

CHAPTER II

LITERATURE REVIEWS

2.1 Literature reviews

2.1.1 Literature Review: Effects of Lighting on Depth Camera

Performance

Depth cameras are commonly employed in indoor robotic and autonomous applications, including environment perception, spatial understanding, and navigation-related tasks. Despite their widespread use, the effectiveness of depth sensing is strongly influenced by surrounding environmental factors, especially lighting conditions. Previous research has shown that variations in illumination can significantly affect depth measurement quality, leading to increased noise, reduced accuracy, and unstable system performance. (J. Gené-Mola, 2020) (G. Halmetschlager, 2018) (P. Chen, X, 2025)

2.1.2 Drone For Inspecting Inside Building With Object Avoidance And Mapping System (Pawarut Karaked, 2023)

This research focuses on evaluating an integrated indoor drone system that combines multiple SLAM methods, EKF-based sensor fusion, mapping, and autonomous navigation with obstacle avoidance. The positioning system remains functional even when partial SLAM failure occurs, confirming the effectiveness of the Extended Kalman Filter in filtering abnormal data. The mapping system produces maps with dimensions consistent with the real environment, indicating acceptable accuracy. Additionally, the autonomous flight and obstacle avoidance system successfully calculates and follows flight paths while maintaining safe navigation.

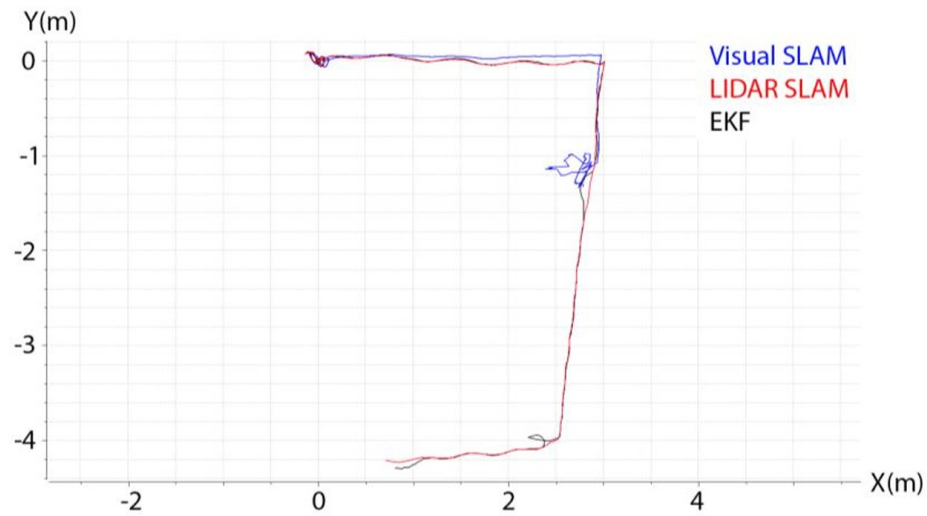


Figure 2.1 The drone's movement path during the positioning system test in case the stereo camera failed to function (Pawrut K., 2023).

In the event that the stereo camera fails to function during testing, a drone will be controlled in a darkened room with controlled lighting. At the start of the test, the room will have sufficient light for the stereo camera to operate. Afterward, all lights in the room will be turned off, making it completely dark and preventing the stereo camera from detecting its surroundings.

The result was that, initially with sufficient lighting, all positioning system, including the drone, functioned correctly. However, when the light was turned off, inaccuracies occurred in the Visual SLAM data. Meanwhile, data from the LiDAR SLAM and EKF remained accurate as in normal circumstances, and the drone itself continued to operate normally.

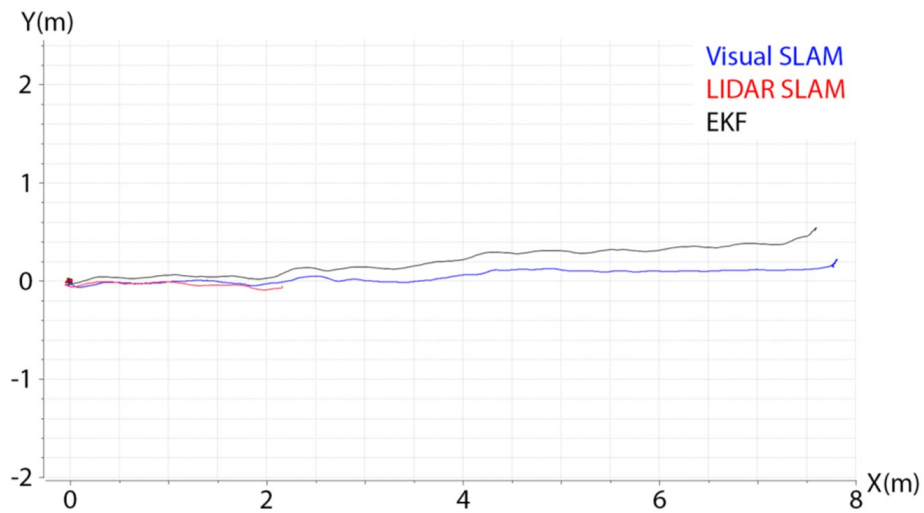


Figure 2.2 Drone trajectory in positioning system test in case LIDAR fails to function. (Pawrut K., 2023).

In the event of LIDAR failure, a test was conducted controlling a drone in a large open room where LIDAR cannot detect all sides simultaneously. In areas where LIDAR could not detect anything at the start of the test, the drone was initially positioned near a wall to ensure the LIDAR SLAM functioned normally. The drone was then controlled to an area where LIDAR could not detect anything, simulating LIDAR SLAM failure.

The result showed that, at the beginning, when LIDAR could still see the wall, the entire positioning system, including the drone, functioned correctly. When simulating LIDAR SLAM damage, Visual SLAM and EKF also worked, even without relying on LIDAR data. A system called Infrared Local Positioning System (IR-LPS) was introduced. This system uses infrared to determine the drone's position, replacing GPS.

2.1.3 LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping (Tixiao ,S ., 2020)

proposed LIO-SAM, a tightly-coupled LiDAR-inertial odometry framework based on factor graph optimization. The system integrates LiDAR measurements and inertial data within a unified smoothing and mapping architecture to achieve high-accuracy state estimation in real time.

LIO-SAM is built upon a factor graph formulation using incremental smoothing techniques. The framework combines. LiDAR scan matching constraints, IMU pre-integration factors, Loop closure constraints and GPS factors (optional, when available)



Figure 2.3 Mapping results (Xu, X, 2022).

Mapping results of LOAM, LIOM, and LIO-SAM using the Walking dataset. The map of LOAM in (b) diverges multiple times when aggressive rotation is encountered. LIOM outperforms LOAM. However, its map shows numerous blurry structures due to inaccurate point cloud registration. LIO-SAM produces a map that is consistent with the Google Earth imagery, without using GPS.

2.1.4 A survey on vision-based UAV navigation (Lu, Xue, Xia, & Zhang)

This research addresses the challenges of indoor autonomous drone operation using a method called Simultaneous Localization and Mapping (SLAM), integrated with an Obstacle Avoidance system. SLAM is a process that identifies its own position and maps its environment using distance data from proximity sensors. This research recognizes that both types of data are applicable in solving indoor autonomous drone problems. The localization component provides the drone's current location, while the mapping component monitors and records the environment. This information is then used in conjunction with the Obstacle Avoidance system, which guides the drone safely to its target location.

2.1.5 Development of Autonomous Electric Golf Cart System for Passenger Transportation Between Building (Wongsathon Angkhem, 2023)

The researchers were therefore interested in developing an autonomous electric golf cart controlled by artificial intelligence, using: Mapping and positioning with data from optical rangefinders; creating movement paths with high-resolution maps; using artificial intelligence to detect objects and identify their positions in 3D; and creating movement paths for obstacle avoidance. This information is then used to make decisions and control the golf cart, controlling speed and turning angle according to the curvature of the movement path to reach the destination and evaluating its efficiency.

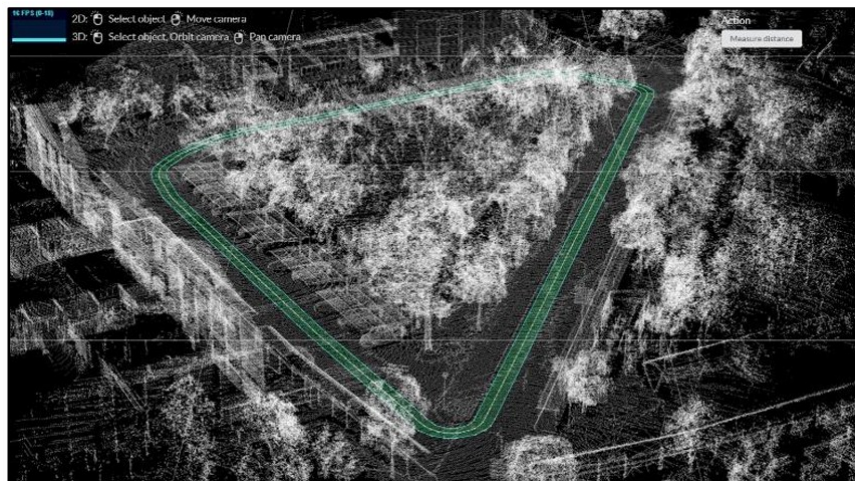


Figure 2.4 An example of creating an outdoor map using 3D LiDAR (Wongsathon A., 2023)

2.1.6 A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation (Mourikis, A., 2007)

The key innovation of MSCKF lies in its state representation and update strategy. Instead of explicitly estimating the positions of visual landmarks within the state vector, the algorithm maintains a sliding window of past camera poses. Feature observations are used to impose multi-state constraints across these poses. IMU data is used for state propagation, while visual measurements provide correction updates.

MSCKF is commonly categorized as a loosely-coupled or filtering-based visual–inertial fusion framework.

Table 2.1 Comparison of the advantages and disadvantages of each method

Method	Advantage	Disadvantage
EKF	It can operate even when one of the sensors is unable to function due to various sensor limitations	The location tracking system lacks accuracy in identifying locations
separation between odometry estimation and mapping (loosely-Coupled Sensor)	High localization accuracy without vision	not fully exploited
Tightly-coupled	Increased computational complexity compared to loosely-coupled systems	ay affect the entire estimation pipeline because all constraints are jointly optimized
Depth camera	Gained dense of information, including color and depth.	Sensitive to light and easily disturbed
SLAM	They are able to perform tasks immediately, have freedom in carrying out tasks, and can perform tasks with precision	High-precision sensors are required, as well as computers with high computing power
3D LiDAR	Captures height, shape, and structure of the environment	Often unnecessary for indoor environment Much higher price
2D LiDAR	Great for localization Much cheaper than 3D LiDAR	Planer laser scanner (Measures distance on a single horizontal plane) Wake for full environment understanding

2.2 Robot

2.2.1 F1TENTH

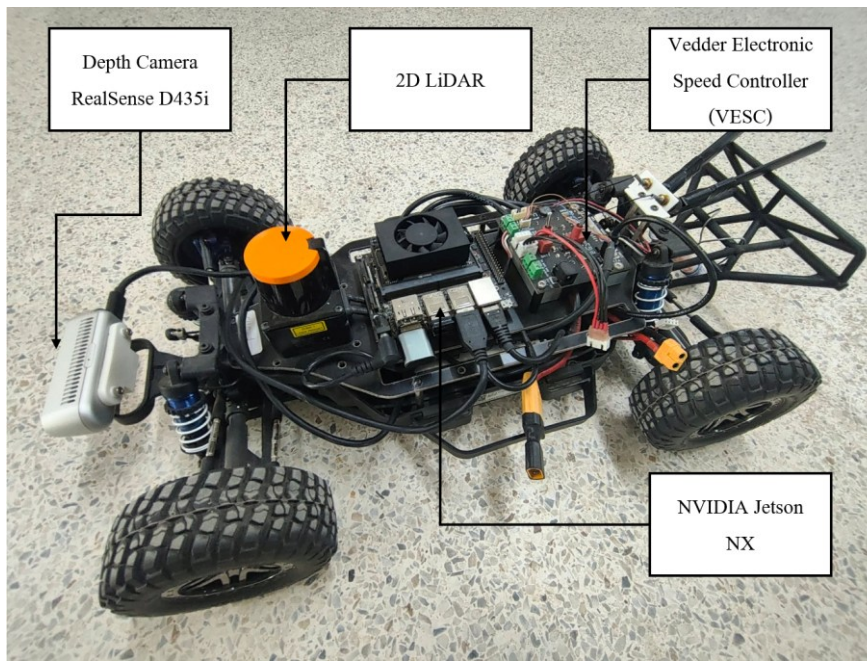


Figure 2.5 illustrates the experimental F1TENTH autonomous vehicle platform used in this study. The system integrates a 2D LiDAR sensor and an RGB-D depth camera for perception, an NVIDIA Jetson NX for onboard computation, and a VESC motor controller for vehicle actuation

F1TENTH is a compact autonomous vehicle platform developed for research and educational purposes in the field of self-driving systems. Built at a one-tenth scale, the platform integrates onboard sensors, computing hardware, and open-source software to support the development and evaluation of autonomous driving algorithms. It is commonly used for studying core tasks such as environment perception, vehicle localization, mapping, path planning, and control, providing a practical and cost-effective testbed for autonomous vehicle research.

The NVIDIA Jetson NX is a core processing board that serves as a computer operating system, including SLAM and control systems, and supports various sensor protocols for connecting different sensors. VESC (Vedder Electronic Speed Controller)

is an open-source motor controller widely used in robotic and autonomous systems for precise control of brushless DC (BLDC), VESC has been extensively adopted in robotics due to its flexibility, robustness, and support for advanced motor control techniques. 2D LiDAR Provides planar laser scans for LiDAR-based SLAM and localization. Depth Camera (Intel RealSense D435i) Used for RGB-D perception, dense point cloud generation, and visual/depth-based SLAM

2.3 Vedder Electronic Speed Controller VESC

It is used to control BLDC motors by adjusting the power supplied to the motor windings using a technique called Field-Oriented Control (FOC). This allows the motor to operate smoothly by controlling the stator current in relation to the rotor's magnetic field. In the Robot Operating System (ROS), data obtained from VESC is used to calculate the robot's position. This is represented by a topic named /odom.

```
---
transforms:
- header:
  stamp:
    sec: 2098
    nanosec: 83442756
  frame_id: odom
  child_frame_id: base_link
  transform:
    translation:
      x: 6.755178242572232
      y: -2.945560400643042
      z: 0.10562776241232164
    rotation:
      x: -0.0029785800181471806
      y: -0.004572191248174758
      z: -0.2475520980636135
      w: 0.9688592167454303
```

Figure 2.6 ROS2 shows a ROS 2 transform message representing the robot pose relative to the odometry frame, including position and orientation information used in the localization process

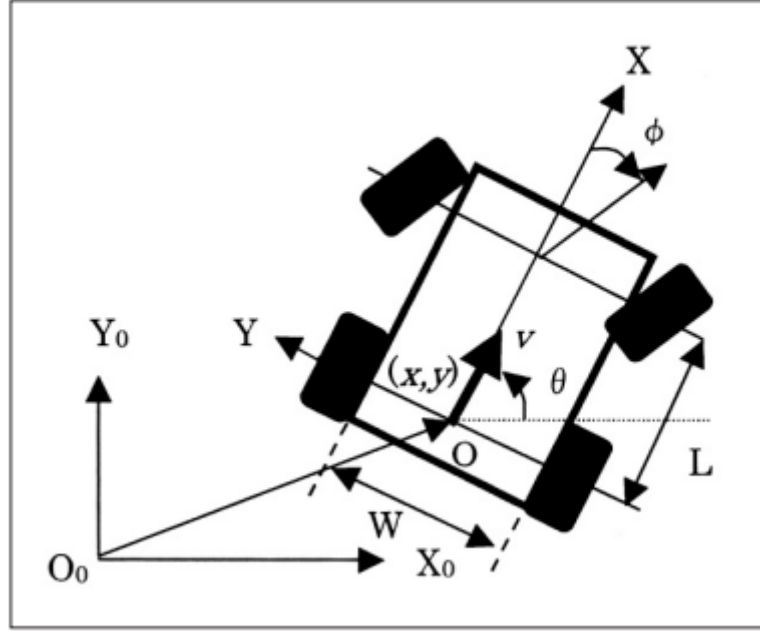


Figure 2.7 Wheel Mobile Robot Model four-wheel mobile (Zunaidi, I, 2006)

illustrates the kinematic model of a four-wheeled mobile robot. The variables x and y denote the robot's position in the navigation coordinate frame, while θ represents the robot's orientation. The steering angle is denoted by ϕ , and v corresponds to the linear velocity of the vehicle. The parameter L indicates the distance between the front and rear axles, and W defines the lateral separation between the left and right wheels.

$$x(k+1) = x(k) + v(k) \cdot \cos(\theta)(k) \cdot \Delta k \quad (2.1)$$

$$y(k+1) = y(k) + v(k) \cdot \sin(\theta)(k) \cdot \Delta k \quad (2.2)$$

$$\theta(k+1) = \theta(k) + \frac{v_l(k) - v_r(k)}{W} \cdot \Delta k \quad (2.3)$$

$$v(k) = \frac{v_l(k) + v_r(k)}{2} \quad (2.4)$$

The robot's planar position and heading are estimated using incremental wheel encoder measurements, as described by Equation (2.6). The position update is obtained by integrating the robot's forward velocity along its heading direction, whereas the orientation update is derived from the difference between the left and

right wheel velocities. The linear velocity $V(k)$ is calculated as the average speed of the left and right rear wheels. Here, k represents the current sampling index, $k + 1$ denotes the subsequent time step, and Δk is the fixed sampling interval. (Zunaidi, I., 2006)

Robot speed and position will be published in the topic /odom via the message nav_msgs/Odometry

Odometry data is not without its challenges. Odometry errors accumulate overtime leading to drift and inaccuracies in the robot's estimated tractor (Thrun, S., 2001) (Grisetti, G., 2010)

2.4 Low level

Low-level control refers to the basic hardware control of robotic systems, involving direct communication with sensors, actuators, and hardware interfaces. This allows the robot to perform basic tasks such as movement and receiving signals from sensors. This research focuses on.

Motor Control: A Vedder Electronic Speed Controller (VESC) is used to control the motors for robot movement. The VESC receives speed commands from the remote control via software and converts them into motor driving signals. The speed depends on user input or automatic commands. For steering, PWM (Pulse Width Modulation) signals are used to adjust the turning radius. Odometry values can be received from the VESC and can be used to approximate the position of the F1 TENTH robot using the odometry data found in the /odom topic.

LiDAR: 2 D LIDAR is installed on the F1 TENTH for sensing the surrounding environment. In low-level operations, this involves measuring the distances of the surrounding environment and converting this data into a usable format for Simultaneous Localization and Mapping (SLAM). In low-level operations, it connects directly to the hardware, receiving distance and angle data from the LiDAR and publishing/scanning it.

RealSense Depth Camera: A depth camera provides a three-dimensional environment. At the low level, it connects directly to the camera hardware to capture depth and Red-Green-Blue (RGB) data and processes this data. The point cloud data,

or depth map, is generated directly from the RealSense camera and published in topics such as `/camera/depth/points` or `/camera/depth/image`. This data is then used for further mapping.

Hardware Abstraction Layer (HAL): This layer integrates system communication with various sensors, including the depth camera, VESC controller, and LiDAR. It utilizes the Robot Operating System (ROS) to isolate hardware-level communication with sensors through built-in drivers and designated topics for high-level communication, such as `/camera/depth/points`, eliminating the need for manual USB communication management.

Communication Protocols (e.g., I2C, CAN Bus, USB): VESC communication utilizes the CAN bus for high-speed, real-time, low-level motor control. This includes programming and sending/receiving data from the VESC via CAN, such as speed commands or distance measurements. The LiDAR unit communicates via I2C or USB at a low level; the driver receives raw data for publication in ROS messages. RealSense communicates via USB on the onboard computer for video and depth data processing.

Real-Time Systems: The F1 TENTH operates in a real-time environment, requiring communication between sensors and the processor. The processing board is the NVIDIA Jetson NX, which demands sufficient and fast processing power.

2.5 Robot Operating System ROS

Robot Operating System (ROS) is an open-source software framework that supports the development of robotic systems by offering a standardized set of libraries, tools, and communication mechanisms. It is built on top of a conventional operating system, most commonly Ubuntu Linux, and enables modular and distributed robot software design (M. Quigley, 2009) (Berka, Z. 2020)

In the F1TENTH platform, ROS serves as the central software infrastructure that integrates sensing, computation, and control components. It manages data exchange and coordination among multiple nodes, including the 2D LiDAR, depth camera, and motor controller (VESC). By using a publish-subscribe communication model, ROS allows these components to operate independently while sharing information in real time. ROS (2024)

The ROS framework consists of several core components that support message passing, hardware abstraction, and system coordination, which are essential for autonomous robotic applications.

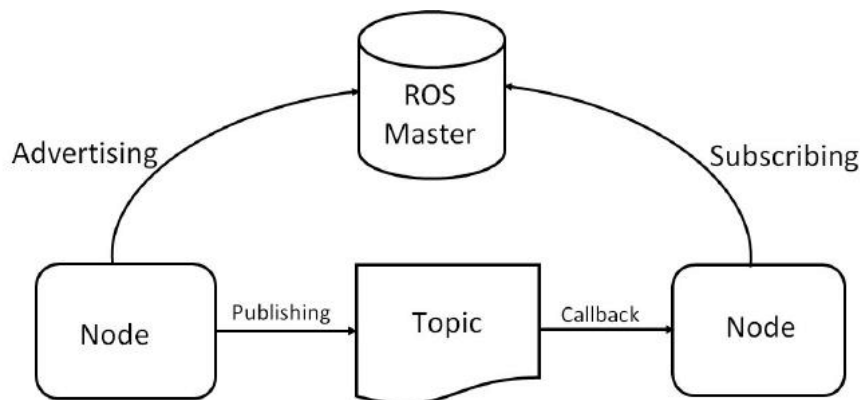


Figure 2.8 The ROS Master Source (ROS, 2024).

Nodes are standalone processes that make up the functional units of a robotic system. Each node is typically designed to handle a specific task, such as processing sensor data or managing actuator control. Communication between nodes occurs through mechanisms like topics, services, or actions. In this communication model, the node that transmits data is referred to as a publisher, while the node that receives the data is known as a subscriber. A single node may perform both roles simultaneously. In addition, ROS allows nodes to be developed using multiple programming languages, including Python and C++, enabling flexible system design and integration.

Topics serve as logical communication channels that allow nodes to exchange data within the ROS ecosystem. Data are transmitted in the form of messages over these channels, enabling nodes to publish information while others subscribe to receive it. A node may both send and receive data through different topics as needed. This publishes–subscribe communication approach operates asynchronously, making it well suited for continuous data streams such as LiDAR measurements, camera imagery, velocity instructions, and motor control signals.

Messages define the data formats exchanged between nodes and act as the fundamental units of communication in ROS. Each message specifies the structure of the transmitted data, which may include numerical values, floating-point variables, or collections such as arrays. For instance, information obtained from a LiDAR sensor can be organized and transmitted as an array within a message.

Unlike topics, which are designed for continuous data transmission, services provide a synchronous communication mechanism based on a request–reply pattern. Each service is defined by two message types: one that carries the request and another that returns the response. This form of communication is typically used for brief, discrete operations, such as querying a node’s status or triggering a specific action within the robotic system.

The ROS Master acts as a coordination service that oversees communication within the system. It keeps track of active nodes, manages information about publishers, subscribers, and available services, and allows nodes to locate one another. Through this mechanism, nodes are able to form connections dynamically during runtime and exchange data efficiently.

ROS structures its software components into packages that group related resources together. A package may include source code written in C++ or Python, along with launch scripts, message definitions, configuration files, and supporting data. These packages serve as the primary unit for organizing and distributing ROS-based software, and they can be shared and reused by other developers through version control platforms such as GitHub.

Launch files provide a convenient way to start multiple nodes and load their associated configuration files at the same time. They are commonly used to specify parameters required by individual nodes, ensuring consistent system setup. These files are typically written using XML or YAML formats, allowing structured and readable system configuration.

The Parameter Server functions as a centralized storage system that allows nodes to access and retrieve shared configuration parameters. It enables nodes to obtain necessary values without hardcoding them into the source code, since parameters can be defined or adjusted during runtime. This approach supports flexible system initialization and simplifies configuration management across the robotic application.

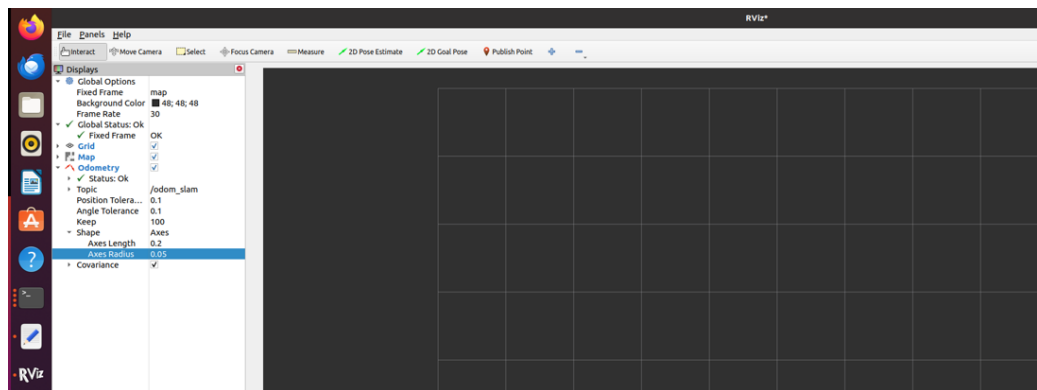


Figure 2.9 RViz2 terminal

RViz (ROS Visualization) is a three-dimensional visualization utility that presents sensor outputs, robot models, maps, and other system data in real time. It allows developers to observe how a robot interprets its surroundings and how different components of the system interact within the operating environment.

Gazebo is a robotics simulation platform that integrates with ROS to model realistic physical environments, including factors such as lighting conditions and sensor behavior. It enables accurate simulation of robot dynamics and sensor outputs, allowing developers to evaluate and refine robotic algorithms in a virtual setting before deploying them on real-world hardware.

In addition to tools that support software development, ROS includes the `tf` library, also known as the Transform Library, which represents various components of a robotic system as independent coordinate frames. This library manages the spatial relationships between these frames, enabling consistent tracking of positions and orientations across the system.

2.6 Simultaneous Localization and Mapping (SLAM)

Simultaneous Localization and Mapping (SLAM) refers to a computational process in which a robot or autonomous system constructs a representation of an unfamiliar environment while concurrently estimating its own position within that space. This capability is essential for autonomous navigation, particularly in situations where global positioning systems are unavailable, such as indoor environments, enclosed areas, or building interiors. SLAM consists of two tightly coupled processes:

localization, which determines the system's current pose, and mapping, which involves capturing and updating information about the surrounding environment. These two processes operate in parallel and continuously inform one another.

In this research, a comprehensive literature review was carried out covering SLAM techniques, LiDAR sensing, sensor fusion methods, depth camera technologies, robotic control systems, and data communication within ROS. The purpose of this review was to establish a solid theoretical foundation for the proposed system. Based on these principles, various testing methodologies were designed, and appropriate data were collected to support a complete and reliable evaluation. The goal was to develop a system capable of accurately estimating position within environments suitable for camera-based operation (REP, 103).

2.6.1 Coordinate frame conventions

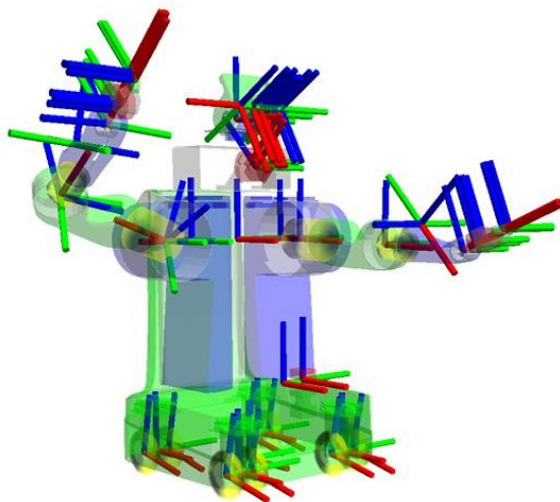


Figure 2.10 Illustration of multiple reference frames used in a robotic system, showing the relationships between different coordinate transformations within the robot structure (Foote, 2013)

The functioning of SLAM is fundamentally based on coordinate frame conventions. A coordinate frame defines a spatial reference system used to describe positions and orientations within an environment. It provides a consistent way to represent motion and spatial relationships and is composed of several key elements:

Origin: A fixed reference point within the coordinate system that serves as the basis for all position measurements.

Axes: Three mutually perpendicular directions, typically labeled X, Y, and Z, which define orientation in three-dimensional space.

Unit of Measurement: A defined scale used to quantify distances along each axis.

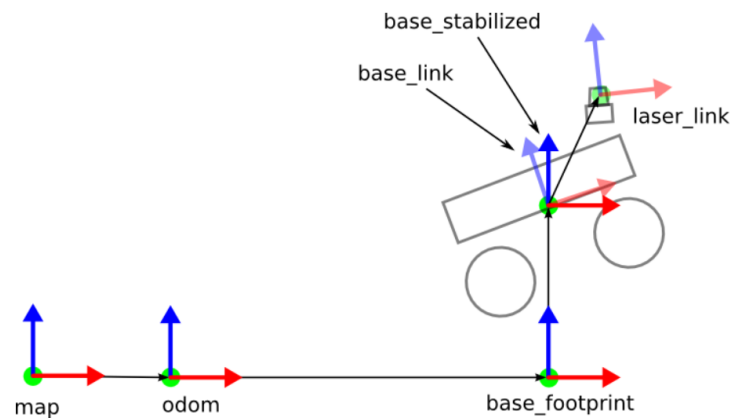


Figure2.11 Diagram illustrating the coordinate frame hierarchy in a mobile robot, including the map, odometry, base footprint, base link, stabilized base, and laser sensor frames, and their relative transformations within the ROS tf framework (ROS) (M. Quigley., 2009).

`base_link` represents the coordinate frame attached to the robot itself and moves consistently with the robot's orientation and direction of motion. The exact location of this frame depends on the specific robot platform and its hardware configuration. ROS (2024).

`odom` refers to a globally referenced frame derived from odometry data that reflects the robot's movement over time. While it provides smooth and continuous position estimates, accumulated drift errors inevitably occur and cannot be corrected retroactively. (Ref 103)

`map` denotes a fixed reference frame associated with the surrounding environment, providing a global coordinate system that is independent of the robot's motion.

2.6.2 Simultaneous Localization and Mapping (SLAM)

In SLAM operation, the `/scan` topic obtained from the LiDAR sensor is continuously used to observe and interpret the surrounding environment. This data allows the system to determine the robot's position relative to its environment by analyzing the geometric information captured in each scan. The processed localization information is referenced within the `/map` frame, which represents a global coordinate system of the environment.

At the same time, the `/odom` frame serves as an additional reference for estimating the robot's motion based on incremental movement. These two coordinate frames operate together to determine the position of the `/base_link`, which represents the robot's physical reference frame. By combining information from both `/map` and `/odom`, the SLAM system can maintain a consistent estimate of the robot's pose over time.

Based on this formulation, the SLAM problem at a given time step t consists of several key elements. One of the primary components is the robot's pose, denoted as X_T which in this research corresponds to the F1TENTH platform. The robot pose can be represented in either two-dimensional or three-dimensional vector form, depending on the application. Furthermore, the sequence of robot poses over time can be combined to describe the robot's entire trajectory, denoted as X_T , where T represents the total number of time steps. This trajectory can be expressed as:

$$X_T = \{x_0, x_1, x_2, \dots, x_T\} \quad (2.5)$$

For a SLAM system to operate, an initial robot pose X_0 is required. In situations where the system begins operation in an environment with no prior information, this initial pose is typically defined as a reference position. All subsequent pose estimates are then computed relative to this initial reference frame.

Odometry refers to data that describes the relationship between the robot's positions at different time steps. This information represents the robot's motion from time $t-1$ to time t , or more generally, the change in position between any two-time instances. Odometry data may include measurements such as traveled distance, velocity, acceleration, or a combination of multiple motion-related variables.

This information is commonly denoted as u_t , and the complete sequence of odometry measurements over time can be expressed as:

$$U_T = \{u_0, u_1, u_2, \dots, u_T\} \quad (2.6)$$

From a theoretical perspective, if the odometry measurements were free of noise, the sequence U_T could be directly integrated to recover the robot's trajectory X_T . However, in real-world applications, odometry data inevitably contains noise and uncertainty. As a result, trajectories estimated using odometry alone tend to drift over time and gradually deviate from the true path of the robot.

Objects represent abstract descriptions of the surrounding environment and may correspond to landmarks, physical objects, surfaces, or other environmental features, depending on the type of sensor used for perception. These objects are used to model the environment within the SLAM framework and are typically assumed to be static. Each object is described by its spatial location, which does not vary over time. The collection of all detected objects in the environment can be represented as:

$$m = \{m_1, m_2, m_3, \dots, m_n\} \quad (2.7)$$

The relationship between the robot's pose at a given time t , denoted as x_t , and the set of environmental objects m is captured through sensor measurements. These measurements describe the relative distances or spatial relationships between the robot and each detected object at time t . Such observations provide information about how far the robot is from the surrounding objects and are essential for correcting pose estimates. This measurement relationship can be expressed as:

$$Z_{n:T} = \left\{ \begin{array}{ccccc} Z_{1:1} & Z_{2:1} & A_{3:1} & \cdots & Z_{n:1} \\ Z_{1:2} & Z_{2:2} & Z_{3:2} & \cdots & Z_{n:2} \\ Z_{1:3} & Z_{2:3} & Z_{3:3} & \cdots & Z_{n:3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ Z_{1:T} & Z_{2:T} & Z_{3:T} & Z_{n:T} & Z_{n:T} \end{array} \right\} \quad (2.8)$$

Based on the components described above, SLAM can be summarized as a process that simultaneously estimates the robot's trajectory X_T and constructs a representation of the environment m_n . This process integrates motion information obtained from odometry data U_T with sensor observation $Z_{n:T}$. The complete formulation, which aims to estimate both the full trajectory and the entire map, is commonly referred to as **Full SLAM**. This relationship can be expressed probabilistically as:

$$P(X_T, m_n | Z_{n:T}, U_T) \quad (2.9)$$

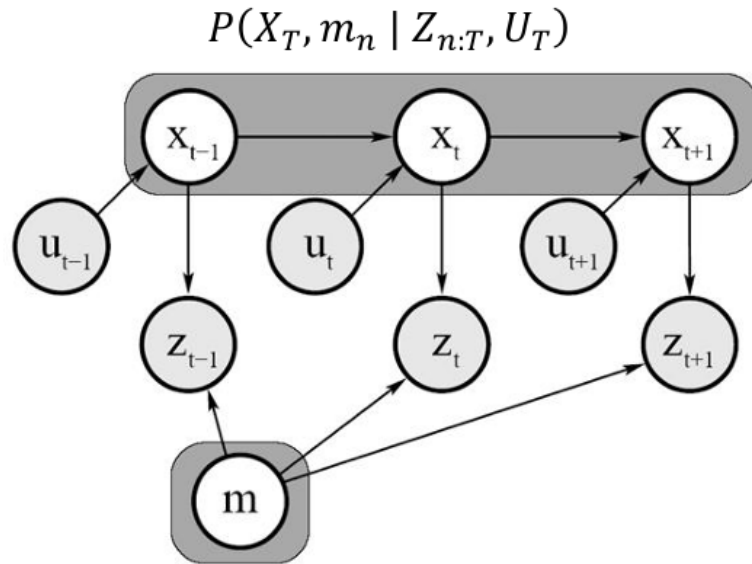


Figure 2.12 illustrates the overall process of Full SLAM, (thrun, 2007)

In Full SLAM, the system computes the entire sequence of robot poses X_T together with all environmental features m_n that can be inferred from the available data. While this comprehensive estimation provides a detailed representation of both motion and environment, it is not always necessary in practical applications. In many scenarios, the robot or user may only require the robot's current pose rather than the complete trajectory history.

To address this requirement, an alternative formulation known as **Online SLAM** has been developed. Unlike Full SLAM, Online SLAM focuses on estimating only the current robot pose x_T instead of the full trajectory X_T . By marginalizing past states,

Online SLAM reduces computational complexity while still maintaining sufficient localization accuracy for real-time operation. This formulation can also be described using a probabilistic framework derived from the Full SLAM representation.

In this research, two SLAM approaches are employed: LiDAR-based SLAM for processing laser range measurements, and odometry data to support motion estimation. However, odometry measurements are inherently imperfect. Errors in odometry accumulate over time, leading to drift and increasing inaccuracies in the estimated robot trajectory, as reported in previous studies. This limitation highlights the importance of incorporating sensor-based observations, such as LiDAR measurements, to correct accumulated motion errors and improve localization reliability. (Thrun, s., 2001)(Trun, s.,2007)

2.6.3 LiDAR SLAM

LiDAR SLAM is a type of SLAM technique that is fundamentally based on a principle known as scan matching. This approach operates using laser-based distance sensing devices, commonly referred to as LiDAR (Light Detection and Ranging). LiDAR functions through a laser scanning process in which distances between the sensor and surrounding objects are measured. This is achieved by emitting laser pulses toward objects and calculating the distance based on the time taken for the reflected light to return to the sensor. (Saeefi. S, 2020)

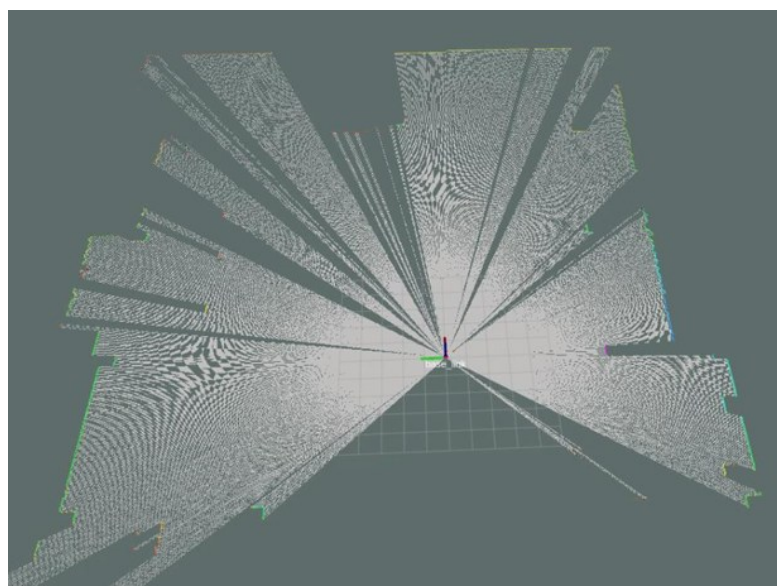


Figure 2.13 Example of LiDAR SLAM usage

There are several types of LiDAR sensors, ranging from single-direction distance-measuring LiDARs, rotating LiDARs capable of scanning surrounding objects in a two-dimensional plane, to advanced LiDAR systems that can capture three-dimensional information of the environment. Despite these differences, all LiDAR sensors operate on the same fundamental physical principle: the propagation and reflection of light.

In LiDAR SLAM, consecutive laser scans are aligned by matching geometric features between scans. This scan matching process allows the system to estimate the robot's motion over time while simultaneously constructing a representation of the surrounding environment. Because LiDAR directly measures spatial geometry, LiDAR SLAM is generally robust to lighting conditions and can operate reliably in environments where visual features are limited or inconsistent.

$$Distance = \frac{\text{speed of light} \times 4 \times \text{Time of flight}}{2} \quad (2.10)$$

2.6.4 Visual SLAM

Visual SLAM is a SLAM approach that relies on image data for processing. In this method, the SLAM system extracts visual features from images, where features typically correspond to regions with noticeable changes in pixel intensity or color, such as edges and corners of objects within the scene. These features are tracked across consecutive frames using feature tracking techniques, which enables the system to estimate motion, construct a map, and determine the robot's position simultaneously.

One advantage of this approach is that Visual SLAM can be implemented using various types of cameras. When stereo cameras are employed, additional depth information can be obtained for each detected feature. Stereo cameras are specifically designed for three-dimensional imaging by capturing two images simultaneously from slightly different viewpoints. The captured image pairs are processed to extract visual features based on surface color, texture, intensity, and structural information present in both images. These features are then used to generate point cloud representations for each view.

Subsequently, corresponding points that represent the same physical locations in the real world are matched between the point clouds. This

correspondence allows the system to compute the distance from the camera to each point and reconstruct a three-dimensional representation of the environment. The resulting 3D information enhances both mapping accuracy and localization performance.

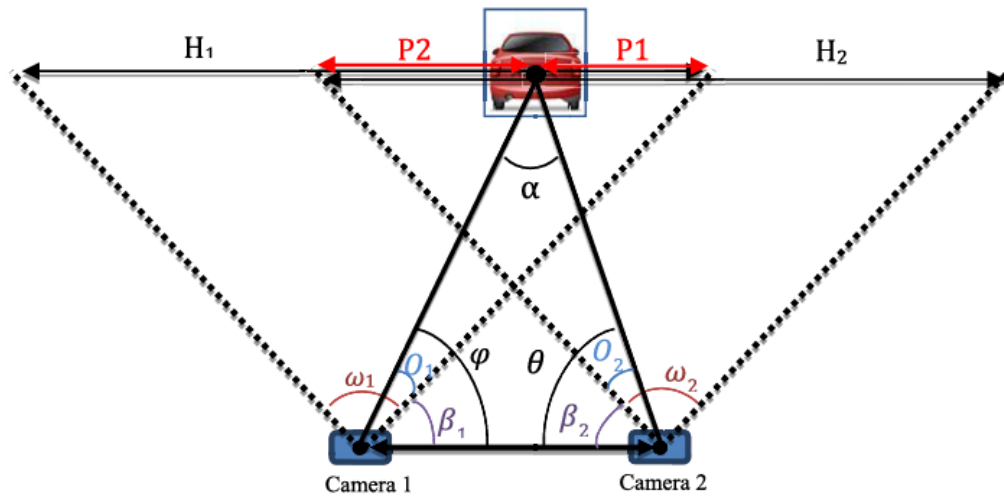


Figure 2.14 Illustration of the angles used for computing the distance (A. Zaarane, 2020)

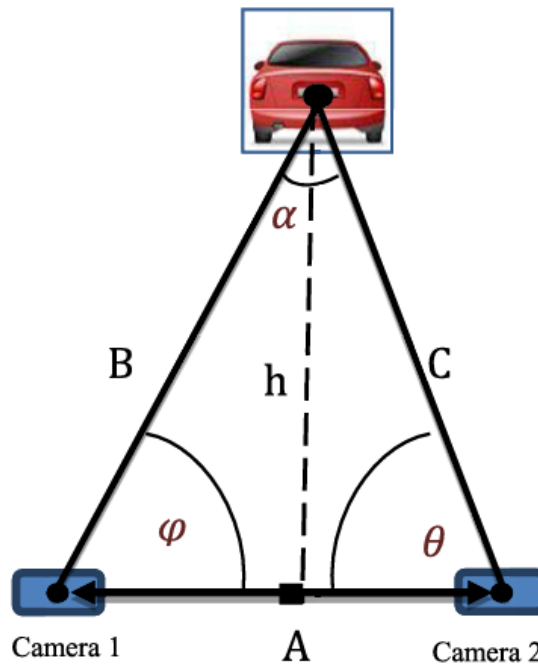


Figure 2.15 Example of two cameras mounted as a stereo camera (A. Zaarane, 2020)

The distance can be calculated following equation.

$$h = \frac{A \sin \left(P_2 \cdot \left(\frac{\omega_2}{H_2} + \beta_2 \right) \sin \left(P_1 \cdot \left(\frac{\omega_1}{H_1} + \beta_1 \right) \right) \right)}{\sin \left(180 - \left(P_2 \cdot \frac{\omega_2}{H_2} + \beta_2 + P_1 \cdot \frac{\omega_1}{H_1} + \beta_1 \right) \right)} \quad (2.11)$$