UNIVERSITY OF HONG KONG

FINAL YEAR PROJECT FINAL REPORT

Intelligent Robot Design and Implementation

Author:
LEUNG Cham Chung, Terry

Supervisors:
Dr. Tat Wing CHIM
Mr. David LEE
Examiner:
Dr. Chun Kit CHUI

Department of Computer Science
Faculty of Engineering

April 27, 2024

Abstract

The project explores the possibilities of deploying quadrupedal robot for search and rescue (SAR) operations. Given the high mobilities of quadrupedal robot in rough terrain, it is believed that the deployment of quadrupedal robot in disaster fields, the efficiency and safety of search and rescue operations can be significantly improved. To validate our hypothesis, a quadrupedal robot prototype is assembled using 3D-printed materails and off the shelf components. The key focus of the project is the potential capability to perform intelligent behaviours to assist search and rescue missions. For example, allowing the quadrupedal robot to explore unvisisted environment while reconstructing the 3D environment. Given the modularity nature of the software suite implemented, it is believed that the high-level interaction package can be adapted to a variety of production-ready robots impacting the research and innovation of robotics in the of SAR.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

IMU	**I**nertial **M**easurement **U**unit
IR	**I**nfrared
IK	**I**nverse **K**inematics
CNN	**C**onvolutional **N**eural **N**etwork
URDF	**U**nified **R**obot **D**escription **F**ormat
SLAM	**S**imultaneous **L**ocalization **A**and **M**apping
LiDAR	**Li**ght **D**etection **A**and **R**anging

Chapter 1

Introduction

1.1 Background

In disaster events, search and rescue operations have always depended on human involvement heavily. It is believed that the main reason behind such a phenomenon is the lack of appropriate machines and tools. Besides, with the risk of building collapsing in the event of earthquakes or landslides, it might not be appropriate for search and rescue personnel to explore the unknown environment.

Commonly seen wheel robots are highly efficient in everyday life, where navigation environment are often smooth and free with obstacles. However, in natural disasters, impacted regions are often unnavigable for conventional wheeled robots. Therefore, quadrupedal robots, which demonstrates high mobilities in rough terrains, would be a optimal solution to execute the search and rescue tasks.

1.2 Objectives

The objective of the project is to implement a high-level interaction package that performs intelligent tasks (e.g., exploration) and interact with operator facilitating search and rescue tasks.

First, the prototype quadrupedal robot would be assembled using 3D-printed material with off the shelf electronics components. Then, by deploying an appropriate control framework, the quadrupedal robot should be able to navigate based on any movement command. Second, upon starting search and rescue mission, the quadrupedal robot should be able to explore unknown environment while reconstructing a 3D representation of the environment.

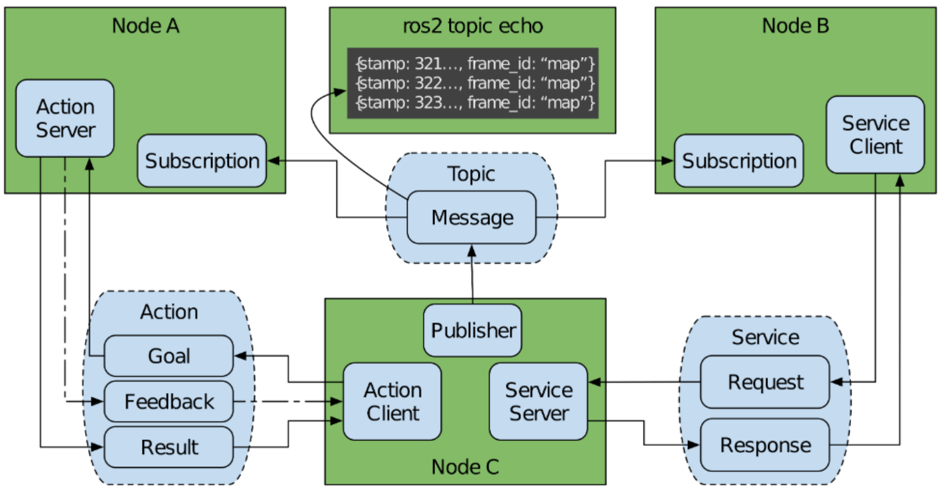
In addition, if interested objects are detected, the quadrupedal robot should be dynamically follow the object of interest. Apart from the quadrupedal robot's capabilities, a web-based dashboard should be implemented to enable visualization of live information regarding the quadrupedal robot (e.g., live camera feed, current position) and alert the operator for the detection of target objects.

Chapter 2

Literature Review

2.1 Robot Operating System (ROS)

Robot Operating System (ROS) is an open-source robot framework that provides a structured communications layer between computing nodes [1]. In the field of robotics, numerous hardware configuration can be used to assemble a purpose-built robot depsite most modern robots often use off the shelf components to provide specific functionalities. The characteristics fo varying hardware makes code reuse challenging [1]. The challenge presented hinders the rapid prototyping and integration effort.



```
{stamp: 321..., frame_id: "map"}
{stamp: 322..., frame_id: "map"}
{stamp: 323..., frame_id: "map"}
```

Fig. 1: ROS 2 node interfaces: topics, services, and actions.

Figure 2.1: ROS 2 node interfaces

To resolve these challenges, ROS introduces a modular and adaptable architecture running on a peer-to-peer network as shown in Fig. 2.1 [1]. Each purpose-built software would be run as an independent process known as node, communicating with a well-defined interfaces (e.g., topics, services, and action) [1] [2]. The standardized messaging protocol enables end packages to be implemented in a variety of programming languages inlcuding C++, Python, Octave, and LISP [1].

In addition, ROS 2 offers a comprehensive software ecosystem including the mentioned middleware, algorithms, and developer tools [2]. For instance, commonly used algorithms (e.g., perception, SLAM) can be readily deployed by aligning the configuration with the hardware configuration. Meanwhile, the standardized developer tools allows formal system analysis and debugging processes across the field [2].

2.2 Hierarchical Control Algorithm for Quadrupedal Locomotion

This subsection will cover the theories behind the hierarchical control algorithm for quadrupedal locomotion. The discussion presented in this section formed the basis of the open-source development framework for quadrupedal robots, CHAMP [3].

Most of the widely studied and deployed land-based robots have uncomplicated dynamic systems and control algorithms. For instance, wheeled robots can change their angular velocity by varying the relative rotational frequency. However, quadrupedal robots typically have 12 degrees of freedom. Therefore, they are often controlled by a complicated control algorithm involving leg trajectory generation and gait pattern modulation. Therefore, it is essential to have an efficient hierarchical control algorithm for quadrupedal locomotion which will be discussed below.

2.2.1 Hybrid Dynamic System

The dynamics of quadrupedal robots can be described by a hybrid dynamic system, which exhibits both continuous and discrete dynamic behavior. A classic hybrid dynamic system can be described by state machines and differential equations simultaneously[4].

To model the dynamics of quadruped robots, each of their legs would be described as an isolated hybrid dynamic system. First, a state machine consisting of swing (flight) phase and stance phase would be used to describe discrete behavior. Then, touch down and lift off events would be detected by sensing abrupt changes in ground reaction forces or foot sensor and hence state transition would be triggered. For continuous behavior, differential equations taking one's velocity and acceleration into account would be derived.

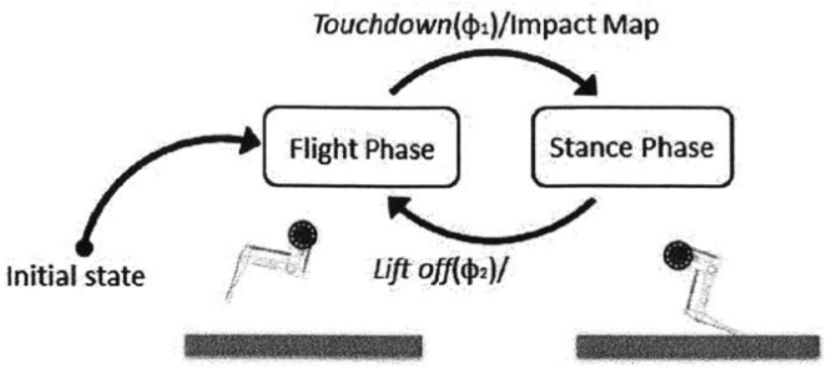


Figure 2.2: Finite state machine to model dynamics of legged mechanism

2.2.2 Constrained Equations of Motion

To realize the hybrid dynamic system mentioned above, multi-link rigid body equations of motion would be derived through Lagrangian mechanics [4]. Then, constrained equations of motion would preciously describe the posture and position of the quadrupedal robot in a controlled environment [4].

In classical mechanics, Lagrangian mechanics is a formulation that describes a dynamic system using the energies in the system (e.g. kinetic energy, elastic potential energy, gravitational potential energy) while Newtonian mechanics describes a dynamic system using the forces in the system (e.g. ground reaction force, gravitational force). Compared to Newtonian mechanics, Lagrangian mechanics offers a systematic and efficient method to solve general mechanical problems allowing equations of motion to be derived in an algorithmic approach.

To derive the equations of motion, the generalized coordinates of the quadrupedal robot would be represented in a vector q

$$q := [q_{pitch}, q_{1,FR}, q_{2,FR}, \dots, q_{1,BL}, q_{2,BL}, x, y]^T \in R^{11} \tag{2.1}$$

Algorithm 1: Generalized coordinates of the quadrupedal robot

Then, a Lagrangian L is computed as the difference between the total kinetic energy (T) and the total potential energy (V) of the system. Subsequently, substitute the Lagrangian L into the Lagrange's equations of the first kind

$$\frac{d}{dt}\frac{\partial L(q,\dot{q})}{\partial \dot{q}_i} - \frac{\partial L(q,\dot{q})}{\partial \dot{q}_i} = Z_i \tag{2.2}$$

Algorithm 2: Lagrange's equation first kind

, where t and Z represent time and the generalized forces of the system. Finally, second order differential equations for each coordinate can be derived by solving the equation above and hence obtaining the equations of motion.

2.2.3 Control Framework

Several challenges are presented in designing an efficient control algorithm for quadrupedal locomotion, which are namely stabilization of the quadrupedal robot, control of ground reaction forces, and gait patterns modulation [4].

Therefore, the control framework used three strategies to address these challenges [4]. First, self-stabilization is achieved by implementing virtual compliance in legs. Second, ground reaction forces are modulated by introducing penetration depth in foot-end trajectories. Third, gait patterns are modulated with respect to the targeted velocity and sensory feedback.

Framework Architecture

The structure of the control framework for quadrupedal locomotion is shown below, in which the main components are operator, gait pattern modulator, leg trajectory generator, and leg controller [4].

Operator

Within the control framework, operator is the first main component responsible for outputting the desired velocity and the optimal gait pattern. According to biological studies, the performance of quadrupedal locomotion is heavily dependent on the modulation of gait patterns including walk, grot, and gallop

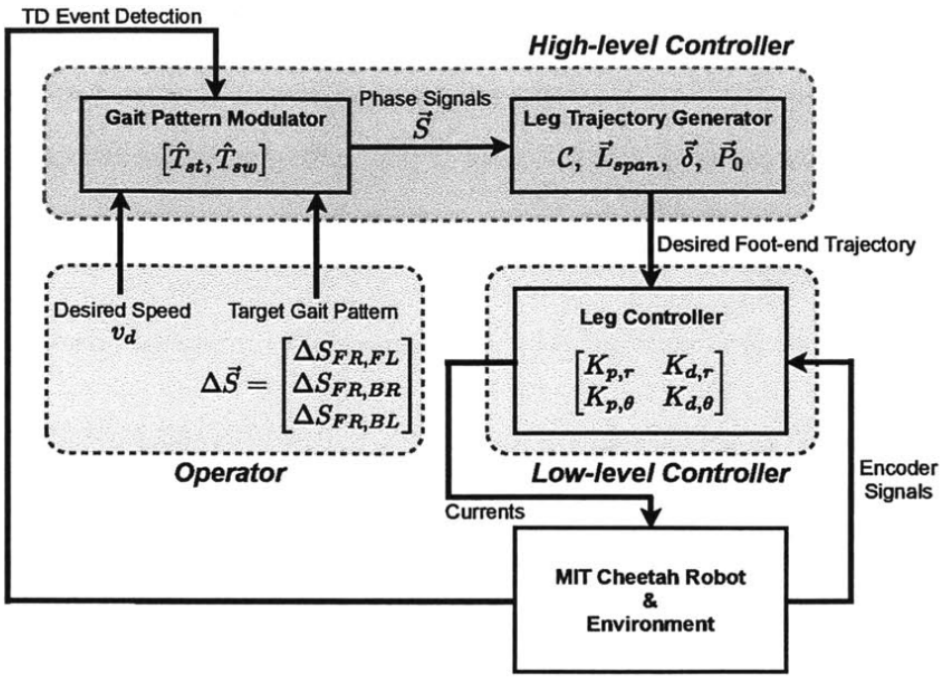

TD Event Detection
High-level Controller
Gait Pattern Modulator
$[\hat{T}_{st}, \hat{T}_{sw}]$
Phase Signals
$\vec{S}$
Leg Trajectory Generator
$C, \vec{L}_{span}, \vec{\delta}, \vec{P}_0$
Desired Foot-end Trajectory
Desired Speed
v_d
Target Gait Pattern
$\Delta\vec{S} = \begin{bmatrix} \Delta S_{FR,FL} \\ \Delta S_{FR,BR} \\ \Delta S_{FR,BL} \end{bmatrix}$
Operator
Leg Controller
$\begin{bmatrix} K_{p,r} & K_{d,r} \\ K_{p,\theta} & K_{d,\theta} \end{bmatrix}$
Low-level Controller
Encoder Signals
Currents
MIT Cheetah Robot & Environment

Figure 2.3: Architecture diagram of the hierarchical lcomotive framework [4]

[4]. By setting the optimal gait pattern, one can exploit the potential advantage of legged locomotion.

To determine the optimal gait pattern, one can evaluate the Froude number (Fr) of the quadrupedal robot given desired velocity. Froude number is a speed-length ratio that is strongly correlated to gait patterns. For example, quadrupedal animals normally transit their gait from walk to trot at Fr = 1. The Froude number is calculated as

$$Fr = \frac{v^2}{gh} \tag{2.3}$$

Algorithm 3: Froude number

, where v is the desired velocity, g is the gravitational acceleration and h is the characteristics length (hip length). Subsequently, the optimal gait pattern can be chosen based on the Froude number calculated.

The target gait pattern would then be represented as a phase lag ΔS. Gait patterns typically represent a set of predefined delays in leg motions. Meanwhile, the foot-end trajectories generally remain unchanged. Taking the front right leg as the reference leg, one would then calculate the phase lag between the reference leg and the remaining legs and eventually output the phase lag ΔS as a column vector along with the desired velocity.

Gait Pattern Modulator

The gait pattern modulator would then realize the desired velocity and target gait pattern by outputting the phase signal S. To obtain the appropriate phase signal S, the modulator would first calculate the desired stance phase period as

$$\dot{T}_{st} = \frac{2L_{span}}{v_d} \tag{2.4}$$

Algorithm 4: Desired stance phase period

where L is half of the stroke length and v is the desired velocity. Then, the desired swing phase would be set to a constant of 0.25 seconds as proposed by locomotion studies. Consequently, the phase signal Ss,i, indexed by both legs and states (stance, swing), would be assigned.

Note that the gait pattern modulator is triggered by a touch down event as seen in the structure diagram above. The algorithmic design enables the synchronization of the quadrupedal robot's legs with the physical environment. Besides, the motion errors in foot-end trajectory can be addressed by sensing the completion of swing phase.

Leg Trajectory Generator

Upon receiving phase signal S, the leg trajectory generator would generate the desired trajectories for each foot-end. The trajectories are designed using the Bezier curve defined by a set of 12 control points shown below.

The horizontal line passing through control points c0, c1, c10, and c11 at y = 500 represents the ground. The Bezier curve above the line describes the swing phase trajectories whereas the curve below modulates the ground reaction forces parameterized by stroke length and penetration depth. The generator's design is based on equilibrium point hypothesis, which suggests limb dynamics can be studied as the movement of the equilibrium point simplifying the complicated dynamics system of limbs and joints . The hypothesis further proposed that virtual compliance can be realized through introducing penetration depth in trajectory planning . Virtual compliance stimulates elastic limb motions and modulates ground reaction forces without external intervention.

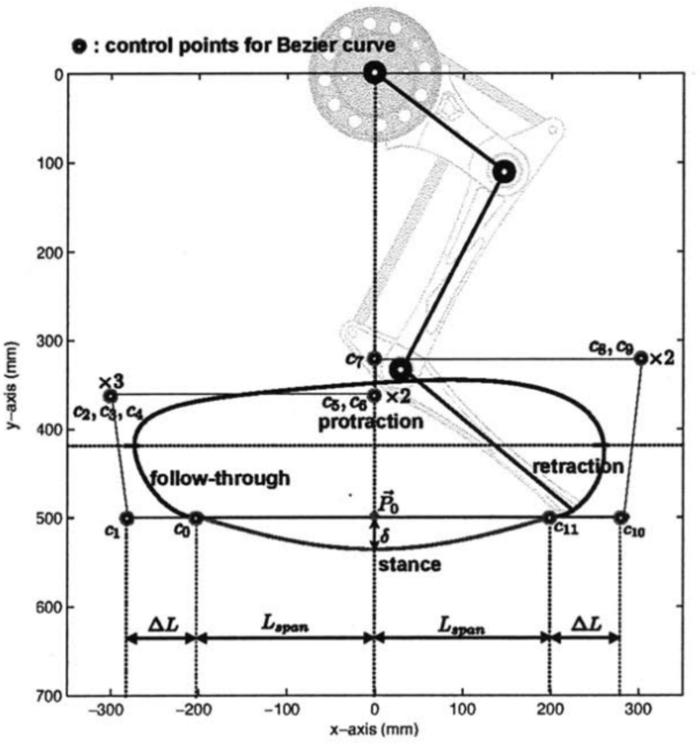


Figure 2.4: Leg trajectory generator with Bezier curve control points [4]

Self-stabilization can then be attained given that the unbalanced generalized forces would be compensated by the ground reaction forces.

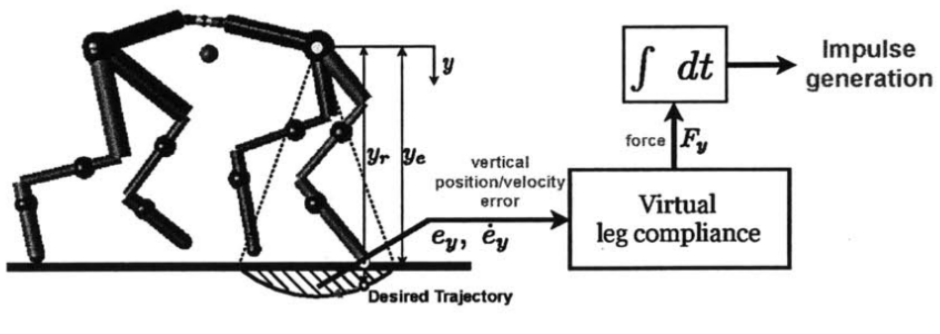


Figure 2.5: Stance trajectory design with equilibrium-point hypothesis [4]

Leg Controller

Given the desired leg trajectories, leg controller would calculate and send the torque commands to motors through impedance control. Impedance control is an approach to dynamically control the motions of a robotic system. Through impedance control, motion and interaction control in swing and stance phrases can be achieved without inverse kinematics nor inverse dynamics.

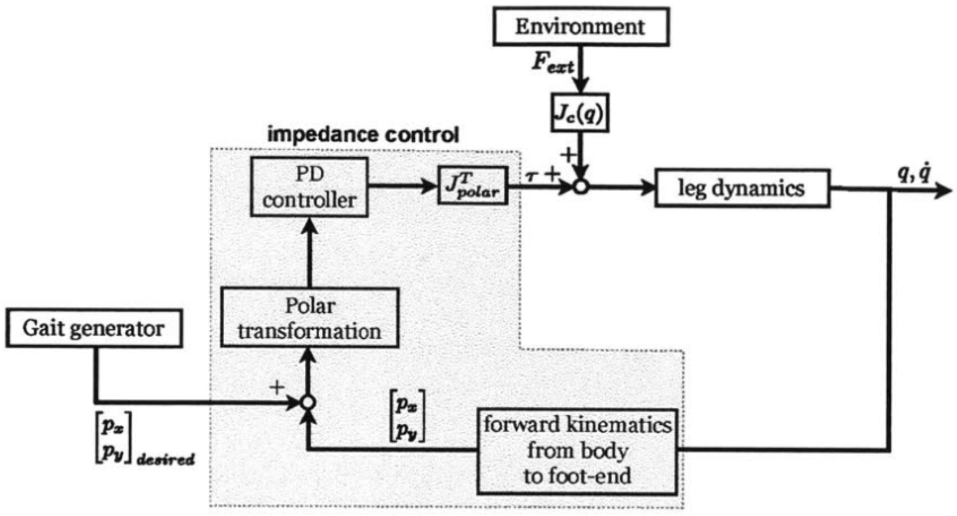


Figure 2.6: Block diagram for leg control [4]

The working principle of the control algorithm can be described as such. First, one can calculate the current foot-end position through joint angles measured by rotary encoders and forward kinematics. Then, the position and velocity errors can be obtained by determining the offset between the target foot-end position and the current foot-end position given the trajectory. Finally, the torque command can be calculated for motors with the equation below

$$u = J^T_{polar} \begin{bmatrix} K_{p,r}e_r + K_{d,r}\dot{e}_r \\ K_{p,\theta}e_\theta + K_{d,\theta}\dot{e}_\theta \end{bmatrix} \tag{2.5}$$

Algorithm 5: Torque command

, where J is Jacobian matrix in polar form, $K_{p,r}$, $K_{d,r}$, $K_{p,theta}$, $K_{d,theta}$ are radial stiffness, radial damping, angular stiffness and angular damping, and e_r, $\dot{e}_r$, e_θ, $e_{t\dot{h}eta}$ are radial position error, radial velocity error, angular position error and angular velocity error.

However, the impedance control approach requires a closed-loop control system. As mentioned above, the leg controller depends on the joint information collected from the rotary encoders to apply forward kinematics. Besides, the torque commands are applicable to motors but not servos. Taking an alternate approach, inverse kinematics can realize the motion control for servo-based quadrupedal robots in an open-loop control system. First, the target foot-end position would be obtained from the desired trajectory. Then, the target joint angles can be calculated by solving trigonometric equations, which would then be outputs as PWM signals to servos. Although the inverse kinematics approach has a higher computational cost compared to the impedance control approach, it can be implemented on cost effective components without sensory feedback.

2.3 Simultaneous Localization and Mapping (SLAM)

Simultaneous Localization and Mapping (SLAM) is the process of concurrently estimating the state of the robot and construct a representation of the environment surrounding the robot [5]. SLAM has always been one of the most critical task in the field of robotics given that SLAM forms the foundation of a variety of tasks. For example, the constructed map can provide visualization for human operators and enable path planning by informing the obstacle information. Besides, localization estimates the cuurent state of the robot, which limits the accumlated error of sensor data (e.g., IMU) [5]. Otherwise, dead-reckoning would suffer from the accumlated errors quickly.

2.3.1 What is SLAM

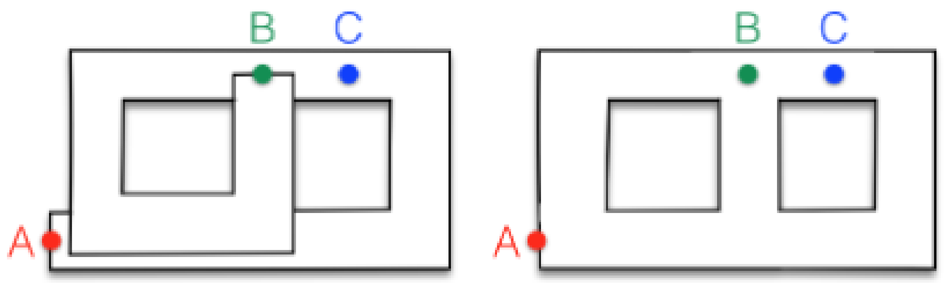


Figure 2.7: Left: map built from odometry, Right: map built from SLAM which resets the localization errors [5]

SLAM concurrently localize its state and map the surrounding environment through detecting and identifying a set of landmarks known as a priori [5]. Using dead-reckoning, inertial odometry would dift noticably within a short period because of the error accumlation nature. In contrast, the measurements of other sensor data (e.g., image, LiDAR scan) are independent from all other key frames. Therefore, the priori found by these types of sensor data would be free from accumlated errors making it appropriate as ground truth to minimize localization errors [5]. See Fig. 2.7 for an illustration of localization errors.

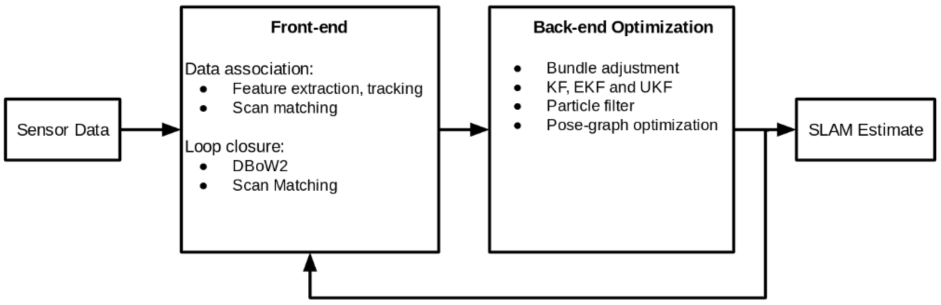


Figure 2.8: Architecture diagram of a SLAM system [5]

To implement the mentioned SLAM system, two major components would be implemented as shown in Fig. 2.8. First, visual sensor data and pointcloud data from LiDAR scans would be forwarded into the front-end component. Within the front-end component, the priori would be extracted from the sensor data through methods like corner detection [6]. Then, features extracted between keyframes would be associated to estimate the pose transform [6]. For poointcloud data from LiDAR scans, they can be associated using variats of Iterative Closest Point (ICP) [7].

2.3.2 Types of SLAM

Given the usefulness of SLAM, extensive researches have been carried out over the last 30 years [5]. Several types of SLAM methods have arisen, which are namely visual SLAM, LiDAR SLAM, and multi-sensor SLAM [8].

Visual SLAM involves extracting and associating image data from camera or other image sensors [8]. Given the versatility of image sensors, the performance of visual SLAM might vary. Taking monocular visual SLAM as an example, a single image data making depth estimation challenging. Under such a configuration, limited methods can be used to extract the priori from the image (e.g., corner detection). In contrast, stereo visual SLAM can compute the depth information given the image pairs by estimating disparities between matching key points [9]. In addition, RGB-D visual SLAM can further extract and assoicate features using the RGB information improving the accuracy while providing colorized visualization to human operators.

Light detection and ranging (LiDAR) is another type of SLAM that have been

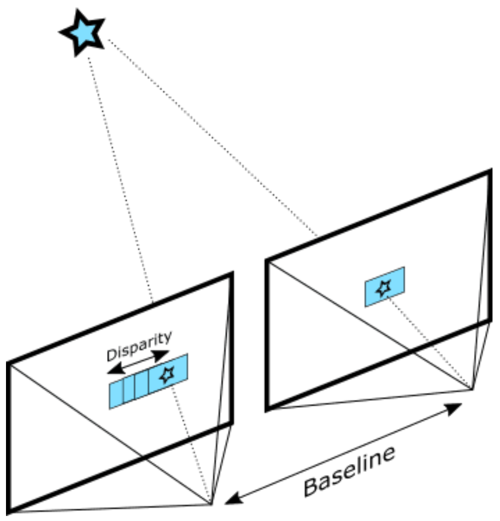


Figure 2.9: Estimate disparities between matching key points to get depth information [9]

widely deployed. Compared to image sensors, LiDAR sensors have an unrestricted field of view (FOV) while maintaining high precision [8]. LiDAR sensors can be further classified into two categories, which are planar LiDAR and 3D LiDAR. The former can effective collect occupancy status on a specific layer height, which is appropriate for smooth surface and planar map reconstruction [8]. On the other hand, 3D LiDAR provides point cloud data in a 3D space making it superior in handling complex environmnet where coverage and precision are highly prioritized [8]. For example, 3D LiDAR-based SLAM have been used in self-driving vehicles and drones [8].

Finally, multi-sensor SLAM often incorporate sensor data from both image sensors and LiDAR devices. The pose transform estimated from each sensor data stream can be merged using a weighting approach to enhance the precision and robustness of SLAM output [8]. For instance, the RGB-D camera might only have a limited field of view (FOV) but generating colorized ouput while 3D LiDAR can provide unrestricted depth information of the surrounding with high accuracy as a diverse and adaptable configuration.

2.3.3 Previous Works

In this subsection, multiple SLAM implementation would be discussed which are namely ORB-SLAM, maplab, and voxblox.

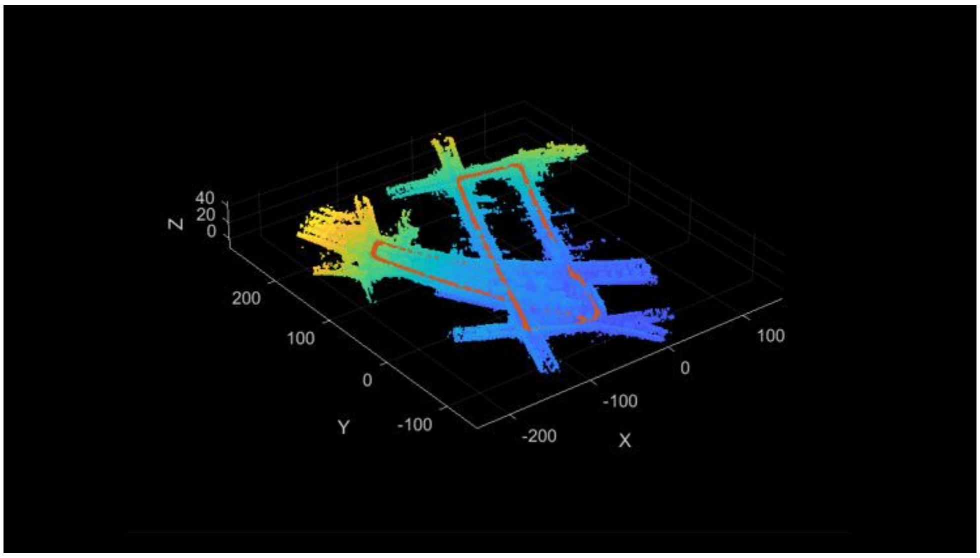


Figure 2.10: SLAM with 3D LiDAR [8]

ORB-SLAM

ORB-SLAM is a monocular SLAM system that achieves real-time performance [10]. Oriented FAST and Roated BRIEF (ORB) are binary features that generally remains constant depsite rotation and scaling [11]. ORB are chosen because it is efficient to compute and match ORB from wide baselines [10]. Therefore, three threads including tracking, local mapping and loop closing are running in parallel using the same ORB enhancing efficiency and precision [10]. See Fig. 2.11 for the architecture design of ORB-SLAM.

Given that ORB-SLAM offers robust real-time Localization and mapping services using a monocular camra, it is state-of-the-art monocular SLAM system available with minimum hardware requirment [10]. However, as ORB-SLAM only tracks and maps the ORB features available map, the constructed point map would be sparse. While a sparse point map would offer a decent performance in localization tasks and resonable visualization, it is not ideal for navigation tasks compared to dense map [10]. Fig. 2.12 is a sample reconstruction of ORB-SLAM.

ROVIOLI

ROVIOLI is another open-source visual-inertial mapping framework for maplab [12]. ROVIOLI is a lightweight and yet reliable SLAM system targeting micro aerial vehicle (MAV) [13]. To mitigate the impact of motion-blur on MAVs,

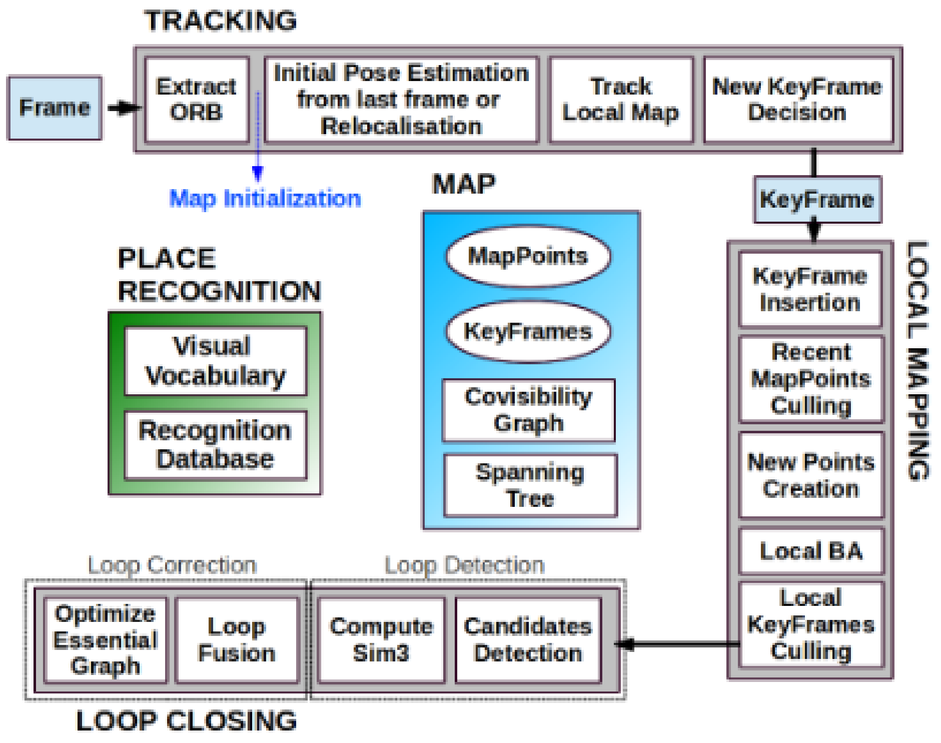


Figure 2.11: Architecture diagram for ORB-SLAM [10]

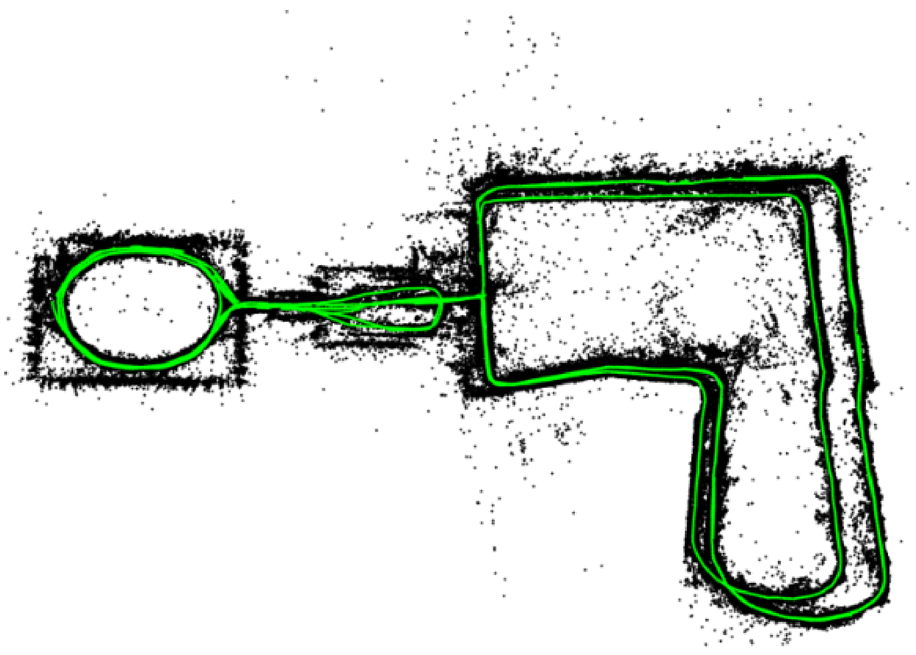


Figure 2.12: ORB-SLAM sample reconstruction [10]

ROVIOLI extracts trackable features from image patches [13]. Then, combining with IMU data, the system would be able to reconstruct a sparse point map on a large-scale environment that is over 200 meters width [12]. However, the trackable features extracted are often sparsely distributed making it not ideal for navigation tasks like ORB-SLAM [13]. See Fig. 2.13 for the architecture design of ROVIOLI.

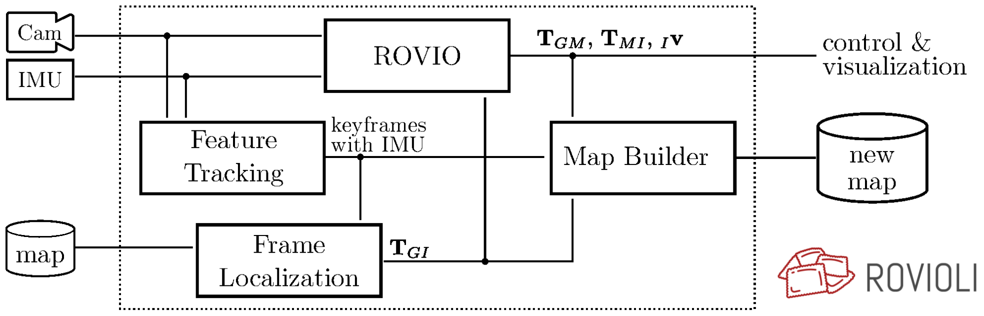


Figure 2.13: Architecture diagram for ROVIOLI [12]

Chapter 3

Project Methodology

3.1 Hardware

This section will cover the hardware design of the quadrupedal robot.

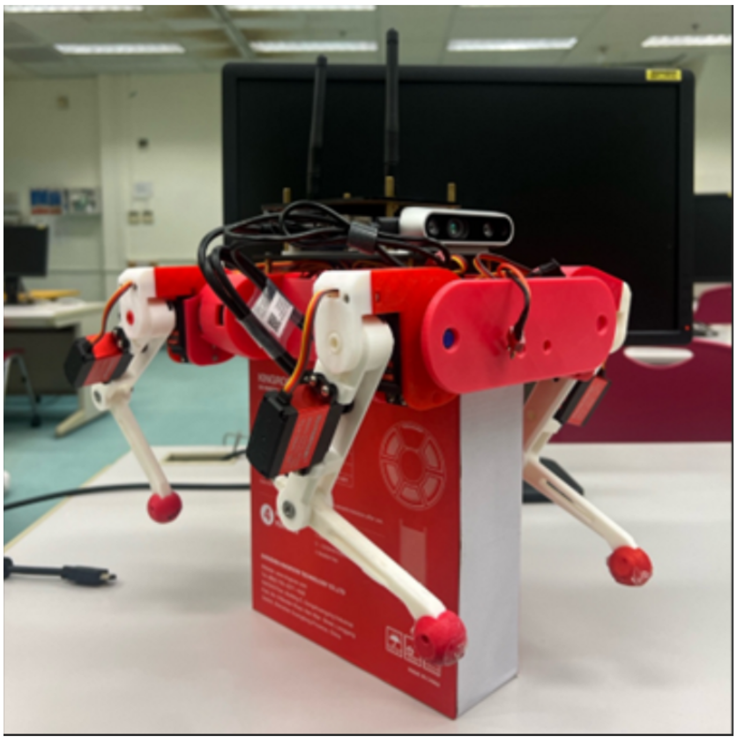


Figure 3.1: Picture of the quadrupedal robot

The frame of the robot is mainly composed of 3D-printed components, laser-cut acrylic boards, and assembling parts. Then, a set of electronic components are installed on the quadrupedal robot, which would be further described below.

3.1.1 Motion Processing Unit

A 9-axis motion processing unit MPU-9250 is used. The unit consists of a gyroscope, an accelerometer, and an onboard digital motion processor providing orientation, liear accleartion, and angular velocity information.

3.1.2 Servo Motors

Servo motors TD-8135MG are used to support precise joint movements.

3.1.3 ESP32 Module

An ESP32 module is used to distribute power and control 12 servo motors installed. In addition, it is connected to the motion processing unit to forward IMU messages.

3.1.4 NVIDIA Jetson Orin Nano

The quadrupedal robot is designed to be a self-contained agent. Therefore, computations involving leg motion planning, path planning, and object classification would be computed on device. Therefore, a GPU-equipped edge computing device, NVIDIA Jetson Orin Nano, has been installed on the quadrupedal robot.

3.1.5 Depth Camera

Intel Realsense D435i has been installed to capture the RGB-D information of the surroundings. In addition, the camera module is embedded with a inertial measurement unit (IMU) to determine the camera position and orientation information.

3.2 Software

This section will cover the software components of the quadrupedal robot.

3.2.1 Quadrupedal Control Framework

To coordinate the joint movements of the quadrupedal robot given a movement command, a package named CHAMP has been used. CHAMP is an open source development framework for quadrupedal robots basing on the hierarchical controller framework as reviewed in 2.

Unified Robotics Description Format

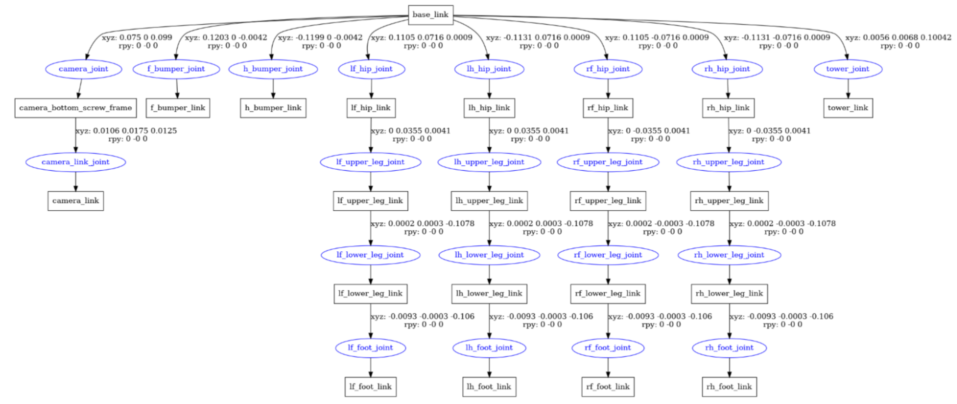

base_link
xyz: 0.075 0 0.099 rpy: 0 -0 0
xyz: 0.1203 0 -0.0042 rpy: 0 -0 0
xyz: -0.1199 0 -0.0042 rpy: 0 -0 0
xyz: 0.1105 0.0716 0.0009 rpy: 0 -0 0
xyz: -0.1131 0.0716 0.0009 rpy: 0 -0 0
xyz: 0.1105 -0.0716 0.0009 rpy: 0 -0 0
xyz: -0.1131 -0.0716 0.0009 rpy: 0 -0 0
xyz: 0.0056 0.0068 0.10042 rpy: 0 -0 0
camera_joint
f_bumper_joint
h_bumper_joint
lf_hip_joint
lh_hip_joint
rf_hip_joint
rh_hip_joint
tower_joint
camera_bottom_screw_frame
f_bumper_link
h_bumper_link
lf_hip_link
lh_hip_link
rf_hip_link
rh_hip_link
tower_link
xyz: 0.0106 0.0175 0.0125 rpy: 0 -0 0
xyz: 0 0.0355 0.0041 rpy: 0 -0 0
xyz: 0 0.0355 0.0041 rpy: 0 -0 0
xyz: 0 -0.0355 0.0041 rpy: 0 -0 0
xyz: 0 -0.0355 0.0041 rpy: 0 -0 0
camera_link_joint
lf_upper_leg_joint
lh_upper_leg_joint
rf_upper_leg_joint
rh_upper_leg_joint
camera_link
lf_upper_leg_link
lh_upper_leg_link
rf_upper_leg_link
rh_upper_leg_link
xyz: 0.0002 0.0003 -0.1078 rpy: 0 -0 0
xyz: 0.0002 0.0003 -0.1078 rpy: 0 -0 0
xyz: 0.0002 -0.0003 -0.1078 rpy: 0 -0 0
xyz: 0.0002 -0.0003 -0.1078 rpy: 0 -0 0
lf_lower_leg_joint
lh_lower_leg_joint
rf_lower_leg_joint
rh_lower_leg_joint
lf_lower_leg_link
lh_lower_leg_link
rf_lower_leg_link
rh_lower_leg_link
xyz: -0.0093 -0.0003 -0.106 rpy: 0 -0 0
xyz: -0.0093 -0.0003 -0.106 rpy: 0 -0 0
xyz: -0.0093 -0.0003 -0.106 rpy: 0 -0 0
xyz: -0.0093 -0.0003 -0.106 rpy: 0 -0 0
lf_foot_joint
lh_foot_joint
rf_foot_joint
rh_foot_joint
lf_foot_link
lh_foot_link
rf_foot_link
rh_foot_link

Figure 3.2: URDF model in a tree format

Using CHAMP to control a quadrupedal robot, a Unified Robotics Description Format (URDF) should be created and loaded into CHAMP. URDF is a representation that encodes the physical properties of a robot using XML format. In the URDF model, links, represented by meshes or collision boxes, are connected using joints while offsets and rotations define the range of motion.

Joints	Lower Limit	Upper Limit
Hip Joints	-50 °	50 °
Upper Leg Joints	-60 °	60 °
Lower Leg Joints	-120 °	60 °

Table 3.1: Configuration for joint limits

The above table shows the configuration for joint limits defining the range of motion.

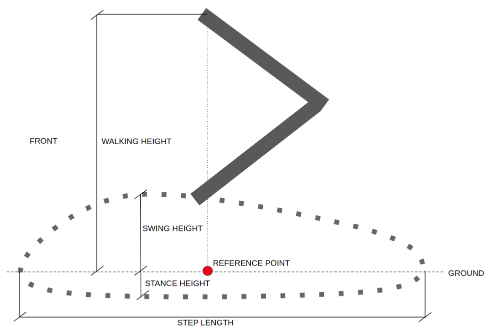


Figure 3.3: Gait parameters for the quadrupedal robot

Gait Configuration

Furthermore, depending on other physical properties (e.g., motor strength, floor friction), the gait parameters have to be fine tuned to achieve an ideal result.

3.2.2 3D Scene Recontruction

3D scene reconstruction has been leveraged to support a number of tasks, namely scene visualization for human operators, autonomous navigation, and frontier exploration. As mentioned, one of the objectives is to deploy the quadrupedal robot into unexplored environment implying that prior mapping information are often unavailable. To faciliate 3D scene reconstruction in real time, a GPU-acclearted library named Nvblox has been used.

As dicussed in the section of literature review 2, typical SLAM packages construct a sparse representation of the environment. Thsee methods have been proven effective for real-time localization and mapping tasks. However, navigation tasks (e.g., exploration and dynamic object following) requires dense obstacle information [5]. To support the desired use cases, Nvblox has been leveraged to construct Truncated Signed Distance Function (TSDF) and Euclidean Signed Distance Function (ESDF) [14]. Signed Distance Function (SDF) computes the orthogonal distance of a given point from a predefined boundary while the sign of the function output indicates the point is located inside or outside the boundary (TODO). TSDF and ESDF are variations of SDF that represent the 3D environment and have been extensively studied in the fields of computer vision and robotics application [15]. TSDF would be a 3D voxel array, in which each cell contains the distance from each voxel to the nearest surface. While ESDF is closely related to TSDF, ESDF computes the Euclidean distance without truncating the values below a predefined threshold and hence requiring high memory usage.

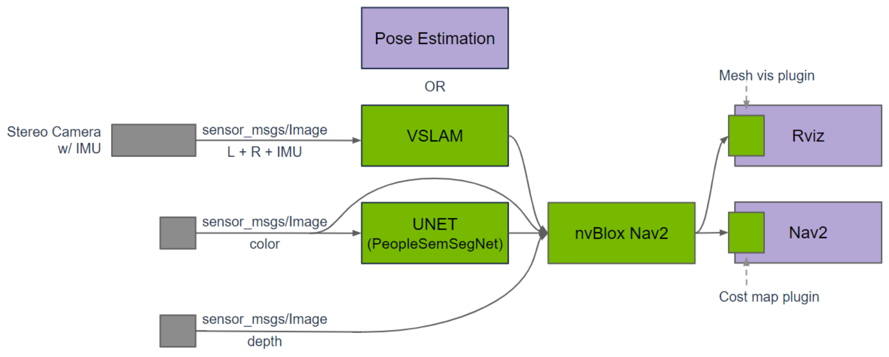


Figure 3.4: ROS Node diagram for Nvblox

Nvblox library is designed to work with depth camera and possibly 3D LiDAR as supplementary data. From Fig. 3.4, three inputs data are taken by Nvblox which are pose, RGB image and depth image. First, the pose information of the camera link can be obtained from the infrared image pairs with embedded IMU data. It is commonly computed using visual SLAM, which estimate pose by detecting visual landmarks in time-synchronized image pairs. If visual landmarks are insufficient to determine current pose, IMU data would be fused to create an estimate for odometry. Second, the RGB image would be passed into Nvblox. Notice that in certain environment, there might be dynamic objects (e.g., human) moving around. Under these circumstances, pixels that include those dynamic objects would be isolated to avoid including them in the reconstructed scene. Therefore, the RGB image would be processed by an image segmentation neural network to intelligently identify dynamic objects, in which the segmentation output would be evaluated as a mask during reconstruction. Finally, the depth image obtain from the infrared image pair would be passed into Nvblox.

Having RGB image, depth image, and pose information as camera extrinsic parameters, one can deproject the 2D images into a colored point cloud. Having a point cloud for a particular update loop, raycasting or projection mapping would be used to select the voxels that should be updated [16]. Finally, the sensor data would be integrated into a TSDF by weighting and merging strategies [16]. More importantly, Nvblox's implementation achieved GPU parallelization to enable real-time performance on edge device unlike previous works like Voxblox [14] [16].

Upon computing a TSDF of the scene, a 2D costmap can be generated which

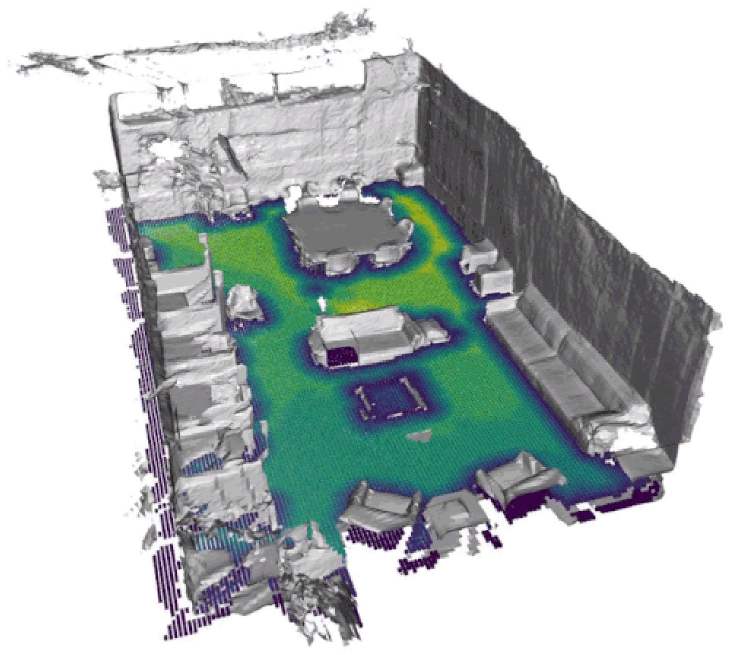


Figure 3.5: Illustration showing a slice of a TSDF constructed by Nvblox

describes the occupancy status in the scene. The conversion from a TSDF to a 2D costmap can be achieved through a process named slicing. As mentioned, the distances between each voxel and the nearest surface are stored in a TSDF. One can set desired minimum and maximum obstacle height and then simply aggregate the voxel layers' values in between. See Fig. 3.5 for illustration.

Apart from 2D costmap, a mesh can be constructed for visualization purposes. A highly efficient marching cubes algorithm has been used to construct the mesh from a TSDF. As descirbed in the algorithm, there is cube iterating through the 3D voxel array in a TSDF [17]. For each iteration, 8 voxels would be evaluated to determine the appropriate triangulation pattern [17].

3.2.3 Autonomous Navigation

Autonomous navigation offers the foundation of the quadrupedal robot's intelligent behaviours (e.g., Frontier Exploration, Dynamic Object Following). To enble autonomous navigation, Nav2 stack has been deployed to handle complex navigation applications.

Nav2 is a modular behaviour tree-based navigation stack, in which behaviour

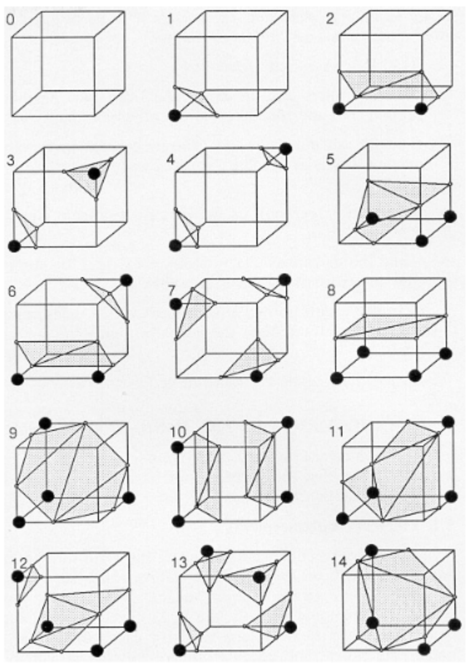


Figure 3. Triangulated Cubes.

Figure 3.6: 15 triangulated cube patterns in Marching Cubes Algorithm

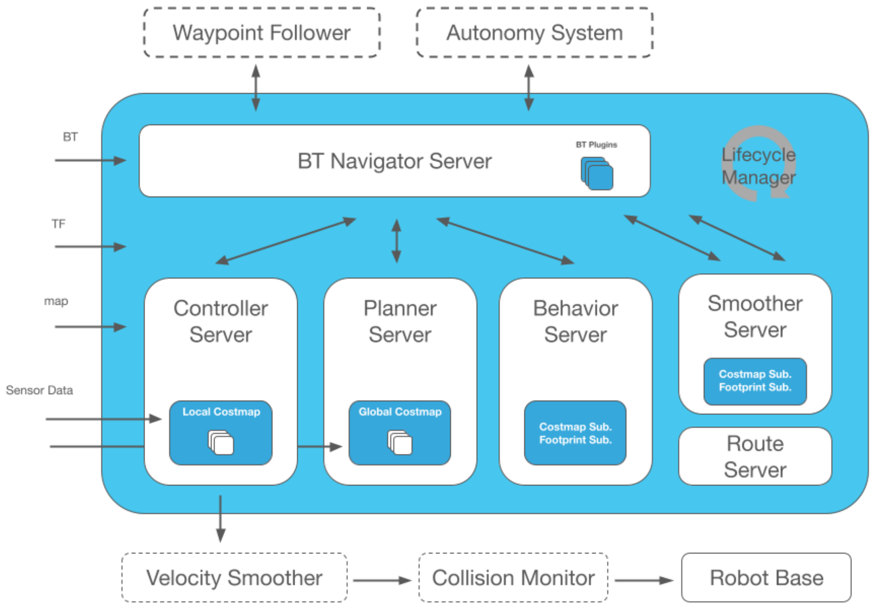


Figure 3.7: Nav2 architecture diagram

tree is a tree-based execution model to represent navigation logics [18]. By introducing the concept of behaviour tree, complex navigation logics can be encoded in human-readable and expressive format [19]. For example, one robot might have to go back to charging stations or play as an agent in a Pac-Man game. Not only can behaviour tree create highly complex navigation system, human-readable property of behaviour tree makes a formalized analysis possible using a state space description to evaluate the safety and robustness of the robotics system in critical situations [19]. In addition, custom plugins can be defined to provide custom behaviour tree node maximizing extensibility.

Besides, as shown in Fig. 3.7, Nav2 is designed to be a modular navigation stack in which individual component can be swapped out for a type-compatible component. Take smoother server as an example, Savitzky-Golay smoother would only reduce the noise from the path generated by planner server while Constrained smoother would maintain a minimum turning radius during optimization [18]. Then, it would be more beneficial for a quadrupedal robot, which has a turning radius close to 0, to use Savitzky-Golay smoother over Constrained smoother. Similarly, other components can be customized to maximize the efficiency of a certain robot configuration.

3.2.4 Dynamic Object Following

Upon deploying the quadrupedal robot to the field, the quadrupedal robot should be able to maintain a close distance with the operator without any human interventions. Therefore, a dynamic object following algorithm has been implemented to achieve the task. In addition, given the generalizability of the algorithm, the dynamic object following algorithm can be easily modified to track and follow different objects offering possbilities for a variety of use cases.

The algorithm can be broken into the following steps:

1. Identify the presence of human object in RGB image
2. Approximate the 2D coordinate of the human object
3. Deproject the 2D coordinate to 3D world coordinate
4. Update the goal pose with behaviour tree

To identify the presence of human object, an image segmentation neural network would be deployed. Image segmentation is the process of partitioning images into multiple segments, which represent classes or objects [20]. In other words, each pixel would be assigned a class ID representing the respective label. The pretrained PeopleSegNet ShuffleSeg model has been chosen because the training dataset includes a mix of camera heights, and field-of-view (FOV) making it ideal for the camera configuration of the quadrupedal robot [21]. Besides, the ShuffleSeg variation of the pretrained model implements grouped convolution and channel shuffling in the encoders [22]. Hence, the model can achieve real time inference on edge devices because of the improved computation efficiency. See Fig. 3.8 for sample segmentation output.

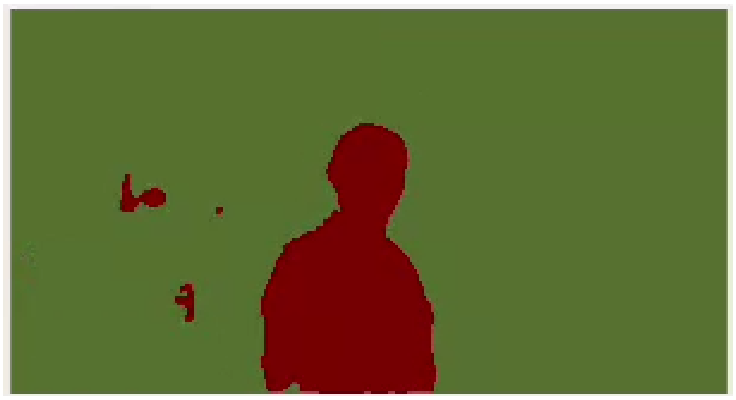


Figure 3.8: Segmentation output from PeopleSegNet ShuffleSeg

The 2D coordinate of the human object can be approximated form the segmentation output. Given that the image segmentation model only assigns two class labels (e.g., background, human), the segmentation output can be treated as a binary mask [21]. Then, one can identify all connected components and sort by their areas using OpenCV [11]. Finally, the 2D coordinate of the human object can be computed as the centroid of the largest connceted component in the mask. Several assumptions have been made in the algorithm. First, the target human object should have the largest area in the RGB image given that misclassified objects should have relatively small areas. Second, the RGB image only include a single human object. Otherwise, occlusion problem might occur affecting the calculation of the 2D coordinate while it is undetermined that which human object should the quadrupedal robot follow.

Before proceeding to the deprojection step, the depth information of the 2D coordinate has to be obtained. Notice that the dimensions of the segmentation mask and the depth image may vary. The segmentation mask and its respective 2D coordinate have to be aligned with the depth image. Then, the depth value can be accessed directly through the rescaled 2D coordinate.

Obtaining the 2D coordinate and its depth information, the target 2D point can be deprojected into 3D world space. In addition to the mentioned information,

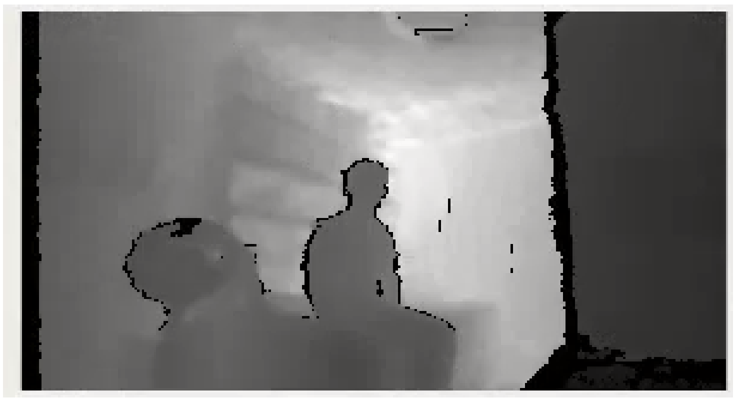


Figure 3.9: Depth image obtained from Intel Realsense depth camera

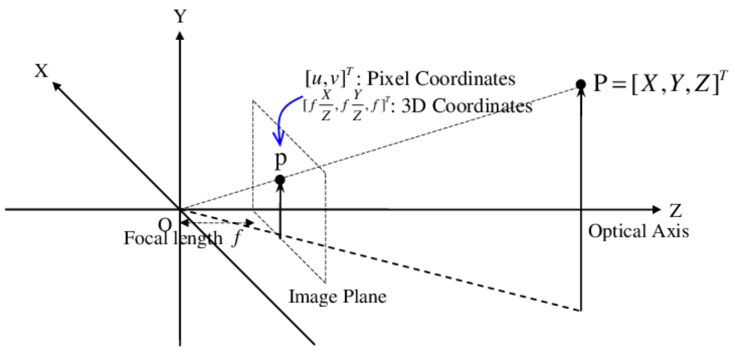
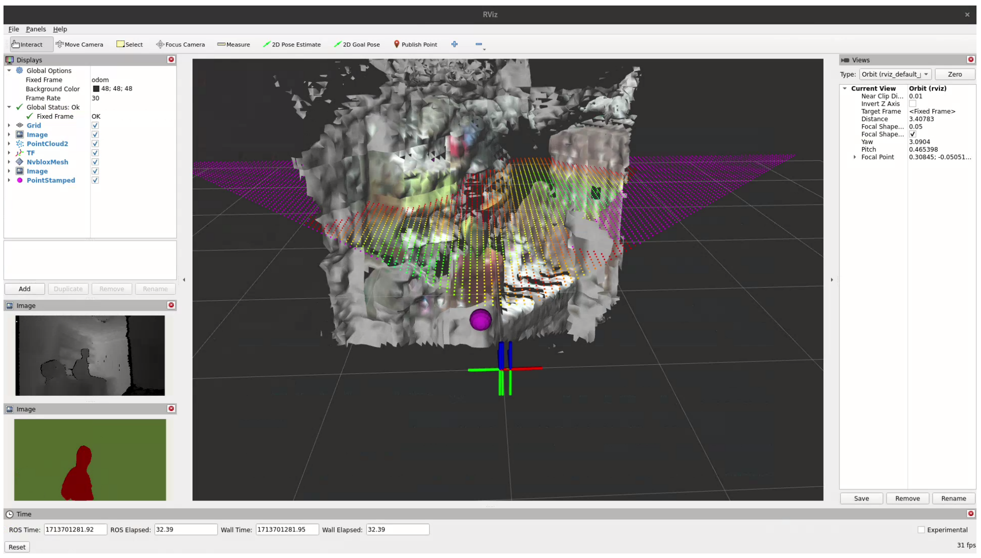

Y
X
$[u,v]^T$: Pixel Coordinates
$[f\frac{X}{Z}, f\frac{Y}{Z}, f]^T$: 3D Coordinates
$P=[X,Y,Z]^T$
p
O
Focal length f
Z
Optical Axis
Image Plane

Figure 3.10: Perspective projection

the camera intrinsic and extrinsic parameters (e.g., focal length, image dimension, camera position, and camera orientation) are required. The camera intrinsic parameters can be subscribed through the "camera_info" topic while the camera extrinsic parameters can be obtained as the pose information estimated by visual SLAM package.

Finally, the 3D world coordinate would be transformed into a pose by setting the appropriate x and y coordinates as the goal pose. As mentioned, Nav2 is a behaviour tree-based navigation stack [18]. Therefore, a behaviour tree would be configured to facilitate the dynamic object following behaviour. See below for the definition of the behaviour tree.

```
<root BTCPP_format="4" main_tree_to_execute="MainTree">
  <BehaviorTree ID="MainTree">
    <PipelineSequence name="NavigateWithReplanning">
      <ControllerSelector
        selected_controller="{selected_controller}"
        default_controller="FollowPath"
        topic_name="controller_selector"
```


RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Goal Pose Publish Point
Displays
Add Duplicate Remove Rename
Image
Image
Views
Save Remove Rename
Time
Reset

Global Options	
Fixed Frame	odom
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
Image	✓
PointCloud2	✓
TF	✓
NvbloxMesh	✓
Image	✓
PointStamped	✓

Type: Orbit (rviz_default_ Zero

Current View	Orbit (rviz)
Near Clip Di...	0.01
Invert Z Axis	☐
Target Frame	<Fixed Frame>
Distance	3.40783
Focal Shape...	0.05
Focal Shape...	✓
Yaw	3.0904
Pitch	0.465398
Focal Point	0.30845; -0.05051...

ROS Time: 1713701281.92 ROS Elapsed: 32.39 Wall Time: 1713701281.95 Wall Elapsed: 32.39 ☐ Experimental

31 fps

Figure 3.11: Visualization of the 3D point representing the human object's centroid (purple sphere)

```
/>
<PlannerSelector
  selected_planner="{selected_planner}"
  default_planner="GridBased"
  topic_name="planner_selector"
/>
<RateController hz="1.0">
  <Sequence>
    <GoalUpdater
      input_goal="{goal}"
      output_goal="{updated_goal}"
    >
      <ComputePathToPose
        goal="{updated_goal}"
        path="{path}"
        planner_id="{selected_planner}"
        error_code_id="{compute_path_error_code}"
      />
    </GoalUpdater>
   <TruncatePath
     distance="1.0"
     input_path="{path}"
     output_path="{truncated_path}"
    />
  </Sequence>
</RateController>
<KeepRunningUntilFailure>
  <FollowPath
```

```
          path="{truncated_path}"
          controller_id="{selected_controller}"
          error_code_id="{follow_path_error_code}"
        />
      </KeepRunningUntilFailure>
    </PipelineSequence>
  </BehaviorTree>
</root>
```

In short, the above behaviour tree would navigate to the input goal, which is the current location of the human object. Then, the plannar server would replan the path given that costmap information would be updated during navigation. In addition, the goal is updated each second by subscribing to the /goal_update topic. The navigation would then be terminated once the quadrupedal robot is 1 meter away from the destination or no available path is found.

3.2.5 Frontier Exploration

To effectively explore unvisited field, the quadrupedal robot should be capable of frontier exploraion. The exploration process enables the quadrupedal robot to effectively map the field with 3D scene reconstruction.

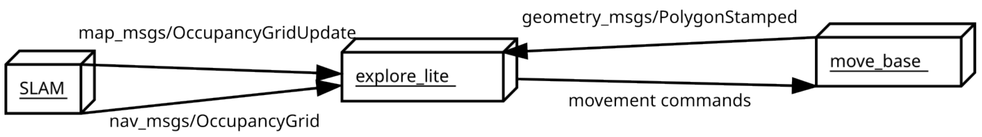


Figure 3.12: Explore lite architecture diagram

The package explore_lite would be used to achieve the task. As shown in Fig. 3.12, the package subscribes to messages of types nav_msgs/OccupancyGrid and
map_msgs/OccupancyGridUpdate to keep track of the costmap of interest. Then, a breadth-first search would be performed on the costmap while considering the occupancy status of each cell [23]. A list of frontiers would then be computed and be explored by publishing movement commands to the navigation server [23].

To configure the exploration behaviour, one can finetune the following parameters [23]

1. potential_scale: determines if the robot is prefered to explore distanced frontiers

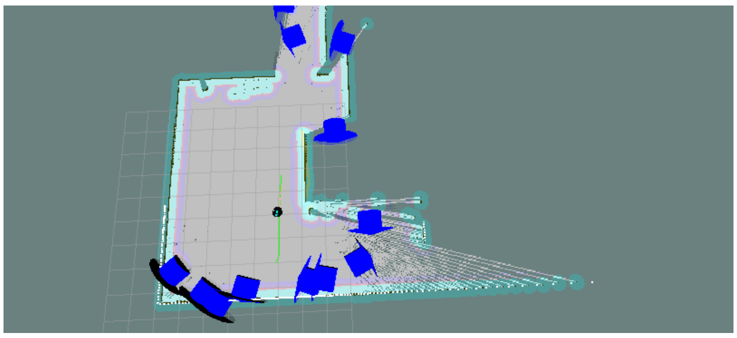


Figure 3.13: Visualization of frontiers generated by explore lite (blue points)

2. orientation_scale: determines if the robot is prefered to explore frontiers ahead

3. min_frontier_size: determines the minimum frontier size to be explored, which is essential to exploration task in restricted, indoor environment

By deploying the exploration node, the quadrupedal robot would greedily explore the unvisited field until all frontiers have been exhaustively explored.

3.2.6 Object Detection

During exploration tasks, it is essential for the quadrupedal robot to idnetify potential hazards and injuried people. To achieve the task, object detection can be leveraged. Object detection detect the presence of objects by outputing the bounding box representation with an associative confidence score [24]. See Fig. 3.14 for sample prediction.

Object detection models are often computational efficient with YOLO detector being able to maintain a real-time performance [24]. The characteristics of object detection models make it suitable to efficiently classify a number of objects in real time. For example, there could be a large number of potentially hazardous objects (e.g., pressurized gas cylinder, damaged transmission tower) in a disaster field. Alerting operators about potential dangers in real time would minimize the risk factor of the rescue team if human involvment is required.

Besides, the dataset preparation for training object detection models are less costly. For example, one can collect a large amount of dataset by recording and labelling farme within a video. The labelling process can be speed up through a semi-automatic annotation in CVAT, in which only key frames have to be manually annotated and the frames in between would be tracked automatically [25].

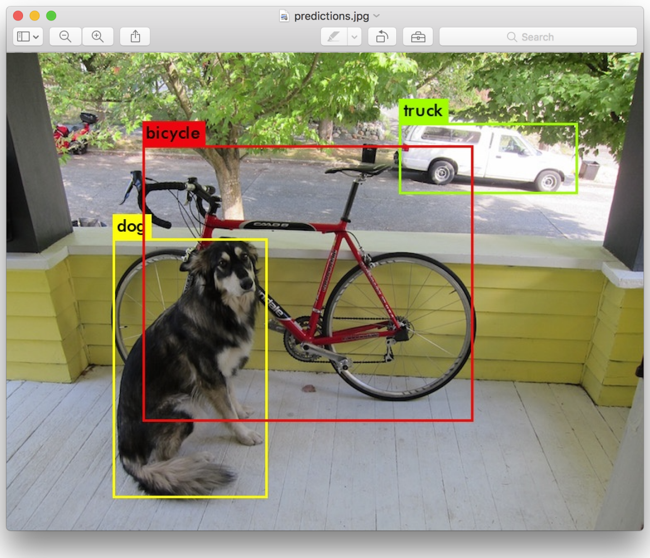


Figure 3.14: Prediction result evaluated by YOLO

The ease of dataset preparation enables object detectors to be trained on a customized dataset for each event type of disaster enhancing the performance and versatility of the object detector deployed.

3.2.7 Web Application

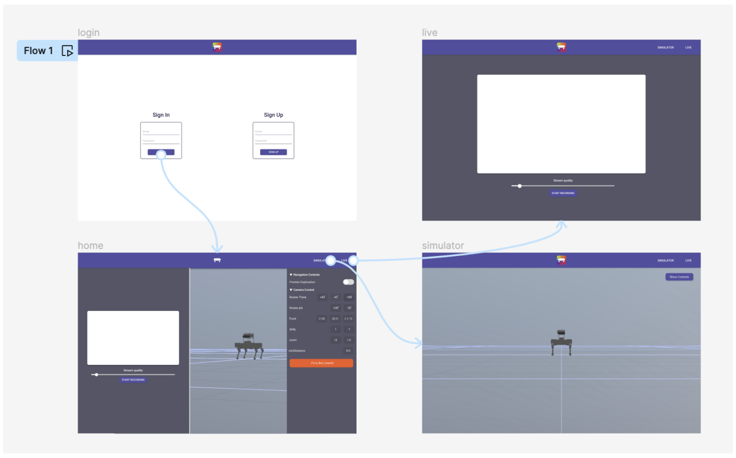


Figure 3.15: Illustration of the web application

To provide a interactive dashboard for human operators, a web dashboard has been implemented using React JS. To connect the ROS network running on the quadrupedal robot, the rosbridge_suite package is used, which starts a WebSocket server publishing client-requested topics. The web dashboard provides several features including live camera feed and live visualization.

Live Camera Feed

To faciliate live camera streaming on web clients, a pacakge web_video_server is installed and deployed on the ROS network. The package enables web clients to subscribe to a image topic as a stream with various quality and encoding format settings.

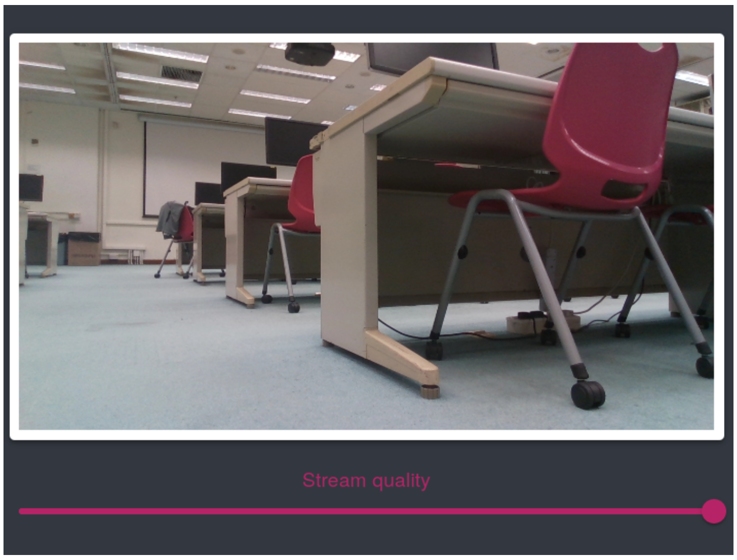


Figure 3.16: Live camera streaming

Live Visualization

To visualize the quadrupedal robot movement, a live simulator is implemented using ThreeJS and URDF loader. When connected to the ROS network via WebSocket, the movement of the simulated quadrupedal robot would be synchronized to that the real counterpart. Nevertheless, the top-right control panel enables configuring field of view and toggling froniter exploration.

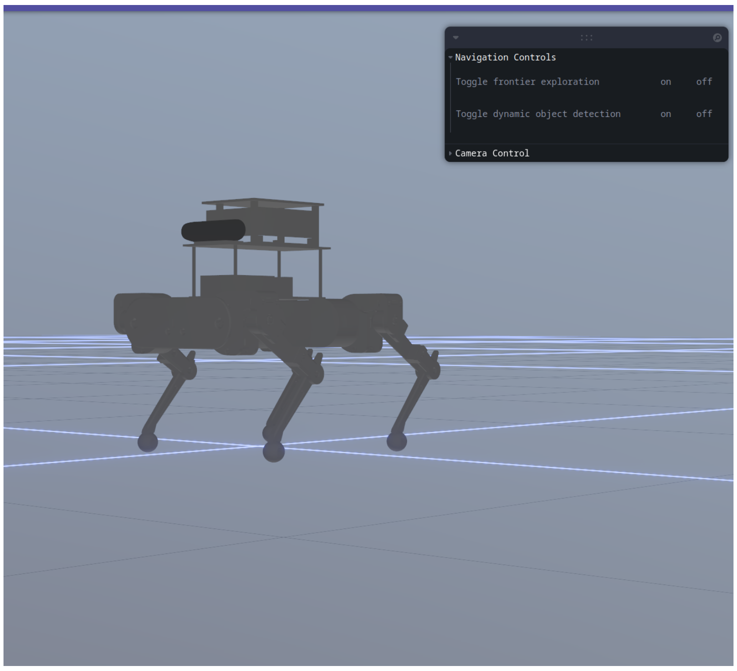

Navigation Controls
Toggle frontier exploration on off
Toggle dynamic object detection on off
Camera Control

Figure 3.17: Live visualization with controls

3.3 Development and Deployment

Given that the installed ROS packages are distributed on distinct ROS distributions, multiple Docker container images have been built and deployed to launch different packages. Below are the Docker containers created.

- ROS1 Melodic: Install CHAMP and rosbridge_suite packages
- ROS1/2 Bridge: Build ros1_bridge package from source to add bridging capability for custom message types (e.g., "vision_msgs/BoundingBox2D")
- ROS1 Humble: Install computer vision and Nav2 packages

In addition, Docker compose files have been created to launch a developer container for debugging purposes.

Chapter 4

Results and Findings

4.1 Robot Construction

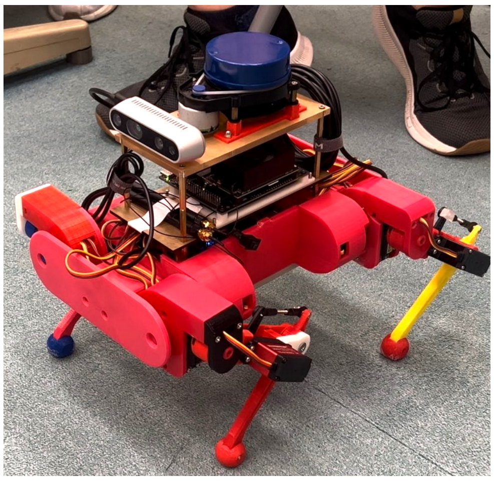


Figure 4.1: Quadrupedal robot with Jetson Nano, stereo camera, LiDAR, power bank, and lithium battery

Several robot configurations were tested. Take Fig. 4.1 as an example, the self-contained robot configuration is equipped with both stereo camera and LiDAR. Combining the sensor data from both stereo camera and LiDAR, one can see the maximum potential in autonomous navigation and 2D mapping. However, given the insufficient strength of the building material and the servo motors, the motion of the quadrupedal robot is unfavourable as significantly movement variations were observed using any movement commands.

Therefore, depsite the availability of planar LiDAR, it is decided to remove planar LiDAR and the power bank to reduce the weight of the quadrupedal robot. Under this configureation, a type-C to DC plug cable has to be connected to the onboard Jetson for power supply. However, significant improvements

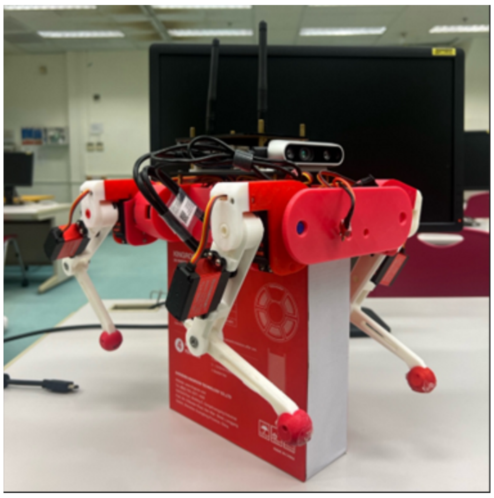


Figure 4.2: Quadrupedal robot with Jetson Nano, stereo camera, and lithium battery

in motion execution have been observed. Notice that without the planar LiDAR, the obstacle detection capbility has decreased given the limited FOV of the stereo camera. However, the performance of remaining computer vision remains steady.

4.2 3D Scene Reconstruction

As mentioned in 3, the RGB-D sensor data have been used to reconstruct a 3D represenetation of the environment using Nvblox. However, the quality of the reconstructed scene vary across different environments.

The Fig. 4.3 shows the reconstructed map of the bedroom during testing. One can observe the reconstructed mesh is smooth without minimal noise.

However, when performing tests in Computer Science lab, the quality of the depth image obtained is poor depsite manually tuning the camera parameters. Hence, the output mesh is often noisy. The issue further hinders the capability of autonomous navigation as noise in the mesh has been mistakenly recognised as obstacles.

4.3 Dynamic Object Following

A dynamic object following algorithm has been implemented to enable the quadrupedal robot to maintain a close distance with the operator. However,

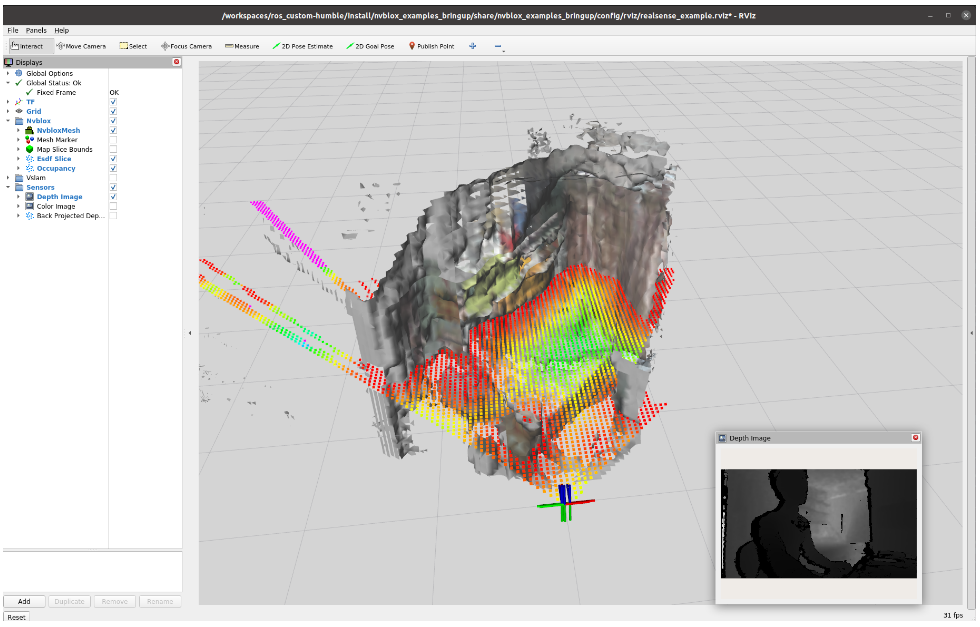


Figure 4.3: Scene reconstructed using Nvblox

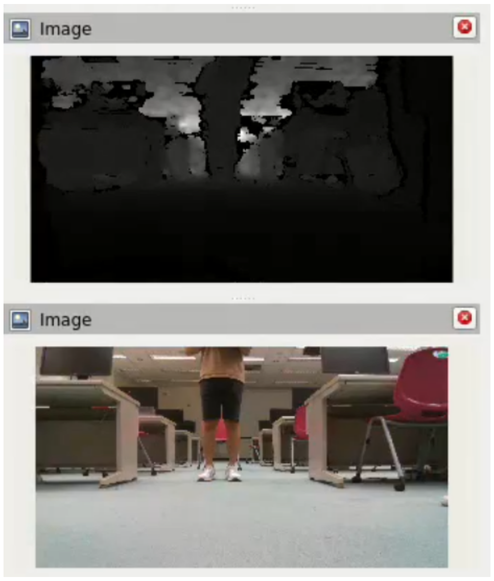
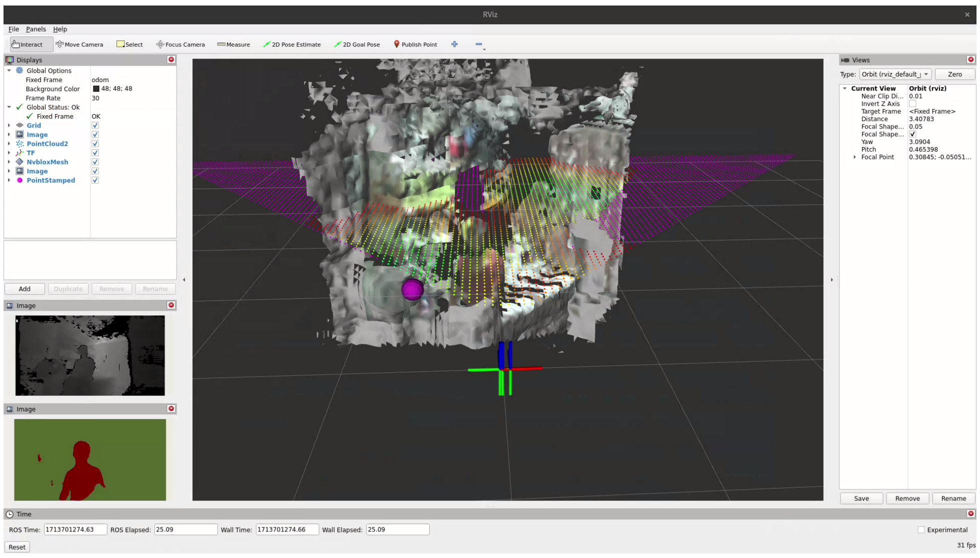


Figure 4.4: Color and depth image pairs

during testing, depsite the robot identify the 3D coordinate of the tester, it fails to navigate to the goal pose. It is believed that synchronization issues and hardware invariance are the major issues.

First, as dicussed in 3, the detection of the human object is achieved by passing the RGB image into the segmentation network. Not only the network imposes a processing delay on the image, the camera module might not output the time



```
Global Options
  Fixed Frame          odom
  Background Color     48; 48; 48
  Frame Rate           30
Global Status: Ok
  Fixed Frame          OK
Grid                   ✓
Image                  ✓
PointCloud2            ✓
TF                     ✓
NvbloxMesh             ✓
Image                  ✓
PointStamped           ✓

Type: Orbit (rviz_default_
Current View           Orbit (rviz)
  Near Clip Di...      0.01
  Invert Z Axis        ☐
  Target Frame         <Fixed Frame>
  Distance             3.40783
  Focal Shape...       0.05
  Focal Shape...       ✓
  Yaw                  3.0904
  Pitch                0.465398
  Focal Point          0.30845; -0.05051...

ROS Time: 1713701274.63  ROS Elapsed: 25.09  Wall Time: 1713701274.66  Wall Elapsed: 25.09    ☐ Experimental
                                                                                              31 fps
```

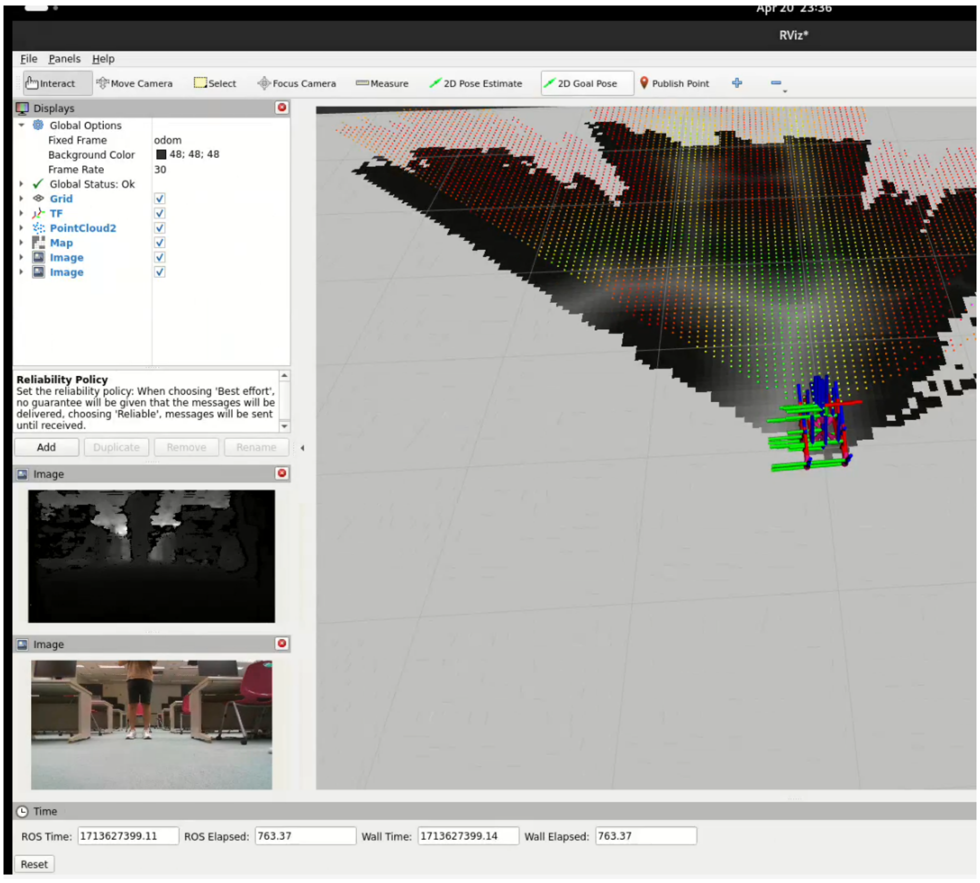
Figure 4.5: Pose estimation

synchronized RGB and depth image pairs. As a result, the centroid position obtained from the segmentation output might not align with that of the depth image resulting a drift in the predicted 3D coordinate. As shown in Fig. 4.5, the purple sphere indicating the predictionn coordinate is on the left of the tester (in red shirt).

Second, the robot fails to navigate to the goal pose depsite successful identification. The second issue might arise from inaccurate sensor data from IMU. When moving, the quadrupedal experience significant vibration. Given that the calculation of the target 3D coordinate depends on the camera extrinsic parameters (e.g., position, orientation), the predicted 3D coordinate would drift a lot once it starts moving. This aligns to the observation that the prediction would experience significant errors once started moving.

4.4 Frontier Exploration

During testing, the exploration process stops almost immediately. It is later discovered that because of the limited FOV of the stereo camera. The robot fails to determine if there are obstacle right in front of it. Therefore, the frontier searching process is unable to proceed. However, by manually navigation to the mapped zone, frontier exploration can be achived.



Global Options
Fixed Frame odom
Background Color 48; 48; 48
Frame Rate 30
Global Status: Ok
Grid ✓
TF ✓
PointCloud2 ✓
Map ✓
Image ✓
Image ✓

Reliability Policy
Set the reliability policy: When choosing 'Best effort', no guarantee will be given that the messages will be delivered, choosing 'Reliable', messages will be sent until received.

ROS Time: 1713627399.11 ROS Elapsed: 763.37 Wall Time: 1713627399.14 Wall Elapsed: 763.37

Figure 4.6: Exploration

Chapter 5

Future Works

5.1 Instance Segmentation

In the dynamic object following algorithm, it is assumed that only 1 human object would appear in the RGB image. However, it is obvious that the assumption is not realistic. See the figure below, when two human objects are close to each other, a problem known as occlusion problem might happen. It affects the computation of target 3D coordinate. Even worse when two human objects walk in a opposite direction, it is uncertain whether the robot would follow which person.

To solve this issue, it is believed that deploying an instance segmentation model would be the solution. Apart from assigning each pixel a class label, it assigns an instance ID. Using such a network, even if occlusion problem occurs, the network would still be able to identify individual object.

Then, back to the two people walking in opposite direction problem, it can then keep track of the target coordinate in the previous frame. For each frame, the distance between the previous target and the new potential targets can be calculated. Finding the minimum distance can enable the robot confidently follows the same person dynamically.

5.2 Sensor Fusion

In preivous section, it is discussed that hardware invariance issue might impact performance on dynamic object following and 3D scene reconstruction. In curent robot configuration, two IMU sensors are available in the system. To minimize IMU errors, it is possible to apply a EKF filter to fuse readings from multiple IMU sensors in addition to the magnetometer.

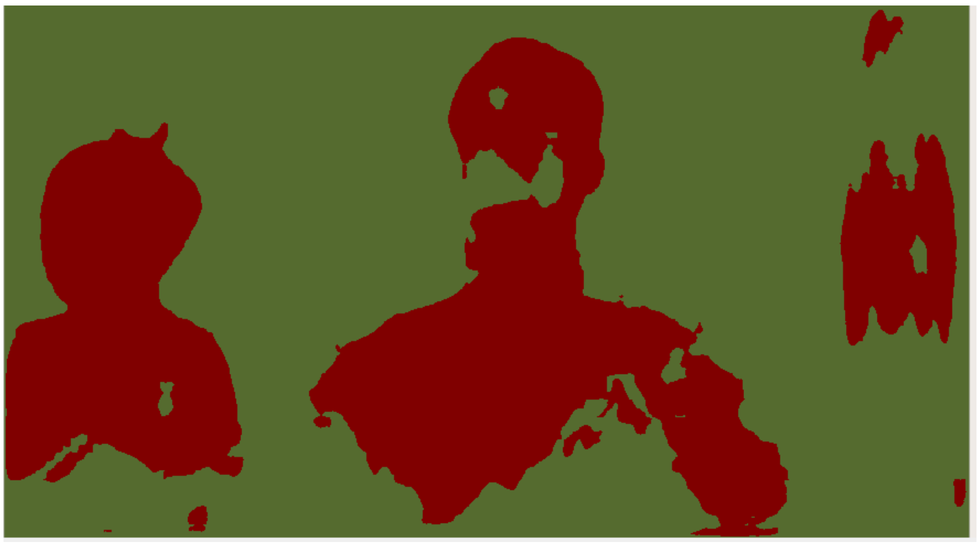


Figure 5.1: Occlusion Problem

Similar technique can be applied to depth sensing. With the limited FOV of depth camera, the quadrupedal robot cannot percept the environment surrounding it. By installing planar or 3D LiDAR, the accuracy of depth sensing can be significantly improved enhancing path planning tasks.

5.3 Reinforcement Learning

With 12 degree of freedom, determining the optimal leg movment can be a huge challenge. Besides, it is currently assumed that the robot only works inclined but even terrain. To dynamically determine the optimal leg movement across a variety of terrain, it is recommended to use reinforcement learning to train a optimized nerual network for this specific task. With the current advancement of edge devices and GPU computing, it is believed that the RL approach would significantly improve navigation performance on unseen terrain.

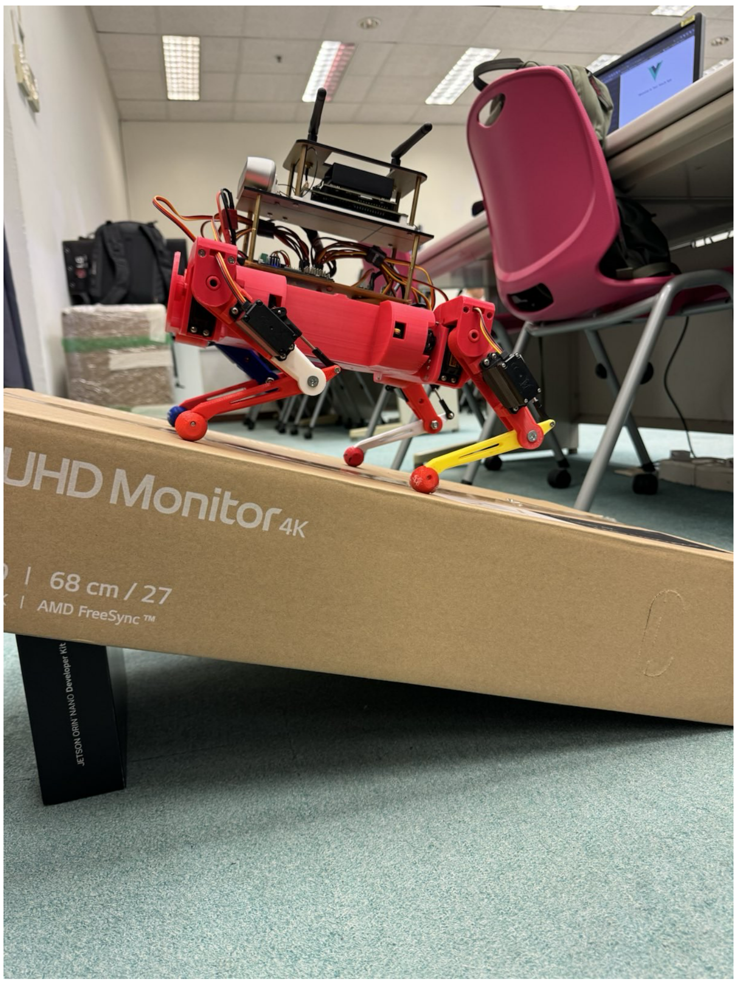


Figure 5.2: Standing on inclined surface

Bibliography

[1] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng. "ROS: an open-source Robot Operating System". In: vol. 3. Jan. 2009.

[2] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall. "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[3] chvmp. *chvmp/champ: MIT Cheetah I Implementation*. 2024. URL: https://github.com/chvmp/champ (visited on 04/17/2024).

[4] J. Lee. "Hierarchical controller for highly dynamic locomotion utilizing pattern modulation and impedance control: Implementation on the MIT Cheetah robot". PhD thesis. Massachusetts Institute of Technology, 2013.

[5] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age". In: *IEEE Transactions on Robotics* 32.6 (Dec. 2016), pp. 1309–1332. ISSN: 1941-0468. DOI: 10.1109/tro.2016.2624754. URL: http://dx.doi.org/10.1109/TRO.2016.2624754.

[6] B. Garigipati, N. Strokina, and R. Ghabcheloo. *Evaluation and comparison of eight popular Lidar and Visual SLAM algorithms*. 2022. arXiv: 2208.02063 [cs.RO].

[7] S. Rusinkiewicz and M. Levoy. "Efficient variants of the ICP algorithm". In: *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*. 2001, pp. 145–152. DOI: 10.1109/IM.2001.924423.

[8] MathWorks. *What is SLAM (Simultaneous Localization and Mapping (SLAM) - MATLAB SimuLink)*. Last accessed 26 April 2024. 2024. URL: https://www.mathworks.com/discovery/slam.html.

[9] I. RealSense. *The basics of stereo depth vision*. Last accessed 26 April 2024. 2024. URL: https://www.intelrealsense.com/stereo-depth-vision-basics/.

[10] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. "ORB-SLAM: A Versatile and Accurate Monocular SLAM System". In: *IEEE Transactions on Robotics* 31.5 (Oct. 2015), pp. 1147–1163. ISSN: 1941-0468. DOI: 10.1109/tro.2015.2463671. URL: http://dx.doi.org/10.1109/TRO.2015.2463671.

[11] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[12] T. Schneider, M. T. Dymczyk, M. Fehr, K. Egger, S. Lynen, I. Gilitschenski, and R. Siegwart. "maplab: An Open Framework for Research in Visual-inertial Mapping and Localization". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1418–1425. DOI: 10.1109/LRA.2018.2800113.

[13] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart. "Robust visual inertial odometry using a direct EKF-based approach". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 298–304. DOI: 10.1109/IROS.2015.7353389.

[14] A. Millane, H. Oleynikova, E. Wirbel, R. Steiner, V. Ramasamy, D. Tingdahl, and R. Siegwart. *nvblox: GPU-Accelerated Incremental Signed Distance Field Mapping*. 2024. arXiv: 2311.00626 [cs.RO].

[15] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. "KinectFusion: Real-time dense surface mapping and tracking". In: *2011 10th IEEE International Symposium on Mixed and Augmented Reality*. 2011, pp. 127–136. DOI: 10.1109/ISMAR.2011.6092378.

[16] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto. "Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Sept. 2017. DOI: 10.1109/iros.2017.8202315. URL: http://dx.doi.org/10.1109/IROS.2017.8202315.

[17] W. Lorensen and H. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In: *ACM SIGGRAPH Computer Graphics* 21 (Aug. 1987), pp. 163–. DOI: 10.1145/37401.37422.

[18] S. Macenski, F. Martin, R. White, and J. Ginés Clavero. "The Marathon 2: A Navigation System". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020.

[19] M. Colledanchise and P. Ögren. *Behavior Trees in Robotics and AI*. July 2018. DOI: 10.1201/9780429489105. URL: http://dx.doi.org/10.1201/9780429489105.

[20] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos. *Image Segmentation Using Deep Learning: A Survey*. 2020. arXiv: 2001.05566 [cs.CV].

[21] NVIDIA. *PeopleSegNet Model Card | NVIDIA NGC*. Last accessed 26 April 2024. 2024. URL: https://catalog.ngc.nvidia.com/orgs/nvidia/teams/tao/models/peoplesegnet.

[22] M. Gamal, M. Siam, and M. Abdel-Razek. *ShuffleSeg: Real-time Semantic Segmentation Network*. 2018. arXiv: 1803.03816 [cs.CV].

[23] J. Hörner. *Map-merging for multi-robot system*. Bachelor's thesis. Prague, 2016. URL: https://is.cuni.cz/webapps/zzp/detail/174125/.

[24] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[25] B. Sekachev, N. Manovich, M. Zhiltsov, A. Zhavoronkov, D. Kalinin, B. Hoff, TOsmanov, D. Kruchinin, A. Zankevich, DmitriySidnev, M. Markelov, Johannes222, M. Chenuet, a-andre, telenachos, A. Melnikov, J. Kim, L. Ilouz, N. Glazov, Priya4607, R. Tehrani, S. Jeong, V. Skubriev, S. Yonekura, vugia truong, zliang7, lizhming, and T. Truong. *opencv/cvat: v1.1.0*. Version v1.1.0. Aug. 2020. DOI: 10.5281/zenodo.4009388. URL: https://doi.org/10.5281/zenodo.4009388.