Multi-objective Mapping and Path Planning using Visual SLAM and Object Detection

by

Ami Woo

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2019

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Path planning of the autonomous robots is one of the crucial tasks that need to be achieved for mobile robots to navigate through the environment intelligently. The robot paths are typically planned utilizing map that is accessible at the time with a certain optimization objective such as to minimizing the travel distance, or time. This thesis proposes a multi-objective path planning approach by integrating Simultaneous Localization And Mapping (SLAM) with a graph based optimization approach and an object detection algorithm. The proposed approach aims not only to find a path that minimizes travel distance but also to minimize the number of obstacles in the path to be followed.

This thesis uses Visual SLAM (VSLAM) as the basis to generate graphs for global path planning. VSLAM generates a trajectory network which is usually in the form of a spare graph (if odometry based) or probabilistic relations on landmark estimates relative to the robot. An object detection algorithm is run in parallel to provide additional information on trajectory network graphs generated by the VSLAM, to be used in multi-objective path planning. The VSLAM, object detection, and path planning fields are typically studied independently, but this thesis links the these fields to solve the multi-objective path planning problem.

The first part of the thesis presents the connections and methodology on using the VSLAM and object detection to generate trajectory network graphs. The nodes are inserted to the graph when a new keyframe is needed in VSLAM. The distance travelled between the nodes is the first criterion to minimize and is computed while traversing. In parallel to VSLAM, the object detection component quantifies the number of objects detected between the nodes. Only the pre-trained objects to detect are quantified and the trained objects in the thesis are cars and trucks. The number of objects are the two additional edge information added to the graph. Later in the thesis, the multi-objective path planning on the generated graphs is presented. The objective of path planning on graph is not just on minimizing the distance to travel but also on minimizing the number of cars and trucks it passes. The proposed design is tested using KITTI dataset which is specialized for autonomous driving and consists of many cars and trucks. The design is not limited to autonomous driving applications, but can be applied to other fields such as surveillance, rescuing, and many more with different objects to detect.

Acknowledgements

First and foremost, I would like to thank you God for your amazing love and power, and blessings. I would like to express my gratitude to all the people who made this thesis possible. I would like to thank my supervisor, Professor Baris Fidan, for his unwavering support and encouragement. Without the advises, both technical and professional, this thesis would not have been possible. I would like to thank the members of my committee, Professor William Melek and Professor Hyock Ju Kwon, for their insightful recommendations to improve this thesis.

Thanks to all the colleagues who I encountered during the masters. A special thanks to DENSO and GDLS research teams. It was a great pleasure to work and gain knowledge from the teams. Thanks to all my friends for their continuous support and encouragement when I was in the low tide of my life. Special thanks to my best friend Jay Woo for his immeasurable support, advice, and encouragement from the beginning.

Lastly, I would like to thank my family for patience, support, and love to pursue my goals and dreams.

Dedication

This is dedicated to my beloved family.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Scope, Motivation and Objective

In the past three decades, there has been great interest and progress in the field of autonomous robots for both researchers and industries. The autonomous robots are to perform in wide range of applications with varying purposes. Consumer robots, such as vacuum cleaner robots, already became commercialized and this type of robots can be applied for similar usages such as to mow lawn or to shovel snow. Some robots are used for industries to transport goods such as in warehouses. They are also used for searching and rescuing such as to retrieve victims from potentially dangerous areas. Another application includes exploration of new environment such as planets. The applications of the autonomous robots are not limited to these and is expected to appear in many more fields. The continuous development of sensing and computation technologies has led to the development and advancement of robots to become truly autonomous. There are still many challenges to be solved to allow fully autonomous robots to appear into the market.

For an autonomous robot to navigate without human support, the robot needs to locate itself with respect to a map and plan paths to steer itself. Without knowing its location within an area, the robot cannot reach its destination nor be aware of where it is heading to. While path planning, the robot would want to find a route that optimizes certain criteria (e.g., minimize distance), which may be single or multiple objectives. For robots to make path planning of such routes, the robot needs to have sufficient knowledge about the environment, be able to locate itself while traversing, and have information about the criteria it is required to fulfil.

Simultaneous localization and mapping (SLAM) is the process of a mobile robot to incrementally build a map of an unknown environment while localizing itself within the map. Since estimating the environment and localizing itself in the environment requires information from its counterpart, solving environment estimation and self-localization problems simultaneously is a difficult task. SLAM has been one of the most actively researched topics in robotics for last three decades. The *classical age* (1986-2004) formulated SLAM in probabilistic world utilizing Extended Kalman Filters (EKF), Rao-Blackwellised Particle Filters, and maximum likelihood estimation [8, 15, 41]. During the subsequent period referred to as *algorithmic-analysis age*, the SLAM community focused on the fundamental properties such as observability, convergence and consistency of SLAM [8]. The current state of art SLAM can efficiently manage thousands of landmarks in the map [22, 82], use visual inertial algorithms to perform visual SLAM (VSLAM) [17, 55].

Despite these achievements and continuous progress in SLAM, not much research has been touched upon utilizing the maps built by SLAM for navigation purposes. One of the common goals for autonomous robots is not to just build a map, but to use the map it builds for navigation purposes. The applications that utilize the map the robot builds, for instance use of the map to plan the future path has received less attention than the map building process. In terms of robot path planning, much of the research is focused on finding optimal routes from prescribed start and target positions. This thesis focuses on planning of a multi-objective path via the map built through SLAM.

1.2 Contributions of the Thesis

The main contribution of this thesis, which is novel to the best of my knowledge, is to development of an approach which allows for a mobile robot to plan multi-objective paths using the map it builds with SLAM. Most of the research on path planning of the mobile robot is focused on finding a path with the shortest distance, and lacks in utilizing the map built via SLAM. This thesis approaches the mobile robot path planning problem by relating VSLAM with an object detection algorithm to build maps and multi-weighted graphs and plan pareto-optimal paths, integrating mapping, object detection, and path planning, which are often studied independently. The proposed solution can be utilized in different fields via optimizing different objectives, such as finding a short path to travel while avoiding predefined obstacles (e.g., fire, heavy rocks or hazardous material) using the acquired map. As the test scenario of the thesis, a city environment is considered with the goal of finding a path that minimizes the distance to travel yet avoids car and trucks.

The overall system architecture is shown in Figure 1.1. This architecture utilizes thesis

implements already developed solutions from the VSLAM and object detection algorithms to generate a multi-weighted graph. The VSLAM algorithm consists of front-end and back-end processes, and produces a map along with a pose graph, whose vertices represent the keyframes and edges the constraints between the keyframes. The object detection algorithm uses a convolutional neural network (CNN) to recognize trained objects given an image. The predefined objects to detect are cars and trucks. Then the thesis combines the pose graph from VSLAM and the object detection information to generate a multi-weighted graph. In addition to edges as the distance information between the keyframes, the multi-weighted graph also contains information about the number of objects detected between the vertices, i.e. the keyframes. The multi-objective path planning algorithm optimizes the multi-criteria represented by this multi-graph and generates paths for navigation. The proposed path planning scheme is adequate for scenarios where a robot is initially guided for exploration such as for map construction or scouting of the environment, and autonomous navigation follows afterwards.

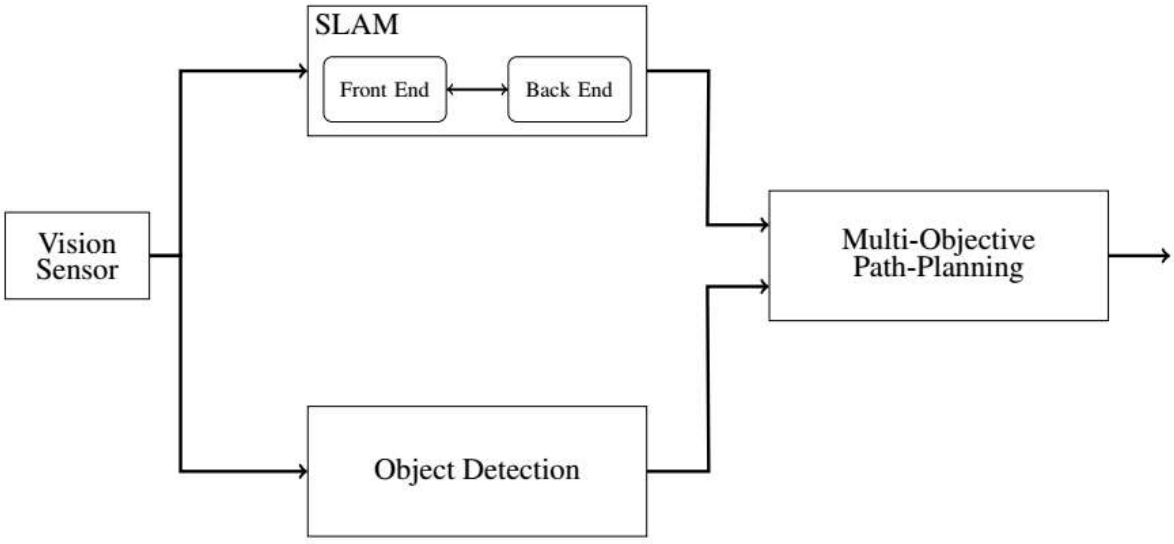


Figure 1.1: System architecture.

1.3 Organization of the Thesis

The outline of the thesis is as follows:

Background information on, VSLAM, image recognition, and path planning along with an overview of what other researchers in the fields worked on is provided in Chapter 2. Further in this chapter, the evolutions of SLAM and current approaches in VSLAM are introduced with background of computer vision techniques, the evolution of object detection and CNN is explained, and single and multi-criteria shortest path planning methodologies are reviewed.

Chapter 3 presents the overall design of the proposed solution. In Chapter 4, the main VSLAM algorithm utilized in the thesis is explained. Chapter 5 discusses the image recognition implemented. Thereafter Chapter 6 presents method to calculate multi-criteria paths. Using the proposed approach Chapter 7 provides the experimental results. Concluding remarks are given in Chapter 8.

Chapter 2

Background and Literature Review

2.1 Simultaneous Localization and Mapping

SLAM incrementally constructs a map with information from sensors while simultaneously localizing itself to the map. One particular approach to update the maps for autonomous robot localization and embed the dynamic environment information in such maps is SLAM. The solution was primarily developed by Hugh Durrant Whtye and John J. Lenonard in 1991 [41] that was based on work by Smith et al. in 1990 [76]. SLAM is considered as one of the key autonomous localization methodologies. When the robot is in an unknown environment, SLAM incrementally builds the map while localizing itself and navigating through the environment. SLAM research has been substantially active for past 30 years in robotics. Depending on the main task the robot is trying to accomplish, SLAM can be distinguished in many ways.

Full SLAM refers to posterior estimation of the entire path [81]. Online SLAM refers to the estimation of the current posterior only [81]. Depending on the types of map that SLAM builds, it can be referred to as feature-based SLAM or volumetric SLAM [81]. The map is built based on points of interest, such as features, using feature-based SLAM, while the map is built at high resolution, which allows photorealistic reconstruction with volumetric SLAM [81]. Other methods of distinction are metric and topological mapping [81]. Metric SLAM have metric information in environment such as metric information from one place to another, while topological SLAM provides qualitative information about the places [81]. The SLAM can also be distinguished by static and dynamic SLAM. Static assumes that the environment does not contain any dynamics of the surrounding, and if anything changes treat the changes as noise [81]. Dynamic SLAM takes into account the

dynamics and sometimes track the changes [81]. Depending on the objective of SLAM, different approaches and variations to tackle the problem have been proposed.

2.1.1 Formulation of SLAM

The formulation of SLAM is depicted in Figure 2.1. At time step k, states of the vehicle including position and orientation are represented by x_k, the control input applied at time step $k-1$ to move the vehicle state to x_k is denoted as u_k. The observable map points, and the measurements by sensors are denoted, respectively by m_k and z_k. Note that the representation of the map can depend on the implemented sensors. In this thesis, the map is assumed as observable points such as location of landmark. The state of the robot, map, control input, and measurement sequences are denoted as $x_{1:k} = \{x_1, x_2, ..., x_k\}$, $m_{1:M} = \{m_1, m_2, ..., m_M\}$, $u_{1:k} = \{u_1, u_2, ..., u_k\}$, $z_{1:k} = \{z_1, z_2, ..., z_k\}$ respectively. The goal of SLAM is to find the $x_{1:k}$, and $m_{1:M}$ from the $u_{1:k}$ and $z_{1:k}$.

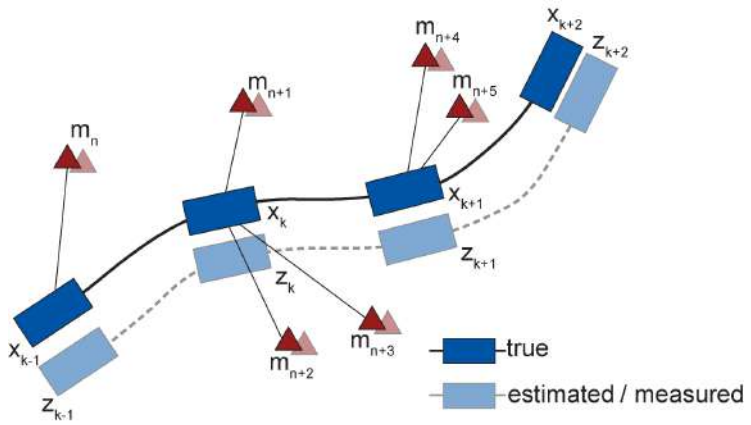


Figure 2.1: Formulation of SLAM.

2.1.2 Paradigms of SLAM

The three main paradigms to solve the SLAM problems are Kalman filter, Particle filter, and Graph-based SLAM. Many effective variations of the three methods have been proposed and implemented in the literatures. The following dynamic state-space model

notation is used to describe the robot motion and sensor model:

$$\begin{aligned} x_k &= f(x_{k-1}, u_k, \epsilon_k) \\ z_k &= h(x_k, m, +\iota_k) \end{aligned} \tag{2.1}$$

where $f(\cdot), h(\cdot)$ model nonlinear vehicle kinematics and measurement correspondingly, and x, m, u, z, ϵ and ι as the state of the vehicle, map, control input, measurement, process noise and measurement noise.

Bayes filter is the basis of the two filter based SLAM. It uses all known inputs $u_{1:k}$, and measurements $z_{1:k}$, to estimate the joint posterior probability density function (pdf) of the states of the vehicle $x_{1:k}$ and map m. This probability distribution is denoted as:

$$p(x_k, m | z_{1:k}, u_{1:k}) \tag{2.2}$$

The belief about the current state prior to measurement and after the measurement is denoted as:

$$\begin{aligned} \bar{\beta}(x_k, m) &= p(x_k, m | z_{1:k-1}, u_{1:k}) \\ \beta(x_k, m) &= p(x_k, m | z_{1:k}, u_{1:k}) \end{aligned} \tag{2.3}$$

Using the Bayes rule, Markov property, law of total probability, Bayes filter algorithm recursively calculates the pdf in two-step as shown in Table 2.1.

Table 2.1: Bayes Filter Algorithm

Prediction:	$\bar{\beta}(x_k, m) = \int p(x_k \| x_{k-1}, u_k) \beta(x_{k-1}) dx_{k-1}$
Update:	$\beta(x_k, m) = \eta p(z_k \| x_k) \bar{\beta}(x_k)$

In Table 2.1 $p(x_k|x_{k-1}, u_k)$ is the motion model, $p(z_k|x_k)$ is the sensor model, and η is the normalizing constant that does not depend on the state. There are many solutions to Bayesian filter for different systems.

Kalman or Extended Kalman filter (EKF) can be the optimal estimation in terms of minimizing the mean square error sense. Kalman and EKF assumes the posterior density is Gaussian distributed with added zero mean Gaussian process noise ϵ and measurement noise ι and its covariance represented by $\mathcal{R}_k$ and $\mathcal{Q}_k$, respectively [80]. EKF SLAM have been the earliest approach and was introduced in [76, 77, 54]. The EKF algorithm has been summarized in Table 2.2, where $\mu = (x, m)$, $\partial f_k = \frac{\partial f(\hat{\mu}_{k-1}, u_{k-1})}{\partial \mu}$, and $\partial h_k = \frac{\partial h(\hat{\mu}_k)}{\partial \mu}$.

The state estimates includes the poses of the vehicle and the features of the environment. As the vehicle moves, the state vector and its covariance matrix are updated. When new

Table 2.2: Extended Kalman Filter Algorithm

Prediction:	
Predicted State:	$\hat{\mu}_{k\|k-1} = f(\mu_{k-1\|k-1}, u_k)$
Predicted Covariance:	$\Sigma_{k\|k-1} = \partial f_k \Sigma_{k-1\|k-1} \partial f_k^T + \mathcal{R}_k$
Update:	
Kalman Gain:	$K_k = \Sigma_{k\|k-1} \partial h_k^T (\partial h_k \Sigma_{k\|k-1} \partial h_K^T + \mathcal{Q})^{-1}$
Estimated Covariance:	$\Sigma_{k\|k} = (I - K_k \partial h_k) \Sigma_{k\|k-1}$
Estimated States:	$\mu_{k\|k} = \hat{\mu}_{k\|k-1} + K(z_k - h(\hat{\mu}_{k\|k-1}))$

feature is observed, it is added to the state vector. EKF SLAM and its variations have been implemented for many mobile robot navigation. In practice, the robots often exhibit nonlinear motion dynamics. Kalman filter and its variant filters do not perform well during non-linear systems, and has many limitations resulting in other two methods to become more popular.

Particle filter is another popular filter which is uses N number of particles to model arbitrary distribution of true posterior [37, 14, 63]. The i^{th} particle pose, x^i, and its normalized weight, w^i are represented as $X = \{x^i, w^i\}_{i=1,..,N}$. These particles are sampled from a distribution such as posterior distribution. However, it is often difficult to sample from the true posterior density, instead the particles are sampled from a proposal distribution $\pi(x_k|X_{0:k-1}, z_{0:k})$, chosen by the designer. The idea behind the particle filter is to use the particles from a proposal distribution to approximate the target distribution (posterior density). To compensate difference between target distribution, importance weight, w^i, is computed. The samples that are less likely represent the state, x , are replaced by more likely samples through resampling process. The algorithm of particle filter is presented in Table 2.3.

Monte Carlo Localization (MCL) is a specific form of a particle filter based localization that is being used widely in mobile robot localization [21]. In MCL, the proposal distribution is the motion model; the particles are propagated through motion model. The weight of the particles are computed via the measurement model. This type of localization, and its variants perform well in low-dimensional spaces, but as the size of the state space increases, particle filter becomes impractical. This is the major drawback of particle filter based localization, as the environment gets complicated, more particles are required for pose estimation and the computational complexity grows.

In 2002, Montemerlo et al introduced FastSLAM to allows less computational power to solve SLAM problem using particle filter [51]. FastSLAM was able to reduce the high

Table 2.3: Particle Filter Algorithm

Sampling:
for $i = 1, ..., N$
  Draw $x_k^{(i)}$ from $\pi(x_k|X_{0:k-1}, z_{0:k})$
  $w_k^{(i)} = \frac{p(x^{(i)})}{\pi(x_k^{(i)})}$
  $X_{0:k}^{(i)} = X_{0:k-1}^{(i)} + (x^{(i)}, w_k^{(i)})$
Resampling:
for $i = 1, ..., N$
  Draw $i \in 1, ..., N$ with probability $w_k^{(i)}$
  Add $x_k^{(i)}$ to X_k

dimensional state space by independently estimating the landmarks. By exploiting the dependences between variables, the trajectory of the vehicle is estimated by samples, while mapping is computed analytically using 2 dimensional Kalman filter.

There are two versions of FastSLAM. FastSLAM 1.0 uses the motion model for proposal distribution, and the particle of weights are computed according to the marginalized observation model [51]. FastSLAM 2.0 takes into account the current observation for the proposal distribution and the particle weight is calculated according to the importance function [50].

Both filtering based localization techniques have been active research areas and important development in the SLAM literature. The current state of art is solving SLAM problem by constructing a graph. The graph-based SLAM approach was proposed in 1997 by Lu and Milios [47]. The basic idea of graph-based SLAM is to represent nodes with the location of robot, and the edges as spatial measurements between the nodes. The edge can be represented by odometry measurements taken from consecutive frames or by observation, which results in virtual measurements about the position of x_j from x_i, when the vehicle observes the same part of measurements. With the graph, the localization of the vehicle is implemented by finding the poses that satisfy the constraint. The architecture of graph based SLAM is shown in Figure 2.2, having two main processes, front-end and back-end. The front-end process interprets sensor data and constructs the graph by defining nodes and edges, while the back-end performs inference on the graph by optimizing it.

The front-end heavily depends on the types of available sensors. The back-end formulates SLAM as a Maximum A Posterioir (MAP) estimation. In MAP, x is estimated

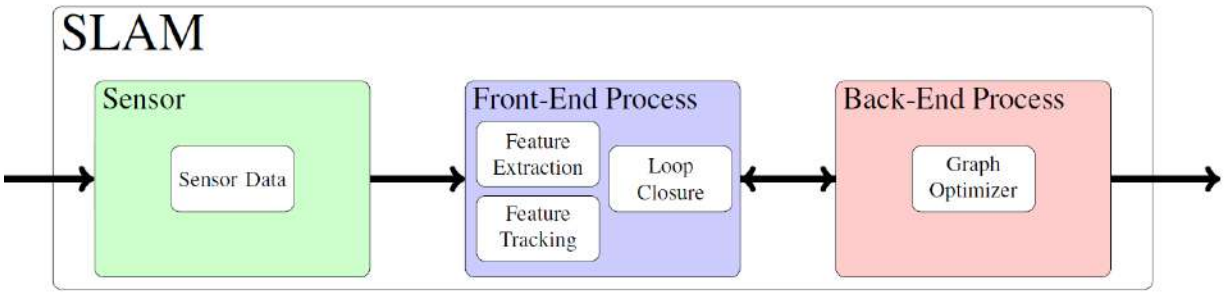


Figure 2.2: Graph-based SLAM architecture.

as

$$x^* = \underset{x}{\operatorname{argmax}}\, p(x|z) = \underset{x}{\operatorname{argmax}}\, p(z|x)p(x) \tag{2.4}$$

Assuming measurements are independent, and measurement noise is Gaussian with information matrix, Ω, (2.4) can be rewritten as the

$$x^* = \underset{x}{\operatorname{argmin}} -log\Big(p(x)\prod_{k=1}^{m} p(z_k|x_k)\Big) = \underset{x}{\operatorname{argmin}} \sum_{k=1}^{m} \varepsilon^T \Omega_k \varepsilon \tag{2.5}$$

where $\varepsilon = \big\|h_k(x_k) - z_k\big\|$. (2.5) minimizes square of error which is suitable for least squares problem. Depending on the nature of available sensors, the meaning of error is different.

The basic graph-based SLAM is shown in Table 2.4, where $\varepsilon_{ij} = z_{ij} - h_{ij}(x_i, x_j)$, the virtual measurement of node j from node i. SLAM algorithms are mainly based on iterative nonlinear optimization assuming that good initial guess is available, and local minima is in the neighborhood. The matrix H is symmetric, positive semi-definite, and sparse. With these porperties, different solvers can be used to compute Δx [13, 29, 44, 40, 36].

Compared to other SLAM techniques, graph-based SLAM is capable generating higher dimensional maps. For more details about graph-based SLAM, and the state-of-art for least-squares error minimization, readers are referred to [8, 28].

2.1.3 Loop Closure

The front end process of SLAM uses a technique, called "loop closure", to recognize previously visited places in order to make SLAM robust and long-term. After some period of

Table 2.4: Graph Based Algorithm

while (not converged)
  for all $(\varepsilon_{ij}, \Omega_{ij})$

$$A_{ij} = \left.\frac{\partial \varepsilon_{ij}(x)}{\partial x_i}\right|_{x=\hat{x}} \qquad B_{ij} = \left.\frac{\partial \varepsilon_{ij}(x)}{\partial x_j}\right|_{x=\hat{x}}$$

$$H_{ii} += A_{ij}^T \Omega_{ij} A_{ij} \qquad H_{ij} += A_{ij}^T \Omega_{ij} B_{ij}$$

$$H_{ji} += B_{ij}^T \Omega_{ij} A_{ij} \qquad H_{jj} += B_{ij}^T \Omega_{ij} B_{ij}$$

$$b_i += A_{ij}^T \Omega_{ij} \varepsilon_{ij} \qquad b_j += B_{ij}^T \Omega_{ij} \varepsilon_{ij}$$

$$\Delta x = -H^{-1} b$$

$$\hat{x}^* += \Delta x$$

time of running the SLAM algorithm, the map or the graphs are continually added and the error accumulates. Nevertheless, if the vehicle visits the places it visited before and recognizes the place, this could reduce the uncertainty and the error of SLAM. Therefore, it is clear that having a reliable and accurate technique is essential as wrong loop closure could result in failure of SLAM.

The loop closure techniques are sensor dependent. The early approaches of loop closure were involved with laser scan [11, 30, 47]. The technique of overlapping the scan with respect to the reference scan is often referred to as scan matching. Various scan matching algorithms were proposed in the last decade, among them one of the most widely used is Iterative Closest Point (ICP) [4]. ICP, which iterates to align the scan have been bases for many variations on scan matching, and the improvements are promising [59, 5, 10, 6].

Relatively recent approach of loop closure involved using camera, the technique often referred to as place recognition or image/visual matching. Some approaches of image matching involved using the feature descriptors while some proposed to use patches of image to recognize the places, also known as Bag of Words (BoW).

2.2 Visual SLAM (VSLAM)

The term Visual Odometry (VO) was first used by Nister [61]. VO is similar to dead reckoning, but it is a more accurate approach and has gained attention in research and industrial field with the advancement of computational power and computer vision [72]. It is a particular case of a technique called Structure From Motion (SFM). The SFM reconstructs 3D scene and camera poses from an unordered set of images and usually is

implemented off-line. VO on the other hand is often accomplished in real time but using an ordered image sequence. In addition to estimating the motion of the camera and mapping, VSLAM also recognizes the already visited places.

Using the images captured at each instant, the goal of VSLAM is to compute relative transformation from one image to the next and to find the full trajectory of the camera. VSLAM is free from any errors that are terrain or vehicle parameter dependent such as wheel variations. It is a much cheaper solution compared to LiDAR based localization which can perform similar tasks but with higher computational power.

As images are full of useful information, there are many approaches to estimate the poses. The two main approaches, (1) feature-based, and (2) appearance based method to approximate the poses have been developed using various techniques. Both approaches estimate motion by estimating the movement of a point (a feature or pixel) from one image frame to the next. As VSLAM uses images, this methods work under an assumption that there is sufficient illumination in the surrounding of the cameras to observe enough texture in the environment. This thesis assumes that the images captured are sufficient enough to observe necessities to do motion estimation of the robot.

2.2.1 Fundamentals

The recent advances in computer and computer vision technology brought back the attention in vision sensors and have been one of the key components in mobile robotics for navigation purposes. With the abundant information that they provide, vision sensors are considered as excellent tools to observe details of immediate surrounding. To use the data from the vision sensors for navigation purposes, it is essential to relate 3D world into 2D images. Various techniques are proposed to recover 3D features from images. This section relates 3D point in the monocular camera to 2D images and discuss the fundamentals of the computer vision.

3D to 2D Perspective Projection

The camera model proposed for the discussion is pin-hole camera model, which is a simple model to approximate the image processing. This model assumes that all points of an object possess sufficient amount of illumination, and radiate light rays to the small openings of pin-hole. Pin-hole only allows small amount of light to pass through, requiring long exposure time compared to larger holes, which allows more light but cause blurry images. The camera model uses lens instead of pin-hole, which controls lens aperture opening and

accordingly the amount of light that passes through. The same projection system as pin-hole camera is applicable; the lights pass through centre-of-point (COP) of the camera and focuses on the image plane. The image projected on the image plane is upside down as the radiated ray pass through. For convenience, the virtual image plane is placed in front of COP; hence, the image projected is not inverted.

Mapping 3D world into 2D world can be accomplished by the following equation in homogeneous coordinates [33]

$$p = \begin{bmatrix} p_u \\ p_v \\ 1 \end{bmatrix} = \gamma \begin{bmatrix} R & t \end{bmatrix} P = \begin{bmatrix} \mathrm{f}_u & \alpha & p_{u_0} \\ 0 & \mathrm{f}_v & p_{v_0} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R & t \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \tag{2.6}$$

where $P = \begin{bmatrix} p_x, p_y, p_z, 1 \end{bmatrix}^T$ is the homogeneous vector of the 3D world point relative to the camera reference frame and $p = [p_u, p_v, 1]^T$ is the homogeneous vector of the projection point on the image plane. The skew parameter α is for non-rectangular pixels. Most of the current cameras have rectangular pixels, for which $\alpha = 0$. γ is the intrinsic camera parameter matrix, f_u and f_v are focal lengths, and p_{u_0} and p_{v_0} are the center points in x and y direction on the image plane, respectively. Normally the focal lengths, f_u and f_v are assumed equal; hence, throughout the chapter they will be denoted as f. Rotation matrix $R \in \mathrm{SO}(3)$ and translation vector $t \in \mathbb{R}^{3\mathrm{x}1}$ map 3D world into camera frame.

The knowledge of the camera parameters, and transformation matrix allow the estimation of points from the real world to the image. This is only possible when the image is on focus. In real life, cameras have distortion which is caused by imperfection of lenses especially when the rays pass through the edges of the lens. In addition, images are exposed to noise. More comprehensive discussions and techniques on such problems and models can be found in [33, 49].

Triangulation

Obtaining depth and features from 3D point to 2D from image is often required to use the camera data for vision based localization. This is usually accomplished via triangulation of two images. Given correspondence pairs, the simplest method to implement triangulation is through stereo normal case. The stereo normal case assumes that the images are fixed and share the same baseline, ξ, as shown in Figure 2.3.

The 3D points can be determined by the intersection of the projected rays as shown in Figure 2.3. The epioplar plane cuts through the image plane and forms an epipolar line

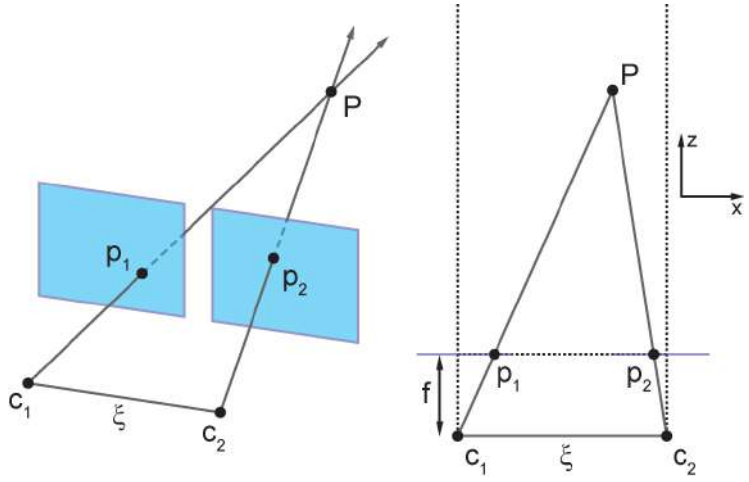


Figure 2.3: Triangulation of stereo vision.

on the image plane. The same point on the left image plane and right image plane must pass through the epipolar line. This justifies the simplicity of the parallel alignment of the camera compared to other geometries; the epipolar line is horizontal, whereas if the cameras are not horizontally aligned, the epipolar line may not be horizontal.

Using the intercept theorem, the depth of a 3D point, $P = \left[p_x, p_y, p_z, 1\right]^T$ is given by

$$p_z = \frac{\mathrm{f}\xi}{\delta} \tag{2.7}$$

where f is the focal length, ξ is the length of epipolar axis, δ denotes the disparity, the difference between a point in x direction from left to the right image ($\delta = p_{u_L} - p_{u_R}$). This shows that the depth is inversely proportional to the disparity. Then, triangle similarity allows to map the 3D real-world points into 2D image, using the following equations:

$$p_x = \frac{p_z}{\mathrm{f}} u_L, \qquad p_y = \frac{p_z}{\mathrm{f}} v_L \tag{2.8}$$

where (p_{u_L}, p_{v_L}) is the corresponding 2D point on the left image plane.

Epipolar Geometry

To estimate the motion of a camera using features, the essential step is to find the corresponding features from one image to another. The simple naive way to solve this problem

is to fully search the entire image. Using the epipolar geometry [49, 33] the search space reduces from 2D (full image) to 1D (a line). Epipolar geometry describes geometric relations in image pairs as shown in Figure 2.4. A point, P, observed by two different image planes through projection center, c_i is shown. For each correspondence in the images, there is an epipolar plane, which is a plane that contains P, c_1 and c_2. Within the epipolar plane, there is an epipolar axis, ξ, which is a line connecting two projection centers, c_1, and c_2. Epipole, e, is the projection center of one image that is projected on another image. For instance, e_1 is the projection of c_2 on image 1. The line connects the projection center to the corresponding point in one image, projected onto another image is epipolar line; l_1 is the projected ray of Pc_2 onto left image. The line contains every parts of a ray from a point, P, to the projection center c_i of one image, and is projected on the second image.

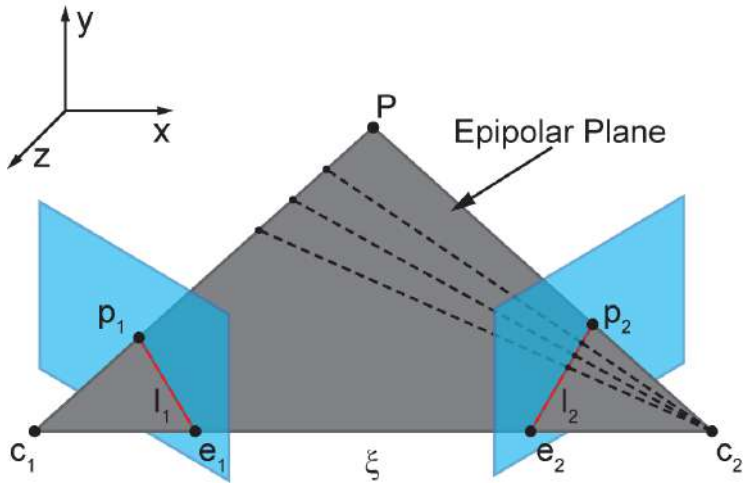


Figure 2.4: Epipolar geometry and epipolar constraint.

Using the epipolar geometry, the epipolar constraint is [49, 33]

$$p_2^T E p_1 = 0 \tag{2.9}$$

where $p_1 = [p_u, p_v, 1]^T$ is a normalized projection point, P, in the first image, p_2 is its normalized correspondence in the second image, and E is the essential matrix. Essential matrix describes the rotation and unknown scale translation of images. There is an overall scale ambiguity; the true scale of the observed scene is unknown unless additional information is available. The essential matrix with multiplicative scalar denoted by the symbol $\simeq$ is:

$$E \simeq \hat{t}R \tag{2.10}$$

where R is the rotation matrix and $\hat{t}$ is a skew-symmetric matrix of the form

$$\hat{t} = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \tag{2.11}$$

where t_x, t_y, t_z denote translations in corresponding directions. Note that essential matrix is used for calibrated cases only. Essential matrix has five degrees of freedom; two for translation and three for rotation. Fundamental matrix, which contains seven degrees of freedom, is used for non-calibrated case [49, 33].

The matrix E computed from a set of epipolar constraint, (2.9), in general is not an essential matrix. Calculation of the actual essential matrix, $\hat{E}$, involves non-linear constrained optimization. A simpler and faster approach, which is discussed in this chapter, projects the estimated E to the essential space. This approach minimizes the Frobenius norm $\|\hat{E} - E\|_f$ [**?**]. Using the singular value decomposition (SVD), E can be factorized in the form

$$E = U diag\{s_1, s_2, s_3\} V^T \tag{2.12}$$

where $s_1 \geq s_2 \geq s_3 \geq 0$ are the singular values of E (square roots of the eigenvalues of $E^T E$), U and V are unitary matrices of eigenvectors of EE^T and $E^T E$, respectively. Then $\hat{E}$ that minimizes the Frobenius norm is given by [**?**]

$$\hat{E} = U diag\{\sigma, \sigma, 0\} V^T \tag{2.13}$$

where $\sigma = \frac{s_1 + s_2}{2}$. Often σ in (2.13) is normalized to 1, to project onto the normalized essential space [49].

Image Similarity

Finding similarities between images or aligning images can be accomplished by searching the full or parts of an image that correspond from one image to the next in a sequence of images. Image-to-image correspondence is often accomplished at the pixel level of images, using the luminous (light) intensity information on each pixel. In practice, a full search of the image is time consuming, instead, part of an image from one image, referred to as template, is used. Usually, the size of templates is much smaller compared to the original image or the image that is being compared. With the small template, a method referred to as template matching, finds the matching point on one image, image 1, with image matrix g_1 to the next image, image 2, with image matrix g_2, by finding a motion vector

$t_g = \left[u_g, v_g\right]^T$ between these two images. Here, the image matrix g_j is a matrix function mapping each pixel index to the value of a certain image quantity, e.g., the intensity I_j. The motion vector t_g maps each point (p_u, p_v) on image 1 to a similar point $(p_{u'}, p_{v'})$ on image 2, that maximizes similarities of the corresponding image matrix value. Finding the similarity of points across image 1 and 2 is defined as

$$\begin{bmatrix} p_{u'} \\ p_{v'} \end{bmatrix} = \begin{bmatrix} u_g \\ v_g \end{bmatrix} + \begin{bmatrix} p_u \\ p_v \end{bmatrix} \tag{2.14}$$

where $[p_u, p_v]^T$ and $[p_{u'}, p_{v'}]^T$ are points on images 1 and 2, respectively, assuming the pixel mapping from image 1 to image 2 is linear. To measure the similarity of images, a particular error matrix is minimized. Some common error matrices include sum of squared difference (SSD) defined as [49]

$$SSD(t_g) = \sum_{p_u, p_v} (g_1(p_u, p_v) - g_2(p_u + u_g, p_v + v_g))^2 \tag{2.15}$$

and sum of absolute differences (SAD) defined as [49]

$$SAD(t_g) = \sum_{p_u, p_v} |g_1(p_u, p_v) - g_2(p_u + u_g, p_v + v_g)| \tag{2.16}$$

Another similarity measure, which is desired to be maximized, is the sum of cross correlation coefficients over all possible locations that is defined as [49]

$$CC(t_g) = -\sum_{p_u, p_v} g_1(p_u, p_v) g_2(p_u + u_g, p_v + v_g) \tag{2.17}$$

Measuring the similarities using (2.15), (2.16), or (2.17) is limited by the fact that these measures are not invariant to the changes in the environment conditions, e.g., illumination. This issue can be solved by normalization, e.g. of the cross correlation coefficients as [49]

$$NCC(t_g) = \frac{-\sum_{p_u, p_v} \left(g_1(p_u, p_v) - \bar{g_1}\right)\left(g_2(p_u + u_g, p_v + v_g) - \bar{g_2}\right)}{\sqrt{\sum_{p_u, p_v} \left(g_1(p_u, p_v) - \bar{g_1}\right)^2 \sum_{p_u, p_v} \left(g_2(p_u + u_g, p_v + v_g) - \bar{g_2}\right)^2}} \tag{2.18}$$

where $\bar{g_1}$ and $\bar{g_2}$ are the mean intensities in each image.

The template matching can be implemented via exhaustive search of the entire image. A more efficient implementation is accomplished by constructing pyramid of images [3]. Using

an image, the pyramid is constructed by subsampling, often by a factors of 2, as shown in Figure 2.5. A popular method of subsampling incorporates a Gaussian filter. An image in the lower level of the pyramid contains more information compared to its immediate upper level image. The template match using the pyramid is firstly implemented on the highest level on the pyramid. This enables initialization for its immediate lower level and the procedure repeats to the original image. Whether it is an exhaustive search or image pyramid method, template method is not invariant to scaling or rotation because both approaches are implemented at a pixel level. The template matching is a practical method for scenarios where there are two images with variations in translation and illumination such as stereo vision cameras.

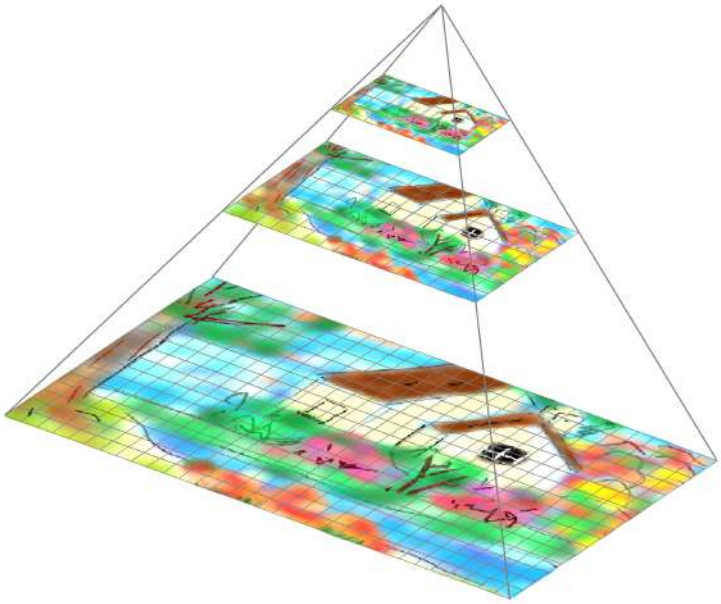


Figure 2.5: Pyramid of Images.

Another frequency domain using Fourier transform. Similar to exhaustive search and a pyramid of images, this method has been mostly implemented for translational shift of the point of interest, but rotations and scales using such technique have also been proposed [12]. Another popular method for finding the motion vector involves optical flow, and is discussed in the following section.

Motion Field and Optical Flow:

Motion field describes the real world motion and assigns 3D velocity vector to every point in an image. Optical flow describes the changes in the intensity of the pixels in 2D image. The goal of the optical flow is to approximate the motion field. The motion field and optical flow do not always correspond to each other, but the rest of the thesis treats them as similar.

In 1981, two famous works that pioneered optical flow estimation were proposed by Horn and Schunck [35], and Lucas and Kanade [48]. Both techniques assume small motion between the two images, and brightness constancy between corresponding points in consecutive images, referred to as brightness constancy assumption:

$$I(p_u, p_v, t) \approx I(p_u + \dot{x}dt, p_v + \dot{y}dt, t + dt) \tag{2.19}$$

where $I(p_u, p_v, t)$ is the intensity value at points (p_u, p_v) at time instant t and the same point after $t + dt$ with displacement $\begin{bmatrix} u_g, v_g \end{bmatrix}^T = [\dot{x}dt, \dot{y}dt]^T$. This assumption can be approximated by Taylor series and further simplified to a result that is often referred to as optical flow constraint and corresponds to [35, 48]

$$I_u\dot{x} + I_v\dot{y} + I_t \approx 0 \tag{2.20}$$

where $I_u = \frac{\partial I}{\partial p_u}$, $I_v = \frac{\partial I}{\partial p_v}$, and $I_t = \frac{\partial I}{\partial t}$, and $\dot{x} = \frac{dp_u}{dt}$, $\dot{y} = \frac{dp_v}{dt}$ are optical flows. This constraint leads to an equation with two unknowns, $\dot{x}$ and $\dot{y}$. To solve for the unknowns, the two tehcniques in [35, 48] employ different approahces. Horn and Schunck imposed further constraint assuming that flow will be smooth for all pixels [35]. The assumption formulates the flow as an energy function that needs minimization. Using techniques adopted from variational calculus to the energy function leads to [35]

$$\begin{aligned} (I_u\dot{x} + I_v\dot{y} + I_t)I_u + \Lambda\Delta^2\dot{x} = 0, \\ (I_u\dot{x} + I_v\dot{y} + I_t)I_v + \Lambda\Delta^2\dot{y} = 0 \end{aligned} \tag{2.21}$$

where Λ is a Lagrange multiplier used to enforce the constraints, $\Delta^2\dot{x} = \frac{\partial^2\dot{x}}{\partial u^2} + \frac{\partial^2\dot{x}}{\partial v^2}$ and $\Delta^2\dot{y} = \frac{\partial^2\dot{y}}{\partial u^2} + \frac{\partial^2\dot{y}}{\partial v^2}$. This method iteratively finds $\dot{x}$ and $\dot{y}$ and results in dense optical flow, i.e., intensity variations in almost all the pixels in the image. In reality, the assumption proposed for dense optical flow is not valid, especially, when there are objects moving in different directions. Instead, Lucas and Kanade assume constant motion in a neighbourhood [48] and producing sparse flow, i.e., intensity variations in only certain pixels of the image. The

flow is computed using least squares fit that corresponds to [48]

$$\begin{bmatrix} \sum_{p_{u_i},p_{v_i}} I_{u_i}^2 & \sum_{p_{u_i},p_{v_i}} I_{u_i} I_{v_i} \\ \sum_{p_{u_i},p_{v_i}} I_{u_i} I_{v_i} & \sum_{p_{u_i},p_{v_i}} I_{v_i}^2 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} -\sum_{p_{u_i},p_{v_i}} I_{u_i} I_{t_i} \\ -\sum_{p_{u_i},p_{v_i}} I_{v_i} I_{t_i} \end{bmatrix} \tag{2.22}$$

The sparse algorithm usually requires feature extraction steps to compute sparse points that need to be tracked. It is challenging to say which method performs better. In practice, the assumptions on both methods are violated; hence, the performance depends on the application.

2.2.2 Feature-based Method

Currently, the feature based methods are the dominant practical solution for VO and VSLAM methods. Instead of using all of the pixels, the motion of the camera is computed by analysing the displacement of the features in feature based methods. Features have properties that make them distinctive from their surroundings in terms of luminous intensity, color, texture or any other characteristics that make them different. A feature can be as simple as a point, line, edge, or a corner, and can be more distinctive such as a trees, buildings, or other vehicles. To use features for VSLAM, the features must be invariant to translation, rotation, scaling, and luminous intensity. Using the features, the feature-based SLAM is to minimize reprojection error which is denoted as:

$$e_{proj} = p_i - \pi_m(RP_i + t) \tag{2.23}$$

where π_m is a projection function that maps 3D point in world coordinates to 2D point in camera coordinates, and p_i is a projection of 3D point P_i.

Feature Detector

To estimate the motion of a robot from the images, frame by frame changes of the vehicle over point of interest, referred to as features, should be determined. The robot must detect such features, usually via a low level image processing operation. Feature detectors search for salient keypoints with respect to local neighbourhood [74]. Some desired properties of feature detector include robustness, invariance under geometric transformation, precise localization of the features and sometimes support of image interpretation. Many feature detectors in VO field have been proposed including corner detectors (Moravec[52], Harris

[31], Shi-Tomasi [73] and FAST [69]) and blob detectors (SIFT [46], and SURF [2]). Depending on the environment conditions and the primary goal, one detector can outperform the other.

In general, the corner detectors are computationally faster than blob detectors. The corner detector introduced by Moravec calculates the variation in the gradient in x and y directions by shifting the rectanguar windows [52, 53]. This can be expressed as [52, 53]

$$S(t_g) = \sum_{p_u, p_v} w(p_u, p_v)[I(p_u + u_g, p_v + v_g) - I(p_u, p_v)]^2 \tag{2.24}$$

where w is the rectangular window function. The displacement in x and y directions is expressed by u_g and v_g, respectively. Intuitively, $S(t_g)$ in (2.24) would be close to 0 in constant areas, while the value would be large in distinctive areas such as corners. Moravec's corner detector works reasonably well under certain conditions, but it suffers from many problems such as noisy and anisotropic response. In 1988, Harris and Stephen improved upon the Moravecs's corner detector, producing one of the most well known corner detectors in current days, and is still being widely used [31]. Instead of shifting the rectangular window function, Harris detector uses a Gaussian as window function. Discrete shift is avoided by considering the partial derivatives using the Taylor expansion. It can be shown that (2.24) can be expressed as [31]

$$S(t_g) \approx \begin{bmatrix} u_g & v_g \end{bmatrix} \nu \begin{bmatrix} u_g \\ v_g \end{bmatrix} \tag{2.25}$$

where

$$\nu = \sum_{p_u, p_v} w(p_u, p_v) \begin{bmatrix} I_u^2 & I_u I_v \\ I_u I_v & I_v^2 \end{bmatrix}, \tag{2.26}$$

I_u and I_v represent the image derivatives. The eigenvalues λ_1, λ_2 of ν distinguish the region as flat, edge, or corner. Small λ_1, λ_2 implies no interesting feature, one large λ_j is an edge, which implies one dominant direction of an image gradient, and large λ_1, λ_2 is a corner, which implies two dominant directions. The popular region of interest for Harris detector are edges and corners. The computation of eigenvalues requires relatively high computation power. Harris detector instead computes traces and determinants, which are less expensive. This allows fast computation and high repeatability. The measure of the features are calculated by [31]:

$$D(p_u, p_v) = \det \nu - \kappa \operatorname{tr}^2(\nu) \tag{2.27}$$

where κ is an empirically determined constant. $D(p_u, p_v)$ is measured for each (p_u, p_v) in the image coordinates and depends only on the eigenvalues of ν. If the value of D is large, it implies that the selected area is a corner. If it is negative with large magnitude, it is an edge, and small $|D|$ represents a flat region. Harris detector is rotation and partially intensity invariant. The major drawback of Harris detector is that it is not invariant to image scale. For example, a feature which is classified as an edge can be classified as a corner, when the image is zoomed out.

A very similar method to Harris detector was proposed in 1994 by Shi-Tomasi [73]. Instead of using (2.27) to measure corner, Shi-Tomasi detector (also referred to as Kanade-Tomasi detector) evaluates corner by [73]

$$D(p_u, p_v) = \min(\lambda_1, \lambda_2) \tag{2.28}$$

If $D(p_u, p_v)$ is smaller than a threshold, it is considered a corner. Similar to Harris detector, it is invariant to rotation but not invariant to image scale.

Blob detectors are slower compared to corner detectors but are more distinctive. In 2004, David Lowe proposed a method that is scale and rotation invariant and partial invariant to illumination changes [46]. The proposed method is known as Scale-Invariant Feature Transform (SIFT) not only detects the features but also describes them. This was the continuation of his previous work on invariant feature detection [45]. The detector and descriptor in SIFT have distinctive roles.

In the feature detection stage, SIFT smooths the original image using different sizes of Gaussian filters, which yields different levels of blurred images. The blurred images are subtracted from one another for which the result is referred to as Difference of Gaussian (DoG) images. The parts of the image that are plain will not be much different among different levels of blurred images. On the other hand, the regions that are more distinctive will have changes in the intensity values. This process is repeated on different scales of images, which are subsampled from the original image, often by a factor of 2. The potential interest points are searched by finding local maxima or minima at different scales. This task is implemented by comparing the nearest 8 neighboring pixels on the selected scale image and 9 neighbors on its immediate adjacently scaled images. These locally distinct features include edges, corners, and blobs. Among the features, edges can be problematic. Given an image, an edge can be easily localized in one direction but can be difficult to find in other directions. Hence, the last step of SIFT detector performs extremum suppression on points which cannot be localized well. Because of this step, SIFT, in general, performs better in blob rich regions rather than corner rich regions.

The discussed feature extraction methods are good, but these are not fast enough for real-time applications with limited computational power. In 2006, Edward Rosten proposed

one solution to the problem known as Features from Accelerated Segment Test (FAST) [69]. FAST is well known for its computational efficiency. For each point $p = (p_u, p_v)$ on image, FAST considers points around p by radius of n, 3 for the example in Figure (2.6). The contiguous pixels in the circle provides information whether they are brighter or darker than $I(p_u, p_v)$ plus a threshold t.

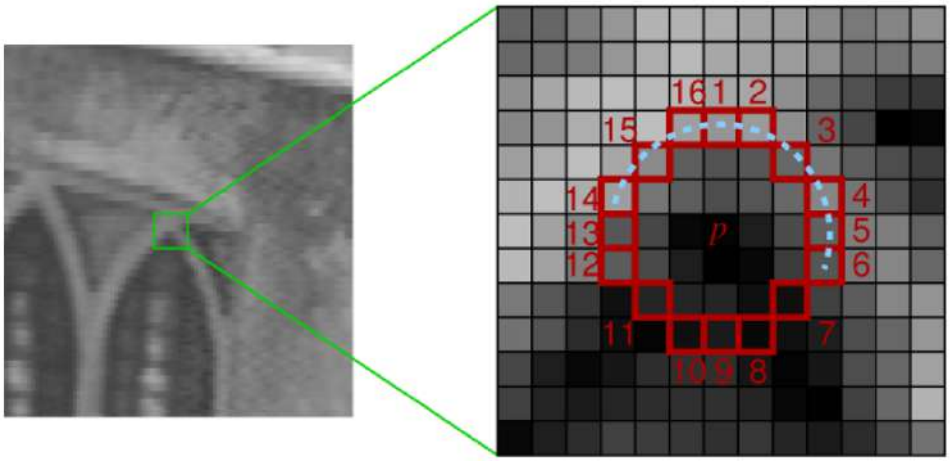


Figure 2.6: Example of FAST for an interest point p. The pixels on the circle with a radius of 3 are considered [69].

For high-speed test, only the pixels at $1, 9, 5, 13$ are examined. p cannot be a corner unless at least three of the pixels are brighter or darker than the $I(p_u, p_v) \pm t$. If it is, the full test is applied to the all pixels in the circle. The detector upto this points exhibits high performance, but there are weaknesses [69]. The weaknesses are improved by machine learned approach, and non-maximal suppression. This thesis does not uses machine learned approach in [69]. One weakness to the detector at this point is it may detect multiple features adjacent locations and is solved by non-maximal suppression. This step first computes a score for each detected corner, denoted to as $\mathcal{V}$. Then the non-maximal suppression step removes any corners with $\mathcal{V}$ value lower than adjacent $\mathcal{V}$. The FAST detector in comparison to other detector is very fast. However, it is not robust to high levels of noise and is dependant on a threshold [69].

The feature descriptor is used to uniquely describe a point of interest. Most of the feature descriptors including SIFT are based on building histograms around the point of interest. SIFT uses 16 x 16 window of image gradient at the selected scale. This window is divided into 4 x 4 subblocks as shown in Figure 2.7. Each subblock encodes gradient

and orientation as a histogram with 8 bins which describes 8 orientations. These values describe the feature and are saved as a vector. Hence, each key point contains 4 x 4 x 8 = 128 dimensions describing the feature. The vector is further processed to make the descriptor rotation invariant, and to reduce the effects of linear and non-linear illumination changes.

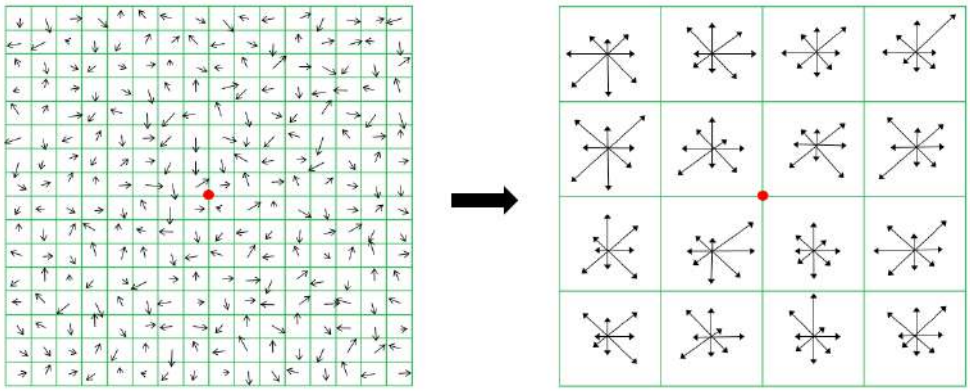


Figure 2.7: Image gradient and key descriptor of SIFT.

SIFT by far is the most robust and have become a popular method in different applications. However, SIFT is very slow. In 2006 faster version that is comparable to SIFT named Speeded Up Robust Features (SURF) was proposed by Bay et al. [2]. The SURF still works on the same principle as SIFT. It builds different levels of images and have detector, and descriptor pairs. However, instead of using DoG, it is approximated with box filters and works based on Haar wavelet responses. This makes SURF more computationally feasible but at the cost of robustness.

Another descriptor which is computationally efficient and can be performed in a real-time application was proposed in 2010 by Calonder [9]. Binary Robust Independent Elementary Features (BRIEF) uses simple binary tests between pixels in a smoothed image patch to describe features. Being a 128-vector for SIFT and 64-vector for SURF, these descriptors requires good storage when it requires to describe a cluster of features. In contrast, BRIEF builds short descriptor requiring only 256 bits, or even 128 bits to produce a comparable result.

The BRIEF descriptor is defined as a vector of n binary tests [9]:

$$f_n(\mathcal{P}) = \sum_{1 \geq i \geq n} 2^{i-1} \tau(\mathcal{P}; I(p_x), I(p_y)) \tag{2.29}$$

where f_n is a feature descriptor, $\mathcal{P}$ is an image patch of size S x S pixels, and $I(p_x)$ is an intensity of a pixel in a smoothed version of $\mathcal{P}$ at $x = (u, v)^T$. The binary test $\tau(\mathcal{P}; I(p_x), I(p_y))$ is as follow [9]:

$$\tau(\mathcal{P}; I(p_x), I(p_y)) = \begin{cases} 1 & \text{if } I(p_x) < I(p_y) \\ 0 & \text{otherwise} \end{cases} \tag{2.30}$$

Choosing the right test points $(I(p_x), I(p_y))$ for the BRIEF descriptor is an important step. Different approaches chooses the test locations. One example is shown in Figure (2.8), which are sampled from an isotropic Gaussian distribution.

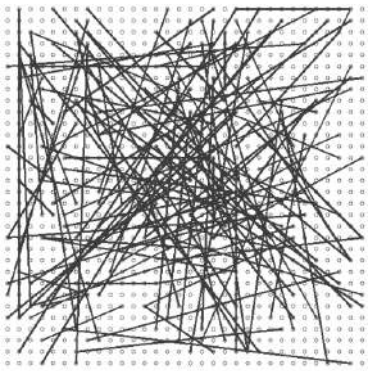


Figure 2.8: Example of choosing the test locations (p_x, p_y) where $(X, Y) \sim i.i.d$ Gaussian$(0, \frac{1}{25}S^2)$ [9]

Feature Matching and Tracking

Correctly identifying corresponding features is the critical task in VSLAM and this allows accurate motion estimation of the vehicle. This task can be categorized into two: feature matching and feature tracking. The feature matching method finds the features from one image and matches them to the next image based on certain criteria, whereas the second feature tracking method tracks certain identified features.

In the absence of feature descriptors, the corresponding features can be found by comparing the similarity of pixel information as discussed in Section 2.2.1. If the descriptors of the features are available, feature matching can be accomplished by comparing the similarity of these feature descriptors. The simple descriptor matching eventually uses the same similarity measures or their variants as highlighted in Section 2.2.1. For SIFT or SURF

descriptors this would minimize or maximize different metrics such as Euclidean distance, depending on the method.

Finding the corresponding feature that has minimum distance among the feature descriptors is called the nearest neighbour method. This method finds the most similar descriptor, and hence, the most alike feature. However, the method could result in false correspondences, which can be mitigated by thresholding.

Thresholding would claim the features correspond, only if the similarity measure is under certain threshold. This yields more accurate feature matching. However, if the surrounding of the robot has many repetitive features such as a building with many windows, false matches are still unavoidable. The distinctive matching can be computed by comparing the ratio of the closest and second closest features. This ratio will be close to 1, if the first and second features are similar. The corresponding feature will be accepted, if the ratio is within predefined threshold. The exhaustive search with the descriptors can be computationally heavy, especially for scenes that are rich with features. Various methods for efficient matching such as hashing [25], and its variants [84] have been proposed.

Another method to match features is to keep track of the features from one frame to another. This approach involves an optical flow estimation with an assumption that sequential images are taken from nearby points. Often the tracking is implemented by Kanade-Lucas-Tomasi (KLT) tracker, which is highly efficient and can be processed in real-time on CPU [48]. The basis of the optical flow is discussed in Section 2.2.1. The KLT tracker aims to find features on the first frame, and then keeps the track of the features from frame to frame. If the error exceeds certain threshold, the tracking of the feature is complete, and if there are not enough features to track, it reinitializes the new features.

Given a set of samples, it is essential to correctly identify correspondences and remove outliers. Determining corresponding features using feature descriptor or feature tracking results in reasonable matches between the samples. However, the points could be exposed to noise and the result could contain mismatched samples. In 1981, Fischler and Bolles proposed a technique to robustly reject outliers, referred to as RANdom SAmple Consensus (RANSAC) [19]. RANSAC is an iterative approach for finding the outliers. The RANSAC samples s number from the data points to estimate the model, such as essential matrix, E. Using the sampled points, model parameters such as R and t are computed assuming the chosen samples are correct points. With the estimated model parameters, the rest of the samples are further processed by classifying it as inliers or outliers by checking if other matches agreed with the chosen samples within certain threshold. The process is recursively applied to find the best model with the highest consensus. To guarantee convergence to

the correct solution, the number of iterations, $\mathcal{N}$, required can be calculated by [19]

$$\mathcal{N} = \frac{log(1 - p_{good})}{log(1 - (1 - p_{fail})^s)} \tag{2.31}$$

where s is the number of corresponding samples needed, p_{fail} is the probability that a point is an outlier and p_{good} is the desired probability to identify inliers correctly. According to the (2.31), the number of trials, $\mathcal{N}$, is greatly influenced by s. The algorithm that requires lower number of s is more feasible and computationally efficient.

Motion Estimation

The main goal of the motion estimation is to find camera motion $T_{k,k-1} \in SE(3)$ between the two sequential images g_k, g_{k-1}. The transformation contains rotation matrix, $R_{k,k-1} \in$ SO(3) , and translation vector, $t_{k,k-1} \in \mathbb{R}^{3x1}$ and can be represented by [33, 49, 72]

$$T_{k,k-1} = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} \tag{2.32}$$

The transformation computes the transformation of the camera poses from one view, C_{k-1}, to the next, C_k, using $C_k = C_{k-1}T_{k,k-1}$. The final goal is to recover the trajectory of the vehicle using the transformation matrix $T_{1:n} = \{T_{1,0}, \ldots, T_{n,n-1}\}$. For convenience of notation, the rest of the chapter denotes $T_{k,k-1}, R_{k,k-1}, t_{k,k-1}$ as T_k, R_k, t_k, respectively. There are three different methods to estimate the motion: 2D to 2D, 3D to 2D, and 3D to 3D. The 2D to 2D estimates the poses of the camera using the points, $p = [p_u, p_v, 1]^T$ from the images, while 3D to 2D uses the points information in 3D, $P = \begin{bmatrix} p_x, p_y, p_z, 1 \end{bmatrix}^T$, and 2D points. The 3D to 3D uses only the points in 3D to estimate the poses.

2D to 2D Method: This method estimates the motion using n features and its correspondences from two images g_1 and g_2. There are many famous solutions for this set up, such as 8-point algorithm [43], and 5-point algorithm [60]. For simplicity, 8-point algorithm is explained in this section.

2D to 2D Method incorporates epipolar constraint discussed in Section 2.2.1. Expanding (2.9) yields [33, 49]

$$\begin{aligned} &p_{u_1}p_{u_2}e_{11} + p_{v_1}p_{u_2}e_{12} + p_{u_2}e_{13} + \\ &p_{u_1}p_{v_2}e_{21} + p_{v_1}p_{v_2}e_{22} + p_{v_2}e_{23} + \\ &p_{u_1}e_{31} + p_{v_1}e_{32} + 1e_{33} = 0 \end{aligned} \tag{2.33}$$

where (p_{u_i}, p_{v_i}) is a normalized point on each image plane, and e_{ii} is each element in essential matrix, E. Defining the two vectors, $a = [p_{u_1}p_{u_2}, p_{v_1}p_{u_2}, \ldots, 1]^T$, $E(:) = [e_{11}, e_{12}, \ldots, e_{33}]^T \in \mathbb{R}^9$, (2.33) corresponds to:

$$a^T E(:) = 0 \tag{2.34}$$

This is one linear equation for one correspondence in image pairs. With n multiple correspondences for the same image pairs results in n linear equations which can be expressed as

$$\chi^T E(:) = 0 \tag{2.35}$$

where $\chi = [a_1, a_2, \cdots, a_n]$. The first correspondence is $a_1^T E = 0$, second is $a_2^T E = 0$ and so on. For 8-point algorithm (2.35) is unique (up to a scaling factor), if there are at least $n \geq 8$ correspondences [43]. By solving the system (2.35) through different techniques such as via singular value decomposition (SVD), E can be computed.

The second part of the algorithm involves retrieving the rotation and translation parts of the images from the estimated E. Unless additional information such as information about absolute distance of any part in the image is available, the recovered translation is only up to scale. The rotation and translation that can be decomposed from essential matrix is [49, 72] :

$$\begin{aligned} R &= \pm UW^T V^*, \\ \hat{t} &= \pm UW diag\{1, 1, 0\} U^T, \\ W^T &\in \left\{ \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right\} \end{aligned} \tag{2.36}$$

where U, V^* are from SVD as discussed in 2.2.1. The sign of E cannot be recovered. In principal, each E yields two possible assignments of R and $\hat{t}$ as shown in (2.36). Therefore, there are four possible solutions for R and $\hat{t}$. The correct solution can be identified by selecting the two solutions with $\det R = 1$, then triangulating the points, and selecting the solution with largest number of points in front of the camera.

3D to 2D Method: This method estimates the pose from features in 3D, P_{k-1}, and its correspondences 2D points, p_k. The goal of 3D to 2D motion estimation is to find

transformation, T_k, that minimizes the image reprojection error, i.e., [72, 20]

$$\underset{T_k}{\operatorname{argmin}} \sum_i \|p_k^i - \hat{p}_{k-1}^i\|^2 \tag{2.37}$$

where $\hat{p}_{k-1}^i$ is the reprojection point of P_{k-1} into image g_k with transformation T_k and p_k is its correspondence 2D point in image g_k. This problem is often known as Perspective-n-Point Problem (PnP). There are many PnP solvers such as direct least squares [34], nonlinear minimization using levenberg-Marquadrt optimization [70], and minimal case invovling three correspondences known as P3P [38]. P3P solution is a robust and performs well with RANSAC outlier rejection [19]. This involves a large number of elimination steps. For simplicity, P6P is presented in this section (works with $n \geq 6$). 6 point correspondence results in a linear system of equation in $LT' = 0$ form of [9] :

$$\begin{bmatrix} 0 & -P_1^T & p_{v_1} P_1^T \\ P_1^T & 0 & -p_{u_1} P_1^T \\ \vdots & \vdots & \vdots \\ 0 & -P_6^T & p_{v_1} P_6^T \\ P_6^T & 0 & -p_{u_1} P_6^T \end{bmatrix} \begin{bmatrix} T'^T_1 \\ T'^T_2 \\ T'^T_3 \end{bmatrix} = 0 \tag{2.38}$$

where $P_i = \begin{bmatrix} p_{x_i}, p_{y_i}, p_{z_i} \end{bmatrix}^T$, and (p_{u_i}, p_{v_i}) are 3D and 2D points of feature i respectively, and $T' = \begin{bmatrix} R \mid t \end{bmatrix}$. T'^T_j is the jth row of T' matrix. The elements of T' can be found by computing the nullvectors of L. The rotation matrix R, and translation vector t can be retrieved from T'.

The 3D to 2D Method works both on monocular and stereo vision. For monocular vision, triangulation technique is required to observe 3D points. This approach requires three views to compute motion estimation. The transformation of correspondences are first estimated by 2D to 2D Method using two views then followed by 3D to 2D Method.

3D to 3D Method: As opposed to 3D to 2D Method which minimizes the reprojection error to estimate the motion, 3D to 3D Method estimate the motion by minimizing the 3D feature position errors, i.e., [72, 62]

$$\underset{T_k}{\operatorname{argmin}} \sum_i \|P_k^i - T_k P_{k-1}^i\| \tag{2.39}$$

where P_k, P_{k-1} are the 3D points of a feature and its correspondence, and T_k is the transformation matrix. The method is mostly used for stereo vision as the depth can be easily

observed by triangulation of features at each instant. The same method can also used for LiDAR data and its point cloud sets.

Similar to other two methods, many solutions have been investigated to estimate the poses using 3D points. A famous approach considers when number of correspondences is $n \geq 3$ [1, 72]. One approach is to estimate the translation of points as [1, 72]

$$t_k = \bar{P}_k - R\bar{P}_{k-1} \tag{2.40}$$

where $\bar{P}_k, \bar{P}_{k-1}$ are the centroid of the features in the feature set, and R is the rotation matrix. The rotation matrix can be computed by SVD as [1, 72]

$$\begin{aligned} (P_{k-1} - \bar{P}_{k-1})(P_k - \bar{P}_k)^T &= U\mathcal{S}V^* \\ R_k &= VU^T \end{aligned} \tag{2.41}$$

Place Recognition

The advancement in computational power and recognition of computer vision in recent years allowed improvements and contributions to place recognition field in recent years. Place recognition allows loop closure, which separates VSLAM from pure VO. The recognition of previously seen place reduces the uncertainty, drift and error. A popular method of place recognition for feature-based method is via BoW.

BoW is a method which represents an image with sparse numerical vector. Bow uses local descriptors as a training dataset and clusters the feature descriptors. These clusters are referred to as "words" in BoW. Different descriptors and clustering methods can quantize the words differently. One popular approach is using $k - mean$, which clusters the descriptors into k words. Each word represents the centroids of each cluster. For comparison, all the descriptors are compared with the centroids to find the nearest neighbor. Using these words, each image can be represented with the histogram of the words.

2.2.3 Direct Method

The appearance based (also referred to as direct or featureless method) have recently gained interest from the field of VO as opposed to feature based method. This method eliminates preprocessing steps such as feature extraction described in the previous section. Instead, it uses displacement of every pixel intensity from sub or full image to recover the pose by minimizing an error called photometric error. This process is more robust and accurate

in low textured environment. Direct method that utilize all pixels on the image results in dense map [58]. There are other approaches such as considering only a subset of data such as pixels with high gradients to build semi-dense map [17]. Even smaller set of pixels can be utilized in direct method to results in sparse maps [16].

The direct method requires depth estimates associated with each pixel. This requires minimizing photometric error which can be formulated as :

$$e_{photo} = I_i(p) - I_j(\pi_m(p, D(p), T_{ij})) \tag{2.42}$$

where π_m is a warp function which projects 2D point from an image with its corresponding inverse depth value, $D(p)$, into a new image by the transformation of the camera T_{ij}

While high computational power is required to build dense map and estimating camera motion, real time application has been done in spatially dense reconstructions. One of the first real time dense reconstruction was proposed in [57], which estimates depth map by minimizing an error function which is denoted as :

$$\min_d \sum_{i=1}^{n} \|I_1(p) - I_i(\pi_m(p, D(p), T_{ij})\| dp + \lambda \int \|\nabla d\| dp \tag{2.43}$$

where d is the depth map, p is homogeneous coordinates, $T_i \in SE(3)$ is the rigid body motion of the camera between frame 1 to frame i, and λ is a regularization parameter. A similar idea was carried out in Direct Tracking and Mapping (DTAM) algorithm [58]. DTAM uses monocular camera to estimates motion and builds dense map by minimizing photometric error of intensity. The trajectory of the camera is tracked by aligning images. The DTAM algorithm works well in real time but is very computationally expensive as it relies on GPU.

A method that works in real time using CPU became promising with approach that uses semi-dense formulation. A popular state of the art for semi-dense method is Large Scale Direct (LSD) SLAM [17]. LSD SLAM is designed to work in large scale environments, and builds semi-dense map based on certain keyframes. The algorithm select keyframes and estimates depth over pixels that have sufficient intensity gradients.The motion estimation of camera is done using direct image alignment and geometry is estimated using semi-dense depth maps. The tracking and depth map estimation both involves keyframes. The information on keyframes are updated with frames that are not keyframes. New keyframe is promoted as a keyframe when there are not enough information to be compared.

The main limitation of dense and semi-dense approaches compared to a sparse direct approach is the computational complexity that forbids to jointly optimize in real-time

structure and cameras, which reduce the achievable accuracy of these methods. In general direct optimizations only work if the initial seed for the optimization is close to the optimal, due to the nature of the photometric error. The reprojection of a pixel has to be close to the optimal projection so that the intensity gradients on the image can guide the optimization to its true location. To mitigate this problem, direct methods use image pyramids, but still the basin of convergence is narrower than for feature-based methods. This makes these methods more sensitive to rolling shutter or low frame-rate. Finally direct methods cannot provide initial solutions to geometry problems and rely on features to detect loops, compute the associated drift, or relocalize the camera.

2.2.4 Graph Optimization

The VSLAM allows not only the transformation between the consecutive frames, but as well as transformation between any time step if the frames share commonly observed points. With these information, the pose graph of the VSLAM can be improved.

Pose-Graph Optimization

The pose graph's vertices represent camera poses from the VSLAM and its edges are the constraints between the poses. The error function about edge constraints can be represented by:

$$\sum_{e_{ij}} \|C_i - T_{e_{ij}} C_j\| \tag{2.44}$$

where e_{ij} and $T_{e_{ij}}$ represents edges and transformation between frame i and j respectively. The optimization involves finding camera poses that minimizes the cost function. Since the error term contains rotation components, this involves non-linear optimization, such as levenberg-Marquadrt optimization [70].

Bundle Adjustment

Bundle Adjustment is similar to pose-graph optimization but it considers not only the pose parameters but as well as landmark, or map points as well. It is the core optimization for modern feature-based SLAM. The error function to be minimized for bundle adjustment is:

$$\sum_{i,k} \|p_k^i - w(P^i, C_k)\| \tag{2.45}$$

where p_k^i is the ith image point of 3D point P on image i. The point could be landmark, or feature points on the ith image. w is a warp function which projects 3D points according to the camera pose C_k. Similar to pose-graph optimization, this involves non-linear optimization.

2.3 Image Recognition

The field of object detection prior to 2012 is quite different from what is observed today. The image recognition used to be done in two steps: feature extraction followed by feature classifier. The object to be detected had to be accurately represented as a vector of features, then was used to train a classifier. For multiple object detection, a more general feature which fit different object had to be found. Determining features was a complex process and the result produced was unsatisfactory accuracy.

With improved graphics cards and interfaces, the work presented in 2012 ImageNet competition started a new era in image recognition field. The work presented by Krizhevsky proved potential of Convolution Neural Network (CNN) in image object detection [39]. Unlike previous attempts, powerful GPU with sufficient training dataset became a new prominent solution in the classification problem. Much previous work was revised to fit image classification problem. From 2012 to 2014, the focus of CNN in object detection was on locating the object within the image and classifying it. In 2014, the research shifted towards multiple object detection.

2.3.1 Artificial Neural Network (ANN)

ANN is an information processing paradigm which is primarily inspired by the biological neural networks. Unlike conventional problems, the goal of ANN is to solve a problem as human brains do. An ANN is based on a collection of artificial neurons. An artificial neuron receives many signals, process them and then signal the output to other connected neurons. The inputs are real numbers and are weighted by w_i , the output is computed by some non-linear functions of the inputs. The weights signify the strength of the signals. ANN learns to adjust these weights from the training.

An ANN usually consists of multiple layers. The first and last layer is the input and output layer respectively. The layers between the first and last are referred to as hidden layers. Each layer consists of many neurons, and in general ANN network is fully connected. The information is passed between the layers as the output of the previous layer becomes

the input of the preceding layers. As the information propagates, each layer makes smarter decisions. ANN trains the weights using the errors between the prediction and the truth. A cost function defines the error, and during the training process, the errors are gradually decreased.

2.3.2 Convolution Neural Network (CNN)

Convolution Neural Network (CNN) is one type of deep neural network which is most powerful in image processing tasks. CNN takes an image as an input and its layers process visual information. Two traits that distinct CNN from traditional neural network are its partial connectivity and pooling operation.

Unlike other neural networks, it is impractical to utilize fully connected layers in image classification. The fully connected layers do not account for the spatial context of layers. A pixel on the image has more meaningful information if its neighboring pixel information is also available. This is where CNN comes into action with its partial connectivity.

There exist different kinds of layers in CNN, one of them being a convolution layer. Instead of having fully connected neurons as an input, in convolution layer, the input is 2D neurons in a rectangular or square shape. The input to the first layer is the pixel intensities in an image. A feature filter slides down on an image vertically and horizontally to produce a feature map. Each connection from the first layer to the hidden layer corresponds to weight, and bias. These weights and bias are learned to detect the features. During the training process, the filters search for a specific pattern while the testing stage checks to see if the patterns exist within the image. Different filter extracts different filters. Hence in practice, many feature filters are used to learn different patterns.

Following the convolution operation, the spatial context of feature maps is reduced by an operation called pooling. The output of feature maps is condensed during the pooling process. Two common pooling methods are average and max-pooling. Using a filter, average pooling outputs the average value inside the filter. Equivalently, the output of the max-pooling is the maximum value from the pooling field.

The result of several convolution layer detects different features and as proceeds CNN detects more complex features. From recognizing a single line to picking up dogs and bicycles, CNN is able to classify objects. This classification happens in a fully connected layer.

Since 2012, CNN became one of the major keys for image classification problem. The solution introduced by Krizhevsky known, as AlexNet outperformed other previously pro-

posed algorithms. It classified 1.2 million images into 1000 different classes with 16% error using 5 convolution layers and 3 fully connected layers [39].

A team from Google also participated in ImageNet 2014 challenge which introduced GoogLeNet [79] and later improved version referred to as Inception v4 model [78]. This model involved a layer setup called an inception module, which involved much fewer parameters compared to AlexNet with better accuracy. However, it is a very complex model to understand and even harder to modify and tune. The runner up of 2014 challenge was VGGNET [75], which is an extension of AlexNet but more efficient and deeper. It is known for its simplicity to understand and modify. It uses a 3x3 convolution layer with max pooling methods for feature extraction. VGGNet is often more preferred method compared to GoogLeNet due to its more straightforward structure.

General Object Detection

There are different approaches to how CNN is used for classification. One method is to generate possible regions of object locations then running a classification solution. Region-based CNN (R-CNN) [27] with its variants proposes independent candidate regions. It is structured with a fixed size neural network. R-CNN first extracts 2000 class independent regions from the input images and the candidates are sent to CNN to detect classes as shown in Figure 2.9. In terms of classification, R-CNN performed well, but there can be some localization errors. The main limitation of R-CNN is its slow processing time. Speed up version of R-CNN known as Fast R-CNN was proposed and increased its performance by a factor of 10 [26]. Instead of computing features for every region, Fast R-CNN computes feature maps once to use it to classify different regions. Even with great success in increasing the performance, Fast R-CNN still could not be run in real time. Faster R-CNN saw the weakness of Fast R-CNN for creating region candidates [66]. Faster R-CNN uses region proposal networks to solve the problem and the detection time reduced up to 7 frames per second with powerful GPU [66].

Unlike R-CNN method, there is a method which detects objects and classifies them at one time. You Only Look Once (YOLO) [64], and Single Shot Multibox Detection [42] are two known methods that can detect objects in real time. Both approaches predict the fixed size of objects and only the estimates with high confidence are returned. The slow process of region proposals then classification are avoided which greatly speed up the process.

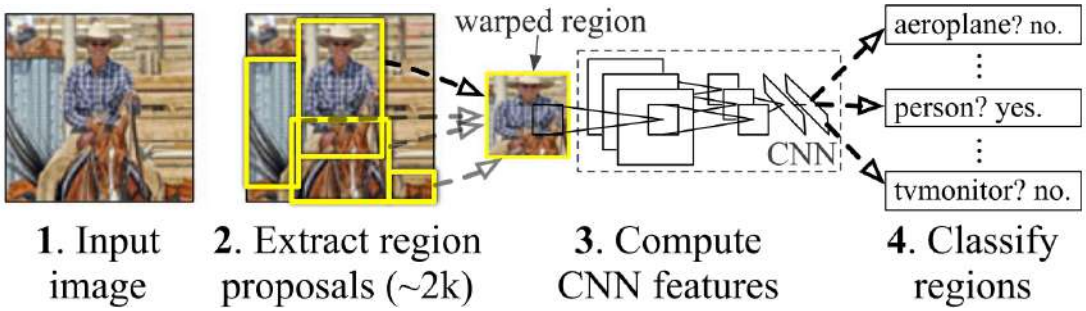


Figure 2.9: Overview of R-CNN [27]

2.4 Path Planning

Finding a path for a robot is one of the most important aspects in mobile robot navigation research. Starting from a prescribed start location, the path planning task is to find a route to the desired goal while fulfilling certain optimization criteria. Depending on the application and what is available to the robot, the planning problem can be categorized into (1) closed-loop planning and (2) open-loop planning. The robot senses the environment and takes appropriate actions based on what is observed in the closed-loop planning problem. On the other hand, the robots with the open-loop problems do not plan base on the sensors, but with the map of the environment. The thesis is focused on the open-loop path planning and the discussion throughout the rest of the thesis will be only on the open-loop path planning.

2.4.1 Graphs

Graphs are widely used in a variety field of applications such as to describe transportation networks, communication networks, electric circuit, and many more. A graph, G, consists of a set of nodes (also referred to as vertices), V, and a set of edges, E. If a node u and a node v is linked, then u and v are neighbors of each other and the edge can be represented as $\{u, v\}$. A graph can be denoted as $G = (V, E)$. The edges can be directed or undirected, and also single criterion or multiple criteria.

Different problems can be solved using graph theories. Among them, the shortest path problem is one of the common problems which is to find the shortest path from the start to the goal node. A path is a sequence of nodes such that the all the nodes in the sequence are connected. The shortest path between two nodes is a path with the minimum lengths between them. The shortest path problem is not limited to just the distance but could

imitate different problems such as a path with the lowest amount of fuel, or the least amount of time to travel etc.

2.4.2 Single-criterion Shortest Path Problem

Currently, a single-criterion shortest path algorithms are used widely in a range of applications. There are many proposed solutions and yet it is still a popular research topic in robotics, networks, artificial intelligence studies and many more.

The Breadth-First Search (BFS) Algorithm

The BFS algorithm is one of the simplest algorithms for the shortest path problem. The algorithm visits the neighbors of each node layer by layer until there are no unvisited neighbors as shown in Figure 2.10. To efficiently implement BFS, a queue, denoted as Q is used. The algorithm initializes the parents of each node denoted as par[] to NONE except for the start where the parent is itself. Q initially contains start node only and gets retrieved at the beginning of each iteration. For each new layer visited, the unvisited nodes are marked as visited by denoting its parents' node as the node it visited from. The new visited nodes are added into the Q and its parent is updated in par[]. When the node reached is the goal, the path is reconstructed. The pseudocode of the algorithm is shown in Algorithm 1.

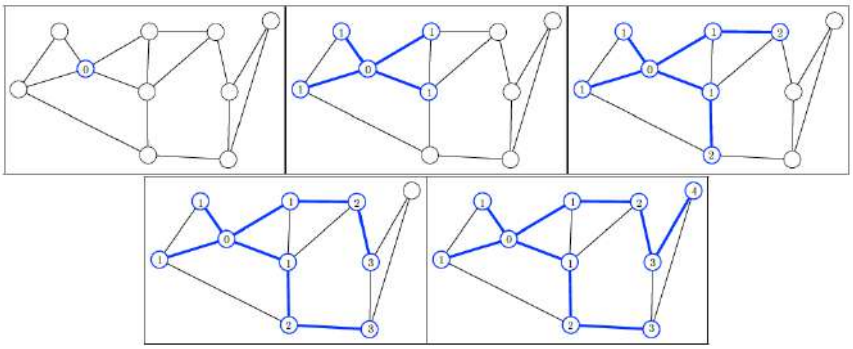


Figure 2.10: An example of the BFS algorithm in an unweighted graph. [7]

Algorithm 1 The Breadth-First Search Algorithm

Input: G: a graph, *start*: start node, *goal*: goal node
Output: P: a path from *start* to *goal* if it exists, otherwise a failure notice

```
1: Q = {start}
2: for each node v in G do
3:    par[v] = NONE
4: end for
5: par[start] = SELF
6: while ! Q = ∅ do
7:    current = retrieve[Q]
8:    for each neighbor of current do
9:       if parent[neighbor] == NONE: then
10:         par[neighbor] = current
11:         Q = Q ∪ {current}
12:      end if
13:      if current == goal : then
14:         extract-path to compute the path from start to goal
15:         return success and the path from start to goal
16:      end if
17:   end for
18: end while
19: return failure notice along with par[ ]
```

Dijkstras Algorithm

Dijkstra's algorithm and its variations are one of the most commonly used algorithms as a single source shortest path algorithm. Both the directed and undirected graphs can be used in the algorithm to find the shortest path and it will find the shortest path from the start node to all other nodes in the graphs. To compute the shortest path, the algorithm initially sets all the par $[v]$ = undefined. It also sets the distances which are denoted as dist to dist$[v]$ = inf except for the *start* which is set to zero and sets all the. The queue, Q, initially contains all the vertices but gets removed whenever the node is visited. The node to visit is selected according to the minimum distance to visit. The node's neighbors get checked if the new shortest path which is the summation of the current distance traveled dist$[u]$ and the weights between the nodes $w(u, v)$. If it is shorter than previously stored dist$[u]$, set the update the par and dist value to the new value. The algorithm terminates when Q is empty and returns dist and par. The pseudocode of the algorithm is shown in Algorithm 2.

A* Algorithm

A* is another popular pathfinding algorithm due to its performance and accuracy [32]. Instead of visiting all the nodes, A* expands by considering the nodes that appear to leads to the solution quicker and terminates when it reaches the goal. The selection is based on the node that minimizes fScore which is computed as

$$\text{fScore}(n) = \text{gScore}(n) + \text{hScore}(n) \tag{2.46}$$

where n is the last node on the path, gScore(n) is the cost of the path from *start* to node n and hScore is the heuristic cost estimates of the cheapest cost from node n to *goal*. The heuristic function should be admissible and is problem specific. The algorithm initializes and computes the gScore, hScore, and fScore value of *start* node. It also initializes two sets referred to as openSet and closedSet. The openSet contains a set of nodes that are visited but not evaluated whereas the closedSet contains a set of nodes that are already evaluated. At each iteration, the *current* node is selected by picking the nodes in the openSet with the cheapest value of fScore. If *current* node is the *goal*, the algorithm returns the path and terminates. If it is not *goal*, the *current* node gets removed from the openSet and added into the closedSet. Then each neighbor of *current* nodes get evaluated by calculating the tentative gScore of the neighbor. If the *neighbor* is not already in the openSet, it is added into the openSet. If it is already in the openSet but its tentative gScore is higher than already computed gScore for the node, it is not the better path. The algorithm continues

Algorithm 2 Dijkstras Algorithm

Input: G: a graph, *start*: start node, *goal*: goal node
Output: P: a path from *start* to *goal* if it exists, otherwise a failure notice

```
1: for each node v in G do
2:    dist[v] = ∞
3:    par[v] = undefined
4: end for
5: dist[start] = 0
6: while ! Q = Ø do
7:    current = min (dist[ ], Q)
8:    Q = Q - {current}
9:    for each neighbor of current do
10:      alt := dist(neighbor) + w(neighbor, start)
11:      if alt < dist(neighbor) then
12:         dist(neighbor) := alt
13:         par(neighbor) := current
14:      end if
15:   end for
16: end while
17: if dist[goal]! = ∞ then
18:   return  success and the dist[ ] and par[ ]
19: else
20:   return  failure notice along with the par[ ]
21: end if
```

to the next iteration. If it is a better path or it just got added into the openSet, the gScore, fScore gets updated. Also to retrieve the path, par gets updated. The pseudocode of the algorithm is shown in Algorithm 3.

Algorithm 3 A* Algorithm

Input: G: a graph, *start*: start node, *goal*: goal node
Output: P: a path from *start* to *goal* if it exists, otherwise a failure notice

```
1: openSet = { start }
2: closedSet = { }
3: gScore[start] = 0
4: hScore[start] = heuristicCostEstimate (start, goal)
5: fScore[start] = gScore[start] + hScore[start]
6: par = undefined
7: while ! openSet = ∅ do
8:    current = min (fScore[ ], openSet)
9:    if current == goal then
10:      extract-path to compute the path from start to goal
11:      return success and the path from start to goal
12:   end if
13:   openSet = openSet - {current}
14:   closedSet = closedSet ∪current
15:   tentativegScore := gScore[current] + w(current, neighbor)
16:   if neighbor not in openSet then
17:      openSet ∪{neighbor}
18:   else if tentativegScore ≥ gScore[neighbor] then
19:      continue
20:   end if
21:   par[neighbor] = current
22:   gScore[neighbor] = tentativegScore
23:   fScore[neighbor] = gScore[neighbor] + heuristicCostEstimate(neighbor, goal)
24: end while
25: return failure notice
```

2.4.3 Multi-criteria Shortest Path Problem

Until now, the algorithms discussed are the solutions for the single criteria path planning problem. In real life applications, the mobile robot path planning is not only on path

efficiency but with multiple parameters such as time, distance, energy, etc. This is referred to as Multi-Objective Optimization (MOO) problem. In mathematical terms, it is to minimize a scalar objective function, $C(x)$, with $m-$dimensional size denoted as:

$$C(x) = [c_1(x), c_2(x), \ldots, c_{m-1}(x), c_m(x)]^T \tag{2.47}$$

In general, the optimal solution to MOO problem would not be a single solution. Instead, there may be any number of different solutions that are better according to one objective function but worse in another. One way to describe the solution is in terms of Pareto-optimality. A solution $\boldsymbol{x}$ is said to dominate a solution $\boldsymbol{y}$, denoted as $\boldsymbol{x} \preceq \boldsymbol{y}$, if and only if all of its objective functions are good or better and at least one objective function is strictly better. This can be expressed as :

$$\boldsymbol{x} \preceq \boldsymbol{y} \equiv \forall i(c_i(\boldsymbol{x}) \geq f_i(\boldsymbol{y})) \wedge \exists j(c_j(\boldsymbol{x}) \geq f_j(\boldsymbol{y})) \tag{2.48}$$

A path, $path^*$, is said to be Pareto-optimal if and only if there does not exist another path in the solution space that dominates the path. There can be a set of Pareto-optimal solutions, Pareto set. One path can be better in one way compared to another path from the Pareto set, but neither of the solutions should dominate each other.

Reduction to a Single Objective

Combining a multiple constraints into a single objective with weighting factors on each objective function according to the priorities is a one approach to solve the problem. Each objective function with the weights can be summed together to become one numerical value. This type of approach is called a weighted sum approach. The weighted sum approach can then be treated as a single-criteria path problem and can be solved as a single-objective optimizing algorithm. This approach tends to be very sensitive to the weighting factors and it is hard to justify the chosen weights [67].

Evolutionary Algorithm

The evolutionary algorithm, inspired by evolution in nature, is a population-based optimization algorithm. The algorithm iteratively finds the solution by crossing a set of solutions. In an evolutionary algorithm, a word "chromosome" directly or indirectly is referred to as a solution. Genotype is the information stored in a chromosome and the solution it encodes is a phenotype. A set of chromosomes form a population. Evaluation

and iteration of the population is called a generation. The evolution from one generation to the next is done by evolutionary operators.

The algorithm selectively picks parents for reproduction. A "child" from the parent is reproduced until a prefixed number of offspring are created. The intermediate population is selected based on some survival selection method. The main challenge in applying the evolutionary algorithm for MOO problem is on selecting the child and maintaining a good population. The selection method is usually based on the fitness evaluation function. The function provides information about how good the candidates are, and usually, this is the function to optimize. Through the evolution, the generations are evaluated, selected and reproduced. The algorithm terminates when the acceptable solution is found. The varied and near Pareto-optimal solutions can be returned if the algorithm ensures that solutions that are far from other solutions in objective space survive.

Chapter 3

Overall Design

This chapter explains the overall system architecture, which is composed of VSLAM, object detection, and path planning. The VSLAM algorithm produces a map along with a pose graph which contains vertices as keyframes and edges as the constraints between the keyframes. The object detection module uses convolution neural network to recognize trained objects given an image. The results of VSLAM and object detection algorithms are combined to generate multi-weighted graph. The multi-weighted graph contains information about the distance travelled between the nodes and the number of object between the nodes. The multi-objective path planning algorithm optimizes multi-criteria and proposes path for navigation. The detailed architecture is shown in Figure 3.1. Each of the three modules in this architecture are briefly explained next. Further details are provided in Chapters 4,5, and 6.

3.1 VSLAM

VSLAM module assigns the the center of the first keyframe as the origin $(0, 0, 0)$. The tracking thread tracks the movement relative to the latest keyframe. A new keyframe is inserted when there are not enough information to be tracked. The pose of the new frame is estimated each time the tracking thread receives new information. Using the pose estimate, the distance travelled from the previous frame to the new frame is calculated by the Euclidean distance as

$$d(C_k, C_{k-1}) = \sqrt{(x_k - x_{k-1})^2 + (y_k - y_{k-1})^2 + (z_k - z_{k-1})^2} \tag{3.1}$$

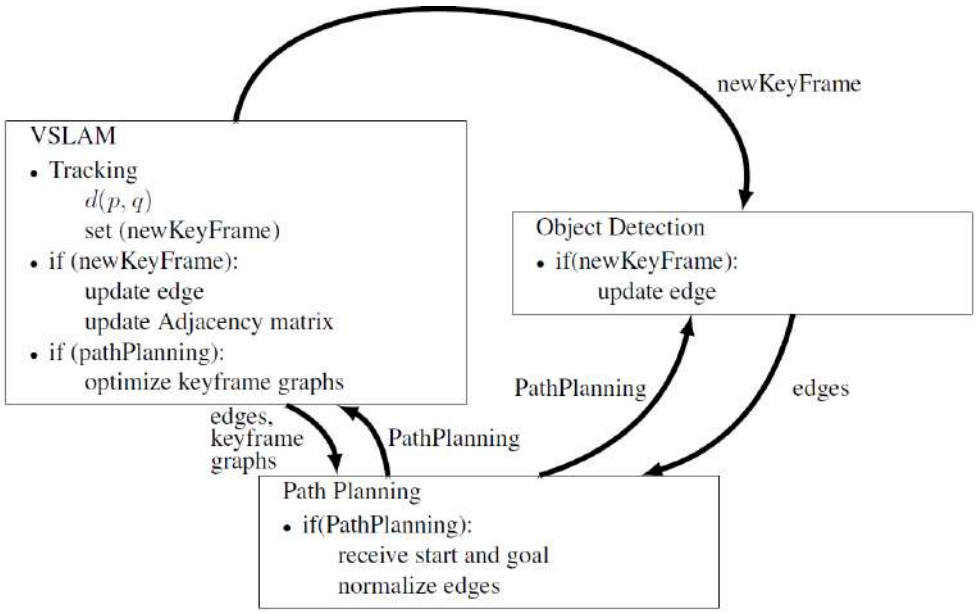


Figure 3.1: The detailed system architecture

where $C_k = (x_k, y_k, z_k)$ and $C_{k-1} = (x_{k-1}, y_{k-1}, z_{k-1})$ are the 3D point of a center of the frame at k and $k-1$ step correspondingly. The distance is accumulated until the the next new keyframe is assigned which will set the distance to zero and begin over calculating the distance. When a new frame is promoted as a new keyframe, VSLAM publishes a flag referred to as $newKeyFrame$ and updates the distance information between the nodes as well as the adjacency matrix of the graph. The adjacency matrix is a square matrix used to represent whether the pairs of nodes are connected or not in the graph. The current and previous keyframe should be connected, and the relationships can be indicated as

$$Adj(G) = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \tag{3.2}$$

where 1 in the matrix imply the corresponding nodes are connected otherwise not.

When required to start path planning, *pathPlanning* flag from path planning node will be promoted. Then the VSLAM will optimize the pose graph for the last time and shifts the first frame to origin if it was shifted during the progress. The pose of the optimized keyframe graphs are published to path planning node.

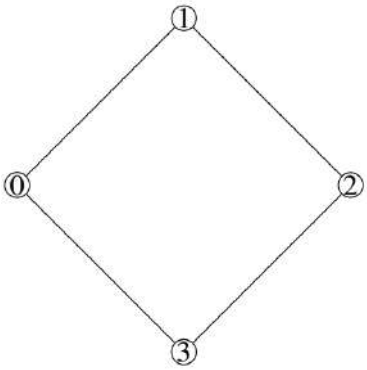


Figure 3.2: Example of graph with 4 vertices

3.2 Object Detection

Image recognition part of the node detects predefined and trained objects on each frame it receives. The predefined objects to recognize in the thesis are truck and car. The objects that are detected within each frame have certainty of 30% or higher for each objects. The detected objects among the frames are stored. When the $newKeyFrame$ flag is detected, the object detection node assigns the number of detected objects between the keyframe as the maximum number of the object detected for each objects. Similar to VSLAM, the object detection node publishes all the edge information gathered during the process if $pathPlanning$ flag is notified.

3.3 Multi-Objective Path Planning

When one expects paths, the user initiates path planning and defines the start and the goal node. The $pathPlanning$ flag will trigger two other nodes to publish necessary informations, and path planning node will subscribe to them. The pose graph from the VLSAM produces a graph with no cycles. The path planned based on this graph will be a single path as there are no other ways to reach from start to the goal. To make it as a cycle, the path planning will insert an additional edge between the last node from the graph to the closest node in the graph. This new edge information needs to be updated in adjacency matrix, as well as other weights information. The distance between the two nodes are calculated the same way as (3.1). Since it is the closest point between the two nodes, it is assumed that the object detected in the edges are zero. Once all the weight information

are updated, path planning nodes finds all the Pareto-optimal paths and shows the found paths.

Chapter 4

VSLAM Design

The following chapter discusses in-depth about a popular VSLAM method utilized in this thesis: ORB SLAM. The technique is one of the most popular approaches in feature-based SLAM.

The feature-based can be explained in two processes: (1) front-end process, and (2) back-end process. The front-end process interprets sensor data and constructs the graph by defining nodes and edges. Depending on the types of sensors available, different algorithms are used in the front-end process. A camera is a sensor utilized in the thesis. With visual sensors, the motions of the robot is estimated between frame to frame via VO. The front-end process also recognizes if the scenery has appeared before and performs loop closure. The back-end process performs inference on the graph by optimizing it.

4.1 Problem Definition

Given a sequence of images $g_1, g_2, ..., g_n$ by vision sensor, the goal of VSLAM is to generate a pose graph of the robot, $G = (V, E)$. This is done by minimizing reprojection error for feature-based SLAM. Given a 3D point of a feature P_k, and its corresponding 2D point p_{k-1}, the reprojection error is formulated as:

$$e_{proj} = p_{k-1} - \pi(R_k P_k + t_k) \tag{4.1}$$

where $R_{k,k-1} \in \mathrm{SO}(3)$ and $t_k \in \mathbb{R}^{3\mathrm{x}1}$ are rotation matrix and translation vector. With the pose estimates of the camera C_k, SLAM module is to update the pose graph and the edges.

4.2 ORB SLAM

ORB SLAM is a feature-based SLAM which generates keyframe based graph using ORB feature descriptor. Map points in ORB SLAM are in world coordinates and store information about the keyframe the point was observed from. In addition to observed features, the keyframe stores camera parameters and poses. The map points and keyframes generate a graph called covisibility graph. The nodes in the covisbility graph are the keyframes and the edges between the node exist if the two nodes share at least 15 map points. Two other graphs referred to as a spanning tree and an essential graph is also generated. A spanning tree connects the nodes with most map point observations. The edges in the essential graph are only for the edges that have a high covisibility and loop closures. which allows faster optimization.

As shown from Figure 4.1, ORB SLAM performs with three main components: tracking, local mapping and loop closing. The rest of the chapter elaborates the three main components based on work presented by [55].

4.2.1 Feature Extraction

ORB SLAM uses Oriented FAST and Rotated Brief, (ORB), for feature extraction [71]. The first step in feature extraction of ORB SLAM is to build 8 levels of image pyramid with a scale factor of 1.2 [55]. The image in each level of the pyramid is divided into regions of 30 x 30 pixels for FAST corner detection. Within each region, FAST detection is applied with $t_{init} = 20$ then $t_{init} = 7$ if initial threshold fails to detect any corner. As explained in Section 2.2.2, FAST itself is a lot faster detector compared to other detectors and allows wide baselines matches. Building the image of the pyramid allows multi-scale features and prevents missing features on a single level of scale. After finding any corner, the algorithm divides the image into 4 cells. Each cell is further divided into four subcells. The process detects the corners until the cells contain at least one corner or maximum amount of desired corners are detected. The ORB improved upon FAST detector by considering orientation component. ORB measure corner orientation by using intensity centroid, which defines the moments of an image patch as [68]:

$$m_{ab} = \sum_{p_u, p_v} p_u^a p_v^b g(p_u, p_v) \tag{4.2}$$

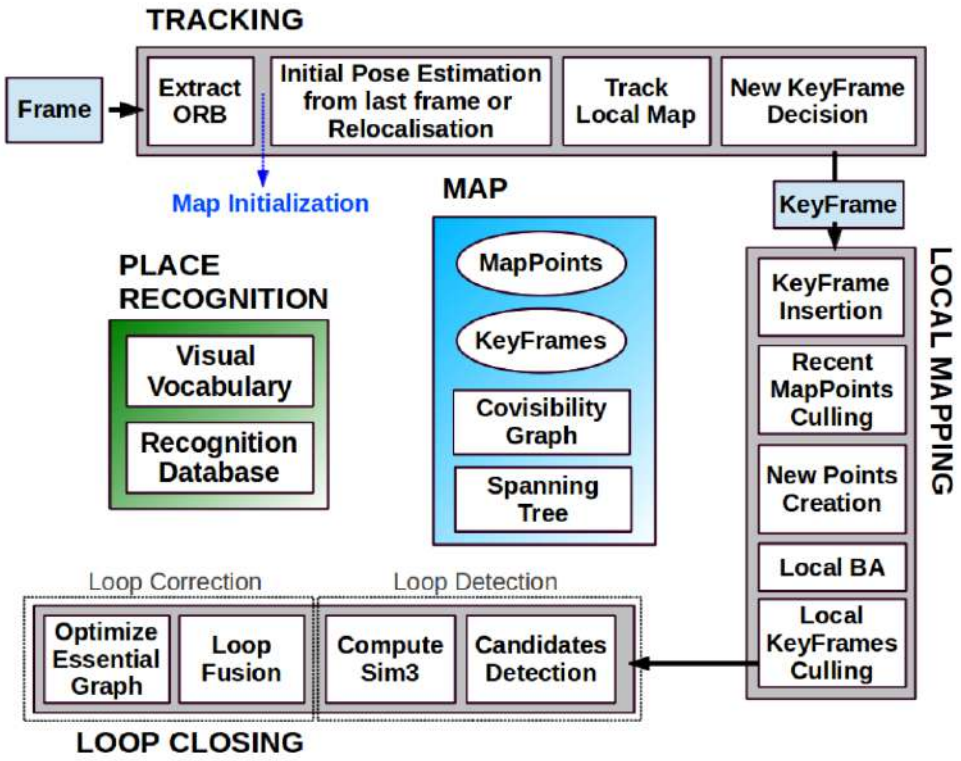


Figure 4.1: The detailed system architecture of ORB SLAM.

where $I(p_u, p_v)$ is the intensity at $p(u, v)$. Using the moment, the centroid, C, can be computed as

$$Cen = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right) \tag{4.3}$$

Using the centroid, the vector from the corner's center, O is computed as OC and the orientation is defined as:

$$\theta = atan2(m_{01}, m_{10}) \tag{4.4}$$

The ORB SLAM uses BRIEF descriptor. As discussed in Section 2.2.2, BRIEF is a vector of n binary tests and it is is set as $n = 256$, $S = 31$ [71]. To allow in-plane rotation of BRIEF, ORB introduces rotation-aware BRIEF (rBRIEF), which picks tests points from learned step that maximizes variance and minimizes correlation.

4.2.2 Map Initialization

After finding sufficient correspondences, ORB SLAM in parallel computes two geometric models: a homography, H, and a fundamental matrix, F. A homography and fundamental matrix fulfill:

$$p_1 = Hp_2 \tag{4.5}$$

$$p_1^T F p_2 = 0 \tag{4.6}$$

where p_1, p_2 are 2D points in image 1 and 2 respectively. Usually, frame 1 and 2 are referred to as the reference frame, and the current frame correspondingly. Both geometries are computed using normalized DLT and 8-point algorithm respectively from the techniques explained in [33]. The number of iterations for both models are prefixed and are the same. The points used for homography are 4 and fundamental matrix are 8.

Whether to use homography or fundamental matrix for pose recovery depends on the motion. If the scene is nearly planar, the motion is better explained by the homography matrix, whereas the non-planar scene is preferred by the fundamental matrix. ORB SLAM approaches this problem using the heuristic method, which is defined by:

$$R_H = \frac{S_H}{S_H + S_F} \tag{4.7}$$

where R_H is a decision heuristic, S_H is a homography score, and S_F is a fundamental matrix score. If $R_H > 0.45$ homography is selected for pose recovery, else fundamental matrix. Eight motion hypothesis are recovered using a method from [18] for homography. For the fundamental matrix, it is converted to an essential matrix, E, using the calibration matrix. Then the hypothesis is recovered using the method from [33]. The final step for map initialization is to perform full bundle adjustment. If the map points initialized at this stage do not contain at least 100 points, the initialization is rejected and the algorithm reinitializes the map.

4.2.3 Tracking

After the initialization, tracking thread estimates the camera motion from frame to frame and promotes frames as a keyframes, $\mathcal{K}_i$, whenever necessary. Each keyframe consists of camera intrinsic parameters, and all ORB features and relative camera pose C_i. Depending on the availability of the motion model, ORB SLAM tracks the new frames differently.

Initial Pose Estimation from Previous Frame

When the tracking is successful from the previous frame, the algorithm assumes the camera will move in the similar way. The estimates of the camera pose is searched with prediction based on observations from the previous frame and constant velocity motion model. Depending on which level the features were found from the pyramid, the search area varies. If the matches found are not sufficient enough, wider area is searched then the pose is optimized with the correspondences.

Initial Pose Estimation via Global Relocalization

If tracking is lost, global relocalization is done via BoW place recognition. ORB SLAM is based on DBoW [24], which builds a vocabulary tree using ORB features. The words are built offline using a large set of images. The system builds a hierarchical database of vocabulary tree, where the nodes represent the cluster of feature vectors. All the feature descriptors are stored in the root node. Each level in the tree is clustered using k mean clustering method based on Hamming distance. Every word in DBoW stores information about which image the word was found and the weight for word in the image. Well trained DBoW can be applied to different scenarios. Approximately 100000 images were used to train the dataset and about one million words were created [56].

With lost tracking, feature descriptors in the lost frame are compared with the database to recognize keyframes for global relocalization. This is done by comparing BoW vectors, v_t, which shows how often the words appear in the image. The vectors are compared as:

$$s(v_1, v_2) = 1 - \frac{1}{2}\left|\frac{v_1}{|v_1|} - \frac{v_2}{|v_2|}\right| \tag{4.8}$$

Then RANSAC iterations for each keyframe are performed.

Track Local Map

After initial pose estimation, the map points on each keyframe are projected on to the map. To reduce the complexity, the algorithm projects local map only. The local map consists of a set of keyframes, $\mathcal{K}_1$ which share the same map points with the current frame and a set of keyframes, $\mathcal{K}_2$ which are direct keyframes to $\mathcal{K}_1$. Within the sets, $\mathcal{K}_1$ a keyframe that shares the most map points to the current frame is used as a reference frame. The optimization of the camera pose using the local map are done as follows:

1. Compute the map points projection into the current frame. If the points are out of the frame bound, discard.

2. Compute the angle between the current viewing ray, v, and the map point mean viewing direction, n. If $v \cdot n < \cos 60$ discard the points.

3. Compute the distance from the map point to the camera center. Discard if $d \notin [d_{min}, d_{max}]$

4. Compute the scale of the frame as $\frac{d}{d_{min}}$

5. With any left unmatched features in the current frame, compare the descriptor of the map points, at the predicted scale. Associate the map points with the best match.

New Keyframe Decision

Promoting a current frame as a new keyframe requires some conditions. The correctly inserted frame allows more robust tracking, especially for complex camera movements. The frame is decided as a new keyframe if all of the following conditions are satisfied:

1. The last global relocalization occurred more than 20 frames ago

2. Local mapping is idle or last new keyframe was promoted more than 20 frames ago

3. The current frame tracks at least 50 points

4. Less than 90% of the points are tracked in the current frame from the last keyframe

The pose estimate of the new keyframe is temporarily stored as a reference to calculate the edges between the nodes.

4.2.4 Mapping

As mentioned in the tracking section, ORB SLAM keeps track of local maps. Mapping thread updates the map points by adding and removing keyframes and optimizing them.

The new keyframe is firstly added on the covisibility graph and spanning tree, the edges on the graphs are updated by checking the shared map points. For triangulation purposes, BoW representation features are computed on the newly inserted keyframe. To be retained in the local map, the map points must:

1. The points must be trackable more than 25% of the frames in predicted frames

2. After map point creation, the points should be observed in at least three keyframes

If the map points fulfill both conditions, the points can only be discarded if the points do not appear in three keyframes or local bundle adjustment considers them as an outlier.

The triangulation of ORB features using the covisibility graph allows generation of new map points. The unmatched ORB points in keyframes are compared with others via BoW. Any points that do not satisfy the epipolar constraint is discarded. The matched points are triangulated then checked for positive depth, parallax, reprojection error, and scale consistency. Then the points are projected to other keyframes and are tracked as explained in 4.2.3.

Through local bundle adjustment, currently connected keyframes in the covisibility graph and its corresponding map points are optimized. The rest keyframes are also involved with optimization but do not go through changes.

4.2.5 Loop Closing

Loop closing thread checks if there are a possibility of closing the loops. When a new keyframe is inserted, this process first checks the candidates for loop closure detection. The similarity of the keyframe is computed using the BoW vectors in the covisibility graph ($\theta_{min} = 30$). The lowest score of the similarity, s_{min} is used to check other keyframes that have common BoW vectors. The keyframes that score less than s_{min} are discarded. If at least three loop candidates are detected consecutively and consistently, similarity transformation is computed.

Chapter 5

Object Detection

The following chapter discusses a unified object detection approach utilized in the thesis: YOLO. Many proposed multiple object detection method involves a combination of region proposals and classification which achieves high accuracy, but very time-consuming. YOLO is a recent object detection approach which predicts and classify an object at one time [64].

5.1 Problem Definition

Given an image, g_1, the goal of object detection thread is to detect instances of objects with its confidence level.

5.2 Unified Detection Design

YOLO is a unified detection method that localizes an object with regression. This solution returns a bounding box from the algorithm. Each bounding box contains five elements: (w, h, x, y) and box confidence. The width, w, and height,h are relative to the size of the image and the center coordinates of the box (x, y). There are similar previous works such as R-CNN [27], Fast R-CNN [26], Faster R-CNN [66] and SSD [42] which returns bounding box as the solution. However, YOLO is a much faster solution compared to other solutions and its accuracy is also comparable or even superior. For implementation, faster version referred to as Tiny YOLO is used. Tiny YOLO contains 9 convolution layers (original YOLO has 24), and the class size to detect are $C = 4$ (car, truck, stop and yield

sign). Other than the size of the network, the training and testing parameters are the same ($S = 7, B = 2$) and the details of the parameters are explained in the following section.

5.2.1 YOLO

YOLO proposes to solve object detection problem using a single neural network to predict the location as well as classification. Unlike other approaches where the same region is potentially checked for several thousand times, YOLO only looks at the part of the image once. This is firstly done by evenly dividing an input image into S x S grids as shown in Figure 5.1. Each grid predicts B bounding boxes where the center is the grid. Each bounding box estimates the probability of containing an object as $P(Object)$. YOLO also predicts the C conditional class probabilities as $P(Class_i|Object)$, where C is the number of objects on each grid. As each bounding box is centered on the individual grid, it assumes that the object in the grid is the same as the object in the cell. The predictions are encoded as S x S x (B x $5 + C$) tensors as shown in Figure 5.1.

During the testing stage, YOLO predicts the class-specific probability for each grid as:

$$P(Class_i) = P(Class_i|Object)P(Object) \tag{5.1}$$

where $P(Class_i)$ is the probability of i th class. This gives how well the predicted box fits the object and the probability of the class appearing in the box. To eliminate unsatisfactory predictions, any $P(Class_i) \leq P_{threshold}$ is rejected. The algorithm also computes class-specific confidence which is defined as:

$$P(Class_i)\text{IOU}_{pred}^{truth} \tag{5.2}$$

where IOU is intersection over union between the predicted box and the ground truth.

The overall process of YOLO is shown in 5.2. As discussed in 2.3.2, YOLO's convolution layer extracts features. Inspired by GoogLeNet [79], YOLO has 24 convolution layers for feature extraction and 2 fully connected layers to predict bounding boxes with class scores as shown in Figure 5.2. The input images are resized to 448 x 448 with RGB channels. The first convolution layer involves 7x7x64 filter with a stride of 2 resulting in output with dimensions of 224x224x64. Then 2x2 max pooling reduces the size to 112x112x64. Some of the proceeding layers have filters with the size of 3 x 3 or 1 x 1 to reduce the depth of the feature maps. In total there are 24 convolution layers for YOLO. The paper uses Pascal VOC dataset, where the parameters are set as $S = 7, B = 2$ and $C = 20$, resulting in final feature map of 7x7x(2x5+20)=1470 tensor predictions.

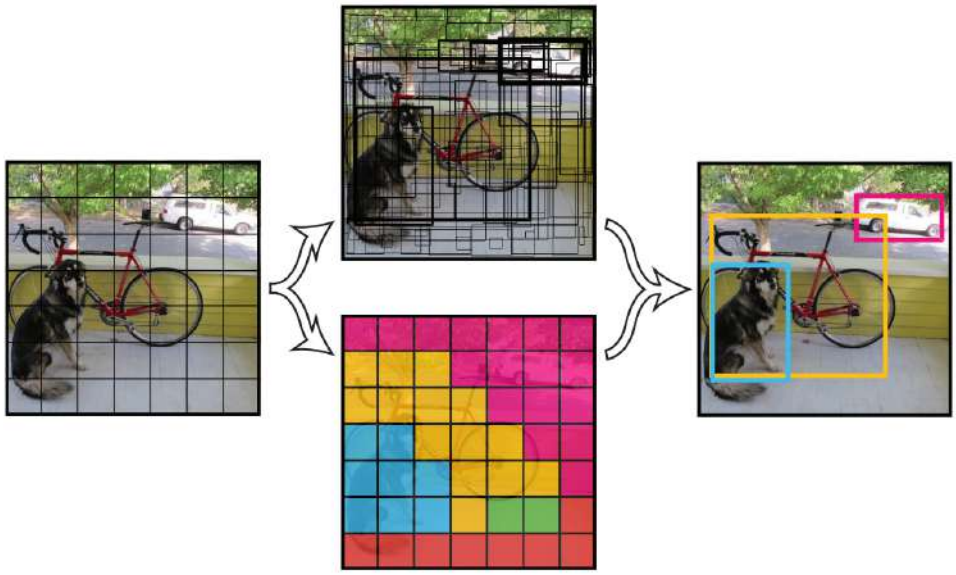


Figure 5.1: Structure of YOLO design showing its even grid, bounding boxes, confidence and class probabilities [64].

Cost Function

During the training of YOLO, the cost function to be minimized consists of three different costs: localization loss, confidence loss, and classification loss.

The localization loss penalizes the location and size of the predicted bounding box as:

$$\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}(\sqrt{w_i}-\sqrt{\hat{w}_i})^2+(\sqrt{h_i}-\sqrt{\hat{h}_i})^2 \quad (5.3)$$

where (w, h, x, y) are the predicted bounding box and $(\hat{w}, \hat{h}, \hat{x}, \hat{y})$ are true size and locations from the training data. The size deviations are relative to the true size of the bounding box. This is partially accounted by squaring width and height. λ_{coord} is multiplied to emphasize on bounding box accuracy.

The confidence loss is associated with the confidence of each bounding box. The loss term is denoted as:

$$\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}(C_i-\hat{C}_i)^2+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{noobj}(C_i-\hat{C}_i)^2 \quad (5.4)$$

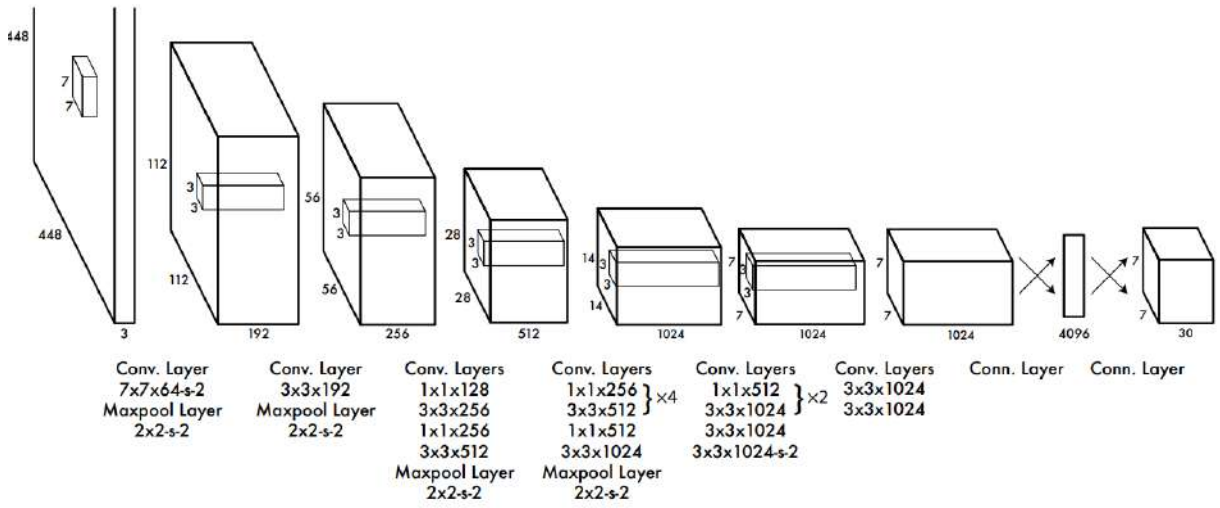


Figure 5.2: The overall architecture of YOLO [64].

where C is the confidence score and $\hat{C}$ predicted confidence score. $\mathbb{1}_{ij}^{obj}$ and $\mathbb{1}_{ij}^{noobj}$ are complements of each other, where $\mathbb{1}_{ij}^{obj} = 1$ if an object appears in the cell, 0 otherwise. λ_{noobj} is used to increase model stability; usually $\lambda_{coord} > \lambda_{noobj}$ to penalize less for predictions with no object.

The classification loss is computed as:

$$\sum_{i=0}^{S^2} \mathbb{1}_1^{obj} \sum_{c \in classes} P_i(c) - \hat{P}_i(c) \tag{5.5}$$

where $P_i(c)$, $\hat{P}_i$ denotes the conditional and predicted conditional class probability for class c in cell i respectively. $\mathbb{1}_1^{obj}$ is 1 if an object appears in the grid, 0 otherwise. The $\mathbb{1}_1^{obj}$ term ensures that there is no penalty for classification error if no object is presented in the cell.

5.2.2 YOLO V2

Faster and more accurate version of YOLO referred to as YOLO V2 [65] was proposed in 2017. This thesis utilized YOLO V2 for object detection and this section highlights some of the major differences between YOLO and YOLO V2.

Convolution with Anchor Boxes

During the training stage, YOLO predicts the arbitrary size of bounding boxes using fully connected layers in the last phase. This method works sufficiently well for some types of objects, but not all. YOLO V2 removes these fully connected layers and improves upon the arbitrary dimensions by accounting the fact that the objects have approximately the same aspect ratio for the same type of objects. The arbitrary bounding boxes are replaced with 5 anchor boxes with offset values. The class-specific probability is computed at the boundary box level instead of grid level.

Dimension Clusters

During the training, YOLO V2 learns to find the appropriate anchor box sizes. To determine the reasonable anchor boxes to start with, the algorithm uses k-mean clustering method on the training data. Standard Euclidean distance clustering method does not reflect true clusters since the data are boundary boxes and not points. The distance for clustering of anchor box is defined as :

$$d(box, centroid) = 1 - IOU(box, centroid) \tag{5.6}$$

this leads to good IOU scores that are independent of the sizes of the box. Various values of k are tested on VOC and COCO dataset and its average IOU with the closest centroid is shown in Figure 5.3. YOLO V2 choose $k = 5$ as the good value for anchor boxes. The 5 clustering method performs similar to 9 hand-picked anchor boxes. The thesis also used 5 anchor boxes with COCO model which has a greater variation in size than VOC for trading.

Direct Location Prediction

During the training, the anchor box requires prediction of where the box should be located. The estimates include 5 parameters, $(t_x, t_y, t_w, t_h$ and $t_o)$, and are predicted relative to the location of the grid cell. The prediction involves sigma function to constrain and is computed as:

$$b_x = \sigma(t_x) + c_x \tag{5.7}$$

$$b_y = \sigma(t_y) + c_y \tag{5.8}$$

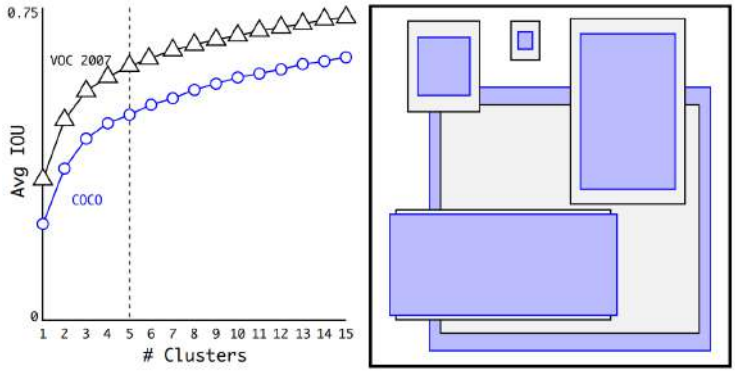


Figure 5.3: Clustering box dimensions on VOC and COCO dataset [65].

$$b_w = p_w \exp(t_w) \tag{5.9}$$

$$b_h = p_h \exp(t_h) \tag{5.10}$$

$$P(Object)IOU(b, Object) = \sigma(t_o) \tag{5.11}$$

where b_x, b_y, b_w, b_h are the predicted boundary box, c_x, c_y are the top left corner of the anchor, and c_w, c_h are width and height of the anchors. Figure 5.4 visualizes how the estimates are computed. This allows initial training to be more stable.

Multi-Scale Training

The exclusion of fully connected layers allows YOLO V2 to resize the images flexibly. YOLO V2 incorporates this to train the images of different sizes which makes the overall algorithm to be robust on various sizes. YOLO V2 takes an images of size with 320x320, 352x352, $\cdots$ and 608 x 608 (steps of 32). On every 10 batches, the network randomly selects any sizes sizes from $\{320, 352, \cdots, 608\}$ to train. This allows the same trained model to perform similarly with different resolutions and allows a trade-off between accuracy and speed. For testing, the input image is resized to 416 x 416.

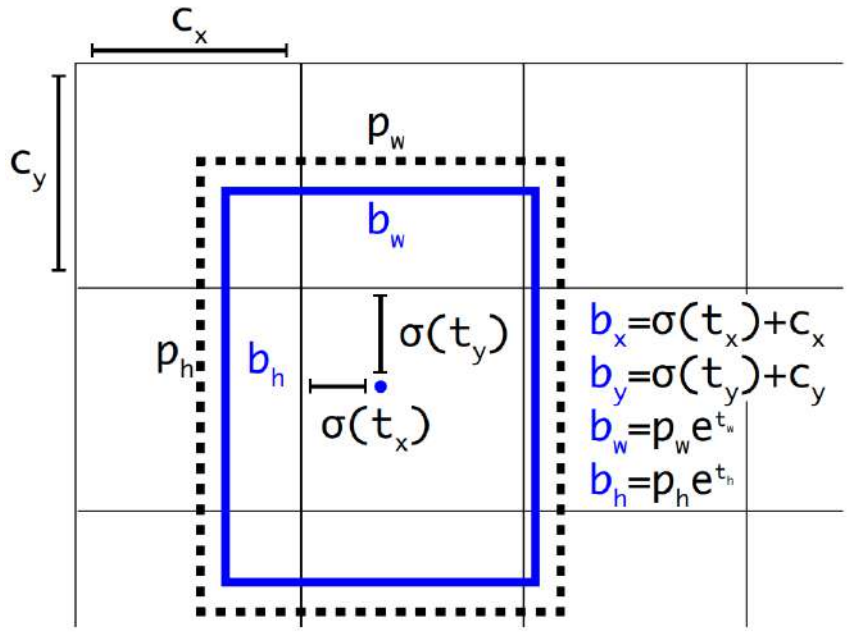


Figure 5.4: Boudning box prediction where the blue box is predicted boundary box and dotted box is the anchor [65].

Chapter 6

Multi-objective Path Planning with Pose SLAM

The combination of the previous two chapters, Chapter 4 and, 5 generates a pose graph of the system with multiple edge information. The adopted VSLAM generates the pose graph with edge information on the distance it traveled from one node to another node while the object detection algorithm determines the number of cars, trucks, stop signs, and, yield signs detected between the nodes. Using the graph, this chapter presents the proposed multi-objective route planning algorithm.

6.1 Problem Definition

The objective of multi-objective path planning is to find all efficient paths from start to goal that minimizes path cost. A pose graph is denoted as $G = (V, E)$, where V represents the set of nodes, and E represents the set of undirected links. Let $n = |V|$ and $m = |E|$, where n and m represent the numbers of nodes and edges, respectively. Each link E_{uv} from node u to v is associated with weights $c_1(u, v)$, $c_2(u, v)$, ... , $c_m(u, v)$. The multiple criteria are not limited to any number, however, for this thesis only three are considered. First weight $c_1(u, v)$ represents a distance from node u to node v, $c_2(u, v)$ and $c_3(u, v)$ can represent any arbitrary criteria such as safety, cost, energy consumed etc. For this thesis, $c_2(u, v), c_3(u, v)$ is the number of cars, and trucks the agent passes from one node to another node respectively.

The efficient paths $P(s, g)$ from start node s to goal node g can be represented as

$$P(s,g) : s = v_1, v_2, ..., v_k = g \tag{6.1}$$

where $v_1, v_2, ..., v_k$ are sequences of node numbers that path visits. Each given path can be evaluated with a cost vector C where

$$C(P(s,g)) = \begin{bmatrix} \sum_{i=2}^{k} c_1(v_{i-1}, v_i) \\ \sum_{i=2}^{k} c_2(v_{i-1}, v_i) \\ \vdots \\ \sum_{i=2}^{k} c_m(v_{i-1}, v_i) \end{bmatrix} \tag{6.2}$$

The efficiency of path can be evaluated by analyzing the cost vector of each path. A path $P(u,v)$ dominates another path $Q(u,v)$ if the cost vector $C(P(u,v))$ dominates $C(Q(u,v))$ as

$$C(P(u,v)) \leq C(Q(u,v)) \tag{6.3}$$

The path $P(s,g)$ is a Pareto-optimal path if there is no other path $Q(s,g)$ that dominates $P(s,g)$. The goal of multi-objective path planning is to find the set of all efficient paths from a start node s to g.

6.2 Multi-Objective Path Planning Algorithm

To find Pareto-optimal paths from a start node s, to a goal node g, a path is evaluated on steps whenever it branches to another node. Every time a node is visited, e.g., node z, the node will be given a label $\theta^1(z), \theta^2(z), \ldots$ If the node z is reached for the k^{th} time, the node will consist of the following information:

$$\theta^k(z) = [x^i, \alpha^k(z)] \tag{6.4}$$

where x is the parent node of z and $\alpha^k(z)$ is an objective vector, which is computed as:

$$\alpha^k(z) = \alpha^i(x) + c(x,z), \tag{6.5}$$

where $\alpha^i(x)$ is the objective vector associated with i^{th} label of node x.

Some parameters need to be calculated beforehand for the main algorithm to work. Given an edge (x,y) and a path P_{uv} define $h(x,y)$, $h(P_{uv})$, and $h^*(u,v)$ as

$$h(x,y) = c_1(x,y) + c_2(x,y) + \ldots c_m(x,y) = C(x,y), \tag{6.6}$$

$$h(P_{uv}) = c_1(P_{uv}) + c_2(P_{uv}) + \dots c_m(P_{uv}) = C(P_{u,v}), \tag{6.7}$$

$$h^*(u,v) = \min h(P_{uv}) \tag{6.8}$$

The minimum weights on each cost $c_1, c_2, \dots c_m$ are also defined as

$$q_j(u,v) = \min c_j(P_{uv}), \tag{6.9}$$

Each minimum weight is computed by considering all the paths from u to v. This can be determined using Dijkstra's algorithm discussed in Section 2.4.2. With the computation and initialization of the parameters, the main algorithm evaluates the path as:

$$e^k(z) = \alpha^k(z) + h^*(z,g) \tag{6.10}$$

If $z = g$, the objective vector, $\alpha^k(z)$, is directly compared with already found efficient vectors to determine whether the path is Pareto-optimal path or not. If it is, the objective vector is added into efficient vector sets, zΓ, which are associated with each Pareto-optimal paths. If $z = g$, consider the neighboring nodes of z. Reject the neighboring nodes if it is already on the path tip, or if the following condition is true:

$$\alpha^k(z) + c(z,x) + q(x,g) \geq \Gamma \tag{6.11}$$

For the remaining neighboring nodes, assign a tentative label; if $k-1$ tentative $\theta^k(z)$

6.3 Verification of the algorithm

The algorithm performs an exhaustive search by examining all possible paths when extending from a parent node. The algorithm will find all Pareto-optimal paths, as long as there is a path from the starting node to the goal node [83].

Starting from a prescribed source node s, the algorithm considers and evaluates all the neighboring nodes. Following the evaluation, the next step involves considering all the temporary evaluated paths and extending the paths to the best-estimated paths (the paths with the lowest $e^k(z)$ according to (6.10)). The path that is evaluated as the best path becomes and remains as permanent. The algorithm makes sure the paths are loopless and any of the proceeding paths that is exceeding found efficient vector is rejected from the rejection rule (6.11).

The goal reached will be given the label $\theta^v(g) = [x^k, \alpha^v(g)]$, and is a loopless path. The path is added in the Pareto-optimal path set, only if the path's objective vector, $\alpha^v(g)$ does not dominate any of the objective vectors in the Γ as expressed in (6.3).

Algorithm 4 Multi criteria Pareto-optimal Path Algorithm

Input: G: the graph computed by SLAM and object detection, s: start pose, g: goal pose
Output: P: Pareto-optimal path , Γ : efficient objective vector

for $i = 1$ **to** m **do**
  $q_i(z,t)$ = Dijkstra's algorithm $(G(V,c_i),z,g)$ where $z \in V$
end for
$h^*(z,t) = min(C(P(z,g)))$ where $z \in V$
$\theta^1(s) = [-, \alpha^1(s)]$
$e^1(s) = h^*(s,g)$
$L_{temp} = [\theta^1(s)]$
$L_{perm} = \emptyset$
$\Gamma = \emptyset$
$path = \emptyset$
while ! $L_{temp} = \emptyset$ **do**
  $\theta^v(x) = \min(L_{temp})$
  $L_{perm} = L_{perm} \cup \theta^v(x)$
  $L_{temp} = L_{temp} - \theta^v(x)$
  $path$ = trace back the path
  if $x == g$ **then**
    if $\alpha^v(x)$ is efficient **then**
      $\Gamma = [\Gamma, \alpha^v(x)]$
      $ParetoPath = [ParetoPath, path]$
    end if
  else
    for each neighbors of x **do**
      z = neighbor of x
      if $z \in path$ or $\alpha^k(x) + c(x,z) + q(z,g) \geq \Gamma$ **then**
        reject the neighbor
      end if
      assign label $\theta^k(z) = [x^v; \alpha^k(z)]$ if $k-1$ labels have been given to z previously
      $e^k(z) = \alpha^k(z) + h^*(z,g)$
      $L_{temp} := L_{temp} \cup \{\theta^k(z)\}$
    end for
  end if
end while
return $ParetoPath, \Gamma$

The evaluation step of the algorithm involves summing up the objective functions (6.10). The weights on the edges can be different. Hence, each of the weights is normalized prior to the algorithm.

Chapter 7

Simulation and Experimental Tests

This chapter presents the results from individual components of the proposed design and the results from the overall design. The VSLAM module, object detection module, and the overall scheme are tested using odometry benchmark KITTI dataset [23] . The multi-objective path planning scheme is tested using randomly generated data from MATLAB. The design is implemented on Intel Core i7-7700HQ (4 cores @ 2.80GHz) processor and 8Gb RAM.

7.1 VSLAM

The KITTI dataset is captured by driving around the mid-size city of Karlsruhe, in rural areas and on highways. The dataset includes 22 stereo sequences in grayscale and color images and laser data. There are 11 ground truth and 11 test trajectories included for evaluation. This thesis is not intended to evaluate the accuracy of the ORB SLAM, the detailed comparison and accuracy of different trajectories are in [55]. From the stereo dataset, the right color image with a resolution of 1382 x 512 pixels is used. Figures 7.1 and 7.2, respectively, show an example map and feature extraction generated by ORB SLAM with sequence 00. The example is captured when the ORB SLAM closes the loop for the first time in sequence 00. The blue frames are keyframe estimates, black and red points are map points. The points in the current local map are visualized as red dots.

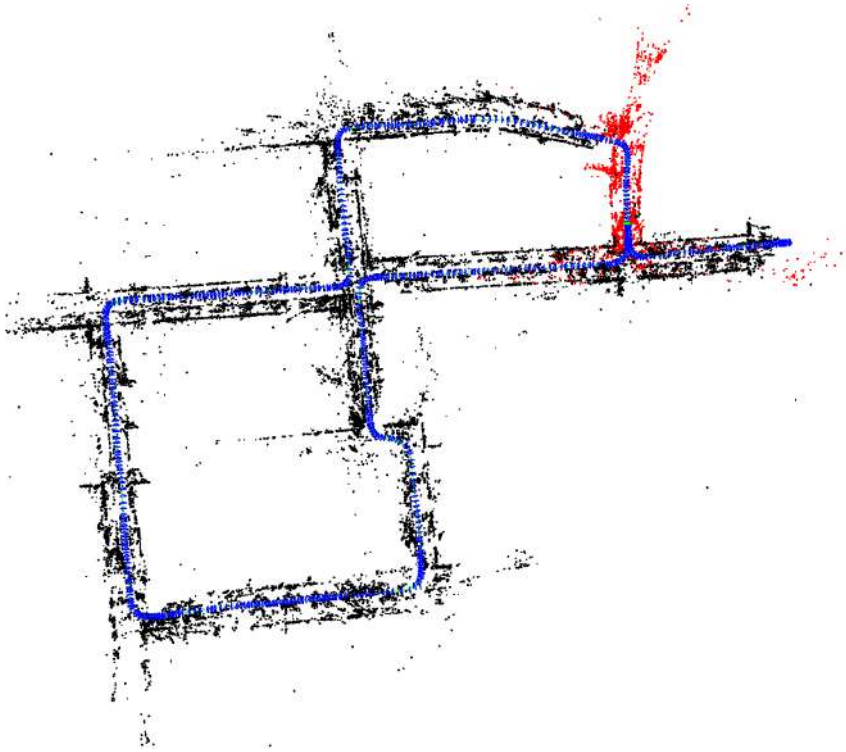


Figure 7.1: A map created and visualized by ORB SLAM with sequence 00. The blue frames indicate the estimated camera poses for keyframes. The map points are visualized with black and red dots. The red points belong to the current local map.

7.2 Object Detection

Figure 7.3 shows an example of cars and trucks detected by Tiny YOLO with confidence of 20% or higher. The object detected with confidence of 20% or higher is counted as a valid object in the proposed design. Figure 7.4 and 7.5 show comparison of the scenes with different thresholds. As confidence decreases, there tends to be more misclassification. Higher confidence avoids the misclassification but causes decrease in detection rate. Depending on priority and preference between avoidance of misclassification and avoidance of misdetection, the confidence level can be selected. This thesis adopted confidence value of 20%.

The object detection module detects objects that are trained to detect in images. Given a sequence of images, this module finds objects each time. Figure 7.6 shows time elapse comparison with the original frame (a) after (b) 3 and (c)10 frames passed relative to

Figure 7.2: A feature extraction on an image by ORB SLAM.

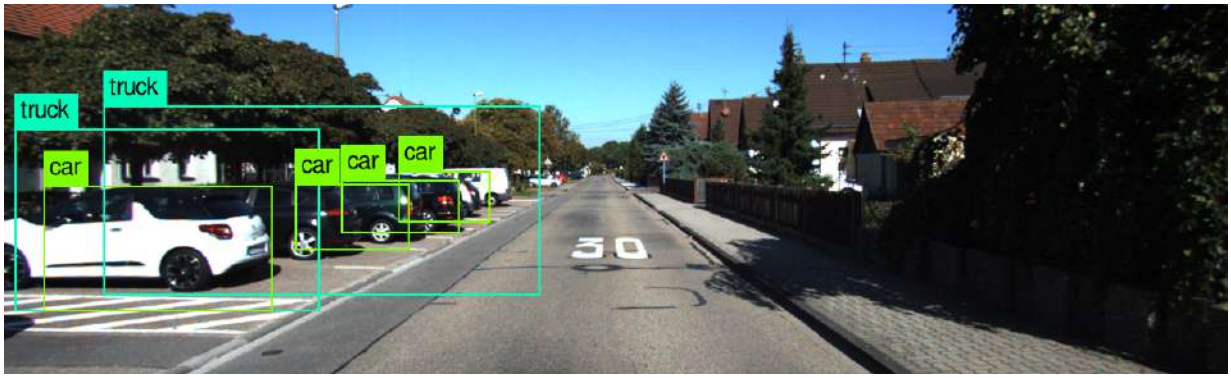
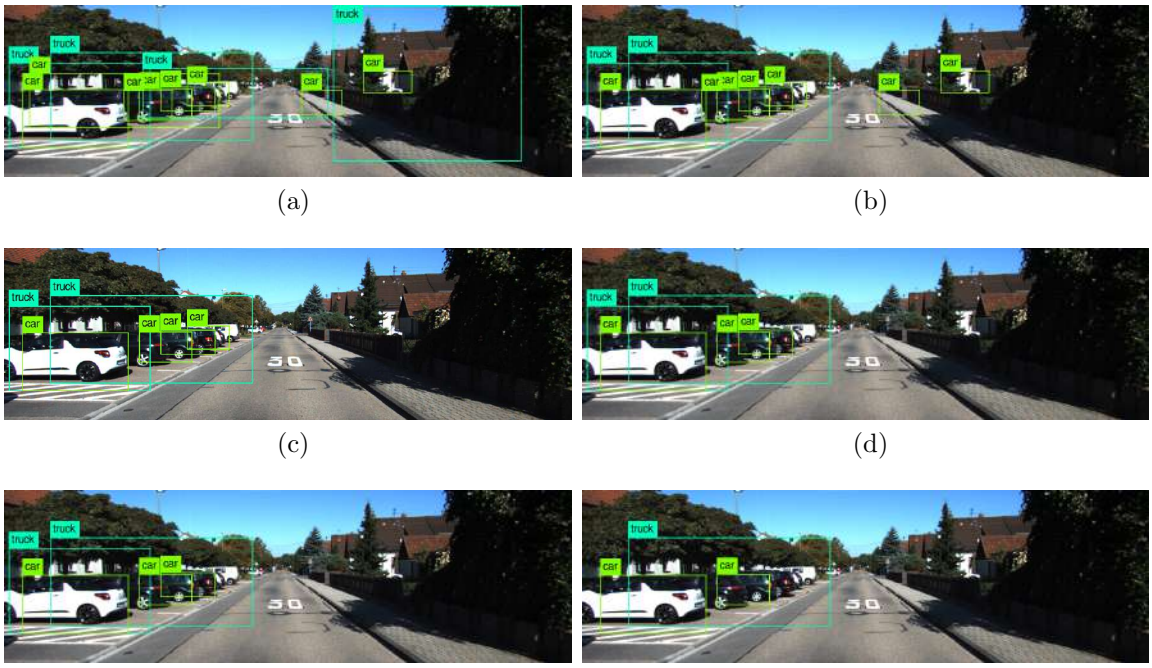


Figure 7.3: Object detection prediction on an image by YOLO.

(a). If the objects are not tracked well, the same object can be counted multiple times. The KITTI dataset is captured approximately at 10 frames per second. The detection rate when three nodes are performing in parallel is approximately 1 frame per second. Therefore, in the proposed scheme, multi-counting error in the graph would appear similar to image (a) to (c). A different dataset with different frame rate would result in different misclassification rate.

7.3 Multi-objective Path Planning

The pareto-optimal path algorithm discussed in Chapter 6 utilized in the thesis is verified with two examples. Figure 7.7 illustrates 5 nodes with 3 weights on each edges and the

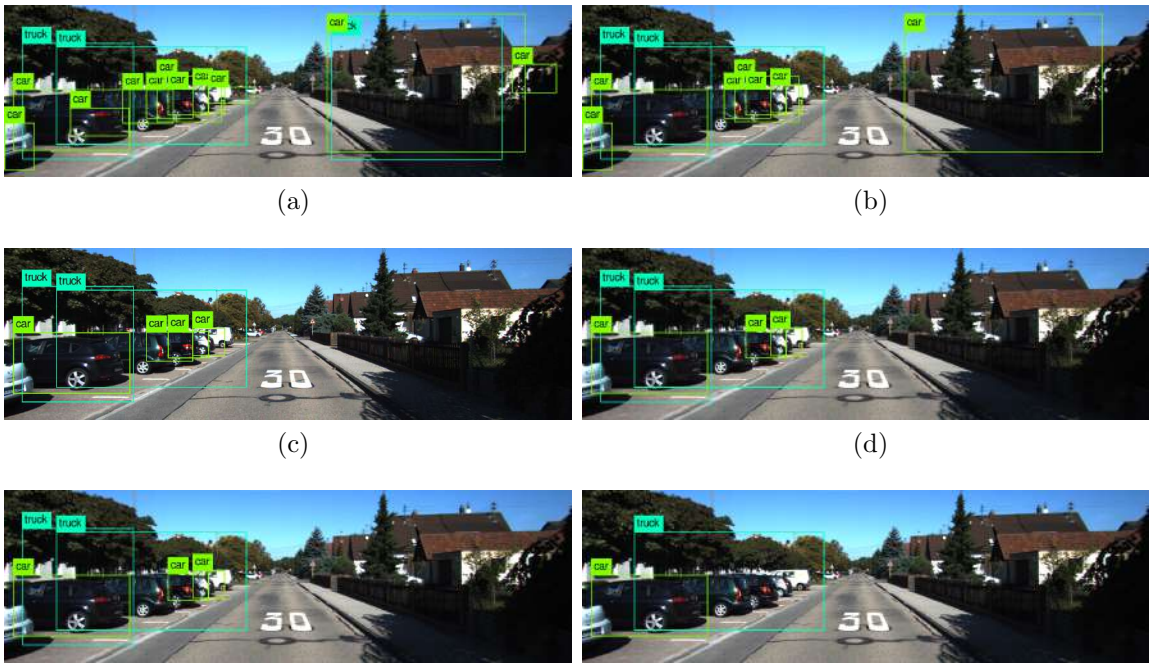


(a) (b)

(c) (d)

(e) (f)

Figure 7.4: Comparison of object detection with a confidence of (a)10%, (b)15%, (c)20%, (d)25%, (e)30%, and (f)40%.

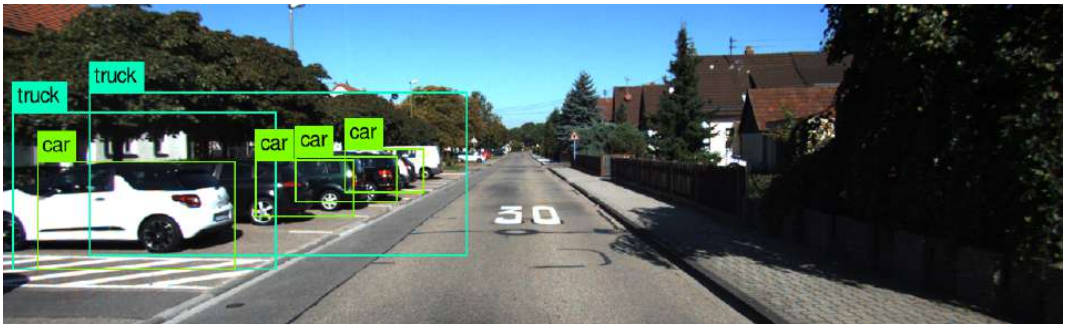


(a) (b)

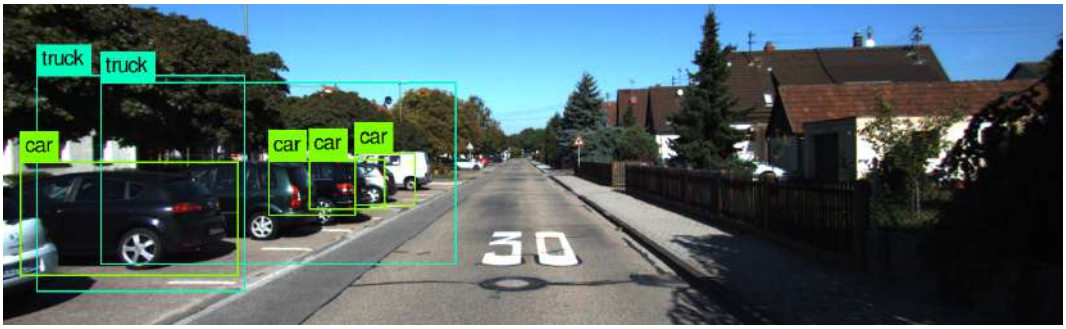
(c) (d)

(e) (f)

Figure 7.5: Comparison of object detection with a confidence of (a)10%, (b)15%, (c)20%, (d)25%, (e)30%, and (f)40%.



(a)



(b)

(c)

Figure 7.6: Time elapse comparison with the original frame (a) after (b)3 frames (c)10 frames.

proposed paths from the algorithm are shown in Figure 7.8. The $q_i(z, g)$ and $h^*(z, g)$ for $i = 1, 2, 3$ and $z \in N$ are shown in Table 7.1. The initialization of Figure 7.7 are computed as shown in Table 7.1. Figure 7.9 shows a 40 node examples and the weight information is presented in Table 7.2. The pareto-optimal paths with efficient vectors for 40 nodes example are presented in Figure 7.10.

7.3.1 5 Nodes Example

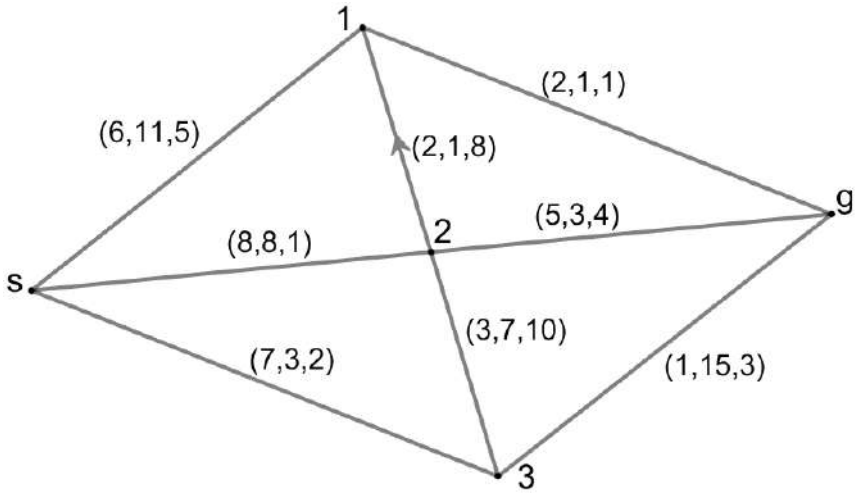
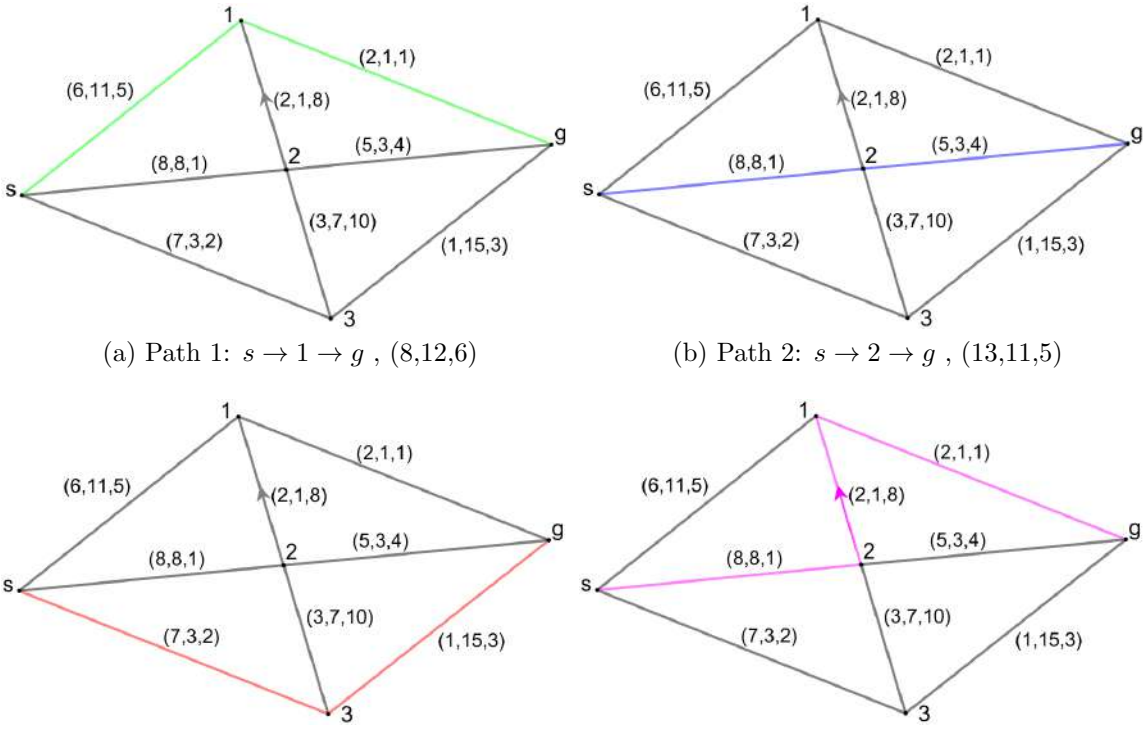


Figure 7.7: 5 nodes multicriteria graph.

Table 7.1: 5 Nodes Initialization Table

z	s	1	2	3	t
$q_1(z, g)$	8	2	4	1	0
$q_2(z, g)$	10	1	2	9	0
$q_3(z, g)$	5	1	4	3	0
$h^*(z, g)$	26	4	12	19	0

As presented in Figure 7.8, there are four pareto-optimal paths with different efficient vectors. Comparing the efficient vector, (8,12,6) for path 1, (8,18,5) for path 3, the second criteria is superior by path 1, but the third criteria is superior in path 3. The efficient vectors for path 2,4 is inferior to those for path 1,3 for the first criteria. However, path 2 and 4 dominate other efficient vectors for criteria three and two respectively. All of the



(a) Path 1: $s \to 1 \to g$, (8,12,6)

(b) Path 2: $s \to 2 \to g$, (13,11,5)

(c) Path 3: $s \to 3 \to g$, (8,18,5)

(d) Path 4: $s \to 2 \to 1 \to g$, (12,10,10)

Figure 7.8: 5 nodes multicriteria Pareto-optimal paths with its efficient vectors.

paths found are pareto-optimal paths since none of the efficient vectors are dominated by others.

7.3.2 40 Nodes Example

More complex example using 40 nodes and the results are presented in Figures 7.9 and 7.10.

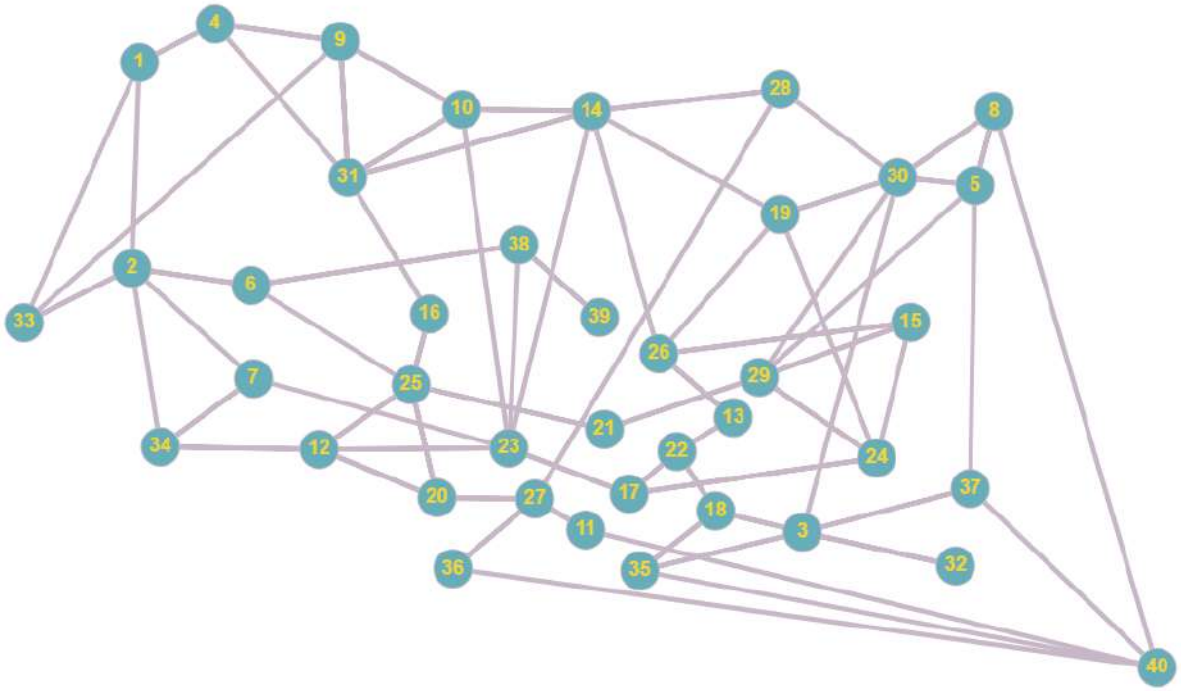
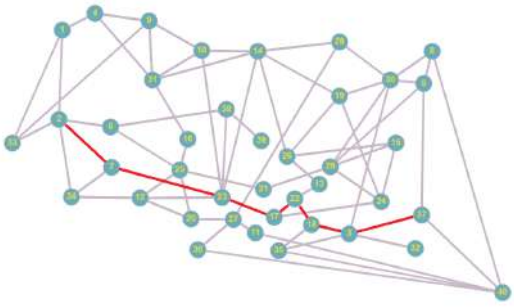


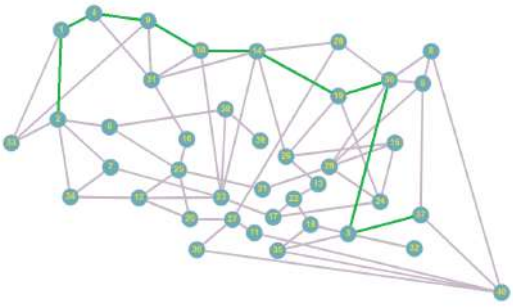
Figure 7.9: 40 nodes multicriteria graph.

7.4 Multi-objective Path Planning using Visual SLAM and Object Detection

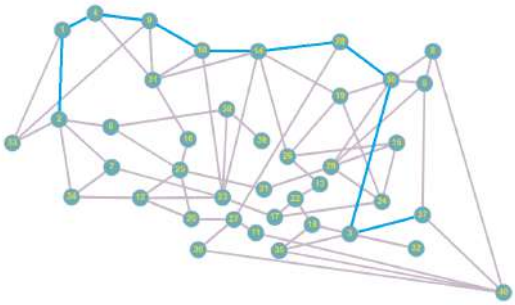
The combined proposed scheme is tested with sequence 00, 05, 06, and 07, which are sequences that ORB SLAM correctly detects and closes the loop. Due to computational



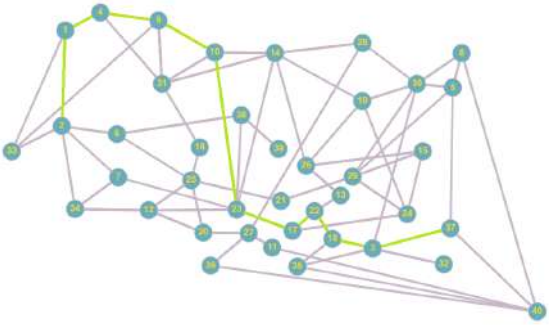
(a) Path1: 2 → 7 → 23 → 17 → 22 → 18 → 3 → 37 , (149,235,232)



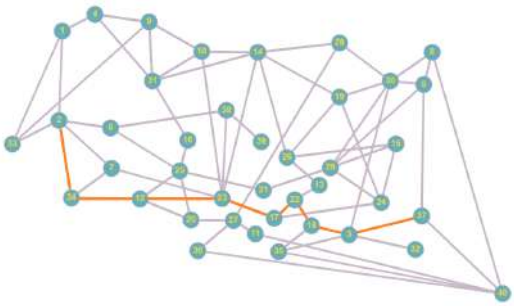
(b) Path2: 2 → 1 → 4 → 9 → 10 → 14 → 19 → 30 → 3 → 37 , (253,210,254)



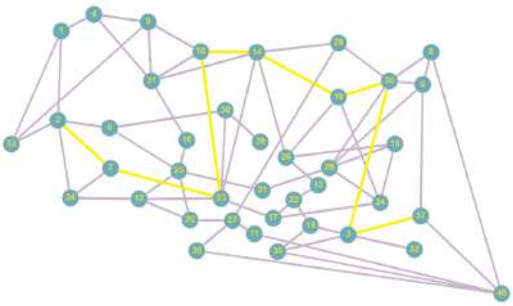
(c) Path3: 2 → 1 → 4 → 9 → 10 → 14 → 28 → 30 → 3 → 37 (291, 192, 257)



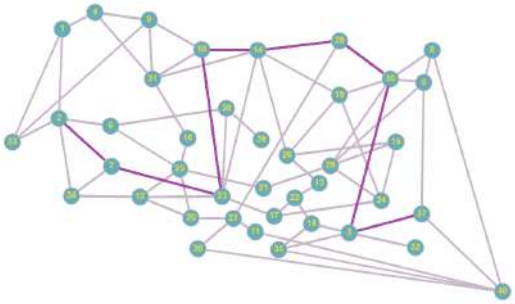
(d) Path4: 2 → 1 → 4 → 9 → 10 → 23 → 17 → 22 → 18 → 3 → 37 , (271,271,210)



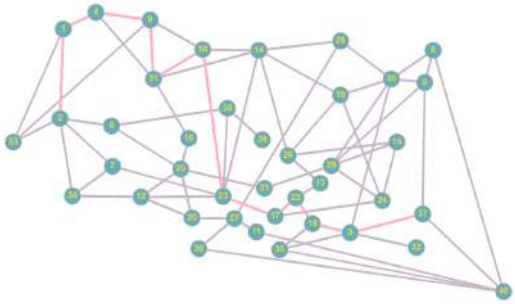
(e) Path5: 2 → 34 → 12 → 23 → 17 → 22 → 18 → 3 → 37 , (249,289,218)



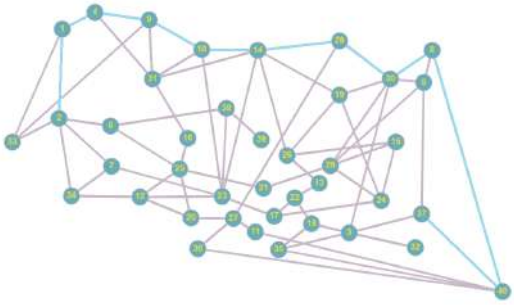
(f) Path6: 2 → 7 → 23 → 10 → 14 → 19 → 30 → 3 → 37 , (201,232,348)



(a) Path7: $2 \to 7 \to 23 \to 10 \to 14 \to 28 \to 30 \to 3 \to 37$, (239,214,351)

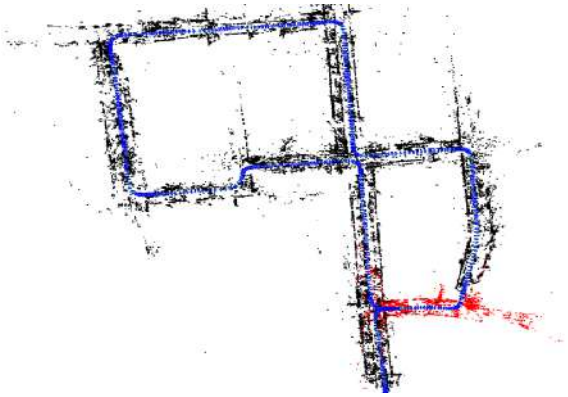


(b) Path8: $2 \to 1 \to 4 \to 9 \to 31 \to 10 \to 23 \to 17 \to 22 \to 18 \to 3 \to 37$, (321,328,208)

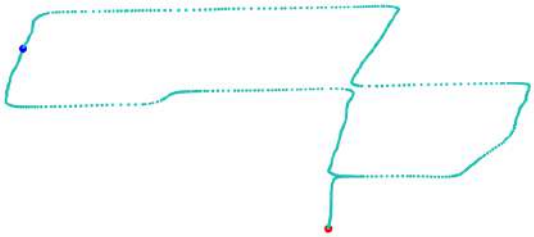


(c) Path9: $2 \to 1 \to 4 \to 9 \to 10 \to 14 \to 28 \to 30 \to 8 \to 40 \to 37$, (437,184,309)

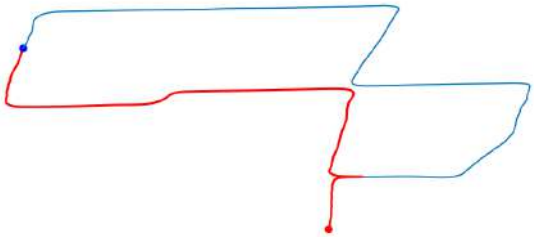
Figure 7.10: 40 nodes multi-criteria Pareto-optimal paths with its efficient vectors.



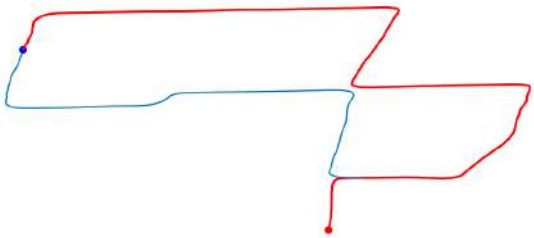
(a) Pose SLAM map built with ORB SLAM



(b) Pose graph with nodes, where red and blue node indicates start and goal respectively



(c) Path 1, (41.545,394,422)



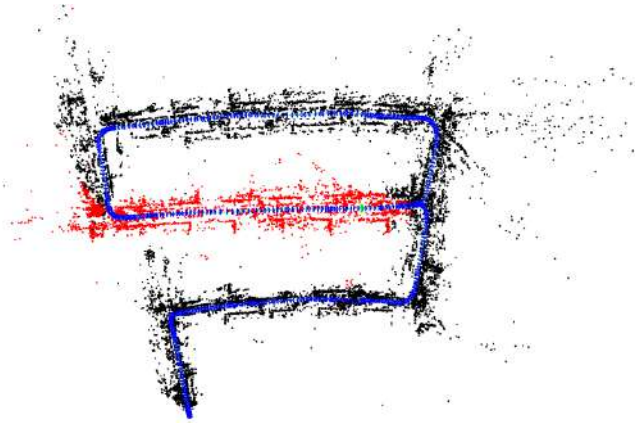
(d) Path 2, (31.977,498,301)

Figure 7.11: multi-criteria Pareto-optimal paths with its efficient vectors for KITTI dataset sequence 00.

limitation, sequences 00 and 05 are only partially tested. The start node is chosen as the first node from the graph, the goal node is particularly selected as a node that provides two pareto-optimal paths to present the result. Figures 7.11, 7.12, 7.13, and 7.14 show the two pareto-optimal path in the sequences.

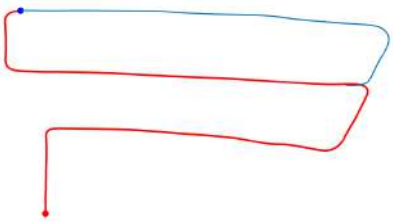
7.5 Discussion

The object detection module was originally trained to detect cars, trucks, stop and yield signs. The appearance of the stop and yield sign from KITTI dataset is once or none from different sequences tested. Hence, the proposed design is tested only with the number of

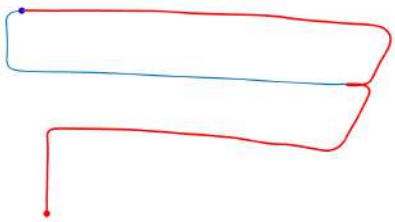


(a) Pose SLAM map built with ORB SLAM

(b) Pose graph with nodes, where red and blue node indicates start and goal respectively

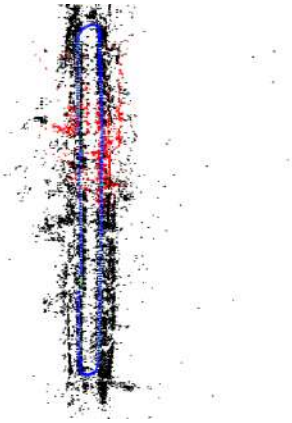


(c) Path 1, (33.484,498,410)

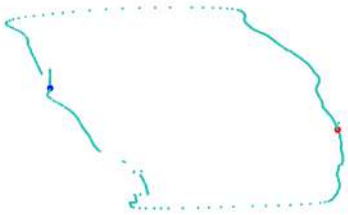


(d) Path 2, (38.236,496,485)

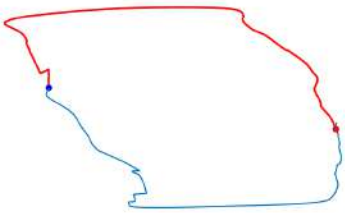
Figure 7.12: multi-criteria Pareto-optimal paths with its efficient vectors for KITTI dataset sequence 05.



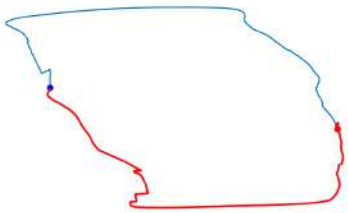
(a) Pose SLAM map built with ORB SLAM



(b) Pose graph with nodes, where red and blue node indicates start and goal respectively



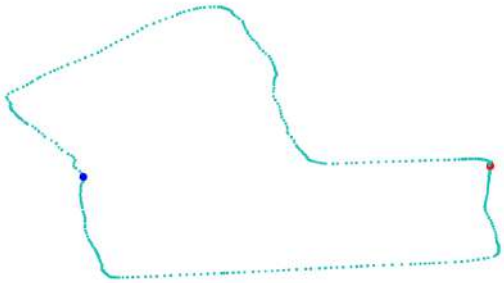
(c) Path 1, (59.066,226,138)



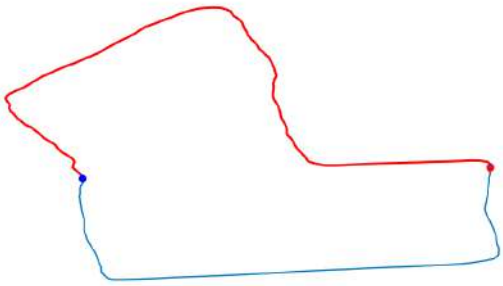
(d) Path 2, (47.443,218,228)

Figure 7.13: multi-criteria Pareto-optimal paths with its efficient vectors for KITTI dataset sequence 06.

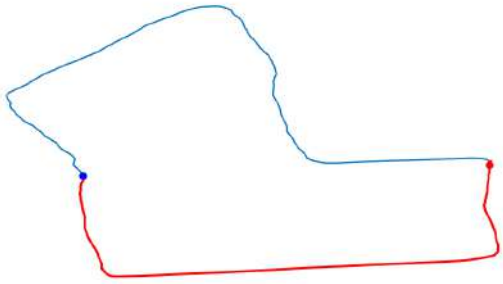
(a) Pose SLAM map built with ORB SLAM



(b) Pose graph with nodes, where red and blue node indicates start and goal respectively



(c) Path 1, (35.558,235,193)



(d) Path 2, (56.304,217,232)

Figure 7.14: multi-criteria Pareto-optimal paths with its efficient vectors for KITTI dataset sequence 07.

cars and trucks. The result presents pareto-optimal paths with different sequences.

Table 7.2: 40 Nodes Edge Information Table

node1	node2	weights	node1	node2	weights
1	2	(8,5,10)	1	4	(1,1,8)
1	33	(21,2,23)	2	33	(13,8,6)
2	34	(15,8,41)	2	7	(51,17,63)
2	6	(41,82,98)	4	9	(59,8,10)
9	33	(10,57,52)	6	38	(21,27,26)
34	7	(45,18,73)	12	34	(62,20,9)
12	20	(83,80,5)	12	23	(90,84,74)
12	25	(27,23,76)	20	25	(35,41,32)
25	21	(10,55,53)	25	16	(96,72,76)
25	6	(59,11,82)	31	16	(28,5,33)
7	23	(16,41,75)	23	38	(20,79,36)
38	39	(88,37,23)	9	10	(86,51,52)
9	31	(37,36,34)	4	31	(36,85,49)
23	10	(35,29,36)	23	14	(57,73,13)
10	14	(32,10,89)	14	31	(30,26,75)
10	31	(99,72,16)	14	26	(1, 58, 2)
23	17	(45,39,32)	17	22	(9,44,8)
20	27	(62,77,84)	22	18	(20,10,11)
22	13	(32,13,65)	21	29	(90,75,53)
17	24	(15,54,63)	26	13	(20,79,36)
14	19	(1,37,16)	26	19	(4,36,70)
14	28	(2,26,7)	26	15	(3,24,9)
19	24	(51,41,74)	15	24	(30,43,5)
19	30	(57,29,30)	18	35	(42,49,39)
30	28	(94,22,42)	15	29	(82,56,22)
29	24	(11,8,38)	27	28	(33,82,89)
27	11	(40,58,53)	27	36	(92,1,49)
18	3	(7,54,10)	32	3	(32,13,65)
3	35	(61,8,39)	36	40	(1,61,1)
3	37	(1,30,33)	3	30	(8,39,6)
29	30	(63, 30,45)	29	5	(56,74,47)
30	5	(49,72,65)	5	8	(51,71,47)
30	8	(20,7,20)	8	40	(40,1,10)
5	37	(79,60,23)	37	40	(95,53,61)
40	11	(91,3,69)	35	40	(45,59,11)

Chapter 8

Conclusion and Future Work

This thesis has proposed a multi-objective path planning scheme integrating VSLAM and object detection for autonomous robots. The objective of the scheme is to to minimize distance and number of predefined objects that are traversed during the path. The proposed approach has application in different fields where a robot is initially guided for exploration and path planning of the same place is needed. While the robot manoeuvres, VSLAM module builds the map and trajectory graph. In parallel to the VSLAM module, object detection module detects cars, trucks, stop and yield signs. The combined information is used to find a path with minimal distance, and minimal number of objects passed by. The proposed design is tested with KITTI dataset (sequences 00, 05, 06, and 07) [23] and the results demonstrate the high performance of the design. However, the stop and yield sign on KITTI dataset appeared only once or none in the tested sequences. Hence, the multi-objective path planning scheme has only considered the number of cars and trucks. The results show pareto-optimal paths and its efficient vectors from different sequences. The efficient vectors show how one path travels more distance compared to another path but passes fewer cars or trucks.

When the robot traverses the areas that were seen before, the current setup continually adds new nodes, which requires memory and computation. For long-term applications, and to reduce the computational complexity for path planning module, the node reduction of trajectory graph can be considered.

Currently, the proposed design does not keep track of detected objects. This sometimes results in multiple counting of the same objects, which causes misleading information on the graph. For the future work, the detected objects should be tracked for few frames to reduce miscounting. Another potential improvement is to speed up the object detection in

CPU. The implemented process is not fast enough to provide information on the number of objects on every edge: sometimes the number of objects detected is zero due to the computational limitation. The object detection module implemented is state-of-art on CPU. With the availability of faster version, the proposed design can be improved. The proposed design can be tested with an experimental set-up and can be applied for different scenarios, including disaster and the terrain environment.

References

[1] K Somani Arun, Thomas S Huang, and Steven D Blostein. Least-squares fitting of two 3-d point sets. *Proc. IEEE Transactions on Pattern Analysis and Machine Intelligence*, (5):698–700, 1987.

[2] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *Proc. European Conference on Computer vision*, pages 404–417, 2006.

[3] James R Bergen, Patrick Anandan, Keith J Hanna, and Rajesh Hingorani. Hierarchical model-based motion estimation. In *European conference on computer vision*, pages 237–252. Springer, 1992.

[4] Paul J Besl, Neil D McKay, et al. A method for registration of 3-d shapes. *IEEE Transactions on pattern analysis and machine intelligence*, 14(2):239–256, 1992.

[5] Dorit Borrmann, Jan Elseberg, Kai Lingemann, Andreas Nüchter, and Joachim Hertzberg. Globally consistent 3d mapping with scan matching. *Robotics and Autonomous Systems*, (2):130–142, 2008.

[6] Michael Bosse and Robert Zlot. Continuous 3d scan-matching with a spinning 2d laser. In *Proc. IEEE International Conference on Robotics and Automation*, pages 4312–4319, 2009.

[7] F. Bullo and S. L. Smith. *Lectures on Robotic Planning and Kinematics*. 2015.

[8] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, (6):1309–1332, 2016.

[9] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *Proc. European Conference on Computer Vision*, pages 778–792. Springer, 2010.

[10] Andrea Censi. An icp variant using a point-to-line metric. In *Proc. IEEE International Conference on Robotics and Automation*, pages 19–25. IEEE, 2008.

[11] Ingemar J Cox. Blanche-an experiment in guidance and navigation of an autonomous robot vehicle. *IEEE Transactions on robotics and automation*, (2):193–204, 1991.

[12] E De Castro and C Morandi. Registration of translated and rotated images using finite fourier transforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (5):700–703, 1987.

[13] Frank Dellaert and Michael Kaess. Square root sam: Simultaneous localization and mapping via square root information smoothing. *The International Journal of Robotics Research*, (12):1181–1203, 2006.

[14] Arnaud Doucet. On sequential simulation-based methods for Bayesian filtering. 1998.

[15] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics & Automation Magazine*, (2):99–110, 2006.

[16] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40:611–625, 2018.

[17] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[18] Olivier D Faugeras and Francis Lustman. Motion and structure from motion in a piecewise planar environment. *International Journal of Pattern Recognition and Artificial Intelligence*, pages 485–508, 1988.

[19] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, (6):381–395, 1981.

[20] Wolfgang Förstner. A feature based correspondence algorithm for image matching. *International Archives of Photogrammetry and Remote Sensing*, (3):150–166, 1986.

[21] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *AAAI/IAAI*, (343-349):2–2, 1999.

[22] Udo Frese and Lutz Schroder. Closing a million-landmarks loop. In *Proc. IEEE/RSJ Conference on Robots and Systems*, pages 5032–5039, 2006.

[23] Jannik Fritsch, Tobias Kuehnl, and Andreas Geiger. A new performance measure and evaluation benchmark for road detection algorithms. In *International Conference on Intelligent Transportation Systems (ITSC)*, 2013.

[24] Dorian Gálvez-López and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, pages 1188–1197, 2012.

[25] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Proc. 25th International Conference on Very Large Data Bases*, number 6, pages 518–529, 1999.

[26] Ross Girshick. Fast r-cnn. In *Proc. IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.

[27] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Region-based convolutional networks for accurate object detection and segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 142–158, 2016.

[28] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, (4):31–43, 2010.

[29] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Nonlinear constraint network optimization for efficient map learning. *IEEE Transactions on Intelligent Transportation Systems*, (3):428–439, 2009.

[30] J-S Gutmann and Christian Schlegel. Amos: Comparison of scan matching approaches for self-localization in indoor environments. In *Advanced Mobile Robot, 1996., Proceedings of the First Euromicro Workshop on*, pages 61–67. IEEE, 1996.

[31] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, number 50, pages 10–5244. Manchester, UK, 1988.

[32] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Proc. IEEE Transactions on Systems Science and Cybernetics*, 1968.

[33] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, 2003.

[34] Joel A Hesch and Stergios I Roumeliotis. A direct least-squares (dls) method for pnp. In *Proc. IEEE International Conference on Computer Vision*, pages 383–390, 2011.

[35] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[36] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, (2):216–235, 2012.

[37] Genshiro Kitagawa. Monte carlo filter and smoother for non-gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, (1):1–25, 1996.

[38] Laurent Kneip, Davide Scaramuzza, and Roland Siegwart. A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 2969–2976, 2011.

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[40] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g 2 o: A general framework for graph optimization. In *Proc. IEEE International conference on Robotics and Automation (ICRA)*, pages 3607–3613, 2011.

[41] John J Leonard and Hugh F Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, (3):376–382, 1991.

[42] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[43] H Christopher Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, (5828):133–135, 1981.

[44] Manolis IA Lourakis and Antonis A Argyros. Sba: A software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software (TOMS)*, (1):2, 2009.

[45] David G Lowe. Object recognition from local scale-invariant features. In *Proc. IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157, 1999.

[46] David G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, (2):91–110, 2004.

[47] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, (4):333–349, 1997.

[48] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981.

[49] Yi Ma, Stefano Soatto, Jana Kosecka, and S Shankar Sastry. *An invitation to 3-d vision: from images to geometric models*. Springer, 2012.

[50] Montemerlo Michael, Thrun Sebastian, Koller Daphne, and Wegbreit Ben. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proc. 16th International Joint Conference on Artificial Intelligence*, 2003.

[51] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Proc. 8th National Conference on Artificial Intelligence*, pages 593–598, 2002.

[52] Hans P Moravec. Towards automatic visual obstacle avoidance. In *Proc. 5th International Joint Conference on Artificial Intelligence*, 1977.

[53] Hans P Moravec. Visual mapping by a robot rover. In *Proc. 6th International Joint Conference on Artificial Intelligence*, pages 598–600. Morgan Kaufmann Publishers Inc., 1979.

[54] Philippe Moutarlier and Raja Chatila. An experimental system for incremental environment modelling by an autonomous mobile robot. In *Experimental Robotics I*, pages 327–346. Springer, 1990.

[55] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, pages 1147–1163, 2015.

[56] Raúl Mur-Artal and Juan D Tardós. Fast relocalisation and loop closing in keyframe-based slam. In *Proc. IEEE International Conference on Robotics and Automation*, pages 846–853, 2014.

[57] Richard A Newcombe and Andrew J Davison. Live dense reconstruction with a single moving camera. In *Proc. IEEE Computer Vision and Pattern Recognition*, pages 1498–1505, 2010.

[58] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *Proc. IEEE Computer Vision*, pages 2320–2327, 2011.

[59] Juan Nieto, Tim Bailey, and Eduardo Nebot. Recursive scan-matching slam. *Robotics and Autonomous systems*, (1):39–49, 2007.

[60] David Nistér. An efficient solution to the five-point relative pose problem. *Proc. IEEE Transactions on Pattern Analysis and Machine intelligence*, (6):756–770, 2004.

[61] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages I–I, 2004.

[62] Clark F Olson, Larry H Matthies, H Schoppers, and Mark W Maimone. Robust stereo ego-motion for long distance navigation. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 453–458, 2000.

[63] Michael K Pitt and Neil Shephard. Filtering via simulation: Auxiliary particle filters. *Journal of the American Statistical Association*, (446):590–599, 1999.

[64] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[65] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 6517–6525. IEEE, 2017.

[66] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[67] Jon T Richardson, Mark R Palmer, Gunar E Liepins, and Mike R Hilliard. Some guidelines for genetic algorithms with penalty functions. In *Proc. International Conference on Genetic Algorithms*, pages 191–197. Morgan Kaufmann Publishers Inc., 1989.

[68] Paul L Rosin. Measuring corner properties. *Computer Vision and Image Understanding*, pages 291–307, 1999.

[69] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. *Proc. 9th European Conference on Computer Vision*, pages 430–443, 2006.

[70] Sam Roweis. Levenberg-marquardt optimization. *Notes, University Of Toronto*, 1996.

[71] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Proc. IEEE International Conference on Computer Vision*, pages 2564–2571, 2011.

[72] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE Robotics and Automation Magazine*, (4):80–92, 2011.

[73] Jianbo Shi et al. Good features to track. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[74] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[75] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[76] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating uncertain spatial relationships in robotics. In *Autonomous Robot Vehicles*, pages 167–193. Springer, 1990.

[77] Randall C Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, (4):56–68, 1986.

[78] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, page 12, 2017.

[79] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[80] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[81] Sebastian Thrun and John J Leonard. Simultaneous localization and mapping. In *Springer Handbook of Robotics*, pages 871–889. 2008.

[82] Sebastian Thrun and Michael Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, pages 403–429, 2006.

[83] Chi Tung Tung and Kim Lin Chew. A multicriteria pareto-optimal path algorithm. *European Journal of Operational Research*, pages 203–209, 1992.

[84] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*, 2014.