Chapter 2

Problems Set-up

This section is to understand the challenges of 3D vision problem, and what should we learn to solve it - this conducts to the contents of later chapters. Thus it is expected to focus on the main idea and scope, even for Maths, but not a detailed derivation of maths. Do not be afraid of if you do not understand a particular part as you will learn it in later chapters.

For a Robot or an Autonomous car to navigate autonomously on road, it is needed that it has a deep understanding of 3D environment which is at utmost important so that it can do path planing for obstacle avoidance or keeping a right path to reach to a target location. Assume that we have a robot or an autonomous car which mounts many sensors on it. Sensors might include LIDAR, RADAR, SONAR, PMD-camera, Stereo-Cameras, RGB-D Camera, Monocular-Camera, etc. Point Cloud (a set of 3D points) of surrounding environment is captured by active sensors (LIDAR, RGB-D camera, RADAR, SONAR, etc.) which tells us about objects' positions and structures. Due to a scattering effect (a farther object gets much less points then a near one), we actually do not get a **dense** Point Cloud, but a sparse one, which leads to the fact that partially only some parts of objects are captured by such kind of sensors.

RGB-D, PMD camera, or Kinect-camera can give a dense Point Cloud, and even telling the texture of corresponding 3D structure because it has an imaging sensor in it. Howver, those use infra-red light in their modulation, which are easily confused by the full spectra ligth from the sun - and thus they very not reliable in outdoor environment [14].

Stereo-vision, utilizing the overlapped area viewed by both camera, can provide a dense Point Cloud, but only for that overlapped region [19]. Anyhow camera is a passive sensor and works in a projective space, which limit its capacity in stably

Figure 2.1: Autonomous Mobile Robot at my former Robotic Lab, at University of Siegen (Champion ELROB 2007, First Innovation Award ELROB 2010).

capturing 3D information of environment.

A monocular camera basically only capture a 2D image. In computer vision, based on the idea of Structure from Motion, it is possible for it to estimate 3D positions of **static** environment.

In the scope of this book, we want to have a concrete understand of 3D environment from its structure and texture, regardless sensors in use. In fact, this means we need, in principle, a mechanism for capturing 3D structures and a camera for taking 2D images.

This leads to two core-problems to be solved as follow

- Mapping environmental structure and its texture; or a mapping between Point Cloud and 2D image
- Simultaneous Localization and Map Updating (SLAM)

Recently, due to a need of having a high accuracy dense Point Cloud, researchers have paid much attention on a mutual interaction between 2D and 3D spaces: the idea is to initially start with a given Point Cloud from an active sensor, but then by mapping and tracking in 2D image, it is possible to reconstruct a dense Point Cloud with a quite high accuracy and stability. Still this idea relies on the fundamental set-up of SLAM solution, therefore one can easily understand and realize such by following steps from the SLAM solution.

2.1 Mapping environmental structure and its texture

In many Robotics systems, researchers directly use Point Cloud given by a LIDAR or active sensors. Thus, they only need to do a 2D-3D mapping - note that this means both mapping from 3D world to 2D image and vice versa.

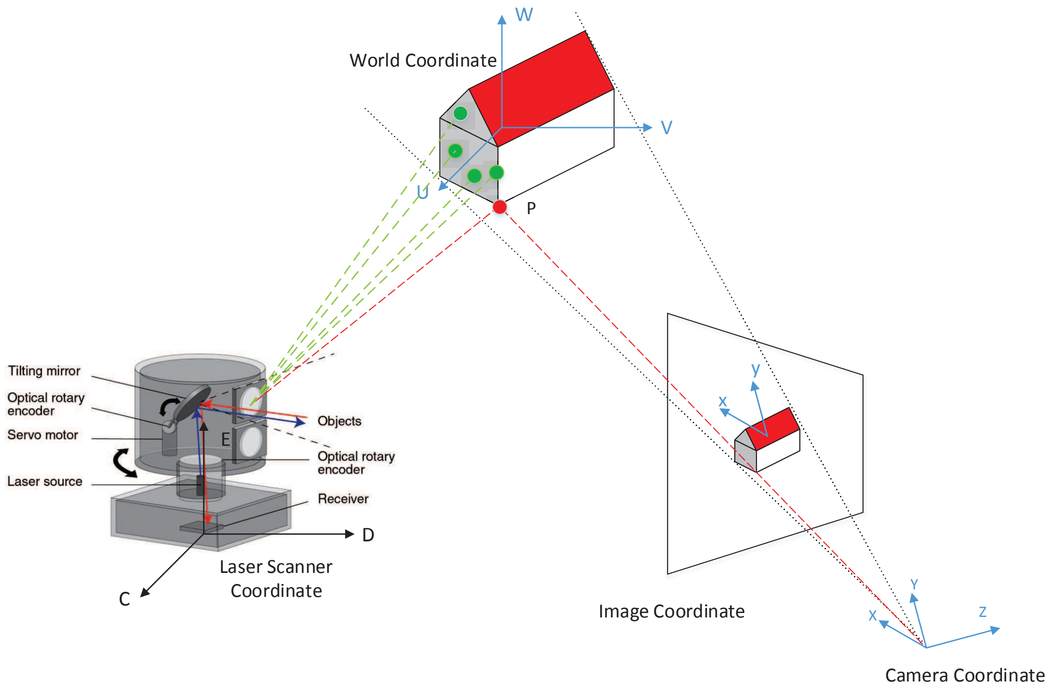

Figure 2.2: Mapping between LIDAR and Camera

For any sensors, before usage we should do sensor calibration to know where it is in our system and its principle operation. Thus, one of a key topic to learn is **Camera Calibration** as a pre-requisite to lear 3D vision. In this book, we will introduce the optic in the chapter 4. LIDAR calibration is trivial to be done, which is not described in this document.

Since sensor calibrations are done, we know the extrinsic transformation between

LIDAR and Camera coordinates, denote as Ω_{C2L}.

A world point P is viewed by a LIDAR at time t with a transformation from world to LIDAR denoted by Ω_{L2W}.

$$P_l = \Omega_{L2W} P$$

A projection of P on image

$$p = \Omega_{C2W} P = \Omega_{C2W} \Omega_{W2L} P_l \tag{2.1}$$

By doing such projection we can associate all 3D points from Point Cloud to image domain. However Point Cloud is a set of 3D points which are sparse, and thus a 3D rendering of environment is NOT perfect. Note that by Laser Scanner we cannot know which point matched to the other because they are just simple 3D point without any discriminative feature. In contrast, it is possible to do such in image domain as we can have many different descriptors to do that.

To improve the density of 3D points, we first track the pixel p in the image from t to t'. A world point P is viewed by the camera at time t'.

$$p' = \Omega'_{C2W} P$$

Combine the above equations, we have

$$p' = \Omega'_{C2W} \Omega_{W2L} P_l \tag{2.2}$$

From a set of points p which satisfy Equation 2.1, we can estimate the transformation

$$\Omega_{C2W} \Omega_{W2L} = p_{vec} P^T_{l-vec} (P_{l-vec} P^T_{l-vec})^{-1} = M_1$$

where a set of points p

$$p_{vec} = \begin{bmatrix} p_1 \\ p_2 \\ \dots \\ p_n \end{bmatrix}$$

A set of P_l points denoted by

$$P_{l-vec} = \begin{bmatrix} P_l 1 \\ P_l 2 \\ \dots \\ P_l n \end{bmatrix}$$

Therefore,

$$\Omega_{C2W} = M_1 \Omega_{W2L}^{-1} \tag{2.3}$$

We can apply **Land-Mark based approach or using IMU to determine** Ω_{W2L}. This is trivial, so do not discuss here.

From a set of points p' which satisfy Equation 2.2, we can estimate the transformation

$$\Omega'_{C2W}\Omega_{W2L} = p'_{vec}P_{l-vec}^T(P_{l-vec}P_{l-vec}^T)^{-1} = M_2$$

$$\Omega'_{C2W} = M2\Omega_{W2L}^{-1} \tag{2.4}$$

Equations 2.3 and 2.3 can be double-checked as follow

$$\Omega'_{C2W}\Omega_{C2W}^{-1} = M_2 M_1^{-1}$$

Since $p_{vec}, P_{l-vec}, \Omega_{W2L}$ are known, we can compute any transformation of camera poses.

Since knowing the camera motion or camera poses, by tracking points in images, we can triangulate them to obtain 3D points in world. This explains why we can achieve a dense Point Cloud. Examples of 3D rendering using such 2D-3D mapping are illustrated in Fig.2.3.

Overall, to understand 2D-3D Mapping, you need to learn at least

- Basics of Computer Vision and 3D Geometry
- Camera Calibration
- LIDAR Calibration
- Depth or Inverse Depth estimation

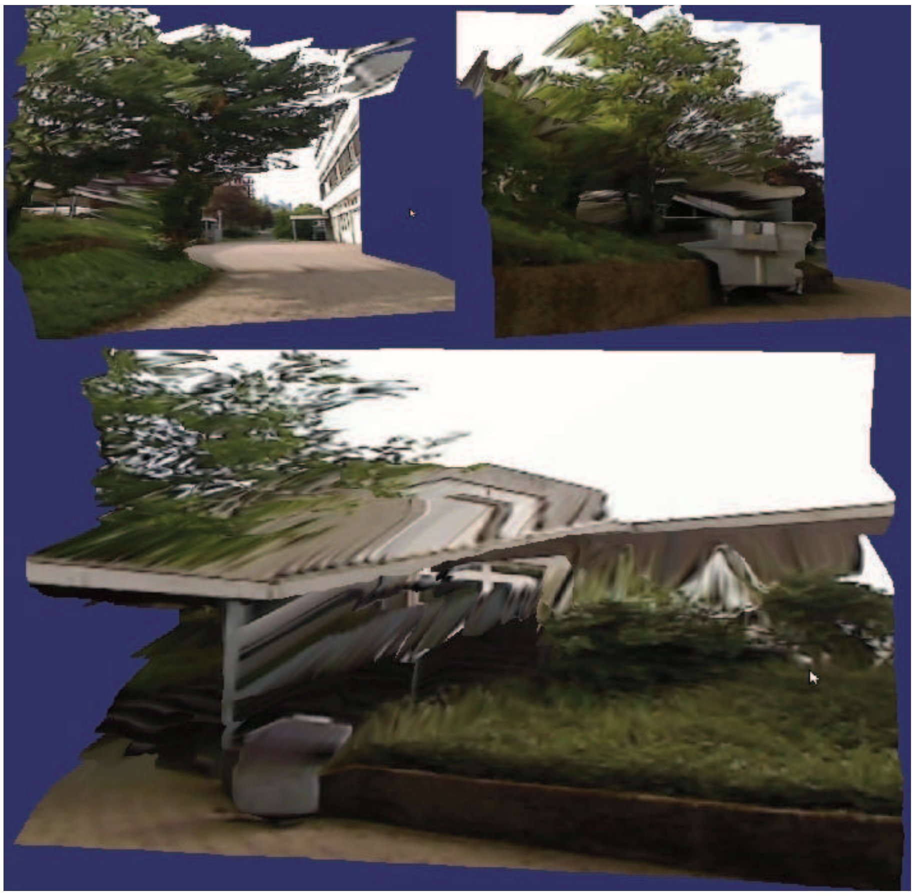
Figure 2.3: Examples of 2D-3D mapping between LIDAR and Camera [13]

- Machine Learning: Kalman Filter, Particle Filter, Bundle Adjustment, Gradient Descent, Least Square Solutions

2.2 Simultaneous Localization and Map Updating (SLAM)

In many Automotive systems, LIDAR is not preferable due to its expensiveness and low reliability in a mass production. In this case, only camera, RADAR, and SONAR are commonly used. For the purpose of reconstructing a dense Point Cloud of environment, it is done mainly by cameras because RADAR or SONAR only provide a very sparse data of object position (exceptionally modern near-range RADAR can provide a more details of objects' poses, which however is still pretty much in research & prototype phase, and thus we do not investigate in this document). By using a camera, based on the idea of Structure from Motion, we can reconstruct "static" environment **if we know the motion of camera's pose**. Also, **if we know the positions of surrounding objects** we can then estimate the motion of camera's pose. The problem of estimating both localization and 3D reconstruction is called the SLAM problem. Unfortunately we do not know either the motion of camera's pose or the positions of surrounding objects. Therefore the SLAM problem is well-known to be a **chicken and eggs** problem.

2.2.1 Abstract Formulation

Assume to have n 3D points in world, $\{P_1, P_2, .., P_n\}$ which are corresponding with N pixels in image, $\{p_1, p_2, .., p_n\}$. A transformation from a world coordinate to a camera coordinated is denoted by Ω_{CW}. A projection from the point in the camera coordinate into an image plane is denoted by ΩIC, see Fig.2.4

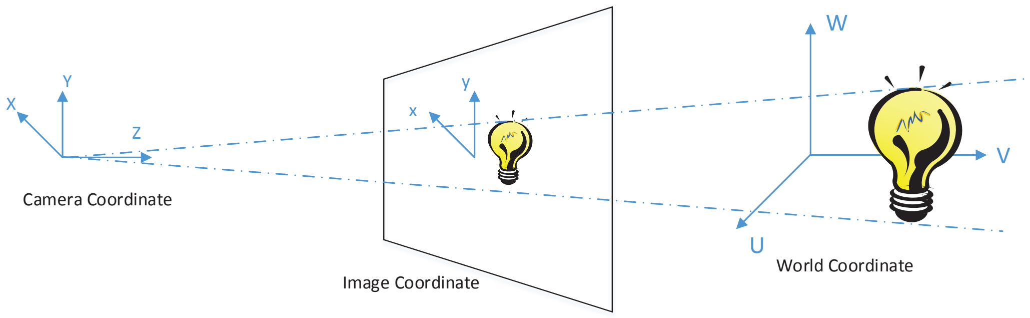

Figure 2.4: A simple Pin-hole camera model

Hence at location j of camera, mapping from 3D $P_i\{U, V, W\}$ to 2D $p_i\{x, y\}$

$$p_i = \Omega^j_{IC}\Omega^j_{CW}P_i \quad , \forall i \in \{1, n\}$$

Mapping from 2D $p_i\{x, y\}$ to 3D $P_i\{U, V, W\}$

$$P_i = (\Omega^j_{CW})^{-1}\Omega^j_{CI} \quad p_i \quad , \forall i \in \{1, n\} \tag{2.5}$$

Note that due to the imperfection of lens or camera component device, it is not always possible to have a parametric inverse $\Omega^j_{CI} = (\Omega^j_{IC})^{-1}$, especially for a Fisheye camera.

Camera moves to a location k, by tracking the world point P_i, we see it at p'_i in the image coordinate

$$p'_i = \Omega^k_{IC}\Omega^k_{CW}P_i \quad ,\forall i \in \{1, n\} \tag{2.6}$$

Substitute Equation 2.5 into Equation 2.6,

$$p'_i = \Omega^k_{IC}\Omega^k_{CW}(\Omega^j_{CW})^{-1}\Omega^j_{CI} \quad p_i \quad ,\forall i \in \{1, n\} \tag{2.7}$$

Multiply p_i in both side of Equation 2.7,

$$p'_i p_i^T = \Omega^k_{IC}\Omega^k_{CW}(\Omega^j_{CW})^{-1}\Omega^j_{CI} \quad p_i p_i^T \quad ,\forall i \in \{1, n\} \tag{2.8}$$

Assume that camera is calibrated, then Ω^j_{IC} and Ω^k_{IC} are known or can be estimated for each p_i, so called intrinsic parameters. $\{(p_i, p'_i), \forall i \in (1, n)\}$ are known from an image tracking process.

Equation 2.8 is in form of $Ax = b$. Thus, one needs to learn how to solve a **linear equation in form of $Ax = b$**. This is introduced in Chapter 3. Note that solving such matrix equation is not simple because it is not necessary to give a unique solution or having noise from the inputs (p_i, p'_i), and thus we need to work around with Machine Learning to stabilize our solution, which is described in Chapter 5. Due to the fact the Camera System, similar to Human-Eye System, works on a projective space, and thus estimating **Scale** is a big challenge or solution for the linear equation can have many at different scales. In many recent works, a bundle adjustment technique is used for a back-end of SLAM for a global optimization of all camera poses and 3D points.

To ease the problem, some assume to know a set of world points $\{P_{U_i,V_i,V_i}, i \in (j, k)\}$ which belong to a plane (typically Robotics researchers always assume that a small area at very front or back of the robot is road or a planar surface). In this case, we have an additional equation,

$$AU + BV + CW = 1 \tag{2.9}$$

A combination of of two equations 2.8 and 2.9 can lead to a unique solution. Due to a computational expensiveness in its processing (RANSAC selection of points and solutions, essential matrix decomposition, etc.) this approach can only provide a small set of 3D points for real-time application, and thus it is mainly used for estimating the Localisation - This solution is well-known in the computer vision community as often called **Visual Odometry**, see Fig.2.5.

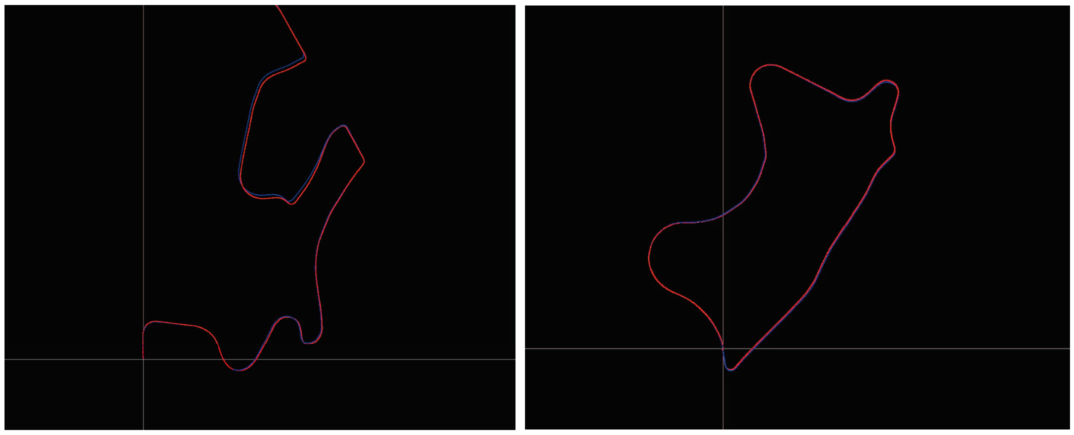

Figure 2.5: Example of Visual Odometry Results: Ground-Truth in Red; Our estimation in Blue. Our estimation is very good so that it is almost overlapped with the Ground-Truth

Alternatively to ease the problem, some can use additional sensors, like IMU or Vehicle CAN, to estimate the motion of vehicle or car, and thus deriving the motion of the camera. Whereby it becomes a triangulation problem with given camera poses. Example of 3D Reconstruction is shown in 2.6.

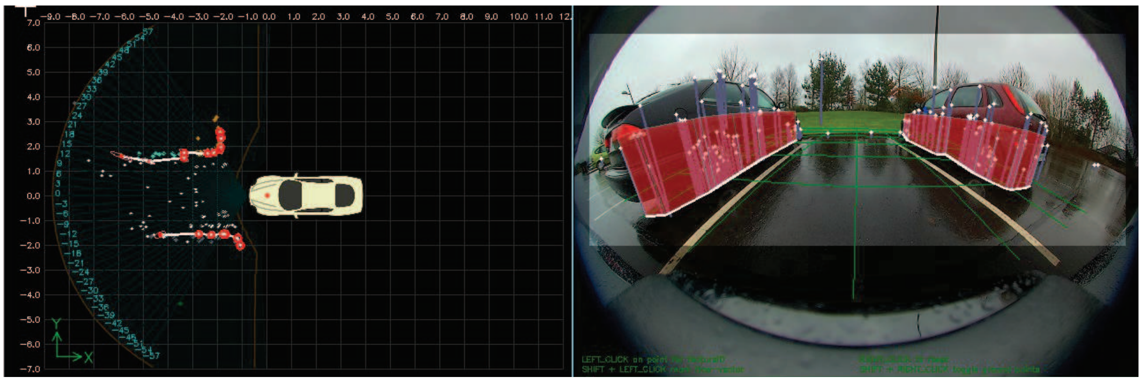

Figure 2.6: Example of 3D reconstruction [13]

Meanwhile camera pose is moving in centimetre range and we have to triangulate points at 10 meter range. This requires a very high accurate of camera pose estimation in order to correctly triangulate the points. Because motion estimation given by IMU or Vehicle CAN cannot be so accurate, it is usually a refinement step, local or global optimization process needed.

Overall, to understand SLAM, you need to learn at least

- Basics of Computer Vision and 3D Geometry
- Camera Calibration
- Epipolar Geometry
- Interpolation of Epipolar Curbs to apply Epipolar Geometry property for Fish-eye cameras
- Depth or Inverse Depth estimation
- Machine Learning: Kalman Filter, Particle Filter, Bundle Adjustment, RANSAC-based optimization, Gradient Descent, Gaussian-Newton, Levenberg Marquardt, and other Least Square Solutions

2.2.2 Direct-SLAM

The section 2.2.1 is known as Indirect-SLAM even though all fundamentals of SLAM is described there. The term "Indirect" was given only when there was a new SLAM technique introduced, the so called Direct-SLAM. The meaning of Indirect is that it uses traditional computer vision techniques, for example forming feature descriptors or building up discriminative representations of image, to enable feature tracking for image registration. The Direct-SLAM, on the other hand, directly associate photometric data between images, for example pixel intensity or colour, etc. Since we are directly matching pixels, non-discriminative features, between two images it is technically impossible to do it correctly following a normal matching procedure as commonly used in Indirect-SLAM method. First, formulation of Direct-SLAM is partially similar to an Indirect-SLAM, as mentioned above, but instead of tracking features between images, we directly minimizing the photometric errors between pixels, between the current image I and the reference image I_{ref} [3]

$$E(\xi) = \sum\nolimits_i (I_{ref}(p_i) - I(\omega(p_i, D_{ref}(p_i), \xi)))^2 \tag{2.10}$$

Where p_i is a pixel in I_{ref}; $\xi \in SIM(3)$ in Lie-Algebra, which will be explained later in Chapter 3; D is the inverse depth.

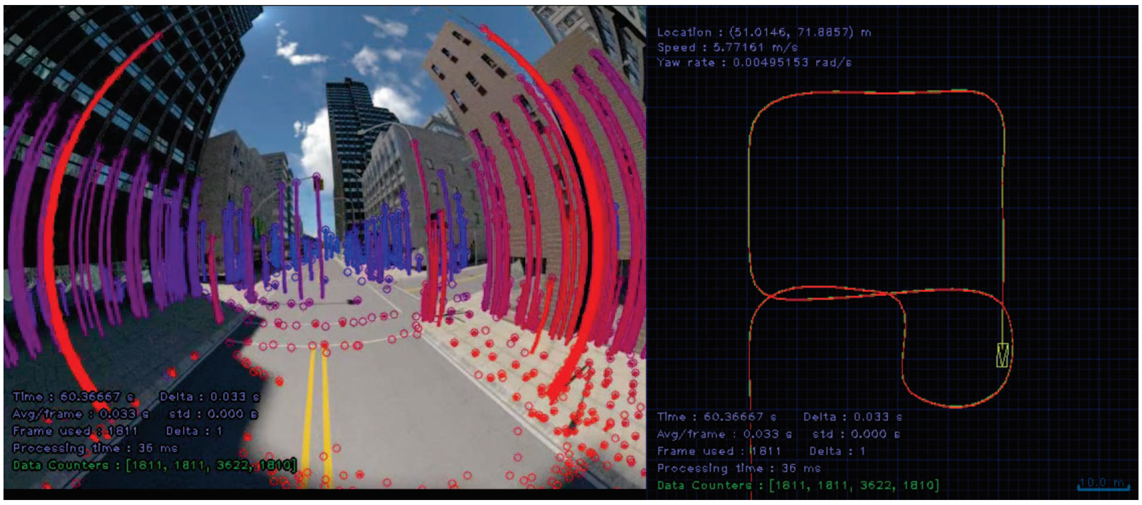
Location : (51.0146, 71.8857) m
Speed : 5.77161 m/s
Yaw rate : 0.00495153 rad/s
Time : 60.36667 s Delta : 0.033 s
Avg/frame : 0.033 s std : 0.000 s
Frame used : 1811 Delta : 1
Processing time : 36 ms
Data Counters : [1811, 1811, 3622, 1810]
10.0 m

Figure 2.7: Example of Direct-SLAM result [15]

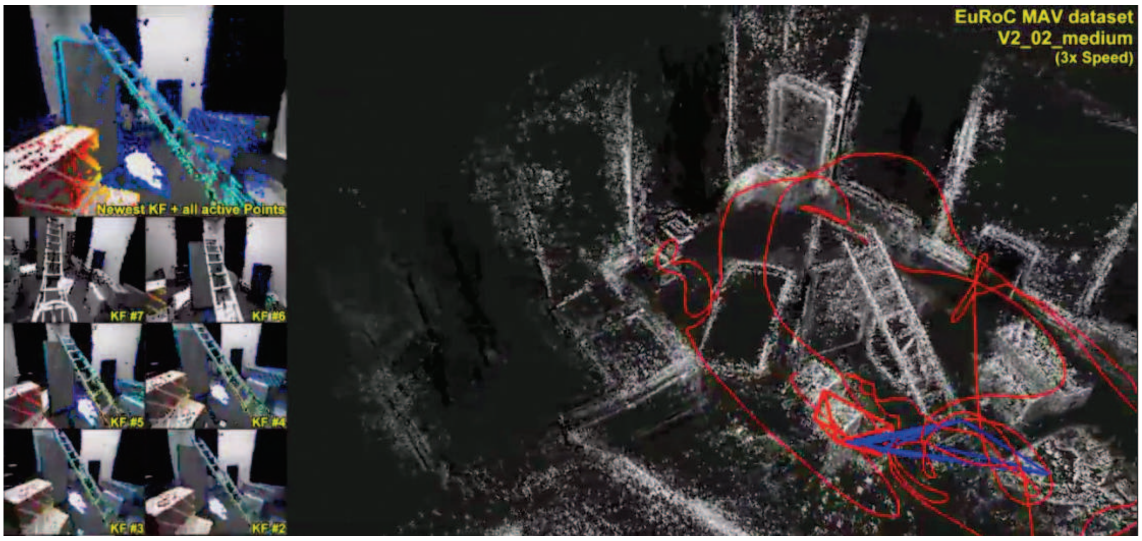

Figure 2.8: Example of Direct-SLAM result [5]

In general, if searching randomly through the whole image I, we would find so many pixel which has similar value as of p_i in I_{ref} because it is just simply an intensity ranging from 0 to 255. Therefore, the idea is to find a transformation which leads us to a correct solution or a correct matching. For that aim, we minimize the photometric error subject to a transformation ξ - this can be done by applying Gaussian-Newton or Levenberg Marquardt. For example using a Gauss-Newton second-order approximation of E, let denote the residual $r_i = (I_{ref}(p_i) - I(\omega(p_i, D_{ref}(p_i), \xi)))$, with some Maths operations, we have

$$\delta\xi^{(n)} = -(J^T J)^{-1} J^T r(\xi^{(n)}) \tag{2.11}$$

Starting with an initial estimate $\xi^{(0)}$, after n iteration, we expect to reach a convergence, or found a pixel location which minimizes the photometric error. Importantly, J is the Jacobian of the residual with respect to the transformation, and thus to compute J it is much easier to use the Lie Group transformation ξ, instead of a common Euclidean transformation. This will be explained in Chapter 3.

Additionally, in a Pinhole camera, we should utilize its Epipolar Geometry property to constraint the solution space; in a Fisheye camera, we can interpolate Epipolar Curbs to constraint the solution space.

Overall, to understand Diect-SLAM, you need to learn at least

- Basics of Computer Vision and 3D Geometry
- Camera Calibration
- 3D Transformation Group or Lie Groups SO(3), SE(3), SIM(3)
- Epipolar Geometry
- Interpolation of Epipolar Curbs to apply Epipolar Geometry property for Fisheye cameras
- Depth or Inverse Depth estimation
- Machine Learning: Kalman Filter, Particle Filter, Bundle Adjustment, RANSAC-based optimization, Gradient Descent, Gaussian-Newton, Levenberg Marquardt, and other Least Square Solutions