30W	15W-60W	115W (Laptop)
OS	Linux Mendel	Ubuntu based	Ubuntu based	any
Weight	250g	630g (with fan)	1580g (with fan)	~1kg (with fan)
Price	~175€	~1600	~2400	~500€

5.2.4 Mini-PCs

Mini PCs offer an alternative path, bridging the gap between compact SBCs and powerful workstations. This subchapter explores their key capabilities and limitations.

- **Performance**: Mini PCs occupy the middle ground in terms of processing power. They harness components typically found in laptops or even desktops, albeit in compact versions often lacking dedicated cooling systems. While this grants them significant performance gains over SBCs, thermal limitations can emerge under sustained

workloads. The thermal management capabilities of any mini-PC must be evaluated carefully to ensure it can deliver consistent performance for the perception system's demands.

- **Interface**: One of the key advantages of mini-PCs lies in their seamless integration with existing infrastructure. They function like traditional PCs, supporting common operating systems and software packages out of the box. This minimizes the need for extensive hardware or software modifications, allowing the usage of established tools and development environments. Additionally, they typically offer a wide range of ports similar to full-sized PCs, ensuring compatibility with diverse sensors and devices.
- **Power Consumption**: Compared to full-sized desktops, mini-PCs boast improved power efficiency. However, they are significantly less power-efficient than SBCs. The presence of dedicated graphics cards, often crucial for AI and vision applications, alone pushes power consumption above 100 watts. Including separate RAM and CPU units can raise total consumption under load to over 200 watts. The robot's battery life and operating environment must be carefully considered when evaluating this trade-off.
- **Weight**: While not bulky, mini-PCs are heavier than SBCs, typically ranging from 1.5 to slightly over 2 kilograms. This increased weight might impose limitations on robots prioritizing maximum agility or operating on weight-sensitive platforms.
- **Price**: Basic mini-PCs, lacking dedicated GPUs, offer attractive affordability, readily available for under 900€. However, for AI and vision applications requiring hardware acceleration, the costs escalate. Integrating an external graphics card or opting for a mini-PC with a built-in laptop GPU pushes the price closer to 2000€.

Table 4-2 presents three popular mini-PCs compared to an Nvidia GeForce RTX 3070 mobile graphics card for intuition purposes.

Table 5-2: Popular Mini-PCs currently in the market

Specification	**MinisForum UM700** (16GB) [106]	**11th Gen Intel NUC + Nvidia 3070 EGPU** (64GB) [107]	**Zotac MAGNUS EN173070C** [108]	**Nvidia GeForce RTX 3070** [105]
AI performance	3 TFLOPS[1]	<150TOPS(usbc bottleneck)	>200 TOPS	~ 200 TOPS
Cuda Cores	n/a	5888	5888	5888
Tensor Cores	n/a	184	184	184
Memory	16 GB RAM (max 32GB)	64GB shared	64 GB/8 GB VRAM	8GB VRAM
Storage	256 GB (expandable)	64GBeMMC	2 SSD slots expandable	n/a

Power	65W	10W-30W + 90W	150W-220W	60W - 115W (Laptop)
OS	Any	Ubuntu based	any	any
Weight	500g	504g + 4kg	1.8kg	~1kg (with fan)
Price	~430€	~1100€ (NUC barebone is 460€)	~2000€	~500*€

5.2.5 Laptops

While laptops share many similarities with mini-PCs in terms of performance, interface, and price, their inherent form factor presents both advantages and disadvantages for mobile robot applications. On the positive side, laptops offer readily available hardware configurations, diverse port options, and familiar operating systems, streamlining integration and development. Additionally, their built-in batteries provide some level of independent operation, potentially beneficial for short-term deployments.

However, their larger size and weight significantly impact a mobile robot's maneuverability and energy efficiency. The constant demand for external power limits their operational range, and heat dissipation becomes a critical concern, often requiring active cooling systems that further increase weight and power consumption. Furthermore, their ruggedness and protection against environmental elements might require additional modifications, adding complexity and cost. Last, their screen accounts for a great part of their total power consumption.

5.2.6 Comparison and Decision

Having analyzed various computing platforms for our mobile robot's perception system, we arrived at a compelling choice: the NVIDIA Jetson AGX Orin, an SBC (Single-Board Computer).

Microcontrollers, while boasting exceptional power efficiency and minimal weight, were ultimately disqualified by their limited performance and memory, rendering them incapable of handling the complex computations required for real-time perception applications. Mini PCs, while offering increased performance and familiar interfaces, presented challenges in terms of power consumption and weight, potentially hindering the robot's agility and operational range. Laptops, despite their versatility, were deemed unsuitable due to their inherent bulk and limited battery life, compromising the robot's portability and independence.

SBCs emerged as the ideal compromise, balancing performance with portability and efficiency. Their diverse range allows for tailored selection based on specific needs, and their compact size minimizes weight concerns. The NVIDIA Jetson AGX Orin, specifically, stood out amongst SBCs due to its exceptional capabilities. Figure 5-8 illustrates a comparative analysis which underscores the Orin's suitability for real-time applications. Orin features both NVIDIA cores and Tensor Cores, delivering exceptional processing power for demanding perception and machine learning tasks. Despite its impressive performance it

maintains moderate power consumption, ensuring longer operation and compatibility with battery-powered robots. The SBC's small form factor and light weight minimize its impact on

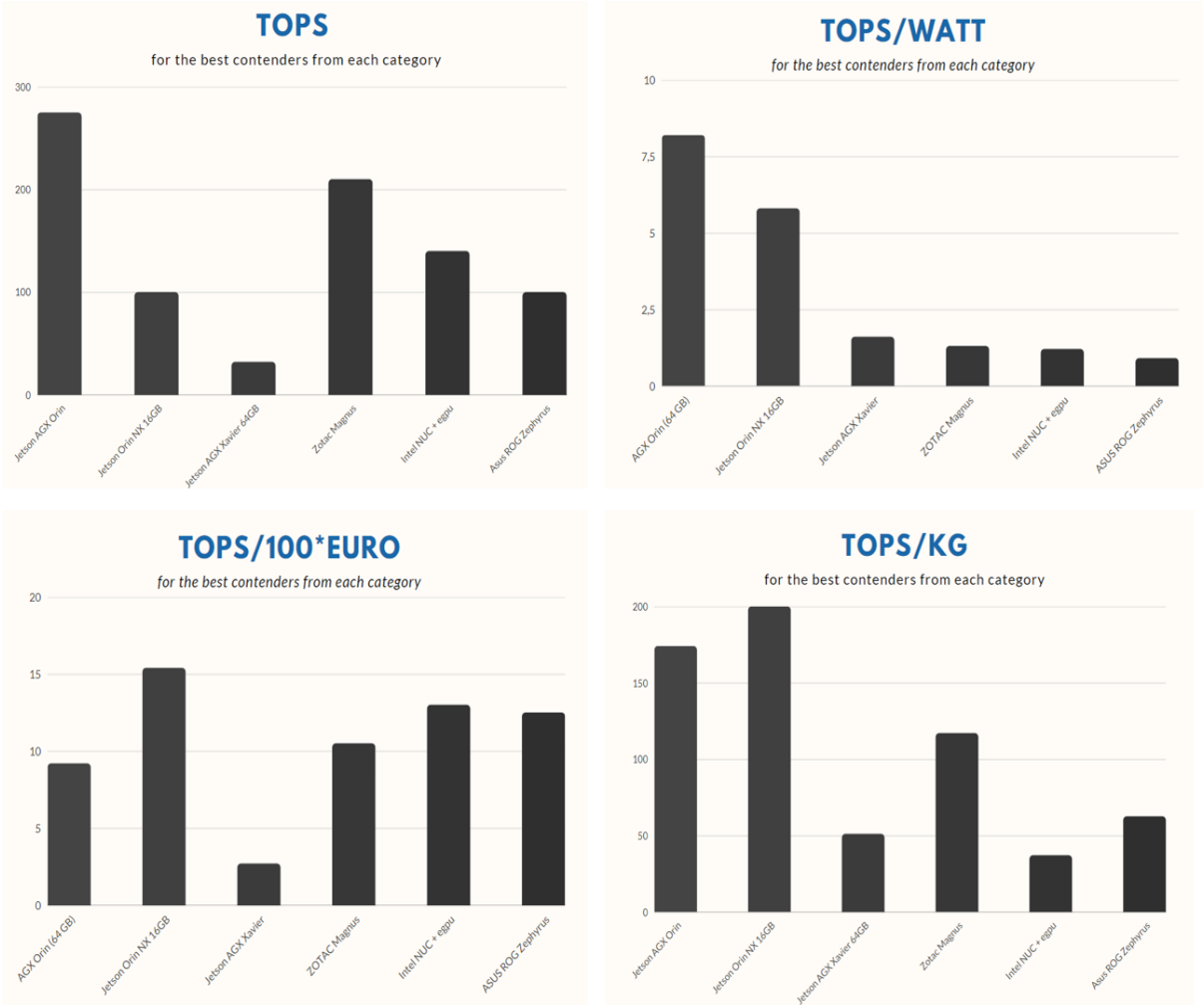


Figure 5-8: Bar graph illustrating the relationship between performance and other features for the top platform of each category.

the robot's mobility and agility. The Orin offers a wide range of ports and connectivity options, enabling seamless integration with various sensors and devices. It can not only host all ZED depth cameras, but it also allows for connectivity with the ZED X mini depth camera, one of Stereolabs latest additions to its plethora of cameras. Additionally, the NVIDIA ecosystem provides extensive development tools and resources, streamlining the implementation and optimization of perception and machine learning algorithms.

By leveraging the NVIDIA Jetson AGX Orin's strengths, any mobile robot can be efficiently empowered with a perception system capable of real-time obstacle detection, navigation, and environment understanding.

5.3 Field of View Visualization and Lidar Resolution Analysis Tool (FoVaLiRa)

The process of designing a perception system for a robot involves deciding not only what sensors are going to be used but also what their position and rotation on the robot will be. The ability to visualize the combined field of view produced by camera and lidar sensors on the robot in simulation assists with such design decisions. While several online tools

visualize field of view (FOV), they lack the detail, realism, and versatility necessary for effectively comparing diverse 3D perception system designs. Consequently, we opted to develop our custom FOV visualization tool within the Unity game engine. This tool facilitates informed design decisions regarding blind spots, coverage, and resolution for both depth cameras and LiDARs, commonly employed for 3D mapping on mobile robots.

5.3.1 The Scene

Unity

The Unity Development Platform was used to create the 3D simulated environment and code the behavior of the sensors in it. Unity was chosen over other simulators for several reasons:

- Unity currently supports over 25 different platforms making it a cross-platform engine. [109]
- As of 2020, Unity-made applications were used by 2 billion monthly active users, with 1.5 million monthly creators [110]. The vast community behind Unity enables developers to ask questions and quickly find solutions to their issues.
- Unity is lately transitioning to Robotics, AI and simulation applications with new and actively supported packages [111].
- Unity provides a visually intuitive and user-friendly development environment, allowing researchers and developers with varying levels of programming expertise to create and interact with simulations. This can be particularly beneficial for future projects in CSL requiring rapid prototyping and iterative design, where quick visualization and modification of the simulated environment are crucial.
- Unity offers a vast library of pre-built 3D assets and environments, encompassing various objects and scenarios relevant to agriculture and mobile robotics. Additionally, its robust physics engine enables realistic simulation of object interactions, force dynamics, and sensor responses within the virtual environment, further enhancing the validity and reliability of the simulation for research purposes.

Simulated Vineyard

Within the Unity platform, we constructed a 3D simulated vineyard scene reflecting typical viticulture practices in Greece. The scene incorporates essential vineyard row parameters, including an inter-plant spacing of approximately 1 meter and a minimum grape height of 0.60 meters. Virtual grape clusters are modeled as capsules, while the vine canopy exhibits a realistic density, leading to diverse visibility conditions. Specifically, a portion of the grapes are partially obscured by leaves, while others are situated in the foreground plane. The vineyard layout comprises two parallel rows, each measuring 4 meters in length, situated on even terrain.

Figure 5-9 illustrates the scene that a user views after downloading, installing, and launching the developed Unity Project. To the left of the scene tab, on the Hierarchy tab, the user can see that the vines are registered as Vine Trees under the Obstacles object category. On the scene tab, within the branches of the vines, the user can observe the blue capsules that represent grape clusters and are registered as targets on the Hierarchy tab, as those are the objects of interest for the robot.

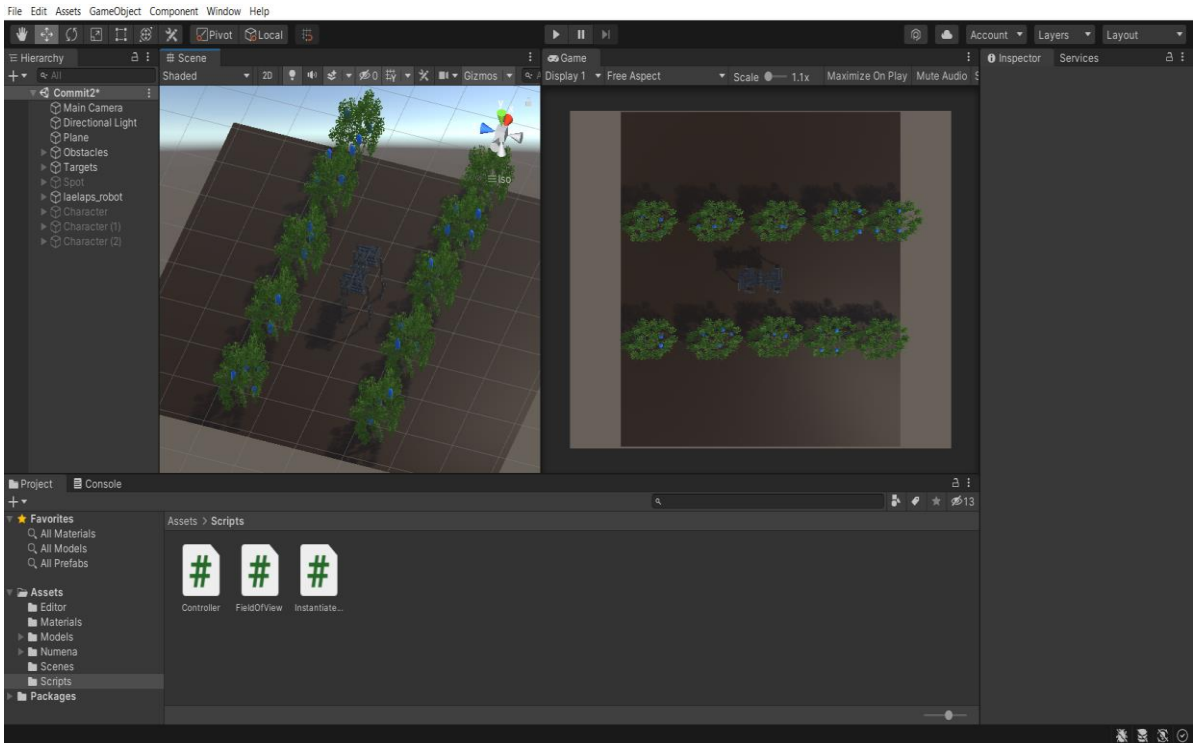


Figure 5-9: The default starting scene in Unity.

In the Hierarchy tab, some object names can be seen in fainted white color. These objects are also present in the scene but they are deactivated. The user can select any of these from the Hierarchy tab. Thus, the Inspector tab on the right side of the window will display information about the selected object. To activate the selected object, the user can check the checkbox near the name of the selected object in the Inspector tab.

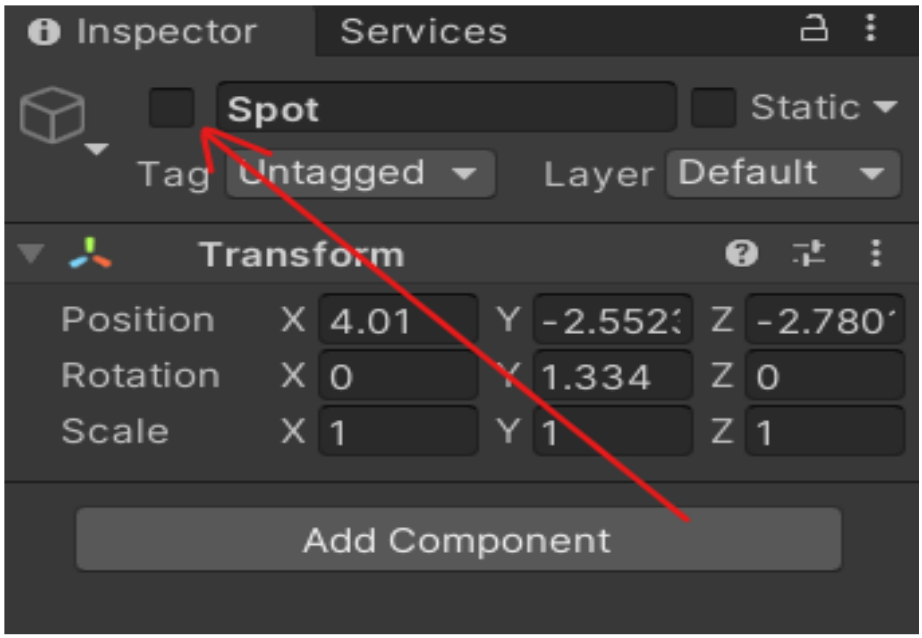
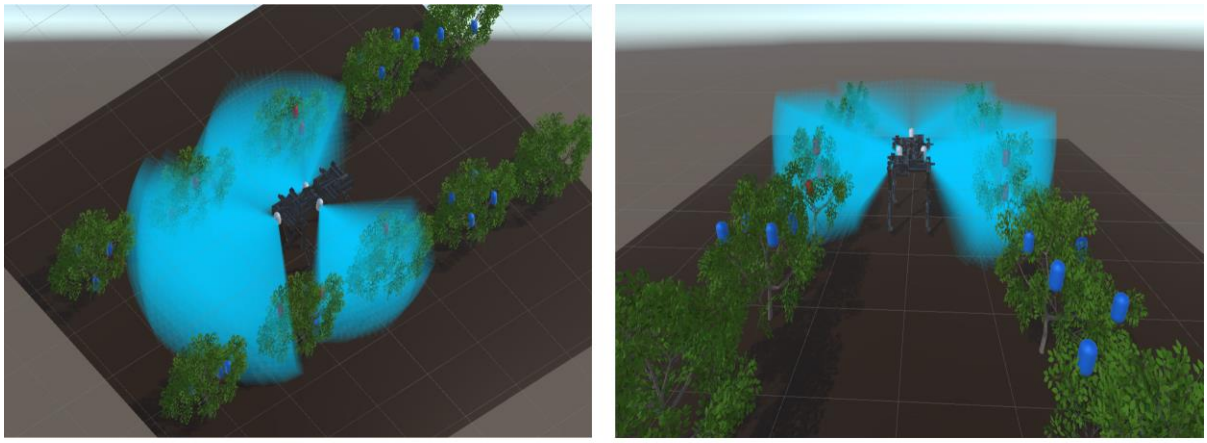


Figure 5-10: Activating and Deactivating an Object

The objects that are deactivated by default are the Spot robot model and three objects labeled as Sensors. The Spot robot model can be used as an alternative to the Laelaps robot model. The Sensors represent lidar (or camera) sensors and are by default placed on

the sides and on the front face of the Lealaps robot model. Activating the three objects labeled as Sensor, Sensor (1) and Sensor (2) and clicking on play to run the simulation - the play and pause buttons are on the middle of the Unity toolbar on the top of the screen - will start the visualization of each sensor's field of view. Figure 4-11 illustrates the running visualization for this configuration.



(a) (b)

Figure 5-11: The scene while running the visualization.

5.3.2 Configuration

The user can configure several parameters to simulate different sensor setups. The list of parameters that can be configured for each sensor is as follows:

position(x,y,z) : The position of the sensor in the scene, relative to the laelaps robot transform. Values are in decimeters.

rotation(x,y,z) : The rotation of the sensor in the scene, relative to its own transform. Values are in degrees.

View Radius: The range of the lidar while scanning horizontally (or max range for camera).

View Angle: The horizontal view angle of the lidar (or camera).

Vertical View Radius: The range of the lidar while scanning vertically (or max range for camera).

Vertical View Angle: The vertical view angle of the lidar (or camera).

Mesh Resolution: Number of rays cast for each horizontal mesh divided by the horizontal view angle in degrees (See section "How it was made" for more intuition).

Horizontal Offset Resolution: Number of horizontal meshes to be cast, divided by the vertical view angle in degrees (See section "How it was made" for more intuition).

Vertical Mesh Resolution: Number of rays cast for each vertical mesh divided by vertical view angle in degrees (See section "How it was made" for more intuition).

Vertical Offset Resolution: Number of vertical meshes to be cast, divided by the horizontal view angle in degrees. (See section "How it was made" for more intuition).

Edge Resolve Iterations: Binary search iterations to resolve the "edge problem" (See section "How it was made" for more intuition).

Edge Distance Threshold: Minimum distance at which an obstacle would be considered to be too far to be accounted for the resolution of the "edge problem" (See section "How it was made" for more intuition).

There are two ways to easily set the position and rotation of the sensors on the scene as well as to configure the sensor parameters: One is through modifying the instantiation.txt

file located in the project folder. The other is using the Unity Editor to directly edit the scene and the sensor parameters.

Configuration using the instantiation file

The "instantiation.txt" file is located inside the project folder. Essentially, the instantiation file represents an array. The elements of the array inside each row are separated by commas as the file is in csv format. The first row contains the headers of each column. Each row added after the first will cause the instantiation of a new sensor on the scene. The new sensor's parameters will be the elements of the new row in correspondence with the headers. As an example, the "typical_instantiation.txt" can be used to instantiate three sensors on the lealaps robot, at predefined positions and rotations. These sensors are also parameterized arbitrarily. To try the instantiation file, the user can copy the contents of "typical_instantiation.txt" inside the "instantiation.txt" file, deactivate the preexisting sensors in the scene and run the simulation. If the "instantiation.txt" file cannot be edited while the Unity Development Platform is open, then the user might need to close it when editing and then open it again to run the simulation.

Configuration using the Unity Editor

In the unity editor, the user can activate, deactivate, or duplicate the preexisting sensors Sensor, Sensor (1), Sensor (2), Sensor (3) and Sensor (4) which can be found in the object Hierarchy. Clicking on a sensor object triggers the inspector to show information about that object. The user can configure all the parameters by editing the field of view script for each sensor directly in the Inspector. In this case, the "instantiation.txt" file should be empty except for the first row (except, of course, if the user desires to both have sensors preexisting in the scene and instantiate some more sensors when starting the simulation). To change the position and rotation of the sensors on the scene, the user can use the move and rotate tools that the Unity Editor offers. To duplicate a sensor, the user can hold the 'alt' key while translating a sensor in the scene. That will leave the old sensor object in place and instantiate a new one.

Configuring the robot's animation

Locating the Lealaps object - which is a child of the LealapsII object - in the object Hierarchy and clicking on it will open the inspector tab. The animator component which controls the animation of the body of the lealaps robot will be active in this tab. To deactivate it, the user can simply click on the checkbox next to its name. The same can be done for the robot's legs: Each leg has two child objects named TopLeg and BottomLeg. Both have animator components which can be deactivated from the Inspector tab. To access the animator controller for the body or any of the legs of the robot, the user can open the inspector of the desired object by clicking on it. Then the user can click on the name box near the controller variable in the animator component. An Animator window will appear. The sequence and the transition between animations can be controlled in this window. For now, the body executes a single translation animation. Each part of each leg executes a translation-rotation animation. The right hind and front left legs of the robot execute an idle animation before executing the translation-rotation animation to simulate a phase difference. To edit an

animation individually, the user can open the animation window (Ctrl+6 hotkey). In there, animations can be created and edited via the Dopesheet or the Curves tab. More information on creating and editing animations in the Unity Development Platform are provided in online documentation [112].

5.3.3 The FoVaLiRa development process

Ray casting

In the domain of computer graphics, ray casting represents a foundational technique for rendering three-dimensional environments onto a two-dimensional display. It simulates light rays originating from a specific viewpoint, typically the camera, and calculates their intersections with objects within the scene [113]. The closest intersection determines the visible portion of an object rendered onto a particular pixel. While computationally efficient and particularly effective in early 3D games, ray casting inherently struggles with intricate lighting effects and smooth object curvature. Modern rendering techniques, such as ray tracing, have emerged to address these imitations: Ray tracing builds upon the principles of ray casting but goes further by simulating the full path of light, including reflections and refractions, leading to superior visual fidelity [114]. However, this enhanced realism comes at the cost of significantly increased processing power [115]. Within the Unity game engine, ray casting remains a valuable tool for tasks like object selection and collision detection due to its efficient nature.

FoVaLiRa employs a ray casting approach, where rays are emitted from the modeled LiDAR or depth camera sensors towards the simulated environment. These rays are organized into a grid-like structure composed of individual meshes, each representing a circular sector within the sensor's FOV. The user-defined grid size directly corresponds to the desired FOV of the LiDAR. Upon encountering a target object (e.g. grape cluster) within the environment, a ray triggers the extraction and return of information regarding the target's location. This information can be subsequently utilized for further analysis or visualization purposes.

Forming a single horizontal mesh

To understand the formation of the grid of meshes, it is simpler to first study how a single horizontal 2D mesh is made: Several rays are cast from the center of the sensor radially inside the horizontal view angles determined by the user. Figure 5-12 illustrates this from a top view.

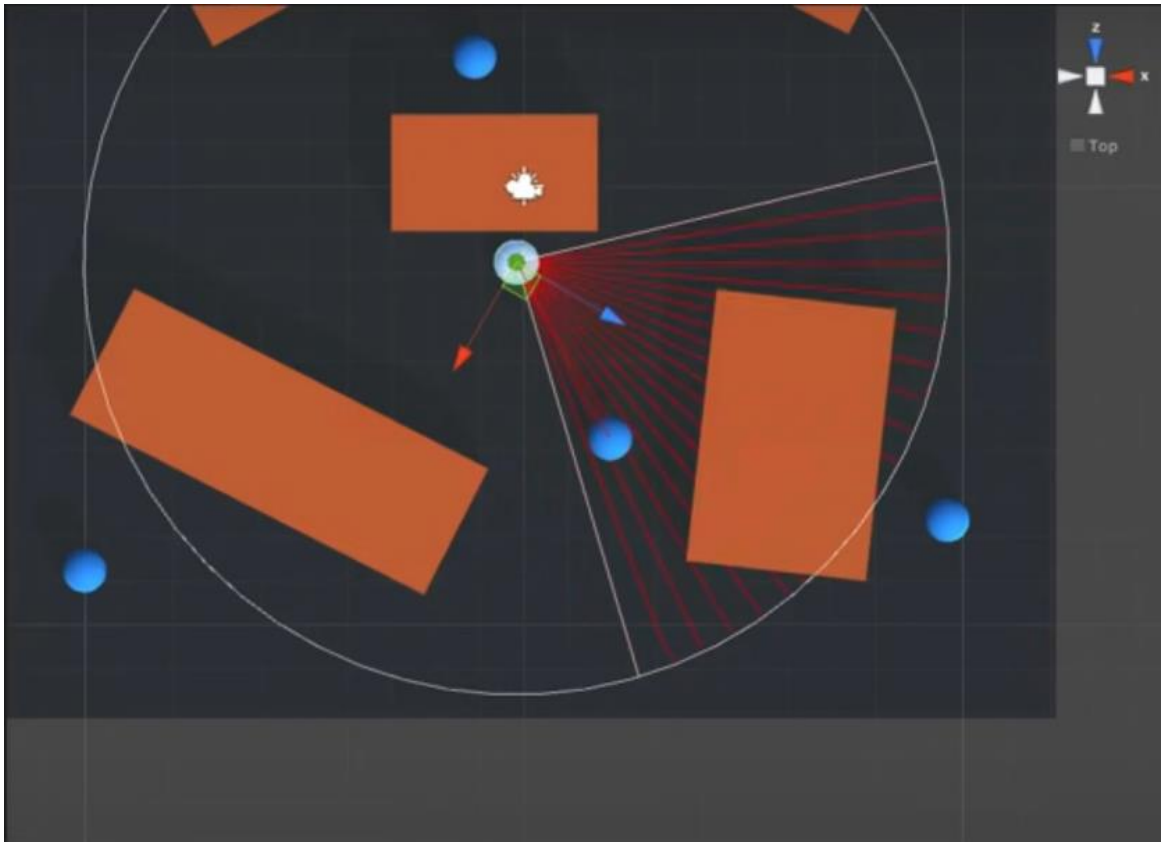


Figure 5-12: Raycasting visualization. The sensor is the white capsule. Rays are in red color. Targets are blue capsules. Obstacles are orange rectangular cuboids.

The number of rays for a given field of view angle is determined by the mesh resolution parameter which can be configured by the user. This parameter simulates a lidar's resolution and corresponds to the modelled sensor's ability to capture detail in the part of the scene it covers. The rays do not pass through obstacles. The starting and ending point of each ray are sequentially saved in a list as 3D points. This list is then used with Unity's Mesh class to form a flat mesh. Figure 5-13 presents the formed colored mesh, cast from a sensor modelled with a capsule. The higher the mesh resolution parameter is set to, the denser the formed mesh.

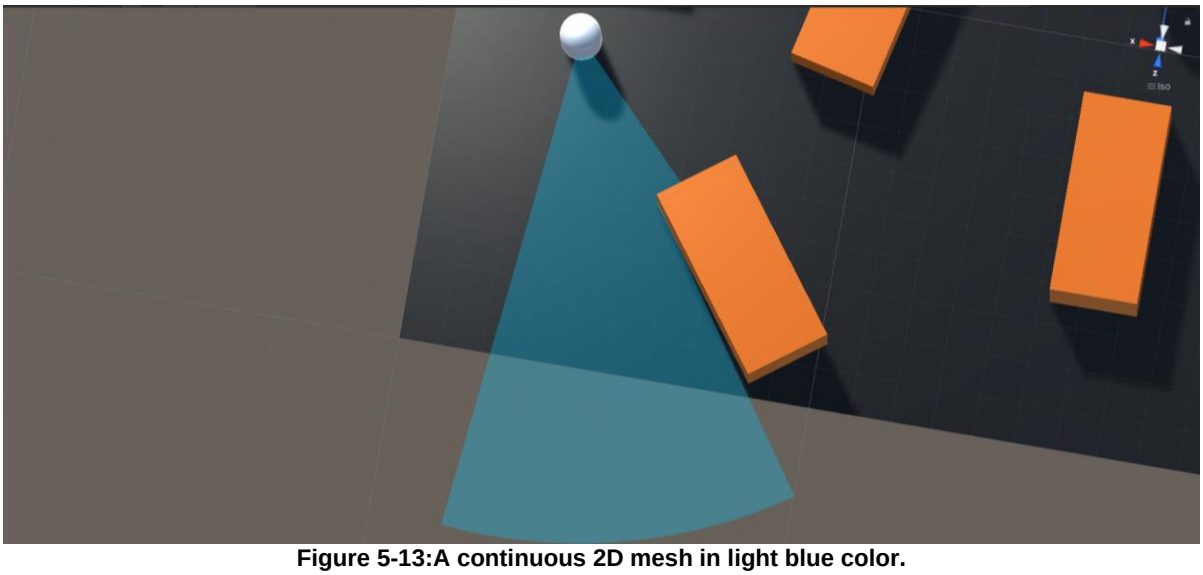


Figure 5-13:A continuous 2D mesh in light blue color.

The edge artifact

The distance that an unobstructed ray travels, is a user-defined parameter called view radius. It models the sensor's range. An edge artifact appears when the edge of an obstacle lies between any two cast rays. Thus, one ray hits an obstacle and the next one misses, ending naturally on the view radius. The two endpoints of the rays that are used to form the colored mesh by the mesh renderer are now forming a triangle (artifact) that passes through the obstacle. Figure 5-14 demonstrates that the resulting, colored mesh does not represent the actual field of view correctly.

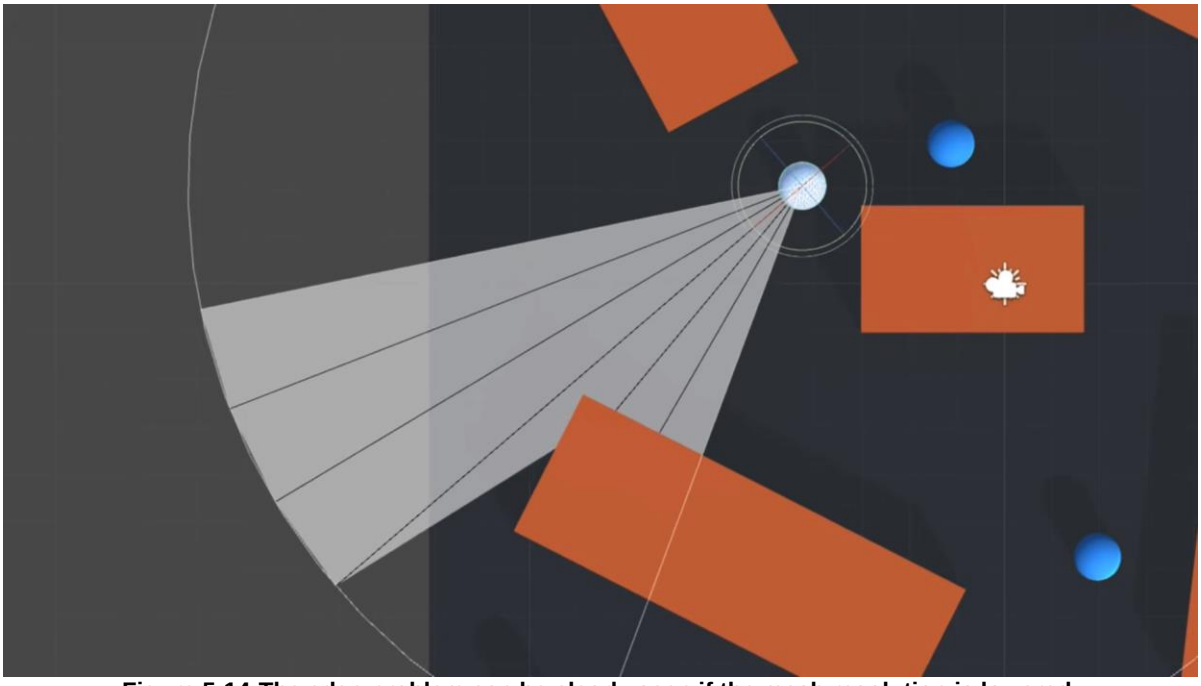


Figure 5-14:The edge problem can be clearly seen if the mesh resolution is lowered.

To solve the problem, a binary search process is performed. Angles in the domain that is formed by the rays that cause the artifact are checked, until an angle, namely a ray direction, that passes closely enough to the edge, is found. Then a ray is cast in that direction. The more the iterations of the performed binary search, the closer the new ray will be at the real edge. The number of iterations can be adjusted by the user by modifying the EdgeResolveIterations parameter. Figure 5-15 illustrates the result.

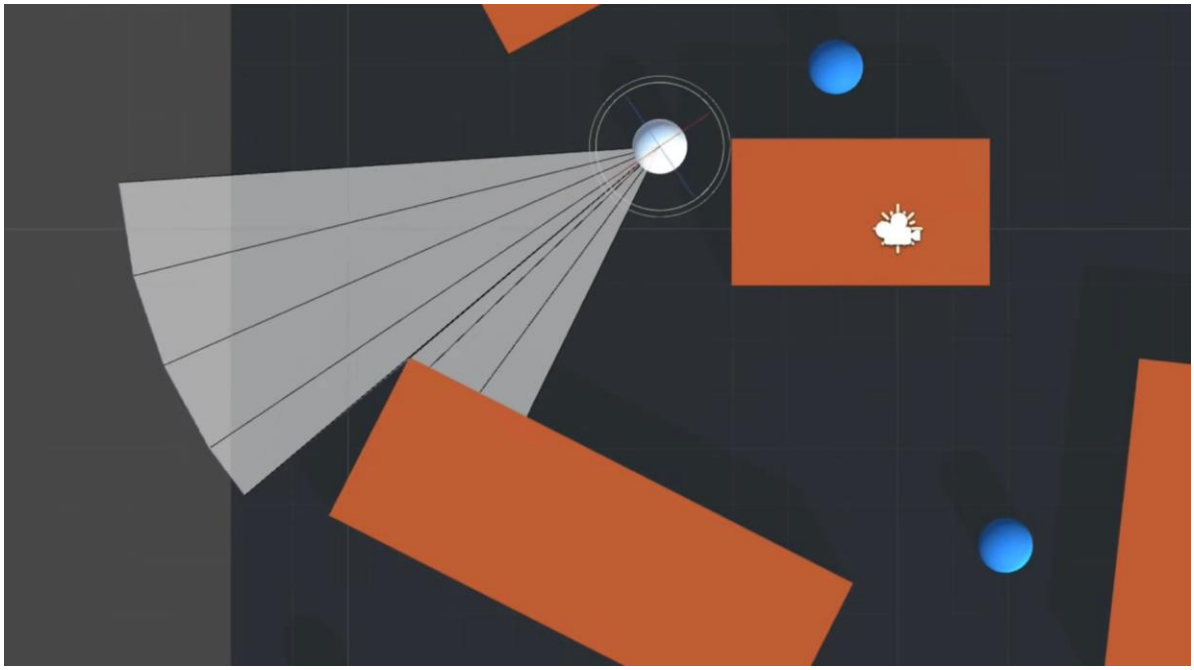


Figure 5-15: Solving the edge problem with 5 binary search iterations.

There is however a problem regarding the condition which triggers the solution of the edge problem. Currently, the condition is that when one ray hits an obstacle and the next one does not hit an obstacle, ending on the view radius, the solution of the edge problem is triggered. The problem is that when both rays hit, but the hit points are a large distance apart, the solution should be triggered, but it does not. Figure 5-16 demonstrates this issue.

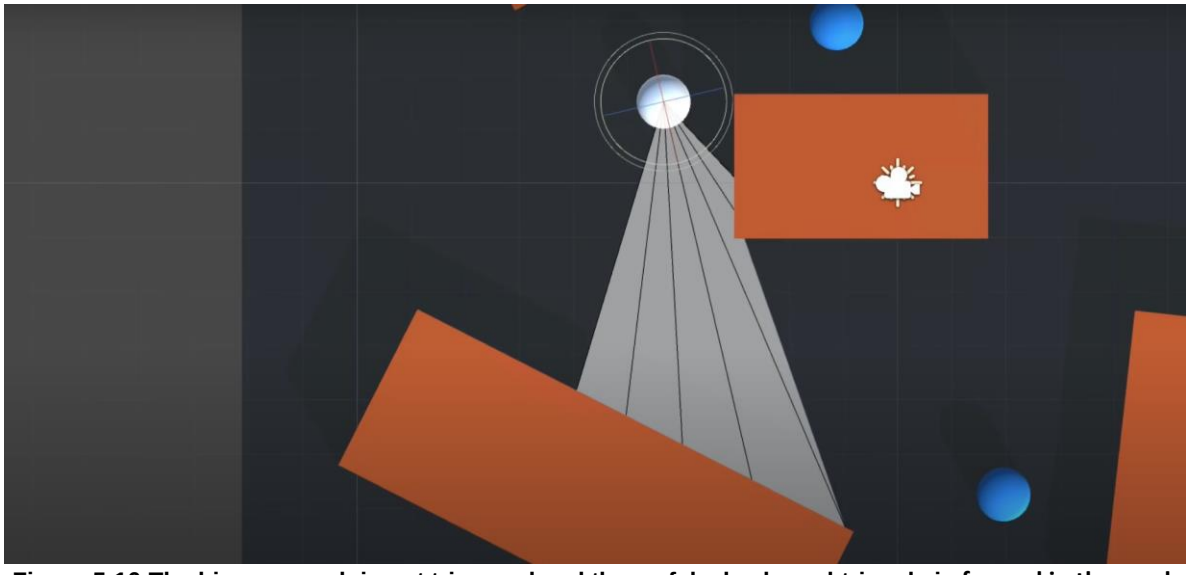


Figure 5-16:The binary search is not triggered and thus a falsely shaped triangle is formed in the mesh

To solve the problem, the condition which triggers the solution of the edge problem was changed so that the process is triggered either when a ray hits and the next one misses ending on the view radius, or two consecutive rays hit and the hit points are a distance apart greater than the EdgeDstThreshold (parameter set by the user). Figure 5-17 shows the result. The produced mesh is smooth and continuous. Algorithm 4-1 contains the pseudo-code for the edge artifact problem solution.

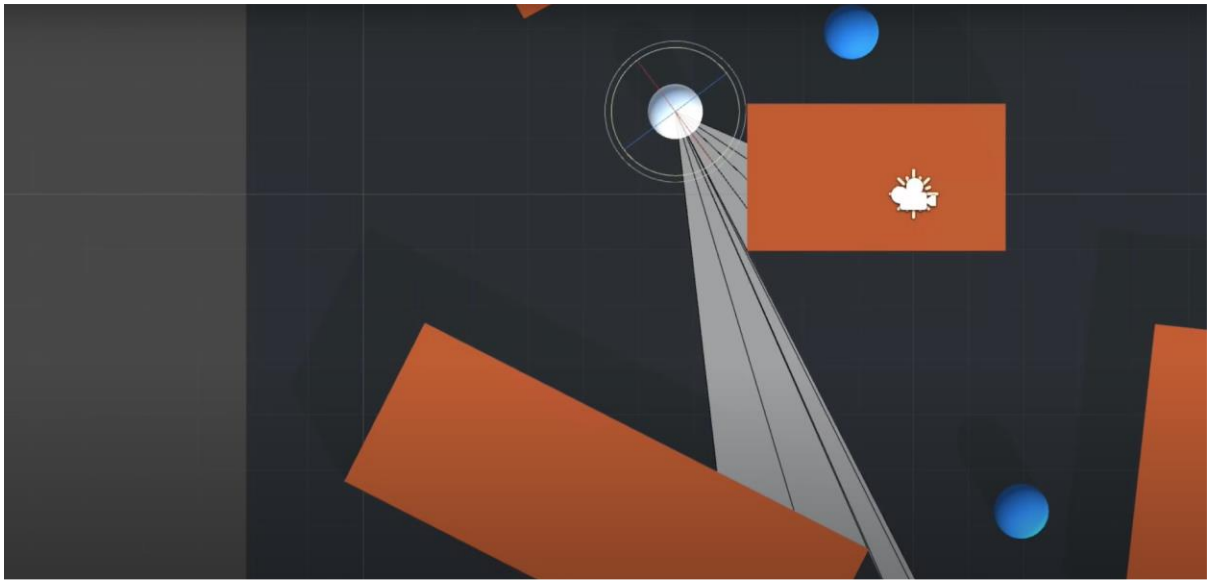


Figure 5-17:Smooth mesh after applying the edge problem solution.

ALGORITHM 4-1 EDGEARTIFACTPROBLEMSOLUTION

Require: (previous ray hits **and** next ray does not hit) **or** (previous ray hits **and** next ray hits **and** next ray endpoint distance > EdgeDstThreshold)

Ensure: There exists a ray that passes close enough to the obstacle edge.

min ← *angleOfRayThatHit*

max ← *angleOfRayThatMissed*

i ← 1

while *i* ≤ *EdgeResolveIterations* **do**

$angle \leftarrow \left(min + \frac{max-min}{2}\right)$

Cast newRay with angle, min < angle < max

angleOfNewRay ← *newRay.angle*

if *newRay* hits **then**

min ← *angleOfNewRay*

else

max ← *angleOfNewRay*

end if

end while

Add newRay.endpoint to the mesh

5.3.4 Forming a 3D Mesh

To form a 3D Mesh, multiple 2D Meshes, rotated around a determined sensor axis were utilized. Figure 5-19 demonstrates how rotating horizontal meshes around the sensor's x-axis by a specific angle (horizontal offset) simulates a lidar's scanning slices created by the laser's horizontal motion. The rotated 2D meshes are saved as a single object, a 3D mesh.

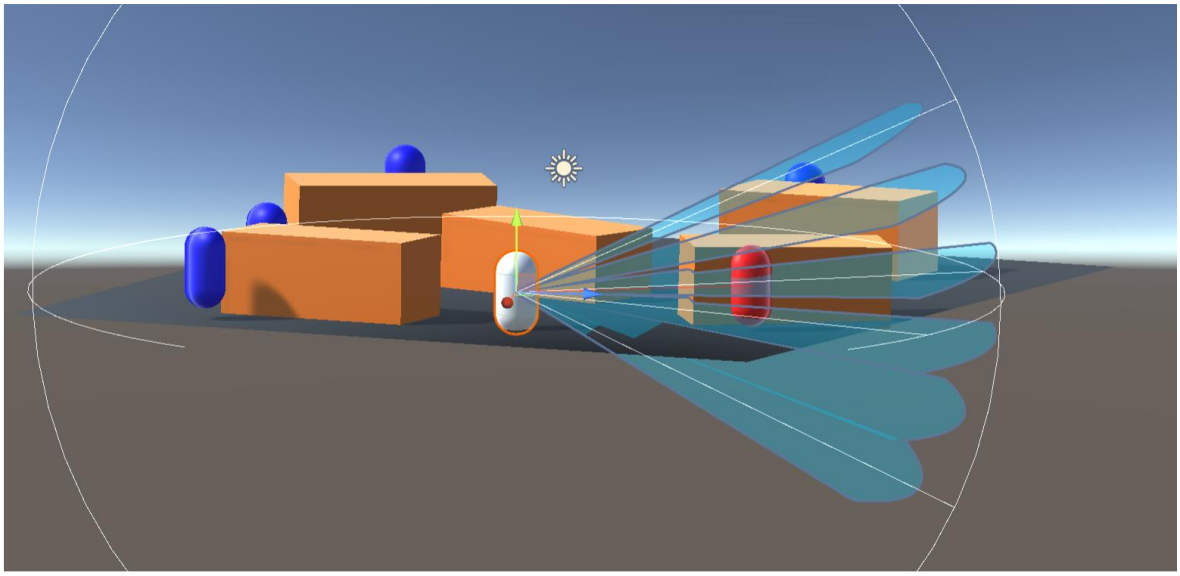
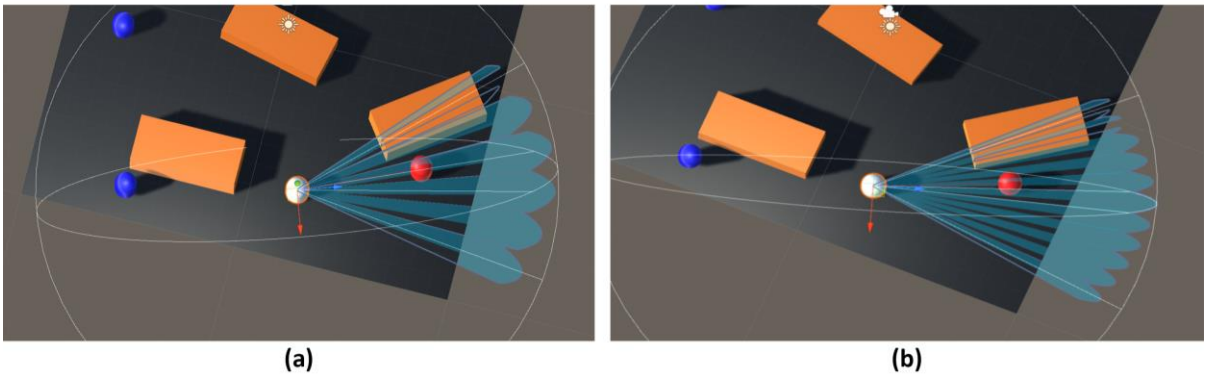


Figure 5-18: Creating a 3D Mesh by rotating multiple 2D meshes around the sensor's x axis.

To make such a 3D mesh denser, the HorizontalOffsetResolution parameter can be increased. Similarly, rotating vertical meshes around the sensor's y-axis by a specific angle (vertical offset) simulates a lidar's scanning slices created by the laser's vertical motion. To make the 3D mesh denser, the VerticalOffsetResolution parameter can be increased. Figure 5-19 demonstrates this.

The rotation of each mesh is achieved by rotating the rays that are cast to create it. The rays need to be rotated relative to the sensor's transformation. For each ray there is a vector3 variable that determines its direction. It is easier to use rotation matrices to rotate each vector around the desired sensor axis [116]. The matrix of a proper rotation R by angle θ around the axis $u = (u_x, u_y, u_z)$, a unit vector with $u_x^2 + u_y^2 + u_z^2 = 1$ is given by:

$$\begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) + u_y\sin\theta \\ u_y u_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) + u_x\sin\theta \\ u_z u_x(1-\cos\theta) - u_y\sin\theta & u_z u_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{bmatrix}$$

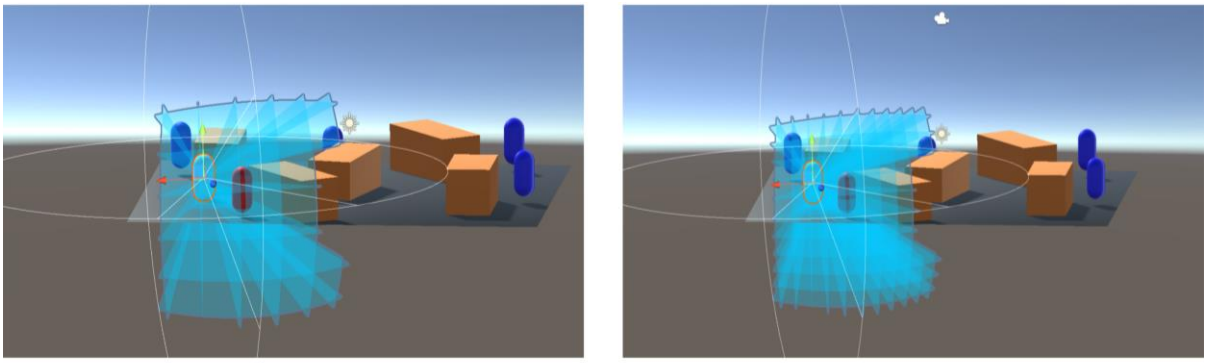


(a) (b)

Figure 5-19: (a) Creating a 3D Mesh by rotating multiple 2D meshes by the vertical y-axis. (b) Increasing the mesh resolution.

To form a denser and uniform 3D mesh, the rotation of both vertical and horizontal 2D meshes is necessary. Rotating horizontal meshes around the sensor's x-axis by a specific angle (horizontal offset) and vertical meshes around the sensor's y-axis by a specific angle (vertical offset) simulates a lidar's scanning slices created by the laser's horizontal and

vertical motion. Figure 5-20a demonstrates that the resulting field of view closely resembles the geometry of a depth camera or lidar field of view. Figure 5-20b Illustrates a denser 3D mesh. To make a uniform 3D mesh denser, the VerticalOffsetResolution parameter, the HorizontalOffsetResolution parameter or both parameters can be increased. This type of field of view visualization is the most accurate of those discussed so far and was extensively used to assist with decisions regarding a robot's perception system.



(a) (b)

Figure 5-20:3D meshes formed by rotating horizontal and vertical meshes.

5.3.5 Detecting Visible Targets

The blue capsules in the scene represent targets. Targets that are in the overall field of view of the robot's perception system are marked with red color instead of blue. The information that a target is inside the overall field of view can be acquired in two ways: The first one is utilizing the rays that are cast to form the 2D meshes. A check can be run for each ray to determine if the ray hits a target or not. If a target is hit, it can then be marked as visible, and it can be added to a list containing all the visible targets. In an editor script, the red color can be assigned to every target in the list with the visible targets. This method simulates a lidar's visibility constraints, attributed to its resolution. This method can test a lidar's both vertical and horizontal resolution. Namely, as Figure 5-21 illustrates, if a target's dimensions are smaller than the dimension of a single cell in the 3D mesh grid formed by the rays, then the target could remain undetected even if its position is in the sensor's effective field of view. This test case simulates a lidar that is unable to map a small object or detail despite the latter being inside the lidar's declared field of view.

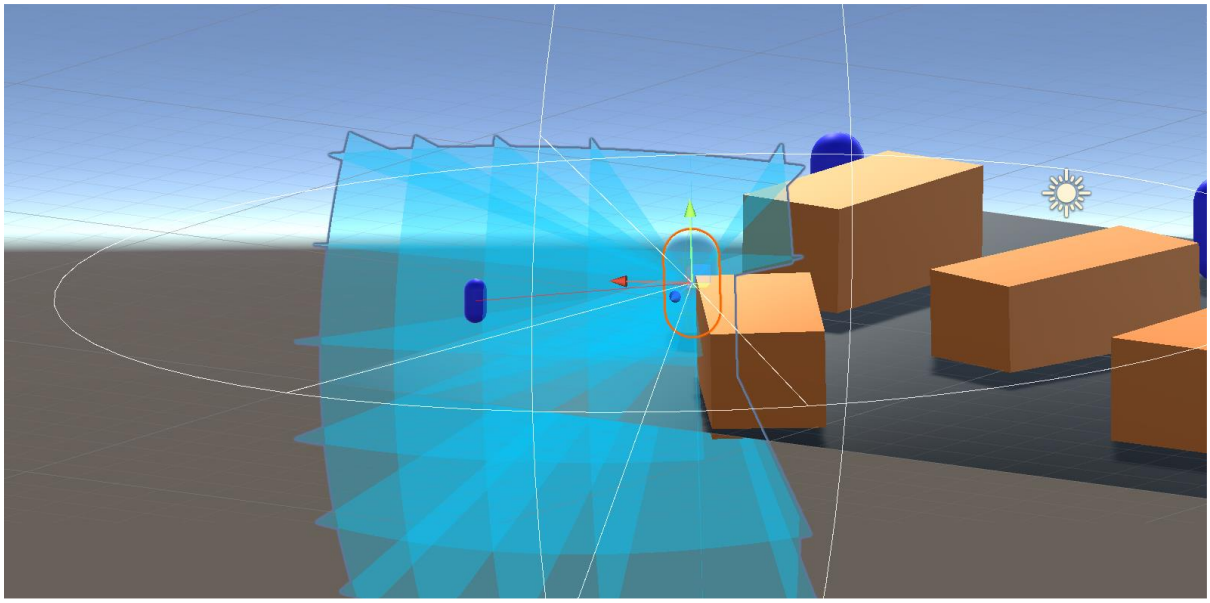


Figure 5-21: Capsule target is positioned in the effective field of view but is too small to be detected by the sensor. Thus, it is not colored red.

The second way to acquire the information that a target is inside the overall field of view, is to simply use math to check if the center of the target lies within the field of view locus. For that to be true, three conditions must be met.

- The distance between the sensor's center and the target's center must be smaller than the view radius.
- The angle α formed by the sensor's z transform vector (front vector) and the projection of the target's relative position vector to the sensor's $\vec{y}$ plane, must be smaller than the horizontalViewAngle (HOVA). Figure 5-22a illustrates this constraint.
- The angle β formed by the sensor's z transform vector (front vector) and the projection of the target's relative position vector to the sensor's $\vec{x}$ plane, must be smaller than the verticalViewAngle (VEVA). Figure 5-22b illustrates this constraint.

This method simulates a depth camera's visibility effectiveness, if it is assumed that the camera has no blind spots within its FOV.

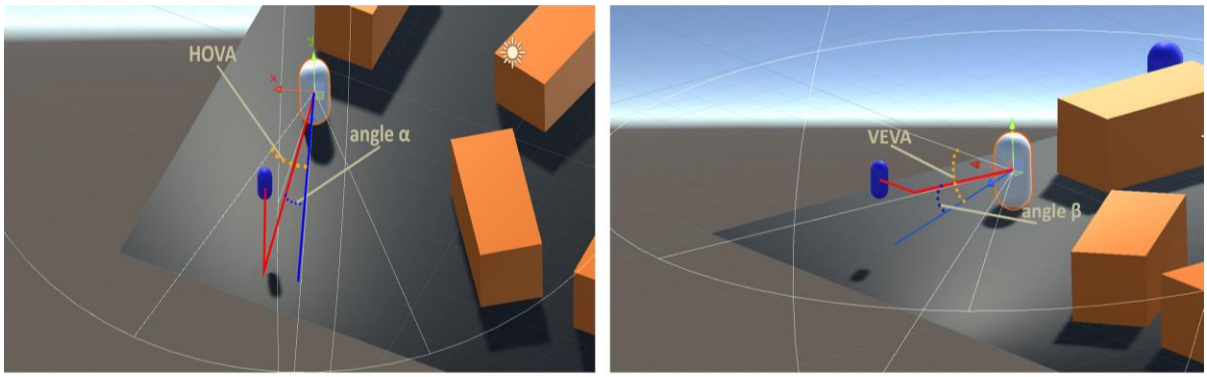


Figure 5-22: Using the field of view locus method to determine if a target is visible.

5.4 The Lealaps Perception System

5.4.1 Perception System Requirements

To determine the ideal position and rotation of the sensors on the lealaps robot, a list of specific requirements was considered. These requirements resulted from the intended use of the robot, namely in outdoor agricultural environments and more specifically in vineyards for grape and vine inspection. Naturally, each of these requirements cannot always be fulfilled and therefore, trade-offs play an essential role when designing the lealaps 3D perception system. In summary, the 3D perception system requirements for the lealaps robot are shown in **Error! Reference source not found.**.

Table 5-3: Summary of 3D perception system requirements.

index	Requirement
1	The 3D perception system should have the best possible observation area and the least possible blind spots.
2	The 3D perception system should be energy efficient.
3	The view radius of the robot should be large enough for the robot to detect obstacles early enough to avoid them.
4	The vision of the robot should not be heavily impaired under adverse weather or lighting conditions.
5	The 3D perception system should be capable of distinguishing visual features and areas of interest in ranges of at least up to two meters.
6	The update rate of the sensors of the perception system should be high enough to allow for fast perception in all directions.
7	The 3D perception system should be capable of mapping whole vines and similar plants which can reach a little more than a meter in height.

5.4.2 Discussing possible sensor configurations

With these requirements in mind, seven different sensor configurations for the laelaps 3D perception system were tested using the Field of view visualization tool in unity.

1. **Laelaps with 2x Zed2 Depth Cameras** [16] **and a Velarray M1600 Lidar** [117]. The FOVALIRA tool in unity was used to check for blind spots and overall visibility constraints for this configuration. Figure 5-23 illustrates that this option offers a large observation area covering the whole front and sides of the robot, with small and not substantial blind spots. However, there is no view from the back of the robot, which could be useful in case the quadruped performs maneuvers which require it to move backwards. An extra Zed2 stereo camera could be added at the back of the robot providing the setup with a near 360◦ field of view. In terms of energy efficiency, this setup is lightweight and efficient. The ZED2 cameras consume a mere 1.9W each [118] while the Velarray M1600 Lidar, being a solid state lidar with no moving parts, needs 15 W [119] at most to be fully operational. In addition, the update rate of solid state lidars is typically higher than that of conventional lidars. Conventional lidars are sometimes slow when it comes to coverage in all directions. The limitations are clear in state-of-the-art robot platforms such as those that participated in the DARPA Robotics Challenge in 2017 (DRC 2017), which had slow update rate as reported by some of the teams [120]. The M1600 can manage a 25Hz refresh rate which is higher than its conventional lidar counterparts. Furthermore, combining lidars and depth cameras allows for versatility even in adverse weather or lighting conditions. Lidars do not require external light sources to operate, and their functionality is not affected by changes in lighting conditions. Cameras on the

other hand are generally not as susceptible to rain, fog, or dust as lidars. They can also map reflective surfaces with more accuracy than lidars. However, given that the field of view of the cameras does not overlap with the field of view of the lidar, if a sensor underperforms then the whole system is affected.

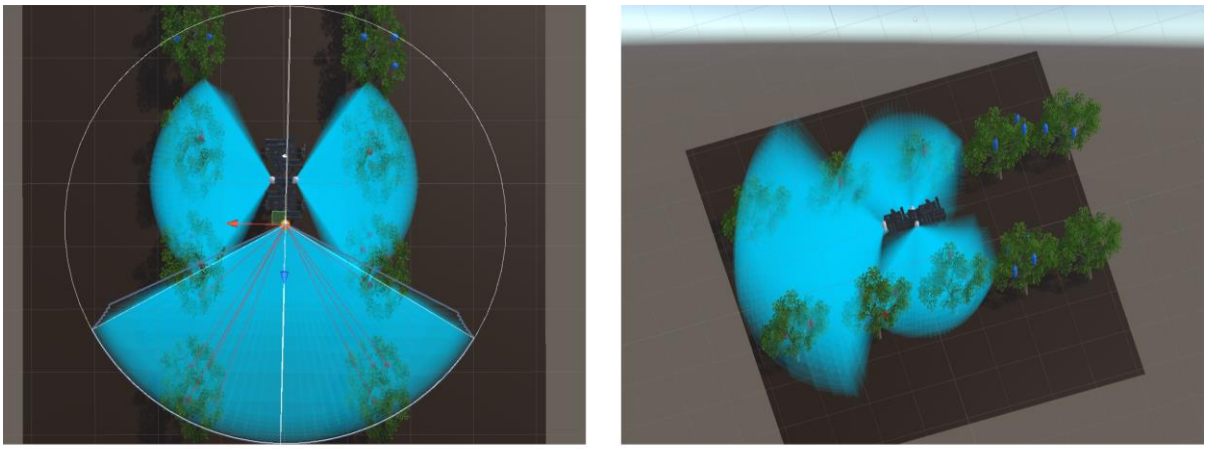


Figure 5-23: Lealaps with 2x Zed2 Cameras and a Velarray M1600 Lidar. (a) Top view. (b) Side view.

2. **Laelaps with 4x Intel Realsense D435 depth Cameras** [121]. This option also offers a large observation area covering the whole front, sides and back of the robot, with small although not negligible blind spots. Figure 5-24 illustrates this configuration. In addition, the D435 depth cameras offer a large vertical field of view angle (58◦), which allows the robot to easily obtain depth images of tall plants, even when standing close to them. In terms of energy efficiency, this setup is lightweight and very efficient. The maximum power draw of the Vision Processor D4 Board, which handles power for both the Vision Processor D4 and the Depth Module of the D435 camera, is rated at 700 mA [122]. With a nominal supply voltage of 5V, each camera consumes a mere 3.5W for a total of 14W for the whole setup. Nevertheless, this setup utilizes only depth cameras. The absence of a lidar means that the field of view radius or the effective range of the perception sensors is smaller, while the point cloud quality at the front side of the robot would be inferior as lidars do produce higher quality and relatively artifact-free depth-clouds.

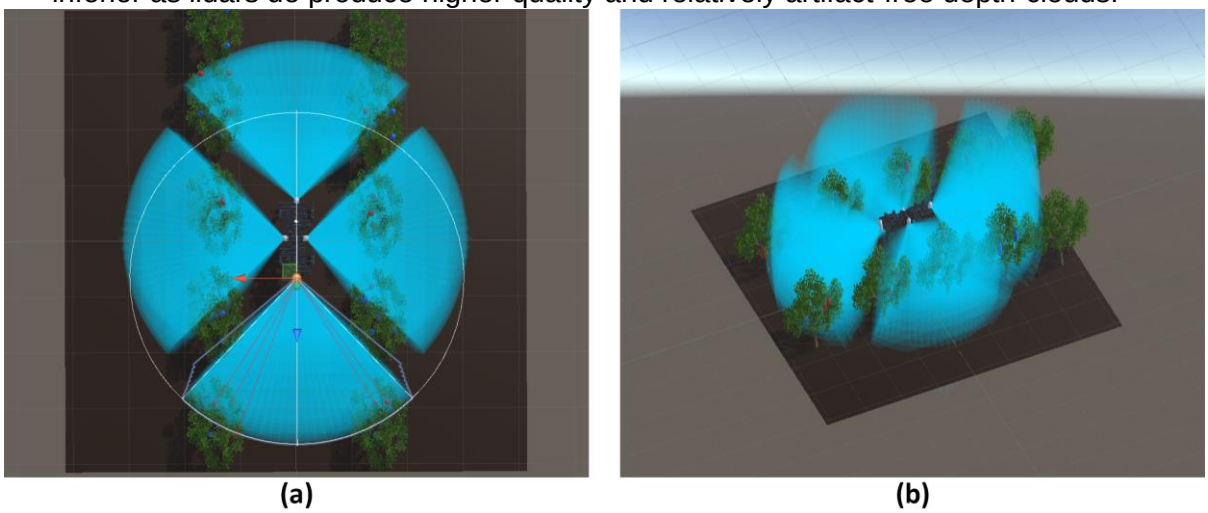
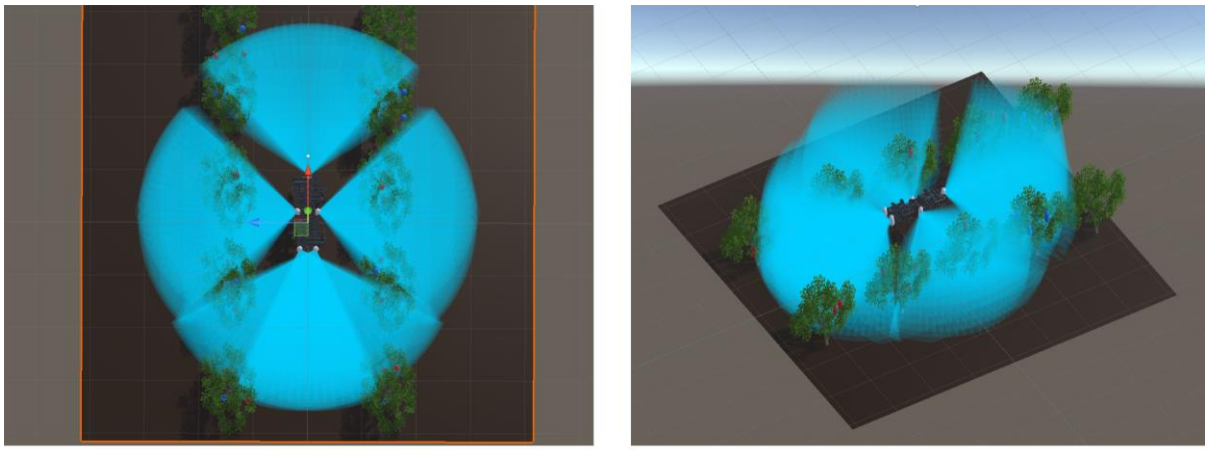


Figure 5-24: Lealaps with 4x Intel Realsense D435 depth Cameras. (a) Top view. (b) Side view.

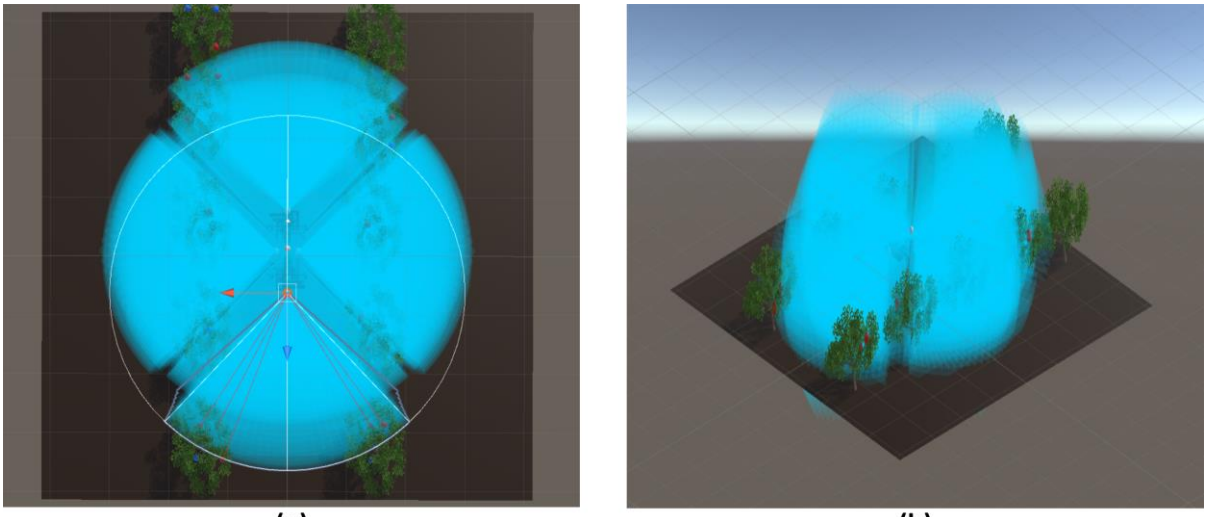
3. **Laelaps with 5x Intel Realsense D435 depth Cameras.** An interesting approach to keeping the simplicity and energy efficiency of the second setup, while raising the depth-cloud quality at the same time is adding a fifth Intel Realsense D435 sensor. Since most obstacles and points/areas of interest will appear in front of the robot, the extra camera should be placed on the front side of the robot, alongside the preexisting front depth camera. In this particular approach, the two front cameras are rotated around the y-axis, which in unity is the vertical axis, at 20◦ inward so that their field of views overlap. Figure 5-25 illustrates this configuration. This creates information redundancy but yields a higher quality depth-cloud overall, since a single underperforming front sensor (for example due to dust or strong direct light on the lenses) will not affect the generated front depth map. This setup, nevertheless, does not increase the effective range of the robot's perception. It does, however, reduce the size of the blind spots on the front side of the overall field of view. Boston Dynamic's spot robot utilizes a similar perception system [123].



(a) (b)

Figure 5-25: Lealaps with 5x Intel Realsense D435 depth Cameras. (a) Top view. (b) Side view

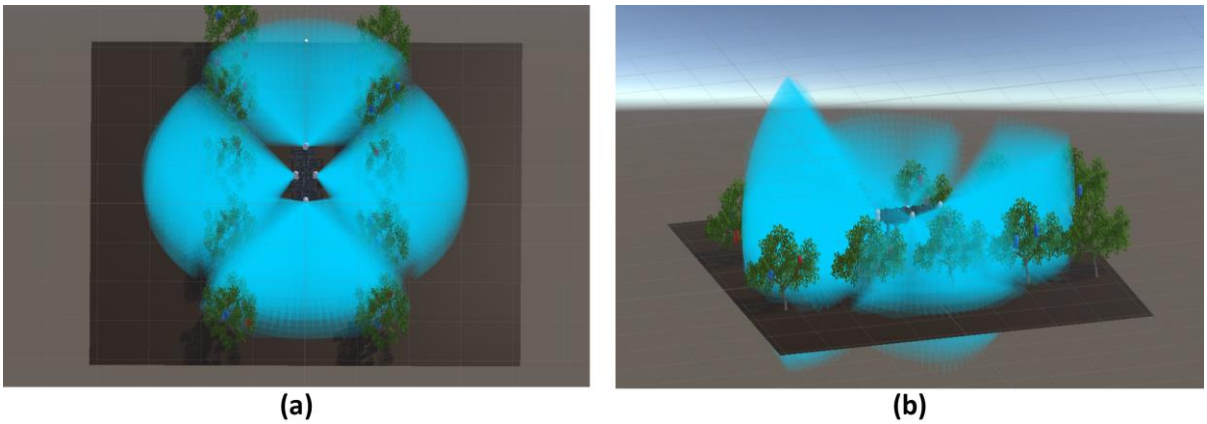
4. **Laelaps with 4xD435 + Velodyne Ultra Puck Surround View Lidar** [124]**.** Informational redundancy can lead to a more robust perception system. This approach utilizes overlapping 3D fields of view to accomplish just that. On the center top of the robot, a Velodyne Ultra Puck Surround View Lidar offers a 360◦ field of view with high resolution. To refine the resulting point cloud, 4xD435 depth cameras, one in every side of the robot, provide additional depth images. Figure 5-26 clearly shows that there are no blind spots with this setup and an underperforming sensor will not significantly affect the produced point cloud. A major drawback of this system is the weight and the limited energy efficiency. The lidar alone is around 1kg heavy. It consumes 10W of power at typical operating conditions [125]. The cameras combined weigh 1kg and consume a total of 14W. Thus, the setup weighs 2kg and consumes 24W of power. This makes this perception system the heaviest and most energy-consuming system discussed so far. In addition, it is not focused on one side of the robot. The field of view expands evenly away from the robot. This is usually not desired, as four legged robots are generally more agile when moving forwards and thus it is a preferable design choice for the perception system to be focused on the front side of the robot. Anybotics Anymal C robot utilizes a similar perception system [126].



(a) (b)

Figure 5-26: Laelaps with 4xD435 + Velodyne Ultra Puck Surround View Lidar. (a) Top view. (b) Side view.

5. **Laelaps with 4xD435 and an extra D435 at the front, rotated around the x-axis.** This configuration aims to improve the robot's perception in the front, while also improving the overall vertical field of view. Robots that will commonly encounter obstacles (e.g. branches) high above their body height or need to gather visual information from tall objects and structures will benefit from a design like this. Figure 5-27vdemonstrates that several -though not significant -blind spots appear when visualizing the overall field of view of this perception system. The small overlap between the two front camera's field of view will yield slightly better point cloud quality in the fused map shaped in the front of the robot. The system is very energy efficient and lightweight since it only utilizes lightweight depth cameras with low consumption (typically 3.5W). Xiaomi's Cyberdog 2 utilizes a similar setup, due to its need to have visual contact with human faces and gestures [127].



(a) (b)

Figure 5-27:Top view of Lealaps with 4x Intel Realsense D435 depth Cameras and an extra D435 at the front, rotated around the x-axis.

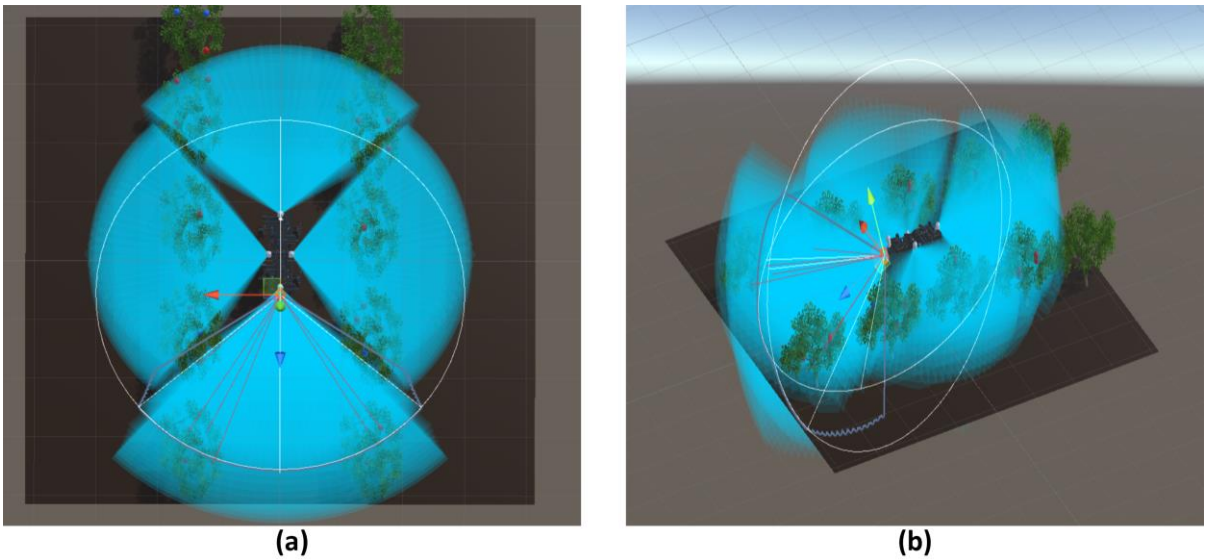
5.4.3 Deciding on a near optimal sensor configuration for Laelaps

The Laelaps robot, being optimally designed using a combination of criteria related to forward speed, would benefit from a perception system that is focused on the field of view in front of the robot. However, in agricultural environments, the robot will encounter difficult terrain, dead-ends and obstacles that cannot be easily avoided. In such conditions, the

quadruped will be required to rotate, turn and even walk backwards. Therefore, the robot should be equipped with a setup that produces a near 360◦ field of view. Considering this, as well as the summary of 3D perception system requirements presented in Table 1, the Laelaps sensor configuration would benefit from 4x ZED2 Stereo Depth cameras and one Velarray M1600 Solid State Lidar. Combining a Lidar with depth cameras will provide versatility and robustness even in adverse weather or lighting conditions. Placing the lidar on the front side of the robot and overlapping its field of view with that of the front depth camera will add to the valuable point cloud quality in front of the robot.

Capitalizing on the increased range of the lidar, the lidar is placed at a subtler downward angle than the front depth camera. Figure 5-28 illustrates the setup. Thus, the camera mainly maps the ground close and in front of the robot, while the lidar captures information about terrain and obstacles further ahead. This system is both lightweight and energy efficient. The lidar consumes at most 15W of power while all the depth cameras combined consume an average of 6W of power. Each depth camera weighs about 125g while the solid state lidar weighs well below 1kg. The lidar offers premium quality mapping at distances from 0.1m to 30m while the depth cameras have depth ranges of up to 20m.

For the initial laboratory experiments involving the robotic rover, a simplified sensor configuration was utilized due to budgetary and accessibility constraints. This temporary setup consisted of three ZED depth cameras: one ZED2 facing forward, one ZED2 facing backward, and one ZED X mini with a wide field of view positioned on the side of the rover. This configuration provided adequate view coverage for the current research phase, which focused on single-row vineyard mapping without complex obstacle detection requirements. However, in future deployments involving the quadruped robot and potentially more challenging environments, the sensor configuration encompassing four ZED2 cameras and the Velarray M1600 LiDAR will be implemented.



(a) (b)

Figure 5-28: Top view of Lealaps with 4x ZED2 depth Cameras and a Velarray M1600 Solid State Lidar at the front, placed at an angle.

6 Simulation Experiments

6.1 Simulated World

The vineyard inspection and 3D reconstruction framework proposed in this work was extensively tested in simulation. To thoroughly evaluate the framework, a comprehensive simulation environment was established using Gazebo [128]. This simulated environment closely mirrored the real-world testing compartment located within the Control Systems Laboratory. This room comprises a synthetic vineyard designed specifically for agricultural robot experimentation. The controlled lighting conditions within the laboratory are replicated in the simulation. The vineyard layout features three equally spaced vine rows, forming 1-meter-wide corridors for robot navigation. The ground texture replicates a mosaic pattern, and the inclusion of three cast iron radiators near the walls further enhances the realism of the simulation. A high-fidelity robotic platform (RP) model was designed and integrated within the simulated environment. This model meticulously replicates the actual RP intended for deployment in subsequent real-world experiments.

Figure 6-1:The simulated world in Gazebo.

An April Tag was placed at the beginning of the second row. Its purpose is to facilitate the loop closure process which takes place as soon as the robot effectively aligns its left camera with the April Tag. April Tags and their significant impact on correcting accumulated odometry errors in the context of the developed framework are thoroughly described in

section 2.1.4 of this thesis. Finally, a two-dimensional map of the simulated world was designed for visualization purposes. The map reflects the real-world dimensions of the development space with high-fidelity. Rviz [129] and foxglove studio [130] are the two software platforms that were utilized to visualize RP's movement through the simulated vineyard, as well as to monitor valuable data.

Figure 6-2: The RP and the April Tag in the Simulated Environment

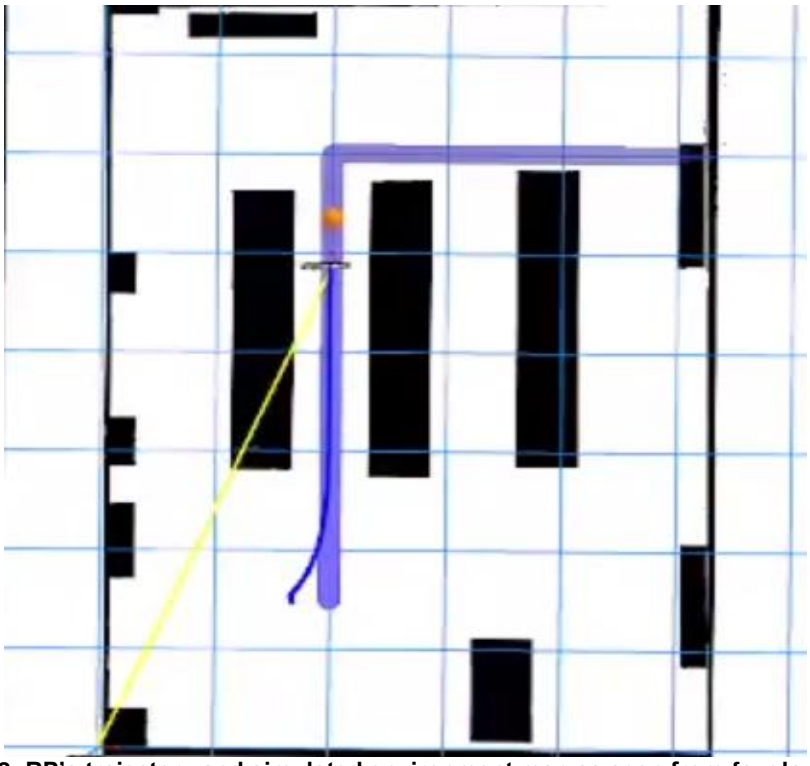


Figure 6-3: RP's trajectory and simulated environment map as seen from foxglove studio.

6.2 Simulated Robotic Platform

The simulated Robotic Platform's design closely follows that of the real robotic platform that is deployed at the Control Systems Laboratory in subsequent experiments. Its motion system features four mecanum wheels to provide the robot with omnidirectional motion capabilities. Mecanum wheels are a type of wheeled drive system that allows land-based vehicles to move in any direction. They are commonly used in mobile machines, such as forklifts and industrial or research robots. Mecanum wheels consist of a series of rollers with an axis of rotation at 45° to the wheel plane and at 45° to the axle line. This allows the wheel to produce both longitudinal and transverse forces, which enables the vehicle to move sideways, diagonally, and even spin in place [25]. Mecanum wheels are used in situations where a small turning radius, high maneuverability and movement on difficult terrain are desired. In the experiments conducted for the purposes of developing the vineyard inspection and reconstruction framework, these wheels were especially valuable for aligning the robot's side camera with the April Tag before performing loop closure.

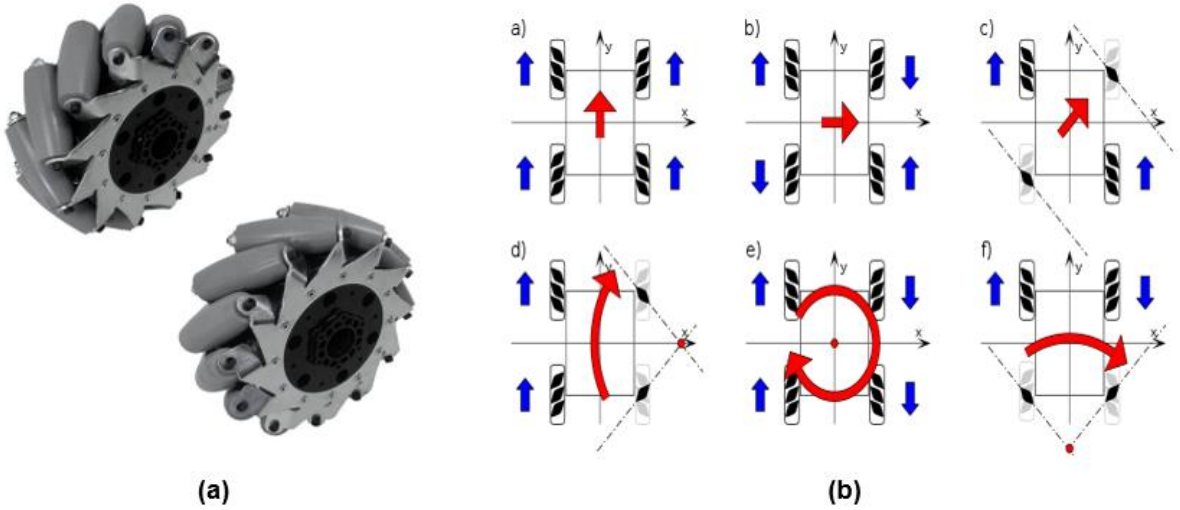


Figure 6-4: (a) A pair of mecanum wheels used on the RP.
(b) Utilizing mecanum wheels to move in various directions:
***blue:* wheel drive direction, *red:* vehicle moving direction, red dot: center of rotation.**

For a comprehensive understanding of the proposed robotic platform's behavior and effective control strategy development, it is crucial to analyze its kinematic properties. If we consider a $x_s O_s y_s$ frame attached to the center of the robot's chassis, we can write the body speed equations as follows:

$$\begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = \frac{R}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ -\frac{1}{l_1+l_2} & \frac{1}{l_1+l_2} & -\frac{1}{l_1+l_2} & \frac{1}{l_1+l_2} \end{bmatrix} \cdot \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix}, \tag{6-1}$$

where R is the wheel radius, ω_i is the angular velocity of the wheel i and l_1, l_2 are the distances between wheel axis and body center. If the speed of the robot is imposed, we have to compute the angular speed of each wheel (inverse velocity solution):

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{1}{R} \begin{bmatrix} 1 & 1 & -(l_1+l_2) \\ 1 & -1 & l_1+l_2 \\ 1 & -1 & -(l_1+l_2) \\ 1 & 1 & l_1+l_2 \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix}, \tag{6-2}$$

to acquire for the RP:

$$v_x(t) = (\omega_1 + \omega_2 + \omega_3 + \omega_4) \cdot \frac{r}{4},$$

$$v_y(t) = (-\omega_1 + \omega_2 + \omega_3 - \omega_4) \cdot \frac{r}{4}, \quad \textbf{(6-3)}$$

$$\omega_z(t) = (-\omega_1 + \omega_2 - \omega_3 + \omega_4) \cdot \frac{r}{4(l_x + l_y)},$$

where $v_x(t)$ is the longitudinal velocity, $v_y(t)$ is the transversal velocity and $\omega_z(t)$ is the angular velocity of the RP.

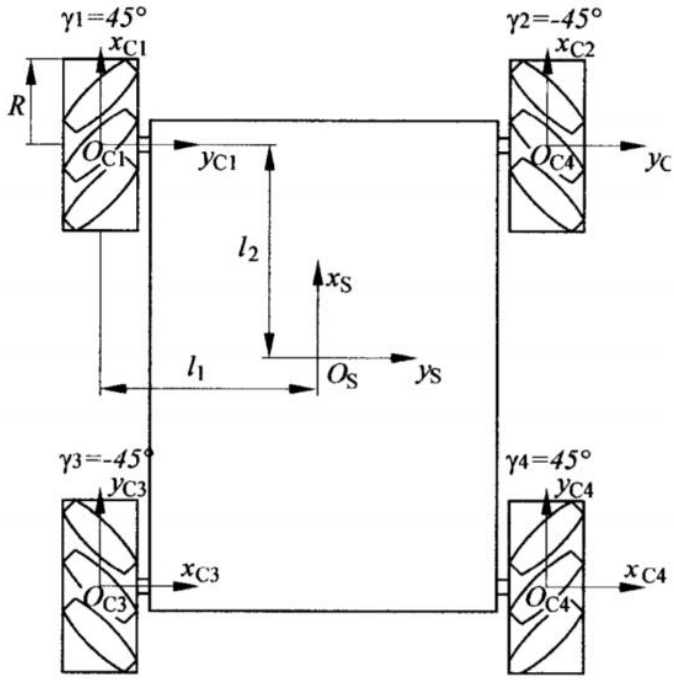


Figure 6-5: Kinematics of the RP.

The robotic platform was fitted with four simulated stereo depth cameras each facing in a different direction. The appropriate ZED sensor plugins and STL CAD descriptions were used so that the simulated cameras closely resemble real ZED cameras.

The design of the perception system mounted on the simulated robotic platform (RP) aligns demonstrably with the theoretical optimal perception system identified through our FOVALIRA evaluations. Future iterations of the RP perception system may incorporate a LiDAR sensor to further enhance its capabilities and achieve even closer alignment with the theoretical optimal system.

A 3D Position Interface for Ground Truth (P3D) was applied on the base of the RP platform. P3D broadcasts the inertial pose of the RP's base link in simulation over ROS via an odometry msg. This is necessary to evaluate the accuracy of the Multi-camera visual odometry by comparing it to the ground-truth odometry produced by P3D.

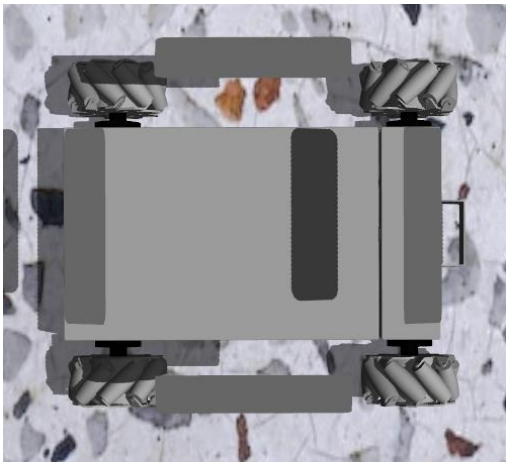


Figure 6-6: The simulated robotic platform.

6.3 Simulation Software architecture

6.3.1 Tracking PID

The robotic platform is tasked with traversing the first and second rows of the simulated vineyard by following a strict trajectory designated to it by the path planning algorithm described in [19]. A trajectory tracking proportional-integral-derivative (PID) controller named tracking_pid ROS [131] is leveraged to enable it to closely follow the designated trajectory. Tracking PID is a versatile and flexible ROS package that comprises two main components: an interpolator and a controller. The interpolator takes a nav_msgs/Path message containing a sequence of waypoints and generates a reference global point (GP) that moves along the path at a specified velocity. This GP serves as the target for the controller, which employs three separate PID loops – longitudinal, lateral, and angular – to track the GP precisely.

Tracking PID offers numerous advantages for trajectory tracking applications. It is highly customizable, allowing for adjustment of various parameters, such as controller gains, acceleration and deceleration limits, and desired velocity. Additionally, it supports various types of ROS path representations, including nav_msgs/Path, geometry_msgs/PoseArray, and moveit_msgs/RobotTrajectory. One of Tracking PID's key features is its ability to accurately track trajectories without compromising velocity smoothness. This is achieved by implementing a carrot tracking strategy, where the GP always lies in front of the robot and the controller keeps the robot at a constant distance l from GP.

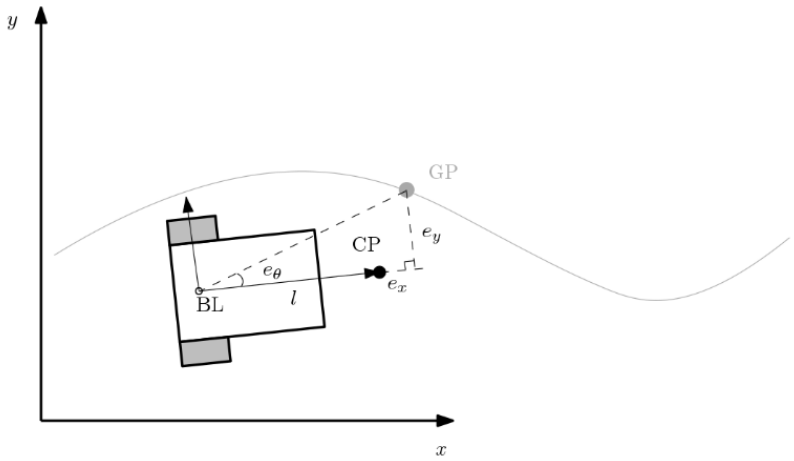


Figure 6-7: Tracking PID: Carrot tracking strategy [131].

e_x, e_y, e_θ: $errors$. CP: $Control\ Point$.

Another notable aspect of Tracking PID is its capability to track trajectories relative to the robot's base_link frame. When enabled, the controller utilizes a Projected Global Point (PGP) that is projected onto the robot's base_link frame. This allows the robot to follow the path in a stricter manner.

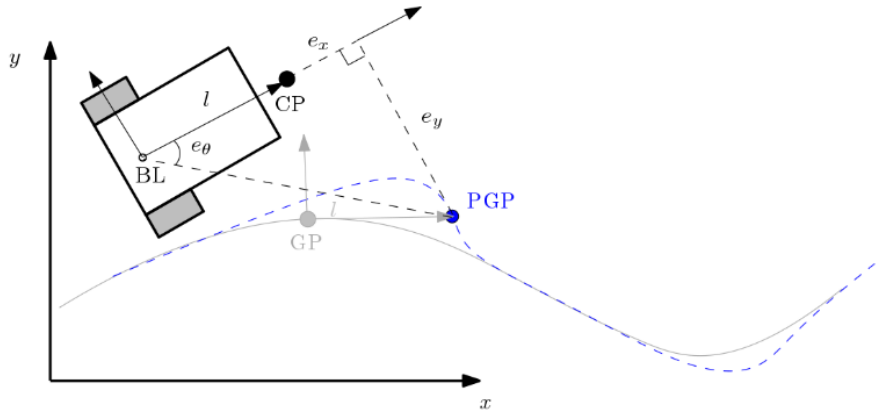


Figure 6-8: Tracking PID: Base_link tracking strategy [131].
e_x, e_y, e_θ: $errors$. PGP: $Projected\ Global\ Point$.

6.3.2 April Tags and Loop Closure

An AprilTag is placed before the first grapevine of the second row of the simulated vineyard. It can be detected using the ZED camera at the right side of the Robotic Platform (RP) by using the apriltag_ros package [132]. The apriltag_ros package outputs the transformation from the camera frame to the detected AprilTag frame. The position and orientation of the April Tag in the map is fixed and accurately known. When the April Tag is detected, the RP can acquire an accurate estimation of its position relative to the April Tag and -because the pose of the April Tag is known- an accurate estimation of its position in the map. The latter pose estimation is compared to the pose estimation acquired from the RP's dual-camera visual odometry. Then, a loop closure thread initiates global bundle adjustment to refine all the previously estimated poses and 3D landmarks, therefore correcting the whole estimated trajectory of the robot. Many state-of-the-art vSLAM algorithms leverage a Bag of Words image representation to detect a loop closure and perform bundle adjustment, as explained in 2.2.1. Using April Tags to initiate bundle adjustment is superior to Bag of Words (BoW) loop closure approaches in a vineyard setting. This is because the crop is organized in parallel and similarly sized rows. There, the bag-of-word approach could result in incorrect loop detection due to the high similarity between images. As a result, such vSLAM approaches are prone to false positive loop detection in homogeneous environments like vineyards.

The pose of the April Tag relative to that of the robot can be accurately estimated from great distances and from various view angles. However, it was observed that the best results were produced when the robot detected the April Tag when it was close to it and with a view angle of around 90^o. Thus, to amplify the valuable odometry corrections of the bundle adjustment process, an alignment node was implemented. The alignment node is active throughout the experiment but only influences the RP after it has detected the April Tag. It then takes over control of the RP and aligns it with the April Tag before registering the April Tag pose. More specifically, as soon as the right camera of the RP completely aligns its left

lens with the April Tag, its pose is registered, and the bundle adjustment node corrects the so-far trajectory of the RP.

6.3.3 Simulation Experiment Pipeline

There are many processes running to facilitate the simulation experiments, mainly in the form of ROS nodes. The main modules are:

- The free corridors path planning algorithm that runs in Matlab utilizes polytopic decomposition and calculates a viable and non-obstructed path through the first two rows of the vineyard, using the 2D vineyard map and producing a dense series of waypoints for the RP to pass through. This is the trajectory planning module.
- The Gazebo simulated world that loads with the synthetic_vineyard.launch launch file. This launch file loads the synthetic vineyard world and spawns the Robotic Platform in a designated position in the vineyard. It also loads the 2D map for visualization purposes and broadcasts necessary static transform messages.
- The dual-camera, real-time (RT) visual odometry node (visual tracking module) that is launched with the DualCamRTAT.launch launch file. This launch file also launches the April Tag (AT) Continuous Detection node. Those functions comprise the Real Time Localization Module.
- The April Tag Continuous Detection node publishes over the ROS network whenever a Tag is detected. It calculates the transformation between the April Tag and the camera that detects it, by analyzing the visual footprint of the April Tag on the camera RGB video.
- The tracking_pid node that controls the RP and ensures that it smoothly tracks the trajectory that is defined by the waypoints produced by Kunal's path planning algorithm. This node comprises the Precise Velocity Control module.
- The view_planning node, which ensures that the robot aligns with the April Tag after detecting it. After the robot is properly aligned, the DualCamRT node will register the April Tag and perform bundle adjustment to correct the so-far obtained odometry data. The view_planning node is also launched by the synthetic_vineyard.launch launch file.

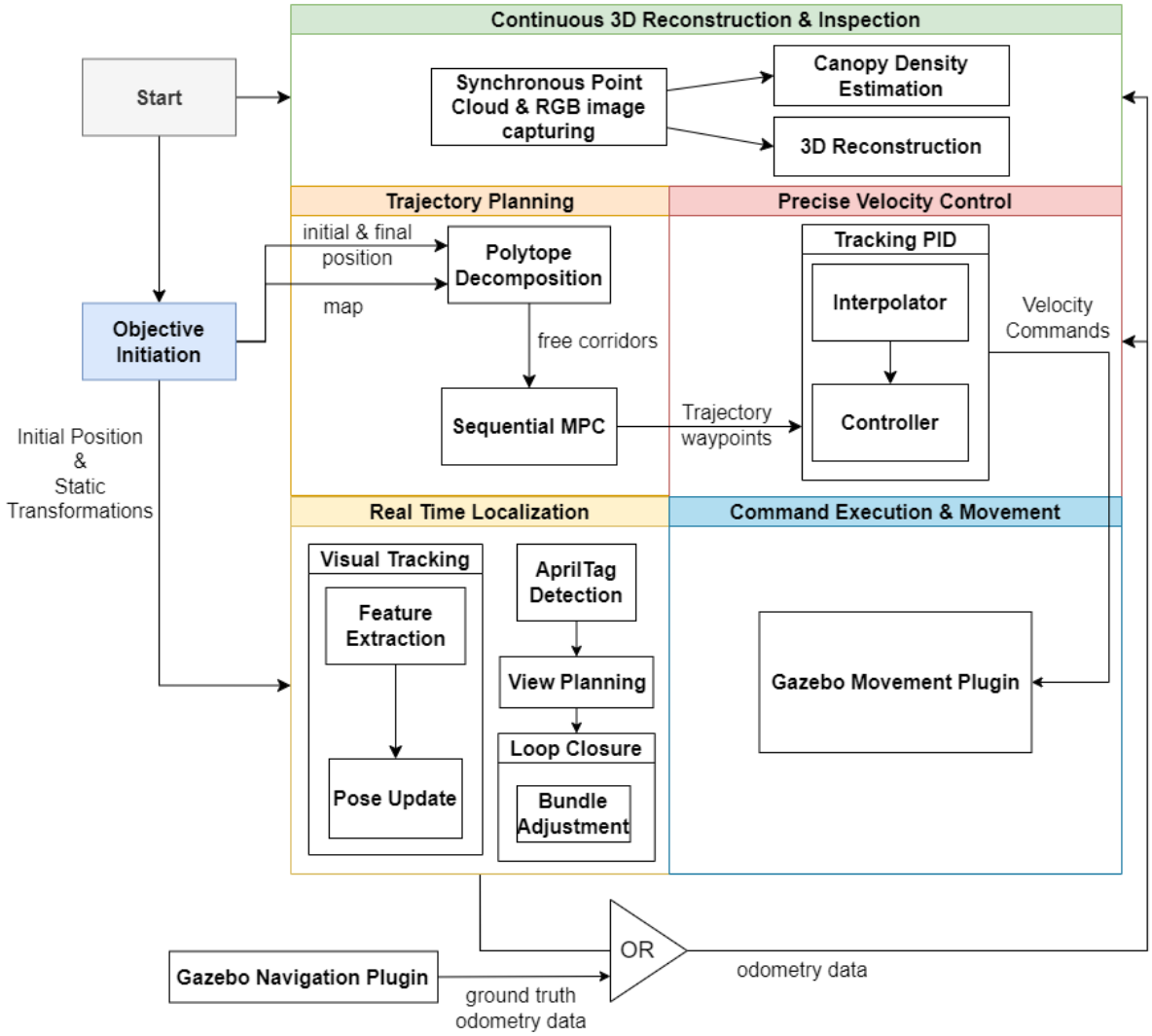


Figure 6-9: Simulation Experiment Pipeline.

Secondary nodes add more functionality to the experiment:

- The position_to_velocity node takes position and rotation data about the RP from the DualCamRTAT visual tracking node and calculates the derivative to output the velocity of the RP in real time.
- The point_cloud_to_pcd node takes point clouds from the right ZED depth camera of the RP in real time and saves them in ".pcd" format in the specified folder. These ".pcd" files are to be used by the Vinymap quality assessment and reconstruction algorithm for canopy density estimation and projective texture mapping. It should be noted that this node simultaneously and synchronously saves RGB images in the specified folder. These are used in the projective texture mapping process of the Vinymap algorithm.
- Rosbag record is a command-line tool used to record ROS messages to a file called a bagfile. Bagfiles are a flexible format that can store a wide variety of ROS messages, including sensor data, actuator commands, and user interactions. They can be used to replay recorded data, analyze data offline, and share data across platforms. Valuable

messages that are broadcasted across a variety of topics are recorded during the experiment to be studied and visualized later in Rviz or Foxglove Studio.

- The P3D gazebo navigation plugin broadcasts ground truth odometry data about the position of the RP in the vineyard. These can be fed into the view_planning node to allow for best alignment with the April Tag for development and testing purposes. They are also used to compare the DualCamRTAT's odometry data accuracy to ground truth data accuracy. When conducting finalized simulation experiments or real-world laboratory experiments, all odometry data used during the experiments are taken from the DualCamRTAT visual tracking node which provides the visual odometry of the RP. During laboratory experiments, a PhaseSpace Impulse X2E [133] motion capture system was used to substitute ground truth data, replacing the P3D gazebo plugin.

7 Laboratory Experiments

7.1 The synthetic vineyard setup

After extensively testing the proposed vineyard inspection and 3D reconstruction framework in a simulation environment, to conduct controlled and repeatable experiments with varying light conditions, a vineyard with artificial grapes and leaves was built at CSL. Each row consists of multiple plants on a trellis system so that the vegetation form resembles a natural canopy. The basic vineyard row parameters, such as the distance between plants (~1m) and grapes' minimum height (0.60m) is based on common viticulture practices in Greece. The artificial grapes' grid features varying density, grape size, creating different visibility conditions since some grapes are partly covered with leaves, whereas others lie on the front plane. The vineyard consists of three 3-meter-long rows on even terrain.

(a) (b)

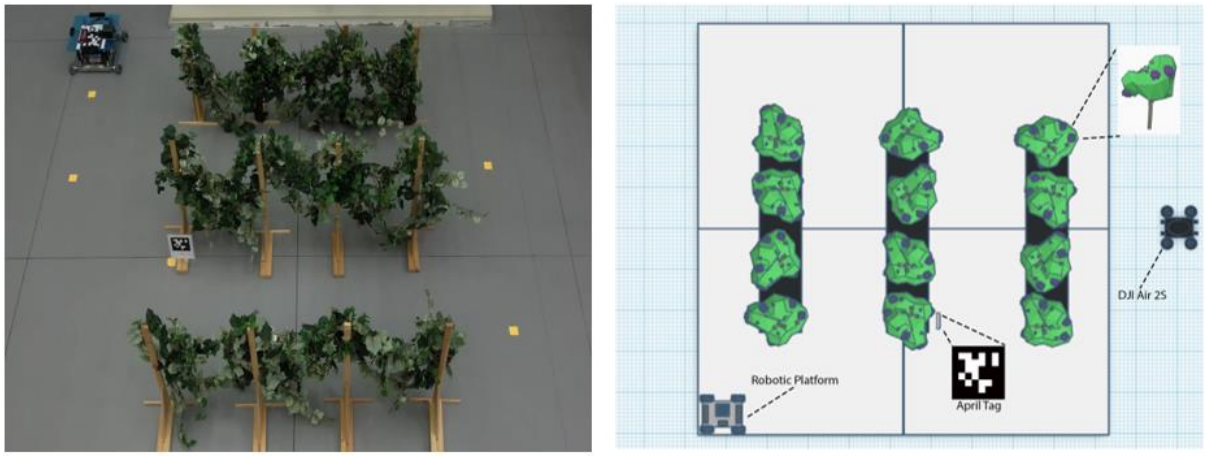
Figure 7-1: (a) Synthetic vineyard in CSL. (b) Natural Vineyard located in the Blue Ridge Mountains, USA.

An April Tag was placed at the start of the second row, at a fixed position, behind the pole of the first vine tree of the second row. To guarantee controlled and repeatable performance of our custom dual-camera visual odometry (VO) algorithm, we opted for a two-pronged testing approach. Firstly, we aimed to acquire ultra-high-quality, near-faultless odometry data to fuel our algorithm development without concerns about erroneous measurements. Secondly, we sought to verify our algorithm's performance in a real-world scenario.

For the first objective, we leveraged the PhaseSpace Impulse X2E [133] motion capture system. This state-of-the-art system employs active LED markers attached to the robot, enabling high-precision tracking within a defined capture space. Unlike camera-based systems susceptible to lighting variations and feature quality, the PhaseSpace system delivers robust motion tracking independent of ambient conditions. This allowed us to acquire highly accurate and reliable ground truth odometry data for meticulous algorithm development and evaluation.

In the second phase, we transitioned to real-world testing settings. Our robotic platform navigated two distinct environments: a naturally lit garage (Figure 6-2) and a variably lit room (Figure 6-3). The PhaseSpace system was deployed in the second location for continuous performance validation. Additionally, during the garage experiments, a DJI Air 2S [134] quadcopter captured aerial photographs and videos, providing valuable supplementary data for multi-angle analysis. This multifaceted approach enabled us to assess our VO algorithm's

efficacy in diverse operational scenarios, bridging the gap between controlled testing and real-world deployment.



(a) (b)

Figure 7-2: (a) Aerial Photography of the Laboratory Experiment Setup taken by the quadcopter. (b) 2D graphical representation of the Laboratory Experiment Setup.

A Robotic Platform developed for research purposes and altered to host the software developed in this work was tasked with traversing the first and second corridors formed by the rows of the vineyard. Its trajectory ends near the April Tag. There, the RP was tasked with aligning the left lens of its right ZED Depth Camera with the stationary April Tag, exactly how it performed in the simulation experiment pipeline. With the knowledge of the position of the stationary April Tag, as well as the knowledge of its relative position of its camera to the April Tag, the RP can recalculate its pose at the time of registering the April Tag and then perform bundle adjustment to correct the so-far odometry data which it collected using the dual camera visual slam node.

Figure 7-3: Indoor testing environment with artificial lighting and PhaseSpace system.

7.2 The Robotic Platform

CSL's Robotic Platform (RP) was used to validate and test the proposed software framework. The RP is designed and constructed for research purposes, comprising custom built in-house parts as well as off-the-shelf parts (e.g. aluminum profiles, bearing units etc.). Its motion system features four mecanum wheels [25] to provide the robot with omnidirectional motion capabilities. The wheels are powered by four Maxon DC motors (RE 35) combined with planetary gearboxes (GP 42) and incremental encoders (HEDL 5540), providing 5 Nm of continuous torque per wheel. GT2 timing belts and pulleys are used to protect actuator shafts from increased robot payloads and to transmit power to the wheels. Two RoboClaw [135] 2x30A motor controllers are used to drive the actuators, since each controller can drive two DC brushed motors. The encoders attached to the motors are read by the controllers, which run local PID control schemes that can precisely follow speed commands for all wheels. The two motor controllers are connected via USB to the system's master computer, which is a Raspberry Pi 2 model B [136] (RPi) running the Raspbian OS. The operator can connect to the RPi using WiFi and Secure Shell (SSH) Network Protocol to run a Python script that establishes two serial connections with the motor controllers and sends the desired motion commands. The system is powered by two Wild Scorpion 6S 22.2v 4200mAh 60C LiPo batteries [137] for the RoboClaw controllers and a power bank for the master computer (RPi).

The perception system of the RP was designed according to the guidelines defined in Chapter 4 of this thesis. Although the form factor of a rover differs a lot from that of a quadruped robot such as Lealaps or Argos, the hardware which best suits the application of vineyard inspection stays, in principle, the same. Thus, the RP was fitted with 3 ZED Depth cameras – one at the front of the RP, one at the rear and one at the right side. Due to the

lower height of the RP, the cameras were not tilted downwards; This would be beneficial for a legged robot with a larger form factor as described in section 4.4 of this work, but this is not the case for the CSL RP since doing so would move most of the vineyard out of the camera field of view. The front and rear cameras are ZED 2 Depth cameras, while the right side camera is a ZED X Mini Depth Camera with a 2.2 mm focal length [138]. A shorter focal length implies a larger field of view which is desired in this configuration, as a larger area of the grapevines can be inspected with a single traversal of the rows. The ZED X Mini Depth Camera is also fitted with polarizing lens, which increases the RP's ability to operate under diverse lighting conditions.

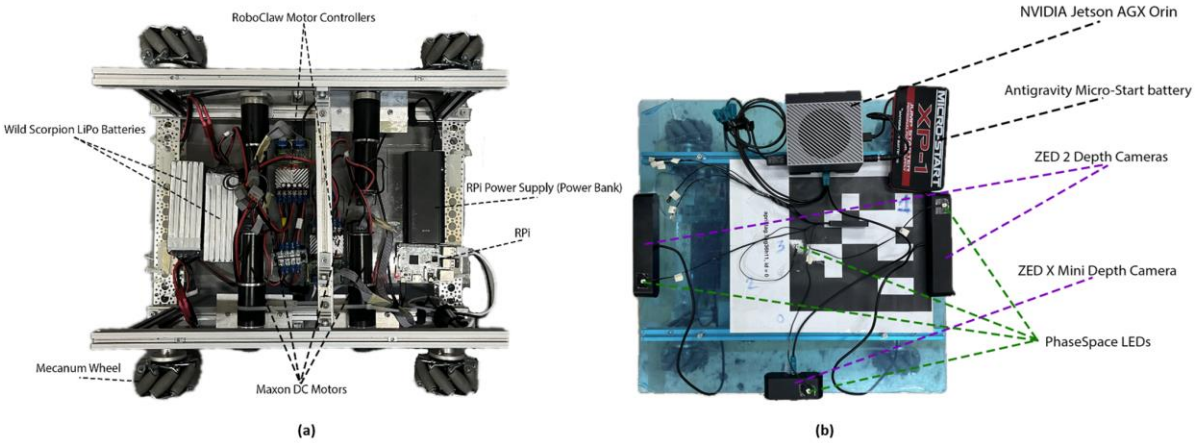


Figure 7-4:(a) The Robotic Platform without its perception system. (b) The perception system of the Robotic Platform.

The software framework runs on the Nvidia Jetson AGX Orin which is the processing platform purposefully chosen for the perception system proposed in this thesis, as described in section 4.2. All three depth cameras are connected to Orin. The ZED 2 cameras are connected via USB 3.0, while the ZED X Mini is connected via a GMSL2 Fakra Cable. To support this connection, the Nvidia Jetson Orin is fitted with a GMSL2 Capture Card. To power the Orin, an Antigravity Micro-Start battery [139] was chosen to provide it with 12V at 3.5A DC.

7.3 Laboratory Experiment Software Architecture

There are many processes running to facilitate the experiments conducted at the synthetic vineyard in the laboratory, mainly in the form of ROS nodes. The main modules are:

- The free corridors path planning algorithm that runs in Matlab utilizes polytopic decomposition and calculates a viable and non-obstructed path through the first two rows of the vineyard, using the 2D vineyard map and producing a dense series of waypoints for the RP to pass through. This is the trajectory planning module.
- The v-slam.launch launch file which is the file that contains the main software functionality. It includes the DualCamRTAT.launch visual tracking module file and launches the map loading node, the zed_multicamera_nodelet node, the tracking_pid node, the view_planning node, several static tf_publisher nodes and three nodes that offer extra functionality; the position_to_velocity, point_cloud_to_pcd and rosbag record nodes.

- The zed_multicamera_nodelet is launched by the v-slam.launch file and activates all three cameras connected to the Jetson AGX Orin. It launches the cameras one by one using their serial numbers and sets up their output video feed.
- The map loading node which publishes a static 2D vineyard map of the fixed laboratory environment over ROS, mainly for visualization purposes.
- The dual-camera, real-time (RT) visual odometry node that is launched with the DualCamRTAT.launch launch file. This launch file also launches the April Tag (AT) Continuous Detection node. These nodes are part of the Real Time Localization module.
- The April Tag Continuous Detection node publishes over the ROS network whenever a Tag is detected. It calculates the transformation between the April Tag and the camera that detects it – in this case the ZED X Mini - by analyzing the visual footprint of the April Tag on the camera RGB video.
- The tracking_pid node that controls the RP and ensures that it smoothly tracks the trajectory that is defined by the waypoints produced by Kunal's path planning algorithm.
- The view_planning node, which ensures that the robot aligns with the April Tag after detecting it. After the robot is properly aligned, the DualCamRT node will register the April Tag and perform bundle adjustment to correct the so-far obtained odometry data.
- The udp_client node which receives velocity messages from the tracking_pid node and sends them via udp to Raspberry Pi. This node is part of the Command Execution and Movement module.
- The RPi python script running on the RPi computer receives messages from the udp_client node and sends the respective velocity commands to the RoboClaw Motor Controllers over serial communication protocol. The RP's motors are then powered accordingly.

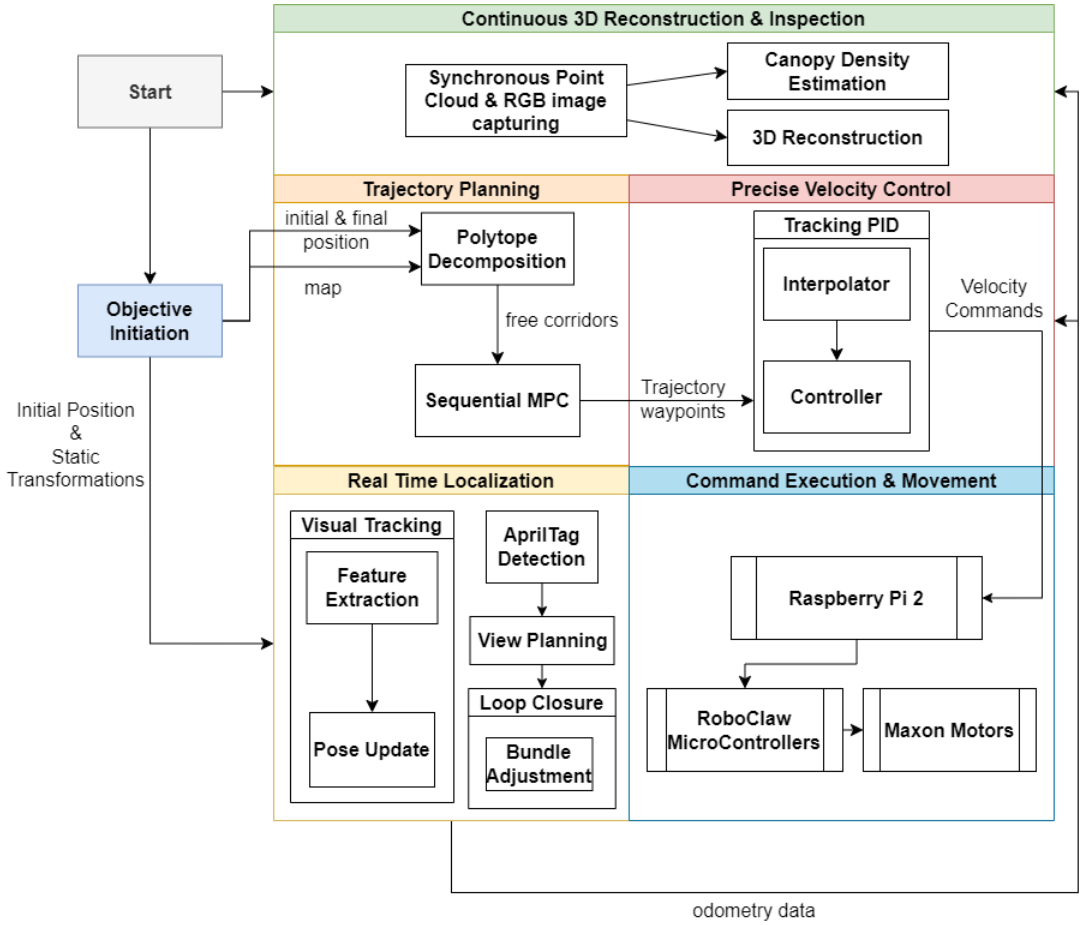


Figure 7-5: Laboratory Experiments Pipeline.

Secondary nodes add more functionality to the experiment:

- The position_to_velocity node takes position and rotation data about the RP from the DualCamRTAT visual tracking node and calculates the derivative to output the velocity of the RP in real time.
- The point_cloud_to_pcd node takes point clouds from the right ZED depth camera of the RP in real time and saves them in ".pcd" format in the specified folder. These ".pcd" files are used by the Vinymap quality assessment and reconstruction algorithm. It should be noted that this node simultaneously and synchronously saves RGB images in the specified folder. These are used in the projective texture mapping process of the Vinymap algorithm.
- Rosbag record is a command-line tool used to record ROS messages to a file called a bagfile. Bagfiles are a flexible format that can store a wide variety of ROS messages, including sensor data, actuator commands, and user interactions. They can be used to replay recorded data, analyze data offline, and share data across platforms. Valuable messages that are broadcasted across a variety of topics are recorded during the experiment to be studied and visualized later in Rviz or Foxglove Studio.
- During experiments within the room with artificial lighting, a PhaseSpace Impulse X2E motion capture system was used to substitute ground truth data. The data from this setup were captured and saved separately to aid with post-experimental analysis and review.

8 The Vinymap Quality Assessment and Reconstruction Algorithm

8.1 Custom SOPCQA

Recent literature has witnessed advancements in PCQA methodologies, with both full-reference (FR) and no-reference (NR) metrics being developed. However, there is room for improvement as most no-reference PCQA methods do not yet statistically correlate well with subjective quality assessments [140]. In addition, most recent studies propose learning based methods that come with greater complexity and are harder to interpret and improve. As a result, the development of a Simple Objective Point Cloud Quality Assessment algorithm was deemed valuable for a perception system focused on vineyard inspection. The proposed in-house SOPCQA algorithm can be subdivided into three parts:

(a) Sparsity Index Calculation
(b) Hole Detection
(c) Cluster Outlier Detection.

8.1.1 Sparsity Index Calculation

One way humans use to realize if a 3D point cloud is of low quality or not is to observe the density of the point cloud. Point clouds of higher quality are homogenous in terms of density and do not include low density sub-clouds.

The calculation of a point cloud's (pcd) sparsity index comes down to detecting sparse areas in the assessed point cloud i.e. areas where the points of the point cloud have on average a small number of neighbors. The volume that these areas take up is the sparse volume of the point cloud. The sparse volume of the point cloud is calculated by leveraging Open3D's mesh creation and volume calculation functions. It is then divided by the total point cloud volume (sparse and dense). The result of this division is the sparsity index of the point cloud. The higher the sparsity index, the sparser the point cloud and consequently the noisier the sensor measurements that created it and the lower its quality. Algorithm 7-1 presents the process in greater detail. Figure 7-1 provides an intuitive illustration. To find sparse areas in the point cloud, a sparsity metric for each point must be defined. The number of the neighbors of each point in its vicinity epsilon was used.

ALGORITHM 7-1 SPARSITYINDEXCALCULATION

Require: pcd, epsilon, sparsity_threshold

Output: sparcity_index, remapping_recommendation

```
pcd.remove_non_finite_points()
sum_of_neighbours = 0
sparse_points = [ ]
for point in pcd:
    point.neighbours = find_neighbours_using_kdtree(point, epsilon)
```

```
        sum_of_neighbours = sum_of_neighbours + point.neighbours
    end for
    pcd_mean_density = sum_of_neighbours / len(pcd)
    for point in pcd:
        if point.neighbours < pcd_mean_density then:
            sparse_points.append(point)
        end if
    end for
    sparse_pcd = pcd_from_points(sparse_points)
    sparse_mesh = BPA(sparse_pcd)
    total_mesh = BPA(pcd)
    sparse_area = sparse_mesh.calculate_surface_area
    total_area = total_mesh.calculate_surface_area
    sparsity_index = sparse_area / total_area
    remapping_recommendation = (sparsity_index > sparsity_threshold)
output sparsity index, remapping_recommendation
```

The problem of efficiently finding the close neighbours for each point of the point cloud is formulated as follows: Given a point cloud (a set of points in a multidimensional space) and a radius epsilon, identify all neighboring points within a radius epsilon for each individual point in the cloud. K-d Trees were utilized to speed up the process.

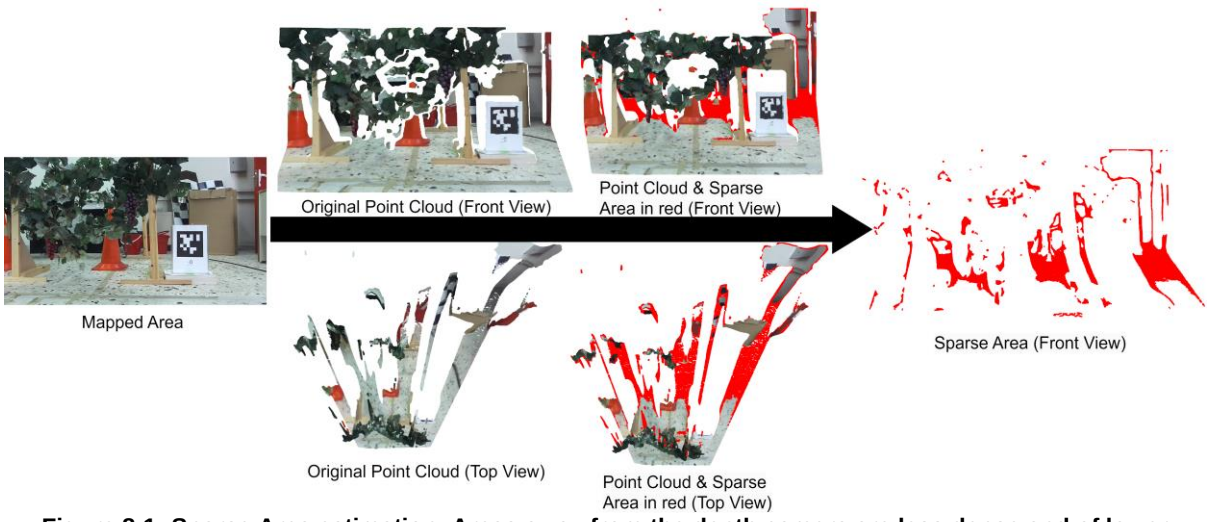


Figure 8-1: Sparse Area estimation. Areas away from the depth camera are less dense and of lower quality, as expected from such a sensor.

A k-d tree (k-dimensional tree) is a space-partitioning data structure that recursively subdivides the point cloud along different axes. The subdivision process creates a tree-like hierarchy where each node represents a region of space, enabling efficient spatial searches [141]. To find neighbors within radius epsilon of a query point q, the point cloud k-d tree

needs to be traversed; Starting at the root, q is compared with the division value at the current node. The process continues with the corresponding child node (left or right) based on the comparison. This process is repeated along successive dimensions until reaching a leaf node. Then the distance between q and points within the leaf node is calculated. If the distance is less than epsilon, the points are added as neighbors. Finally, the region in the opposite branch of the previous division is checked, as it might contain points closer than the current radius. Any subtrees whose bounding regions are fully beyond the epsilon distance from q are pruned.

The Ball Pivoting Algorithm (BPA) [142] is used for mesh creation. The resulting sparsity index comes from the division of the surface area of the sparse point cloud by that of the whole point cloud and if it is found to be more than a user-defined threshold, then the program outputs a remapping recommendation, as the assessed point cloud is found to contain a large sparse area.

8.1.2 Hole Detection

Hole detection refers to the identification of vacant spaces inside the cloud. Oftentimes, when a reflective object (like a mirror or a pc monitor) is placed in the reconstructed scene, the sensor cannot accurately reconstruct the object and void space, or noise is registered in its place. This can also happen when the sensor's view is obstructed in some way and a blind spot is created. This results in holes in the point cloud. It is intuitive to consider that in a 3D reconstruction of a vineyard there should not be any holes of large volume in the ground or in the vegetation of the vine trees, especially in the dense vegetative states of the vineyard. Nevertheless, the presence of holes could indicate a real problem in the vegetation and not a loss of point cloud quality. Therefore, this algorithm's parameters should be tuned carefully to match the intended use case and conditions. The purpose of this procedure is to identify abnormal holes in the point cloud and not physical holes in the vegetation. The latter is dealt with by the canopy density assessment algorithm which is extensively analyzed in section 7.5.

The hole detection algorithm works by filling the whole 3D workspace with a homogenous dense point cloud (HMDpcd). The assessed source point cloud is then subtracted by the HMDpcd. The resulting point cloud mask is then clustered using DBSCAN clustering. Of the resulting clusters the larger ones do not represent holes and are thrown away. The remaining smaller clusters represent holes (void spaces) of the assessed point cloud and are saved. Depending on their size and quantity, the user can infer the assessed point cloud's quality. To make the algorithm's output more objective, the clusters that represent holes are turned into meshes via surface reconstruction with BPA. Their combined surface area is divided by the total surface area of the initial point cloud and the result is the hole index, the output of the algorithm. Algorithm 7-2 presents the hole detection algorithm in greater detail. Figure 7-3 illustrates the process.

ALGORITHM 7-2 HOLEDETECTION

Require: pcd, clustering_parameters, hole_size_threshold

Output: hole_index

```
bbox = pcd.get_bounding_box()
all_space_pcd = bbox.fill_with_points()
```

```
    free_space_pcd = all_space_pcd.remove(pcd)
    free_space_pcd.erode()
    free_space_pcd.dilate()
    free_space_pcd.cluster(DBSCAN, clustering_parameters)
    free_space_pcd.remove_largest_cluster()
    holes = [ ]
    for hole in free_space_pcd.clusters:
        if hole.size > hole_size_threshold then:
            holes.append(hole)
        end if
    end for
    hole_area = 0
    total_hole_area = 0
    for hole in holes:
        hole_mesh = BPA(hole)
        hole_area = hole_mesh.calculate_surface_area
        total_hole_area = total_hole_area + hole_area
    end for
    pcd_mesh = BPA(pcd)
    pcd_area = pcd_mesh.calculate_surface_area
    hole_index = total_hole_area / pcd_area
output hole_index
```

The *get_bounding_box()* function returns the bounding polygon of the point cloud it is applied on. The *fill_with_points()* function fills this polygon with random points and returns a uniform pointcloud that covers all the space inside the polygon with a specified point density. The *remove()* function, when applied on a point cloud receives as arguments another point cloud and removes all points from the first point cloud that are closer than a specified threshold to the second point cloud. Finally, the *erode()* and *dilate()* functions operate on a point cloud similarly to how erosion and dilation effect a 2D image. Figure 7-2 illustrates the process of dilation after erosion on a 2D image.

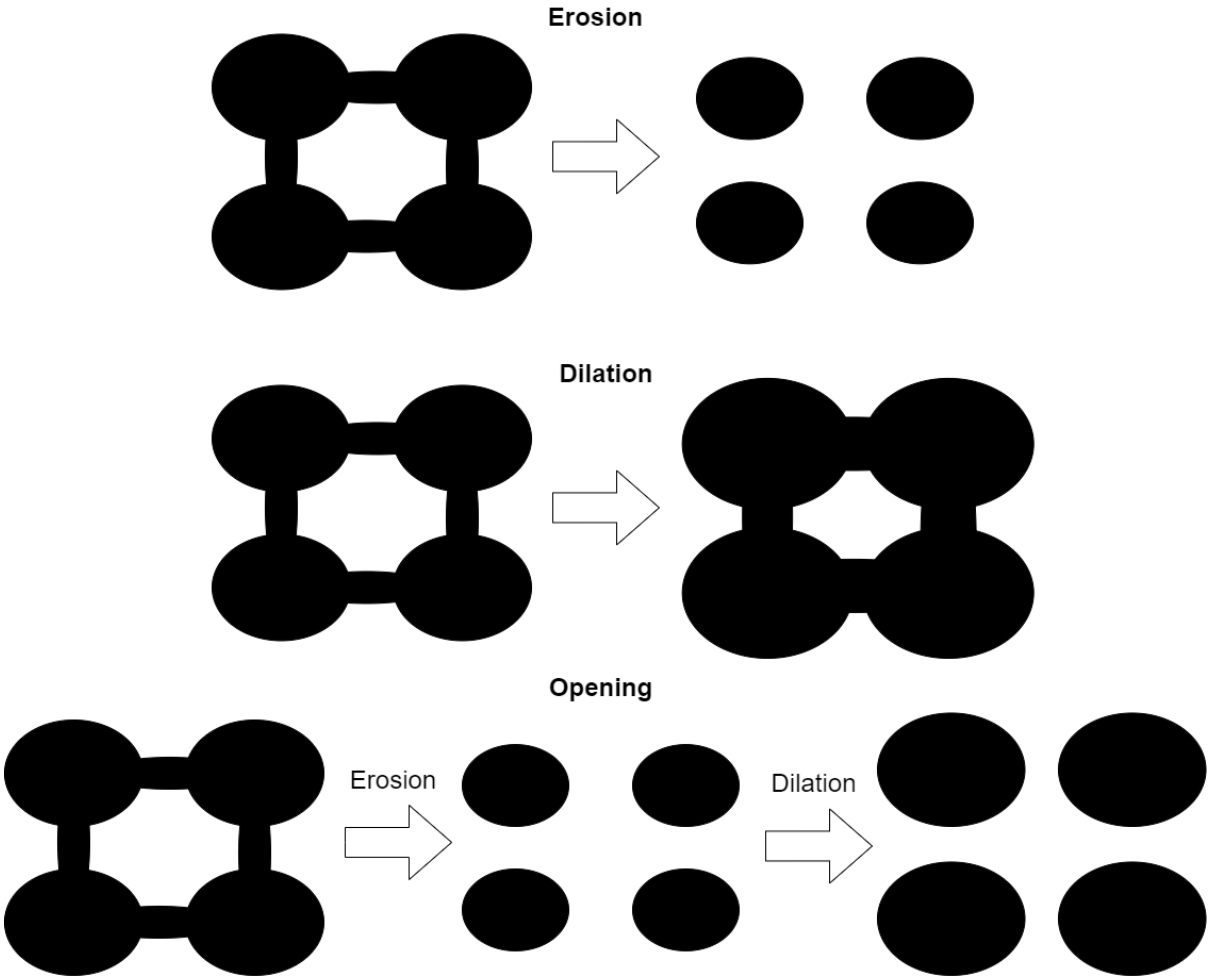


Figure 8-2: Erosion, Dilation & Opening performed on a 2D image.

This operation is also known as opening in image processing. The *free_space_pcd* represents all the space around the source point cloud. After eroding and dilating the *free_space_pcd*, groups of points of the *free_space_pcd* that were isolated inside crevices and holes of the source point cloud are completely separated from the other points of the *free_space_pcd.* Thus, where there were once holes of the source point cloud, now there are points of the *free_space_pcd*. These points represent the holes of the source point cloud. The algorithm then clusters the *free_space_pcd*, so that the holes become separate point clouds. The largest cluster, however, does not represent a hole but the rest of the free space. So, it is removed. In addition, the algorithm discards holes that are too small. Finally, the algorithm calculates the area that the holes and the source point cloud occupy and it produces and outputs the hole index, which is a valuable quantity for objective point cloud quality assessment.

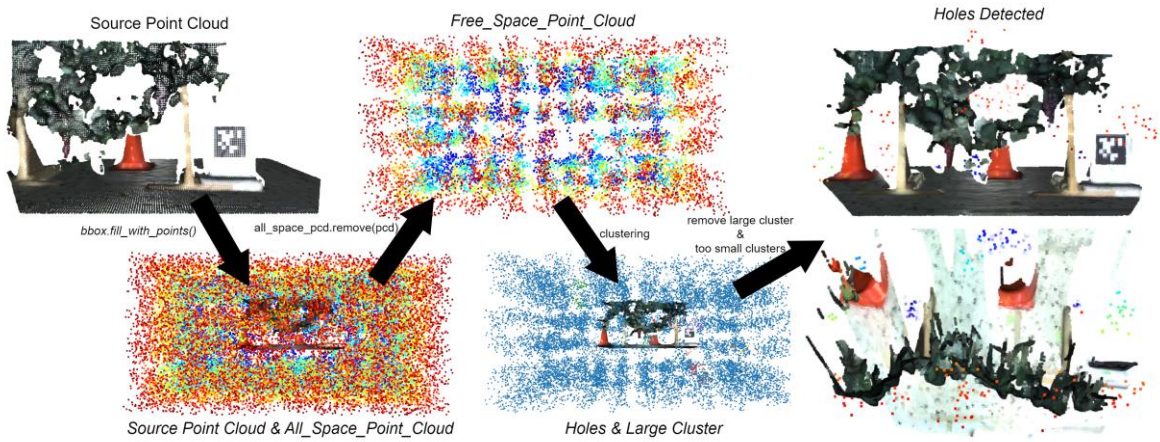


Figure 8-3: The hole detection algorithm visualized.

8.1.3 Cluster Outlier Detection

Detecting small and isolated clusters is referred to as cluster outlier detection in this work. When creating a point cloud of a vineyard, these clusters are intuitively considered as noise because they do not match any existing object in the scene. The quantity and size of these noise clusters is therefore a valuable and simple metric of point cloud quality. The assessed point cloud is clustered using DBSCAN. The user heuristically defines a cluster size threshold; clusters that are smaller than this threshold are considered as noise and their aggregated surface area and quantity is calculated. These two metrics are used in the final quality assessment step. Algorithm 7-3 presents the cluster outlier detection algorithm in greater detail. Figure 7-4 illustrates the process.

ALGORITHM 7-3 CLUSTEROUTLIERDETECTION

```
Require: pcd, clustering_parameters, cluster_size_threshold
Output: noise_clusters_quantity, total_noise_clusters_area
    pcd.cluster(DBSCAN, clustering_parameters)
    noise_clusters = [ ]
    noise_clusters_quantity = 0
    for cluster in pcd.clusters:
        if cluster.size < cluster_size_threshold then:
            noise_clusters.append(cluster)
            noise_clusters_quantity++
        end if
    end for
    noise_cluster_area = 0
    total_noise_clusters_area = 0
    for noise_cluster in noise_clusters:
        noise_cluster_mesh = BPA(noise_cluster)
        noise_cluster_area = noise_cluster_mesh.calculate_surface_area()
```

total_noise_clusters_area = total_noise_clusters_area + noise_cluster_area

end for

output noise_clusters_quantity, total_noise_clusters_area

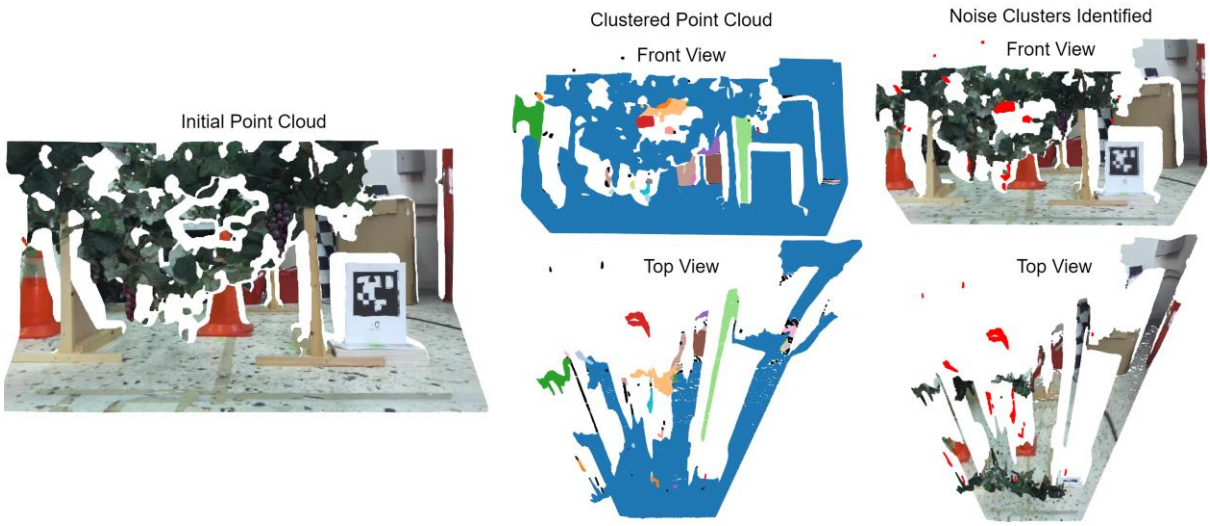


Figure 8-4: Cluster Outlier Detection. The noise clusters are pictured in red.

8.1.4 Final SOPCQA Algorithm

Table 7-4 shows the final SOPCQA algorithm. This algorithm uses the previous three algorithms as implemented functions to objectively assess the quality of an input point cloud. Eq. (7-1) shows the calculation of the final quality index (FQI). The quality threshold (QT) mentioned in the algorithm is determined heuristically by the user.

ALGORITHM 7-3 SOPCQA

Require: pcd, epsilon, sparsity_threshold, clustering_parameters, hole_size_threshold, cluster_size_threshold, fqi_parameters, QT

Output: final_quality_index, remapping_recommendation

sparsity_index = sparsityIndexCalculation(pcd, epsilon, sparsity_threshold)

hole_index = holeDetection(pcd, clustering_parameters, hole_size_threshold)

noise_clusters_quantity, total_noise_clusters_area = clusterOutlierDetection(pcd, clusutering_parameters, cluster_size_threshold)

FQI = calculateFQI(sparsity_index, hole_index, noise_clusters_quantity, total_noise_clusters_area)

if FQI < QT **then:**

remapping_recommendation = True

else:

remapping_recommendation = False

end if

output FQI, remapping_recommendation

The calculateFQI function in Algorithm 7-3 applies the following equation:

$$f = e^{-\beta_s s} e^{-\beta_h h} - \beta_n q - \beta_n a \,, \quad (87-$$

, where f is the final quality index (FQI), $(\beta_s, \beta_h, \beta_n)$ are user-defined parameters, s is the sparsity index, h is the hole index, q is the quantity of outlier clusters and a is the total area of outlier clusters.

In the current thesis, the SOPCQA algorithm is used in conjunction with a vineyard point cloud dataset. This dataset contains point clouds that have been successively and periodically captured by a ZED depth camera mounted on a moving rover robot that traverses a synthetic vineyard with constant linear velocity. Sections 5 and 6 describe this setup in greater detail. The algorithm works by iterating through the point clouds in the dataset and calculating the FQI for each one of them. If the FQI is lower than the QT for an assessed point cloud, then the point cloud is deemed of lower quality and the user is prompted to recapture the point cloud from another view angle by appropriately maneuvering the rover. This process could be automated but this is not implemented in the context of this work. However, as described in section 7.2, after capturing the vineyard point cloud dataset and assessing each point cloud's quality, software is used to improve each of the point clouds' quality. The user can leverage SOPCQA to label point clouds of lesser quality and only apply improvements to them, thus saving on computational resources.

8.2 Point Cloud Quality Improvement and Registration

8.2.1 Quality Improvement

Point cloud registration is the process of finding a transformation (a combination of rotation, translation, and scaling) that best aligns the two or more point clouds. High quality point clouds can be accurately registered using simple optimization-based algorithms like Iterative Closest Point or Point Pair Feature Descriptors (PPF) [143]. For more elaborate or lower quality point clouds resorting to more complex learning-based algorithms is considered beneficial. In the context of this work, due to the noise in the available point clouds, a decision had to be made between improving the point clouds' quality to register them with simple methods and using the raw point cloud data to register them with more complex algorithms. The first option was chosen in the interests of simplicity as statistical point cloud quality enhancement algorithms have been thoroughly studied and efficiently implemented in the literature [144]. Algorithm 7-4 and Figure 7-5 present the custom pipeline developed for that purpose.

ALGORITHM 7-4 POINTCLOUDQUALITYENHANCEMENT

Require: Set of point clouds $P = \{p_1, \dots, p_k\}$, v, b, n, e, t

Output: Set of enhanced point clouds B

B = LoadPointClouds(P, v)

for p **in** B:

CropPointCloud(P,v)

CloudFilterByDensity(p,n,e)

CloudFilterByClusters(p,t)

NormalEstimation(p)

end for

output B

The utilized functions are described as follows:

LoadPointClouds: Loads a batch B of point clouds out of the set of point clouds P that were captured by the ZED X Mini camera and formats them appropriately for processing with Open3D. It removes non-finite points and transforms the point clouds to align with Open3D's world frame. It downsamples the point clouds utilizing voxel_downsampling with voxel size v for memory management purposes.

CropPointCloud: Receives a point cloud p and a set of bounding limits b as input. Calculates p's oriented bounding box and shrinks it to satisfy b. It then crops p to fit inside the modified bounding box. This function is utilized to ignore areas of the point cloud that lie in the limits of ZED X's field of view and are particularly noisy.

CloudFilterByDensity: Removes points of the input point cloud that have few neighbours n within a specified radius e. Low density areas in the point cloud are usually products of noise.

CloudFilterByClusters: Separates the input point cloud in clusters using DBSCAN clustering [22]. Clusters that are smaller than a specified threshold t are usually products of noise and are removed.

NormalEstimation: The function finds adjacent points in the input point cloud and calculates the principal axis of the adjacent points using covariance analysis.

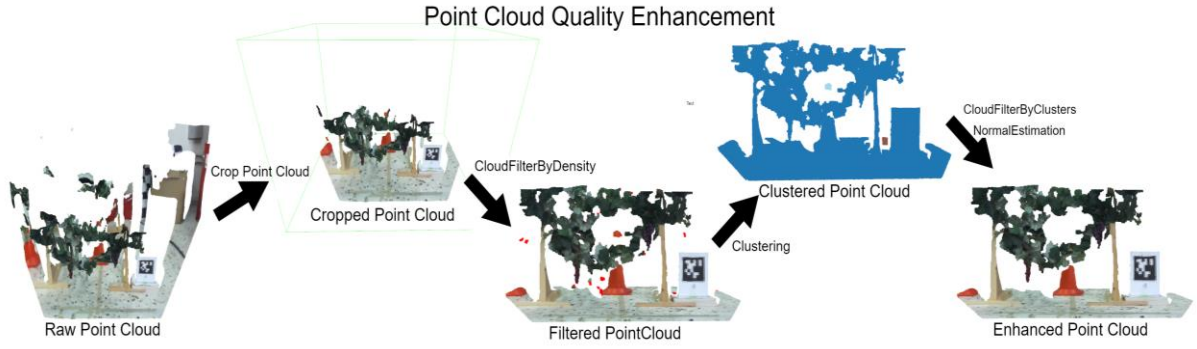


Figure 8-5: The Point Cloud Quality Enhancement Pipeline.

8.2.2 Registration

After being enhanced, the point clouds are registered in pairs using a custom variant of the ICP optimization algorithm. ICP works by finding the closest point in one point cloud (source) for each point in the other point cloud (target). This process is called data association. The algorithm then finds the target point cloud's transformation (which can be represented by a rotation matrix and a translation vector) that minimizes the distance between the corresponding point pairs. After applying this transformation to the target point cloud, the data association changes and thus must be recomputed. The transformation matrix that

minimized the distance between the corresponding point pairs is also recomputed and applied again. This process is repeated until the overall error between the two point clouds is minimized. This error is typically measured using a distance metric, such as the sum of squared distances between corresponding point pairs.

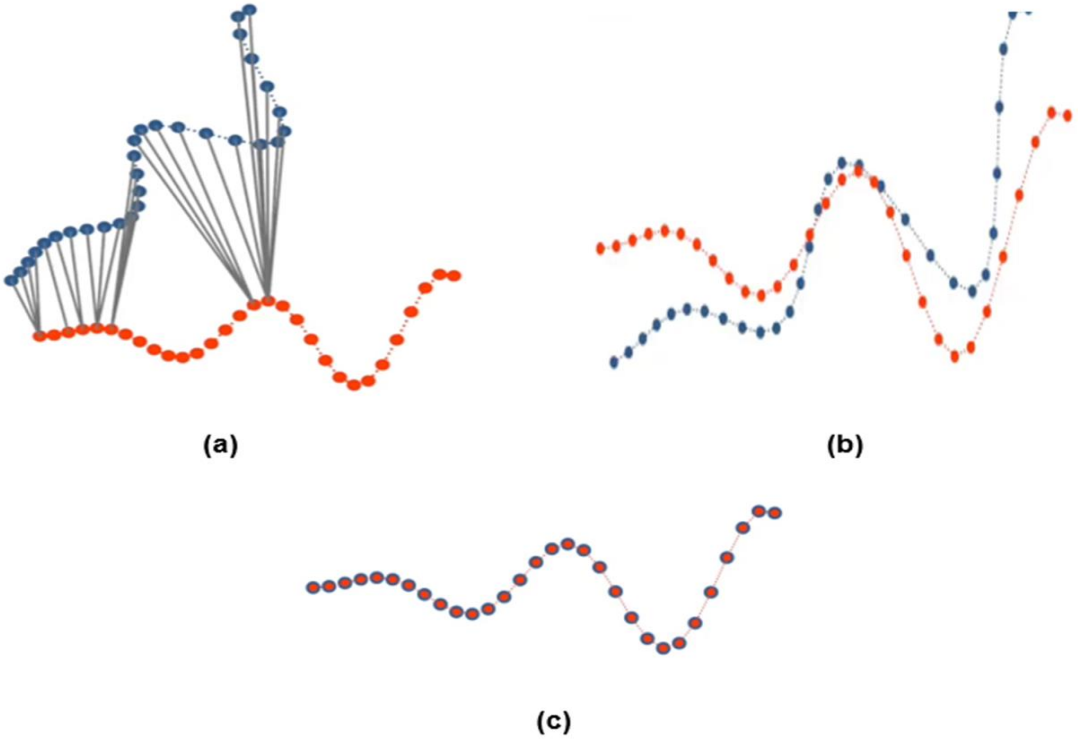


Figure 8-6: a) Data association (b) Target point cloud transformation. Result after first iteration (c) Final point cloud registration after four iterations.

There exist several variants of the ICP algorithm. Estimating the data association to avoid convergence into a local optimal solution is the key challenge to the development of a robust ICP algorithm [145]. Similar to the approach followed by the authors in [22], a simple yet effective ICP variant was constructed for Vinymap.

The point-to-plane data association method was chosen. Instead of matching individual points, point-to-plane data ICP involves fitting a plane to a set of points in the target point cloud (normal estimation) and then selecting the closest point to that surface in the source point cloud. The distance between point and plane is calculated using the perpendicular distance metric. This approach is more robust to noise and outliers compared to point-to-point data association, making it suitable for handling real-world data with imperfections. To further reduce the effect of existent noise and outliers, correspondences of points that are two far away from one another are not considered. The distance that defines how close two paired points must be in order to be considered is called maximum correspondence distance (MCD).

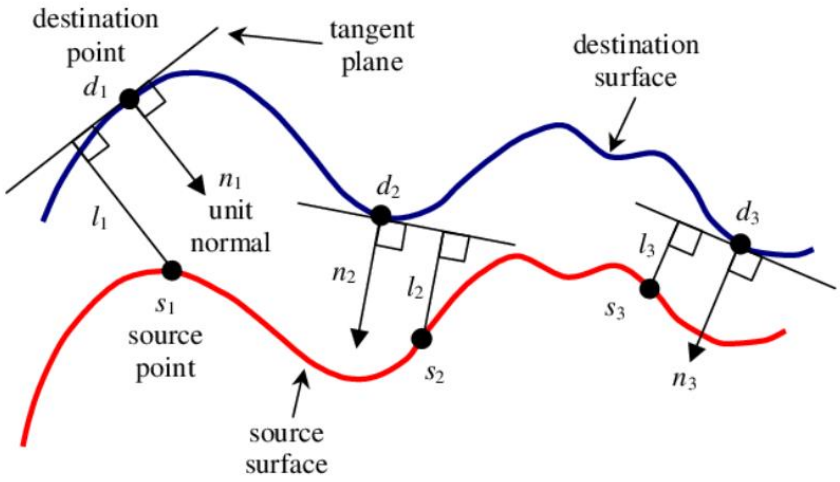


Figure 8-7: Point-to-Plane correspondence illustration in 2D [146].

The first two point clouds are read from the dataset. The initial point cloud is assigned the identity transformation matrix as its pose. The second point cloud must be correctly registered to the first one. At this point no assumptions can be made about the second point cloud's pose and it is therefore also assigned the identity transformation matrix. An initial ICP transformation with a relatively high MCD is initiated. A second ICP transformation with a finer MCD follows and finishes the registration. The final transformation of the newly added point cloud is saved in an odometry list which is valuable for the next steps of the Vinymap algorithm. The newly added point cloud is then fused with the initial point cloud and the resulting point cloud is called a local map.

The third point cloud is read from the dataset and is initially transformed with the same matrix as the second one, but with an added 10% translation and rotation in the same direction. This percentage is defined heuristically and can be changed as an internal registration parameter. ICP is then applied to register the third point cloud with the local map (which is currently comprised of the first two point clouds). The resulting final transformation is applied to the third point cloud and saved in the odometry list. The third point cloud is then added to the local map.

This process is repeated until every point cloud has been correctly registered and a local map comprised of all the aligned point clouds is created. It should be noted that in parallel to the local map, a global map is also being constructed. The global map contains only the point clouds whose registration yielded a registration error lower than a predefined threshold. This is done to ensure that misregistered point clouds will be discarded.

Finally, for each registered point cloud, a bounding polygon is created and saved. This polygon is calculated by leveraging open3D's convex hull computation function, flattening the hull and then extracting a bounding volume which tightly encloses the point cloud. The resulting bounding polygon is utilized in the mesh projection phase of Vinymap.

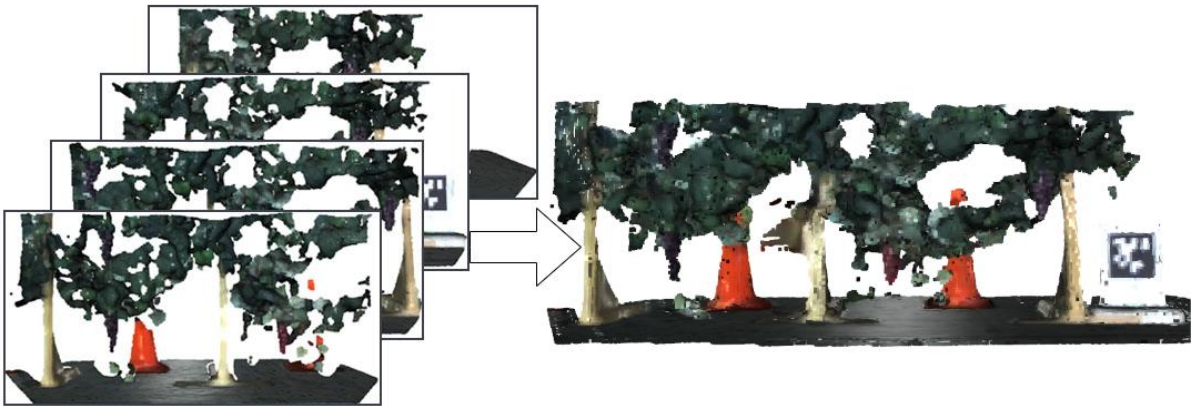


Figure 8-8: Point Cloud Registration in the synthetic vineyard.

8.3 Canopy Density Assessment

The Vinymap framework incorporates a novel strategy for identifying gaps within the vineyard canopy. This method capitalizes on the inherent color properties of the vine tree leaves, as well as the topological properties of surface reconstruction with alpha shapes. We implement Algorithm 7-5 to output a comprehensive Canopy Density Index (CDI) for each input point cloud batch. This novel index serves as a powerful tool for farmers to objectively assess the health and vigor of their vines within the mapped area of the vineyard. Algorithm 7-5 calls the following functions:

PointCloudQualityEnhancement: Loads a batch of point clouds and enhances their quality. Described in section 7-2.

PointCloudRegistration: This function merges individual point cloud scans captured from successive viewpoints as the robot traverses the vineyard. It leverages the KISS-ICP [22] variant of the well-established Iterative Closest Point (ICP) algorithm. The function implements a tiered ICP approach utilizing three distinct point correspondence distances. This sequential strategy, employing a coarse-to-fine logic, progressively refines the alignment between successive scans. The function's primary output is a unified point cloud map fm encompassing all the input batch's scans, stitched together through the registration process. Additionally, the function provides a list of individual transformations tr. Each transformation corresponds to a specific input point cloud and details the precise positional and rotational adjustments necessary to integrate it into the overall map.

CanopySeparation: This function operates on the fused point cloud map fm generated by the PointCloudRegistration function. It isolates points within the map exhibiting green hues. This targeted color-based segmentation effectively filters out extraneous elements, resulting in a distinct point cloud ic exclusively encompassing the lush green canopy of the vineyard.

FullCanopyEstimation: Reconstructs a mesh m from the ic point cloud acquired by the CanopySeparation function utilizing alpha shapes. By strategically setting a relatively large alpha parameter, the reconstruction excludes all gaps and irregularities.

CanopyGapExtraction: The mesh m produced from the FullCanopyEstimation is sampled and a point cloud fc representing the gapless full canopy is acquired. The initial canopy point cloud ic is subtracted from the full canopy point cloud using distance-based subtraction. The resulting point cloud gc represents the gaps in the canopy.

The algorithm then calculates the CDI by dividing the number of points of the output point cloud from the CanopyGapExtraction gc by the number of points of the full canopy point cloud fc, after downsampling both clouds using the same voxel size. Figure 7-18 illustrates the Canopy Density Assessment process in detail.

ALGORITHM 7-5 CANOPYDENSITYASSESSMENT

Require: Set of point clouds $P = \{p_1, \dots, p_k\}$, v, b, n, e, t, a

Output: CDI

B = PointCloudQualityEnhancement(P, v, b, n, e, t)

fm, tr = PointCloudRegistration(B)

ic = CanopySeparation(fm)

m = FullCanopyEstimation(ic,a)

fc, gc = CanopyGapExtraction(m, ic)

$$CDI = 1 - \frac{NumberOfPoints(gc)}{NumberOfPoints(fc)}$$

output CDI

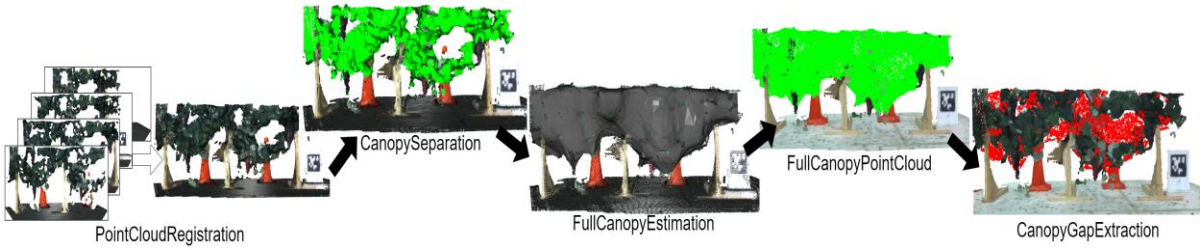


Figure 8-9: Canopy Density Assessment Illustrated.

8.4 Mesh Generation and Filtering

In the context of 3D graphics, a mesh is a collection of vertices connected by edges to form triangles or other polygons. These polygons represent the surface of the object, and they are responsible for defining its shape and appearance. Textures are typically 2D images that contain color information. When a texture is applied to a mesh, the image is stretched or warped to fit the surface of the mesh object. This process is used to create a more realistic and visually appealing appearance by adding color and detail to the 3D mesh without sacrificing a lot of memory or computational resources.

The idea behind the Vinymap reconstruction algorithm is to leverage the concept of mesh texturing in 3D graphics to add valuable detail to an otherwise sparse and rough 3D reconstruction of complex environments such as vineyards. To achieve this, the local map point cloud that we produced in the previous steps must be converted to a dense 3D geometry, a triangle mesh. This process is known as surface reconstruction. There is a plethora of surface reconstruction methods in the literature [147], [148]. Favoring simplicity over elaborate algorithms, two well tested and widely used methodologies have been combined to implement surface reconstruction in this work: The alpha shapes algorithm and the ball pivoting algorithm.

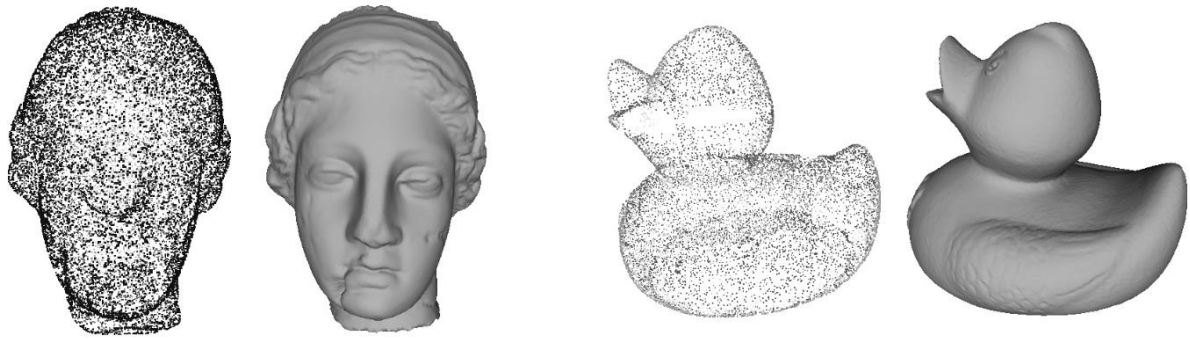


Figure 8-10: Reconstructed mesh objects from dense point clouds [149].

8.4.1 Alpha shapes

The alpha shapes algorithm [150], conceived as a generalization of the convex hull, can be intuitively understood as follows: 3D space can be imagined as a large block of soft, creamy material containing the point cloud points S embedded in it as small, non-deformable, hard pieces. Using a spherical scoop, all portions of the soft material that can be reached without encountering the hard pieces are meticulously carved away. This process may involve excavating internal cavities, as some areas of the soft material may be inaccessible by simply maneuvering the scoop from the outside. Eventually, a piecewise-linear, non-convex object composed of caps, arcs, and points will be left. If all rounded surfaces are then flattened to form triangles and line segments, a representation of the alpha shape of S is produced. The alpha shapes surface reconstruction algorithm can easily produce watertight meshes (meshes without holes or gaps). There is, however, a significant loss in detail when reconstructing with this approach.

8.4.2 The Ball Pivoting Algorithm

The ball pivoting algorithm (BPA) [142] can be intuitively described by imagining a ball of user-specified radius p that rolls on the 3D point cloud. If the ball touches three points without containing any others, then a triangle is formed between these points. The algorithm starts with an initial (seed) triangle, the ball rotates (pivots) around one of its edges, maintaining contact with its endpoints, until it makes contact with a third point, thereby forging a new triangle. This process iterates until all reachable edges have been explored and then commences anew with a fresh seed triangle until all points have been incorporated. The process may be repeated with a larger radius ball to accommodate disparate sampling densities. The BPA has been successfully deployed on datasets comprising millions of points, faithfully replicating actual scans of intricate 3D objects. Its modest memory footprint, efficient execution speed, and high-quality outcomes stand favorably against existing methodologies. The ball pivoting algorithm produces detailed surfaces but is prone to errors related to holes and gaps in the mesh.

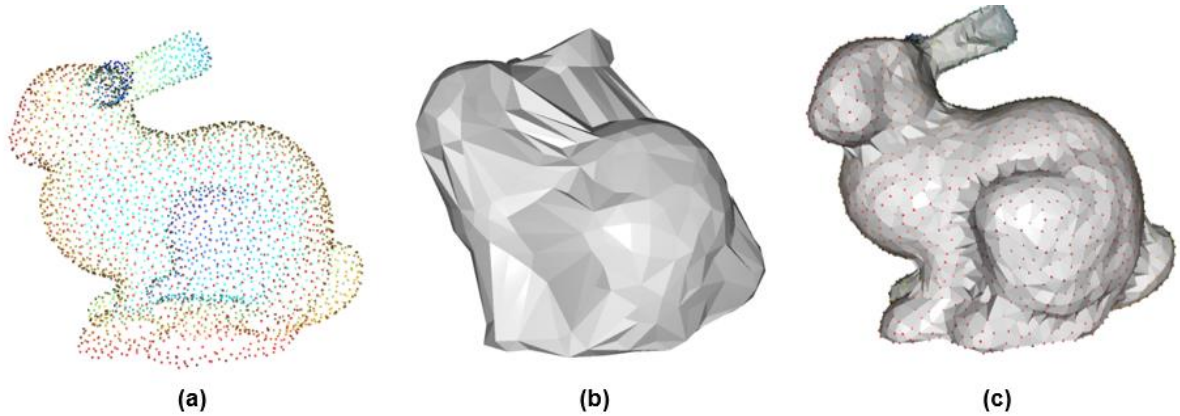


Figure 8-11: (a) a bunny shaped source point cloud. (b) mesh generated with alpha shapes (c) mesh generated with BPA [151].

8.4.3 Chosen methodology and Filtering

To perform surface reconstruction on the complex and unstructured vineyard point cloud, both aforementioned methods were combined. The local map point cloud produced by the quality improvement and registration processes is initially converted into a mesh using BPA. The radius of the pivoting ball chosen is twice as big as the average distance between points in the point cloud. To refine the results, several more, larger radii are used.

The resulting BPA mesh is also filtered using a simple average filter. Thus, a given vertex u_i is given by the average of the adjacent vertices N :

$$\frac{\boldsymbol{u_i} + \sum_{\boldsymbol{n}\in \boldsymbol{N}} \boldsymbol{u_n}}{|\boldsymbol{N}|+\boldsymbol{1}}, \quad \textbf{(87-}$$

This filter is commonly used to denoise meshes. It can be applied to the mesh any given number of times, until the mesh is perfectly smooth. It comes with a non-desirable side effect though. The mesh shrinks after applying the filter iteratively. It was therefore used for a single iteration on the BPA mesh.

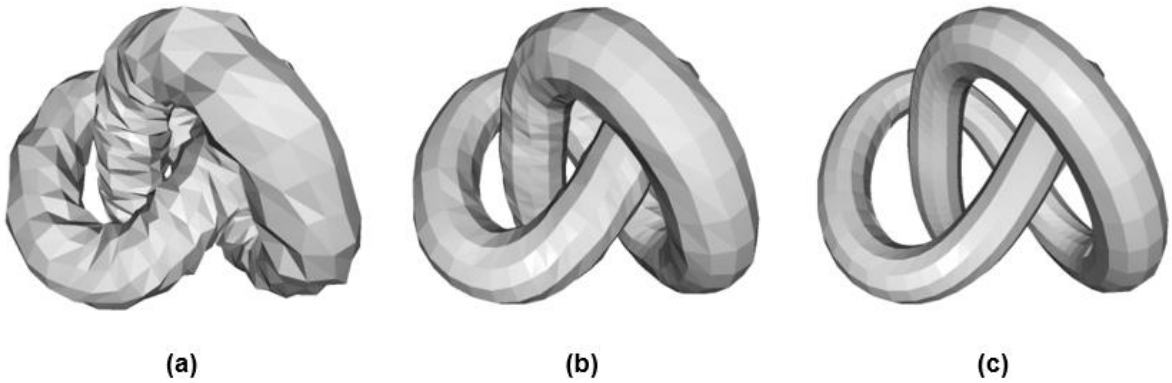


Figure 8-12: (a) noisy mesh (b) mesh after 1 iteration of average filtering (c) mesh after 5 iterations of average filtering [151].

The produced BPA mesh does not completely reconstruct the complex surface of the vineyard vegetation. There are gaps where leaves should be and sparse structures in the point cloud have not been included in the mesh. The alpha shapes algorithm is the perfect tool for filling these gaps. It was applied to the full map point cloud and the resulting Alpha mesh was also filtered with a simple average filter.

To construct the final mesh, the BPA mesh and the Alpha mesh are combined into a single geometry, the final full-map mesh. A function is used to merge vertices of the final mesh that are too close to one another. The vertex position, normal and color of the resulting vertices the average of the vertices. This function helps to close triangle soups caused by the merge, i.e. areas of the mesh where many triangles overlap. Finally, the final full-map mesh is also filtered with an average filter. Filtering the BPA mesh, the alpha mesh and the final mesh separately reduces shrinkage and yields a better result than filtering the final mesh three times. Using other common filtering techniques like the Laplacian filtering or the Taubin filtering was also tested but resulted in deformations in certain parts of the mesh.

8.5 RGB Image Projection and Texture Generation

8.5.1 Vertex Coloring and 2D Textures

The way open3d and most libraries add color to a mesh is by vertex coloring. Vertex coloring refers to coloring a mesh triangle with the average color of the three vertices that define it. That means that the texture resolution (color resolution) of the resulting mesh is limited to the model resolution, meaning that if a mesh has few triangles per unit of volume, it will be colored inaccurately. When using vertex colors, the more triangle-dense a mesh is, the more accurate its coloring.

There have been improvements to the concept of vertex coloring over the years. Using color gradients to seamlessly blend triangle colors and create a more realistic coloring is common among graphics libraries. In addition, extensive research in [152] is focused on extending the concept of vertex coloring where color values are kept on each vertex, by also keeping color values on edges and faces. This approach allows higher texture resolution than model resolution and at the same time it guarantees one-to-one correspondence between the model surface and the color data while also reducing discontinuities.

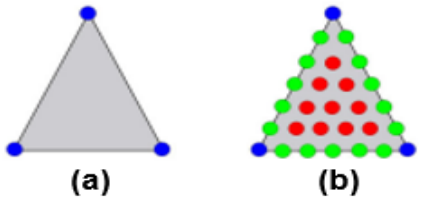


Figure 8-13: (a) With the vertex colors method, only vertex colors (blue) are used to color triangle. (b) With mesh colors, color positions on vertices (blue), edges (green) and faces (red) are all used to determine triangle color [152].

Nevertheless, applying 2D textures to meshes is by far the most used coloring strategy in both literature and commercial graphics applications. Mapping a 2D image to a mesh produces high quality coloring regardless of the model resolution. Even simple meshes with only a handful of vertices can accurately resemble real-life complex structures. As far as color accuracy is concerned, leveraging 2D image textures to color meshes produced from

depth cameras is ideal, as these meshes have a small vertex density. The challenge is accurately mapping 2D image textures to the newly formed full-map mesh.

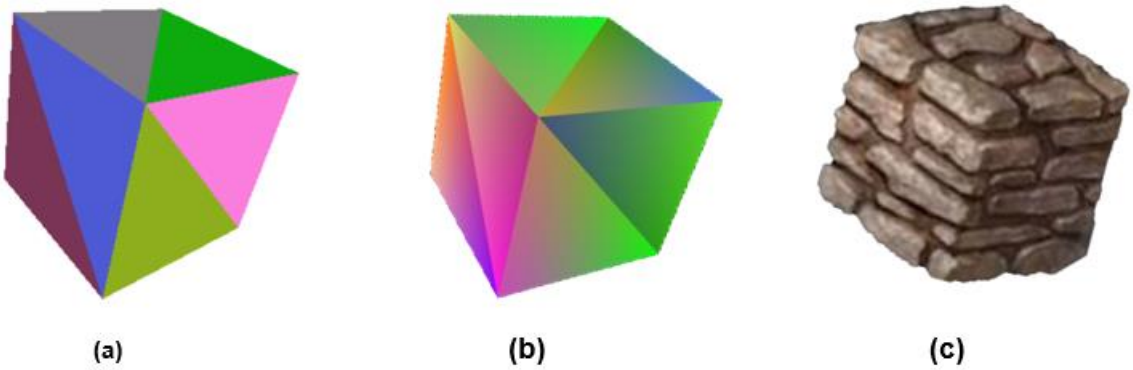


Figure 8-14: (a) Simple cube mesh colored with simple vertex colors (b) cube colored with vertex colors and linear filtering (contouring) (c) cube colored using a 2D texture (image) of a stone wall.

8.5.2 Texture Application

The mathematical solution to the problem of 2D texture application is well formulated and it comes down to computing the so-called UV coordinates of a mesh. In the realm of 3D modeling and computer graphics, UV coordinates and UV maps play a crucial role in seamlessly applying textures to intricate 3D objects and meshes. UV coordinates, also known as texture coordinates, serve as a bridge between the 3D geometry of an object and the 2D texture image that will define its surface appearance. These coordinates, represented by a pair of values (U, V), correspond to specific points on the 3D object's surface, allowing the texture image to be mapped onto the object's form accurately. UV maps, collections of UV coordinates arranged in a grid-like structure, provide a detailed representation of the 3D object's surface, enabling the texture image to be seamlessly applied across its complex geometry. The creation of UV maps involves unwrapping the 3D object's surface, flattening it onto a virtual plane, and assigning UV coordinates to each vertex. The resulting UV map acts as a blueprint for mapping the texture image onto the object's surface, ensuring that the texture adheres to the object's shape and details without distortion or seams. This process is essential for creating visually appealing and realistic 3D models, as it allows for the application of textures that accurately represent the object's surface properties and characteristics.

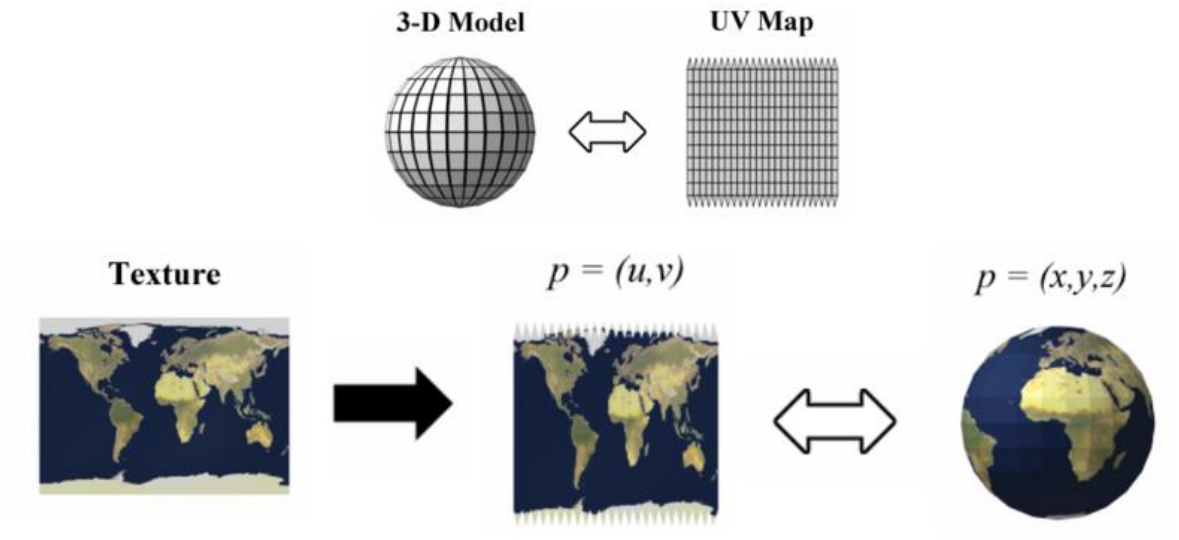


Figure 8-15: Texturing a 3D model of the Earth using a typical sphere UV map.

The problem is how to generate an accurate UV map that matches the complex and intricate vineyard full-map mesh to a 2D image captured with the ZED camera.

8.5.3 Projective Texture Mapping

As described in sections 5 and 6, the ZED camera simultaneously captures and saves point clouds using its stereo vision capabilities and RGB images using its left lens. Thanks to odometry data collected during the process of point cloud registration which is described in 7.2.2, the position of the ZED camera at the time of capturing each point cloud and image is accurately known. Thus, the problem transforms into the task of projecting a 2D RGB image to the part of the mesh that corresponds to the point cloud which was captured at the time of capturing the 2D image. As described in 7.2.2, a bounding polygon is saved for each registered point cloud. So, these polygons can be used to crop the full-map final mesh in pieces and each of these pieces corresponds to a 2D image captured from ZED. Therefore, there is a one-to-one correlation between 2D images captured and pieces of the final mesh. Projecting these 2D images to the pieces of the final mesh and then re-assembling the textured pieces to a final reconstructed mesh is what is solved next.

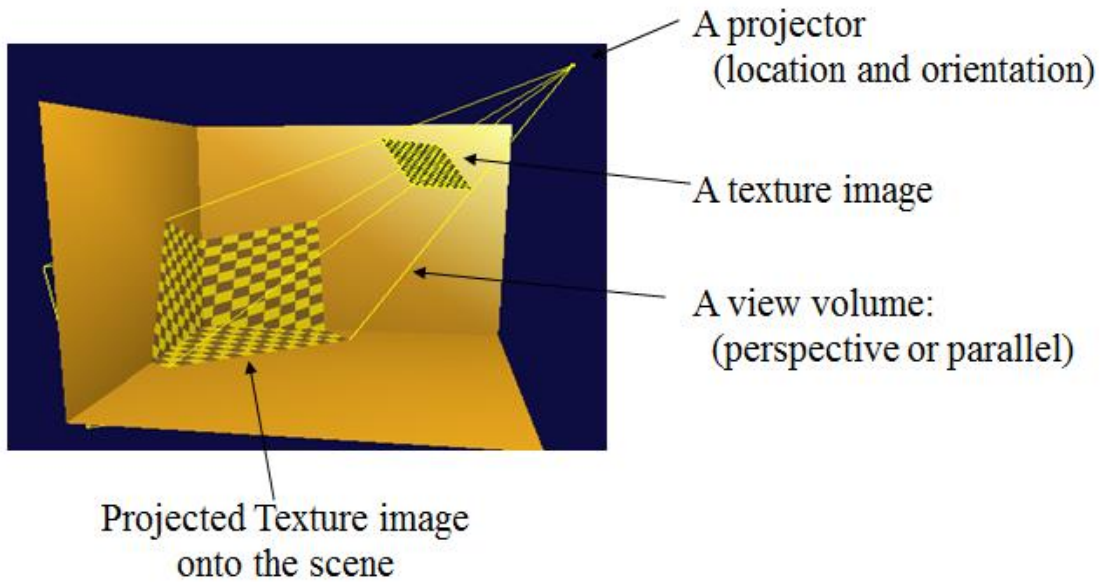


Figure 8-16: Projective Texture Mapping

Let there be a (X,Y,Z) coordinate system in the vineyard mesh's scene where X is width, Y is height and Z is depth. It is initially natural to assume that the UV coordinates of each vertex of the mesh would be its (X,Y) coordinates. So (u,v) = (X,Y). That means that each vertex of the mesh would be matched to the pixel of the corresponding 2D image which has the same (X,Y) coordinates as the vertex. This makes intuitive sense as a vertex of the mesh that is up and to the left of the scene would be assigned a texture pixel which sits up and to the left of the corresponding 2D image. This solution could even be viable if the lenses of the ZED camera captured orthographic images of the world. This is not true, however, and perspective must be taken into consideration.

Due to perspective, objects that are further away from the camera appear smaller than objects that are closer. Intuitively, a division by the depth Z would account for this. Consequently, the uv coordinates would come to be:

$$u = \frac{X}{Z} \tag{7-3}$$

and

$$v = \frac{Y}{Z} \tag{7-4}$$

Nevertheless, this projection model is inaccurate as well and defines an oversimplified camera model with focal length $f = 1$. To understand what focal length is and obtain the camera model which will produce the desirable results, the pinhole camera model must be defined.

The pinhole camera model provides a simplified representation of how light interacts with a camera to form an image. In this model, the camera is represented as a simple box with a tiny hole on one side and a flat image plane on the other. All rays of light are blocked by the walls of the box except those passing through the tiny hole.

When an object is placed in front of the aperture, rays of light emanating from different points of the object pass through the aperture and form an inverted image on the image plane.

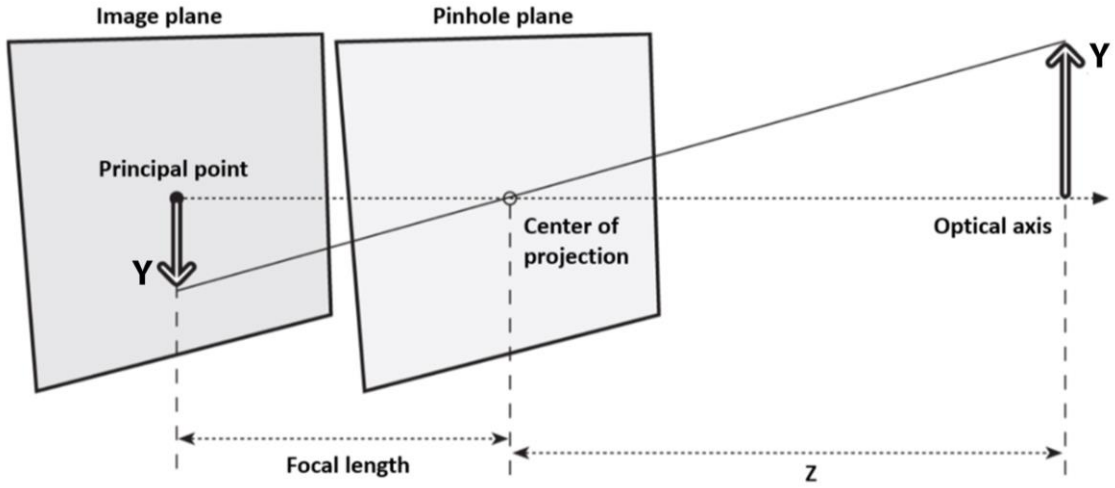


Figure 8-17: The pinhole camera model.

The fundamental parameters of the pinhole camera model are as follows:

- The center of projection is the point where the pinhole aperture is located.
- The focal length is the distance between the center of the projection and the image plane.
- The optical axis is the line perpendicular to the image plane and passing through the center of projection.
- The principal point is defined as the point of intersection between the optical axis and the image plane.

The projection process with the pinhole camera model involves several key factors: As explained, objects farther away from the camera appear smaller in the image. Secondly, due to the light rays intersecting at a single point (the pinhole), the resulting image on the image plane is inverted. Furthermore, the size of the aperture and its distance from the image plane (focal length) determines the field of view of the camera. A larger aperture or a shorter focal length result in a wider field of view, capturing more of the scene. The image size of the distant object is proportional to the focal length.

We can further simplify the projection's description by placing the image plane in front of the pinhole. A point in space $Q = (X, Y, Z)$ is projected on the image plane by tracing the line passing through the point Q and the center of projection. The resulting projection of Q is $q = (x, y, f)$. In this abstraction, the image appears right side up.

The relation between a point $Q = (X, Y, Z)$ in the real space and its projection q on the image plane at the pixel location (x_s, y_s) is represented by the following equations:

$$x_s = f_x \frac{X}{Z} + c_x \tag{7-5}$$

$$y_s = f_y \frac{X}{Z} + c_y \tag{7-6}$$

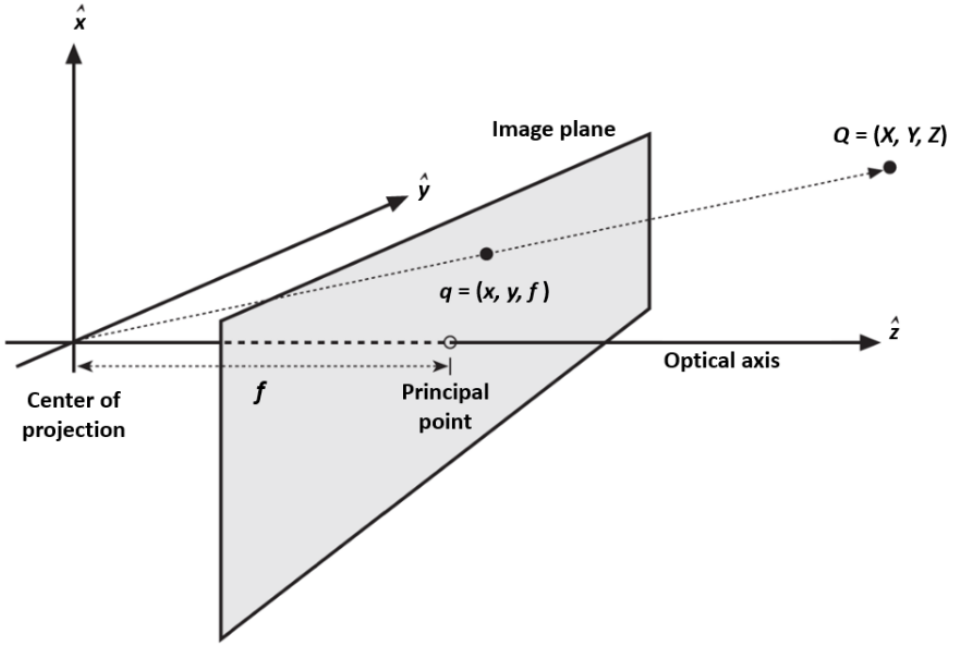


Figure 8-18: A simplified pinhole camera model.

Thus, in the context of texture mapping:

$$u = f_x \frac{X}{Z} + c_x \tag{7-7}$$

$$v = f_y \frac{X}{Z} + c_y \tag{7-8}$$

Where c_x and c_y are two parameters that handle possible misalignment of the principal point with the center of the image and f_x and f_y are essentially separate focal lengths expressed in pixels that are introduced to describe digital cameras with rectangular pixels.

It is customary to express projective transforms using the homogeneous coordinates, hence the equations (7-5) and (7-6) can be written in the following matrix form:

$$\vec{q} = M\vec{Q}, \text{ where}$$

$$\vec{\boldsymbol{q}} = \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix}, \ \boldsymbol{M} = \begin{bmatrix} \boldsymbol{f_x} & \boldsymbol{0} & \boldsymbol{c_x} \\ \boldsymbol{0} & \boldsymbol{f_y} & \boldsymbol{c_y} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \text{ and } \vec{\boldsymbol{Q}} = \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{Y} \\ \boldsymbol{Z} \end{bmatrix} \tag{7-9}$$

The $\vec{M}$ matrix is called the camera intrinsic matrix and it represents the internal parameters of a camera and it allows to project 3D points in the world onto the 2D image plane. The parameters of the camera intrinsic matrix are estimated by performing the camera calibration procedure. The latter is typically performed using a set of known geometric patterns (e.g., chessboard patterns) or objects with precisely measured dimensions, allowing for the estimation of the camera parameters through mathematical algorithms.

Using Eq. (7-7), the 2D images captured with the ZED camera can be accurately projected as textures onto the 3D mesh. This is done by projecting the vertices of the mesh,

which are 3D points in the scene, onto the corresponding 2D image captured by the ZED camera. The resulting 3D mesh is accurately textured and offers detailed visual information.

8.5.4 Triangle Visibility and Ray Casting

The issue that arises with projective texture mapping on 3D meshes is that the texture is applied not only on the visible part of the mesh, but also on the triangles of the mesh that are obscured, lying behind other parts of the mesh. This is not a problem when dealing with meshes that are produced from individual point clouds formed at some point in time by the ZED camera. The problem occurs when a texture is mapped projectively on the final mesh, which is a product of point cloud fusion and mesh filtering. To address this, Open3D's ray casting module is utilized.

Ray casting, a fundamental technique in computer graphics, is a ray tracing-based rendering technique which works by simulating the propagation of light rays through a virtual scene. Its versatility has led to its adoption in a wide range of applications, ranging from early video games and architectural design to medical imaging and scientific visualization.

At its core, ray casting involves tracing imaginary light rays from the virtual camera (observer's position) into the virtual scene. As each ray encounters an object in the scene, it interacts with its surface, determining the distance from the observer. This process is repeated as many times as desired, until the scene is covered. Ray tracing and ray casting are computationally expensive rendering methods. In the last decade, however, hardware acceleration for real-time ray tracing has become standard on new commercial graphics cards, and graphics APIs have followed suit, allowing developers to use hybrid ray tracing and rasterization-based rendering in real-time applications with a lesser hit to frame render times [153].

Using Open3D's ray casting module, rays are cast from the position of the ZED camera towards the final mesh. Every mesh triangle that is hit by any ray is considered a visible triangle while all other triangles are considered obscured. As 2D images are projected as textures onto the final mesh, the triangles of the mesh that are obscured for a given pose of the camera are temporarily removed and no texture is mapped onto them. These triangles receive a texture once the camera proceeds to a position from where they are visible. In the end, the whole mesh is correctly textured.

8.5.5 Photo-Realistic Vineyard Reconstruction

To generate a visually informative and spatially accurate representation of the vineyard, Vinymap utilizes projective texture mapping. The algorithm receives a set of point clouds P and a set of images I as its input. It is important that each point cloud $p_i \in P$ has been captured simultaneously with a corresponding image $I_i \in I$ as the robot moves. This is achieved with ROS's synchronization policy. The algorithm effectively drapes high-resolution RGB images onto meshes which are reconstructed from the raw point cloud data. The meshes are then registered and aligned, resulting in a photo-realistic 3D representation of the vineyard. This process is presented in Algorithm 7-6 and illustrated in Figure 7-18.

ALGORITHM 7-6 PHOTO-REALISTIC VINEYARD RECONSTRUCTION

Require: Set of point clouds $P = \{p_1, \dots, p_k\}$, v, b, n, e, t

Output: v

B = PointCloudQualityEnhancement(P, v, b, n, e, t)

for p_i **in** B:

m_i = SurfaceReconstruction(p_i)

r_i = ProjectiveTextureMapping(m_i, I_i, M)

end for

fm, tr = PointCloudRegistration(B)

v = MeshFusion(tr)

output v

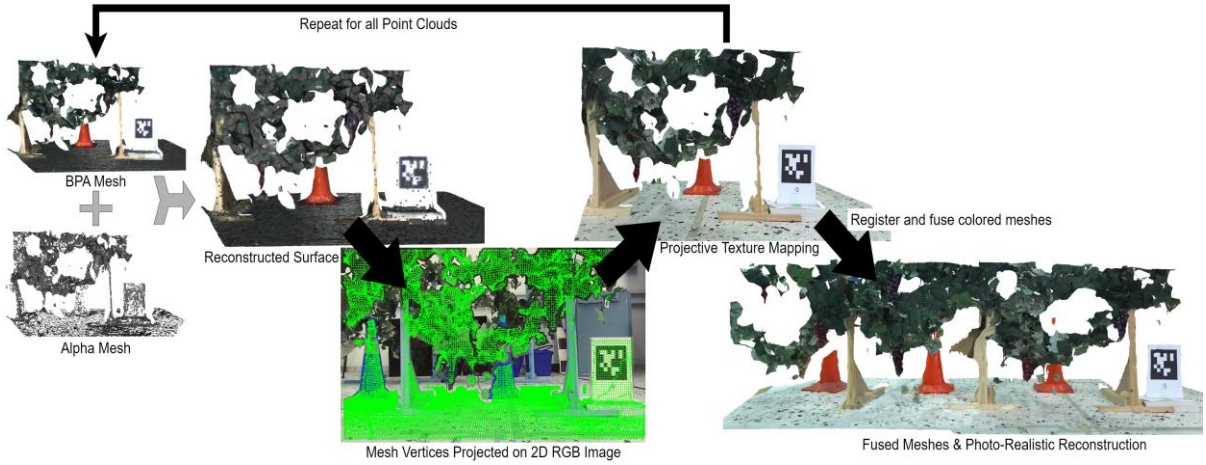


Figure 8-19: Vinymap Photo-Realistic Vineyard Reconstruction Pipeline.

Algorithm 7-6 calls the following functions:

PointCloudQualityEnhancement: Loads a batch of point clouds and enhances their quality. Described in section 7-2.

SurfaceReconstruction: Creates a triangle mesh m that accurately represents a source point cloud p. This is achieved by leveraging the ball pivoting algorithm (BPA) and fusing its results with the alpha shapes reconstruction algorithm.

ProjectiveTextureMapping: This function receives a triangle mesh m_i and an RGB image I_i as input. For each triangle vertex $Q \in m_i$, the function projects it to the 2D image plane of I_i and matches it with a pixel $q \in m_i$ with coordinates $(u,v) \in I_i$ $\vec{q} = M\vec{Q}$, according to Eq. 7-9. The M matrix is the intrinsics camera matrix of the camera that captured I_i. In our case, the ZED X mini camrera. The (u,v) coordinates are used to apply texture on the so-far colorless mesh m_i. Provided that the RGB image I_i and the point cloud p_i were captured simultaneously, the resulting texture application produces a photo-realistic 3D mesh r_i.

PointCloudRegistration: Merges individual point cloud scans captured from successive viewpoints as the RP traverses the vineyard. Described in section 7-2.

MeshFusion: The output transformations *tr* from the PointCloudRegistration are used to identically transform the reconstructed meshes r. This way, the ICP cloud registration is leveraged to accurately register the meshes *r*. The aligned and registered meshes are then fused and the function outputs the result, which is a photo-realistic vineyard mesh *v*.

It should be noted that the point cloud registration and mesh fusion steps can be applied before the projective texture mapping step. In that case, the textures will be applied on the fused mesh and not each triangle mesh separately. To avoid unwanted texture mapping on obscured mesh triangles, raycasting must be utilized. This method usually yields a more

accurate result geometrically but is prone to artifacts caused by projective texture mapping overlap.

8.6 Experimental Results

The Vinymap Quality Assessment and Vineyard Reconstruction framework, as previously described, comprises three core algorithms: the Simple Objective Point Cloud Quality Assessment Algorithm (SOPCQA), the Canopy Density Assessment Algorithm, and the Photo-Realistic Vineyard Reconstruction Algorithm. These algorithms were assessed in controlled laboratory environments utilizing a synthetic vineyard under both artificially lit and naturally lit conditions.

The initial evaluation of the algorithms focused on a subjective comparison of their outputs by human researchers. This approach can be supplemented in future work with the implementation of objective, quantitative metrics to enhance the rigor of the evaluation process.

The real-time viability of the Vinymap framework was evaluated objectively. Resource requirements were measured, and execution times were precisely recorded to assess the framework's suitability for real-time applications.

8.6.1 Simple Objective Point Cloud Quality Assessment Evaluation

The SOPCQA comprises three sub-algorithms: sparsity index calculation, hole detection, and cluster outlier detection. To evaluate their effectiveness, controlled modifications were made to a reference point cloud to introduce specific quality degradations.

Sparsity Index Calculation: This sub-algorithm assesses quality based on point cloud density. To evaluate its effectiveness, controlled downsampling was applied to specific regions of the reference point cloud, resulting in sparser areas. As expected, the SOPCQA higher sparsity index (0.75) to the modified point cloud compared to the reference (0.52), as the sparse area in the modified point cloud is much larger. Figure 7-20 illustrates this.

Hole Detection: This sub-algorithm assesses quality based on the presence of holes (areas completely devoid of points). To evaluate its effectiveness, random holes were introduced within the reference point cloud. As anticipated, the SOPCQA assigned a higher hole index (0.62) to the point cloud with holes compared to the reference (0.48). This indicates lower quality, because of the holes.

Cluster Outlier Detection: This sub-algorithm assesses quality by identifying and removing point clusters too small to be considered valid data (potential noise). To evaluate its effectiveness, small noise clusters were artificially introduced to the reference point cloud. The SOPCQA, as expected, assigned a higher noise_clusters_quantity (24), as well as a larger total_noise_clusters_area (0.92) to the noisy point cloud compared to the reference point cloud which had a noise_clusters_quantity of 17 and a total_noise_clusters_area of 0.58. Table 7-1 documents the results.

Table 8-1: SOPCQA Evaluation

Point Cloud	Sparsity Index	Hole Index	Noise Clusters Quantity	Total Noise Clusters Area	FQI
Reference	0.52	0.742	17	0.58	0.72
Modified	0.75	0.62	24	0.92	0.61

The final SOPCQA algorithm returns a noticeably lower final quality index (FQI) for the intentionally modified point cloud.

Enhanced Point Cloud Evaluation: To assess performance with a more realistic challenge, the SOPCQA's output was compared for a raw point cloud and a quality-enhanced version processed by Vinymap's dedicated enhancement function. The SOPCQA correctly assigned a lower quality index (0.59) to the raw point cloud compared to the enhanced one (0.75)

8.6.2 Canopy Density Assessment Evaluation

To test the performance of our canopy density assessment algorithm, we conducted experiments in both the simulated and the synthetic vineyard setting. The simulation offered a highly controlled environment with almost no canopy density variations. We tested the algorithm on dense, perfect canopies, resulting in very high-density index scores, as expected. The laboratory setting provided a more realistic yet controlled environment. Table 8-2 and Figure 8-20 display the results in detail. We tested two distinct scenarios:

- Moderately dense canopy (control): The algorithm returned a density index within expected range, aligning with the visual assessment of the canopy.
- Sparse canopy: The presence of a gap resulted in a significantly lower density index, validating the algorithm's ability to detect and quantify vegetation irregularities.

Laboratory tests were repeated under natural and artificial lighting. All tests yielded accurate and intuitively useful results, reflecting the algorithm's simplicity and effectiveness in assessing canopy density.

Table 8-2: Canopy Density Index Output

Experiment Type		Subjective Canopy Assessment	Output CDI
Simulated		Dense	0.742
Laboratory	Artificial lighting	Control	0.512
	Natural lighting		0.569
Laboratory	Artificial lighting	Sparse	0.450
	Natural lighting		0.461

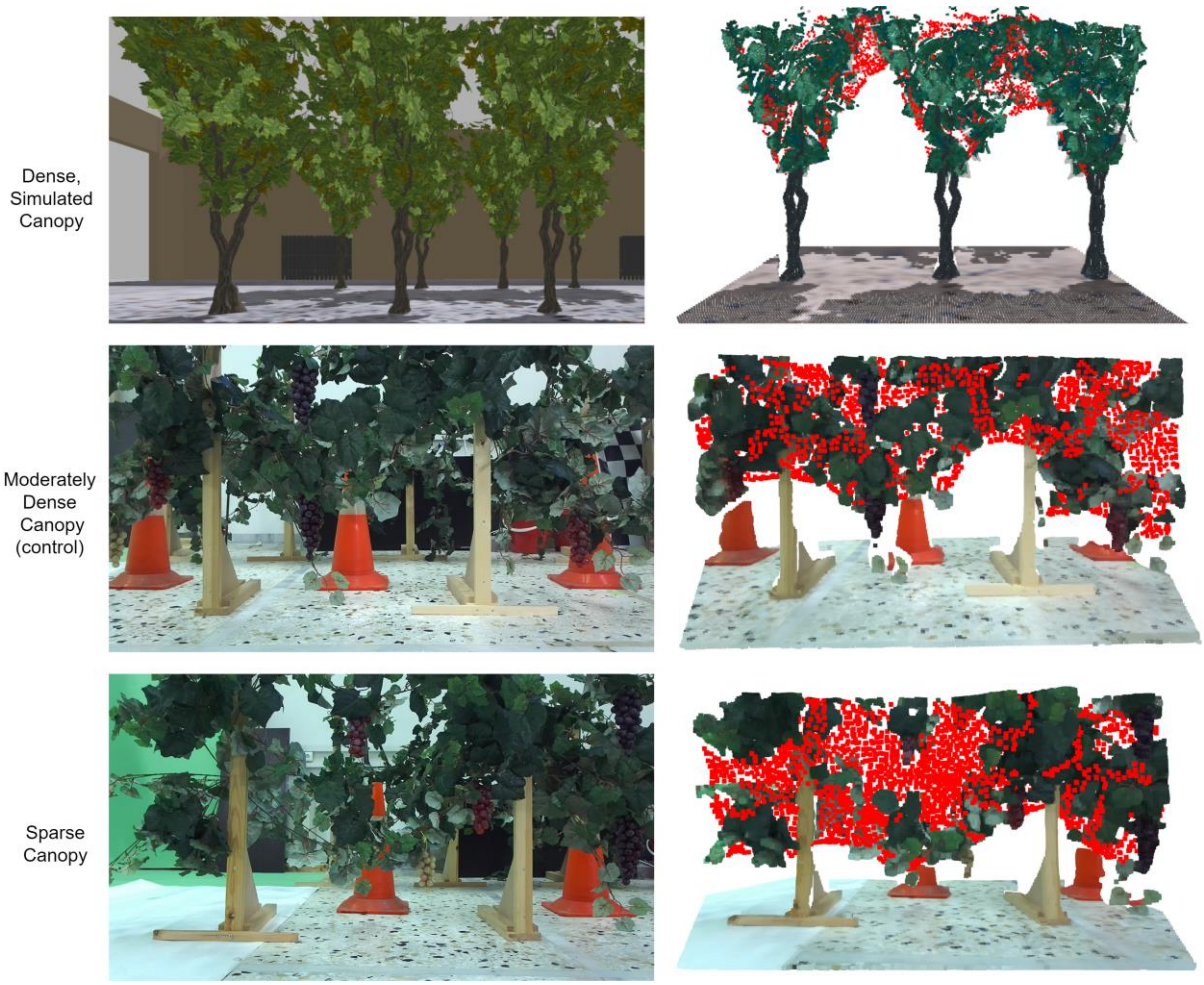


Figure 8-20: Canopy Density Assessment Scenarios.

8.6.3 Reconstruction Quality

Our photo-realistic vineyard reconstruction algorithm is compared to the raw point cloud data and the Stereolabs' ZED Spatial Mapping [72] algorithm in terms of visual fidelity and information richness. Direct comparison with the raw point cloud data readily reveals the difference. While the raw data offers a basic structural representation, our reconstructed mesh delivers a significantly cleaned and visually enhanced depiction of the vineyard.

Compared to the high-quality mesh output of Stereolabs' ZED Spatial Mapping algorithm at its highest settings, our reconstruction emerges as the superior solution. ZED Spatial Mapping uses its vertex coloring scheme to produce a colored mesh. Our projective texture mapping approach generates meshes featuring significantly greater visual detail, enabling farmers to meticulously inspect individual leaves and grapes. This enhanced texture facilitates more informed decision-making in the context of remote precision agriculture. Figure 8-21 depicts a visual comparison.

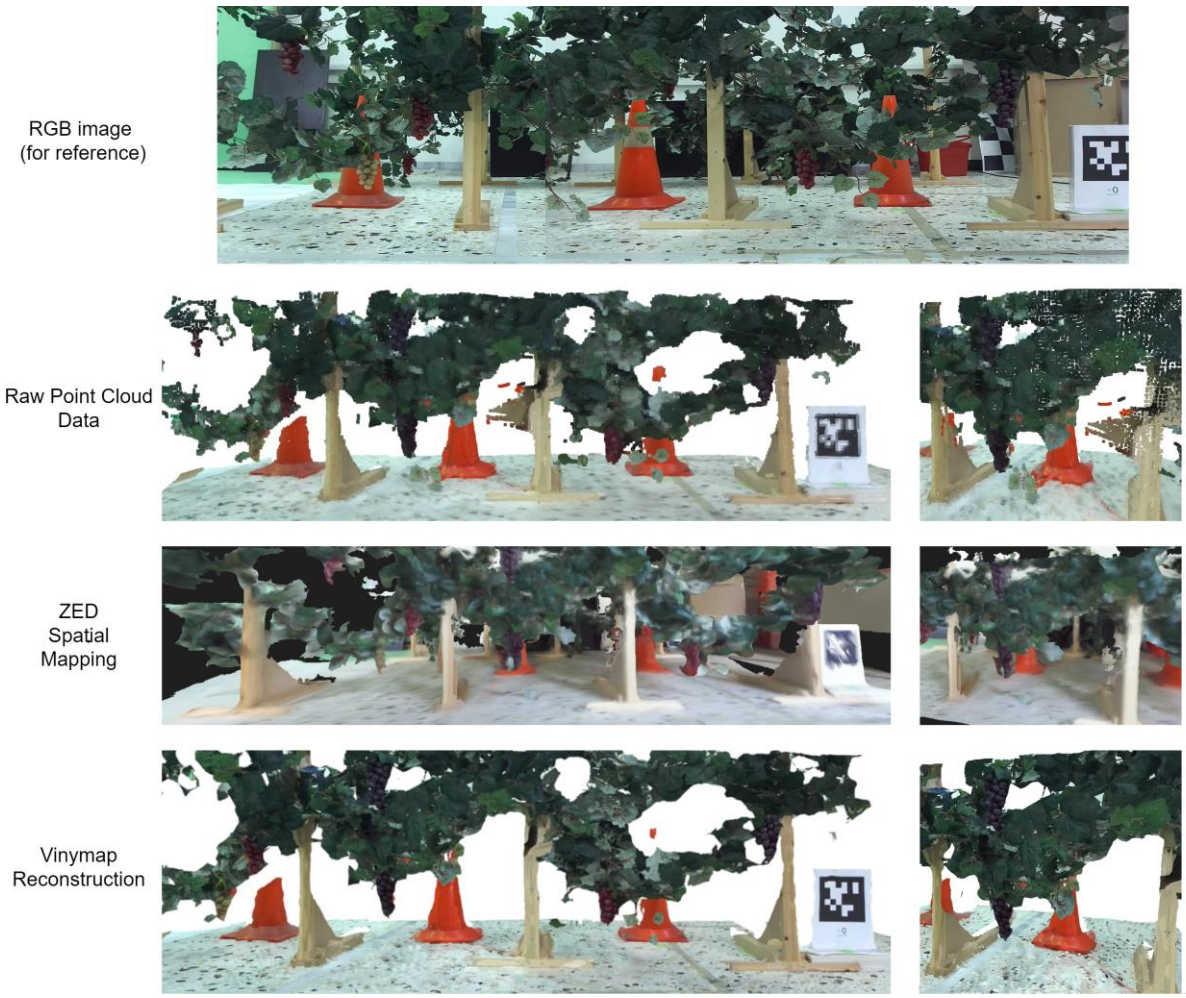


Figure 8-21: 3D Reconstruction Comparison.

8.6.4 Real Time Viability

To evaluate the real-time capabilities of our algorithm, we captured data using ROSbags and SVO files from ZED cameras, allowing for offline review, timing analysis, and benchmarking against state-of-the-art methods. This assessment was conducted on the Nvidia Jetson Orin platform. We compare the Vinymap framework with RTAB-map [12] and ZED Spatial Mapping. Our algorithm achieves real-time performance, performing on-par with the ZED Spatial Mapping algorithm at its high settings. While RTAB-Map exhibits faster execution, it occasionally experiences delay due to loop closure calculations. In contrast, our approach strategically triggers loop closure only upon AprilTag detection, eliminating these delays.

All three algorithms achieve real-time viability. Notably, the desired output quality of each algorithm can influence the robot's operational speed. By carefully balancing robot speed and desired reconstruction detail, farmers can select the optimal solution for their specific needs. In the case of Vinymap, a 20% overlap between captured point clouds is required for effective ICP registration. Assuming the robot moves in a straight line perpendicular to the canopy, Eq. 7-10 yields the maximum speed at which the robot can move to effectively map a vineyard row.

(87-

$$s = fps * fov(1 - overlap),$$

, where s is the maximum speed, fps is the frequency of point cloud and RGB image capturing, fov is the length of the visible canopy at each frame in meters and is dependent on the distance between the robot and the canopy, and overlap is the required overlap in volume percentage between the captured point clouds. With the capability of running with $fps = 2$, a $fov = 0.9\,m,$ and a required overlap of 0.2, Vinymap is suitable for running in real time on robots moving with speeds lower than $1.44\,m/s.$ The RP which our framework was tested on was moving at about $0.12\,m/s$, comfortably within the limit.

9 Conclusions and Future Work

In this thesis, we presented the development of a vineyard inspection and 3D reconstruction framework for mobile robots. This framework encompasses both hardware and software. We investigated mapping algorithms (RTAB-map, Anybotics Elevation Mapping, Zed Spatial Mapping) and evaluated depth camera options including ZED and RealSense depth cameras. Based on this evaluation, we determined that ZED depth cameras offered the most suitable performance for a vineyard setting and opted to use the Zed Spatial Mapping software. To aid in selecting the optimal hardware configuration, we developed a field-of-view and LiDAR resolution analysis tool (FoVaLiRa) using Unity. This tool facilitated the exploration of various configurations and hardware placements on the robot. Through this analysis, we arrived at a configuration consisting of four depth cameras facing outwards in different directions. To further enhance the system's capabilities, we concluded that incorporating a LiDAR sensor, positioned at the front of the robot, would be beneficial.

We then proceeded to evaluate path planning algorithms and methodologies, ultimately selecting a polytopic decomposition planner. This planner is effectively detecting free corridors within the vineyard environment, making it a suitable choice for our application. To implement autonomous navigation for the chosen planner, we leveraged a dual camera visual odometry algorithm developed at the Control Systems Laboratory. This algorithm relies on strategically placed April tags within the vineyard to trigger loop closure, a crucial step to mitigate visual odometry errors. To improve the accuracy of visual odometry localization, we incorporated a visual servoing step that aligns the robot’s camera with the April Tag prior to triggering loop closure optimization.
Given the limitations of existing 3D reconstruction software, particularly the inability of Zed spatial mapping to achieve the desired level of detail and crispness, and the need for vineyard inspection functionality, we developed a novel framework named Vinymap. Vinymap addresses these shortcomings by offering several key functionalities. Firstly, Vinymap achieves real-time viability, enabling fast data collection within vineyard environments. Secondly, Vinymap surpasses the quality of 3D reconstruction obtained through Zed spatial mapping, generating more detailed models. Finally, Vinymap incorporates an inspection solution that includes a canopy density assessment algorithm. This algorithm provides valuable insights for vine-growers, as the canopy density affects the amount of sunlight and air reaching the grapes. Importantly, Vinymap also incorporates a method for objectively assessing the quality of the point cloud data it receives as input. This allows the framework to adapt to varying sensor performance and lighting conditions.

To comprehensively evaluate the performance of the entire system, we conducted experiments on a robotic platform within a synthetic vineyard environment constructed at the CSL lab. This environment allowed for controlled testing of the dual camera visual odometry algorithm, the path planning algorithm, the perception system, and the Vinymap framework, under natural and synthetic lighting conditions.

Looking towards future research directions, the integration of deep learning techniques holds promise for unlocking even more complex vineyard inspection tasks. Deep learning has the potential to enable the framework to detect grapevine diseases or highlight areas of potential concern that warrant further investigation by the vine-grower. Additionally, advancements in autonomous inspection could lead to the development of systems capable

of covering entire vineyards without requiring pre-defined starting and ending points. Furthermore, the creation of benchmark datasets specifically tailored to vineyard reconstruction and inspection tasks would allow for more in-depth evaluation of the Vinymap algorithm's capabilities. We are confident that this framework represents a significant step towards a comprehensive solution for precision viticulture.

10 References

[1] G. Pappalardo, A. Scienza, G. Vindigni, and D. Mario, “Profitability of wine grape growing in the EU member states,” *J. Wine Res.*, vol. 24, Mar. 2013, doi: 10.1080/09571264.2012.724392.

[2] J. M. Bengochea-Guevara, J. Conesa-Muñoz, D. Andújar, and A. Ribeiro, “Merge Fuzzy Visual Servoing and GPS-Based Planning to Obtain a Proper Navigation Behavior for a Small Crop-Inspection Robot,” *Sensors*, vol. 16, no. 3, Art. no. 3, Mar. 2016, doi: 10.3390/s16030276.

[3] A. Costley and R. Christensen, “Landmark Aided GPS-Denied Navigation for Orchards and Vineyards,” in *2020 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, Apr. 2020, pp. 987–995. doi: 10.1109/PLANS46316.2020.9110130.

[4] F. P. Terra, G. R. A. da Rosa, J. J. P. Prado, and P. L. J. Drews-, “A Low-Cost Prototype to Automate Agricultural Sprayers∗,” *IFAC-Pap.*, vol. 53, no. 2, pp. 15835–15840, Jan. 2020, doi: 10.1016/j.ifacol.2020.12.365.

[5] L. Shen *et al.*, “Real-time tracking and counting of grape clusters in the field based on channel pruning with YOLOv5s,” *Comput. Electron. Agric.*, vol. 206, p. 107662, Mar. 2023, doi: 10.1016/j.compag.2023.107662.

[6] S. P. Mohanty, D. P. Hughes, and M. Salathé, “Using Deep Learning for Image-Based Plant Disease Detection,” *Front. Plant Sci.*, vol. 7, 2016, Accessed: Feb. 14, 2024. [Online]. Available: https://www.frontiersin.org/journals/plant-science/articles/10.3389/fpls.2016.01419

[7] S. Kelly *et al.*, “Target-Aware Implicit Mapping for Agricultural Crop Inspection,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, Feb. 2023, pp. 9608–9614. doi: 10.1109/ICRA48891.2023.10160487.

[8] L. Srinivas, A. Bharathy, S. Ramakuri, A. Sethy, and R. Kumar, “An optimized machine learning framework for crop disease detection,” *Multimed. Tools Appl.*, vol. 83, pp. 1–20, May 2023, doi: 10.1007/s11042-023-15446-2.

[9] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, “3D Gaussian Splatting for Real-Time Radiance Field Rendering,” 2023, doi: 10.48550/ARXIV.2308.04079.

[10] T. Thoai, R. J. Ranola, and L. Camacho, “The Importance of Weather Forecasts and Meteorological Information in Adaptation to Climate Change in Agricultural Production: Some Preliminary Findings,” *Philipp. Agric. Sci.*, vol. 101, pp. 377–392, Dec. 2018.

[11] S. E. Spayd, J. M. Tarara, D. L. Mee, and J. C. Ferguson, “Separation of Sunlight and Temperature Effects on the Composition of Vitis vinifera cv. Merlot Berries,” *Am. J. Enol. Vitic.*, vol. 53, no. 3, pp. 171–182, Jan. 2002, doi: 10.5344/ajev.2002.53.3.171.

[12] “RTAB-Map,” RTAB-Map. Accessed: Jul. 29, 2021. [Online]. Available: http://introlab.github.io/rtabmap/

[13] “ANYbotics/elevation_mapping.” ANYbotics, Jan. 27, 2024. Accessed: Jan. 28, 2024. [Online]. Available: https://github.com/ANYbotics/elevation_mapping

[14] K. M. Jatavallabhula, S. Saryazdi, G. Iyer, and L. Paull, “gradSLAM: Automagically differentiable SLAM,” *ArXiv191010672 Cs*, Nov. 2020, Accessed: Dec. 19, 2020. [Online]. Available: http://arxiv.org/abs/1910.10672

[15] Stereolabs, “Spatial Mapping in Computer vision using ZED,” Medium. Accessed: Jan. 30, 2024. [Online]. Available: https://stereolabs.medium.com/spatial-mapping-in-computer-vision-using-zed-69bce43c2e7a

[16] “ZED 2 - AI Stereo Camera.” Accessed: Jul. 10, 2022. [Online]. Available: https://www.stereolabs.com/zed-2/

[17] “csl_legged / dc-vslam-med24 — Bitbucket.” Accessed: Feb. 29, 2024. [Online]. Available: https://bitbucket.org/csl_legged/dc-vslam-med24/src/master/

[18] V. Usenko, L. von Stumberg, A. Pangercic, and D. Cremers, “Real-Time Trajectory Replanning for MAVs using Uniform B-splines and a 3D Circular Buffer,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 215–222. doi: 10.1109/IROS.2017.8202160.

[19] K. S. Narkhede, A. M. Kulkarni, D. A. Thanki, and I. Poulakakis, “A Sequential MPC Approach to Reactive Planning for Bipedal Robots.” arXiv, Apr. 30, 2022. doi: 10.48550/arXiv.2205.00156.

[20] T. Dang, M. Tranzatto, S. Khattak, F. Mascarich, K. Alexis, and M. Hutter, “Graph-based Subterranean Exploration Path Planning using Aerial and Legged Robots,” *J. Field Robot.*, Oct. 2020, doi: 10.1002/rob.21993.

[21] K. A. Mat Said, A. Jambek, and N. Sulaiman, “A study of image processing using morphological opening and closing processes,” *Int. J. Control Theory Appl.*, vol. 9, pp. 15–21, Jan. 2016.

[22] I. Vizzo, T. Guadagnino, B. Mersch, L. Wiesmann, J. Behley, and C. Stachniss, “KISS-ICP: In Defense of Point-to-Point ICP -- Simple, Accurate, and Robust Registration If Done the Right Way,” *IEEE Robot. Autom. Lett.*, vol. 8, no. 2, pp. 1029–1036, Feb. 2023, doi: 10.1109/LRA.2023.3236571.
[23] S. Asaeedi, F. Didehvar, and A. Mohades, “Alpha Convex Hull, a Generalization of Convex Hull,” Sep. 2013.
[24] “Gazebo.” Accessed: Feb. 15, 2024. [Online]. Available: https://gazebosim.org/home
[25] M. A. Abd Mutalib and N. Z. Azlan, “Prototype development of mecanum wheels mobile robot: A review,” *Appl. Res. Smart Technol. ARSTech*, vol. 1, no. 2, pp. 71–82, Nov. 2020, doi: 10.23917/arstech.v1i2.39.
[26] S. Hajjaj and K. Sahari, “Review of agriculture robotics: Practicality and feasibility,” Dec. 2016, pp. 194–198. doi: 10.1109/IRIS.2016.8066090.
[27] T. Utstumo, T. Berge, and J. Gravdahl, “Non-linear Model Predictive Control for constrained robot navigation in row crops,” Mar. 2015. doi: 10.1109/ICIT.2015.7125124.
[28] “VineRobot.” Accessed: Dec. 23, 2023. [Online]. Available: https://www.vinerobot.eu/
[29] A. You *et al.*, “An autonomous robot for pruning modern, planar fruit trees.” arXiv, Jun. 14, 2022. Accessed: Dec. 23, 2023. [Online]. Available: http://arxiv.org/abs/2206.07201
[30] L. Santos, F. Neves Dos Santos, E. Pires, A. Valente, P. Costa, and S. Magalhães, “Path Planning for ground robots in agriculture: a short review,” Apr. 2020, pp. 61–66. doi: 10.1109/ICARSC49921.2020.9096177.
[31] E. Galceran and M. Carreras, “A survey on coverage path planning for robotics,” *Robot. Auton. Syst.*, vol. 61, no. 12, pp. 1258–1276, Dec. 2013, doi: 10.1016/j.robot.2013.09.004.
[32] F. Neves Dos Santos, H. M. P. Sobreira, D. F. B. Campos, R. Morais, A. P. G. M. Moreira, and O. M. S. Contente, “Towards a Reliable Monitoring Robot for Mountain Vineyards,” in *2015 IEEE International Conference on Autonomous Robot Systems and Competitions*, Vila Real: IEEE, Apr. 2015, pp. 37–43. doi: 10.1109/ICARSC.2015.21.
[33] L. Santos, F. N. Santos, S. Magalhaes, P. Costa, and R. Reis, “Path Planning approach with the extraction of Topological Maps from Occupancy Grid Maps in steep slope vineyards,” *2019 IEEE Int. Conf. Auton. Robot Syst. Compet. ICARSC*, pp. 1–7, Apr. 2019, doi: 10.1109/ICARSC.2019.8733630.
[34] C. Xiong, D. Chen, D. Lu, Z. Zeng, and L. Lian, “Path planning of multiple autonomous marine vehicles for adaptive sampling using Voronoi-based ant colony optimization,” *Robot. Auton. Syst.*, vol. 115, pp. 90–103, May 2019, doi: 10.1016/j.robot.2019.02.002.
[35] M. Everett, Y. F. Chen, and J. P. How, “Motion Planning Among Dynamic, Decision-Making Agents with Deep Reinforcement Learning,” 2018, doi: 10.48550/ARXIV.1805.01956.
[36] M. Elhoseny, A. Tharwat, and A. E. Hassanien, “Bezier Curve Based Path Planning in a Dynamic Field using Modified Genetic Algorithm,” *J. Comput. Sci.*, vol. 25, pp. 339–350, Mar. 2018, doi: 10.1016/j.jocs.2017.08.004.
[37] J. A. Placed *et al.*, “A Survey on Active Simultaneous Localization and Mapping: State of the Art and New Frontiers.” arXiv, Feb. 13, 2023. Accessed: Dec. 26, 2023. [Online]. Available: http://arxiv.org/abs/2207.00254
[38] L. Morreale, A. Romanoni, M. Matteucci, and P. D. Milano, “Dense 3D Visual Mapping via Semantic Simplification,” in *2019 International Conference on Robotics and Automation (ICRA)*, Montreal, QC, Canada: IEEE, May 2019, pp. 6891–6897. doi: 10.1109/ICRA.2019.8793256.
[39] Q. Kuang, J. Wu, J. Pan, and B. Zhou, “Real-Time UAV Path Planning for Autonomous Urban Scene Reconstruction,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, Paris, France: IEEE, May 2020, pp. 1156–1162. doi: 10.1109/ICRA40945.2020.9196558.
[40] A. Bacharis, H. J. Nelson, and N. Papanikolopoulos, “View Planning Using Discrete Optimization for 3D Reconstruction of Row Crops,” in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Kyoto, Japan: IEEE, Oct. 2022, pp. 9195–9201. doi: 10.1109/IROS47612.2022.9981209.
[41] C.-Y. Chai, Y.-P. Wu, and S.-L. Tsao, “Deep Depth Fusion for Black, Transparent, Reflective and Texture-Less Objects,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, Paris, France: IEEE, May 2020, pp. 6766–6772. doi: 10.1109/ICRA40945.2020.9196894.
[42] Y. Liu, Q. Yang, Y. Xu, and L. Yang, “Point Cloud Quality Assessment: Dataset Construction and Learning-based No-Reference Metric.” arXiv, Jul. 22, 2022. Accessed: Dec. 29, 2023. [Online]. Available: http://arxiv.org/abs/2012.11895
[43] Z. Zhang, W. Sun, X. Min, T. Wang, W. Lu, and G. Zhai, “No-Reference Quality Assessment for 3D Colored Point Cloud and Mesh Models,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 32, no. 11, pp. 7618–7631, Nov. 2022, doi: 10.1109/TCSVT.2022.3186894.

[44] Z. Zhang *et al.*, “MM-PCQA: Multi-Modal Learning for No-reference Point Cloud Quality Assessment.” arXiv, Apr. 24, 2023. Accessed: Dec. 29, 2023. [Online]. Available: http://arxiv.org/abs/2209.00244
[45] H. Hekmatian, J. Jin, and S. Al-Stouhi, “Conf-Net: Toward High-Confidence Dense 3D Point-Cloud with Error-Map Prediction.” arXiv, Sep. 19, 2019. Accessed: Dec. 29, 2023. [Online]. Available: http://arxiv.org/abs/1907.10148
[46] X. Yuan, H. Chen, and B. Liu, “Point cloud clustering and outlier detection based on spatial neighbor connected region labeling,” *Meas. Control*, vol. 54, no. 5–6, pp. 835–844, May 2021, doi: 10.1177/0020294020919869.
[47] A. Javaheri, C. Brites, F. Pereira, and J. Ascenso, “Joint Geometry and Color Projection-based Point Cloud Quality Metric”.
[48] A. Bacharis, K. D. Polyzos, H. J. Nelson, G. B. Giannakis, and N. Papanikolopoulos, “3D Reconstruction in Noisy Agricultural Environments: A Bayesian Optimization Perspective for View Planning.” arXiv, Sep. 29, 2023. Accessed: Dec. 29, 2023. [Online]. Available: http://arxiv.org/abs/2310.00145
[49] G. Clarkson, S. Luo, and R. Fuentes, “Thermal 3D modelling,” Jul. 2017. doi: 10.22260/ISARC2017/0068.
[50] G. Meynet, J. Digne, and G. Lavoué, “PC-MSDM: A quality metric for 3D point clouds,” in *11th International Conference on Quality of Multimedia Experience (QoMEX 2019)*, Berlin, Germany, Mar. 2019. doi: 10.1109/QoMEX.2019.8743313.
[51] N. Ziadi *et al.*, “Determination of a Critical Nitrogen Dilution Curve for Spring Wheat,” *Agron. J.*, vol. 102, no. 1, pp. 241–250, Jan. 2010, doi: 10.2134/agronj2009.0266.
[52] N. Vigneau, M. Ecarnot, G. Rabatel, and P. Roumet, “Potential of field hyperspectral imaging as a non destructive method to assess leaf nitrogen content in Wheat,” *Field Crops Res.*, vol. 122, no. 1, pp. 25–31, Apr. 2011, doi: 10.1016/j.fcr.2011.02.003.
[53] A. Chlingaryan, S. Sukkarieh, and B. Whelan, “Machine learning approaches for crop yield prediction and nitrogen status estimation in precision agriculture: A review,” *Comput. Electron. Agric.*, vol. 151, pp. 61–69, Aug. 2018, doi: 10.1016/j.compag.2018.05.012.
[54] M. Weyrich, Y. Wang, and M. Scharf, “Quality assessment of row crop plants by using a machine vision system,” presented at the IECON Proceedings (Industrial Electronics Conference), Nov. 2013, pp. 2466–2471. doi: 10.1109/IECON.2013.6699518.
[55] H. Ham, J. Wesley, and H. Hendra, “Computer Vision Based 3D Reconstruction : A Review,” *Int. J. Electr. Comput. Eng. IJECE*, vol. 9, no. 4, p. 2394, Aug. 2019, doi: 10.11591/ijece.v9i4.pp2394-2402.
[56] J. L. Schonberger and J.-M. Frahm, “Structure-from-Motion Revisited,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 4104–4113. doi: 10.1109/CVPR.2016.445.
[57] J. L. Schönberger, E. Zheng, J.-M. Frahm, and M. Pollefeys, “Pixelwise View Selection for Unstructured Multi-View Stereo,” in *Computer Vision – ECCV 2016*, vol. 9907, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., in Lecture Notes in Computer Science, vol. 9907. , Cham: Springer International Publishing, 2016, pp. 501–518. doi: 10.1007/978-3-319-46487-9_31.
[58] V. Yugay, Y. Li, T. Gevers, and M. R. Oswald, “Gaussian-SLAM: Photo-realistic Dense SLAM with Gaussian Splatting,” 2023, doi: 10.48550/ARXIV.2312.10070.
[59] N. Keetha *et al.*, “SplaTAM: Splat, Track & Map 3D Gaussians for Dense RGB-D SLAM,” 2023, doi: 10.48550/ARXIV.2312.02126.
[60] I. Kim and S. Singh, “Bayesian Fusion inspired 3D reconstruction via LiDAR-Stereo Camera Pair,” Sep. 2023.
[61] “Toward real-time and accurate dense 3D mapping of crop fields for combine harvesters using a stereo camera - Haiwen Chen, Jin Chen, Zhuohuai Guan, Yaoming Li, Kai Cheng, Zhihong Cui, Xinxing Zhang, 2023.” Accessed: Dec. 30, 2023. [Online]. Available: https://journals.sagepub.com/doi/full/10.1177/00368504231215974
[62] Y. Li, M. Iida, T. Suyama, M. Suguri, and R. Masuda, “Implementation of deep-learning algorithm for obstacle detection and collision avoidance for robotic harvester,” *Comput. Electron. Agric.*, vol. 174, p. 105499, Jul. 2020, doi: 10.1016/j.compag.2020.105499.
[63] P. K. Panigrahi and S. K. Bisoy, “Localization strategies for autonomous mobile robots: A review,” *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 34, no. 8, Part B, pp. 6019–6039, Sep. 2022, doi: 10.1016/j.jksuci.2021.02.015.
[64] P. Fankhauser, M. Bloesch, and M. Hutter, “Probabilistic Terrain Mapping for Mobile Robots With Uncertain Localization,” *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 3019–3026, Oct. 2018, doi: 10.1109/LRA.2018.2849506.

[65] “https://www.mathworks.com/products/navigation.html.” Accessed: Feb. 29, 2024. [Online]. Available: https://www.mathworks.com/products/navigation.html
[66] Irani, Rousso, and Peleg, “Recovery of ego-motion using image stabilization,” in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94*, Seattle, WA, USA: IEEE Comput. Soc. Press, 1994, pp. 454–460. doi: 10.1109/CVPR.1994.323866.
[67] D. Filliat and J.-A. Meyer, “Map-based navigation in mobile robots - I. A review of localisation strategies”.
[68] “Perception – Legged Robots Team.” Accessed: Jan. 26, 2024. [Online]. Available: https://nereus.mech.ntua.gr/legged/perception/
[69] “AprilTag.” Accessed: Jan. 26, 2024. [Online]. Available: https://april.eecs.umich.edu/software/apriltag
[70] M. Labbé and F. Michaud, “Memory management for real-time appearance-based loop closure detection,” in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2011, pp. 1271–1276. doi: 10.1109/IROS.2011.6094602.
[71] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “OctoMap: an efficient probabilistic 3D mapping framework based on octrees,” *Auton. Robots*, vol. 34, no. 3, pp. 189–206, Apr. 2013, doi: 10.1007/s10514-012-9321-0.
[72] “Spatial Mapping Overview - Stereolabs.” Accessed: Jan. 30, 2024. [Online]. Available: https://www.stereolabs.com/docs/spatial-mapping
[73] D. Scaramuzza and Z. Zhang, “Visual-Inertial Odometry of Aerial Robots,” 2019, doi: 10.48550/ARXIV.1906.03289.
[74] M. Bloesch, M. Burri, S. Omari, M. Hutter, and R. Siegwart, “Iterated extended Kalman filter based visual-inertial odometry using direct photometric feedback,” *Int. J. Robot. Res.*, vol. 36, no. 10, pp. 1053–1072, Sep. 2017, doi: 10.1177/0278364917728574.
[75] M. S. Junayed, A. Sadeghzadeh, M. B. Islam, L.-K. Wong, and T. Aydin, “HiMODE: A Hybrid Monocular Omnidirectional Depth Estimation Model.” arXiv, Apr. 11, 2022. doi: 10.48550/arXiv.2204.05007.
[76] J. R. Sánchez-Ibáñez, C. J. Pérez-del-Pulgar, and A. García-Cerezo, “Path Planning for Autonomous Mobile Robots: A Review,” *Sensors*, vol. 21, no. 23, Art. no. 23, Jan. 2021, doi: 10.3390/s21237898.
[77] D. Ferguson and A. Stentz, “Field D*: An Interpolation-Based Path Planner and Replanner,” in *Robotics Research*, vol. 28, S. Thrun, R. Brooks, and H. Durrant-Whyte, Eds., in Springer Tracts in Advanced Robotics, vol. 28. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 239–253. doi: 10.1007/978-3-540-48113-3_22.
[78] “Accelerate Motion Planning with Deep-Learning-Based Sampler - MATLAB & Simulink.” Accessed: Jan. 31, 2024. [Online]. Available: https://www.mathworks.com/help/nav/ug/accelerate-motion-planning-with-deep-learning-based-sampler.html
[79] Y. Koren and J. Borenstein, “Potential field methods and their inherent limitations for mobile robot navigation,” in *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, Sacramento, CA, USA: IEEE Comput. Soc. Press, 1991, pp. 1398–1404. doi: 10.1109/ROBOT.1991.131810.
[80] R. Menon, T. Zaenker, N. Dengler, and M. Bennewitz, “NBV-SC: Next Best View Planning based on Shape Completion for Fruit Mapping and Reconstruction.” arXiv, Aug. 30, 2023. doi: 10.48550/arXiv.2209.15376.
[81] E. Dunn and J.-M. Frahm, “Next best view planning for active model improvement,” in *Procedings of the British Machine Vision Conference 2009*, London: British Machine Vision Association, 2009, p. 53.1-53.11. doi: 10.5244/C.23.53.
[82] “UAV Toolbox Documentation.” Accessed: Feb. 02, 2024. [Online]. Available: https://www.mathworks.com/help/uav/index.html?s_tid=CRUX_lftnav
[83] A. Viseras, D. Shutin, and L. Merino, “Online information gathering using sampling-based planners and GPs: An information theoretic approach,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 123–130. doi: 10.1109/IROS.2017.8202147.
[84] “Frontier-Based Exploration: Real-World Experiments.” Accessed: Feb. 29, 2024. [Online]. Available: https://robotfrontier.com/frontier/real.html
[85] C. de Boor, “Splines as linear combinations of B-splines,” *Approx. Theory II*, Jan. 1976.
[86] “File:Parametic Cubic Spline.svg - Wikipedia.” Accessed: Feb. 29, 2024. [Online]. Available: https://commons.wikimedia.org/wiki/File:Parametic_Cubic_Spline.svg

[87] M. G. COX, “The Numerical Evaluation of B-Splines*,” *IMA J. Appl. Math.*, vol. 10, no. 2, pp. 134–149, Oct. 1972, doi: 10.1093/imamat/10.2.134.
[88] R. Deits and R. Tedrake, “Computing Large Convex Regions of Obstacle-Free Space Through Semidefinite Programming,” in *Algorithmic Foundations of Robotics XI*, vol. 107, H. L. Akin, N. M. Amato, V. Isler, and A. F. Van Der Stappen, Eds., in Springer Tracts in Advanced Robotics, vol. 107. , Cham: Springer International Publishing, 2015, pp. 109–124. doi: 10.1007/978-3-319-16595-0_7.
[89] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, “Review on model predictive control: an engineering perspective,” *Int. J. Adv. Manuf. Technol.*, vol. 117, no. 5, pp. 1327–1349, Nov. 2021, doi: 10.1007/s00170-021-07682-3.
[90] M. Shan, J. S. Berrio, S. Worrall, and E. Nebot, “Probabilistic Egocentric Motion Correction of Lidar Point Cloud and Projection to Camera Images for Moving Platforms.” arXiv, Mar. 09, 2020. doi: 10.48550/arXiv.2003.03954.
[91] I. K. Alam Bhuiyan, *LiDAR Sensor for Autonomous Vehicle*. 2017. doi: 10.13140/RG.2.2.16982.34887/1.
[92] J. Lemmetti, N. Sorri, I. Kallioniemi, P. Melanen, and P. Uusimaa, “Long-range all-solid-state flash LiDAR sensor for autonomous driving,” Mar. 2021, p. 22. doi: 10.1117/12.2578769.
[93] H. Yoo *et al.*, “MEMS-based lidar for autonomous driving,” *E Elektrotechnik Informationstechnik*, Jul. 2018, doi: 10.1007/s00502-018-0635-2.
[94] D. Wang, C. Watkins, and H. Xie, “MEMS Mirrors for LiDAR: A Review,” *Micromachines*, vol. 11, no. 5, Art. no. 5, May 2020, doi: 10.3390/mi11050456.
[95] “Configuring Stereo Depth — DepthAI documentation | Luxonis.” Accessed: Feb. 21, 2024. [Online]. Available: https://docs.luxonis.com/projects/api/en/latest/tutorials/configuring-stereo-depth/
[96] “Object Perception — CACAO@HOME Robot documentation.” Accessed: Feb. 29, 2024. [Online]. Available: https://gesture-detection-with-ros2.readthedocs.io/en/latest/perception/docs/object_perception.html
[97] Ľ. Kovanič, B. Topitzer, P. Peťovský, P. Blišťan, M. B. Gergeľová, and M. Blišťanová, “Review of Photogrammetric and Lidar Applications of UAV,” *Appl. Sci.*, vol. 13, no. 11, Art. no. 11, Jan. 2023, doi: 10.3390/app13116732.
[98] “What are CUDA Cores?,” Trusted Reviews. Accessed: Feb. 23, 2024. [Online]. Available: https://www.trustedreviews.com/explainer/what-are-cuda-cores-4226433
[99] “Understanding Tensor Cores,” Paperspace Blog. Accessed: Feb. 23, 2024. [Online]. Available: https://blog.paperspace.com/understanding-tensor-cores/
[100] “What Is a Teraflop? What to Know About a GPU’s Performance,” Digital Trends. Accessed: Feb. 23, 2024. [Online]. Available: https://www.digitaltrends.com/computing/what-is-a-teraflop/
[101] “What is TOPS of Tx2 board? - Jetson & Embedded Systems / Jetson TX2,” NVIDIA Developer Forums. Accessed: Feb. 23, 2024. [Online]. Available: https://forums.developer.nvidia.com/t/what-is-tops-of-tx2-board/117375
[102] “Jetson AGX Orin Developer Kit User Guide,” NVIDIA Developer. Accessed: Feb. 15, 2024. [Online]. Available: https://developer.nvidia.com/embedded/learn/jetson-agx-orin-devkit-user-guide/index.html
[103] “Dev Board,” Coral. Accessed: Feb. 23, 2024. [Online]. Available: https://coral.ai/products/dev-board/
[104] “NVIDIA Jetson Xavier Series,” NVIDIA. Accessed: Feb. 23, 2024. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/
[105] “NVIDIA GeForce RTX 3070 Family,” NVIDIA. Accessed: Feb. 23, 2024. [Online]. Available: https://www.nvidia.com/en-eu/geforce/graphics-cards/30-series/rtx-3070-3070ti/
[106] “Minisforum EU,” Minisforum EU. Accessed: Feb. 23, 2024. [Online]. Available: https://store.minisforum.de/
[107] “Search - Intel.com,” Intel. Accessed: Feb. 23, 2024. [Online]. Available: https://www.intel.com/content/www/us/en/search.html
[108] “MAGNUS EN173070C (Barebone),” ZOTAC. Accessed: Feb. 23, 2024. [Online]. Available: https://www.zotac.com/us/product/mini_pcs/magnus-en173070c-barebone
[109] U. Technologies, “Maximize Multiplatform Game Development | Unity.” Accessed: Feb. 26, 2024. [Online]. Available: https://unity.com/solutions/multiplatform
[110] “Unity Technologies,” *Wikipedia*. Apr. 17, 2022. Accessed: May 02, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Unity_Technologies&oldid=1083128423
[111] U. Technologies, “Robotics Simulation | Unity.” Accessed: Feb. 26, 2024. [Online]. Available: https://unity.com/solutions/automotive-transportation-manufacturing/robotics

[112] *How To Animate In Unity 3D*, (2019). Accessed: Aug. 25, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=sgHicuJAu3g
[113] D. M. M. Sathik, “Ray Casting for 3D Rendering – A Review”.
[114] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan, “Ray Tracing on Programmable Graphics Hardware”.
[115] E. Vasiou, K. Shkurko, I. Mallett, E. Brunvand, and C. Yuksel, “A detailed study of ray tracing performance: render time and energy cost,” *Vis. Comput.*, vol. 34, Jun. 2018, doi: 10.1007/s00371-018-1532-8.
[116] “Rotation matrix,” *Wikipedia*. Apr. 22, 2022. Accessed: Jun. 17, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Rotation_matrix&oldid=1084060907
[117] *The Robotics Optimized Velarray M1600 Lidar Sensor from Velodyne Lidar*, (2022). Accessed: Jul. 19, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=jo4Iza4b0LI
[118] stereolabs, “zed2-camera-datasheet.” [Online]. Available: https://www.stereolabs.com/assets/datasheets/zed2-camera-datasheet.pdf
[119] J. Maynard, “Lidar and Power Consumption for EVs,” Velodyne Lidar. Accessed: Feb. 27, 2024. [Online]. Available: https://velodynelidar.com/blog/lidar-and-power-consumption-electric-vehicles/
[120] M. Brandão, R. Figueiredo, K. Takagi, A. Bernardino, K. Hashimoto, and A. Takanishi, “Placing and scheduling many depth sensors for wide coverage and efficient mapping in versatile legged robots,” *Int. J. Robot. Res.*, vol. 39, no. 4, pp. 431–460, Mar. 2020, doi: 10.1177/0278364919891776.
[121] “Depth Camera D435i,” Intel® RealSense™ Depth and Tracking Cameras. Accessed: Sep. 28, 2021. [Online]. Available: https://www.intelrealsense.com/depth-camera-d435i/
[122] “Amp consumption camera d435,” Intel RealSense Help Center. Accessed: Jul. 19, 2022. [Online]. Available: http://support.intelrealsense.com/hc/en-us/community/posts/360051256734-Amp-consumption-camera-d435
[123] “About Spot — Spot 3.1.2.1 documentation.” Accessed: Jul. 14, 2022. [Online]. Available: https://dev.bostondynamics.com/docs/concepts/about_spot
[124] “Ultra Puck Surround View Lidar Sensor,” Velodyne Lidar. Accessed: Jul. 16, 2022. [Online]. Available: https://velodynelidar.com/products/ultra-puck/
[125] lidar velodyne, “Ultra-Puck_Datasheet.” [Online]. Available: https://velodynelidar.com/wp-content/uploads/2019/12/63-9378_Rev-F_Ultra-Puck_Datasheet_Web.pdf
[126] “ANYmal C – Autonomous Legged Robot,” ANYbotics. Accessed: Jul. 16, 2022. [Online]. Available: https://www.anybotics.com/anymal-legged-robot/
[127] “Xiaomi CyberDog 2.” Accessed: Feb. 27, 2024. [Online]. Available: https://www.giztop.com/xiaomi-cyberdog-2.html
[128] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, Sendai, Japan: IEEE, 2004, pp. 2149–2154. doi: 10.1109/IROS.2004.1389727.
[129] “rviz - ROS Wiki.” Accessed: Jan. 21, 2024. [Online]. Available: http://wiki.ros.org/rviz
[130] “Foxglove - Visualizing and debugging your robotics data - Foxglove.” Accessed: Jan. 21, 2024. [Online]. Available: https://foxglove.dev/
[131] “tracking_pid - ROS Wiki.” Accessed: Jan. 21, 2024. [Online]. Available: http://wiki.ros.org/tracking_pid
[132] “apriltag_ros - ROS Wiki.” Accessed: Jan. 21, 2024. [Online]. Available: http://wiki.ros.org/apriltag_ros
[133] “Impulse X2E Motion Capture – PhaseSpace Motion Capture.” Accessed: Mar. 02, 2024. [Online]. Available: https://www.phasespace.com/x2e-motion-capture/
[134] “DJI Air 2S - All In One - DJI.” Accessed: Jan. 24, 2024. [Online]. Available: https://www.dji.com/gr/air-2s
[135] “Pololu - RoboClaw 2x30A Motor Controller (V5E).” Accessed: Jan. 24, 2024. [Online]. Available: https://www.pololu.com/product/3286
[136] “Buy a Raspberry Pi 2 Model B – Raspberry Pi.” Accessed: Jan. 24, 2024. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-2-model-b/
[137] “Wild Scorpion 6S 22.2v 4200mAh 60C. Hobby Hangar.” Accessed: Jan. 24, 2024. [Online]. Available: https://www.hobbyhangar.co.nz/wild-scorpion-6s-222v-4200mah-60c
[138] “ZED X Mini Stereo Camera | Stereolabs.” Accessed: Jan. 24, 2024. [Online]. Available: https://store.stereolabs.com/en-eu/products/zed-x-mini-stereo-camera?variant=42720409780380

[139] “XP-1 MICRO-START Jump-Starter – Antigravity Batteries.” Accessed: Jan. 24, 2024. [Online]. Available: https://antigravitybatteries.com/products/micro-starts/xp-1/
[140] H. Su, Q. Liu, Z. Duanmu, W. Liu, and Z. Wang, “Perceptual Quality Assessment of Colored 3D Point Clouds.” arXiv, Nov. 09, 2021. Accessed: Dec. 31, 2023. [Online]. Available: http://arxiv.org/abs/2111.05474
[141] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975, doi: 10.1145/361002.361007.
[142] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, “The ball-pivoting algorithm for surface reconstruction,” *IEEE Trans. Vis. Comput. Graph.*, vol. 5, no. 4, pp. 349–359, Oct. 1999, doi: 10.1109/2945.817351.
[143] X. Huang, G. Mei, J. Zhang, and R. Abbas, “A comprehensive survey on point cloud registration.” arXiv, Mar. 05, 2021. Accessed: Jan. 06, 2024. [Online]. Available: http://arxiv.org/abs/2103.02690
[144] Q. Zhao, X. Gao, J. Li, and L. Luo, “Optimization Algorithm for Point Cloud Quality Enhancement Based on Statistical Filtering,” *J. Sens.*, vol. 2021, pp. 1–10, Dec. 2021, doi: 10.1155/2021/7325600.
[145] P. Li, R. Wang, Y. Wang, and W. Tao, “Evaluation of the ICP Algorithm in 3D Point Cloud Registration,” *IEEE Access*, vol. 8, pp. 68030–68048, 2020, doi: 10.1109/ACCESS.2020.2986470.
[146] “Figure 1: Point-to-plane error between two surfaces.,” ResearchGate. Accessed: Feb. 29, 2024. [Online]. Available: https://www.researchgate.net/figure/Point-to-plane-error-between-two-surfaces_fig1_228571031
[147] P.-A. Langlois, A. Boulch, and R. Marlet, “Surface Reconstruction from 3D Line Segments,” in *2019 International Conference on 3D Vision (3DV)*, Sep. 2019, pp. 553–563. doi: 10.1109/3DV.2019.00067.
[148] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang, “NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction.” arXiv, Feb. 01, 2023. Accessed: Jan. 07, 2024. [Online]. Available: http://arxiv.org/abs/2106.10689
[149] “POINT CLOUD.” Accessed: Feb. 29, 2024. [Online]. Available: https://elmoatazbill.users.greyc.fr/point_cloud/index.html
[150] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel, “On the shape of a set of points in the plane,” *IEEE Trans. Inf. Theory*, vol. 29, no. 4, pp. 551–559, Jul. 1983, doi: 10.1109/TIT.1983.1056714.
[151] “Surface Reconstruction — Open3D latest (664eff5) documentation.” Accessed: Feb. 29, 2024. [Online]. Available: https://www.open3d.org/docs/latest/tutorial/Advanced/surface_reconstruction.html
[152] C. Yuksel, J. Keyser, and D. H. House, “Mesh colors,” *ACM Trans. Graph.*, vol. 29, no. 2, pp. 1–11, Mar. 2010, doi: 10.1145/1731047.1731053.
[153] Y. Deng, Y. Ni, Z. Li, S. Mu, and W. Zhang, “Toward Real-Time Ray Tracing: A Survey on Hardware Acceleration and Microarchitecture Techniques,” *ACM Comput. Surv.*, vol. 50, pp. 1–41, Aug. 2017, doi: 10.1145/3104067.