RGB-D Visual Simultaneous Localization and Mapping (SLAM) Application

By

Weihan Wang

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Computer Science

October, 31 2019

Nashville, Tennessee

Approved:

Xenofon D. Koutsoukos, Ph.D.

Richard Alan Peters, Ph.D.

ACKNOWLEDGMENTS

Foremost, I would like to express my sincere gratitude to my advisor Prof. Xenofon Koutsoukos and Prof. Alan Peters, for their patience, motivation, enthusiasm, and immense knowledge.

My sincere thanks also goes to Feiyang Cai, who helped me greatly with setting up the experiment environment and those who helped me during the writing of this thesis. Especially, I would like to thank Jiani Li, for all her love and support.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

In the era of Artificial Intelligence (AI), advanced robotics technology is playing a more and more important role in our life. Simultaneous localization and mapping (SLAM), as one of the important techniques for navigation, robotic mapping and odometry, has been paid a lot of attention by both academia and industry. SLAM is a computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it [5, 6, 7]. Obviously, it requires accurate sensing measurements of its environment. Visual SLAM, which depends on visual sensors such as monocular cameras, stereo rigs, RGB-D cameras, may become the next trend of SLAM due to its price advantage than traditional sensors.

One objective of Visual SLAM is to estimate the camera trajectory (localization) while reconstructing the environment [8]. In order to achieve accurate results after long-term error accumulation, our real-time SLAM algorithm uses Bundle Adjustment method [9, 10, 11] which constructs a least square problem for nonlinear optimization.

The other objective for SLAM is mapping. Mapping provides robots with the information of their surrounding environment. Besides, map construction assists in other functions, like path planning, navigation and obstacle avoidance. Research in SLAM usually focuses on metric maps, which emphasizes accurate representation of the positional relationship of objects in the map. Metric maps are classified into sparse maps, semi-dense maps, and dense map. Sparse maps are somewhat abstract that do not need to identify all the objects in the map. In contrast, dense maps focus on modeling everything the camera captures. The sparse map is sufficient for localization. Semi-Dense maps and Dense maps are often used for navigation because they have to recognize obstacles.

In this paper, we propose a visual SLAM algorithm based on RGB-D camera and deploy

it on the real RC vehicle(F1/10).

1.1 Motivation

Robotics and computer vision market is exponentially growing. Many robotic products, augmented reality and mixed reality apps/games, etc rely on visual SLAM algorithms.

Solutions to SLAM are of core importance in providing mobile robots with the ability to operate with real autonomy. For robots in hostile environments like the deep ocean, collapsed buildings or the surface of Mars, SLAM is a foundation technology since the installation of infrastructure (e.g beacons) to aid navigation is inappropriate or impossible [12]. Meanwhile, compared with other sensors, camera has the characteristics of informative, extremely low-size, lightweight, cheap and easy to use. Thus, visual SLAM system is becoming more and more important.

SLAM has been researched for nearly 30 years and has challenged the robotics community for several decades. Even nowadays it is still receiving great research attention world-wise. By modern methods, uncertain data from various types of sensors is adopted and fused using probabilistic algorithms efficient enough to run in real-time on embedded robot processors. Research in SLAM has been broadened into related fields such as computer vision and AI [12].

Among various sensory modalities, camera is superior because it's cheap and informative. We select camera as the only sensor to set up our SLAM system. The contribution is that we deploy our SLAM system on real Racecar(F1/10) and make the source code public.

1.2 Outline

Our Visual SLAM system mainly contains the following modules:

- **Sensor input:** The sensor input is the camera input.
- **Visual Odometry (VO):** The VO module collects camera data and estimates the pose

based on the difference between consecutive camera information. It also updates the cached landmarks information to construct a local map.

- **Back-end Optimization:** The back-end optimization module receives the camera poses of the visual odometer at different time steps, and then optimizes the estimation of poses based on some cost function, and generates a globally consistent trajectory.
- **Mapping:** The mapping module combines the previously calculated local maps to establish a global map based on trajectory.

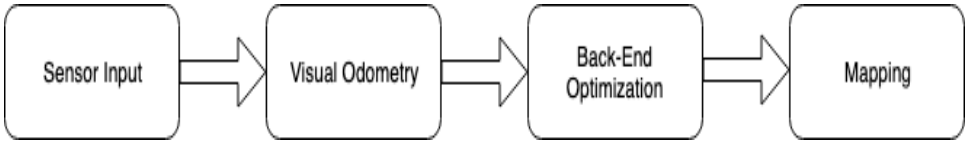


Figure 1.1: Visual SLAM modules

Chapter 2

Related Work

This chapter is organized as follows. We introduce SLAM background firstly. Subsequently, we discuss some state-of-the-art Visual SLAM system in section 2.2. In section 2.3, we introduce some relate work about SLAM on F1/10 race car. Finally, we introduce our visual SLAM system briefly.

2.1 SLAM Background

SLAM stands for Simultaneous Localization and Mapping. The challenge is to place a mobile robot at an unknown location in an unknown environment, and have the robot incrementally build a map of the environment and determine its own location within that map[13].

Researchers SLAM has studied SLAM for more than 30 years. A thorough historical review of the first 20 years of the SLAM problem is given by DurrantWhyte, and Bailey in two surveys[14, 15]. These mainly cover what we call the classical age(1986 - 2004); the classical age saw the introduction of the main probabilistic formulations for SLAM, including approaches based on extended Kalman filters (EKF), RaoBlackwellized particle filters, and maximum likelihood estimation; moreover, it delineated the basic challenges connected to efficiency and robust data association. Two other excellent references describing the three main SLAM formulations of the classical age are the book of Thrun, Burgard, and Fox[16] and the chapter of Stachniss et al [17]. The subsequent period is what we call the algorithmic-analysis age (2004-2015), and is partially covered by Dissanayake et al. in [18]. The algorithmic analysis period saw the study of fundamental properties of SLAM, including observability, convergence, and consistency. In this period, the key role of sparsity toward efficient SLAM solvers was also understood, and the main open-source SLAM

libraries were developed. The popularity of SLAM in the last 30 years is not surprising if one thinks about the manifold aspects that SLAM involves [19].

2.2 Mainstream Visual SLAM System

If according to the sensor as a category, the SLAM can be divided into radar-based SLAM and visual SLAM. Because our SLAM system is based on camera which is visual SLAM system, so we focus on some mainstream visual SLAM system and analyze their advantage and disadvantage.

- **MonoSLAM [20]:** This was presented by Dr. A.J. Davison [20] in 2007. The MonoSLAM [20] uses extended Kalman filtering as the back-end to track very sparse feature points from the front.

- **PTAM [21]:** Parallel Tracking and Mapping (PTAM) was presented by Klein el, al [21] on 2007. PTAM [21] proposes and implements parallelization of the tracking and mapping process. And also, PTAM [21] uses nonlinear optimization and no longer uses traditional filters as back-end optimization. But there are obvious defects in this system such as small scene and loss of tracking easily.

- **ORB-SLAM [22]:** ORB-SLAM [22] was proposed in 2015 and is the most complete modern SLAM system. ORB-SLAM [22] innovatively uses three threads to complete SLAM: tracking threads are for real-time tracking feature points; optimized threads are for local Bundle Adjustment, and looping detection and optimization threads for global pose graph. Since the whole system used feature points for calculation, orb features must be calculated for each frame, which is cost a lot of time-consuming. In addition, this system's three-thread structure puts a burden on CPU, making it

portable to embedded devices.

- **LSD-SLAM [23]:** Large Scale Direct monocular SLAM (LSD-SLAM) [23] was proposed by J. Engel et, al [23] in 2014. Apply direct method to semi-dense monocular SLAM without bundle adjustment over features. Nevertheless they still need features for loop detection and their camera localization accuracy is significantly lower than ORB-SLAM [22] and PTAM [21].

- **SVO [24]:** Semi-direct Visual Odometry (SVO) [24]. It was presented by Forster et, al [24] in 2014. This is a semi-direct based visual odometry. The biggest advantage of this system is that the calculation speed is extremely fast. It can reach a speed of over 400 frames per second on the PC platform. It is very suitable for an embedded device like a drone. However, SVO [24] system has abandoned the back-end optimization and looping closing detection for speed and lightweight reason and also it abandoned the mapping function.

2.3 SLAM on F1/10 Race car

F1/10 race car is an open-source, affordable, and high-performance 1/10 scale autonomous vehicle testbed [25]. In our project, we choose F1/10 race car, because it carries a full suite of sensors, perception, planning, control, and networking software.

The F1/10 vehicle enables a novel mode of research relative to perception tasks in two solution. First solution is computer vision by Deep Learning method. Second solution is SLAM.

Ability for a robot to create a map of a new environment without knowing its precise location (SLAM) is a primary enabler for the use of the F1/10 platform in a variety of locations and environments [25]. In order to allow the F1/10 race car to drive in a new environment, current SLAM solution on F1/10 is a LIDAR-based SLAM package which

provides loop-closures, namely Google Cartographer [26].

2.4 Our Visual SLAM System

Visual SLAM can be performed by using just a monocular camera, which is the cheapest and smallest sensor setup. However as the depth is not observable from just one camera, the scale of the map and estimated trajectory is unknown [22]. Therefore, in this project, we will use an RGB-D camera and build our own visual SLAM system follow by four core steps that I mentioned above. Our visual SLAM system also includes four core parts: Camera input: Visual Odometry, Back-end Optimization, and Mapping.

How does our visual SLAM system work? Visual Odometry is able to estimate the position of the camera through adjacent images and restore the spatial structure of the scene. Then Back-end is focussing on pose optimization during system processing.

RGB-D SLAM method fused all depth data from the sensor into a volumetric dense model that is used to track the camera pose using ICP [27]. But instead of the ICP [27] method, we will use the PnP [28] method for camera pose estimation.

Perspective n Points (PnP) [28] method is that a robot has the number of 3D points and their projections and then it calculates the camera pose. This is the most common situation for robot. Thus, we chose the PnP [28] method for pose estimation calculation. We will introduce our SLAM system in detail in Chapter 4.

In the commercial domain, the Visual SLAM is still in its infancy. But it already implemented on augmented reality (AR) applications and a wide variety of field robots, such as DJI company's drone.

Chapter 3

Visual SLAM Algorithm

Since in visual SLAM, we calculate poses through the camera's pinhole model, we want to briefly introduce camera's pinhole model firstly.

The remainder of this chapter is organized as follows. We first discuss camera pinhole model. Subsequently, in section 3.2 and section 3.3, we introduce camera pose estimation methods. In section 3.4, we characterize the graph optimization problems that are addressed by nonlinear least-squares algorithms such as Gauss-Neton, Levenberg-Marquardt(LM). Finally, in section 3.5, we discuss mapping methods.

3.1 Camera Pinhole Model

Camera pinhole model — This model is a transformation process from the world coordinate system to the camera coordinate system then to the image physical coordinate system, and finally to the image pixel coordinate system.

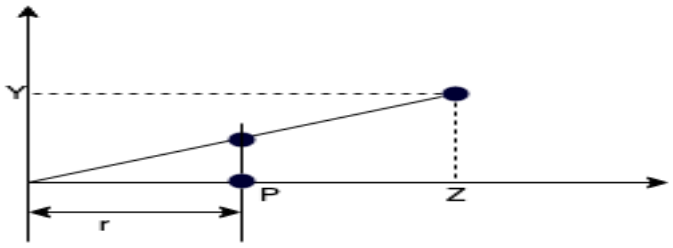


Figure 3.1: CAMERA PINHOLE MODE

We assume point P in world coordinates is $[X,Y,Z]^T$ and its camera coordinates P' is $[X',Y',Z']^T$, f is camera's focal length. According to the principle of triangle similarity,

$$\frac{Z}{f} = -\frac{X}{X'} = -\frac{Y}{Y'}$$

The negative sign indicates that the image is inverted. To simplify the model, the imaging

plane can be symmetrical to the front of the camera. Then it will be,

$$\frac{Z}{f} = \frac{X}{X'} = \frac{Y}{Y'}$$

Therefore:

$$X' = f\frac{X}{Z}$$

$$X' = f\frac{X}{Z}$$

We assume P coordinate in image pixel coordinate is $[u, v]^T$. The pixel coordinate system differs from the imaging physical plane by a scaling and an shift. So we assume that in pixel coordinate system, u axis scale α and v axis scale β, meanwhile, the origin point is translated by $[c_x, c_y]^T$.

Then, the relationship between the coordinates of P' and the pixel coordinates is

$$u = \alpha X' + c_x$$

$$v = \beta Y' + c_y$$

$$\Downarrow$$

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

$$v = f_y\frac{Y}{Z} + c_y$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{Z}\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \triangleq \frac{1}{Z}KP \tag{3.1}$$

$$ZP_{uv} = Z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K(RP_w + t) = KTP_w$$

3.2 Feature Detection

In computer vision and image processing, feature detection includes methods for computing abstractions of image information and making local decisions at every image point whether there is an image feature of a given type at that point or not. The resulting features will be subsets of the image domain, often in the form of isolated points, continuous curves or connected regions.

Feature point include Key-point and Descriptor. Key-point is the position of feature point in image, and Descriptor is to describe the information about the pixels around the feature point. There are many method of feature detection: SIFT, SURF, ORB etc.

In this project, we will use SURF[29] feature method, because SURF[29] method trade off a balance between speed and accuracy.

3.3 Camera Pose Estimation

Now, how to use a set of matching points to calculate the camera's pose?

Table 3.1: Argument Explanation

P_1	The 3D position of a certain landmark (P) in first frame coordinate
P_2	The 3D position of a certain landmark (P) in second frame coordinate
p_1	Landmark P's corresponding pixel in first frame
p_2	Landmark P's corresponding pixel in second frame
s_1	Landmark P's depth information in first frame coordinate
s_2	Landmark P's depth information in second frame coordinate
K	The intrinsic matrix of camera
R	Rotation matrix from the first frame to the second frame
t	the translation vector from first frame to second frame

According to camera pinhole model, we can get following equation.

$$s_1 p_1 = K P_1 \tag{3.2}$$

$$s_2 p_2 = K P_2 \tag{3.3}$$

$$P_2 = R P_1 + t \tag{3.4}$$

$$s_2 p_2 = K(R P_1 + t) \tag{3.5}$$

Our goal is to calculate the $\boldsymbol{R}$ and $\boldsymbol{t}$. Here, depending on the type of the cameras, there

are three different methods to calculate the pose.

- **Epipolar Geometry [30] (2D-2D):** Used in a monocular camera.
- **ICP [27] (3D-3D):** Used in an RGB-D or stereo camera.
- **PnP [28] (3D-2D):** Used in 3D information in one image and 2D information in another image.

We will introduce in detail how these three methods calculate the camera's pose.

3.3.1 Epipolar Geometry (2D-2D)

In 2D-2D situation, we assume the spatial position of point **P** in first frame is $[X,Y,Z]^T$. According to equation 3.2 and equation 3.5, the relationship with p_1, p_2 and P are:

$$s_1 p_1 = KP, s_2 p_2 = K(RP+t).$$

K is the camera intrinsic matrix, **R** and **t** are rotation matrix and translation vector respectively.

Now, x_1, x_2 are the coordinate of two pixels on the normalized plane. Therefore, $x_1 = K^{-1}p_1$, $x_2 = K^{-1}p_2$, and we can get

$$x_2 = Rx_1 + t$$

$$\Downarrow$$

$$t \times x_2 = t \times Rx_1$$

$$\Downarrow$$

$$x_2{}^T t \times x_2 = x_2{}^T t \times Rx_1$$

And we know $t \times x_2$ is a vector which is vertical to t and x_2. Thus:

$$x_2{}^T t \times R x_1 = 0$$

$$\Downarrow$$

$$p_2{}^T K^{-T} t \times R K^{-1} p_1 = 0$$

We know that fundamental matrix(**F**) is $F = K^{-T} E K^{-1}$, essential matrix(**E**) is $E = t \times R$. In addition, we use eight-point-algorithm to estimate the **E** by SVD decomposition.

$$E = U \Sigma V^T$$

Meanwhile,

$$t = U R_Z \Sigma U^T, R = U R_Z{}^T V^T$$

In a 2D-2D situation which has scale ambiguity, we can not get depth information from the image. Therefore, we need to use a triangulation method to estimate the depth. However, due to the influence of noise, we have to use the least squares method to satisfy the polar geometry constraint which is $[s_1 x_1 = s_2 R x_2 + t]$.

3.3.2 ICP (3D-3D)

ICP [27] is abbreviated by iterative closest point. There is two way to use ICP [27] algorithm to estimate **R** and **t**. One way is using linear method which is SVD decomposition, the other way is non-linear method which is Bundle Adjustment.

3.3.2.1 Linear method

We suppose that we have a set of matched 3D points:

$$P = \{p_1, p_2, p_3, ...p_n\}, P' = \{p'_1, p'_2, p'_3..., p'_n\}$$

Now, we want to find European transformation to make:

$$\forall i, p_i = \mathbf{R}p'_i + \mathbf{t}$$

We defined the error term of point i firstly:

$$e_i = p_i - (\mathbf{R}p'_i + \mathbf{t})$$

Then, we construct the least squares problem:

$$\min_{\mathbf{R},\mathbf{t}} J = \frac{1}{2}\sum_{i=1}^{n} ||(p_i - (\mathbf{R}p'_i + \mathbf{t}))||_2^2$$

Then, we define two centroid(p, p') of 3D points:

$$p = \frac{1}{n}\sum(p_i), p' = \frac{1}{n}\sum(p'_i).$$

$$\frac{1}{2}\sum_{i=1}^{n} ||p_i - (\mathbf{R}p'_i + \mathbf{t})||^2 = \frac{1}{2}\sum_{i=1}^{n} ||p_i - p - \mathbf{R}(p'_i - p')||^2 + ||p - \mathbf{R}p' - \mathbf{t}||^2 + N.$$

$$N = 2(p_i - p - \mathbf{R}(p'_i - p'))^T (p - \mathbf{R}p' - \mathbf{t})).$$

And we now N is equal to zero, thus the optimization objective function can be simplified as:

$$\min_{\mathbf{R},\mathbf{t}} J = \frac{1}{2}\sum_{i=1}^{n} ||p_i - p - \mathbf{R}(p'_i - p')||^2 + ||p - \mathbf{R}p' - \mathbf{t}||^2$$

Looking closely at the left and right terms, we found that the left term is only related to the rotation matrix **R**. As long as we get **R**, we can get **t** if we set the second term is zero.

$$q_i = p_i - p, q'_i = p'_i - p'$$

Now, we expand the error term for **R**

$$\frac{1}{2}\sum_{i=1}^{n} \|q_i - \mathbf{R}q'_i\|^2 = \frac{1}{2}\sum_{i=1}^{n} q_i^T q_i + q'^T_i \mathbf{R}^T \mathbf{R} q'_i - 2q_i^T \mathbf{R} q'_i$$

Note that the first term has nothing to do with **R**, and the second term $\mathbf{R}^T\mathbf{R} = \mathbf{I}$ is also nothing to do with **R**. Thus, the optimization function beacomes:

$$\sum_{i=1}^{n} -q_i^T \mathbf{R} q'_i = \sum_{i=1}^{n} -tr(\mathbf{R}q'_i q_i^T) = -tr(\mathbf{R}\sum_{i=1}^{n} q'_i q_i^T)$$

We define matrix c

$$\mathbf{W} = \sum_{i=1}^{n} q_i q'^T_i .$$

Then, we SVD the **W** matrix:

$$\mathbf{W} = \mathbf{U}\Sigma\mathbf{V}^T .$$

$$\mathbf{R} = \mathbf{U}\mathbf{V}^T .$$

After we get the **R**, we can get the **t** according to $p - \mathbf{R}p' - \mathbf{t} = 0$.

3.3.2.2 Non-linear method

To find the optimal solution in an iterative way. Essentially, the algorithm steps are:

- For each point (from the whole set of vertices usually referred to as dense or a selection of pairs of vertices from each model) in the source point cloud, Match the closest point in the reference point cloud (or a selected set) [31].

- Estimate the combination of rotation and translation using a root mean square point to point distance metric minimization technique which will best align each source point to its match found in the previous step. This step may also involve weighting points and rejecting outliers prior to alignment [31].
- Transform the source points using the obtained transformation [31].
- Iterate (re-associate the points, and so on) [31].

3.3.3 PnP (3D-2D)

In Wikipedia definition, perspective-n-point is the problem of estimating the pose of a calibrated camera given a set of n 3D points in the world and their corresponding 2D projections in the image. Meanwhile, 3D-2D is most common situation in situation. We get pixel u_i' in the current frame(k) and 3D position of p_i in the world coordinate, then computer $T_{k,k-1}$.

The solution of P3P starts from the cosine theorem. Let the camera coordinate center be the point O, A, B, and C be three 3D points that are not collinear, and D is the verification 3D point. According to the cosine theorem, the following formula is used:

$$OA^2 + OB^2 - 2OA \cdot OB \cdot \cos < a,b > = AB^2$$

$$OB^2 + OC^2 - 2OB \cdot OC \cdot \cos < b,c > = BC^2$$

$$OA^2 + OC^2 - 2OA \cdot OC \cdot \cos < a,c > = AC^2$$

$$\Downarrow$$

Make three formula divide by OC^2, and x = OA/OC, y = OB/OC

$$x^2 + y^2 - 2xy\cos < a,b > = AB^2/OC^2$$

$$y^2 + 1^2 - 2y\cos < b,c > = BC^2/OC^2$$

$$x^2 + 1^2 - 2x\cos < a,c > = AC^2/OC^2$$

Then denote v = AB^2/OC^2, uv=BC^2/OC^2, wv = AC^2/OC^2,

$$x^2 + y^2 - 2xy\cos < a,b > - v = 0$$

$$y^2 + 1^2 - 2y\cos < b,c > - uv = 0$$

$$x^2 + 1^2 - 2x\cos < a,c > - wv = 0$$

Then we make v = $x^2 + y^2 - 2xy\cos < a,b >$, and substitute v in second and third formula.

$$(1-u)y^2 - ux^2 - \cos < b,c > y + 2uxy\cos < a,b > + 1 = 0$$

$$(1-w)x^2 - wy^2 - \cos < a,c > x + 2wxy\cos < a,b > + 1 = 0$$

The next process is how to solve the coordinates of A, B, C in the current camera coordinate system by the above two equations. The first thing to be clear is that which quantity is the known quantity. In our situation, $u = BC^2/AB^2$, $w = AC^2/AB^2$, $\cos < a,b >$, $\cos < b,c >$, $\cos < a,c >$ are known.

Because the distances of AB, BC, and AC can be obtained from the 3D points of the input, and the input 2D points can solve the three cosine values. I will explain it in detail.

The first is to solve the cosine worth process according to 2D coordinates. The first is the transformation from pixel coordinates to normalized image coordinates. we assume the point A's pixel is $[A_u, A_v]^T$, its normalized coordinates is $[A_x, A_y, A_z]^T$ and based on the camera pinhole model, we get

$$A_x = \frac{A_u - c_x}{f_x}$$

$$A_y = \frac{A_v - c_y}{f_y}$$

$$A_z = 1$$

The we denote $N_A = \sqrt{({A_x}^2 + {A_y}^2 + {A_z}^2)}$, $A'_x = \frac{A_x}{N_A},\ A'_y = \frac{A_y}{N_A},\ A'_z = \frac{A_z}{N_a}$ similarlywe can get point B, pint C.

$$B_x = \frac{B_u - c_x}{f_x}$$

$$B_y = \frac{B_v - c_y}{f_y}$$

$$B_z = 1$$

$N_B = \sqrt{({B_x}^2 + {B_y}^2 + {B_z}^2)}$, $B'_x = \frac{B_x}{N_B},\ B'_y = \frac{B_y}{N_B},\ B'_z = \frac{B_z}{N_B}$.

$$C_x = \frac{C_u - c_x}{f_x}$$

$$C_y = \frac{C_v - c_y}{f_y}$$

$$C_z = 1$$

$N_C = \sqrt{({C_x}^2 + {C_y}^2 + {C_z}^2)}$, $C'_x = \frac{C_x}{N_C},\ C'_y = \frac{C_y}{N_C},\ C'_z = \frac{C_z}{N_C}$.

With the above values, the cosine value can be solved.

$$\cos < a, b > = A'_x \cdot B'_x + A'_y \cdot B'_y + A'_z \cdot B'_z$$

$$\cos < b, c > = B'_x \cdot C'_x + B'_y \cdot C'_y + B'_z \cdot C'_z$$

$$\cos < a, c > = A'_x \cdot C'_x + A'_y \cdot C'_y + A'_z \cdot C'_z$$

Then, $u = \frac{BC^2}{AB^2},\ \ w = \frac{AC^2}{AB^2}$, we can calculate them by 3D points of A, B and C in world coordinate system. Therefore, there are only x and y are unknown.

The next step is to solve a binary quadratic equation, which is difficult to solve, but this can be solved mathematically.

$$(1-u)y^2 - ux^2 - \cos< b,c >y + 2uxy\cos< a,b > +1 = 0$$

$$(1-w)x^2 - wy^2 - \cos< a,c >x + 2wxy\cos< a,b > +1 = 0$$

We need to use Wenjun Wu's method[]. After Wu's method, we will get 4 solution, and then we use the fourth point to validate which solution is most possible. After this, we can get A, B, C of their 3D coordinate in their camera coordinate system. Then, we use the ICP [27] method(3D-3D method) which we discussed in chapter 3 to calculate **R** and **t**.

3.4 Non-linear Optimization

In SLAM system model, we can describe it in mathematical way which include a movement equation and an observation equation:

$$\begin{cases} x_k = f(x_{k-1}, u_k) + w_k \\ z_k = h(y_k, x_k) + v_k \end{cases} \tag{3.6}$$

Here, x_k is a vector of robot's pose at time k, u_k represents the sensor input and w_k represents the noise of measurement. In our project, we set camera's pose to be robot's pose. y_k represents a set of landmarks observed at pose x_k, v_k represents the noise of observation and z_k represents the position of landmarks in image.

$$P(x|z,u) = \frac{P(z|x)P(x)}{P(z)} \propto P(z|x)P(x)$$

According to Bayes formula above, we can know that solving the maximum posterior probability is equal to the product of the maximum likelihood estimate and the prior. Because we know calculate the posterior probability directly is hard.

Thus, we can transfer problem of calculating maximum posterior probability of x to calculate the maximum likelihood estimate of x.

We denote Maximize a Posterior(MAP) of x is x^*_{MAP}:

$$x^*_{MAP} = argmaxP(x|z) = argmaxP(z|x)P(x)$$

Next step is to calculate the maximum likelihood estimate. There is a mode of a certain observation:

$$z_{k,j} = h(y_i, x_k) + v_{k,j}$$

$$P(z_{j,k}|x_k, y_j) = N(h(y_j, x_k), Q_{k,j})$$

According to Gaussian distribution $x \sim N(\mu, \Sigma))$

$$P(x) = \frac{1}{\sqrt{(2\pi^N)det(\Sigma)}} exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$$

$$\Downarrow$$

$$-In(P(x)) = \frac{1}{2}In((2\pi^N)det(\Sigma)) + \frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)$$

Thus, we get

$$x^* = argmin((z_{k,j} - h(x_k, y_j))^T Q^{-1}_{k,j}(z_{k,j} - h(x_k, y_j)))$$

In our SLAM system, we get

$$F(x) = x^* = min(\sum_{k=1}^{n}(z_{k,j} - h(x_k, y_j))^T Q^{-1}_{k,j}(z_{k,j} - h(x_k, y_j)))$$

This is a Bundle Adjustment (BA) problem. In the SLAM back-end algorithm, we use Gauss Newton method to get the optimized pose.

We assume the Error function f(x) and current poses matrix are x. The Gauss Newton method's idea use first-order Taylor expansion on function f(x).

$$f(x + \Delta x) \approx f(x) + J(x)\Delta x$$

. Here, J(x) is the derivative of f(x), which is a Jacobian matrix. Our goal is to find the decent Δx, so that $||f(x+\Delta x)||^2$ is minimized.

In order to find Δ, we need to solve a linear least squares problem:

$$\Delta x^* = argmin \frac{1}{2}||f(x)+J(x)\Delta x||^2$$

Firstly, we expand the square term of the objective function:

$$\begin{aligned} \frac{1}{2}||f(x)+J(x)\Delta x||^2 &= \frac{1}{2}(f(x)+J(x)\Delta x)^T(f(x)+J(x)\Delta x) \\ &= \frac{1}{2}(||f(x)||_2^2+2f(x)^TJ(x)\Delta x+\Delta x^TJ(x)^TJ(x)\Delta x) \end{aligned} \tag{3.7}$$

Find the derivative of the above formula of Δx and make it:

$$2J(x)^Tf(x)+2J(x)^TJ(x)\Delta x = 0$$

$$J(x)^TJ(x)\Delta x = -J(x)^Tf(x)$$

3.4.1 Graph optimization

In our SLAM system, we used graph optimization which use a graph to represent the non-linear optimizing problem. Every node in the graph corresponds to a pose of the robot during mapping, meanwhile, every edge between two nodes corresponds to a spatial constraint between them. Therefore, Build the graph and find a node configuration that minimize the error introduced by the constraints.

Firstly, We define the error function(e_k). Error is typically the difference between the predicted and actual measurement.

$$e_k = z_k - h(x_k)$$

The squared error of a measurement depends only on the state and is a scalar which we talk above

$$(z_{k,j} - h(x_k, y_j))^T Q_{k,j}^{-1} (z_{k,j} - h(x_k, y_j))$$

In our SLAM system, the $F(x)$ is always non- linear, so our goal is to figure out $\frac{dF}{dx} = 0$. However, in the slam system, the form of F(x) is too complicated, so that we can not write form of its derivative or it is hard to calculate the the equation($\frac{dF}{dx} = 0$).Therefore, we have to use iterative method. The iterative stratagem is to find gradient descent direction and step size of gradient descent(Δx)by Gaussian-Newton Method and Levenberg-Marquardt method.

The Graph optimization step is following:

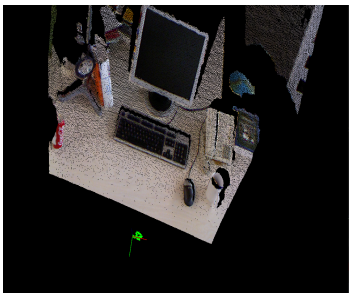
- we initialize Δx.
- In each iteration, we calculate the current Jacobian matrix and Hessian matrix.
- Solving sparse linear equations to get gradient descent direction

$$H_k \Delta x = -bx$$

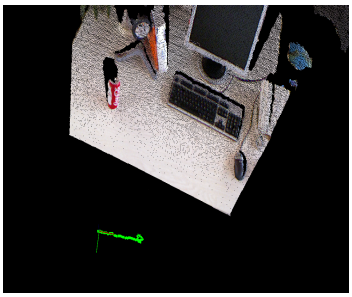
- Use Gaussian-Newton Method or Levenberg-Marquardt method to get iteration until iteration end.

3.5 Mapping

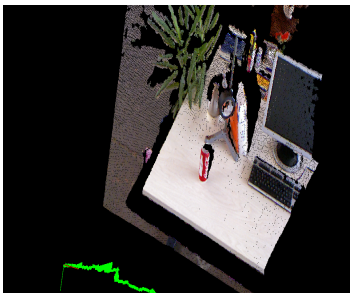
The Mapping module combines the previously calculated local maps, a set of landmarks, to construct a global map based on trajectory. The figure 3.2 shows the 3D reconstruction of global map when our SLAM system running on the TUM dataset [1]. We can easily see the 3D landmark such as computer, plants.



(a) Left: Mapping Result



(b) Middle: Mapping Result



(c) Right: Mapping Result

Figure 3.2: Our Dense Mapping Result of TUM Dataset [1]

Chapter 4

Implementation

In this Chapter, we introduce the detail of our visual SLAM system implementation based on Chapter 3 and hardware setup. Our RGB-D visual SLAM system architecture, see Figure 4.1, incorporates running on Nvidia Jetson TX2 and ZED Stereo Camera. We called this architecture as two-frame architecture (Ref-Cur frame architecture). System extract SURF [29] key-point from the reference frame and current frame firstly, and then computer their corresponding descriptors. Thirdly, system deal with feature matching for pose estimation. Fourthly, we create a local Bundle adjustment for pose estimation optimization and add a new keyframe to the local map.

This Chapter is structed as follows. In section 4.1, we introduce the Hardware equipment setup. We give overview of the software environment and discuss the details in each model of our visual SLAM architecture in sub-section of section 4.2.

4.1 Hardware Setup

Our hardware equipment includes ZED Stereo camera, Nvidia Jetson TX2 board with Ubuntu 16.04, Traxxas 1/10 scale rally car, etc.

The ZED Stereo camera supports depth perception and colour perception.

Figure 4.1: ZED Stereo Camera [2]

Nvidia Jetson TX2 is the fastest, most power-efficient embedded AI computing device. This 7.5-watt supercomputer on a module brings true AI computing at the edge. It's built

around an NVIDIA Pascal-family GPU and loaded with 8GB of memory and 59.7GB/s of memory bandwidth. It features a variety of standard hardware interfaces that make it easy to integrate it into a wide range of products and form factors[32].

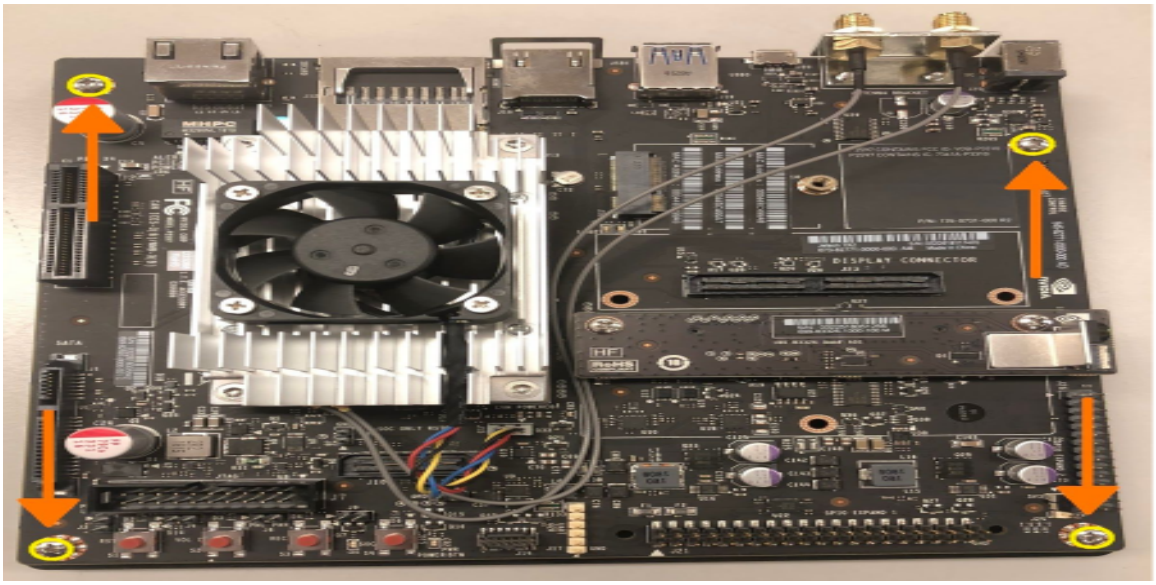


Figure 4.2: Nvidia Jetson TX2 Board [3]

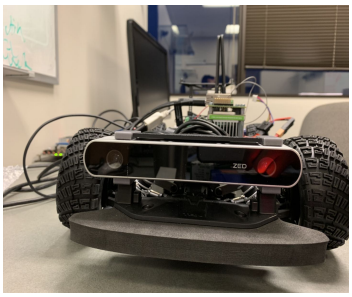
Then, we need to tune our VESC parameters to match them to our car. The VESC (which stands for Vedder Electronic Speed Controller) is a more advanced ESC which allows for features such as better motor and battery protection, regenerative braking, programming options like acceleration and deceleration curves, and other advanced features[33].

To control the car with keyboard, we developed a node called keyboard controller to connect the VESC driver node in ROS after tunning the VESC parameter.

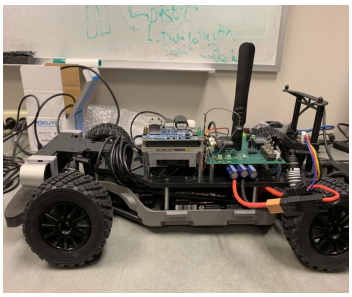
4.2 Software and Algorithm

Our system is a c++ project. These are the things that need to be installed on the Jetson TX2:

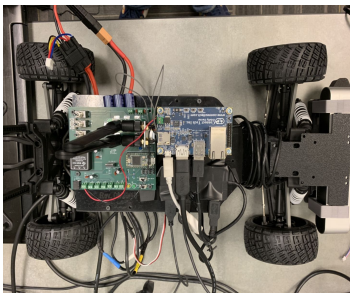
- **Ubuntu 16.04**: A free and open-source Linux distribution based on Debian [34].
- **ROS Kinetic**: Robot Operating System.



(a) Front View



(b) End View



(c) Vertical View

Figure 4.3: Hardware Setup

- **ZED SDK**: SDK for NVIDIA Jetson TX2.
- **OpenCV 3.3.1**: A library of programming functions mainly aimed at real-time computer vision [35].
- **Eigen3**: A high-level C++ library of template headers for linear algebra, matrix and vector operations, geometrical transformations, numerical solvers and related algorithms [36].
- **CUDA 9.1**: A parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs) [37].
- **Sophus**: A c++ implementation of Lie groups commonly used for 2d and 3d geometric problems [38].
- **g2o**: A c++ implementation of non-linear optimization library for graph optimization.

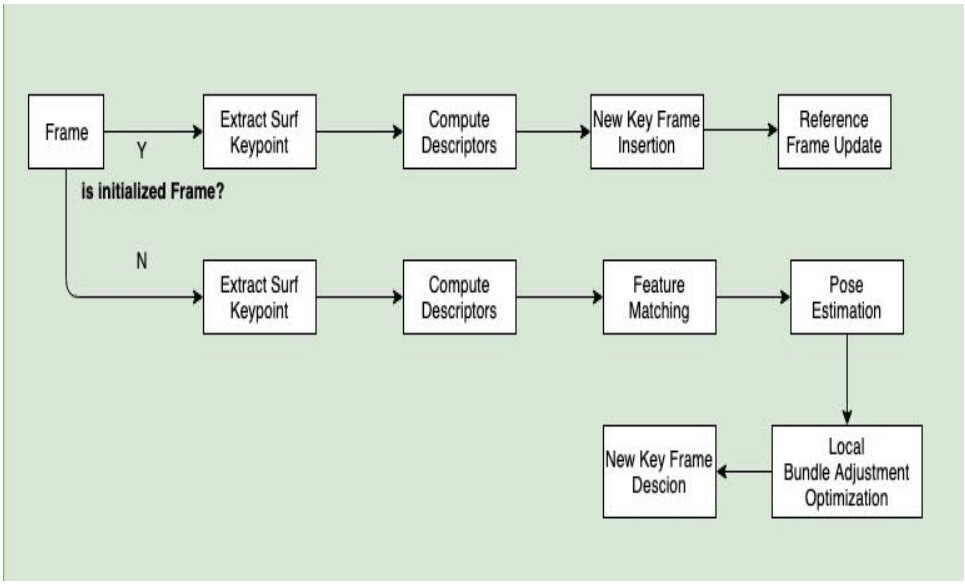


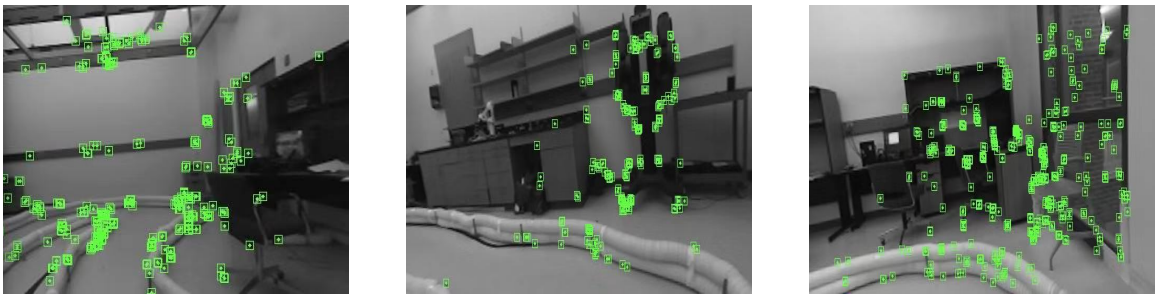
Figure 4.4: Our RGB-D Visual SLAM Architecture

Previously, we chose ORB, which are oriented multi-scale FAST corners with a 256 bits descriptor associated. We chose it because they are extremely fast to compute and match, but ORB detection has some limitations on accuracy. Thus, we choose SURF [29], which is a patented local feature detector and descriptor. SURF [29] uses square-shaped filters as an approximation of Gaussian smoothing. The first step consists of fixing a reproducible orientation based on information from a circular region around the interest point. Then we construct a square region aligned to the selected orientation and extract the SURF [29] descriptor from it. Our visual SLAM algorithm showed below.

4.2.1 SURF Keypoint Extraction

In our SLAM system, we choose SURF method[29] to extract the feature point. We will take detail in SURF keypoint extraction and implement in our SLAM system. In 2006, three people, Bay, H., Tuytelaars, T., and Van Gool, L [29], published another paper, SURF: Speeded Up Robust Features which introduced a new algorithm called SURF [29]. As the paper suggests, it is a speeded-up version of SIFT.

SURF [29] uses square-shaped filters as an approximation of Gaussian smoothing firstly



(a) Left Side in Lab (b) Middle in Lab (c) Right Side in Lab

Figure 4.5: SURF Extraction Result in FGH Lab 434

and then used hessian matrix determinant as an approximation value of image in each pixel. We implement SURF [29] Keypoint Extraction in our SLAM system by using xfeatures2d module via the OpenCV library.

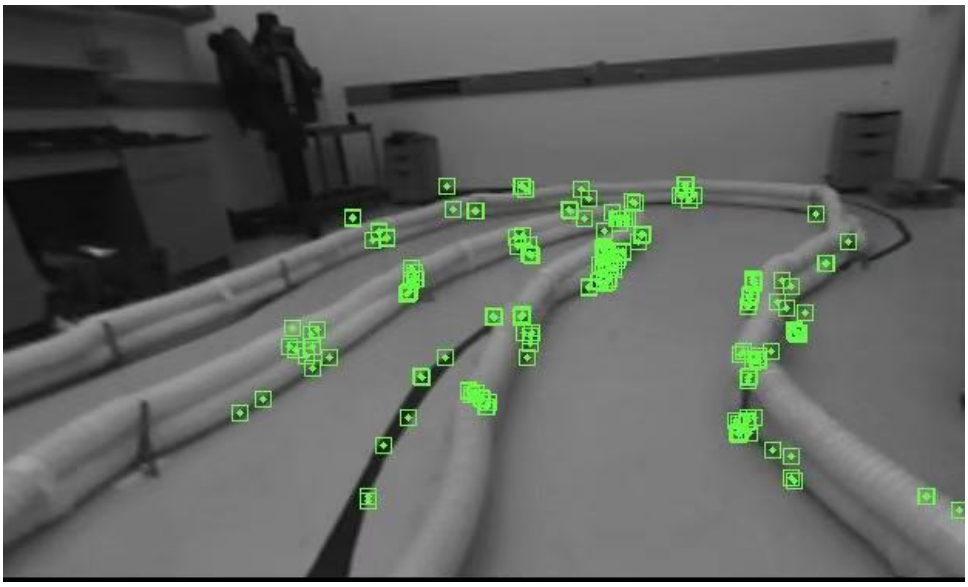


Figure 4.6: SURF Extraction Result in Car perspective

4.2.2 SURF Descriptors Compute

After the step of the SURF [29] keypoints extraction, we will enter step of descriptor computing according to figure 4.4. The descriptors are used for feature matching in our SLAM system.

The goal of a descriptor is to provide a unique and robust description of an image

feature, e.g., by describing the intensity distribution of the pixels within the neighborhood of the point of interest. Most descriptors are thus computed in a local manner, hence the description is obtained for every point of interest identified previously.

The dimensionality of the descriptor has a direct impact on both its computational complexity and point-matching robustness/accuracy. A short descriptor may be more robust against appearance variations, but may not offer sufficient discrimination and thus give too many false positives.

The first step consists of fixing a reproducible orientation based on information from a circular region around the interest point. Then we construct a square region aligned to the selected orientation, and extract the SURF [29] descriptor from it.

4.2.3 Feature Matching

The feature matching is used to estimate the camera's pose. In our SLAM system, SURF[29] feature descriptors are usually compared and matched using the Euclidean distance(L2-norm),since SURF [29] descriptors represent the histogram of oriented gradient (of the Haar wavelet response for SURF [29]) in a neighborhood, alternatives of the Euclidean distance are histogram-based metrics. Instead of brute-force matching, we use Flann-based matcher for SURF [29] feature matching in our system. FLANN is a library for performing fast approximate nearest neighbor searches in high dimensional spaces. It contains a collection of algorithms we found to work best for nearest neighbor search and a system for automatically choosing the best algorithm and optimum parameters depending on the dataset.Meanwhile, in order to eliminate the invalid matching, we set a empirical bound to filter out the invalid matching.

4.2.4 Pose Estimation

As we introduced on Chapter 3, there are three ways to estimate camera's pose. In our SLAM system, we chose P3P Method. P3P algorithm does not directly determine the

camera pose matrix based on the 3D-2D point. Algorithm calculate the 3D coordinates of the corresponding 2D point in the current camera coordinate system firstly. Secondly, algorithm combine the point of the 3D coordinates in the world coordinate and 3D point of the current camera coordinate system to solve the camera pose which transfer 3D-2D problem to 3D-3D problem.

In our system, we find an object pose from 3D-2D point correspondences using the RANSAC [39] scheme in OpenCV module. The function estimates an object pose given a set of object points, their corresponding image projections, as well as the camera matrix and the distortion coefficients. This function finds such a pose that minimizes reprojection error, that is, the sum of squared distances between the observed projections imagePoints and the projected (using projectPoints()) objectPoints. The use of RANSAC [39] makes the function resistant to outliers. The function is parallelized with the TBB library.

4.2.5 Local Bundle Adjustment Optimization on Pose Estimation

This module aims to optimize the pose estimation using Bundle Adjustment method vis the third party library **g2o**.

Table 4.1: Argument Explanation

P_k	The 3D position of a certain landmark in k^{th} frame coordinate
u_k	Landmark P_k's corresponding pixel in k^{th} frame
s_k	Landmark P's depth information in k^{th} frame coordinate
K	The intrinsic matrix of camera
x_k	The pose of camera in k^{th} frame

In our SLAM system, we define the error of reprojection point(P_k) is

$$e(x_k) = ||u_k - \frac{1}{s_k} K \exp(x_k \wedge) P_k||^2$$

Here, u This optimization problem can be easily constructed as an unconstrained optimization problem using Lie algebra, meanwhile, it also easily solved by optimization algorithms such as the Gauss-Newton method[40] and Levinberg-Makua's method[41]. In our system, we chose the Gauss-Newton optimization method. However, before using Gauss-Newton method, we need to know the derivative of each error term with respect to the perturbed variable in our SLAM system:

$$e(x_k + \Delta x_k) \approx e(x_k) + J\Delta x_k$$

, where J is a Jacobian matrix of our SLAM system.

$$J = \begin{bmatrix} \frac{f_x}{Z} & 0 & -\frac{f_x X}{Z^2} & f_x + \frac{f_x X^2}{Z^2} & -\frac{f_x Y}{Z} & -\frac{f_x Y}{Z} \\ 0 & \frac{f_y}{Z} & -\frac{f_y Y}{Z^2} & f_y + \frac{f_y Y^2}{Z^2} & \frac{f_x XY}{Z^2} & \frac{f_y X}{Z} \end{bmatrix}$$

Then using Gauss-Newton method $J(x_k)^T J(x_k)\Delta x_k = -J(x_k)^T e(x_k)$ to solve Δx_k which is optimized pose.

Finally, we listed the algorithm in our RGB-D visual SLAM system below. Each function in our SLAM algorithm has been introduced above in detail.

Algorithm 1 RGB-D Visual SLAM Algorithm

Input: Optical center c_x, c_y, Focal length f_x, f_y, State: Initializing

```
while true do
    if Camera Capture Images then
        color = Read RGB Image & depth = Read Depth image
        frame = createFrame();
        frame.color = color;
        frame.depth = depth;
        if State == Initializing then
            State = OK;
            curr = ref = frame;
            extractKeyPoints();
            computeDescriptors();
            setRef3DPoints();
        end
        else
            curr = frame;
            extractKeyPoints();
            computeDescriptors();
            featureMatching();
            poseEstimationPnP();
        end
        return rotation matrix R & translation vector t
    end
end
```

Chapter 5

Evaluation

We evaluated our SLAM system on TUM datasets [1] and real race car separately. We run our SLAM system with a ZED Stereo camera in an Interl Core i5-3317U desktop with 8 Gb RAM with NVIDIA GEFORCE 740 and also embedding system platform Nvidia Jetson TX2 respectively.

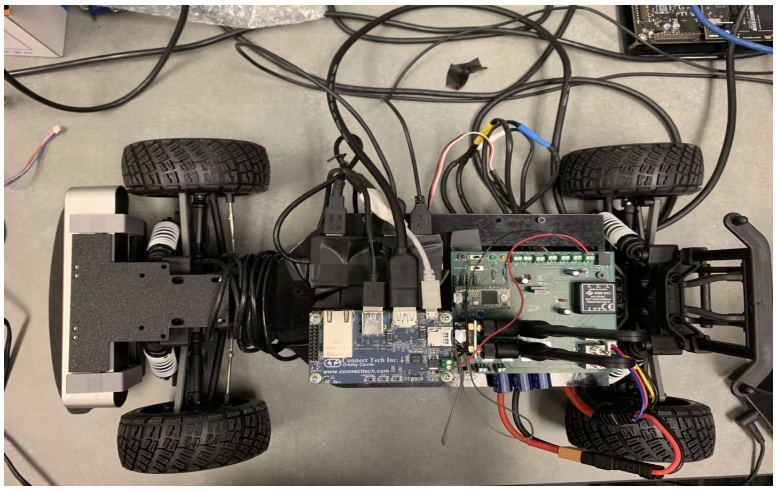


Figure 5.1: Experimental Setup

All dataset contains the color and depth images and also the groud-truth trajectory of the sensor for the evaluation of visual odometry systems.

5.1 TUM Sequence Dataset

For the TUM [1] dataset evaluation, our RGB-D SLAM system generates a trajectory (T_{SLAM}) consisting of a sequence of poses ($P_{k=1,\ldots,K} \in SE(3)$). The ground-truth trajectory is (T_{gt})consisting of a sequence of poses ($Q_{k=1,\ldots,K} \in SE(3)$). $SE(3)$ is a matrix which contains not only position vector of camera (Euclidean-based coordinate position x, y, z), but only rotation of camera. We define the relative pose error (RPE) which measures the local accuracy of the trajectory over a fixed time interval. Therefore, the relative pose error

corresponds to the drift of the trajectory which is in particular useful for the evaluation of visual odometry systems [1]. We define the relative pose error at time step k as

$$e_k = ||log(Q_k^{-1} P_k)||_2$$

From these errors above, we propose to compute the root mean squared error (RMSE) over all time from a sequence of K camera poses as

$$RMSE(e_{1:K}, \Delta) = \sqrt{\frac{1}{K} \sum_{k=1}^{K} {e_k}^2}$$

5.1.1 TUM Sequence Freiburg1 _ xyz Dataset

For this sequence, the Kinect is pointed at a typical desk in an office environment. This sequence contains only translatory motions along the principal axes of the Kinect, while the orientation was kept (mostly) fixed.

The figure 5.2 shows the evaluation result for TUM freiburg1 _ xyz dataset, the red color represents the trajectory we get by running our RGB-D SLAM algorithm, and the blue color represents the ground-truth trajetory. It is clear that the two trajectories match very well, indicating the correctness of our method.

Duration: 30.09s

Duration with ground-truth: 30.00s

Ground-truth trajectory length: 7.112m

Avg. translational velocity: 0.244m/s

Avg. angular velocity: 8.920deg/s

Trajectory dim.: 0.46m x 0.70m x 0.44m

RMSE: 0.013347m

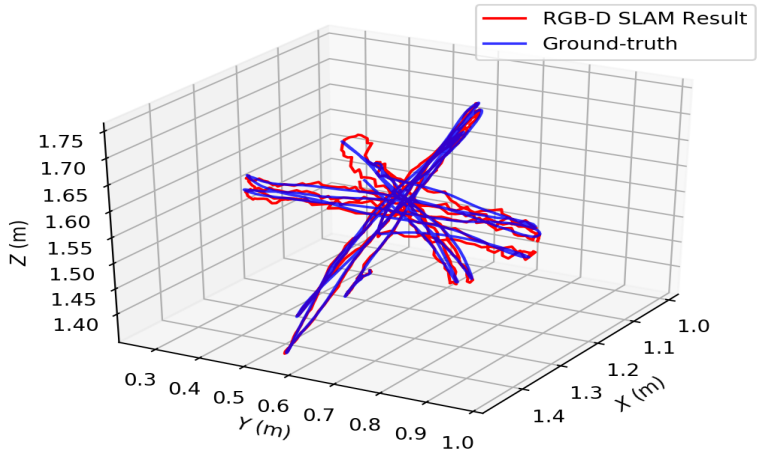


Figure 5.2: Result of Sequence Freiburg1_xyz Dataset

5.1.2 TUM Sequence Freiburg2_xyz Dataset

This sequence contains very clean data for debugging translations. The Kinect was moved along the principal axes in x-, y- and z-direction very slowly. The slow camera motion basically ensures that there is (almost) no motion blur and rolling shutter effects in the data.

The figure 5.3 shows the evaluation result for TUM freiburg2_xyz dataset, the red color represents the trajectory we get by running our RGB-D SLAM algorithm, and the blue color represents the ground-truth trajetory. It is also clear that the two trajectories match very well, indicating the correctness of our method.

Duration: 122.74s

Duration with ground-truth: 121.48s

Ground-truth trajectory length: 7.029m

Avg. translational velocity: 0.058m/s

Avg. angular velocity: 1.716deg/s

Trajectory dim.: 1.30m x 0.96m x 0.72m

RMSE: 0.010593m

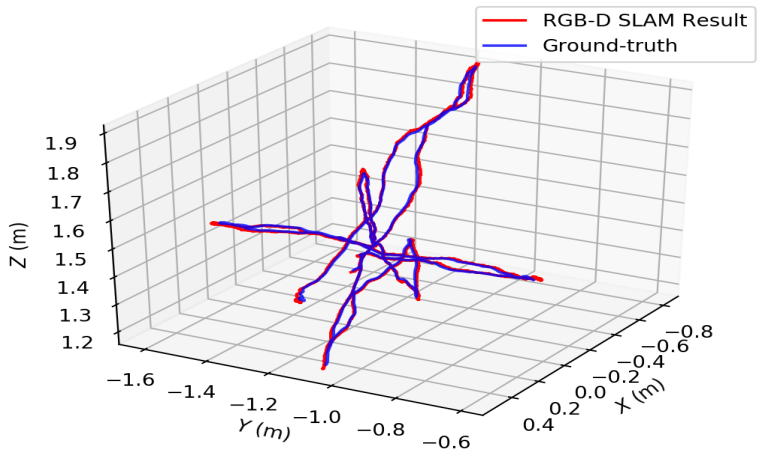


Figure 5.3: Result of Sequence Freiburg2_ xyz Dataset

5.1.3 TUM Sequence Freiburg2_ rpy Dataset

This sequence contains very clean data for debugging rotations. The Kinect was turned around the principal axes very slowly on the spot (RPY stands for roll-pitch-yaw). The slow camera motion basically ensures that there is (almost) no motion blur and rolling shutter effects in the data.

The figure 5.4 shows the evaluation result for TUM freiburg2 _ rpy dataset, the red color represents the trajectory we get by running our RGB-D SLAM algorithm, and the blue color represents the ground-truth trajetory. It is clear that the two trajectories match very well, indicating the correctness of our method.

Duration: 109.97s

Duration with ground-truth: 108.86s

Ground-truth trajectory length: 1.506m

Avg. translational velocity: 0.014m/s

Avg. angular velocity: 5.774deg/s

Trajectory dim.: 0.21m x 0.22m x 0.11m

RMSE: 0.014127m

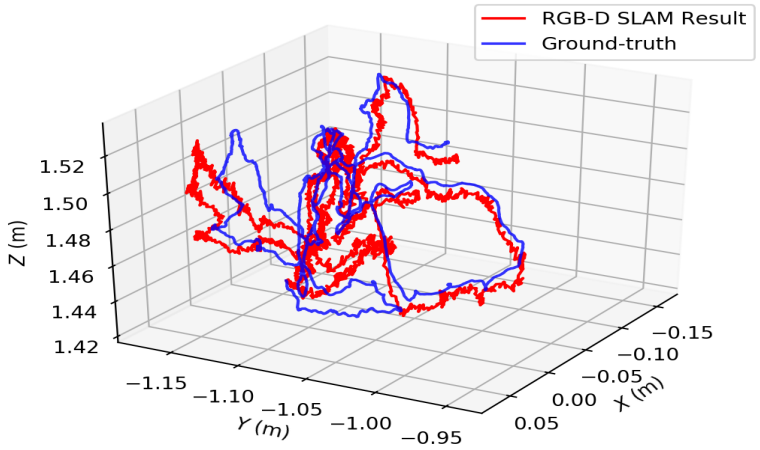


Figure 5.4: Result of Sequence Freiburg2_ rpy Dataset

5.2 Real-World Test on F1/10 Race Car

In this section we test and visualize our SLAM system on the F1/10 race car, meanwhile we compare our trajectory with the ground truth and the CUWB(Ciholas Ultra-Wideband) Server. The CUWB Server is a graphical user interface (GUI) and UWB network manager application[42]. CUWB Server provides users with a GUI and a 2D visualization of the location data.

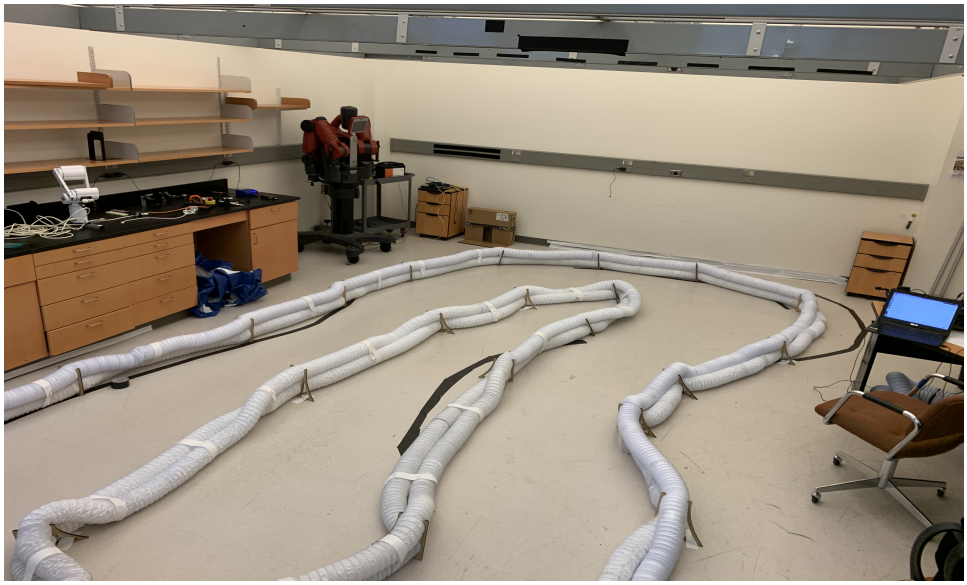


Figure 5.5: Real Route For Test

5.2.1 Visualization of Straight Route Test

In this section, we test our RGB-D SLAM system on a straight section of the track.

For Figure 5.7, 5.9, and 5.11, the blue square represents the trace of the race car in the previous time steps. The green square at the end of the blue trace is the current position of the race car. The black points represent feature points generated by the blue trace. And the red points represent feature points generated by the current position. The figures (5.8, 5.10 and 5.12) on the right is car's perspective to extract feature points. The green squares in the figure 5.8, figure 5.10, figure 5.12 are SURF feature points.

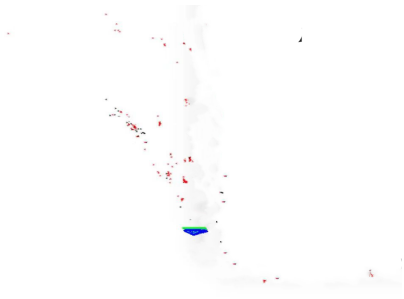


Figure 5.6: Localization at Beginning

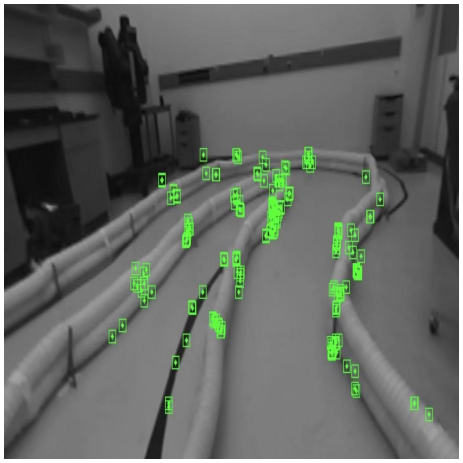


Figure 5.7: Car's perspective at Beginning

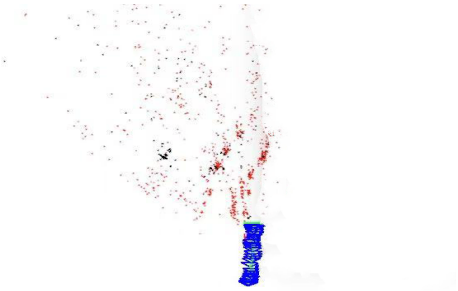


Figure 5.8: Localization at Middle

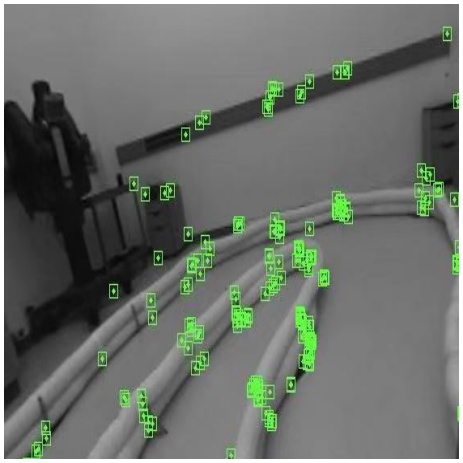


Figure 5.9: Car's perspective at Middle

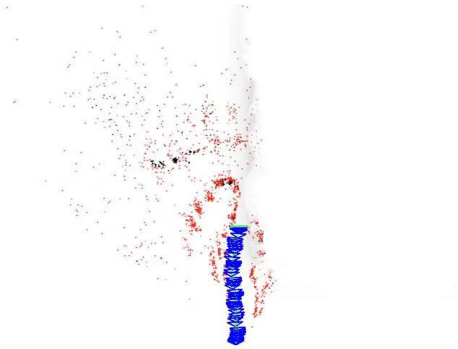


Figure 5.10: Localization at End

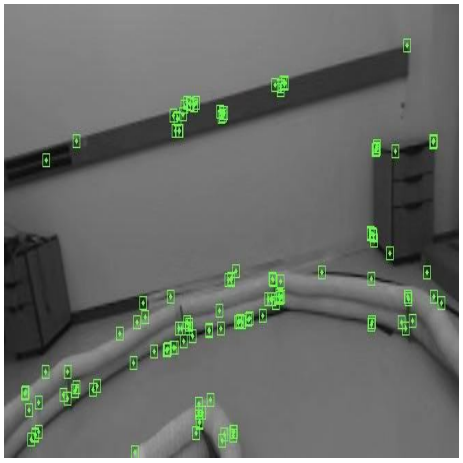


Figure 5.11: Car's perspective at End

5.2.2 Visualization of Curve Route Test

In this section, we visualize the localization estimation result on the curve route.

For Figure 5.13 and 5.15, the blue square represents the trace of the race car in the previous time steps. The green square at the end of the blue trace is the current position of the race car. The black points represent feature points generated by the blue trace. And the red points represent feature points generated by the current position. The figures (5.14 and 5.16) on the right is car's perspective to extract feature points. The green squares in the figure 5.14 and figure 5.16 are SURF feature points.

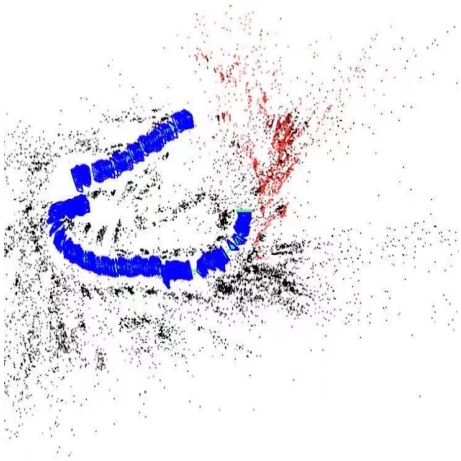


Figure 5.12: Localization at Curve

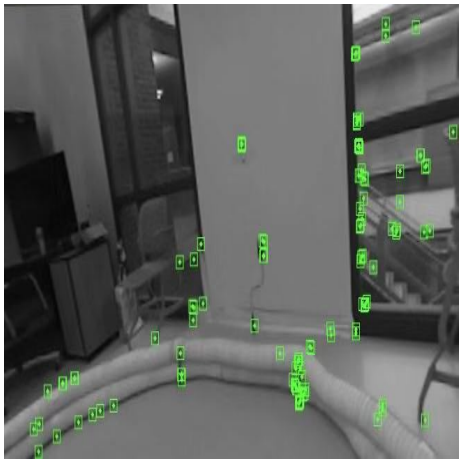


Figure 5.13: Car's Perspective at Curve

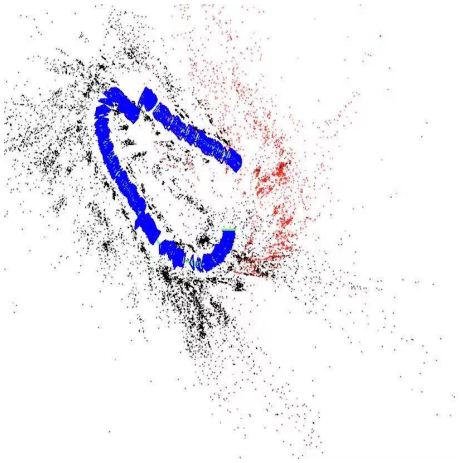


Figure 5.14: Localization at Curve

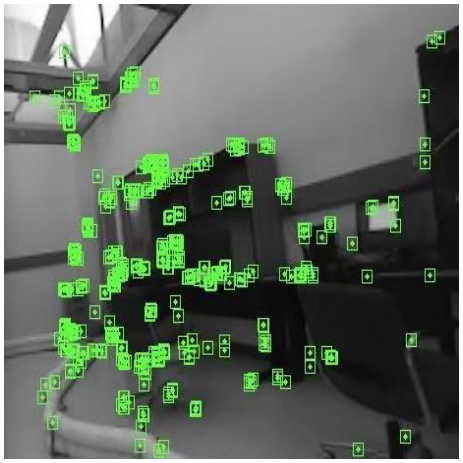


Figure 5.15: Car's Perspective at Curve

5.2.3 Visualization of Total Route Test

In this section, we use the entire track to test our SLAM system.

For Figure 5.17, 5.19 and 5.21, the blue square represents the trace of the race car in the previous time steps. The green square at the end of the blue trace is the current position of the race car. The black points represent feature points generated by the blue trace. And the red points represent feature points generated by the current position. The figures (5.18 and 5.20) on the right is car's perspective to extract feature points. The green squares in the figure 5.18 and figure 5.20 are SURF feature points. The figure 5.22 shows shape of real track.

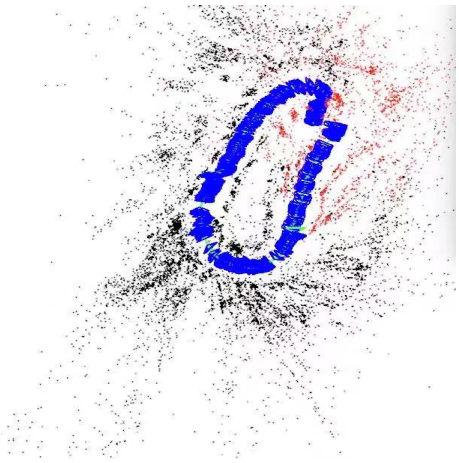


Figure 5.16: Localization Estimation of Total Route

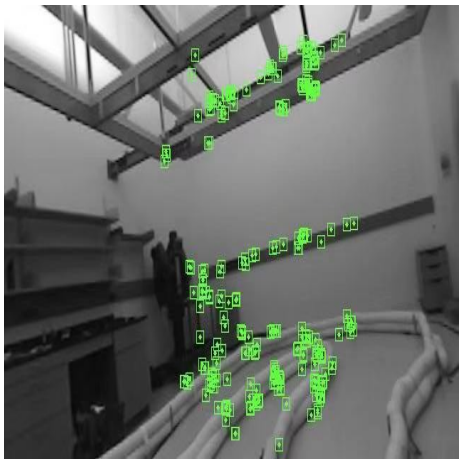


Figure 5.17: Car's Perspective

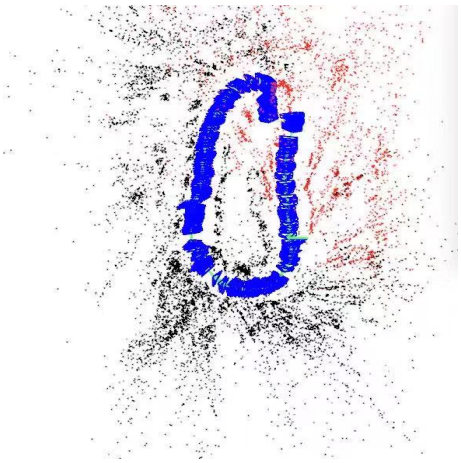


Figure 5.18: Localization Estimation of Total Route

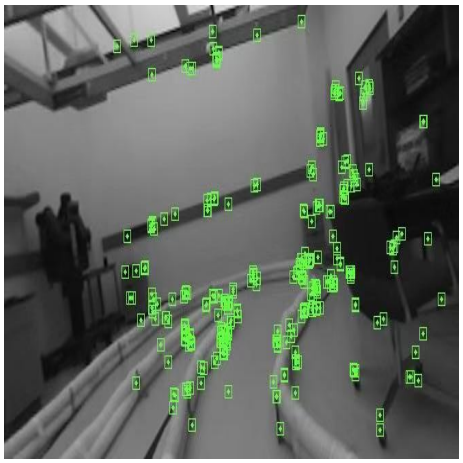


Figure 5.19: Car's Perspective

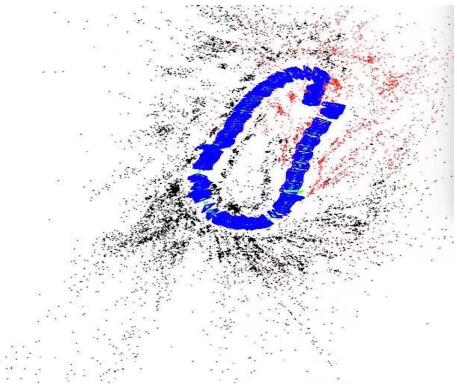


Figure 5.20: Total Trajectory By Our RGBD-SLAM

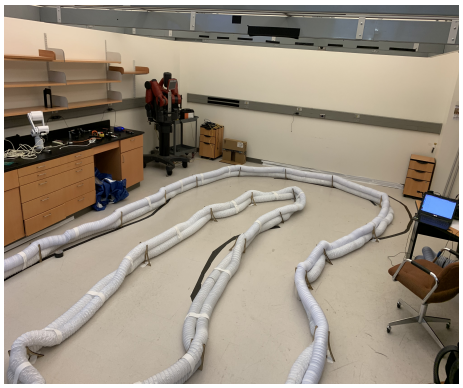


Figure 5.21: Real Track

5.2.4 Comparison with Ground-Truth Measurement

In this section, we show the comparison between our trajectory, CUWB location measurement system and ground-truth measurement.

CUWB location measurement is a CUWB Server which performs Real Time Location System (RTLS) calculations producing location data that is broadcast over ethernet for use by external applications [42]. The CUWB Server contain several devices and assign devices in three different roles.

First role is called master. Master is responsible for configuring the network, collecting

data from UWB based backhaul devices, and maintaining a global time reference for the system[42]. The Master is connected directly to our PC which is close the track. In our ground-truth measurement system, we use the Master position as a world coordinate.

Second role is called Anchor. Anchor is capable of collecting transmissions from other devices[42]. In our measurement system, we put five Anchor devices on the wall which are assumed to be stationary in position.

Third role is called Tag. Tag emits periodic transmissions that are received and timestamped by the Anchors[42]. Tags are usually mobile devices in the ground-truth measurement system, therefore, Tag device is connected directly to our Race car.

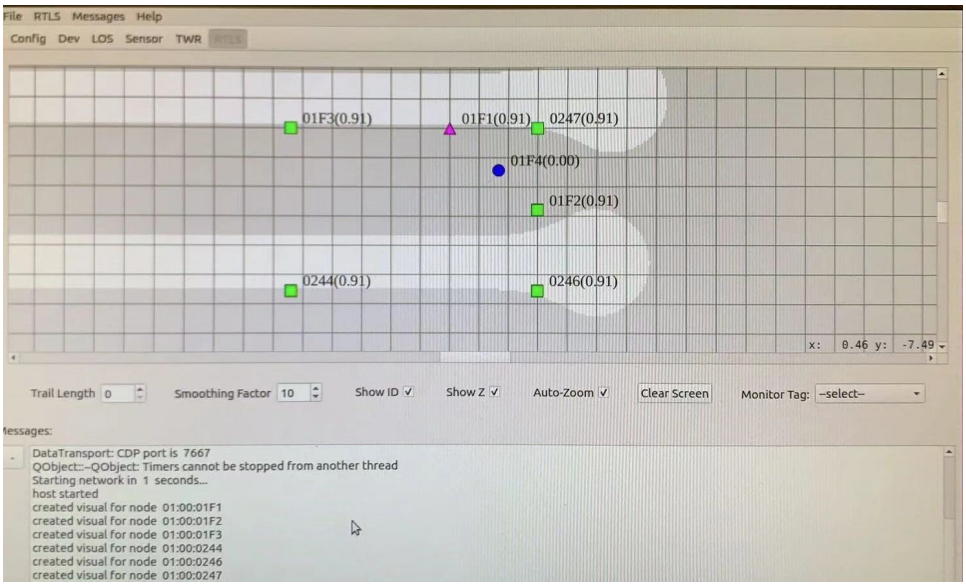


Figure 5.22: CUWB Server Dashboard

The figure 5.22 shows the pink point is a position of Master device, green points are positions of Anchor devices, and the blue point is the Tag(Race car).

The coordinate in ZED camera is quite different with our CUWB location measurement system(World Coordinate). In our world coordinate, the positive direction of Y axis is toward to # 01F3 Anchor device, however, in our ZED Camera coordinate, the positive direction of Z axis is the same direction as Y axis in our CUWB location measurement system. Therefore, we have to calculate rotation matrix between two coordinates.

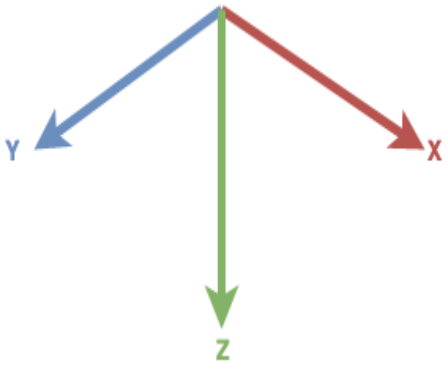


Figure 5.23: World Coordinate System

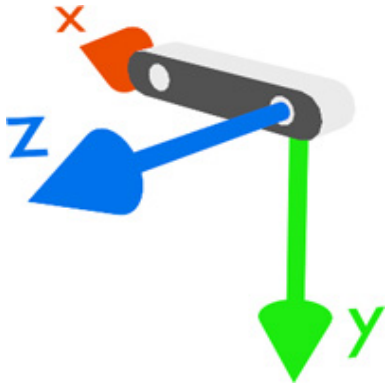


Figure 5.24: ZED Camera Coordinate System [4]

Figure 5.23 and 5.24 shows the difference of two coordinates. The positive direction of Z axis(Blue axis in figure 5.24) is the same direction as Y axis(Blue axis in figure 5.23) in our CUWB location Measurement system. We will ignore the vertical axis, because our race car is running in the 2D plane. The rotation matrix (R) which transfer ZED coordinate to World coordinate as following.

$$R = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$$

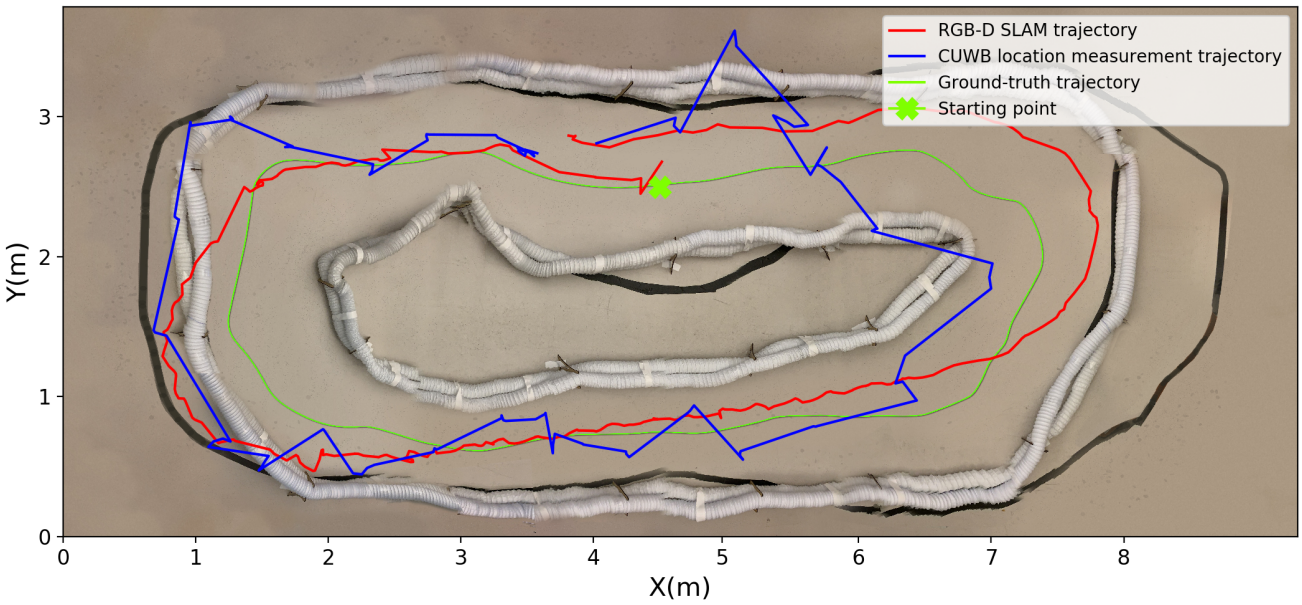


Figure 5.25: Comparison Results With Ground-truth Trajectory

We run the race car along the track deployed in Featheringill Hall 343. We define the starting point in the track where the car begins moving by the green cross marker. And the car runs along the track for one lap and goes back to the starting point.

Figure 5.25 shows the comparison between the trajectory obtained by RGBD-SLAM algorithm(red color), CUWB location measurement system(blue color), and ground-truth trajectory (green color). The ground-truth trajectory is a line depicted on the floor. We make the race car move following this line. One can find from the figure that the CUWB trajectories do not agree with others very well, because the CUWB location measurement system has drifts. And in fact, the trajectory obtained by running our RGBD-SLAM algorithm fits to the ground-truth track better than the CUWB locaiton measurement obtained from CUWB server.

For the real-world evaluation, since CUWB location measurement and ground-truth only contain position information (x, y, z) in the Euclidean-based coordinate, we use Haus-

dorff distance [43] to measure the distance between two trajectories.

The Hausdorff distance (d_H) measures how far two trajectories in a metric space are from each other [43]. Informally, two trajectories are close in the Hausdorff distance if every point of either trajectory is close to some point of the other trajectory. The Hausdorff distance is the greatest of all the distances from a point in one trajectory to the closest point in the other trajectory.

$$d_H(X,Y) = \max \sup_{x \in X} \inf_{y \in Y} d(x,y), \sup_{y \in Y} \inf_{x \in X} d(x,y)$$

Here, X and Y are two non-empty subsets of a metric space (M,d). sup represents the supremum and inf the infimum [43].

Table 5.1: Comparison of Hausdorff Distance

$d_H(X,Y)$ X / Y	RGB-D SLAM	CUWB Location Measurement
Ground-truth	0.634 m	1.057 m

Chapter 6

Conclusion

In this work, we present a SLAM system for RGB-D sensor that is able to perform localization and mapping in real-time on standard CPUs and embedded devices. We show that our SLAM system can successfully process images from various scenarios and can work on either standard PC platforms and embedded devices. We have released the source code of our system on Github [1].

Feature detection is used in landmark extraction and data association. It checks each pixel in an image to find a specific point that differs the other pixels. In this project, we used the SURF feature detection method and then used PnP method based on these points and 3D points of landmark to estimate the camera pose. We also implemented a local bundle adjustment for pose optimization. The map is generated by combining the cloud points of landmarks from different camera poses at different time steps. Once the 3D location of the landmark is determined, the mapping part is completed simultaneously.

Finally, we demonstrated that our SLAM system using feature detection based visual SLAM algorithm has the ability to realize place recognition from severe viewpoint changes. The system also keeps a balance between real-time and accuracy. Further, it can be acelerated by GPU. Moreover, another contribution of our work is that we deploy and test our SLAM system on a race car.

6.1 Discussion

There is still much room for improvement in this system. Firstly, although the method can achieve high accuracy when the noise level is low, the method is not robust to high-level noise.

[1] https://github.com/wwtx9/RGBD-SLAM

Secondly, the architecture of our system relies on the previous image (frame) for pose estimation. Therefore, if we lose the previous frame or the previous frame does not contain any information (e.g., a white wall), then the pose estimation will not be accurate.

Thirdly, since we need a pose estimation for every image, it is time-consuming.

Moreover, our system is very sensitive to input size and illumination, rendering it not applicable to our-door environment.

6.2 Future Work

In the future, in order to make the system more robust to noise and feature-lacking scenarios, we aim to include a loop-closing technique [22] that helps find the best-matching reference found by far when the reference image contains large noise or does not contain useful information.

Besides, we aim to reduce the calculation time by a direct method that does not calculate descriptors of the pixels but only calculate the feature points directly from the images. This will greatly reduce the time by matching the descriptors and feature points extraction.

Finally, it is worth investigating if adding other sensors such as IMU (Inertial Measurement Unit) can improve the accuracy/robustness of the SLAM system.

BIBLIOGRAPHY

[1] Jurgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. Rgb-d slam dataset and benchmark. *Computer Vision Group TUM Department of Informatics Technical University of Munich.*

[2] ZED Stereo Camera Stereolabs. https://www.stereolabs.com/zed/.

[3] Nvidia. Nvidia offical website.

[4] Coordinate Frames. https://www.stereolabs.com/docs/positional-tracking/coordinate-frames/.

[5] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 2006.

[6] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (slam): part ii. *IEEE Robotics Automation Magazine*, 2006.

[7] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose 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.*, 2016.

[8] Raul Mur-Artal and J. M. M. Montiel. Orb-slam: a versatile and accurate monocular slam system. *IEEE TRANSACTIONS ON ROBOTICS*, 2015.

[9] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment a modern synthesis. *ICCV*, 1999.

[10] M.I.A. Lourakis and A.A. Argyros. ”sba: A software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software.*, 2009.

[11] R.I. Hartley and A. Zisserman. Multiple view geometry in computer vision (2nd ed.). *Cambridge University Press.*, 2004.

[12] University of Oxford. Motivation. http://www.robots.ox.ac.uk/~SSS06/Website/Motivation.htm.

[13] Durrant-Whyte and Bailey. Simultaneous localization and mapping: Part i history of the slam problem. *the morning paper*, 2006.

[14] T. Bailey and H. F. Durrant-Whyte. Simultaneous localisation and mapping (slam): Part ii. *Robot. Auton. Syst.*, 2006.

[15] A. Flint, D. Murray, and I. D. Reid. Manhattan scene understanding using monocular, stereo, and 3d features. *Proc. IEEE Int. Conf. Comput. Vis*, 2011.

[16] S. Thrun, W. Burgard, and D. Fox. Probabilistic robotics. *MIT Press*, 2005.

[17] C. Stachniss and S. Thrunand J. J. Leonard. Simultaneous localization and mapping. *Springer Handbook of Robotics*, 2016.

[18] G. Dissanayake, S. Huang, Z. Wang, and R. Ranasinghe. A review of recent developments in simultaneous localization and mapping. *Proc.Int. Conf*, 2011.

[19] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose 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*, 2016.

[20] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, June 2007.

[21] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. *International Symposium on Mixed and Augmented Reality* ., 2007.

[22] Raul Mur-Artal and Juan D. Tardos. Orb-slam2: an open-source slam system for monocular, stereo and rgb-d cameras. *IEEE TRANSACTIONS ON ROBOTICS*, 2017.

[23] Jakob Engel, Thomas Schops, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. *European Conference on Computer Vision.*, 2014.

[24] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. *International Conference on Robotics and Automation.*, 2014.

[25] Matthew OKelly, Houssam Abbas, Jack Harkins, Chris Kao, Yash Vardhan Pant, and Rahul Mangharam. F1/10: An open-source autonomous cyber-physical platform. 2019.

[26] D. Hess, W.and Kohler and D. Rapp, H.and Andor. Real-time loop closure in 2d lidar slam. *IEEE International Conference In Robotics and Automation(ICRA)*, 2016.

[27] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision.*, 1994.

[28] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM.*, 1981.

[29] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *European Conference on Computer Vision*, 2006.

[30] D.V. Papadimitriou and T.J. Dennis. Epipolar line estimation and rectification for stereo image pairs. *IEEE Transactions on Image Processing.*, 1996.

[31] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. *Proceedings Third International Conference on 3-D Digital Imaging and Modeling.*, 2001.

[32] Nvidia jetson tx2. *https://developer.nvidia.com/embedded/buy/jetson-tx2*.

[33] Wikipedia: Electric skateboard. *https://en.wikipedia.org/wiki/Electric$_s$kateboard*.

[34] Ubuntu. *https://en.wikipedia.org/wiki/Ubuntu*.

[35] Kari Pulli, Anatoly Baksheev, Kirill Kornyakov, and Victor Eruhimov. Realtime computer vision with opencv. *Queue*, 2012.

[36] Eigen (c++ library). *https://en.wikipedia.org/wiki/Eigen$_(C$*

[37] Cuda zone. *https://developer.nvidia.com/cuda-zone*.

[38] Sophus. *https://github.com/strasdat/Sophus*.

[39] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Comm. ACM.*, 1981.

[40] A. Bjrck. Numerical methods for least squares problems. *SIAM, Philadelphia.*, 1996.

[41] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics.*, 1944.

[42] CUWB Server. https://cuwb.io/docs/v2.0/installation-and-configuration/user-manual/about-cuwb-server.

[43] R. Tyrrell Rockafellar and Roger J-B Wets. Hausdorff distance. *Variational Analysis. Springer-Verlag.*, 2005.