[BoldFont= $_GBK, ItalicFont = _GBK]_GBK[BoldFont = _GBK]_GBK[BoldFont = _GBK]_GBK_GBK_GBK_GBK_GBK_GBK_GBK$

视觉SLAM十四讲

从理论到实践

高翔　张涛　刘毅　颜沁睿　著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

SLAM
SLAM SLAM

CIP
SLAM / .— 2017.3
ISBN 978-7-121-31104-8
I. ① … II. ① …III. ① IV. ①TP18
CIP 2017 053910

173 100036
720×1000 1/16 25 560
2017 3 1
2018 1 7
75.00

010 88254888 88258888
zlts@phei.com.cn dbqq@phei.com.cn
010 51260888-819 faq@phei.com.cn

www.broadview.com.cn

-
-
-

http://www.broadview.com.cn/31104

SLAM

2016 2018 SLAM

1.

2.

3. SLAM

4.

5.

Contents

Chapter 1

1.1

SLAM SLAM
SLAM
SLAM **S**imultaneous **L**ocalization **a**nd **M**apping “ ”[1] [2]
SLAM SLAM “ ” “ ”
— SLAM

Computer Vision —— [3]
Intelligence Machine Learning —— SLAM 30
— SLAM
SLAM SLA
—
SLAM SLAM 21 **SLAM**
SLAM SLAM
1986 [4] SLAM SLAM SLAM
SLAM *Probabilistic robotics* [5] *Multiple View Geometry in Computer Vision* [3] *State Estimation for Robotics: A Matrix-Lie-Group Approach* [6]† SLAM SLAM SLAM
SLAM SLAM
“ ” §

*
†
‡
§

1.2

1.2.1

“ SLAM ” “ ” “ ” “ ”
— ” “ ”

* *

1. SLAM

- 1
- 2 SLAM SLAM IDE
- 3 Eigen
- 4 Sophus
- 5 OpenCV
- 6 Ceres g2o

2. **SLAM** SLAM

- 7 PnP ICP
- 8
- 9 Bundle Adjustment BA Ceres g2o BA
- 10 SE(3) Sim(3) g2o
- 11 dbow3
- 12 RGB-D RGB-D
- 13 Kitti
- 14 SLAM

1.2.2

GitHub
`https://github.com/gaoxiang12/slambook2`
2 7 ch7 3rdparty git submodule OpenCV GitHub Issues

*

1.2.3

SLAM

- * SLAM
- **C++** C++ C++ C++ 11
- **Linux** Linux Windows Linux Windows **Linux SLAM**

SLAM C++ *C++ Primer Plus* SLAM

1.3

1. $$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}. \tag{1.1}$$

   a, α $\boldsymbol{a}, \boldsymbol{A}, \boldsymbol{\Sigma}$ $\mathbb{R}$ $\mathbb{Z}$ $\mathfrak{so}(3), \mathfrak{se}(3)$

2. Listing 1.1:

```
#include <iostream>
using namespace std;

int main ( int argc, char** argv )
{
  cout<<"Hello"<<endl;
  return 0;
}
```

3. “ ” GitHub
4. GitHub GitHub
5.
6.
7.
8.
9.
10.

*

1.4

- 5
- 7
- 6 10
-

*

gao.xiang.thu@gmail.com

1. $\boldsymbol{Ax} = \boldsymbol{b}$ $\boldsymbol{A}, \boldsymbol{b}$ $\boldsymbol{x}$ $\boldsymbol{A}$ $\boldsymbol{b}$ $\boldsymbol{A}$
2.
3. C++ STL
4. C++ Visual C++ 6.0 C++ C++ C
5. C++11
6. Linux Ubuntu
7. Linux ls, cat
8. Ubuntu eigen
9.* Vim vimtutor **Vim IDE**

* “ ” “ ”

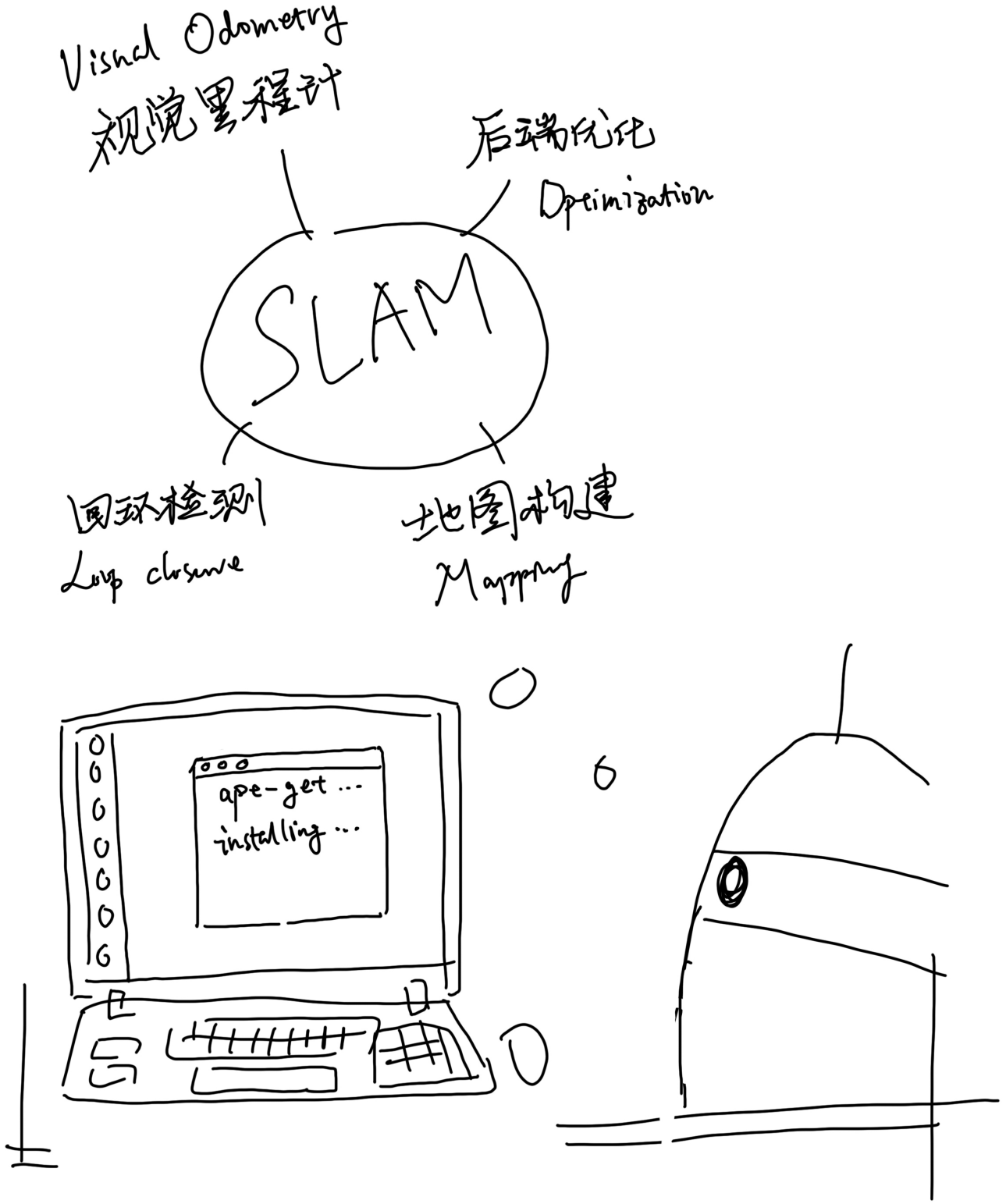
Part I

Chapter 2

SLAM

1. SLAM
2.
3. Linux
4. cmake

SLAM “Hello SLAM”


Visual Odometry
视觉里程计
后端优化
Optimization
SLAM
回环检测
Loop closure
地图构建
Mapping
ape-get ...
installing ...

2.1

“ ” Figure 2-1

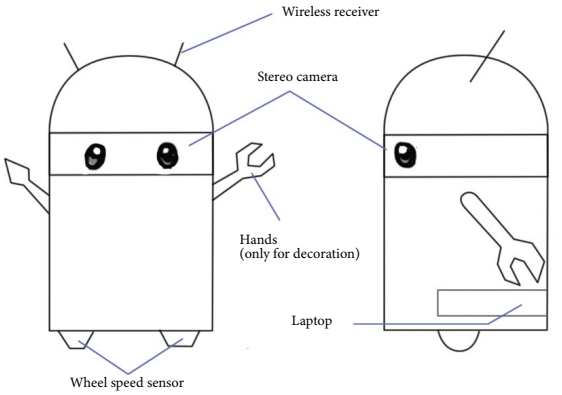


Figure 2-1:

“ ”

—

1. ——

2. ——

“ ” “ ” “ ” “ ” “ ”

2

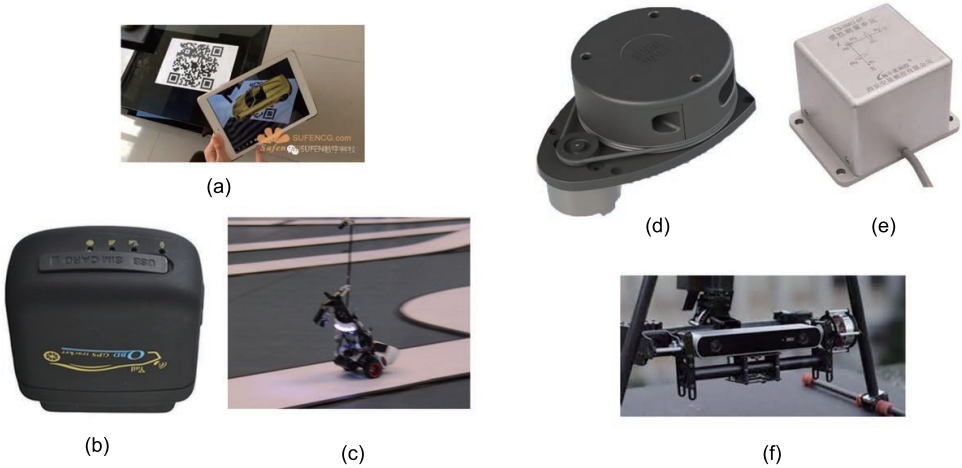


Figure 2-2: a b GPS c d e IMU f

Iner
Measurement Unit IMU IMU
SLAM SLAM † GPS SLAM SL
SLAM SLAM 30
D Figure 2-3 RGB-D 5

Figure 2-3:

SLAM

SLAM SLAM Monocular SLAM SLAM
Scene SLAM
4

Figure 2-4:

SLAM Motion Structure

---

* “ ”
†

— disparity

“ ”

Ambiguity

SLAM

— Figure 2-5

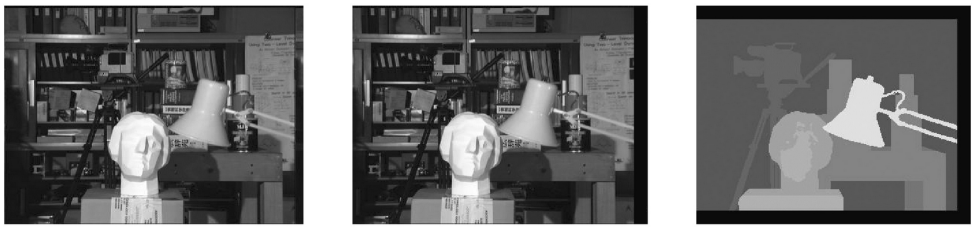


Figure 2-5:

†

RGB-D RGB-D 2010 Time-of-Flight ToF
6 RGB-D Kinect/Kinect V2 Xtion Pro Live RealSense RGB-
D SLAM

Figure 2-6: RGB-D

‡ SLAM

*
†
‡

2.2 SLAM

SLAM Figure 2-7 SLAM

Figure 2-7: SLAM

SLAM

1. SLAM
2. Visual Odometry VO VO Front End
3. Optimization VO Back End
4. Loop Closing
5. Mapping

SLAM SLAM

2.2.1

Figure 2-8 “ ”
8

Figure 2-8:

SLAM
VO VO VO “ ”
10
“ ” VO SLAM
SLAM Accumulating Drift
9 90° 90° 90° 89° −1°
Drift 90° —— † “ ”

2.2.2

SLAM “ ”
— Maximum-a-Posteriori MAP

*
† Back End

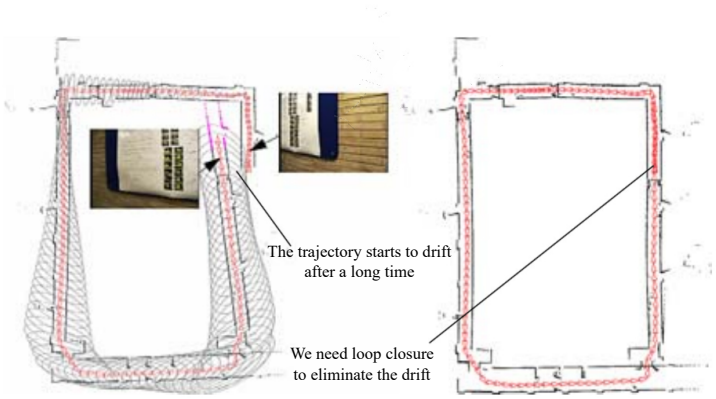


Figure 2-9: [10]

“ ” SLAM **SLAM**
“SLAM ” SLAM —— SLAM “ ” Spatial Uncertainty [4, 11] SLAM SLAM

2.2.3

Loop Closure Detection/Loop Closingn
“ ” “ ”
—
“A B ”

2.2.4

Mapping Figure 2-10 SLAM
6
3D SLAM

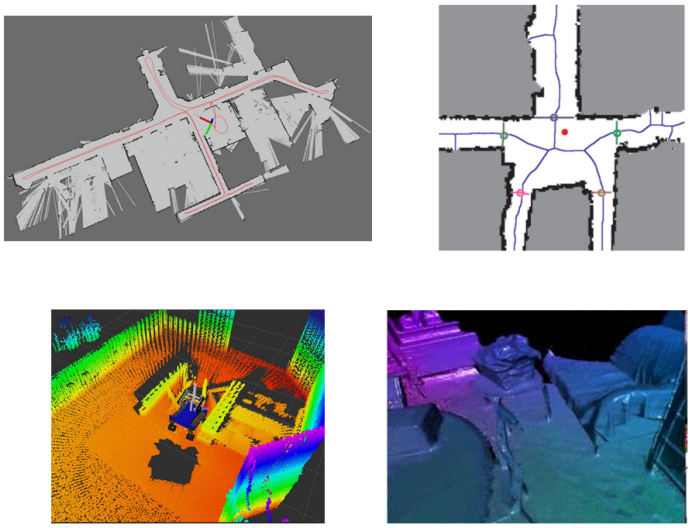


Figure 2-10: [12]

Metric Map

Sparse Dense Landmark

Topological Map

Graph A B A B

2.3 SLAM

SLAM SLAM
$t = 1, \cdots, K$ $\boldsymbol{x}$
“ ”

1. $k-1$ k $\boldsymbol{x}$
2. k $\boldsymbol{x}_k$ $\boldsymbol{y}_j$

“ ” “ 90 ” “ ” “ ”

$$\boldsymbol{x}_k = f\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{w}_k\right). \tag{2.1}$$

$\boldsymbol{u}_k$ † $\boldsymbol{w}_k$ f f /
“ ” 0.9 1.1
$\boldsymbol{x}_k$ $\boldsymbol{y}_j$ $\boldsymbol{z}_{k,j}$ h

$$\boldsymbol{z}_{k,j} = h\left(\boldsymbol{y}_j, \boldsymbol{x}_k, \boldsymbol{v}_{k,j}\right). \tag{2.2}$$

$\boldsymbol{v}_{k,j}$ $\boldsymbol{z}$ h
f, h $\boldsymbol{x}$,$\boldsymbol{y}$,$\boldsymbol{z}$ Parameterization
$[x_1, x_2, \theta]_k^{\mathrm{T}}$ x_1, x_2 θ $\boldsymbol{u}_k = [\Delta x_1, \Delta x_2, \Delta\theta]_k^{\mathrm{T}}$

$$\begin{bmatrix} x_1 \\ x_2 \\ \theta \end{bmatrix}_k = \begin{bmatrix} x_1 \\ x_2 \\ \theta \end{bmatrix}_{k-1} + \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta\theta \end{bmatrix}_k + \boldsymbol{w}_k. \tag{2.3}$$

“ ” “ ”
2D r ϕ $\boldsymbol{y}_j = [y_1, y_2]_j^{\mathrm{T}}$ $\boldsymbol{x}_k = [x_1, x_2]_k^{\mathrm{T}}$ $\boldsymbol{z}_{k,j} = [r_{k,j}, \phi_{k,j}]^{\mathrm{T}}$

$$\begin{bmatrix} r_{k,j} \\ \phi_{k,j} \end{bmatrix} = \begin{bmatrix} \sqrt{(y_{1,j} - x_{1,k})^2 + (y_{2,j} - x_{2,k})^2} \\ \arctan\left(\frac{y_{2,j} - x_{2,k}}{y_{1,j} - x_{1,k}}\right) \end{bmatrix} + \boldsymbol{v}. \tag{2.4}$$

SLAM “ ” 5
SLAM

$$\begin{cases} \boldsymbol{x}_k = f\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{w}_k\right), & k = 1, \cdots, K \\ \boldsymbol{z}_{k,j} = h\left(\boldsymbol{y}_j, \boldsymbol{x}_k, \boldsymbol{v}_{k,j}\right), & (k,j) \in \mathcal{O} \end{cases}. \tag{2.5}$$

---

* https://en.wikipedia.org/wiki/A*_search_algorithm
†
‡ “ ” “ ” “ ”

$\mathcal{O}$ —— SLAM $\boldsymbol{u}$ $\boldsymbol{z}$ $\boldsymbol{x}$ $\boldsymbol{y}$ S

/ / Linear Gaussian LG

Filter KF Non-Linear Non-Gaussian NLNG Extended Kalman

Filter EKF 21 EKF SLAM — 10 SLAM EKF[2]

Filter SLAM Graph Optimization [13] 10

SLAM $\boldsymbol{x}$

2.4

2.4.1 Linux

Linux C++ Linux Windows Linux Ubuntu Linux Ubuntu Ubuntu `http://cn.ubuntu.com` Ubuntu Ubuntu 14.04 **Ubuntu 18.04** Figure 2-11 Ubuntu Kylin Debian Deepin Linux Mint Ubuntu 18.04 **18.04** Ubuntu apt-get Ubuntu Linux Windows OS X

PC Ubuntu 18.04 Ubuntu Figure 2-11 4GB CPU

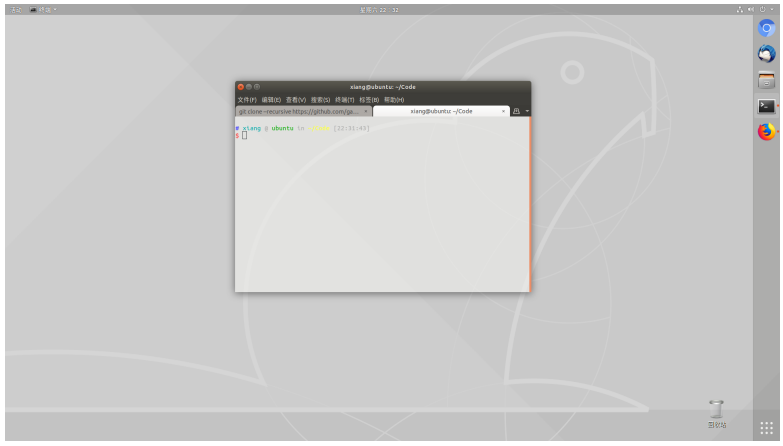


Figure 2-11: Ubuntu 18.04

- “ ” SSD 15
- 10MB/s †

Ubuntu Ubuntu ‡ Ubuntu Linux SLAM /home “slambook2” “ ” Git

2.4.2 Hello SLAM

HelloSLAM Linux Windows .exe cd ls /bin

---

* Pose Rotation Translation

† TUNA

‡ Ubuntu

Listing 2.1: slambook2/ch2/helloSLAM.cpp

```
#include <iostream>
using namespace std;

int main(int argc, char **argv) {
  cout << "Hello SLAM!" << endl;
  return 0;
}
```

—— C++ gedit Vim
g++ g++ C++

Listing 2.2:

```
g++ helloSLAM.cpp
```

“command not found” g++

Listing 2.3:

```
sudo apt-get install g++
```

helloSLAM.cpp a.out ./a.out *

Listing 2.4:

```
% ./a.out
Hello SLAM!
```

“Hello SLAM!”
helloSLAM.cpp g++ g++ a.out

2.4.3 cmake

C++ g++ C++ g++
cmake cmake makefile make makefile makefile helloS

Listing 2.5: slambook2/ch2/CMakeLists.txt

```
# □□□□□ cmake □□□□
cmake_minimum_required( VERSION 2.8 )

# □□□□ cmake □□
project( HelloSLAM )

# □□□□□□□□□□
# □□□add_executable( □□□□□□□□□□
add_executable( helloSLAM helloSLAM.cpp )
```

CMakeLists.txt cmake CMakeLists.txt cmake
slambook2/ch2/ cmake cmake †

Listing 2.6:

```
cmake .
```

cmake MakeFile‡ MakeFile make

* %
† cmake
‡ MakeFile

Listing 2.7:

```
% make
Scanning dependencies of target helloSLAM
[100%] Building CXX object CMakeFiles/helloSLAM.dir/helloSLAM.cpp.o
Linking CXX executable helloSLAM
[100%] Built target helloSLAM
```

CMakeLists.txt helloSLAM

Listing 2.8:

```
% ./helloSLAM
Hello SLAM!
```

g++ cmake make cmake make g++
executable” cmake g++
cmake cm

Listing 2.9:

```
mkdir build
cd build
cmake ..
make
```

“build” build cmake .. cmake build build

2.4.4

C++ main Library
OpenCV Eigen cmake libHe

Listing 2.10: slambook2/ch2/libHelloSLAM.cpp

```
//□□□□□□□
#include <iostream>
using namespace std;

void printHello() {
  cout << "Hello SLAM" << endl;
}
```

printHello main CMakeLists.txt

Listing 2.11: slambook2/ch2/CMakeLists.txt

```
add_library( hello libHelloSLAM.cpp )
```

cmake “hello” cmake

Listing 2.12:

```
cd build
cmake ..
make
```

build libhello.a
Linux * .a .so

Listing 2.13: slambook2/ch2/CMakeLists.txt

```
add_library( hello_shared SHARED libHelloSLAM.cpp )
```

*

libhello_shared.so

.a .so li

Listing 2.14: slambook2/ch2/libHelloSLAM.h

```
#ifndef LIBHELLOSLAM_H_
#define LIBHELLOSLAM_H_
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

// □□□□□□□hello
void printHello();

#endif
```

printHello

Listing 2.15: slambook2/ch2/useHello.cpp

```
#include "libHelloSLAM.h"

// □□ libHelloSLAM.h □□ printHello() □□
int main(int argc, char **argv) {
  printHello();
  return 0;
}
```

CMakeLists.txt

Listing 2.16: slambook2/ch2/CMakeLists.txt

```
add_executable( useHello useHello.cpp )
target_link_libraries( useHello hello_shared )
```

useHello hello_shared

cmake cmake cmake cmake

1.

2. main

3.

CMakeLists.txt cmake

2.4.5 IDE

Integrated Development Environment IDE ID

Linux IDE Windows Visual Studio C++ Eclipse Qt Creator Code::Blocks Clion Studio Code IDE Kdevelop Clion Figure 2-12 Figure 2-15 * Kdevelop Ub get Clion C++

1. cmake

2. C++ 11

3.

4.

IDE Kdevelop clion

* Visual Studio Code

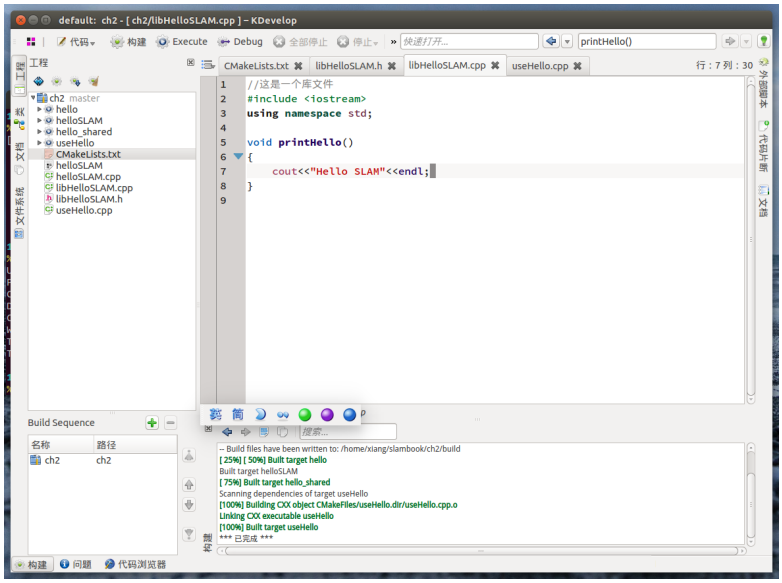


Figure 2-12: Kdevelop

Kdevelop

Kdevelop cmake CMakeLists.txt Kdevelop “ → / ” CMakeLists.txt build
12

IDE Windows Visual C++ Visual Studio Kdevelop
IDE Windows Visual Studio Linux gdb IDE g

1. CMakeLists.txt Debug

2. Kdevelop

3.

CMakeLists.txt

Listing 2.17: slambook2/ch2/CMakeLists.txt

```
1 set( CMAKE_BUILD_TYPE "Debug" )
```

cmake Debug Release Debug Release
“ → ” “Add New→ ” Kdevelop Figure 2-13 cmake add_
executable

main “OK”
“Execute” “Debug” “Execute” printHel
14

Kdevelop F10 F11 F12 “ ” Kdevelo
*

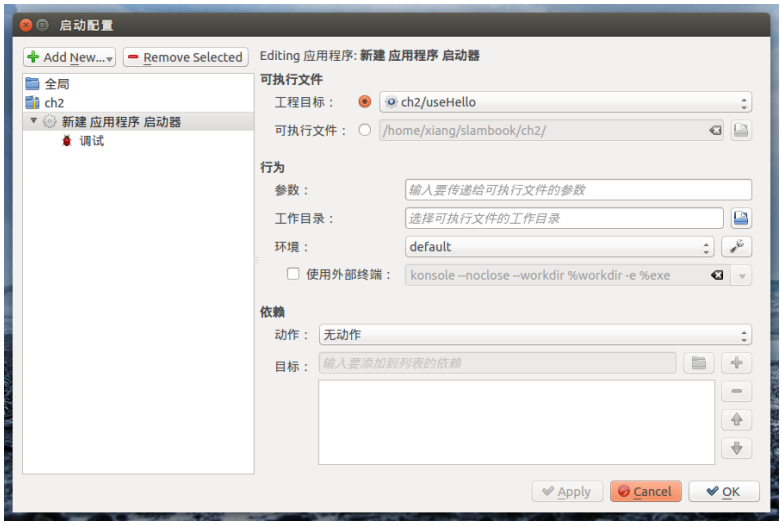


Figure 2-13:

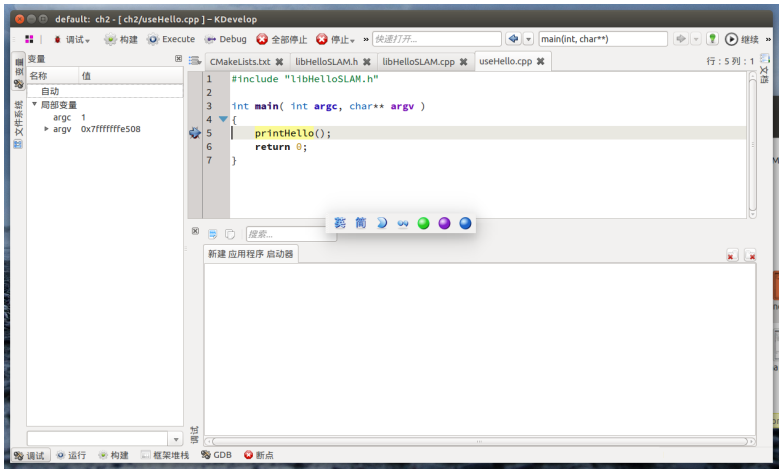


Figure 2-14:

Clion

Clion Kdevelop * Clion CMakeLists.txt Clion cmake-make Figure 2-15

Clion Clion IDE build cmake make cm

*

* CLion 2018 2017

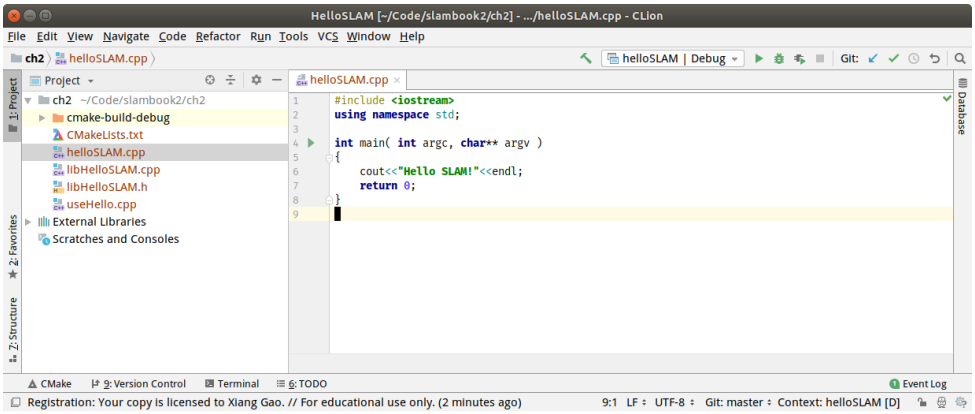


Figure 2-15: Clion

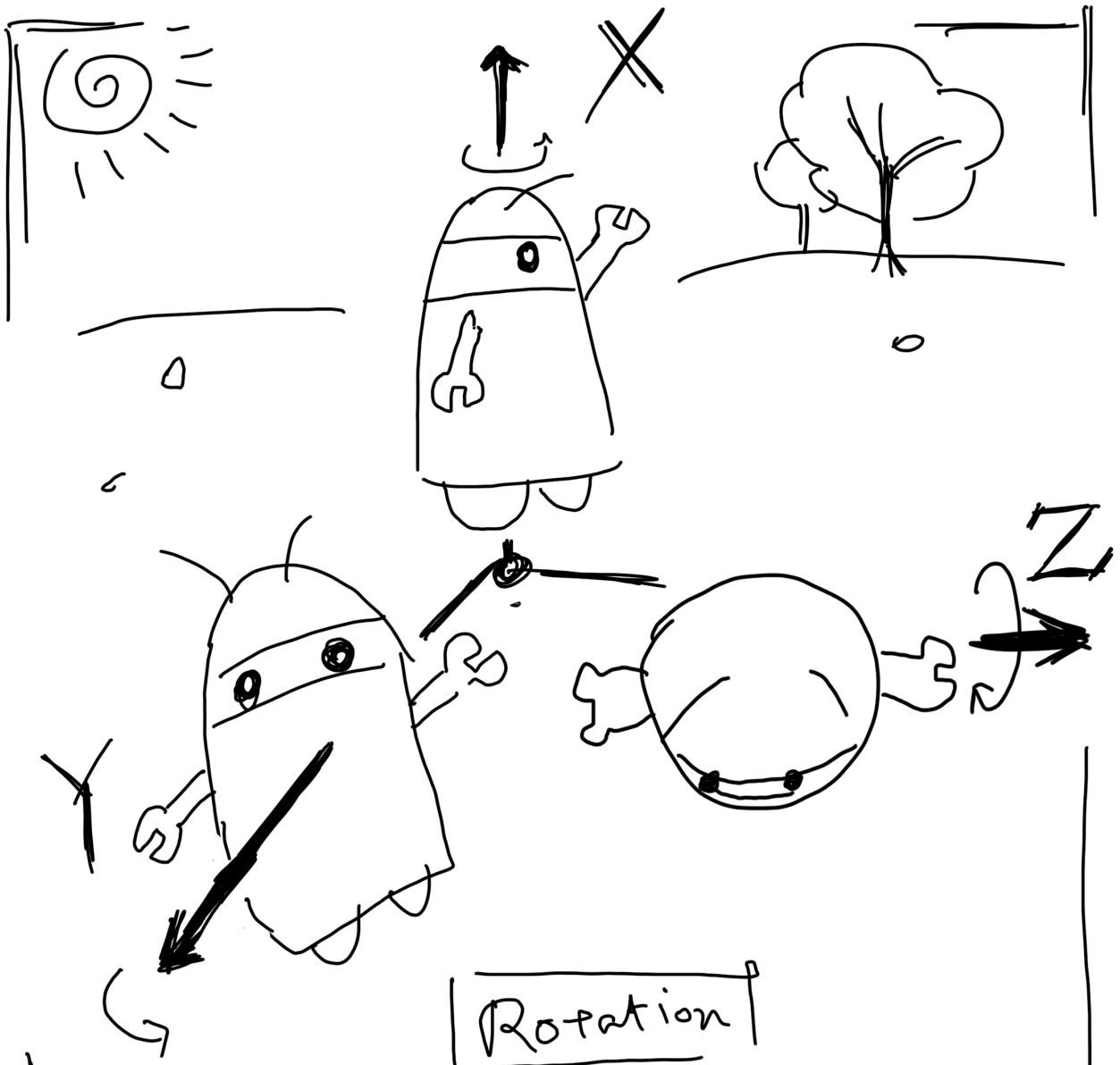
1. [1] [14]
2.* SLAM [9, 15–18] SLAM
3. g++
4. build cmake Kdevelop
5. g++
6.
7.* cmake cmake
8.* hello SLAM find_package
9.* cmake `https://github.com/TheErk/CMake-tutorial`
10. Kdevelop
11. Vim Kdevelop Vim

Chapter 3

1.

2. Eigen

SLAM SLAM



Rotation

$$SO(3) = \{ R \mid R^T R = I, \det(R) = 1 \}$$

roll, pitch, yaw

$$q = q_0 + q_1 i + q_2 j + q_3 k$$

$$T = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \in SE(3)$$

3.1

3.1.1

3 3 “ (

$\boldsymbol{a}$ $\boldsymbol{a}$

$\mathbb{R}^3$ $^*(\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3)$ $\boldsymbol{a}$

$$\boldsymbol{a} = [\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3] \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = a_1\boldsymbol{e}_1 + a_2\boldsymbol{e}_2 + a_3\boldsymbol{e}_3. \tag{3.1}$$

$(a_1, a_2, a_3)^{\mathrm{T}}$ $\boldsymbol{a}$ † 3 $\boldsymbol{x}$ $\boldsymbol{y}$ $\boldsymbol{z}$ $\boldsymbol{x}\times$
$\boldsymbol{y}$ 3 3D OpenGL 3D Max Unity Direct3D

$\boldsymbol{a}, \boldsymbol{b}$

$\mathbb{R}^3$ ‡

$$\boldsymbol{a} \cdot \boldsymbol{b} = \boldsymbol{a}^{\mathrm{T}}\boldsymbol{b} = \sum_{i=1}^{3} a_i b_i = |\boldsymbol{a}|\,|\boldsymbol{b}| \cos\langle \boldsymbol{a}, \boldsymbol{b} \rangle . \tag{3.2}$$

$\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ $\boldsymbol{a}, \boldsymbol{b}$

$$\boldsymbol{a} \times \boldsymbol{b} = \begin{Vmatrix} \boldsymbol{e}_1 & \boldsymbol{e}_2 & \boldsymbol{e}_3 \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{Vmatrix} = \begin{bmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{bmatrix} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \boldsymbol{b} \triangleq \boldsymbol{a}^{\wedge}\boldsymbol{b}. \tag{3.3}$$

$|\boldsymbol{a}|\,|\boldsymbol{b}| \sin\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ $\wedge$ $\boldsymbol{a}$ Skew-symmetric
matrix § $\wedge$ $\boldsymbol{a} \times \boldsymbol{b}$ $\boldsymbol{a}^{\wedge}\boldsymbol{b}$

$$\boldsymbol{a}^{\wedge} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}. \tag{3.4}$$

3.1.2

3D Figure 3-
1 x_W, y_W, z_W x_C, y_C, z_C $\boldsymbol{p}$ $\boldsymbol{p}_c$ $\boldsymbol{p}_w$
¶

Transform

$(\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3)$ $(\boldsymbol{e}'_1, \boldsymbol{e}'_2, \boldsymbol{e}'_3)$ $\boldsymbol{a}$ $[a_1, a_2, a_3]^{\mathrm{T}}$ $[a'_1, a'_2, a'_3$

$$[\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3] \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = [\boldsymbol{e}'_1, \boldsymbol{e}'_2, \boldsymbol{e}'_3] \begin{bmatrix} a'_1 \\ a'_2 \\ a'_3 \end{bmatrix}. \tag{3.5}$$

---

*
†
‡
§ $\boldsymbol{A}$ $\boldsymbol{A}^{\mathrm{T}} = -\boldsymbol{A}$
¶

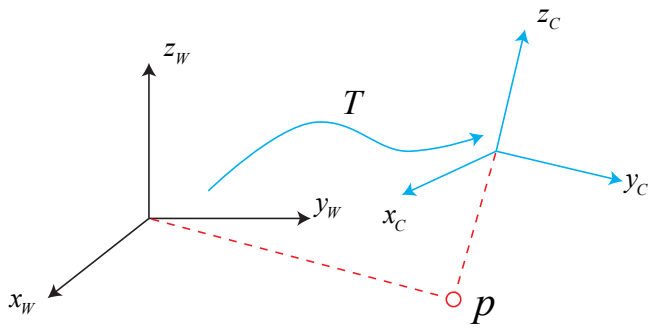


Figure 3-1: $\boldsymbol{p}$ $\boldsymbol{p}_w$ $\boldsymbol{p}_c$ $\boldsymbol{T}$

$$\begin{bmatrix} \boldsymbol{e}_1^{\mathrm{T}} \\ \boldsymbol{e}_2^{\mathrm{T}} \\ \boldsymbol{e}_3^{\mathrm{T}} \end{bmatrix}$$

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{e}_1^{\mathrm{T}}\boldsymbol{e}_1' & \boldsymbol{e}_1^{\mathrm{T}}\boldsymbol{e}_2' & \boldsymbol{e}_1^{\mathrm{T}}\boldsymbol{e}_3' \\ \boldsymbol{e}_2^{\mathrm{T}}\boldsymbol{e}_1' & \boldsymbol{e}_2^{\mathrm{T}}\boldsymbol{e}_2' & \boldsymbol{e}_2^{\mathrm{T}}\boldsymbol{e}_3' \\ \boldsymbol{e}_3^{\mathrm{T}}\boldsymbol{e}_1' & \boldsymbol{e}_3^{\mathrm{T}}\boldsymbol{e}_2' & \boldsymbol{e}_3^{\mathrm{T}}\boldsymbol{e}_3' \end{bmatrix} \begin{bmatrix} a_1' \\ a_2' \\ a_3' \end{bmatrix} \triangleq \boldsymbol{R}\boldsymbol{a}'. \tag{3.6}$$

$\boldsymbol{R}$ $\boldsymbol{R}$ Rotation matrix Cosine matrix

1 *,† 1 n

$$\mathrm{SO}(n) = \{\boldsymbol{R} \in \mathbb{R}^{n\times n} | \boldsymbol{R}\boldsymbol{R}^{\mathrm{T}} = \boldsymbol{I}, \det(\boldsymbol{R}) = 1\}. \tag{3.7}$$

$\mathrm{SO}(n)$ Special Orthogonal Group “ ” n $\mathrm{SO}(3)$

$$\boldsymbol{a}' = \boldsymbol{R}^{-1}\boldsymbol{a} = \boldsymbol{R}^{\mathrm{T}}\boldsymbol{a}. \tag{3.8}$$

$\boldsymbol{R}^{\mathrm{T}}$

$\boldsymbol{a}$ $\boldsymbol{R}$ $\boldsymbol{t}$ $\boldsymbol{a}'$

$$\boldsymbol{a}' = \boldsymbol{R}\boldsymbol{a} + \boldsymbol{t}. \tag{3.9}$$

$\boldsymbol{t}$ $\boldsymbol{R}$ $\boldsymbol{t}$ 1 2 $\boldsymbol{a}$ $\boldsymbol{a}_1, \boldsymbol{a}_2$

$$\boldsymbol{a}_1 = \boldsymbol{R}_{12}\boldsymbol{a}_2 + \boldsymbol{t}_{12}. \tag{3.10}$$

$\boldsymbol{R}_{12}$ “ 2 1” “1 2 ” $\boldsymbol{R}_{21}$ $\boldsymbol{t}_{12}$ 1 2 $\mathbf{1}$ “1 2 ” $\boldsymbol{t}_{21}$ 2 1 $\mathbf{2}$ $-\boldsymbol{t}_{12}$ ‡ “ ”

3.1.3

(3.9) $\boldsymbol{R}_1, \boldsymbol{t}_1$ $\boldsymbol{R}_2, \boldsymbol{t}_2$

---

*

† 1 ± 1 -1

‡

$$\boldsymbol{b} = \boldsymbol{R}_1\boldsymbol{a} + \boldsymbol{t}_1, \quad \boldsymbol{c} = \boldsymbol{R}_2\boldsymbol{b} + \boldsymbol{t}_2.$$

$\boldsymbol{a}$ $\boldsymbol{c}$

$$\boldsymbol{c} = \boldsymbol{R}_2\left(\boldsymbol{R}_1\boldsymbol{a} + \boldsymbol{t}_1\right) + \boldsymbol{t}_2.$$

(3.9)

$$\begin{bmatrix} \boldsymbol{a}' \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{a} \\ 1 \end{bmatrix} \triangleq \boldsymbol{T} \begin{bmatrix} \boldsymbol{a} \\ 1 \end{bmatrix}. \tag{3.11}$$

1 $\boldsymbol{T}$ **Transform Matrix**

$\tilde{\boldsymbol{a}}$ $\boldsymbol{a}$

$$\tilde{\boldsymbol{b}} = \boldsymbol{T}_1\tilde{\boldsymbol{a}},\ \tilde{\boldsymbol{c}} = \boldsymbol{T}_2\tilde{\boldsymbol{b}} \quad \Rightarrow \tilde{\boldsymbol{c}} = \boldsymbol{T}_2\boldsymbol{T}_1\tilde{\boldsymbol{a}}. \tag{3.12}$$

1 1 * $\boldsymbol{b} = \boldsymbol{T}\boldsymbol{a}$ †

$\boldsymbol{T}$ $\boldsymbol{0}$ 1 Special Euclidean Group

$$\mathrm{SE}(3) = \left\{ \boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} | \boldsymbol{R} \in \mathrm{SO}(3), \boldsymbol{t} \in \mathbb{R}^3 \right\}. \tag{3.13}$$

SO(3)

$$\boldsymbol{T}^{-1} = \begin{bmatrix} \boldsymbol{R}^{\mathrm{T}} & -\boldsymbol{R}^{\mathrm{T}}\boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix}. \tag{3.14}$$

$\boldsymbol{T}_{12}$ 2 1 $\boldsymbol{T}\boldsymbol{a}$ $\boldsymbol{R}\boldsymbol{a}$

— C++

SO(3) $\mathbb{R}^3$ SE(3)

3.2 Eigen

Eigen slambook2/ch3/useEigen

Eigen‡ C++ Eigen g2o Sophus Eigen

PC Eigen

Listing 3.1:

```
sudo apt-get install libeigen3-dev
```

Ubuntu Ubuntu apt Eigen Eigen "/usr

Listing 3.2:

```
sudo updatedb
locate eigen3
```

Eigen .so .a Eigen Eigen

---

* 7

†

‡ http://eigen.tuxfamily.org/index.php?title=Main_Page

Listing 3.3: slambook2/ch3/useEigen/eigenMatrix.cpp

```
#include <iostream>
using namespace std;

#include <ctime>
// Eigen □□□□
#include <Eigen/Core>
// □□□□□□□□□□□□□□□□□□□□□□
#include <Eigen/Dense>
using namespace Eigen;

#define MATRIX_SIZE 50

/****************************
* □□□□□□□ Eigen □□□□□□□□
****************************/

int main(int argc, char **argv) {
  // Eigen □□□□□□□□□□□Eigen::□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Matrix
  // □□□□□2*3□□float
  Matrix<float, 2, 3> matrix_23;

  // □□□Eigen □□ typedef □□□□□□□□□□□□□□□□□□Eigen::Matrix
  // □□ Vector3d □□□□ Eigen::Matrix<double, 3, □□□□□□1>
  Vector3d v_3d;
  // □□□□□
  Matrix<float, 3, 1> vd_3d;

  // Matrix3d □□□□ Eigen::Matrix<double, 3, 3>
  Matrix3d matrix_33 = Matrix3d::Zero(); //□□□□□
  // □□□□□□□□□□□□□□□□□□□□□□□□□□
  Matrix<double, Dynamic, Dynamic> matrix_dynamic;
  // □□□□
  MatrixXd matrix_x;
  // □□□□□□□□□□□□□□□□□□□

  // □□□□□□□□□Eigen
  // □□□□□□□□□□
  matrix_23 << 1, 2, 3, 4, 5, 6;
  // □□
  cout << "matrix 2x3 from 1 to 6: \n" << matrix_23 << endl;

  // □()□□□□□□□□
  cout << "print matrix 2x3: " << endl;
  for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) cout << matrix_23(i, j) << "\t";
    cout << endl;
  }

  // □□□□□□□□□□□□□□□□□□□□□□□□
  v_3d << 3, 2, 1;
  vd_3d << 4, 5, 6;

  // □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Eigen
  // Matrix<double, 2, 1> result_wrong_type = matrix_23 * v_3d;
  // □□□□□□□
  Matrix<double, 2, 1> result = matrix_23.cast<double>() * v_3d;
  cout << "[1,2,3;4,5,6]*[3,2,1]=" << result.transpose() << endl;

  Matrix<float, 2, 1> result2 = matrix_23 * vd_3d;
  cout << "[1,2,3;4,5,6]*[4,5,6]: " << result2.transpose() << endl;

  // □□□□□□□□□□□□□□
  // □□□□□□□□□□□□□□□□□□□□□□Eigen
  // Eigen::Matrix<double, 2, 3> result_wrong_dimension = matrix_23.cast<double>() *
       v_3d;

  // □□□□□□□
  // □□□□□□□□□□□□□□□□□□+−*/
  matrix_33 = Matrix3d::Random();      // □□□□□□
```

```
69   cout << "random matrix: \n" << matrix_33 << endl;
70   cout << "transpose: \n" << matrix_33.transpose() << endl; // □□
71   cout << "sum: " << matrix_33.sum() << endl;               // □□□□
72   cout << "trace: " << matrix_33.trace() << endl;           // □
73   cout << "times 10: \n" << 10 * matrix_33 << endl;         // □□
74   cout << "inverse: \n" << matrix_33.inverse() << endl;     // □
75   cout << "det: " << matrix_33.determinant() << endl;       // □□□
76
77   // □□□
78   // □□□□□□□□□□□□□□□□
79   SelfAdjointEigenSolver<Matrix3d> eigen_solver(matrix_33.transpose() * matrix_33);
80   cout << "Eigen values = \n" << eigen_solver.eigenvalues() << endl;
81   cout << "Eigen vectors = \n" << eigen_solver.eigenvectors() << endl;
82
83   // □□□
84   // □□□□ matrix_NN * x = v_Nd □□□□
85   // □□□□□□□□□□□□□□□□□□□□□□□□N
86   // □□□□□□□□□□□□□□□□□□□□□□□□
87
88   Matrix<double, MATRIX_SIZE, MATRIX_SIZE> matrix_NN
89     = MatrixXd::Random(MATRIX_SIZE, MATRIX_SIZE);
90   matrix_NN = matrix_NN * matrix_NN.transpose();  // □□□□□
91   Matrix<double, MATRIX_SIZE, 1> v_Nd = MatrixXd::Random(MATRIX_SIZE, 1);
92
93   clock_t time_stt = clock(); // □□
94   // □□□□
95   Matrix<double, MATRIX_SIZE, 1> x = matrix_NN.inverse() * v_Nd;
96   cout << "time of normal inverse is "
97        << 1000 * (clock() - time_stt) / (double) CLOCKS_PER_SEC << "ms" << endl;
98   cout << "x = " << x.transpose() << endl;
99
100  // □□□□□□□□□□□□□□□□□□□□□□□□□□□□QR
101  time_stt = clock();
102  x = matrix_NN.colPivHouseholderQr().solve(v_Nd);
103  cout << "time of Qr decomposition is "
104       << 1000 * (clock() - time_stt) / (double) CLOCKS_PER_SEC << "ms" << endl;
105  cout << "x = " << x.transpose() << endl;
106
107  // □□□□□□□□□□□□□□□□□□□□□□□□cholesky
108  time_stt = clock();
109  x = matrix_NN.ldlt().solve(v_Nd);
110  cout << "time of ldlt decomposition is "
111       << 1000 * (clock() - time_stt) / (double) CLOCKS_PER_SEC << "ms" << endl;
112  cout << "x = " << x.transpose() << endl;
113
114  return 0;
115 }
```

Eigen CMakeLists.txt Eigen

Listing 3.4: slambook2/ch3/useEigen/CMakeLists.txt

```
1 # □□□□□
2 include_directories( "/usr/include/eigen3" )
```

Eigen target_link_libraries Eigen package

Listing 3.5:

```
1  % build/eigenMatrix
2  matrix 2x3 from 1 to 6:
3  1 2 3
4  4 5 6
5  print matrix 2x3:
6  1 2 3
7  4 5 6
8  [1,2,3;4,5,6]*[3,2,1]=10 28
9  [1,2,3;4,5,6]*[4,5,6]: 32 77
10 random matrix:
```

```
0.680375   0.59688 −0.329554
−0.211234  0.823295  0.536459
0.566198 −0.604897 −0.444451
transpose:
0.680375 −0.211234  0.566198
0.59688  0.823295 −0.604897
−0.329554  0.536459 −0.444451
sum: 1.61307
trace: 1.05922
times 10:
6.80375   5.9688 −3.29554
−2.11234  8.23295  5.36459
5.66198 −6.04897 −4.44451
inverse:
−0.198521   2.22739    2.8357
1.00605 −0.555135  −1.41603
−1.62213   3.59308   3.28973
det: 0.208598……
```

1.

2. Kdevelop C++ Clion

3. Eigen MATLAB Eigen

4. Eigen float double Eigen

5. Eigen C++ C++ float double Eigen MIXED DIFFERENT NUMERIC TYPES ..."

6. "YOU MIXED MATRICES OF DIFFERENT SIZES" C++

7. Eigen `http://eigen.tuxfamily.org/dox-devel/modules.html` Eigen Eigen QR

3.3

3.3.1

6

1. SO(3) 9 3 16 6

2. 1 /

/ Axis-Angle

$\boldsymbol{R}$ $\boldsymbol{n}$ θ $\theta\boldsymbol{n}$ Rodrigues's Formula *

$$\boldsymbol{R} = \cos\theta\boldsymbol{I} + (1-\cos\theta)\,\boldsymbol{n}\boldsymbol{n}^{\mathrm{T}} + \sin\theta\boldsymbol{n}^{\wedge}. \tag{3.15}$$

* `https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula`

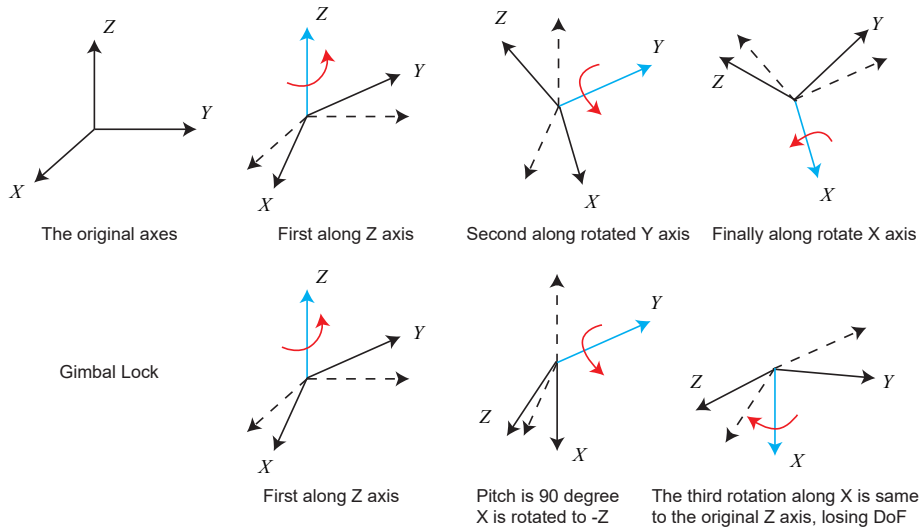


Figure 3-2: ZYX pitch=90°

^ (3.3) θ *

$$\begin{aligned}\operatorname{tr}(\boldsymbol{R}) &= \cos\theta \operatorname{tr}(\boldsymbol{I}) + (1-\cos\theta)\operatorname{tr}\left(\boldsymbol{n}\boldsymbol{n}^{\mathrm{T}}\right) + \sin\theta \operatorname{tr}(\boldsymbol{n}^{\wedge}) \\ &= 3\cos\theta + (1-\cos\theta) \\ &= 1 + 2\cos\theta.\end{aligned} \tag{3.16}$$

$$\theta = \arccos\left(\frac{\operatorname{tr}(\boldsymbol{R})-1}{2}\right). \tag{3.17}$$

$\boldsymbol{n}$

$$\boldsymbol{R}\boldsymbol{n} = \boldsymbol{n}. \tag{3.18}$$

$\boldsymbol{n}$ $\boldsymbol{R}$ 1 “ ” SO(3)

3.3.2

3 3 X Y Z XYZ ZYZ ZY

“ ”“ ” “ – – ” yaw-pitch-roll 3 ZYX ZYX X Y Z Figure 3-2 ZYX 3

1. Z yaw
2. Y pitch
3. X roll

* trace

$[r, p, y]^{\mathrm{T}}$ 3 rpy rpy
—
Gimbal Lock* $\pm 90^\circ$ 3 2 3

3.4

3.4.1

9 3 [19] $\pm 90^\circ$
$\mathbb{C}$ i 90° Quaternion Hamilton
θ $\mathrm{e}^{i\theta}$

$$\mathrm{e}^{i\theta} = \cos\theta + i\sin\theta. \tag{3.19}$$

$\boldsymbol{q}$

$$\boldsymbol{q} = q_0 + q_1 i + q_2 j + q_3 k, \tag{3.20}$$

i, j, k

$$\begin{cases} i^2 = j^2 = k^2 = -1 \\ ij = k, ji = -k \\ jk = i, kj = -i \\ ki = j, ik = -j \end{cases}. \tag{3.21}$$

i, j, k

$$\boldsymbol{q} = [s, \boldsymbol{v}]^{\mathrm{T}}, \quad s = q_0 \in \mathbb{R}, \quad \boldsymbol{v} = [q_1, q_2, q_3]^{\mathrm{T}} \in \mathbb{R}^3,$$

s $\boldsymbol{v}$ $\mathbf{0}$ 0
i 90° i i 90° $ij = k$ i 90° j 90° k 90°
— i 180° $ij = k$ $i^2 = -1$ i 360°

3.4.2

$\boldsymbol{q}_a, \boldsymbol{q}_b$ $[s_a, \boldsymbol{v}_a]^{\mathrm{T}}, [s_b, \boldsymbol{v}_b]^{\mathrm{T}}$

$$\boldsymbol{q}_a = s_a + x_a i + y_a j + z_a k, \quad \boldsymbol{q}_b = s_b + x_b i + y_b j + z_b k.$$

1.

$\boldsymbol{q}_a, \boldsymbol{q}_b$

$$\boldsymbol{q}_a \pm \boldsymbol{q}_b = [s_a \pm s_b, \boldsymbol{v}_a \pm \boldsymbol{v}_b]^{\mathrm{T}}. \tag{3.22}$$

---

* https://en.wikipedia.org/wiki/Gimbal_lock
† θ 2

2.

$\boldsymbol{q}_a$ $\boldsymbol{q}_b$ (3.21)

$$\begin{aligned}\boldsymbol{q}_a\boldsymbol{q}_b = & s_a s_b - x_a x_b - y_a y_b - z_a z_b \\ & + (s_a x_b + x_a s_b + y_a z_b - z_a y_b)\, i \\ & + (s_a y_b - x_a z_b + y_a s_b + z_a x_b)\, j \\ & + (s_a z_b + x_a y_b - y_a x_b + z_a s_b)\, k.\end{aligned} \tag{3.23}$$

$$\boldsymbol{q}_a\boldsymbol{q}_b = \left[s_a s_b - \boldsymbol{v}_a^{\mathrm{T}}\boldsymbol{v}_b, s_a\boldsymbol{v}_b + s_b\boldsymbol{v}_a + \boldsymbol{v}_a \times \boldsymbol{v}_b\right]^{\mathrm{T}}. \tag{3.24}$$

$\boldsymbol{v}_a$ $\boldsymbol{v}_b$ $\mathbb{R}^3$

3.

$$\|\boldsymbol{q}_a\| = \sqrt{s_a^2 + x_a^2 + y_a^2 + z_a^2}. \tag{3.25}$$

$$\|\boldsymbol{q}_a\boldsymbol{q}_b\| = \|\boldsymbol{q}_a\|\|\boldsymbol{q}_b\|. \tag{3.26}$$

4.

$$\boldsymbol{q}_a^* = s_a - x_a i - y_a j - z_a k = [s_a, -\boldsymbol{v}_a]^{\mathrm{T}}. \tag{3.27}$$

$$\boldsymbol{q}^*\boldsymbol{q} = \boldsymbol{q}\boldsymbol{q}^* = [s_a^2 + \boldsymbol{v}^{\mathrm{T}}\boldsymbol{v}, \boldsymbol{0}]^{\mathrm{T}}. \tag{3.28}$$

5.

$$\boldsymbol{q}^{-1} = \boldsymbol{q}^*/\|\boldsymbol{q}\|^2. \tag{3.29}$$

$\mathbf{1}$

$$\boldsymbol{q}\boldsymbol{q}^{-1} = \boldsymbol{q}^{-1}\boldsymbol{q} = \mathbf{1}. \tag{3.30}$$

$\boldsymbol{q}$

$$(\boldsymbol{q}_a\boldsymbol{q}_b)^{-1} = \boldsymbol{q}_b^{-1}\boldsymbol{q}_a^{-1}. \tag{3.31}$$

6.

$$k\boldsymbol{q} = [ks, k\boldsymbol{v}]^{\mathrm{T}}. \tag{3.32}$$

3.4.3

$\boldsymbol{p} = [x, y, z] \in \mathbb{R}^3$ $\boldsymbol{q}$ $\boldsymbol{p}$ $\boldsymbol{p}'$ $\boldsymbol{p}' = \boldsymbol{R}\boldsymbol{p}$

$$\boldsymbol{p} = [0, x, y, z]^{\mathrm{T}} = [0, \boldsymbol{v}]^{\mathrm{T}}.$$

3 3 $\boldsymbol{p}'$

$$\boldsymbol{p}' = \boldsymbol{q}\boldsymbol{p}\boldsymbol{q}^{-1}. \tag{3.33}$$

$\boldsymbol{p}'$ 0

3.4.4

$\boldsymbol{q} = [s, \boldsymbol{v}]^{\mathrm{T}}$ $+ \oplus$ [20]

$$\boldsymbol{q}^{+} = \begin{bmatrix} s & -\boldsymbol{v}^{\mathrm{T}} \\ \boldsymbol{v} & s\boldsymbol{I} + \boldsymbol{v}^{\wedge} \end{bmatrix}, \quad \boldsymbol{q}^{\oplus} = \begin{bmatrix} s & -\boldsymbol{v}^{\mathrm{T}} \\ \boldsymbol{v} & s\boldsymbol{I} - \boldsymbol{v}^{\wedge} \end{bmatrix}, \tag{3.34}$$

4×4

$$\boldsymbol{q}_1^{+}\boldsymbol{q}_2 = \begin{bmatrix} s_1 & -\boldsymbol{v}_1^{T} \\ \boldsymbol{v}_1 & s_1\boldsymbol{I} + \boldsymbol{v}_1^{\wedge} \end{bmatrix} \begin{bmatrix} s_2 \\ \boldsymbol{v}_2 \end{bmatrix} = \begin{bmatrix} -\boldsymbol{v}_1^{T}\boldsymbol{v}_2 + s_1 s_2 \\ s_1\boldsymbol{v}_2 + s_2\boldsymbol{v}_1 + \boldsymbol{v}_1^{\wedge}\boldsymbol{v}_2 \end{bmatrix} = \boldsymbol{q}_1\boldsymbol{q}_2 \tag{3.35}$$

$$\boldsymbol{q}_1\boldsymbol{q}_2 = \boldsymbol{q}_1^{+}\boldsymbol{q}_2 = \boldsymbol{q}_2^{\oplus}\boldsymbol{q}_1. \tag{3.36}$$

$$\begin{aligned} \boldsymbol{p}' &= \boldsymbol{q}\boldsymbol{p}\boldsymbol{q}^{-1} = \boldsymbol{q}^{+}\boldsymbol{p}^{+}\boldsymbol{q}^{-1} \\ &= \boldsymbol{q}^{+}\boldsymbol{q}^{-1^{\oplus}}\boldsymbol{p}. \end{aligned} \tag{3.37}$$

$$\boldsymbol{q}^{+}\left(\boldsymbol{q}^{-1}\right)^{\oplus} = \begin{bmatrix} s & -\boldsymbol{v}^{T} \\ \boldsymbol{v} & s\boldsymbol{I} + \boldsymbol{v}^{\wedge} \end{bmatrix} \begin{bmatrix} s & \boldsymbol{v}^{T} \\ -\boldsymbol{v} & s\boldsymbol{I} + \boldsymbol{v}^{\wedge} \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{0} \\ \boldsymbol{0}^{\mathrm{T}} & \boldsymbol{v}\boldsymbol{v}^{\mathrm{T}} + s^2\boldsymbol{I} + 2s\boldsymbol{v}^{\wedge} + \left(\boldsymbol{v}^{\wedge}\right)^2 \end{bmatrix}. \tag{3.38}$$

$\boldsymbol{p}'\ \boldsymbol{p}$

$$\boldsymbol{R} = \boldsymbol{v}\boldsymbol{v}^{\mathrm{T}} + s^2\boldsymbol{I} + 2s\boldsymbol{v}^{\wedge} + \left(\boldsymbol{v}^{\wedge}\right)^2. \tag{3.39}$$

$$\begin{aligned} \mathrm{tr}(\boldsymbol{R}) &= \mathrm{tr}(\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}} + 3s^2 + 2s\cdot 0 + \mathrm{tr}((\boldsymbol{v}^{\wedge})^2) \\ &= v_1^2 + v_2^2 + v_3^2 + 3s^2 - 2(v_1^2 + v_2^2 + v_3^2) \\ &= (1 - s^2) + 3s^2 - 2(1 - s^2) \\ &= 4s^2 - 1. \end{aligned} \tag{3.40}$$

(3.17)

$$\begin{aligned} \theta &= \arccos(\frac{\mathrm{tr}(\boldsymbol{R} - 1)}{2}) \\ &= \arccos(2s^2 - 1). \end{aligned} \tag{3.41}$$

$$\cos\theta = 2s^2 - 1 = 2\cos^2\frac{\theta}{2} - 1, \tag{3.42}$$

$$\theta = 2\arccos s. \tag{3.43}$$

(3.38) $\boldsymbol{q}$ $\boldsymbol{p}$ $\boldsymbol{q}$

$$\begin{cases} \theta = 2\arccos q_0 \\ [n_x, n_y, n_z]^{\mathrm{T}} = [q_1, q_2, q_3]^{\mathrm{T}} / \sin\frac{\theta}{2} \end{cases}. \tag{3.44}$$

3.5 *

3D

1.

$$\boldsymbol{T}_S = \begin{bmatrix} s\boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix}. \tag{3.45}$$

s x, y, z 1 10 Sim(

2.

$$\boldsymbol{T}_A = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix}. \tag{3.46}$$

$\boldsymbol{A}$

3.

$$\boldsymbol{T}_P = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ \boldsymbol{a}^{\mathrm{T}} & v \end{bmatrix}. \tag{3.47}$$

$\boldsymbol{A}$ $\boldsymbol{t}$ $\boldsymbol{a}^{\mathrm{T}}$ $v \neq 0$ v 1 0 2D 8 3D 15

Table 3-1 “ ”

Table 3-1:

	$\begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix}$	6	
	$\begin{bmatrix} s\boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix}$	7	
	$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix}$	12	
	$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ \boldsymbol{a}^{\mathrm{T}} & v \end{bmatrix}$	15	

3.6 Eigen

3.6.1 Eigen

Eigen

Listing 3.6: slambook2/ch3/useGeometry/useGeometry.cpp

```
#include <iostream>
#include <cmath>
using namespace std;

#include <Eigen/Core>
#include <Eigen/Geometry>

using namespace Eigen;
// □□□□□□ Eigen □□□□□□□□□

int main(int argc, char **argv) {
    // Eigen/Geometry □□□□□□□□□□□□□□□□
    // 3D □□□□□□□□ Matrix3d □ Matrix3f
    Matrix3d rotation_matrix = Matrix3d::Identity();
    // □□□□□□ AngleAxis, □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Matrix
    AngleAxisd rotation_vector(M_PI / 4, Vector3d(0, 0, 1));     //□ Z □□□ 45 □
    cout.precision(3);
    cout << "rotation matrix =\n" << rotation_vector.matrix() << endl;   //□matrix()
         □□□□□
    // □□□□□□□
    rotation_matrix = rotation_vector.toRotationMatrix();
    // □ AngleAxis □□□□□□□
    Vector3d v(1, 0, 0);
    Vector3d v_rotated = rotation_vector * v;
    cout << "(1,0,0) after rotation (by angle axis) = " << v_rotated.transpose() <<
         endl;
    // □□□□□□□
    v_rotated = rotation_matrix * v;
    cout << "(1,0,0) after rotation (by matrix) = " << v_rotated.transpose() << endl;

    // □□□: □□□□□□□□□□□□□□□□
    Vector3d euler_angles = rotation_matrix.eulerAngles(2, 1, 0); // □□□□ZYXroll pitch
         □□yaw
    cout << "yaw pitch roll = " << euler_angles.transpose() << endl;

    // □□□□□□□□ Eigen::Isometry
    Isometry3d T = Isometry3d::Identity();       // □□□□3□□□□□□□□□d44
    T.rotate(rotation_vector);                   // □□□□□□rotation_vector
    T.pretranslate(Vector3d(1, 3, 4));           // □□□□□□□(1,3,4)
    cout << "Transform matrix = \n" << T.matrix() << endl;

    // □□□□□□□□□□□□
    Vector3d v_transformed = T * v;                              // □□□R*v+t
    cout << "v tranformed = " << v_transformed.transpose() << endl;

    // □□□□□□□□□□□□□ Eigen::Affine3d □ Eigen::Projective3d □□□□

    // □□□
    // □□□□□□□□□□□□□□□□□□AngleAxis
    Quaterniond q = Quaterniond(rotation_vector);
    cout << "quaternion from rotation vector = " << q.coeffs().transpose()
    << endl;   // □□□□□□□coeffs(x,y,z,w),□□□□□□□□□□w
    // □□□□□□□□□□□□
    q = Quaterniond(rotation_matrix);
    cout << "quaternion from rotation matrix = " << q.coeffs().transpose() << endl;
    // □□□□□□□□□□□□□□□□□□□□□□□□
    v_rotated = q * v; // □□□□□□qvq^{-1}
    cout << "(1,0,0) after rotation = " << v_rotated.transpose() << endl;
    // □□□□□□□□□□□□□□□□□□□□
    cout << "should be equal to " << (q * Quaterniond(0, 1, 0, 0) * q.inverse()).
```

```
        coeffs().transpose() << endl;

    return 0;
}
```

Eigen d f

- 3×3 Eigen::Matrix3d
- 3×1 Eigen::AngleAxisd
- 3×1 Eigen::Vector3d
- 4×1 Eigen::Quaterniond
- 4×4 Eigen::Isometry3d
- 4×4 Eigen::Affine3d
- 4×4 Eigen::Projective3d

CMakeLists Eigen AngleAxis $\boldsymbol{v}$

`//eigen.tuxfamily.org/dox/group__TutorialGeometry.html`

3.6.2

W R_1 R_2 $\boldsymbol{q}_1 = [0.35, 0.2, 0.3, 0.1]^{\mathrm{T}}, \boldsymbol{t}_1 = [0.3, 0.1, 0.1]^{\mathrm{T}}$ $\boldsymbol{q}_2 = [-0.5, 0.4, -0.1, 0.2]^{\mathrm{T}}, \boldsymbol{t}_2 = [-0.1, 0.5, 0.3]^{\mathrm{T}}$ $\boldsymbol{q}$ $\boldsymbol{t}$ $\boldsymbol{T}_{R_k,W}, k = 1, 2$ $\boldsymbol{p}_{R_1} = [0.5, 0, 0.2]^{\mathrm{T}}$

Listing 3.7: slambook2/ch3/examples/coordinateTransform.cpp

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <Eigen/Core>
#include <Eigen/Geometry>

using namespace std;
using namespace Eigen;

int main(int argc, char** argv) {
    Quaterniond q1(0.35, 0.2, 0.3, 0.1), q2(-0.5, 0.4, -0.1, 0.2);
    q1.normalize();
    q2.normalize();
    Vector3d t1(0.3, 0.1, 0.1), t2(-0.1, 0.5, 0.3);
    Vector3d p1(0.5, 0, 0.2);

    Isometry3d T1w(q1), T2w(q2);
    T1w.pretranslate(t1);
    T2w.pretranslate(t2);

    Vector3d p2 = T2w * T1w.inverse() * p1;
    cout << endl << p2.transpose() << endl;
    return 0;
}
```

$[-0.0309731, 0.73499, 0.296108]^{\mathrm{T}}$

$$\boldsymbol{p}_{R_2} = \boldsymbol{T}_{R_2,W} \boldsymbol{T}_{W,R_1} \boldsymbol{p}_{R_1}$$

3.7

3.7.1

trajectory.txt

$$\text{time}, t_x, t_y, t_z, q_x, q_y, q_z, q_w,$$

time $\boldsymbol{t}$ $\boldsymbol{q}$ “ ” $\boldsymbol{T}_{WR}$ $\boldsymbol{T}_{RW}$ “ ” “ ” $\boldsymbol{O}_R$ $\boldsymbol{O}_W$

$$\boldsymbol{O}_W = \boldsymbol{T}_{WR}\boldsymbol{O}_R = \boldsymbol{t}_{WR}. \tag{3.48}$$

$\boldsymbol{T}_{WR}$ $\boldsymbol{T}_{WR}$ $\boldsymbol{T}_{WR}$ $\boldsymbol{T}_{WR}$ $\boldsymbol{T}_{RW}$ 3D 3D matlab python matplotlib OpenGL linux OpenGL Pangolin *

Listing 3.8: slambook2/ch3/examples/plotTrajectory.cpp

```
#include <pangolin/pangolin.h>
#include <Eigen/Core>
#include <unistd.h>

using namespace std;
using namespace Eigen;

// path to trajectory file
string trajectory_file = "./examples/trajectory.txt";

void DrawTrajectory(vector<Isometry3d, Eigen::aligned_allocator<Isometry3d>>);

int main(int argc, char **argv) {
    vector<Isometry3d, Eigen::aligned_allocator<Isometry3d>> poses;
    ifstream fin(trajectory_file);
    if (!fin) {
        cout << "cannot find trajectory file at " << trajectory_file << endl;
        return 1;
    }

    while (!fin.eof()) {
        double time, tx, ty, tz, qx, qy, qz, qw;
        fin >> time >> tx >> ty >> tz >> qx >> qy >> qz >> qw;
        Isometry3d Twr(Quaterniond(qw, qx, qy, qz));
        Twr.pretranslate(Vector3d(tx, ty, tz));
        poses.push_back(Twr);
    }
    cout << "read total " << poses.size() << " pose entries" << endl;

    // draw trajectory in pangolin
    DrawTrajectory(poses);
    return 0;
}

void DrawTrajectory(vector<Isometry3d, Eigen::aligned_allocator<Isometry3d>> poses) {
    // create pangolin window and plot the trajectory
    pangolin::CreateWindowAndBind("Trajectory Viewer", 1024, 768);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    pangolin::OpenGlRenderState s_cam(
    pangolin::ProjectionMatrix(1024, 768, 500, 500, 512, 389, 0.1, 1000),
    pangolin::ModelViewLookAt(0, -0.1, -1.8, 0, 0, 0, 0.0, -1.0, 0.0)
    );

```

* https://github.com/stevenlovegrove/Pangolin

```
    pangolin::View &d_cam = pangolin::CreateDisplay()
        .SetBounds(0.0, 1.0, 0.0, 1.0, -1024.0f / 768.0f)
        .SetHandler(new pangolin::Handler3D(s_cam));

    while (pangolin::ShouldQuit() == false) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        d_cam.Activate(s_cam);
        glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
        glLineWidth(2);
        for (size_t i = 0; i < poses.size(); i++) {
            // □□□□□□□□□□□□
            Vector3d Ow = poses[i].translation();
            Vector3d Xw = poses[i] * (0.1 * Vector3d(1, 0, 0));
            Vector3d Yw = poses[i] * (0.1 * Vector3d(0, 1, 0));
            Vector3d Zw = poses[i] * (0.1 * Vector3d(0, 0, 1));
            glBegin(GL_LINES);
            glColor3f(1.0, 0.0, 0.0);
            glVertex3d(Ow[0], Ow[1], Ow[2]);
            glVertex3d(Xw[0], Xw[1], Xw[2]);
            glColor3f(0.0, 1.0, 0.0);
            glVertex3d(Ow[0], Ow[1], Ow[2]);
            glVertex3d(Yw[0], Yw[1], Yw[2]);
            glColor3f(0.0, 0.0, 1.0);
            glVertex3d(Ow[0], Ow[1], Ow[2]);
            glVertex3d(Zw[0], Zw[1], Zw[2]);
            glEnd();
        }
        // □□□□
        for (size_t i = 0; i < poses.size(); i++) {
            glColor3f(0.0, 0.0, 0.0);
            glBegin(GL_LINES);
            auto p1 = poses[i], p2 = poses[i + 1];
            glVertex3d(p1.translation()[0], p1.translation()[1], p1.translation()[2]);
            glVertex3d(p2.translation()[0], p2.translation()[1], p2.translation()[2]);
            glEnd();
        }
        pangolin::FinishFrame();
        usleep(5000);   // sleep 5 ms
    }
}
```

Panglin 3D Figure 3-3

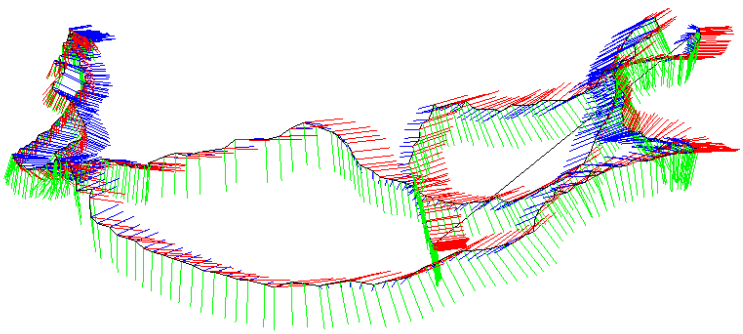


Figure 3-3:

3.7.2

3D slambook2/ch3/visualizeGeometry Figure 3-4

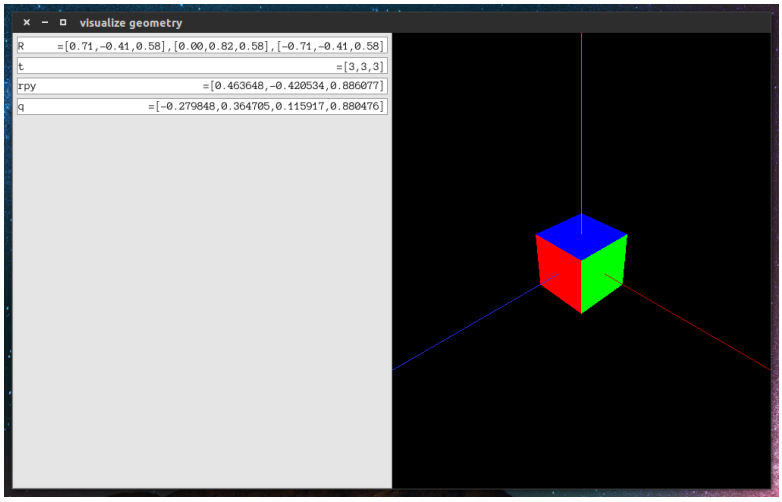


Figure 3-4:

1.

2.*

3. (3.33)

4.

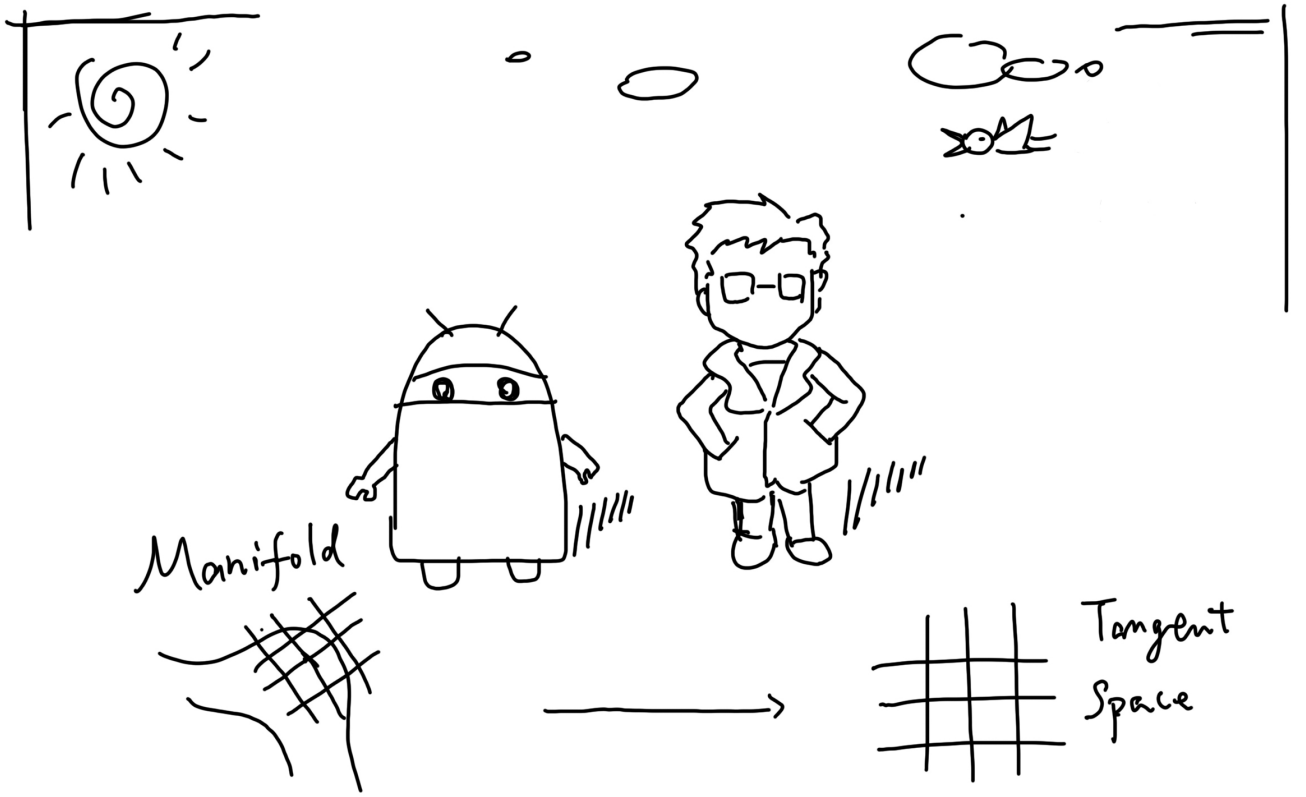
5. Eigen 3×3 $\boldsymbol{I}_{3\times3}$

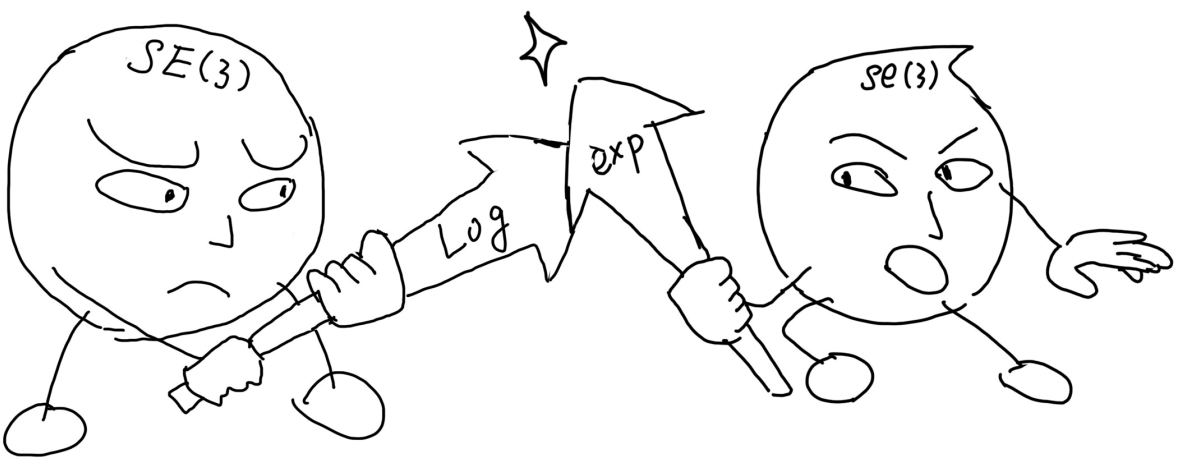
6.* $\boldsymbol{Ax} = \boldsymbol{b}$ Eigen

Chapter 4

1. SO(3), SE(3)
2. BCH
3.
4. Sophus

SLAM SLAM “ ”
1 —





$$T = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \in SE(3) \qquad \xi = [\rho, \phi]^T \in se(3)$$

$$\log(T)^{\vee} = \xi$$

$$\exp(\xi^{\wedge}) = T$$

4×4 $\qquad$ 6×1

4.1

SO(3) SE(3)

$$\mathrm{SO}(3) = \{\boldsymbol{R} \in \mathbb{R}^{3\times 3} | \boldsymbol{R}\boldsymbol{R}^{\mathrm{T}} = \boldsymbol{I}, \det(\boldsymbol{R}) = 1\}. \tag{4.1}$$

$$\mathrm{SE}(3) = \left\{ \boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} | \boldsymbol{R} \in \mathrm{SO}(3), \boldsymbol{t} \in \mathbb{R}^3 \right\}. \tag{4.2}$$

$\boldsymbol{R}_1, \boldsymbol{R}_2$

$$\boldsymbol{R}_1 + \boldsymbol{R}_2 \notin \mathrm{SO}(3), \quad \boldsymbol{T}_1 + \boldsymbol{T}_2 \notin \mathrm{SE}(3). \tag{4.3}$$

SO(3) SE(3) :

$$\boldsymbol{R}_1\boldsymbol{R}_2 \in \mathrm{SO}(3), \quad \boldsymbol{T}_1\boldsymbol{T}_2 \in \mathrm{SE}(3). \tag{4.4}$$

4.1.1

Group $A \quad \cdot \quad G = (A, \cdot)$

1. $\forall a_1, a_2 \in A, \quad a_1 \cdot a_2 \in A.$
2. $\forall a_1, a_2, a_3 \in A, \quad (a_1 \cdot a_2) \cdot a_3 = a_1 \cdot (a_2 \cdot a_3).$
3. $\exists a_0 \in A, \quad \text{s.t.} \quad \forall a \in A, \quad a_0 \cdot a = a \cdot a_0 = a.$
4. $\forall a \in A, \quad \exists a^{-1} \in A, \quad s.t. \quad a \cdot a^{-1} = a_0.$

“ ”* $(\mathbb{Z}, +)$ 0 1 $(\mathbb{Q}\backslash 0, \cdot)$

- GL(n) $n \times n$
- SO(n) SO(2) SO(3)
- SE(n) n SE(2) SE(3)

$\mathbb{Z}$ SO(n) SE(n)
[22] SO(3)

4.1.2

$\boldsymbol{R}$

$$\boldsymbol{R}\boldsymbol{R}^{\mathrm{T}} = \boldsymbol{I}. \tag{4.5}$$

$\boldsymbol{R}$ $\boldsymbol{R}(t)$

$$\boldsymbol{R}(t)\boldsymbol{R}(t)^T = \boldsymbol{I}.$$

---

* “ ”

$$\dot{\boldsymbol{R}}(t)\boldsymbol{R}(t)^{\mathrm{T}} + \boldsymbol{R}(t)\dot{\boldsymbol{R}}(t)^{\mathrm{T}} = 0.$$

$$\dot{\boldsymbol{R}}(t)\boldsymbol{R}(t)^{\mathrm{T}} = -\left(\dot{\boldsymbol{R}}(t)\boldsymbol{R}(t)^{\mathrm{T}}\right)^{\mathrm{T}}. \tag{4.6}$$

$\dot{\boldsymbol{R}}(t)\boldsymbol{R}(t)^{\mathrm{T}}$ (3.3) $\wedge$ $\vee$

$$\boldsymbol{a}^{\wedge} = \boldsymbol{A} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}, \quad \boldsymbol{A}^{\vee} = \boldsymbol{a}. \tag{4.7}$$

$\dot{\boldsymbol{R}}(t)\boldsymbol{R}(t)^{\mathrm{T}}$ $\boldsymbol{\phi}(t) \in \mathbb{R}^3$

$$\dot{\boldsymbol{R}}(t)\boldsymbol{R}(t)^{\mathrm{T}} = \boldsymbol{\phi}(t)^{\wedge}.$$

$\boldsymbol{R}(t)$ $\boldsymbol{R}$

$$\dot{\boldsymbol{R}}(t) = \boldsymbol{\phi}(t)^{\wedge}\boldsymbol{R}(t) = \begin{bmatrix} 0 & -\phi_3 & \phi_2 \\ \phi_3 & 0 & -\phi_1 \\ -\phi_2 & \phi_1 & 0 \end{bmatrix} \boldsymbol{R}(t). \tag{4.8}$$

$\boldsymbol{\phi}^{\wedge}(t)$ $t_0 = 0$ $\boldsymbol{R}(0) = \boldsymbol{I}$ $\boldsymbol{R}(t)$ $t = 0$

$$\begin{aligned} \boldsymbol{R}(t) &\approx \boldsymbol{R}(t_0) + \dot{\boldsymbol{R}}(t_0)(t - t_0) \\ &= \boldsymbol{I} + \boldsymbol{\phi}(t_0)^{\wedge}(t). \end{aligned} \tag{4.9}$$

$\boldsymbol{\phi}$ $\boldsymbol{R}$ SO(3) (Tangent Space) t_0 $\boldsymbol{\phi}$ $\boldsymbol{\phi}(t_0) = \boldsymbol{\phi}_0$ (4.8)

$$\dot{\boldsymbol{R}}(t) = \boldsymbol{\phi}(t_0)^{\wedge}\boldsymbol{R}(t) = \boldsymbol{\phi}_0^{\wedge}\boldsymbol{R}(t).$$

$\boldsymbol{R}$ $\boldsymbol{R}(0) = \boldsymbol{I}$

$$\boldsymbol{R}(t) = \exp\left(\boldsymbol{\phi}_0^{\wedge}t\right). \tag{4.10}$$

$t = 0$ $\exp\left(\boldsymbol{\phi}_0^{\wedge}t\right)$ * $\boldsymbol{R}$ $\boldsymbol{\phi}_0^{\wedge}t$

1. $\boldsymbol{R}$ $\boldsymbol{\phi}$ $\boldsymbol{R}$ $\boldsymbol{R}$ $\boldsymbol{\phi}$ $\boldsymbol{\phi}$ SO(3) $\mathfrak{so}(3)$
2. $\boldsymbol{\phi}$ $\exp(\boldsymbol{\phi}^{\wedge})$ $\boldsymbol{R}$ $\boldsymbol{\phi}$ /

4.1.3

$\mathbb{V}$ $\mathbb{F}$ $[,]$ $(\mathbb{V}, \mathbb{F}, [,])$ $\mathfrak{g}$

1. $\forall \boldsymbol{X}, \boldsymbol{Y} \in \mathbb{V}, [\boldsymbol{X}, \boldsymbol{Y}] \in \mathbb{V}.$

* exp

2. $\forall \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z} \in \mathbb{V}, a, b \in \mathbb{F}$,

$$[a\boldsymbol{X} + b\boldsymbol{Y}, \boldsymbol{Z}] = a[\boldsymbol{X}, \boldsymbol{Z}] + b[\boldsymbol{Y}, \boldsymbol{Z}], \quad [\boldsymbol{Z}, a\boldsymbol{X} + b\boldsymbol{Y}] = a[\boldsymbol{Z}, \boldsymbol{X}] + b[\boldsymbol{Z}, \boldsymbol{Y}].$$

3. * $\forall \boldsymbol{X} \in \mathbb{V}, [\boldsymbol{X}, \boldsymbol{X}] = \boldsymbol{0}$.

4. $\forall \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z} \in \mathbb{V}, [\boldsymbol{X}, [\boldsymbol{Y}, \boldsymbol{Z}]] + [\boldsymbol{Z}, [\boldsymbol{X}, \boldsymbol{Y}]] + [\boldsymbol{Y}, [\boldsymbol{Z}, \boldsymbol{X}]] = \boldsymbol{0}$.

$\mathbb{R}^3$ $\times$ $\mathfrak{g} = (\mathbb{R}^3, \mathbb{R}, \times)$

4.1.4 $\mathfrak{so}(3)$

$\boldsymbol{\phi}$ SO(3) $\mathbb{R}^3$ $\boldsymbol{\phi}$ $\boldsymbol{\phi}$

$$\boldsymbol{\Phi} = \boldsymbol{\phi}^\wedge = \begin{bmatrix} 0 & -\phi_3 & \phi_2 \\ \phi_3 & 0 & -\phi_1 \\ -\phi_2 & \phi_1 & 0 \end{bmatrix} \in \mathbb{R}^{3\times 3}. \tag{4.11}$$

$\boldsymbol{\phi}_1, \boldsymbol{\phi}_2$

$$[\boldsymbol{\phi}_1, \boldsymbol{\phi}_2] = (\boldsymbol{\Phi}_1\boldsymbol{\Phi}_2 - \boldsymbol{\Phi}_2\boldsymbol{\Phi}_1)^\vee. \tag{4.12}$$

$\boldsymbol{\phi}$ $\mathfrak{so}(3)$

$$\mathfrak{so}(3) = \left\{ \boldsymbol{\phi} \in \mathbb{R}^3, \boldsymbol{\Phi} = \boldsymbol{\phi}^\wedge \in \mathbb{R}^{3\times 3} \right\}. \tag{4.13}$$

$\widehat{\boldsymbol{\phi}}$ $\mathfrak{so}(3)$ SO(3)

$$\boldsymbol{R} = \exp(\boldsymbol{\phi}^\wedge). \tag{4.14}$$

$\mathfrak{so}(3)$ SE(3)

4.1.5 $\mathfrak{se}(3)$

SE(3) $\mathfrak{se}(3)$ $\mathfrak{se}(3)$ $\mathfrak{so}(3)$ $\mathfrak{se}(3)$ $\mathbb{R}^6$

$$\mathfrak{se}(3) = \left\{ \boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{\rho} \\ \boldsymbol{\phi} \end{bmatrix} \in \mathbb{R}^6, \boldsymbol{\rho} \in \mathbb{R}^3, \boldsymbol{\phi} \in \mathfrak{so}(3), \boldsymbol{\xi}^\wedge = \begin{bmatrix} \boldsymbol{\phi}^\wedge & \boldsymbol{\rho} \\ \boldsymbol{0}^{\mathrm{T}} & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4} \right\}. \tag{4.15}$$

$\mathfrak{se}(3)$ $\boldsymbol{\xi}$ $\boldsymbol{\rho}$ $\boldsymbol{\phi}$ $\mathfrak{so}(3)$ † $\wedge$ $\mathfrak{se}(3)$ $\wedge$

$$\boldsymbol{\xi}^\wedge = \begin{bmatrix} \boldsymbol{\phi}^\wedge & \boldsymbol{\rho} \\ \boldsymbol{0}^{\mathrm{T}} & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4}. \tag{4.16}$$

$\wedge$ $\vee$ “ ” “ ” $\mathfrak{so}(3)$ $\mathfrak{se}(3)$ “ $\mathfrak{so}(3)$ ” $\boldsymbol{\rho}$ $\mathfrak{se}(3)$ $\mathfrak{so}$

$$[\boldsymbol{\xi}_1, \boldsymbol{\xi}_2] = (\boldsymbol{\xi}_1^\wedge\boldsymbol{\xi}_2^\wedge - \boldsymbol{\xi}_2^\wedge\boldsymbol{\xi}_1^\wedge)^\vee. \tag{4.17}$$

$\mathfrak{so}(3)$ $\mathfrak{se}(3)$

---

*

†

4.2

4.2.1 SO(3)

$\exp(\boldsymbol{\phi}^\wedge)$ Exponential Map $\mathfrak{so}(3)$ $\mathfrak{se}(3)$

$$\exp(\boldsymbol{A}) = \sum_{n=0}^{\infty} \frac{1}{n!}\boldsymbol{A}^n. \tag{4.18}$$

$\mathfrak{so}(3)$ $\boldsymbol{\phi}$

$$\exp(\boldsymbol{\phi}^\wedge) = \sum_{n=0}^{\infty} \frac{1}{n!}(\boldsymbol{\phi}^\wedge)^n. \tag{4.19}$$

$\boldsymbol{\phi}$ $\theta\,\boldsymbol{a}$ $\boldsymbol{\phi} = \theta\boldsymbol{a}$ $\boldsymbol{a}$ 1 $\|\boldsymbol{a}\| =$ 1 $\boldsymbol{a}^\wedge$

$$\boldsymbol{a}^\wedge\boldsymbol{a}^\wedge = \begin{bmatrix} -a_2^2 - a_3^2 & a_1a_2 & a_1a_3 \\ a_1a_2 & -a_1^2 - a_3^2 & a_2a_3 \\ a_1a_3 & a_2a_3 & -a_1^2 - a_2^2 \end{bmatrix} = \boldsymbol{a}\boldsymbol{a}^{\mathrm{T}} - \boldsymbol{I}, \tag{4.20}$$

$$\boldsymbol{a}^\wedge\boldsymbol{a}^\wedge\boldsymbol{a}^\wedge = \boldsymbol{a}^\wedge(\boldsymbol{a}\boldsymbol{a}^{\mathrm{T}} - \boldsymbol{I}) = -\boldsymbol{a}^\wedge. \tag{4.21}$$

$\boldsymbol{a}^\wedge$

$$\begin{aligned}
\exp(\boldsymbol{\phi}^\wedge) &= \exp(\theta\boldsymbol{a}^\wedge) = \sum_{n=0}^{\infty} \frac{1}{n!}(\theta\boldsymbol{a}^\wedge)^n \\
&= \boldsymbol{I} + \theta\boldsymbol{a}^\wedge + \frac{1}{2!}\theta^2\boldsymbol{a}^\wedge\boldsymbol{a}^\wedge + \frac{1}{3!}\theta^3\boldsymbol{a}^\wedge\boldsymbol{a}^\wedge\boldsymbol{a}^\wedge + \frac{1}{4!}\theta^4(\boldsymbol{a}^\wedge)^4 + ... \\
&= \boldsymbol{a}\boldsymbol{a}^{\mathrm{T}} - \boldsymbol{a}^\wedge\boldsymbol{a}^\wedge + \theta\boldsymbol{a}^\wedge + \frac{1}{2!}\theta^2\boldsymbol{a}^\wedge\boldsymbol{a}^\wedge - \frac{1}{3!}\theta^3\boldsymbol{a}^\wedge - \frac{1}{4!}\theta^4(\boldsymbol{a}^\wedge)^2 + ... \\
&= \boldsymbol{a}\boldsymbol{a}^{\mathrm{T}} + \underbrace{\left(\theta - \frac{1}{3!}\theta^3 + \frac{1}{5!}\theta^5 - ...\right)}_{\sin\theta}\boldsymbol{a}^\wedge - \underbrace{\left(1 - \frac{1}{2!}\theta^2 + \frac{1}{4!}\theta^4 - ...\right)}_{\cos\theta}\boldsymbol{a}^\wedge\boldsymbol{a}^\wedge \\
&= \boldsymbol{a}^\wedge\boldsymbol{a}^\wedge + \boldsymbol{I} + \sin\theta\boldsymbol{a}^\wedge - \cos\theta\boldsymbol{a}^\wedge\boldsymbol{a}^\wedge \\
&= (1 - \cos\theta)\boldsymbol{a}^\wedge\boldsymbol{a}^\wedge + \boldsymbol{I} + \sin\theta\boldsymbol{a}^\wedge \\
&= \cos\theta\boldsymbol{I} + (1 - \cos\theta)\boldsymbol{a}\boldsymbol{a}^{\mathrm{T}} + \sin\theta\boldsymbol{a}^\wedge.
\end{aligned}$$

$$\exp(\theta\boldsymbol{a}^\wedge) = \cos\theta\boldsymbol{I} + (1 - \cos\theta)\boldsymbol{a}\boldsymbol{a}^{\mathrm{T}} + \sin\theta\boldsymbol{a}^\wedge. \tag{4.22}$$

(3.15) $\mathfrak{so}(3)$ $\mathfrak{so}(3)$ SO(3) SO(3)

$$\boldsymbol{\phi} = \ln(\boldsymbol{R})^\vee = \left(\sum_{n=0}^{\infty} \frac{(-1)^n}{n+1}(\boldsymbol{R} - \boldsymbol{I})^{n+1}\right)^\vee. \tag{4.23}$$

3 (3.17)

$\boldsymbol{R}$ $\boldsymbol{\phi}$ SO(3) $\mathfrak{so}(3)$ $\mathfrak{so}(3)$ SO

— ±

SO(3) $\mathfrak{so}(3)$

4.2.2 SE(3)

$\mathfrak{se}(3)$ $\mathfrak{so}(3)$ $\mathfrak{se}(3)$

$$\exp\left(\boldsymbol{\xi}^{\wedge}\right) = \begin{bmatrix} \sum_{n=0}^{\infty} \frac{1}{n!}(\boldsymbol{\phi}^{\wedge})^n & \sum_{n=0}^{\infty} \frac{1}{(n+1)!}(\boldsymbol{\phi}^{\wedge})^n \boldsymbol{\rho} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} \tag{4.24}$$

$$\triangleq \begin{bmatrix} \boldsymbol{R} & \boldsymbol{J}\boldsymbol{\rho} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} = \boldsymbol{T}. \tag{4.25}$$

$\mathfrak{so}(3)$ exp $\boldsymbol{\phi} = \theta \boldsymbol{a}$ $\boldsymbol{a}$

$$\begin{aligned} \sum_{n=0}^{\infty} \frac{1}{(n+1)!}(\boldsymbol{\phi}^{\wedge})^n &= \boldsymbol{I} + \frac{1}{2!}\theta \boldsymbol{a}^{\wedge} + \frac{1}{3!}\theta^2(\boldsymbol{a}^{\wedge})^2 + \frac{1}{4!}\theta^3(\boldsymbol{a}^{\wedge})^3 + \frac{1}{5!}\theta^4(\boldsymbol{a}^{\wedge})^4 \cdots \\ &= \frac{1}{\theta}\left(\frac{1}{2!}\theta^2 - \frac{1}{4!}\theta^4 + \cdots\right)(\boldsymbol{a}^{\wedge}) + \frac{1}{\theta}\left(\frac{1}{3!}\theta^3 - \frac{1}{5}\theta^5 + \cdots\right)(\boldsymbol{a}^{\wedge})^2 + \boldsymbol{I} \\ &= \frac{1}{\theta}(1 - \cos\theta)(\boldsymbol{a}^{\wedge}) + \frac{\theta - \sin\theta}{\theta}\left(\boldsymbol{a}\boldsymbol{a}^{T} - \boldsymbol{I}\right) + \boldsymbol{I} \\ &= \frac{\sin\theta}{\theta}\boldsymbol{I} + \left(1 - \frac{\sin\theta}{\theta}\right)\boldsymbol{a}\boldsymbol{a}^{T} + \frac{1 - \cos\theta}{\theta}\boldsymbol{a}^{\wedge} \triangleq \boldsymbol{J}. \end{aligned} \tag{4.26}$$

$\boldsymbol{\xi}$ $\boldsymbol{R}$ SO(3) $\mathfrak{se}(3)$ $\boldsymbol{\phi}$ $\boldsymbol{J}$

$$\boldsymbol{J} = \frac{\sin\theta}{\theta}\boldsymbol{I} + \left(1 - \frac{\sin\theta}{\theta}\right)\boldsymbol{a}\boldsymbol{a}^{\mathrm{T}} + \frac{1 - \cos\theta}{\theta}\boldsymbol{a}^{\wedge}. \tag{4.27}$$

$\boldsymbol{J}$ $\boldsymbol{J}$

$\boldsymbol{T}$ $\mathfrak{so}(3)$ $\boldsymbol{R}$ $\boldsymbol{t}$

$$\boldsymbol{t} = \boldsymbol{J}\boldsymbol{\rho}. \tag{4.28}$$

$\boldsymbol{J}$ $\boldsymbol{\phi}$ $\boldsymbol{\rho}$ Figure 4-1

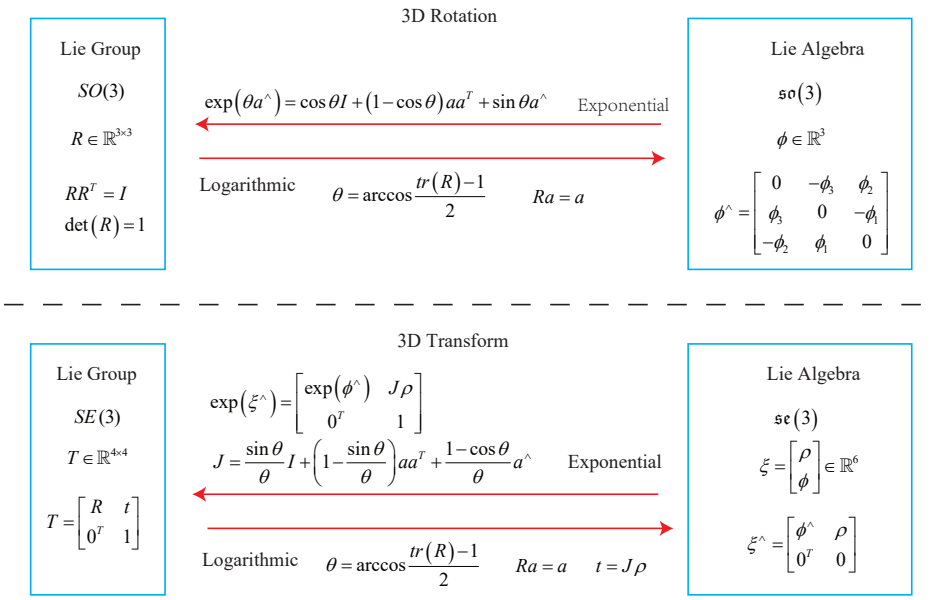


Figure 4-1: $\mathrm{SO}(3), \mathrm{SE}(3), \mathfrak{so}(3), \mathfrak{se}(3)$

4.3

4.3.1 BCH

6 SO(3) SE(3) SO(3) $\mathfrak{so}(3)$ $\mathfrak{so}(3)$

$$\exp\left(\boldsymbol{\phi}_1^\wedge\right)\exp\left(\boldsymbol{\phi}_2^\wedge\right)=\exp\left(\left(\boldsymbol{\phi}_1+\boldsymbol{\phi}_2\right)^\wedge\right)?$$

$\boldsymbol{\phi}_1, \boldsymbol{\phi}_2$

$$\ln\left(\exp\left(\boldsymbol{A}\right)\exp\left(\boldsymbol{B}\right)\right)=\boldsymbol{A}+\boldsymbol{B}\ ?$$

Baker-Campbell-Hausdorff BCH *

$$\ln\left(\exp\left(\boldsymbol{A}\right)\exp\left(\boldsymbol{B}\right)\right)=\boldsymbol{A}+\boldsymbol{B}+\frac{1}{2}\left[\boldsymbol{A},\boldsymbol{B}\right]+\frac{1}{12}\left[\boldsymbol{A},\left[\boldsymbol{A},\boldsymbol{B}\right]\right]-\frac{1}{12}\left[\boldsymbol{B},\left[\boldsymbol{A},\boldsymbol{B}\right]\right]+\cdots \tag{4.29}$$

[] BCH SO(3) $\ln\left(\exp\left(\boldsymbol{\phi}_1^\wedge\right)\exp\left(\boldsymbol{\phi}_2^\wedge\right)\right)^\vee$ $\boldsymbol{\phi_1}$ $\boldsymbol{\phi_2}$ BCH

$$\ln\left(\exp\left(\boldsymbol{\phi}_1^\wedge\right)\exp\left(\boldsymbol{\phi}_2^\wedge\right)\right)^\vee\approx\begin{cases}\boldsymbol{J}_l(\boldsymbol{\phi}_2)^{-1}\boldsymbol{\phi}_1+\boldsymbol{\phi}_2 & \boldsymbol{\phi}_1 \quad ,\\ \boldsymbol{J}_r(\boldsymbol{\phi}_1)^{-1}\boldsymbol{\phi}_2+\boldsymbol{\phi}_1 & \boldsymbol{\phi}_2 \quad .\end{cases} \tag{4.30}$$

$\boldsymbol{R}_2$ $\boldsymbol{\phi}_2$ $\boldsymbol{R}_1$ $\boldsymbol{\phi}_1$ $\boldsymbol{\phi}_2$ $\boldsymbol{J}_l(\boldsymbol{\phi}_2)^{-1}\boldsymbol{\phi}_1$ BCH
BCH $\boldsymbol{J}_l$ (4.27)

$$\boldsymbol{J}_l=\boldsymbol{J}=\frac{\sin\theta}{\theta}\boldsymbol{I}+\left(1-\frac{\sin\theta}{\theta}\right)\boldsymbol{a}\boldsymbol{a}^{\mathrm{T}}+\frac{1-\cos\theta}{\theta}\boldsymbol{a}^\wedge. \tag{4.31}$$

$$\boldsymbol{J}_l^{-1}=\frac{\theta}{2}\cot\frac{\theta}{2}\boldsymbol{I}+\left(1-\frac{\theta}{2}\cot\frac{\theta}{2}\right)\boldsymbol{a}\boldsymbol{a}^{\mathrm{T}}-\frac{\theta}{2}\boldsymbol{a}^\wedge. \tag{4.32}$$

$$\boldsymbol{J}_r(\boldsymbol{\phi})=\boldsymbol{J}_l(-\boldsymbol{\phi}). \tag{4.33}$$

BCH $\boldsymbol{R}$ $\boldsymbol{\phi}$ $\Delta\boldsymbol{R}$ $\Delta\boldsymbol{\phi}$ $\Delta\boldsymbol{R}\cdot\boldsymbol{R}$ BCH $\boldsymbol{J}_l^{-1}(\boldsymbol{\phi})\Delta\boldsymbol{\phi}$-
$\boldsymbol{\phi}$

$$\exp\left(\Delta\boldsymbol{\phi}^\wedge\right)\exp\left(\boldsymbol{\phi}^\wedge\right)=\exp\left(\left(\boldsymbol{\phi}+\boldsymbol{J}_l^{-1}\left(\boldsymbol{\phi}\right)\Delta\boldsymbol{\phi}\right)^\wedge\right). \tag{4.34}$$

$\boldsymbol{\phi}$ $\Delta\boldsymbol{\phi}$

$$\exp\left(\left(\boldsymbol{\phi}+\Delta\boldsymbol{\phi}\right)^\wedge\right)=\exp\left(\left(\boldsymbol{J}_l\Delta\boldsymbol{\phi}\right)^\wedge\right)\exp\left(\boldsymbol{\phi}^\wedge\right)=\exp\left(\boldsymbol{\phi}^\wedge\right)\exp\left(\left(\boldsymbol{J}_r\Delta\boldsymbol{\phi}\right)^\wedge\right). \tag{4.35}$$

SE(3) BCH

$$\exp\left(\Delta\boldsymbol{\xi}^\wedge\right)\exp\left(\boldsymbol{\xi}^\wedge\right)\approx\exp\left(\left(\boldsymbol{\mathcal{J}}_l^{-1}\Delta\boldsymbol{\xi}+\boldsymbol{\xi}\right)^\wedge\right), \tag{4.36}$$

$$\exp\left(\boldsymbol{\xi}^\wedge\right)\exp\left(\Delta\boldsymbol{\xi}^\wedge\right)\approx\exp\left(\left(\boldsymbol{\mathcal{J}}_r^{-1}\Delta\boldsymbol{\xi}+\boldsymbol{\xi}\right)^\wedge\right). \tag{4.37}$$

$\boldsymbol{\mathcal{J}}_l$ 6×6 [6] (7.82) (7.83)

---

* https://en.wikipedia.org/wiki/Baker-Campbell-Hausdorff_formula
† BCH [6]

4.3.2 SO(3)

SLAM SO(3) SE(3) $\boldsymbol{T}$ $\boldsymbol{p}$ $\boldsymbol{z}$

$$\boldsymbol{z} = \boldsymbol{T}\boldsymbol{p} + \boldsymbol{w}. \tag{4.38}$$

$\boldsymbol{w}$ $\boldsymbol{z}$ $\boldsymbol{z} = \boldsymbol{T}\boldsymbol{p}$

$$\boldsymbol{e} = \boldsymbol{z} - \boldsymbol{T}\boldsymbol{p}. \tag{4.39}$$

N N $\boldsymbol{T}$

$$\min_{\boldsymbol{T}} J(\boldsymbol{T}) = \sum_{i=1}^{N} \|\boldsymbol{z}_i - \boldsymbol{T}\boldsymbol{p}_i\|_2^2 . \tag{4.40}$$

J $\boldsymbol{T}$ SO(3), SE(3) $\boldsymbol{T}$

1.

2.

4.3.3

SO(3) $\boldsymbol{p}$ $\boldsymbol{R}\boldsymbol{p}$ *

$$\frac{\partial (\boldsymbol{R}\boldsymbol{p})}{\partial \boldsymbol{R}}.$$

SO(3) $\boldsymbol{R}$ $\boldsymbol{\phi}$ †

$$\frac{\partial \left(\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}\right)}{\partial \boldsymbol{\phi}}.$$

$$\begin{aligned}
\frac{\partial \left(\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}\right)}{\partial \boldsymbol{\phi}} &= \lim_{\delta\boldsymbol{\phi}\to\boldsymbol{0}} \frac{\exp\left((\boldsymbol{\phi}+\delta\boldsymbol{\phi})^{\wedge}\right)\boldsymbol{p} - \exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}}{\delta\boldsymbol{\phi}} \\
&= \lim_{\delta\boldsymbol{\phi}\to\boldsymbol{0}} \frac{\exp\left((\boldsymbol{J}_l\delta\boldsymbol{\phi})^{\wedge}\right)\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p} - \exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}}{\delta\boldsymbol{\phi}} \\
&= \lim_{\delta\boldsymbol{\phi}\to\boldsymbol{0}} \frac{\left(\boldsymbol{I} + (\boldsymbol{J}_l\delta\boldsymbol{\phi})^{\wedge}\right)\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p} - \exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}}{\delta\boldsymbol{\phi}} \\
&= \lim_{\delta\boldsymbol{\phi}\to\boldsymbol{0}} \frac{(\boldsymbol{J}_l\delta\boldsymbol{\phi})^{\wedge}\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}}{\delta\boldsymbol{\phi}} \\
&= \lim_{\delta\boldsymbol{\phi}\to\boldsymbol{0}} \frac{-(\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p})^{\wedge}\boldsymbol{J}_l\delta\boldsymbol{\phi}}{\delta\boldsymbol{\phi}} = -(\boldsymbol{R}\boldsymbol{p})^{\wedge}\boldsymbol{J}_l.
\end{aligned}$$

2 BCH 3 4 5

$$\frac{\partial (\boldsymbol{R}\boldsymbol{p})}{\partial \boldsymbol{\phi}} = (-\boldsymbol{R}\boldsymbol{p})^{\wedge}\boldsymbol{J}_l. \tag{4.41}$$

$\boldsymbol{J}_l$

---

*

† $\mathrm{d}(\boldsymbol{A}\boldsymbol{x})/\mathrm{d}\boldsymbol{x} = \boldsymbol{A}$

4.3.4

$\boldsymbol{R}$ $\Delta\boldsymbol{R}$ $\Delta\boldsymbol{R}$ $\boldsymbol{\varphi}$ $\boldsymbol{\varphi}$

$$\frac{\partial (\boldsymbol{R}\boldsymbol{p})}{\partial \boldsymbol{\varphi}} = \lim_{\boldsymbol{\varphi}\to\boldsymbol{0}} \frac{\exp\left(\boldsymbol{\varphi}^{\wedge}\right)\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p} - \exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}}{\boldsymbol{\varphi}}. \tag{4.42}$$

$$\begin{aligned}\frac{\partial (\boldsymbol{R}\boldsymbol{p})}{\partial \boldsymbol{\varphi}} &= \lim_{\boldsymbol{\varphi}\to\boldsymbol{0}} \frac{\exp\left(\boldsymbol{\varphi}^{\wedge}\right)\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p} - \exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}}{\boldsymbol{\varphi}} \\ &= \lim_{\boldsymbol{\varphi}\to\boldsymbol{0}} \frac{\left(\boldsymbol{I}+\boldsymbol{\varphi}^{\wedge}\right)\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p} - \exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}}{\boldsymbol{\varphi}} \\ &= \lim_{\boldsymbol{\varphi}\to\boldsymbol{0}} \frac{\boldsymbol{\varphi}^{\wedge}\boldsymbol{R}\boldsymbol{p}}{\boldsymbol{\varphi}} = \lim_{\boldsymbol{\varphi}\to\boldsymbol{0}} \frac{-(\boldsymbol{R}\boldsymbol{p})^{\wedge}\boldsymbol{\varphi}}{\boldsymbol{\varphi}} = -(\boldsymbol{R}\boldsymbol{p})^{\wedge}.\end{aligned}$$

$\boldsymbol{J}_l$

4.3.5 SE(3)

SE(3) $\boldsymbol{p}$ $\boldsymbol{T}$ $\boldsymbol{\xi}$ $\boldsymbol{T}\boldsymbol{p}^{*}$ $\boldsymbol{T}$ $\Delta\boldsymbol{T} = \exp\left(\delta\boldsymbol{\xi}^{\wedge}\right)$ $\delta\boldsymbol{\xi} = [\delta\boldsymbol{\rho}, \delta\boldsymbol{\phi}]^{\mathrm{T}}$

$$\begin{aligned}\frac{\partial (\boldsymbol{T}\boldsymbol{p})}{\partial \delta\boldsymbol{\xi}} &= \lim_{\delta\boldsymbol{\xi}\to\boldsymbol{0}} \frac{\exp\left(\delta\boldsymbol{\xi}^{\wedge}\right)\exp\left(\boldsymbol{\xi}^{\wedge}\right)\boldsymbol{p} - \exp\left(\boldsymbol{\xi}^{\wedge}\right)\boldsymbol{p}}{\delta\boldsymbol{\xi}} \\ &= \lim_{\delta\boldsymbol{\xi}\to\boldsymbol{0}} \frac{\left(\boldsymbol{I}+\delta\boldsymbol{\xi}^{\wedge}\right)\exp\left(\boldsymbol{\xi}^{\wedge}\right)\boldsymbol{p} - \exp\left(\boldsymbol{\xi}^{\wedge}\right)\boldsymbol{p}}{\delta\boldsymbol{\xi}} \\ &= \lim_{\delta\boldsymbol{\xi}\to\boldsymbol{0}} \frac{\delta\boldsymbol{\xi}^{\wedge}\exp\left(\boldsymbol{\xi}^{\wedge}\right)\boldsymbol{p}}{\delta\boldsymbol{\xi}} \\ &= \lim_{\delta\boldsymbol{\xi}\to\boldsymbol{0}} \frac{\begin{bmatrix}\delta\boldsymbol{\phi}^{\wedge} & \delta\boldsymbol{\rho}\\ \boldsymbol{0}^{\mathrm{T}} & 0\end{bmatrix}\begin{bmatrix}\boldsymbol{R}\boldsymbol{p}+\boldsymbol{t}\\ 1\end{bmatrix}}{\delta\boldsymbol{\xi}} \\ &= \lim_{\delta\boldsymbol{\xi}\to\boldsymbol{0}} \frac{\begin{bmatrix}\delta\boldsymbol{\phi}^{\wedge}\left(\boldsymbol{R}\boldsymbol{p}+\boldsymbol{t}\right)+\delta\boldsymbol{\rho}\\ \boldsymbol{0}^{\mathrm{T}}\end{bmatrix}}{[\delta\boldsymbol{\rho}, \delta\boldsymbol{\phi}]^{\mathrm{T}}} = \begin{bmatrix}\boldsymbol{I} & -(\boldsymbol{R}\boldsymbol{p}+\boldsymbol{t})^{\wedge}\\ \boldsymbol{0}^{\mathrm{T}} & \boldsymbol{0}^{\mathrm{T}}\end{bmatrix} \triangleq (\boldsymbol{T}\boldsymbol{p})^{\odot}.\end{aligned}$$

$\odot$† 4×6 $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{x}, \boldsymbol{y}$

$$\frac{\mathrm{d}\begin{bmatrix}\boldsymbol{a}\\ \boldsymbol{b}\end{bmatrix}}{\mathrm{d}\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{bmatrix}} = \left(\frac{\mathrm{d}[\boldsymbol{a}, \boldsymbol{b}]^{\mathrm{T}}}{\mathrm{d}\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}\end{bmatrix}}\right)^{\mathrm{T}} = \begin{bmatrix}\frac{\mathrm{d}\boldsymbol{a}}{\mathrm{d}\boldsymbol{x}} & \frac{\mathrm{d}\boldsymbol{b}}{\mathrm{d}\boldsymbol{x}}\\ \frac{\mathrm{d}\boldsymbol{a}}{\mathrm{d}\boldsymbol{y}} & \frac{\mathrm{d}\boldsymbol{b}}{\mathrm{d}\boldsymbol{y}}\end{bmatrix}^{\mathrm{T}} = \begin{bmatrix}\frac{\mathrm{d}\boldsymbol{a}}{\mathrm{d}\boldsymbol{x}} & \frac{\mathrm{d}\boldsymbol{a}}{\mathrm{d}\boldsymbol{y}}\\ \frac{\mathrm{d}\boldsymbol{b}}{\mathrm{d}\boldsymbol{x}} & \frac{\mathrm{d}\boldsymbol{b}}{\mathrm{d}\boldsymbol{y}}\end{bmatrix} \tag{4.43}$$

---

* $\boldsymbol{p}$

† “ ”

4.4 Sophus

4.4.1 Sophus

3 Eigen Strasdat Sophus * Sophus S
Sophus SO(3) SE(3)

Listing 4.1: slambook/ch4/useSophus.cpp

```
#include <iostream>
#include <cmath>
#include <Eigen/Core>
#include <Eigen/Geometry>
#include "sophus/se3.hpp"

using namespace std;
using namespace Eigen;

/// □□□□□□□□□□□sophus
int main(int argc, char **argv) {
  // □□□□□□□□□□Z90
  Matrix3d R = AngleAxisd(M_PI / 2, Vector3d(0, 0, 1)).toRotationMatrix();
  // □□□□□
  Quaterniond q(R);
  Sophus::SO3d SO3_R(R);              // Sophus::□□□□□□□□□□□□SO3d
  Sophus::SO3d SO3_q(q);              // □□□□□□□□□□□
  // □□□□□□
  cout << "SO(3) from matrix:\n" << SO3_R.matrix() << endl;
  cout << "SO(3) from quaternion:\n" << SO3_q.matrix() << endl;
  cout << "they are equal" << endl;

  // □□□□□□□□□□□□□□
  Vector3d so3 = SO3_R.log();
  cout << "so3 = " << so3.transpose() << endl;
  // hat □□□□□□□□□
  cout << "so3 hat=\n" << Sophus::SO3d::hat(so3) << endl;
  // □□□□□□□□□□□vee
  cout << "so3 hat vee= " << Sophus::SO3d::vee(Sophus::SO3d::hat(so3)).transpose() <<
       endl;

  // □□□□□□□□□□
  Vector3d update_so3(1e-4, 0, 0); //□□□□□□□□□
  Sophus::SO3d SO3_updated = Sophus::SO3d::exp(update_so3) * SO3_R;
  cout << "SO3 updated = \n" << SO3_updated.matrix() << endl;

  cout << "*******************************" << endl;
  // □SE(3)□□□□□□
  Vector3d t(1, 0, 0);           // □□□□X1
  Sophus::SE3d SE3_Rt(R, t);           // □R,□□tSE(3)
  Sophus::SE3d SE3_qt(q, t);            // □q,□□tSE(3)
  cout << "SE3 from R,t= \n" << SE3_Rt.matrix() << endl;
  cout << "SE3 from q,t= \n" << SE3_qt.matrix() << endl;
  // □□□se(3) □□□□□□□□□□□□□□□□□typedef
  typedef Eigen::Matrix<double, 6, 1> Vector6d;
  Vector6d se3 = SE3_Rt.log();
  cout << "se3 = " << se3.transpose() << endl;
  // □□□□□□□□□□□Sophusse(3)□□□□□□□□□□.
  // □□□□□□□□□□hatvee
  cout << "se3 hat = \n" << Sophus::SE3d::hat(se3) << endl;
  cout << "se3 hat vee = " << Sophus::SE3d::vee(Sophus::SE3d::hat(se3)).transpose() <<
       endl;

  // □□□□□□□□□□
  Vector6d update_se3; //□□□
  update_se3.setZero();
```

* Sophus Lie

```
  update_se3(0, 0) = 1e-4d;
  Sophus::SE3d SE3_updated = Sophus::SE3d::exp(update_se3) * SE3_Rt;
  cout << "SE3 updated = " << endl << SE3_updated.matrix() << endl;

  return 0;
}
```

SO(3) SE(3) SO(3), SE(3)

Listing 4.2: slambook/ch4/useSophus/CMakeLists.txt

```
# □□□□□□□□ sophus find_package □□□□□
find_package( Sophus REQUIRED )
include_directories( ${Sophus_INCLUDE_DIRS} )

add_executable( useSophus useSophus.cpp )
```

find_package cmake cmake Sophus Sophus_INCLUDE_DIRS Sophus Eigen Sophus cmake

4.4.2

$\boldsymbol{T}_{\text{esti},i}$ $\boldsymbol{T}_{\text{gt},i}$ $i = 1, \cdots, N$

Absolute Trajectory Error, ATE

$$\text{ATE}_{\text{all}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \| \log(\boldsymbol{T}_{\text{gt},i}^{-1}\boldsymbol{T}_{\text{esti},i})^{\vee} \|_2^2}, \tag{4.44}$$

Root-Mean-Squared Error, RMSE [23] Average Translational Error

$$\text{ATE}_{\text{trans}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \| \text{trans}(\boldsymbol{T}_{\text{gt},i}^{-1}\boldsymbol{T}_{\text{esti},i}) \|_2^2}, \tag{4.45}$$

trans

i $i+\Delta t$ Relative Pose Error, RPE

$$\text{RPE}_{\text{all}} = \sqrt{\frac{1}{N-\Delta t}\sum_{i=1}^{N-\Delta t} \| \log\left(\left(\boldsymbol{T}_{\text{gt},i}^{-1}\boldsymbol{T}_{\text{gt},i+\Delta t} \right)^{-1} \left(\boldsymbol{T}_{\text{esti},i}^{-1}\boldsymbol{T}_{\text{esti},i+\Delta t} \right) \right)^{\vee} \|_2^2}, \tag{4.46}$$

$$\text{RPE}_{\text{trans}} = \sqrt{\frac{1}{N-\Delta t}\sum_{i=1}^{N-\Delta t} \| \text{trans}\left(\left(\boldsymbol{T}_{\text{gt},i}^{-1}\boldsymbol{T}_{\text{gt},i+\Delta t} \right)^{-1} \left(\boldsymbol{T}_{\text{esti},i}^{-1}\boldsymbol{T}_{\text{esti},i+\Delta t} \right) \right) \|_2^2}. \tag{4.47}$$

Sophus groundtruth.txt estimated.txt 3D

Listing 4.3: slambook/ch4/example/trajectoryError.cpp

```
#include <iostream>
#include <fstream>
#include <unistd.h>
#include <pangolin/pangolin.h>
#include <sophus/se3.hpp>

```

```
using namespace Sophus;
using namespace std;

string groundtruth_file = "./example/groundtruth.txt";
string estimated_file = "./example/estimated.txt";

typedef vector<Sophus::SE3d, Eigen::aligned_allocator<Sophus::SE3d>> TrajectoryType;

void DrawTrajectory(const TrajectoryType &gt, const TrajectoryType &esti);

TrajectoryType ReadTrajectory(const string &path);

int main(int argc, char **argv) {
  TrajectoryType groundtruth = ReadTrajectory(groundtruth_file);
  TrajectoryType estimated = ReadTrajectory(estimated_file);
  assert(!groundtruth.empty() && !estimated.empty());
  assert(groundtruth.size() == estimated.size());

  // compute rmse
  double rmse = 0;
  for (size_t i = 0; i < estimated.size(); i++) {
    Sophus::SE3d p1 = estimated[i], p2 = groundtruth[i];
    double error = (p2.inverse() * p1).log().norm();
    rmse += error * error;
  }
  rmse = rmse / double(estimated.size());
  rmse = sqrt(rmse);
  cout << "RMSE = " << rmse << endl;

  DrawTrajectory(groundtruth, estimated);
  return 0;
}

TrajectoryType ReadTrajectory(const string &path) {
  ifstream fin(path);
  TrajectoryType trajectory;
  if (!fin) {
    cerr << "trajectory " << path << " not found." << endl;
    return trajectory;
  }

  while (!fin.eof()) {
    double time, tx, ty, tz, qx, qy, qz, qw;
    fin >> time >> tx >> ty >> tz >> qx >> qy >> qz >> qw;
    Sophus::SE3d p1(Eigen::Quaterniond(qx, qy, qz, qw), Eigen::Vector3d(tx, ty, tz));
    trajectory.push_back(p1);
  }
  return trajectory;
}
```

2.207 Figure 4-2

4.5 *

Sim(3) $\mathfrak{sim}(3)$ SLAM RGB-D SLAM
SLAM SE(3) SLAM SE(3) $\boldsymbol{p}$

$$\boldsymbol{p}' = \begin{bmatrix} s\boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix} \boldsymbol{p} = s\boldsymbol{R}\boldsymbol{p} + \boldsymbol{t}. \tag{4.48}$$

s $\boldsymbol{p}$ 3 $\boldsymbol{p}$ SO(3) SE(3) Sim(3)

$$\mathrm{Sim}(3) = \left\{ \boldsymbol{S} = \begin{bmatrix} s\boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \right\}. \tag{4.49}$$

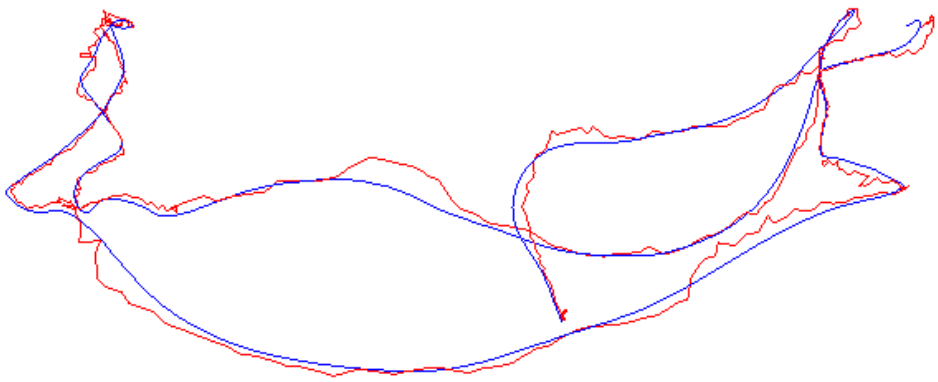


Figure 4-2:

Sim(3) $\mathfrak{sim}(3)$ 7 $\boldsymbol{\zeta}$ 6 $\mathfrak{se}(3)$ σ

$$\mathfrak{sim}(3) = \left\{ \boldsymbol{\zeta} | \boldsymbol{\zeta} = \begin{bmatrix} \boldsymbol{\rho} \\ \boldsymbol{\phi} \\ \sigma \end{bmatrix} \in \mathbb{R}^7, \boldsymbol{\zeta}^\wedge = \begin{bmatrix} \sigma \boldsymbol{I} + \boldsymbol{\phi}^\wedge & \boldsymbol{\rho} \\ \boldsymbol{0}^{\mathrm{T}} & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4} \right\}. \tag{4.50}$$

$\mathfrak{se}(3)$ σ Sim(3) $\mathfrak{sim}(3)$

$$\exp\left(\boldsymbol{\zeta}^\wedge\right) = \begin{bmatrix} \mathrm{e}^\sigma \exp\left(\boldsymbol{\phi}^\wedge\right) & \boldsymbol{J}_s \boldsymbol{\rho} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix}. \tag{4.51}$$

$\boldsymbol{J}_s$

$$\begin{aligned} \boldsymbol{J}_s =& \frac{\mathrm{e}^\sigma - 1}{\sigma} \boldsymbol{I} + \frac{\sigma \mathrm{e}^\sigma \sin\theta + (1 - \mathrm{e}^\sigma \cos\theta)\theta}{\sigma^2 + \theta^2} \boldsymbol{a}^\wedge \\ &+ \left(\frac{\mathrm{e}^\sigma - 1}{\sigma} - \frac{(\mathrm{e}^\sigma \cos\theta - 1)\sigma + (\mathrm{e}^\sigma \sin\theta)\theta}{\sigma^2 + \theta^2} \right) \boldsymbol{a}^\wedge \boldsymbol{a}^\wedge. \end{aligned}$$

$\boldsymbol{\zeta}$

$$s = \mathrm{e}^\sigma, \ \boldsymbol{R} = \exp(\boldsymbol{\phi}^\wedge), \ \boldsymbol{t} = \boldsymbol{J}_s \boldsymbol{\rho}. \tag{4.52}$$

SO(3) $\mathfrak{se}(3)$ $\boldsymbol{\mathcal{J}}$ s σ
Sim(3) BCH SE(3) $\boldsymbol{p}$ $\boldsymbol{Sp}$ $\boldsymbol{S}$ $\boldsymbol{Sp}$ $\exp(\boldsymbol{\zeta}^\wedge)$ $\boldsymbol{S}$
7 $\boldsymbol{Sp}$ 3 $\boldsymbol{q}$

$$\frac{\partial \boldsymbol{Sp}}{\partial \boldsymbol{\zeta}} = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{q}^\wedge & \boldsymbol{q} \\ \boldsymbol{0}^{\mathrm{T}} & \boldsymbol{0}^{\mathrm{T}} & 0 \end{bmatrix}. \tag{4.53}$$

Sim(3) Sim(3) [24]

4.6

SO(3) SE(3) $\mathfrak{so}(3)$ $\mathfrak{se}(3)$ BCH

SO(3) SE(3)

1. SO(3) SE(3) Sim(3)
2. $(\mathbb{R}^3, \mathbb{R}, \times)$
3. $\mathfrak{so}(3)$ $\mathfrak{se}(3)$
4. 4.20 4.21
5. $$\boldsymbol{R}\boldsymbol{p}^{\wedge}\boldsymbol{R}^{\mathrm{T}} = (\boldsymbol{R}\boldsymbol{p})^{\wedge}.$$
6. $$\boldsymbol{R}\exp(\boldsymbol{p}^{\wedge})\boldsymbol{R}^{\mathrm{T}} = \exp((\boldsymbol{R}\boldsymbol{p})^{\wedge}).$$

   SO(3) SE(3)

   $$\boldsymbol{T}\exp(\boldsymbol{\xi}^{\wedge})\boldsymbol{T}^{-1} = \exp\left((\mathrm{Ad}(\boldsymbol{T})\boldsymbol{\xi})^{\wedge}\right), \tag{4.54}$$

   $$\mathrm{Ad}(\boldsymbol{T}) = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t}^{\wedge}\boldsymbol{R} \\ \boldsymbol{0} & \boldsymbol{R} \end{bmatrix}. \tag{4.55}$$

7. SO(3) SE(3)
8. cmake find_package cmake

Chapter 5

1.
2.
3. OpenCV
4.

“ ” SLAM “ ” SLAM

D

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K(Rp+t)$$

5.1

Intrinsics

5.1.1

Figure

1

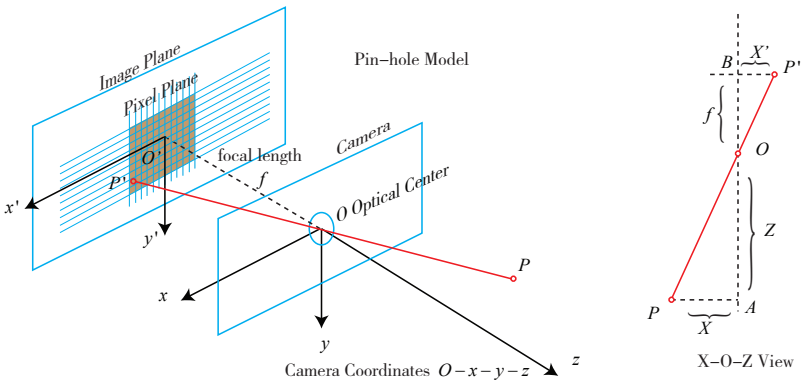


Figure 5-1:

$O{-}x{-}y{-}z$ z x y O P O $O'-$
$x'-y'$ P' P $[X,Y,Z]^{\mathrm{T}}$ P' $[X',Y',Z']^{\mathrm{T}}$ f

$$\frac{Z}{f} = -\frac{X}{X'} = -\frac{Y}{Y'}. \tag{5.1}$$

Figure 5-2

$$\frac{Z}{f} = \frac{X}{X'} = \frac{Y}{Y'}. \tag{5.2}$$

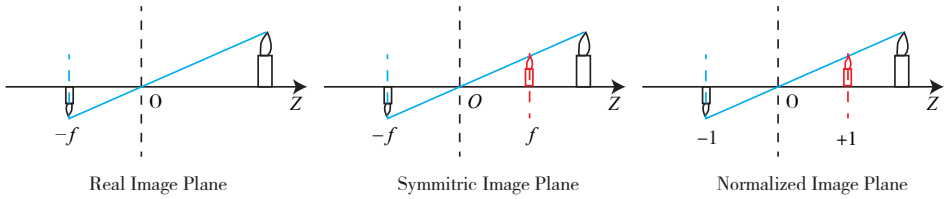


Figure 5-2:

X', Y'

$$\begin{aligned} X' &= f\frac{X}{Z} \\ Y' &= f\frac{Y}{Z} \end{aligned}. \tag{5.3}$$

(5.3) P 0.2 X' 0.14
$u - v$ P' $[u, v]^{\mathrm{T}}$
* o' u x v y u α v β $[c_x, c_y]^{\mathrm{T}}$ P' $[u,$

$$\begin{cases} u = \alpha X' + c_x \\ v = \beta Y' + c_y \end{cases}. \tag{5.4}$$

(5.3) αf f_x βf f_y

$$\begin{cases} u = f_x \frac{X}{Z} + c_x \\ v = f_y \frac{Y}{Z} + c_y \end{cases}. \tag{5.5}$$

f α, β / f_x, f_y c_x, c_y

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \frac{1}{Z} \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \triangleq \frac{1}{Z}\boldsymbol{KP}. \tag{5.6}$$

Z

$$Z \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \triangleq \boldsymbol{KP}. \tag{5.7}$$

Camera Intrinsics $\boldsymbol{K}$

* 2

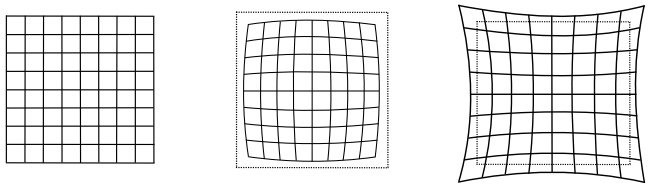


Figure 5-3:

(5.6) P P $\boldsymbol{P}_w$ $\boldsymbol{R}$ $\boldsymbol{t}$

$$Z\boldsymbol{P}_{uv} = Z\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{K}\left(\boldsymbol{R}\boldsymbol{P}_w + \boldsymbol{t}\right) = \boldsymbol{K}\boldsymbol{T}\boldsymbol{P}_w. \tag{5.8}$$

* P $\boldsymbol{R}, \boldsymbol{t}$ Camera Extrinsics † SLAM (5.8) P

$$\left(\boldsymbol{R}\boldsymbol{P}_w + \boldsymbol{t}\right) = \underbrace{[X, Y, Z]^{\mathrm{T}}} \rightarrow \underbrace{[X/Z, Y/Z, 1]^{\mathrm{T}}}. \tag{5.9}$$

‡$z = 1$ $z = 1$ $[u, v]^{\mathrm{T}}$

5.1.2

Distortion §

3

Figure 5-4

$\boldsymbol{p}$ $[x, y]^{\mathrm{T}}$ $[r, \theta]^{\mathrm{T}}$ r $\boldsymbol{p}$ θ

$$\begin{aligned} x_{\mathrm{distorted}} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{\mathrm{distorted}} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned}. \tag{5.10}$$

$[x_{\mathrm{distorted}}, y_{\mathrm{distorted}}]^{\mathrm{T}}$ p_1, p_2

$$\begin{aligned} x_{\mathrm{distorted}} &= x + 2p_1 xy + p_2(r^2 + 2x^2) \\ y_{\mathrm{distorted}} &= y + p_1(r^2 + 2y^2) + 2p_2 xy \end{aligned}. \tag{5.11}$$

(5.10) (5.11) $\boldsymbol{P}$ 5

1. $[x, y]^{\mathrm{T}}$

---

* $\boldsymbol{TP}$ $\boldsymbol{K}$

† “ ”

‡ Z Z

§

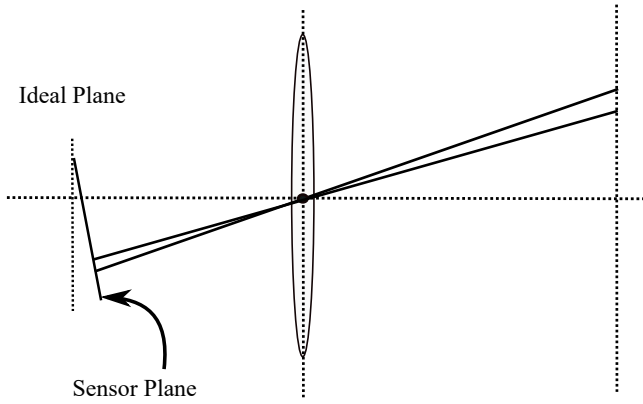


Figure 5-4:

2.

$$\begin{cases} x_{\text{distorted}} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 xy + p_2(r^2 + 2x^2) \\ y_{\text{distorted}} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y^2) + 2p_2 xy \end{cases}. \tag{5.12}$$

3.

$$\begin{cases} u = f_x x_{\text{distorted}} + c_x \\ v = f_y y_{\text{distorted}} + c_y \end{cases}. \tag{5.13}$$

5 k_1, p_1, p_2 3

SLAM

Undistort SLAM

1. P $\boldsymbol{P}_w$
2. $\boldsymbol{R}, \boldsymbol{t}$ $\boldsymbol{T} \in \mathrm{SE}(3)$ P $\tilde{\boldsymbol{P}}_c = \boldsymbol{R}\boldsymbol{P}_w + \boldsymbol{t}$
3. $\tilde{\boldsymbol{P}}_c$ X, Y, Z $Z = 1$ P $\boldsymbol{P}_c = [X/Z, Y/Z, 1]^{\mathrm{T}}$*
4. $\boldsymbol{P}_c$
5. P $\boldsymbol{P}_{uv} = \boldsymbol{K}\boldsymbol{P}_c$

5.1.3

P RGB-D Figure 5-5

Figure 5-6

† x Baseline b

* Z 1
†

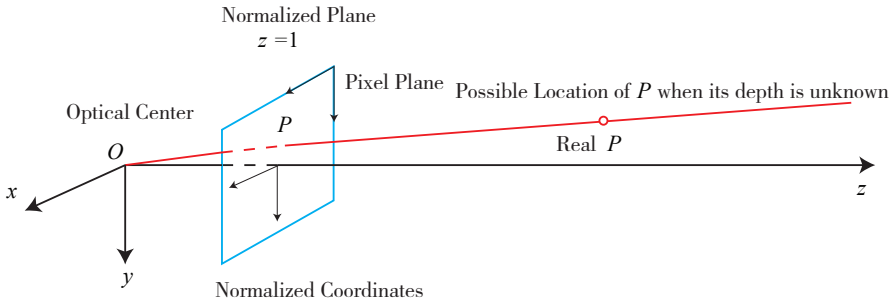


Figure 5-5:

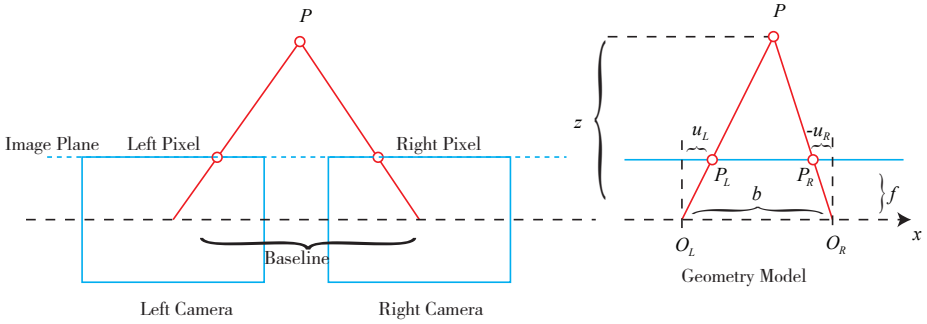


Figure 5-6: O_L, O_R f u_L u_R u_R $-u_R$

P P_L, P_R x P x u u_L u_R Figure 6 $\triangle PP_LP_R$ $\triangle PO_LO_R$

$$\frac{z-f}{z} = \frac{b-u_L+u_R}{b}. \tag{5.14}$$

$$z = \frac{fb}{d}, \quad d \triangleq u_L - u_R. \tag{5.15}$$

d Disparity * fb d “ ”

5.1.4 RGB-D

RGB-D “ ” RGB-D Figure 5-7

1. Structured Light Kinect 1 Project Tango 1 Intel RealSense
2. Time-of-flight ToF Kinect 2 ToF

*

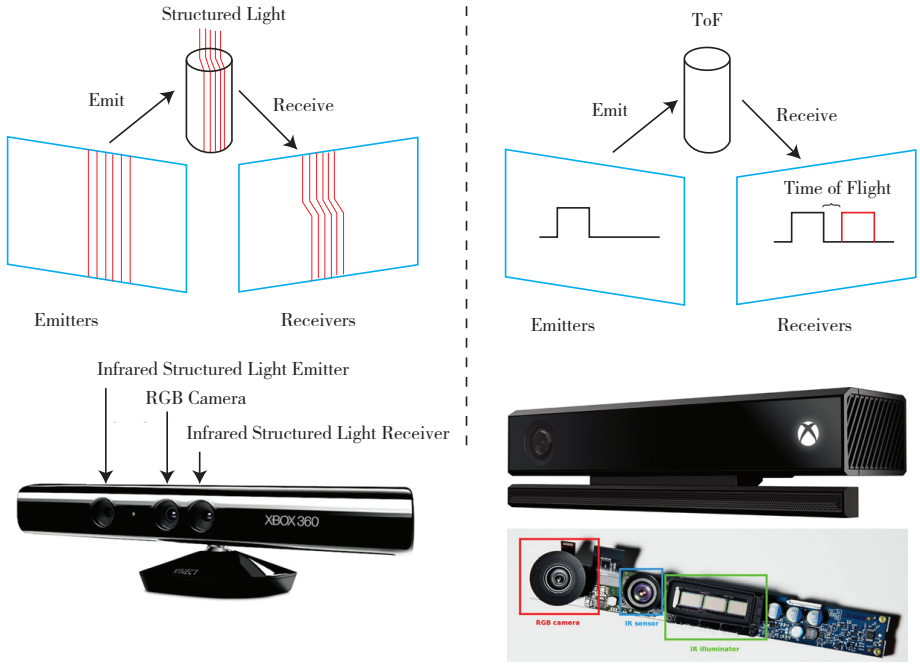


Figure 5-7: RGB-D

RGB-D ToF ToF
D RGB-D
RGB-D 3D Point Cloud RGB-
D RGB-D
RGB-D — RGB-D RGB-
D RGB-D

5.2

OpenCV —— (x, y) I w h

$$I(x, y) : \mathbb{R}^2 \mapsto \mathbb{R}.$$

(x, y) x, y 0 0~255 unsigned char 1 640

Listing 5.1:

```
unsigned char image[480][640];
```

480×640

Figure 5-8 X Y u, v —
—Z Z X Y
x, y

Listing 5.2:

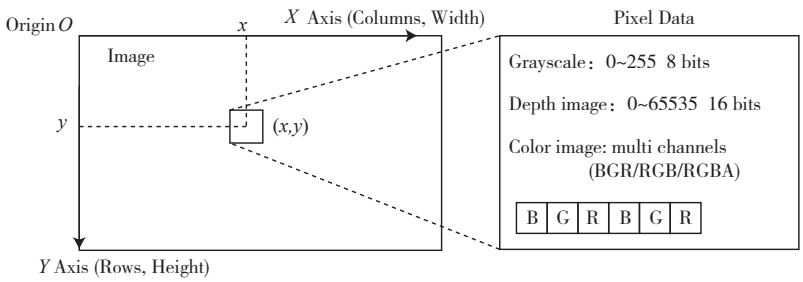


Figure 5-8:

```
unsigned char pixel = image[y][x];
```

$I(x,y)$ x y x, y
8 0~255 RGB-D RGB-D 255
short 0~65535 65 RGB-D
channel R G B 8 2
OpenCV B G R 24 8 8 8 R G B

5.3

5.3.1 OpenCV

OpenCV

OpenCV

OpenCV* OpenCV Ubuntu

1. OpenCV OpenCV
2. Ubuntu Ubuntu

OpenCV GPU OpenCV OpenCV
2.4 OpenCV 3 OpenCV **3**
OpenCV 3rdparty `http://opencv.org/downloads.html` OpenCV
for Linux opencv-3.1.0.zip OpenCV cmake
OpenCV

Listing 5.3:

```
sudo apt-get install build-essential libgtk2.0-dev libvtk5-dev libjpeg-dev libtiff4-
    dev libjasper-dev libopenexr-dev libtbb-dev
```

OpenCV OpenCV cmake GPU OpenCV GP
cmake make sudo make install OpenCV CPU “m
-j4” -j OpenCV /usr/local OpenCV OpenCV
2 OpenCV 3

* `http://opencv.org`

OpenCV

OpenCV

Listing 5.4: slambook/ch5/imageBasics/imageBasics.cpp

```
#include <iostream>
#include <chrono>

using namespace std;

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

int main(int argc, char **argv) {
  // argv[1]
  cv::Mat image;
  image = cv::imread(argv[1]); //cv::imread

  // 
  if (image.data == nullptr) { //,
    cerr << "" << argv[1] << "." << endl;
    return 0;
  }

  // , 
  cout << "" << image.cols << "," << image.rows
       << "," << image.channels() << endl;
  cv::imshow("image", image);      // cv::imshow
  cv::waitKey(0);                  // ,

  // image
  if (image.type() != CV_8UC1 && image.type() != CV_8UC3) {
    // 
    cout << "." << endl;
    return 0;
  }

  // , 
  // std::chrono 
  chrono::steady_clock::time_point t1 = chrono::steady_clock::now();
  for (size_t y = 0; y < image.rows; y++) {
    // cv::Mat::ptr
    unsigned char *row_ptr = image.ptr<unsigned char>(y);  // row_ptry
    for (size_t x = 0; x < image.cols; x++) {
      // x,y 
      unsigned char *data_ptr = &row_ptr[x * image.channels()]; // data_ptr 
      // ,
      for (int c = 0; c != image.channels(); c++) {
        unsigned char data = data_ptr[c]; // dataI(x,y)c
      }
    }
  }
  chrono::steady_clock::time_point t2 = chrono::steady_clock::now();
  chrono::duration<double> time_used = chrono::duration_cast < chrono::duration <
       double >> (t2 - t1);
  cout << "" << time_used.count() << " " << endl;

  // cv::Mat 
  // 
  cv::Mat image_another = image;
  // image_another  image 
  image_another(cv::Rect(0, 0, 100, 100)).setTo(0); // 100*100
  cv::imshow("image", image);
  cv::waitKey(0);

  // clone
  cv::Mat image_clone = image.clone();
  image_clone(cv::Rect(0, 0, 100, 100)).setTo(255);
  cv::imshow("image", image);
```

```
64   cv::imshow("image_clone", image_clone);
65   cv::waitKey(0);
66
67   // ,,,,,OpenCV.
68   cv::destroyAllWindows();
69   return 0;
70 }
```

CMakeLists.txt OpenCV C++ 11 nullptr chrono

Listing 5.5: slambook/ch5/imageBasics/CMakeLists.txt

```
1  # C++ 11
2  set( CMAKE_CXX_FLAGS "-std=c++11" )
3
4  # OpenCV
5  find_package( OpenCV REQUIRED )
6  #
7  include_directories( ${OpenCV_INCLUDE_DIRS} )
8
9  add_executable( imageBasics imageBasics.cpp )
10 # OpenCV
11 target_link_libraries( imageBasics ${OpenCV_LIBS} )
```

1. argv[1] ubuntu.png Ubuntu

Listing 5.6:

```
1 build/imageBasics ubuntu.png
```

IDE

2. 10~18 cv::imread

3. 35~47 OpenCV cv::Mat::data 12.74ms cmake debug release

4. OpenCV OpenCV

OpenCV OpenCV cv::Mat

5.3.2

OpenCV cv::Undistort()

Listing 5.7: slambook/ch5/imageBasics/undistortImage.cpp

```
1  #include <opencv2/opencv.hpp>
2  #include <string>
3  using namespace std;
4  string image_file = "./distorted.png";   //
5
6  int main(int argc, char **argv) {
7    // OpenCV
8    //
9    double k1 = -0.28340811, k2 = 0.07395907, p1 = 0.00019359, p2 = 1.76187114e-05;
10   //
11   double fx = 458.654, fy = 457.296, cx = 367.215, cy = 248.375;
12
13   cv::Mat image = cv::imread(image_file, 0);   // CV_8UC1
14   int rows = image.rows, cols = image.cols;
```

```
  cv::Mat image_undistort = cv::Mat(rows, cols, CV_8UC1);   // 
  // 
  for (int v = 0; v < rows; v++) {
    for (int u = 0; u < cols; u++) {
      // (u,v)(u_distorted, v_distorted)
      double x = (u - cx) / fx, y = (v - cy) / fy;
      double r = sqrt(x * x + y * y);
      double x_distorted = x * (1 + k1 * r * r + k2 * r * r * r * r) + 2 * p1 * x * y
          + p2 * (r * r + 2 * x * x);
      double y_distorted = y * (1 + k1 * r * r + k2 * r * r * r * r) + p1 * (r * r + 2
           * y * y) + 2 * p2 * x * y;
      double u_distorted = fx * x_distorted + cx;
      double v_distorted = fy * y_distorted + cy;

      // 
      if (u_distorted >= 0 && v_distorted >= 0 && u_distorted < cols && v_distorted <
          rows) {
        image_undistort.at<uchar>(v, u) = image.at<uchar>((int) v_distorted, (int)
            u_distorted);
      } else {
        image_undistort.at<uchar>(v, u) = 0;
      }
    }
  }

  // 
  cv::imshow("distorted", image);
  cv::imshow("undistorted", image_undistort);
  cv::waitKey();
  return 0;
}
```

5.4 3D

5.4.1

stereo Figure 5-9

Listing 5.8: slambook/ch5/stereoVision/stereoVision.cpp

```
int main(int argc, char **argv) {
    // 
    double fx = 718.856, fy = 718.856, cx = 607.1928, cy = 185.2157;
    // 
    double b = 0.573;

    // 
    cv::Mat left = cv::imread(left_file, 0);
    cv::Mat right = cv::imread(right_file, 0);
    cv::Ptr<cv::StereoSGBM> sgbm = cv::StereoSGBM::create(
        0, 96, 9, 8 * 9 * 9, 32 * 9 * 9, 1, 63, 10, 100, 32);    // 
    cv::Mat disparity_sgbm, disparity;
    sgbm->compute(left, right, disparity_sgbm);
    disparity_sgbm.convertTo(disparity, CV_32F, 1.0 / 16.0f);

    // 
    vector<Vector4d, Eigen::aligned_allocator<Vector4d>> pointcloud;

    // v++u++v+=2, u+=2
    for (int v = 0; v < left.rows; v++)
        for (int u = 0; u < left.cols; u++) {
            if (disparity.at<float>(v, u) <= 10.0 || disparity.at<float>(v, u) >=
                96.0) continue;

            Vector4d point(0, 0, 0, left.at<uchar>(v, u) / 255.0); // xyz,

```

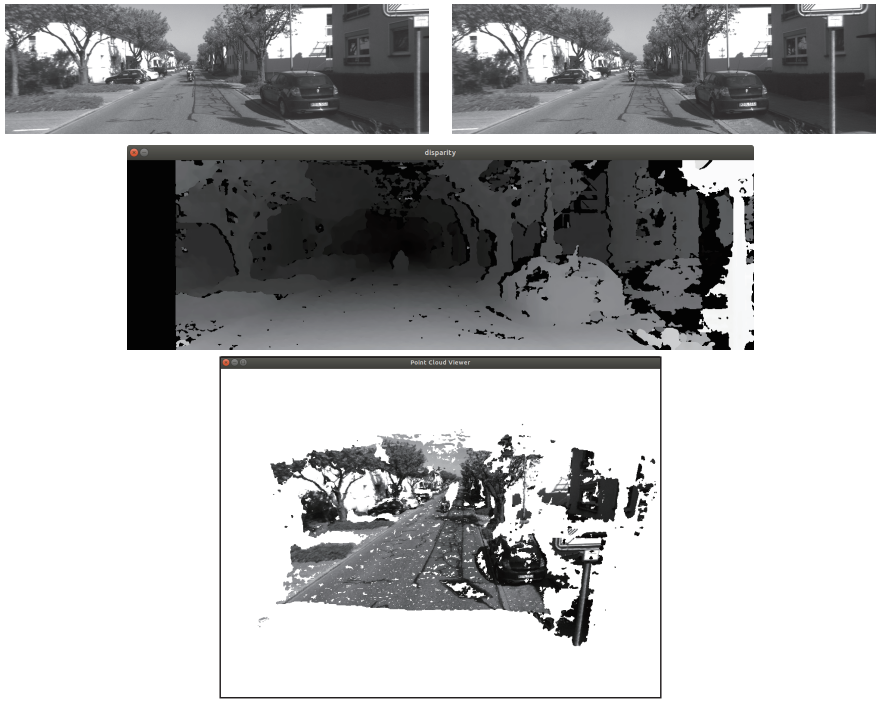


Figure 5-9: SGBM

```
26              // 
27              double x = (u − cx) / fx;
28              double y = (v − cy) / fy;
29              double depth = fx * b / (disparity.at<float>(v, u));
30              point[0] = x * depth;
31              point[1] = y * depth;
32              point[2] = depth;
33
34              pointcloud.push_back(point);
35      }
36
37      cv::imshow("disparity", disparity / 96.0);
38      cv::waitKey(0);
39      // 
40      showPointCloud(pointcloud);
41      return 0;
42  }
```

OpenCV SGBM Semi-global Batch Matching [26] 3D SGBM [27, 28] OpenCV

5.4.2 RGB-D

RGB-D RGB-D

5 slambook/ch5/rgbd color/ 1.png 5.png 5 RGB depth/ 5 pose.txt 5 $\boldsymbol{T}$

$$[x, y, z, q_x, q_y, q_z, q_w],$$

q_w

$$[-0.228993, 0.00645704, 0.0287837, -0.0004327, -0.113131, -0.0326832, 0.993042].$$

(1). RGB-D (2).

Listing 5.9: slambook/ch5/rgbd/jointMap.cpp

```
int main(int argc, char **argv) {
    vector<cv::Mat> colorImgs, depthImgs;    // □□□□□□□□
    TrajectoryType poses;         // □□□□

    ifstream fin("./pose.txt");
    if (!fin) {
        cerr << "□□□pose.□□□□□□□□□txt" << endl;
        return 1;
    }

    for (int i = 0; i < 5; i++) {
        boost::format fmt("./%s/%d.%s"); //□□□□□□
        colorImgs.push_back(cv::imread((fmt % "color" % (i + 1) % "png").str()));
        depthImgs.push_back(cv::imread((fmt % "depth" % (i + 1) % "pgm").str(), -1));
             // □□□□□□□□-1

        double data[7] = {0};
        for (auto &d:data) fin >> d;
        Sophus::SE3d pose(Eigen::Quaterniond(data[6], data[3], data[4], data[5]),
        Eigen::Vector3d(data[0], data[1], data[2]));
        poses.push_back(pose);
    }

    // □□□□□□□
    // □□□□
    double cx = 325.5;
    double cy = 253.5;
    double fx = 518.0;
    double fy = 519.0;
    double depthScale = 1000.0;
    vector<Vector6d, Eigen::aligned_allocator<Vector6d>> pointcloud;
    pointcloud.reserve(1000000);

    for (int i = 0; i < 5; i++) {
        cout << "□□□□□: " << i + 1 << endl;
        cv::Mat color = colorImgs[i];
        cv::Mat depth = depthImgs[i];
        Sophus::SE3d T = poses[i];
        for (int v = 0; v < color.rows; v++)
            for (int u = 0; u < color.cols; u++) {
                unsigned int d = depth.ptr<unsigned short>(v)[u]; // □□□
                if (d == 0) continue; // □□□□□□□□0
                Eigen::Vector3d point;
                point[2] = double(d) / depthScale;
                point[0] = (u - cx) * point[2] / fx;
                point[1] = (v - cy) * point[2] / fy;
                Eigen::Vector3d pointWorld = T * point;

                Vector6d p;
                p.head<3>() = pointWorld;
                p[5] = color.data[v * color.step + u * color.channels()];   // blue
                p[4] = color.data[v * color.step + u * color.channels() + 1]; // green
                p[3] = color.data[v * color.step + u * color.channels() + 2]; // red
                pointcloud.push_back(p);
            }
    }

    cout << "□□□□" << pointcloud.size() << "□□." << endl;
    showPointCloud(pointcloud);
    return 0;
}
```

Pangolin Figure 5-10

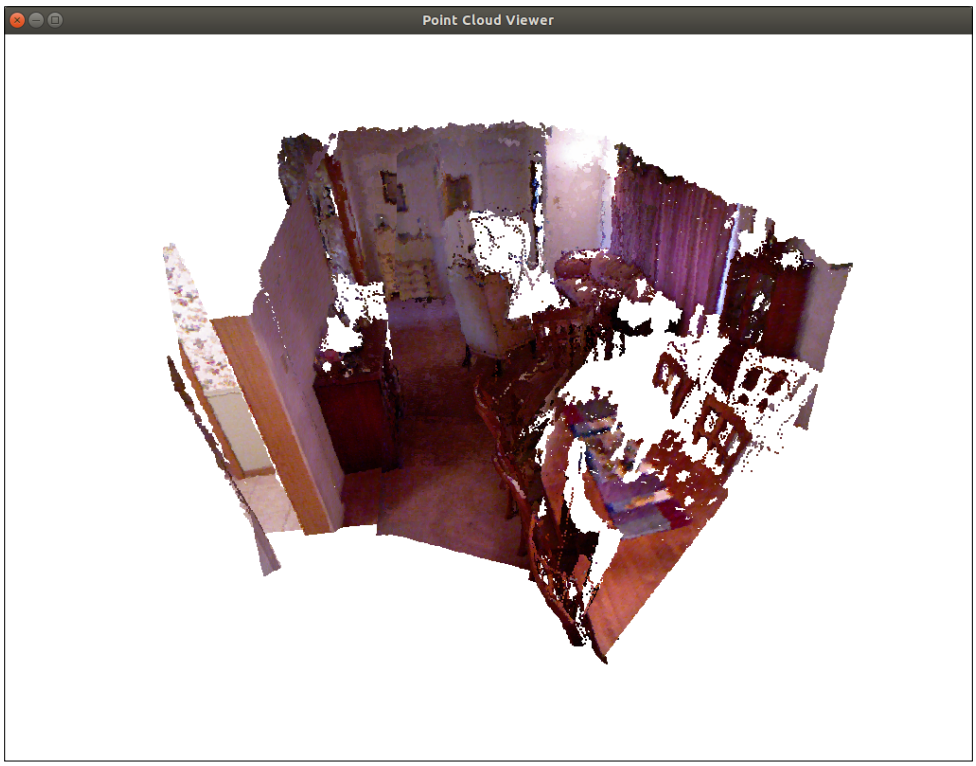


Figure 5-10:

1.*

2.

3.

4. global shutter rolling shutter SLAM

5. RGB-D Kinect `https://github.com/code-iai/iai_kinect2`

6.

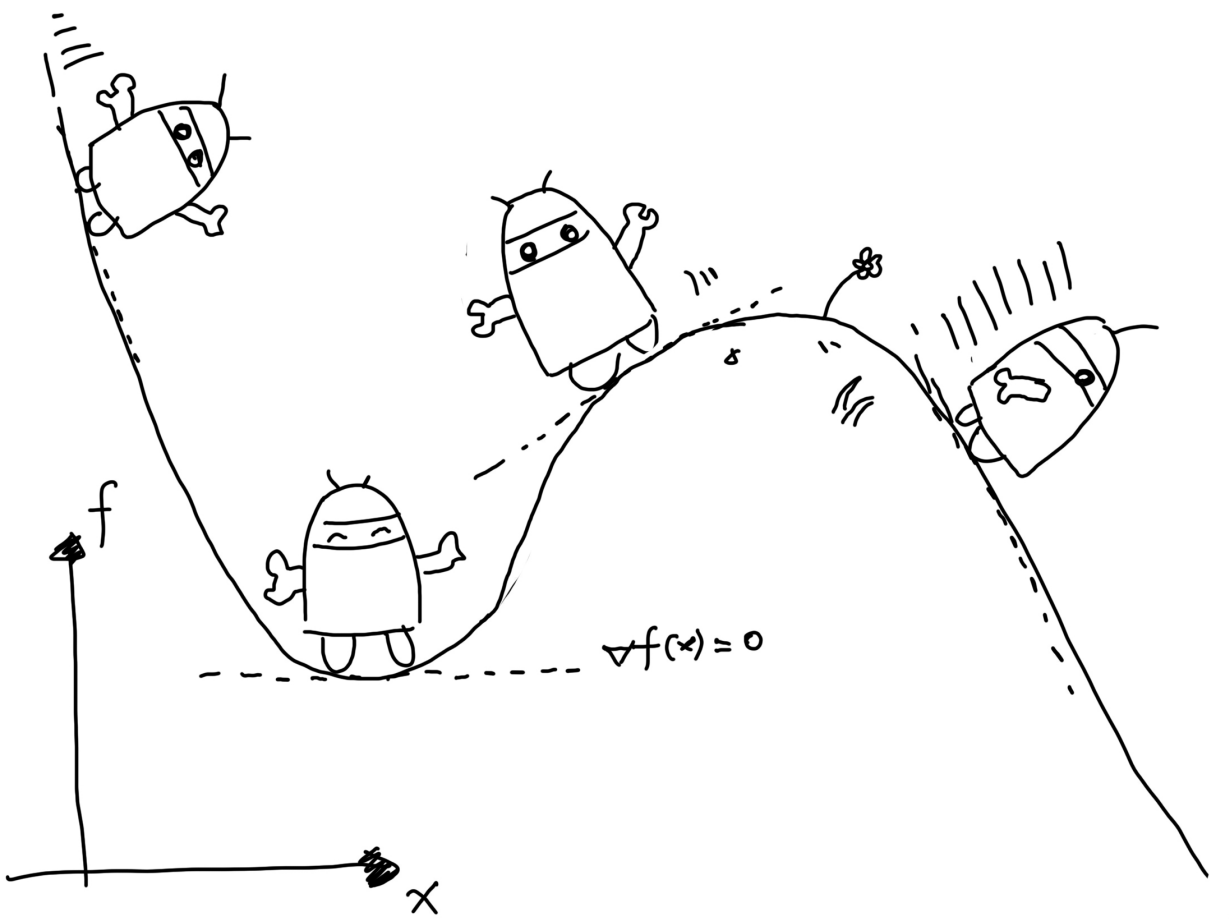
7.* OpenCV

Chapter 6

1.
2. Gauss-Newton — Levenburg-Marquadt
3. Ceres g2o

SLAM SLAM

g2o Ceres



$$f(x+\Delta x) \approx f(x) + \nabla f(x)\Delta x + \frac{1}{2}\Delta x^T H(x)\Delta x + \cdots$$

6.1

6.1.1

2 SLAM (2.5)

$$\begin{cases} \boldsymbol{x}_k = f\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k\right) + \boldsymbol{w}_k \\ \boldsymbol{z}_{k,j} = h\left(\boldsymbol{y}_j, \boldsymbol{x}_k\right) + \boldsymbol{v}_{k,j} \end{cases}. \tag{6.1}$$

4 $\boldsymbol{x}_k$ SE(3) 5 $\boldsymbol{x}_k$ $\boldsymbol{T}_k \in \mathrm{SE}(3)$

$$s\boldsymbol{z}_{k,j} = \boldsymbol{K}(\boldsymbol{R}_k\boldsymbol{y}_j + \boldsymbol{t}_k). \tag{6.2}$$

$\boldsymbol{K}$ s $(\boldsymbol{R}_k\boldsymbol{y}_j + \boldsymbol{t}_k)$ $\boldsymbol{T}_k$ $\boldsymbol{y}_j$
$\boldsymbol{w}_k, \boldsymbol{v}_{k,j}$

$$\boldsymbol{w}_k \sim \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{R}_k\right), \boldsymbol{v}_k \sim \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{Q}_{k,j}\right). \tag{6.3}$$

$\mathcal{N}$ $\boldsymbol{0}$ $\boldsymbol{R}_k, \boldsymbol{Q}_{k,j}$ $\boldsymbol{z}$ $\boldsymbol{u}$ $\boldsymbol{x}$ $\boldsymbol{y}$
SLAM / incremental
$\boldsymbol{x}_k$ [13] SLAM
from Motion SLAM SLAM
1 N M

$$\boldsymbol{x} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}, \quad \boldsymbol{y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_M\}.$$

$\boldsymbol{u}$ $\boldsymbol{z}$ $\boldsymbol{u}$ $\boldsymbol{z}$ $\boldsymbol{x}, \boldsymbol{y}$

$$P(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{z}, \boldsymbol{u}). \tag{6.4}$$

$P(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{z})$ Structure from Motion SfM [29]

$$P\left(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{z}, \boldsymbol{u}\right) = \frac{P\left(\boldsymbol{z}, \boldsymbol{u}|\boldsymbol{x}, \boldsymbol{y}\right) P\left(\boldsymbol{x}, \boldsymbol{y}\right)}{P\left(\boldsymbol{z}, \boldsymbol{u}\right)} \propto \underbrace{P\left(\boldsymbol{z}, \boldsymbol{u}|\boldsymbol{x}, \boldsymbol{y}\right)}\underbrace{P\left(\boldsymbol{x}, \boldsymbol{y}\right)}. \tag{6.5}$$

$P(\boldsymbol{z}|\boldsymbol{x})$ Likehood $P(\boldsymbol{x})$ Prior Maximize
a Posterior MAP

$$(\boldsymbol{x}, \boldsymbol{y})^*_{\mathrm{MAP}} = \arg\max\ P\left(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{z}, \boldsymbol{u}\right) = \arg\max P(\boldsymbol{z}, \boldsymbol{u}|\boldsymbol{x}, \boldsymbol{y})P(\boldsymbol{x}, \boldsymbol{y}). \tag{6.6}$$

$\boldsymbol{x}, \boldsymbol{y}$ Maximize
Likelihood Estimation MLE

$$(\boldsymbol{x}, \boldsymbol{y})^*_{\mathrm{MLE}} = \arg\max P(\boldsymbol{z}, \boldsymbol{u}|\boldsymbol{x}, \boldsymbol{y}). \tag{6.7}$$

“ ” “ ”

6.1.2

$$\boldsymbol{z}_{k,j} = h\left(\boldsymbol{y}_j, \boldsymbol{x}_k\right) + \boldsymbol{v}_{k,j},$$

$\boldsymbol{v}_k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{Q}_{k,j})$

$$P(\boldsymbol{z}_{j,k}|\boldsymbol{x}_k, \boldsymbol{y}_j) = N\left(h(\boldsymbol{y}_j, \boldsymbol{x}_k), \boldsymbol{Q}_{k,j}\right).$$

$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$

$$P(\boldsymbol{x}) = \frac{1}{\sqrt{(2)^N \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right). \tag{6.8}$$

$$-\ln(P(\boldsymbol{x})) = \frac{1}{2}\ln\left((2)^N \det(\boldsymbol{\Sigma})\right) + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}). \tag{6.9}$$

$\boldsymbol{x}$ $\boldsymbol{x}$ SLAM

$$\begin{aligned}(\boldsymbol{x}_k, \boldsymbol{y}_j)^* &= \arg\max \mathcal{N}(h(\boldsymbol{y}_j, \boldsymbol{x}_k), \boldsymbol{Q}_{k,j}) \\ &= \arg\min\left((\boldsymbol{z}_{k,j} - h(\boldsymbol{x}_k, \boldsymbol{y}_j))^{\mathrm{T}} \boldsymbol{Q}_{k,j}^{-1}(\boldsymbol{z}_{k,j} - h(\boldsymbol{x}_k, \boldsymbol{y}_j))\right).\end{aligned} \tag{6.10}$$

Mahalanobis distance $\boldsymbol{Q}_{k,j}^{-1}$ $\boldsymbol{Q}_{k,j}^{-1}$

$$P(\boldsymbol{z}, \boldsymbol{u}|\boldsymbol{x}, \boldsymbol{y}) = \prod_k P(\boldsymbol{u}_k|\boldsymbol{x}_{k-1}, \boldsymbol{x}_k) \prod_{k,j} P(\boldsymbol{z}_{k,j}|\boldsymbol{x}_k, \boldsymbol{y}_j), \tag{6.11}$$

$$\begin{aligned}\boldsymbol{e}_{u,k} &= \boldsymbol{x}_k - f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \\ \boldsymbol{e}_{z,j,k} &= \boldsymbol{z}_{k,j} - h(\boldsymbol{x}_k, \boldsymbol{y}_j),\end{aligned} \tag{6.12}$$

$$\min J(\boldsymbol{x}, \boldsymbol{y}) = \sum_k \boldsymbol{e}_{u,k}^{\mathrm{T}} \boldsymbol{R}_k^{-1} \boldsymbol{e}_{u,k} + \sum_k \sum_j \boldsymbol{e}_{z,k,j}^{\mathrm{T}} \boldsymbol{Q}_{k,j}^{-1} \boldsymbol{e}_{z,k,j}. \tag{6.13}$$

Least Square Problem SLAM
(6.13) SLAM

- $\boldsymbol{x}_{k-1}, \boldsymbol{x}_k$ $\boldsymbol{x}_k, \boldsymbol{y}_j$
- / s.t. $\boldsymbol{R}^{\mathrm{T}}\boldsymbol{R} = \boldsymbol{I}$ $\det(\boldsymbol{R}) = 1$
- “ ” “ ”

SLAM

6.1.3

$$\begin{aligned}x_k &= x_{k-1} + u_k + w_k, & w_k &\sim \mathcal{N}(0, Q_k) \\ z_k &= x_k + n_k, & n_k &\sim \mathcal{N}(0, R_k)\end{aligned} \tag{6.14}$$

x u_k w_k z_k $k = 1, \ldots, 3$ v, y x_0 batch

$$\boldsymbol{x} = [x_0, x_1, x_2, x_3]^{\mathrm{T}} \qquad \boldsymbol{z} = [z_1, z_2, z_3]^{\mathrm{T}} \qquad \boldsymbol{u} = [u_1, u_2, u_3]^{\mathrm{T}}$$

$$\begin{aligned} \boldsymbol{x}^*_{\mathrm{map}} &= \arg\max P(\boldsymbol{x}|\boldsymbol{u}, \boldsymbol{z}) = \arg\max P(\boldsymbol{u}, \boldsymbol{z}|\boldsymbol{x}) \\ &= \prod_{k=1}^{3} P(u_k|x_{k-1}, x_k) \prod_{k=1}^{3} P\left(z_k|x_k\right), \end{aligned} \tag{6.15}$$

$$P(u_k|x_{k-1}, x_k) = \mathcal{N}(x_k - x_{k-1}, Q_k), \tag{6.16}$$

$$P\left(z_k|x_k\right) = \mathcal{N}\left(x_k, R_k\right). \tag{6.17}$$

$$e_{u,k} = x_k - x_{k-1} - u_k, \quad e_{z,k} = z_k - x_k, \tag{6.18}$$

$$\min \sum_{k=1}^{3} e_{u,k}^{\mathrm{T}} Q_k^{-1} e_{u,k} + \sum_{k=1}^{3} e_{z,k}^{\mathrm{T}} R_k^{-1} e_{z,k}. \tag{6.19}$$

$\boldsymbol{y} = [\boldsymbol{u}, \boldsymbol{z}]^{\mathrm{T}}$ $\boldsymbol{H}$

$$\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x} = \boldsymbol{e} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}). \tag{6.20}$$

$$\boldsymbol{H} = \left[\begin{array}{cccc} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ \hline 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right], \tag{6.21}$$

$\boldsymbol{\Sigma} = \mathrm{diag}(Q_1, Q_2, Q_3, R_1, R_2, R_3)$

$$\boldsymbol{x}^*_{\mathrm{map}} = \arg\min \boldsymbol{e}^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} \boldsymbol{e}, \tag{6.22}$$

$$\boldsymbol{x}^*_{\mathrm{map}} = (\boldsymbol{H}^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} \boldsymbol{H})^{-1} \boldsymbol{H}^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} \boldsymbol{y}. \tag{6.23}$$

6.2

$$\min_{\boldsymbol{x}} F(\boldsymbol{x}) = \frac{1}{2} \| f\left(\boldsymbol{x}\right) \|_2^2. \tag{6.24}$$

$\boldsymbol{x} \in \mathbb{R}^n$ f $f(\boldsymbol{x}) : \mathbb{R}^n \mapsto \mathbb{R}$ $\frac{1}{2}$ f

$$\frac{\mathrm{d}F}{\mathrm{d}\boldsymbol{x}} = \boldsymbol{0}. \tag{6.25}$$

f f

1. $\boldsymbol{x}_0$
2. k $\Delta\boldsymbol{x}_k$ $\|f(\boldsymbol{x}_k+\Delta\boldsymbol{x}_k)\|_2^2$
3. $\Delta\boldsymbol{x}_k$
4. $\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\Delta\boldsymbol{x}_k$ 2

$\Delta\boldsymbol{x}_k$ f f
$\Delta\boldsymbol{x}_k$

6.2.1

k $\boldsymbol{x}_k$ $\Delta\boldsymbol{x}_k$ $\boldsymbol{x}_k$

$$F(\boldsymbol{x}_k+\Delta\boldsymbol{x}_k)\approx F(\boldsymbol{x}_k)+\boldsymbol{J}(\boldsymbol{x}_k)^{\mathrm{T}}\Delta\boldsymbol{x}_k+\frac{1}{2}\Delta\boldsymbol{x}_k^{\mathrm{T}}\boldsymbol{H}(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k. \tag{6.26}$$

$\boldsymbol{J}(\boldsymbol{x}_k)$ $F(\boldsymbol{x})$ $\boldsymbol{x}$ Jacobian * $\boldsymbol{H}$ Hessian $\boldsymbol{x}_k$

$$\Delta\boldsymbol{x}^*=-\boldsymbol{J}(\boldsymbol{x}_k). \tag{6.27}$$

λ [30]
k k

$$\Delta\boldsymbol{x}^*=\arg\min\left(F(\boldsymbol{x})+\boldsymbol{J}(\boldsymbol{x})^{\mathrm{T}}\Delta\boldsymbol{x}+\frac{1}{2}\Delta\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\Delta\boldsymbol{x}\right). \tag{6.28}$$

$\Delta\boldsymbol{x}$ $\Delta\boldsymbol{x}$ †

$$\boldsymbol{J}+\boldsymbol{H}\Delta\boldsymbol{x}=\boldsymbol{0}\Rightarrow\boldsymbol{H}\Delta\boldsymbol{x}=-\boldsymbol{J}. \tag{6.29}$$

Newton's method — Levernburg-Marquardt's method

6.2.2

$f(\boldsymbol{x})$ $F(\boldsymbol{x})$ $f(\boldsymbol{x})$

$$f(\boldsymbol{x}+\Delta\boldsymbol{x})\approx f(\boldsymbol{x})+\boldsymbol{J}(\boldsymbol{x})^{\mathrm{T}}\Delta\boldsymbol{x}. \tag{6.30}$$

$\boldsymbol{J}(\boldsymbol{x})$ $f(\boldsymbol{x})$ $\boldsymbol{x}$ $n\times 1$ $\Delta\boldsymbol{x}$ $\|f(\boldsymbol{x}+\Delta\boldsymbol{x})\|^2$ $\Delta\boldsymbol{x}$

$$\Delta\boldsymbol{x}^*=\arg\min_{\Delta\boldsymbol{x}}\frac{1}{2}\left\|f(\boldsymbol{x})+\boldsymbol{J}(\boldsymbol{x})^{\mathrm{T}}\Delta\boldsymbol{x}\right\|^2. \tag{6.31}$$

$\Delta\boldsymbol{x}$

$$\begin{aligned}\frac{1}{2}\left\|f(\boldsymbol{x})+\boldsymbol{J}(\boldsymbol{x})^{\mathrm{T}}\Delta\boldsymbol{x}\right\|^2&=\frac{1}{2}\left(f(\boldsymbol{x})+\boldsymbol{J}(\boldsymbol{x})^{\mathrm{T}}\Delta\boldsymbol{x}\right)^{\mathrm{T}}\left(f(\boldsymbol{x})+\boldsymbol{J}(\boldsymbol{x})^{\mathrm{T}}\Delta\boldsymbol{x}\right)\\&=\frac{1}{2}\left(\|f(\boldsymbol{x})\|_2^2+2f(\boldsymbol{x})\boldsymbol{J}(\boldsymbol{x})^{\mathrm{T}}\Delta\boldsymbol{x}+\Delta\boldsymbol{x}^{\mathrm{T}}\boldsymbol{J}(\boldsymbol{x})\boldsymbol{J}(\boldsymbol{x})^{\mathrm{T}}\Delta\boldsymbol{x}\right).\end{aligned}$$

* $\boldsymbol{J}(\boldsymbol{x})$ $\Delta\boldsymbol{x}$
† B

$\Delta \boldsymbol{x}$

$$\boldsymbol{J}(\boldsymbol{x})f(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\boldsymbol{J}^{\mathrm{T}}(\boldsymbol{x})\,\Delta\boldsymbol{x} = \boldsymbol{0}.$$

$$\underbrace{\boldsymbol{J}(\boldsymbol{x})\boldsymbol{J}^{\mathrm{T}}}_{\boldsymbol{H}(\boldsymbol{x})}(\boldsymbol{x})\,\Delta\boldsymbol{x} = \underbrace{-\boldsymbol{J}(\boldsymbol{x})f(\boldsymbol{x})}_{\boldsymbol{g}(\boldsymbol{x})}. \tag{6.32}$$

$\Delta \boldsymbol{x}$ Gauss-Newton equation Normal equation $\boldsymbol{H}$ $\boldsymbol{g}$

$$\boldsymbol{H}\Delta\boldsymbol{x} = \boldsymbol{g}. \tag{6.33}$$

$\boldsymbol{H}$ $\boldsymbol{J}\boldsymbol{J}^{\mathrm{T}}$ **Hessian** $\boldsymbol{H}$

1. $\boldsymbol{x}_0$
2. k $\boldsymbol{J}(\boldsymbol{x}_k)$ $f(\boldsymbol{x}_k)$
3. $\boldsymbol{H}\Delta\boldsymbol{x}_k = \boldsymbol{g}$
4. $\Delta\boldsymbol{x}_k$ $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k$ 2

$\boldsymbol{H}^{-1}$ $\boldsymbol{H}$ $\boldsymbol{J}\boldsymbol{J}^{\mathrm{T}}$ $\boldsymbol{J}\boldsymbol{J}^{\mathrm{T}}$ ill-condition (line search method) α $\Delta\boldsymbol{x}$ α $\|f(\boldsymbol{x}+\alpha\Delta\boldsymbol{x})\|^2$ $\alpha = 1$ — Damped Newton Method

6.2.3 —

$\Delta\boldsymbol{x}$ Trust Region Trust Region Method

ρ

$$\rho = \frac{f(\boldsymbol{x}+\Delta\boldsymbol{x}) - f(\boldsymbol{x})}{\boldsymbol{J}(\boldsymbol{x})^{\mathrm{T}}\Delta\boldsymbol{x}}. \tag{6.34}$$

ρ ρ 1 ρ ρ

1. $\boldsymbol{x}_0$ μ
2. k

$$\min_{\Delta\boldsymbol{x}_k} \frac{1}{2}\left\| f(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)^{\mathrm{T}}\Delta\boldsymbol{x}_k \right\|^2, \quad \text{s.t.} \quad \|\boldsymbol{D}\Delta\boldsymbol{x}_k\|^2 \leqslant \mu, \tag{6.35}$$

μ $\boldsymbol{D}$

3. (6.34) ρ
4. $\rho > \frac{3}{4}$ $\mu = 2\mu$
5. $\rho < \frac{1}{4}$ $\mu = 0.5\mu$
6. ρ $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k$

7. 2

(6.35) μ $\boldsymbol{D}$ $\boldsymbol{D}$ $\boldsymbol{I}$ $\Delta\boldsymbol{x}_k$ — $\boldsymbol{J}^T\boldsymbol{J}$

— (6.35)

$$\mathcal{L}(\Delta\boldsymbol{x}_k, \lambda) = \frac{1}{2}\left\| f(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)^{\mathrm{T}} \Delta\boldsymbol{x}_k \right\|^2 + \frac{\lambda}{2}\left(\|\boldsymbol{D}\Delta\boldsymbol{x}_k\|^2 - \mu \right). \tag{6.36}$$

λ $\Delta\boldsymbol{x}$

$$\left(\boldsymbol{H} + \lambda \boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}\right)\Delta\boldsymbol{x}_k = \boldsymbol{g}. \tag{6.37}$$

$\lambda\boldsymbol{D}^T\boldsymbol{D}$ $\boldsymbol{D} = \boldsymbol{I}$ *

$$(\boldsymbol{H} + \lambda\boldsymbol{I})\Delta\boldsymbol{x}_k = \boldsymbol{g}.$$

λ $\boldsymbol{H}$ — λ $\lambda\boldsymbol{I}$ — —
$\Delta\boldsymbol{x}$
Dog-Leg[31] SLAM —

6.2.4

—— — [31] —
SLAM
—
SLAM $\Delta\boldsymbol{x}$
SLAM 9 PC

6.3

6.3.1

$$y = \exp(ax^2 + bx + c) + w,$$

a, b, c w $w \sim (0, \sigma^2)$ N x, y

$$\min_{a,b,c} \frac{1}{2}\sum_{i=1}^{N} \left\| y_i - \exp\left(ax_i^2 + bx_i + c\right) \right\|^2. \tag{6.38}$$

a, b, c x x, y :

$$e_i = y_i - \exp\left(ax_i^2 + bx_i + c\right), \tag{6.39}$$

* KKT $\lambda > 0$ $\lambda(\|\boldsymbol{D}\Delta\boldsymbol{x}\|^2 - \mu) = 0$ L-M λ Argumented Lagrangian λ L-M

$$\begin{aligned}\frac{\partial e_i}{\partial a} &= -x_i^2 \exp\left(ax_i^2 + bx_i + c\right) \\ \frac{\partial e_i}{\partial b} &= -x_i \exp\left(ax_i^2 + bx_i + c\right) \\ \frac{\partial e_i}{\partial c} &= -\exp\left(ax_i^2 + bx_i + c\right)\end{aligned} \tag{6.40}$$

$\boldsymbol{J}_i = \left[\frac{\partial e_i}{\partial a}, \frac{\partial e_i}{\partial b}, \frac{\partial e_i}{\partial c}\right]^{\mathrm{T}}$

$$\left(\sum_{i=1}^{100} \boldsymbol{J}_i (\sigma^2)^{-1} \boldsymbol{J}_i^{\mathrm{T}}\right) \Delta \boldsymbol{x}_k = \sum_{i=1}^{100} -\boldsymbol{J}_i (\sigma^2)^{-1} e_i, \tag{6.41}$$

$\boldsymbol{J}_i$

Listing 6.1: slambook2/ch6/gaussNewton.cpp

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <Eigen/Core>
#include <Eigen/Dense>

using namespace std;
using namespace Eigen;

int main(int argc, char **argv) {
    double ar = 1.0, br = 2.0, cr = 1.0;         // □□□□□
    double ae = 2.0, be = −1.0, ce = 5.0;        // □□□□□
    int N = 100;                                 // □□□
    double w_sigma = 1.0;                        // □□□Sigma
    double inv_sigma = 1.0 / w_sigma;
    cv::RNG rng;                                 // □□□□□□OpenCV

    vector<double> x_data, y_data;      // □□
    for (int i = 0; i < N; i++) {
        double x = i / 100.0;
        x_data.push_back(x);
        y_data.push_back(exp(ar * x * x + br * x + cr) + rng.gaussian(w_sigma *
             w_sigma));
    }

    // □□Gauss−□□Newton
    int iterations = 100;    // □□□□
    double cost = 0, lastCost = 0;  // □□□□□□□□□□□□□costcost

    chrono::steady_clock::time_point t1 = chrono::steady_clock::now();
    for (int iter = 0; iter < iterations; iter++) {

        Matrix3d H = Matrix3d::Zero();             // Hessian = J^T W^{−1} J in Gauss−
             Newton
        Vector3d b = Vector3d::Zero();             // bias
        cost = 0;

        for (int i = 0; i < N; i++) {
            double xi = x_data[i], yi = y_data[i];  // □□□□□i
            double error = yi − exp(ae * xi * xi + be * xi + ce);
            Vector3d J; // □□□□□
            J[0] = −xi * xi * exp(ae * xi * xi + be * xi + ce);  // de/da
            J[1] = −xi * exp(ae * xi * xi + be * xi + ce);  // de/db
            J[2] = −exp(ae * xi * xi + be * xi + ce);  // de/dc

            H += inv_sigma * inv_sigma * J * J.transpose();
            b += −inv_sigma * inv_sigma * error * J;

            cost += error * error;
        }

```

```
        // □□□□□□ Hx=b
        Vector3d dx = H.ldlt().solve(b);
        if (isnan(dx[0])) {
            cout << "result is nan!" << endl;
            break;
        }

        if (iter > 0 && cost >= lastCost) {
            cout << "cost: " << cost << ">= last cost: " << lastCost << ", break." <<
                endl;
            break;
        }

        ae += dx[0];
        be += dx[1];
        ce += dx[2];

        lastCost = cost;

        cout << "total cost: " << cost << ", \t\tupdate: " << dx.transpose() <<
          "\t\testimated params: " << ae << "," << be << "," << ce << endl;
    }

    chrono::steady_clock::time_point t2 = chrono::steady_clock::now();
    chrono::duration<double> time_used = chrono::duration_cast<chrono::duration<double
        >>(t2 - t1);
    cout << "solve time cost = " << time_used.count() << " seconds. " << endl;
    cout << "estimated abc = " << ae << ", " << be << ", " << ce << endl;
    return 0;
}
```

Listing 6.2:

```
/home/xiang/Code/slambook2/ch6/cmake-build-debug/gaussNewton
total cost: 3.19575e+06,    update: 0.0455771  0.078164 -0.985329   estimated params:
    2.04558,-0.921836,4.01467
total cost: 376785,     update:  0.065762  0.224972 -0.962521   estimated params:
    2.11134,-0.696864,3.05215
total cost: 35673.6,    update: -0.0670241   0.617616  -0.907497    estimated params:
    2.04432,-0.0792484,2.14465
total cost: 2195.01,    update: -0.522767   1.19192 -0.756452   estimated params:
    1.52155,1.11267,1.3882
total cost: 174.853,    update: -0.537502  0.909933 -0.386395   estimated params:
    0.984045,2.0226,1.00181
total cost: 102.78,     update: -0.0919666   0.147331 -0.0573675    estimated params:
    0.892079,2.16994,0.944438
total cost: 101.937,    update: -0.00117081  0.00196749 -0.00081055   estimated params
    : 0.890908,2.1719,0.943628
total cost: 101.937,    update:   3.4312e-06 -4.28555e-06  1.08348e-06    estimated
    params: 0.890912,2.1719,0.943629
total cost: 101.937,    update: -2.01204e-08  2.68928e-08 -7.86602e-09    estimated
    params: 0.890912,2.1719,0.943629
cost: 101.937>= last cost: 101.937, break.
solve time cost = 0.000212903 seconds.
estimated abc = 0.890912, 2.1719, 0.943629
```

9 Figure 6-1 CPU i7-8700 0.2

6.3.2 Ceres

C++ Ceres [35] g2o [36] g2o Ceres g2o “ ” “ ”

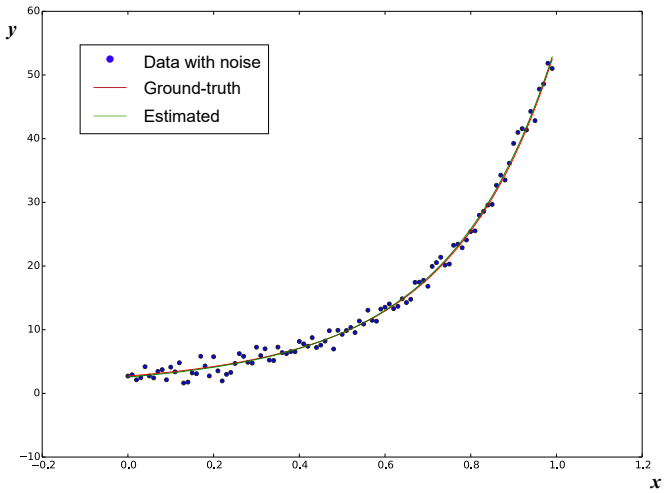


Figure 6-1: $\sigma = 1$

Ceres

Google Ceres Ceres Ceres

$$\begin{aligned} \min_{x} \quad & \frac{1}{2} \sum_{i} \rho_i \left(\left\| f_i \left(x_{i_1}, \cdots x_{i_n} \right) \right\|^2 \right) \\ \text{s.t.} \quad & l_j \leqslant x_j \leqslant u_j. \end{aligned} \tag{6.42}$$

$x_1, \cdots, x_n$ Parameter blocks f_i Cost function Residual blocks SLAM l_j $-\infty, u_j = \infty$ $\rho(\cdot)$ * ρ Ceres

1. SLAM double
2. Ceres
3. Ceres “ ”
4. Ceres Problem Solve

Ceres

Ceres

Ceres Ceres github `https://github.com/ceres-solver/ceres-solver` 3rdpar Ceres cmake Ubuntu apt-get

Listing 6.3:

```
sudo apt-get install liblapack-dev libsuitesparse-dev libcxsparse3 libgflags-dev
    libgoogle-glog-dev libgtest-dev
```

Ceres cmake /usr/local/include/ceres Ceres /usr/local/lib/ li

* 9

Ceres

Ceres

Listing 6.4: slambook/ch6/ceresCurveFitting.cpp

```
#include <iostream>
#include <opencv2/core/core.hpp>
#include <ceres/ceres.h>
#include <chrono>

using namespace std;

// 
struct CURVE_FITTING_COST {
    CURVE_FITTING_COST(double x, double y) : _x(x), _y(y) {}

    // 
    template<typename T>
    bool operator()(
        const T *const abc, // 3
        T *residual) const {
        // y-exp(ax^2+bx+c)
        residual[0] = T(_y) - ceres::exp(abc[0] * T(_x) * T(_x) + abc[1] * T(_x) + abc
            [2]);
        return true;
    }

    const double _x, _y;    // x,y
};

int main(int argc, char **argv) {
    double ar = 1.0, br = 2.0, cr = 1.0;         // 
    double ae = 2.0, be = -1.0, ce = 5.0;        // 
    int N = 100;                                 // 
    double w_sigma = 1.0;                        // Sigma
    double inv_sigma = 1.0 / w_sigma;
    cv::RNG rng;                                 // OpenCV

    vector<double> x_data, y_data;      // 
    for (int i = 0; i < N; i++) {
        double x = i / 100.0;
        x_data.push_back(x);
        y_data.push_back(exp(ar * x * x + br * x + cr) + rng.gaussian(w_sigma *
            w_sigma));
    }

    double abc[3] = {ae, be, ce};

    // 
    ceres::Problem problem;
    for (int i = 0; i < N; i++) {
        problem.AddResidualBlock(     // 
            // struct
            new ceres::AutoDiffCostFunction<CURVE_FITTING_COST, 1, 3>(
                new CURVE_FITTING_COST(x_data[i], y_data[i])
            ),
            nullptr,            // 
            abc                 // 
        );
    }

    // 
    ceres::Solver::Options options;     // 
    options.linear_solver_type = ceres::DENSE_NORMAL_CHOLESKY;  // 
    options.minimizer_progress_to_stdout = true;   // cout

    ceres::Solver::Summary summary;                // 
    chrono::steady_clock::time_point t1 = chrono::steady_clock::now();
    ceres::Solve(options, &problem, &summary);  // 
    chrono::steady_clock::time_point t2 = chrono::steady_clock::now();
```

```
64      chrono::duration<double> time_used = chrono::duration_cast<chrono::duration<double
            >>(t2 - t1);
65      cout << "solve time cost = " << time_used.count() << " seconds. " << endl;
66
67      // □□□□
68      cout << summary.BriefReport() << endl;
69      cout << "estimated a,b,c = ";
70      for (auto a:abc) cout << a << " ";
71      cout << endl;
72
73      return 0;
74  }
```

OpenCV 100 Ceres Ceres

1. () Functor * Ceres a a<double>() Ceres

2. double abc[3] CURVE_FITTING_COST AddResidualBlock
   Diff 2 Numeric Diff † 3 Ceres

3. 1 a, b, c 3 AutoDiffCostFunction 1 3

4. Solve options Line Search Trust Region Options

build/ceresCurveFitting

```
1  iter      cost      cost_change  |gradient|   |step|    tr_ratio  tr_radius  ls_iter
       iter_time  total_time
2  0  1.597873e+06    0.00e+00    3.52e+06   0.00e+00   0.00e+00  1.00e+04        0
       2.10e-05    7.92e-05
3  1  1.884440e+05    1.41e+06    4.86e+05   9.88e-01   8.82e-01  1.81e+04        1
       5.60e-05    1.05e-03
4  2  1.784821e+04    1.71e+05    6.78e+04   9.89e-01   9.06e-01  3.87e+04        1
       2.00e-05    1.09e-03
5  3  1.099631e+03    1.67e+04    8.58e+03   1.10e+00   9.41e-01  1.16e+05        1
       6.70e-05    1.16e-03
6  4  8.784938e+01    1.01e+03    6.53e+02   1.51e+00   9.67e-01  3.48e+05        1
       1.88e-05    1.19e-03
7  5  5.141230e+01    3.64e+01    2.72e+01   1.13e+00   9.90e-01  1.05e+06        1
       1.81e-05    1.22e-03
8  6  5.096862e+01    4.44e-01    4.27e-01   1.89e-01   9.98e-01  3.14e+06        1
       1.79e-05    1.25e-03
9  7  5.096851e+01    1.10e-04    9.53e-04   2.84e-03   9.99e-01  9.41e+06        1
       1.81e-05    1.28e-03
10 solve time cost = 0.00130755 seconds.
11 Ceres Solver Report: Iterations: 8, Initial cost: 1.597873e+06, Final cost: 5.096851e
       +01, Termination: CONVERGENCE
12 estimated a,b,c = 0.890908 2.1719 0.943628
```

Ceres Ceres 1.3
Ceres Ceres

6.3.3 g2o

2 SLAM g2o General Graphic Optimization G^2O

x_j

Graph **Vertex** **Edge**

Figure 6-2 (6.13)
g2o “ ” g2o

---

* C++
†

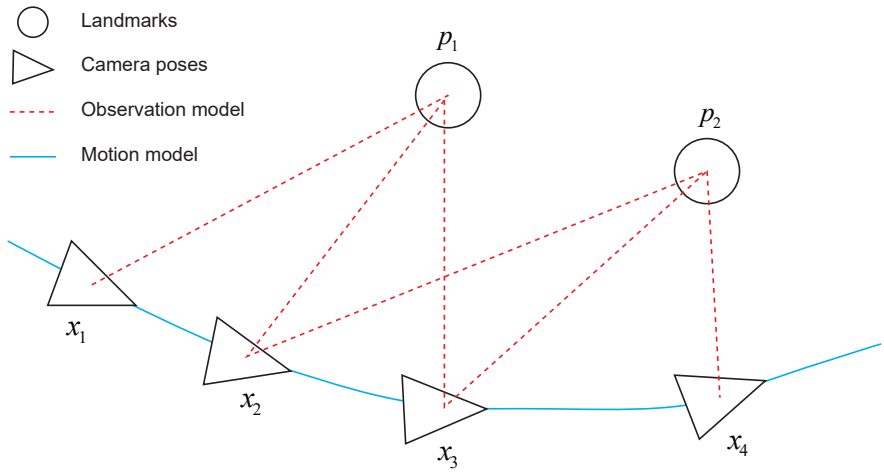


Figure 6-2:

6.3.4 g2o

g2o GitHub `https://github.com/RainerKuemmerle/g2o` g2o 3rdparty g2o g2o cmake Ceres

Listing 6.5:

```
sudo apt-get install qt5-qmake qt5-default libqglviewer-dev-qt5 libsuitesparse-dev
    libcxsparse3 libcholmod3
```

cmake g2o g2o /usr/local/g2o /usr/local/lib/ Ceres g2o

6.3.5 g2o

g2o Figure 6-3

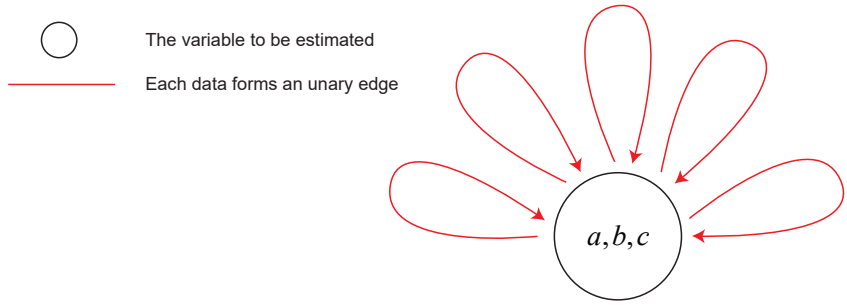


Figure 6-3:

a, b, c Unary Edge
Figure 6-3 Hyper Edge Hyper

Graph *

g2o g2o

1.

2.

3.

4. g2o

Ceres

Listing 6.6: slambook/ch6/g2oCurveFitting.cpp

```
#include <iostream>
#include <g2o/core/g2o_core_api.h>
#include <g2o/core/base_vertex.h>
#include <g2o/core/base_unary_edge.h>
#include <g2o/core/block_solver.h>
#include <g2o/core/optimization_algorithm_levenberg.h>
#include <g2o/core/optimization_algorithm_gauss_newton.h>
#include <g2o/core/optimization_algorithm_dogleg.h>
#include <g2o/solvers/dense/linear_solver_dense.h>
#include <Eigen/Core>
#include <opencv2/core/core.hpp>
#include <cmath>
#include <chrono>

using namespace std;

// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
class CurveFittingVertex : public g2o::BaseVertex<3, Eigen::Vector3d> {
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW

    // □□
    virtual void setToOriginImpl() override {
        _estimate << 0, 0, 0;
    }

    // □□
    virtual void oplusImpl(const double *update) override {
        _estimate += Eigen::Vector3d(update);
    }

    // □□□□□□□□
    virtual bool read(istream &in) {}
    virtual bool write(ostream &out) const {}
};

// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
class CurveFittingEdge : public g2o::BaseUnaryEdge<1, double, CurveFittingVertex> {
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW

    CurveFittingEdge(double x) : BaseUnaryEdge(), _x(x) {}

    // □□□□□□□□
    virtual void computeError() override {
        const CurveFittingVertex *v = static_cast<const CurveFittingVertex *> (
            _vertices[0]);
        const Eigen::Vector3d abc = v->estimate();
        _error(0, 0) = _measurement - std::exp(abc(0, 0) * _x * _x + abc(1, 0) * _x +
            abc(2, 0));
    }

```

*

```
51      // □□□□□□□□
52      virtual void linearizeOplus() override {
53          const CurveFittingVertex *v = static_cast<const CurveFittingVertex *> (
                _vertices[0]);
54          const Eigen::Vector3d abc = v->estimate();
55          double y = exp(abc[0] * _x * _x + abc[1] * _x + abc[2]);
56          _jacobianOplusXi[0] = -_x * _x * y;
57          _jacobianOplusXi[1] = -_x * y;
58          _jacobianOplusXi[2] = -y;
59      }
60
61      virtual bool read(istream &in) {}
62      virtual bool write(ostream &out) const {}
63  public:
64      double _x;  // x □□ y □□ _measurement
65  };
66
67  int main(int argc, char **argv) {
68      // □□□□□□□□□□□□
69      // □□□□□□□□□□g2o
70      typedef g2o::BlockSolver<g2o::BlockSolverTraits<3, 1>> BlockSolverType;  //
            □□□□□□□□□□□□□□□□□□□□□□31
71      typedef g2o::LinearSolverDense<BlockSolverType::PoseMatrixType> LinearSolverType;
            // □□□□□□□□
72
73      // □□□□□□□□□□□GN, LM, DogLeg □□
74      auto solver = new g2o::OptimizationAlgorithmGaussNewton(
75          g2o::make_unique<BlockSolverType>(g2o::make_unique<LinearSolverType>()));
76      g2o::SparseOptimizer optimizer;     // □□□
77      optimizer.setAlgorithm(solver);   // □□□□□
78      optimizer.setVerbose(true);       // □□□□□□
79
80      // □□□□□□□□
81      CurveFittingVertex *v = new CurveFittingVertex();
82      v->setEstimate(Eigen::Vector3d(ae, be, ce));
83      v->setId(0);
84      optimizer.addVertex(v);
85
86      // □□□□□□
87      for (int i = 0; i < N; i++) {
88          CurveFittingEdge *edge = new CurveFittingEdge(x_data[i]);
89          edge->setId(i);
90          edge->setVertex(0, v);                // □□□□□□□□
91          edge->setMeasurement(y_data[i]);      // □□□□
92          edge->setInformation(Eigen::Matrix<double, 1, 1>::Identity() * 1 / (w_sigma *
                w_sigma)); // □□□□□□□□□□□□□
93          optimizer.addEdge(edge);
94      }
95
96      // □□□□
97      cout << "start optimization" << endl;
98      chrono::steady_clock::time_point t1 = chrono::steady_clock::now();
99      optimizer.initializeOptimization();
100     optimizer.optimize(10);
101     chrono::steady_clock::time_point t2 = chrono::steady_clock::now();
102     chrono::duration<double> time_used = chrono::duration_cast<chrono::duration<double
            >>(t2 - t1);
103     cout << "solve time cost = " << time_used.count() << " seconds. " << endl;
104
105     // □□□□□
106     Eigen::Vector3d abc_estimate = v->estimate();
107     cout << "estimated model: " << abc_estimate.transpose() << endl;
108
109     return 0;
110 }
```

g2o CurveFittingVertex CurveFittingEdge g2o BaseVertex BaseUna

1. oplusImpl $\Delta\boldsymbol{x}$ $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}$

g2o $\boldsymbol{x}$

2. setToOriginImpl
3. computeError
4. linearizeOplus
5. read write /

main g2o

Listing 6.7:

```
start optimization
iteration= 0   chi2= 376785.128234   time= 3.3299e-05  cumTime= 3.3299e-05   edges= 100  schur= 0
iteration= 1   chi2= 35673.566018  time= 1.3789e-05  cumTime= 4.7088e-05   edges= 100  schur= 0
iteration= 2   chi2= 2195.012304   time= 1.2323e-05  cumTime= 5.9411e-05   edges= 100  schur= 0
iteration= 3   chi2= 174.853126  time= 1.3302e-05  cumTime= 7.2713e-05   edges= 100  schur= 0
iteration= 4   chi2= 102.779695  time= 1.2424e-05  cumTime= 8.5137e-05   edges= 100  schur= 0
iteration= 5   chi2= 101.937194  time= 1.2523e-05  cumTime= 9.766e-05  edges= 100  schur= 0
iteration= 6   chi2= 101.937020  time= 1.2268e-05  cumTime= 0.000109928  edges= 100  schur= 0
iteration= 7   chi2= 101.937020  time= 1.2612e-05  cumTime= 0.00012254   edges= 100  schur= 0
iteration= 8   chi2= 101.937020  time= 1.2159e-05  cumTime= 0.000134699  edges= 100  schur= 0
iteration= 9   chi2= 101.937020  time= 1.2688e-05  cumTime= 0.000147387  edges= 100  schur= 0
solve time cost = 0.000919301 seconds.
estimated model: 0.890912   2.1719 0.943629
```

— 9 Ceres g2o g2o Ceres Ceres

6.4

SLAM — Ceres g2o
Bundle Adjustment g2o —
—g2o Ceres SLAM
Function
Ceres g2o
Ceres g2o Ceres g2o SLAM
Graph

1. $\boldsymbol{Ax} = \boldsymbol{b}$ $\boldsymbol{A}$ $\boldsymbol{x} = (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$
2. — Ceres g2o MATLAB
3.
4. DogLeg — *
5. Ceres `http://ceres-solver.org/tutorial.html`
6. g2o 10 11
7.* Ceres g2o

* `http://www.numerical.rl.ac.uk/people/nimg/course/lectures/raphael/lectures/lec7slides.pdf`

Part II

Chapter 7

1

1. ,
2.
3. PNP
4. ICP
5.

2 4

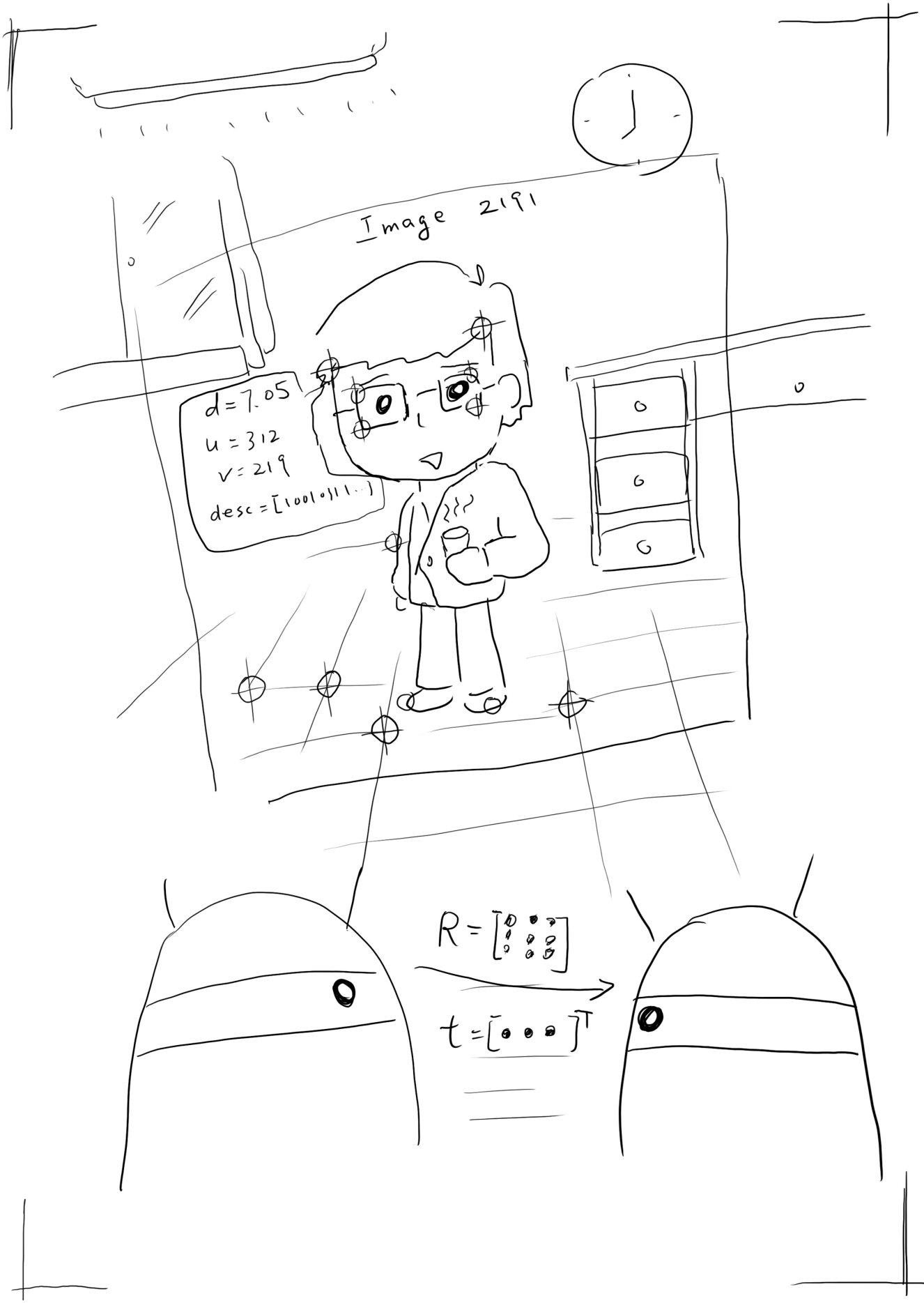


Image 2191
d = 7.05
u = 312
v = 219
desc = [10010111...]
R = [ ]
t = [ ]^T

7.1

SLAM VO VO VO
view geometry

7.1.1

VO
[37]
Figure 7-1
SIF

1. Repeatability
2. Distinctiveness
3. Efficiency
4. Locality

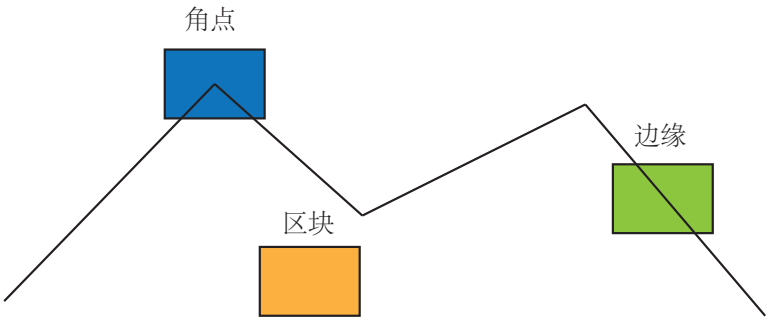


Figure 7-1:

Key-point Descriptor “ SIFT ” “ SIFT SIFT ”
SIFT Scale-Invariant Feature Trans-
form SLAM 2016 PC CPU SIFT
FAST “ ” ORB Oriented FAST and
Rotated BRIEF FAST [39] BRIEF[44] 1000
GPU GPU SIFT GPU SLAM
SLAM ORB ORB [45]

7.1.2 ORB

ORB “Oriented FAST” FAST FAST BRIEF Binary
Robust Independent Elementary Feature ORB

1. FAST “ ” FAST ORB BRIEF
2. BRIEF ORB BRIEF BRIEF
   FAST BRIEF

* 30Hz

FAST

FAST FAST Figure 7-2

1. p I_p
2. T I_p 20%
3. p 3 16
4. N I_p+T I_p-T p N 12 FAST-12 N 9 11 FAST-9 FAST-11
5.

FAST-12 1, 5, 9, 13 4 3 $I_p + T$ I_p-T FAST “ ” Non-maximal suppression

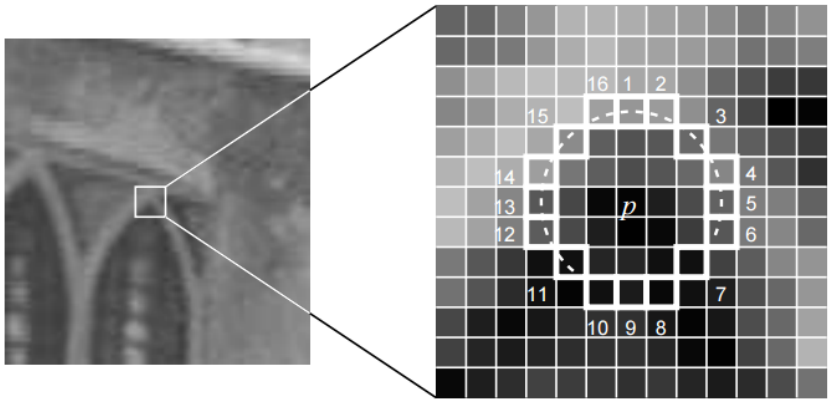


Figure 7-2: FAST [39]

FAST FAST 3 FAST Centroid

Figure 7-3

[46]

1. B

$$m_{pq} = \sum_{x,y \in B} x^p y^q I(x,y), \quad p,q = \{0,1\}.$$

2.

$$C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}).$$

3. O C $\overrightarrow{OC}$

$$\theta = \arctan(m_{01}/m_{10}).$$

FAST ORB FAST Oriented FAST

*

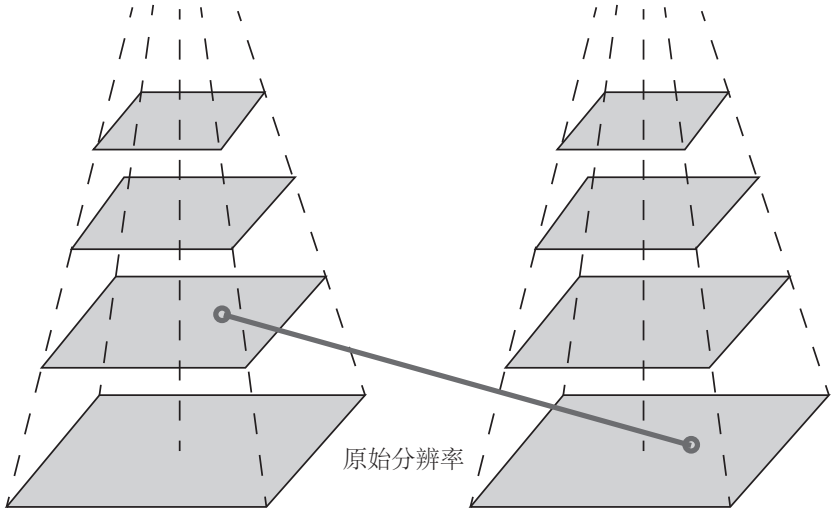


Figure 7-3:

BRIEF

Oriented FAST ORB BRIEF BRIEF
BRIEF 0 1 0 1 p q p q 1 0 128 p, q 128 0 1 [44] BRIEF
BRIEF" ORB
ORB FAST BREIF ORB SLAM Figure 7-
4 ORB

7.1.3

Figure 7-5 SLAM SLAM data association
I_t $x_t^m, m = 1, 2, ..., M$ I_{t+1} $x_{t+1}^n, n = 1, 2, ..., N$
Force Matcher x_t^m x_{t+1}^n B
distance ——
SLAM **FLANN** OpenC

7.2

OpenCV OpenCV ORB ORB O

7.2.1 OpenCV ORB

OpenCV ORB slambook2/ch7/ 1.png 2.png Figure 7-6 [23]
ORB

Listing 7.1: slambook2/ch7/orb_cv.cpp

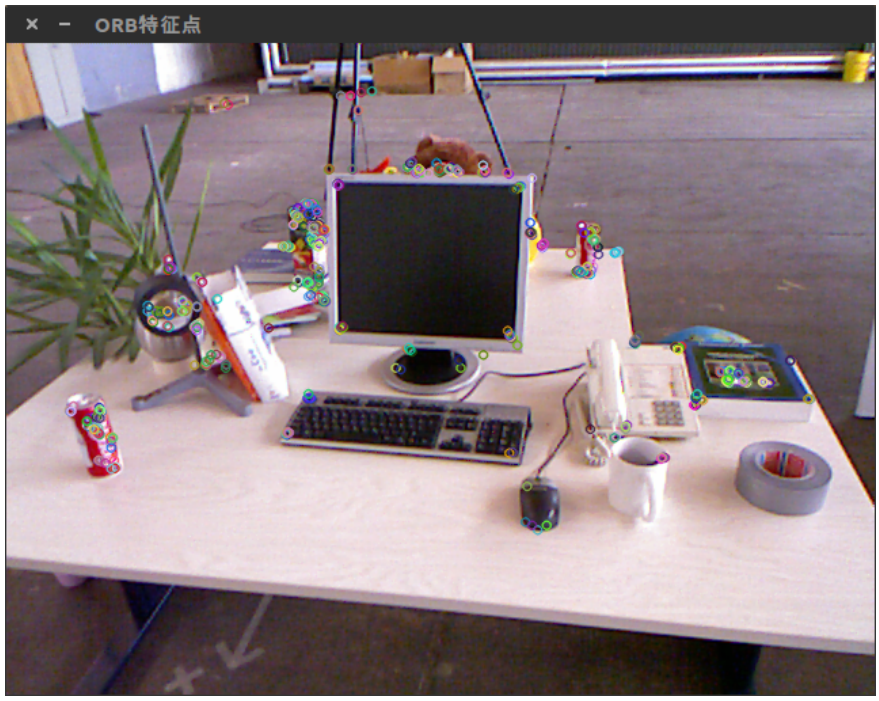


Figure 7-4: OpenCV ORB

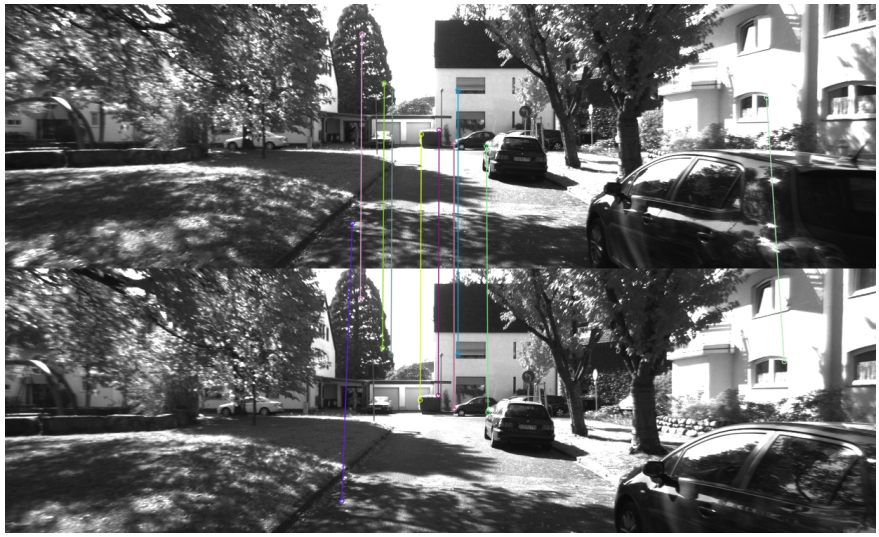


Figure 7-5:

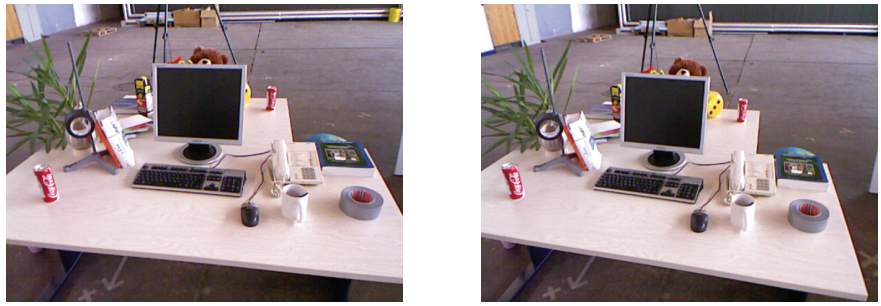


Figure 7-6:

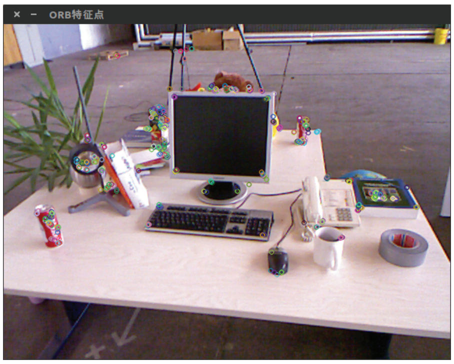
```cpp
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/features2d/features2d.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <chrono>

using namespace std;
using namespace cv;

int main(int argc, char **argv) {
    if (argc != 3) {
        cout << "usage: feature_extraction img1 img2" << endl;
        return 1;
    }
    //-- 
    Mat img_1 = imread(argv[1], CV_LOAD_IMAGE_COLOR);
    Mat img_2 = imread(argv[2], CV_LOAD_IMAGE_COLOR);
    assert(img_1.data != nullptr && img_2.data != nullptr);

    //-- 
    std::vector<KeyPoint> keypoints_1, keypoints_2;
    Mat descriptors_1, descriptors_2;
    Ptr<FeatureDetector> detector = ORB::create();
    Ptr<DescriptorExtractor> descriptor = ORB::create();
    Ptr<DescriptorMatcher> matcher = DescriptorMatcher::create("BruteForce-Hamming");

    //-- : Oriented FAST 
    chrono::steady_clock::time_point t1 = chrono::steady_clock::now();
    detector->detect(img_1, keypoints_1);
    detector->detect(img_2, keypoints_2);

    //-- : BRIEF 
    descriptor->compute(img_1, keypoints_1, descriptors_1);
    descriptor->compute(img_2, keypoints_2, descriptors_2);
    chrono::steady_clock::time_point t2 = chrono::steady_clock::now();
    chrono::duration<double> time_used = chrono::duration_cast<chrono::duration<double
        >>(t2 - t1);
    cout << "extract ORB cost = " << time_used.count() << " seconds. " << endl;

    Mat outimg1;
    drawKeypoints(img_1, keypoints_1, outimg1, Scalar::all(-1), DrawMatchesFlags::
        DEFAULT);
    imshow("ORB features", outimg1);

    //--:BRIEF Hamming 
    vector<DMatch> matches;
    t1 = chrono::steady_clock::now();
    matcher->match(descriptors_1, descriptors_2, matches);
    t2 = chrono::steady_clock::now();
    time_used = chrono::duration_cast<chrono::duration<double>>(t2 - t1);
    cout << "match ORB cost = " << time_used.count() << " seconds. " << endl;

    //--:
    // 
```

```
    auto min_max = minmax_element(matches.begin(), matches.end(),
        [](const DMatch &m1, const DMatch &m2) { return m1.distance < m2.distance; });
    double min_dist = min_max.first->distance;
    double max_dist = min_max.second->distance;

    printf("-- Max dist : %f \n", max_dist);
    printf("-- Min dist : %f \n", min_dist);

    //□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□,.,30.
    std::vector<DMatch> good_matches;
    for (int i = 0; i < descriptors_1.rows; i++) {
        if (matches[i].distance <= max(2 * min_dist, 30.0)) {
            good_matches.push_back(matches[i]);
        }
    }

    //-- □□□□□□□□□:
    Mat img_match;
    Mat img_goodmatch;
    drawMatches(img_1, keypoints_1, img_2, keypoints_2, matches, img_match);
    drawMatches(img_1, keypoints_1, img_2, keypoints_2, good_matches, img_goodmatch);
    imshow("all matches", img_match);
    imshow("good matches", img_goodmatch);
    waitKey(0);

    return 0;
}
```

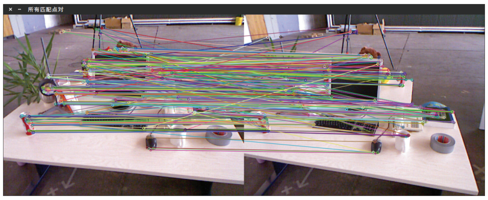
Listing 7.2:

```
% build/orb_cv 1.png 2.png
extract ORB cost = 0.0229183 seconds.
match ORB cost = 0.000751868 seconds.
-- Max dist : 95.000000
-- Min dist : 4.000000
```

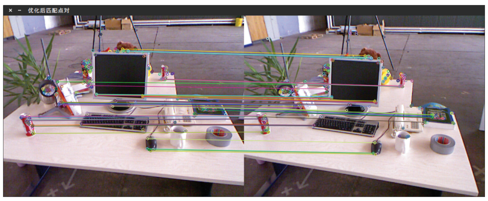
Figure 7-7



ORB关键点



未筛选的匹配



筛选后的匹配

Figure 7-7:

7.2.2 ORB

ORB

Listing 7.3: slambook2/ch7/orb_self.cpp

```
typedef vector<uint32_t> DescType;
// ... 
// compute the descriptor
void ComputeORB(const cv::Mat &img, vector<cv::KeyPoint> &keypoints, vector<DescType> &descriptors) {
    const int half_patch_size = 8;
    const int half_boundary = 16;
    int bad_points = 0;
    for (auto &kp: keypoints) {
        if (kp.pt.x < half_boundary || kp.pt.y < half_boundary ||
        kp.pt.x >= img.cols - half_boundary || kp.pt.y >= img.rows - half_boundary) {
            // outside
            bad_points++;
            descriptors.push_back({});
            continue;
        }

        float m01 = 0, m10 = 0;
        for (int dx = -half_patch_size; dx < half_patch_size; ++dx) {
            for (int dy = -half_patch_size; dy < half_patch_size; ++dy) {
                uchar pixel = img.at<uchar>(kp.pt.y + dy, kp.pt.x + dx);
                m01 += dx * pixel;
                m10 += dy * pixel;
            }
        }

        // angle should be arc tan(m01/m10);
        float m_sqrt = sqrt(m01 * m01 + m10 * m10);
        float sin_theta = m01 / m_sqrt;
        float cos_theta = m10 / m_sqrt;

        // compute the angle of this point
        DescType desc(8, 0);
        for (int i = 0; i < 8; i++) {
            uint32_t d = 0;
            for (int k = 0; k < 32; k++) {
                int idx_pq = i * 8 + k;
                cv::Point2f p(ORB_pattern[idx_pq * 4], ORB_pattern[idx_pq * 4 + 1]);
                cv::Point2f q(ORB_pattern[idx_pq * 4 + 2], ORB_pattern[idx_pq * 4 + 3]);

                // rotate with theta
                cv::Point2f pp = cv::Point2f(cos_theta * p.x - sin_theta * p.y, sin_theta * p.x + cos_theta * p.y) + kp.pt;
                cv::Point2f qq = cv::Point2f(cos_theta * q.x - sin_theta * q.y, sin_theta * q.x + cos_theta * q.y) + kp.pt;
                if (img.at<uchar>(pp.y, pp.x) < img.at<uchar>(qq.y, qq.x)) {
                    d |= 1 << k;
                }
            }
            desc[i] = d;
        }
        descriptors.push_back(desc);
    }

    cout << "bad/total: " << bad_points << "/" << keypoints.size() << endl;
}

// brute-force matching
void BfMatch(
    const vector<DescType> &desc1, const vector<DescType> &desc2, vector<cv::DMatch> &matches) {
    const int d_max = 40;

    for (size_t i1 = 0; i1 < desc1.size(); ++i1) {
        if (desc1[i1].empty()) continue;
        cv::DMatch m{i1, 0, 256};
        for (size_t i2 = 0; i2 < desc2.size(); ++i2) {
            if (desc2[i2].empty()) continue;
            int distance = 0;
            for (int k = 0; k < 8; k++) {
                distance += _mm_popcnt_u32(desc1[i1][k] ^desc2[i2][k]);
            }
```

```
            if (distance < d_max && distance < m.distance) {
                m.distance = distance;
                m.trainIdx = i2;
            }
        }
        if (m.distance < d_max) {
            matches.push_back(m);
        }
    }
}
```

ORB 256 8 32 unsigned int typedef DescType FAST mm_popcnt_u32 unsigned int 1 Figure 7-8

Listing 7.4:

```
bad/total: 43/638
bad/total: 8/595
extract ORB cost = 0.00390721 seconds.
match ORB cost = 0.000862984 seconds.
matches: 51
```

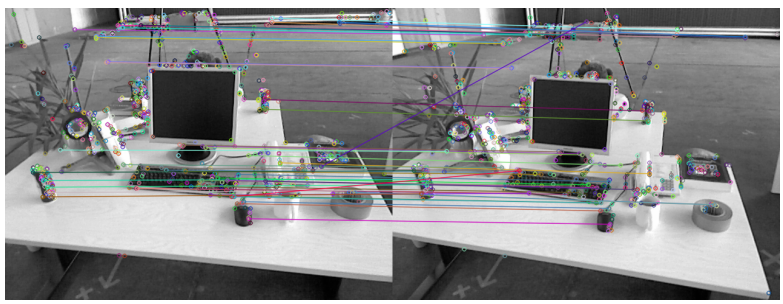


Figure 7-8:

ORB 3.9 0.86 ORB 5.8 CPU SSE CPU

7.2.3

1. 2D **2D**
2. RGB-D **3D** ICP
3. 3D 2D 3D **PnP**

2D−2D

7.3 2D−2D:

7.3.1

Figure 7-9 “ ”

Figure 7-9 I_1, I_2 $\boldsymbol{R}, \boldsymbol{t}$ O_1, O_2 I_1 p_1 I_2 p_2 **plane** O_1O_2 I_1, I_2 e_1, e_2 e_1, e_2 **Epipoles** O_1O_2 **Baseline** I_1, I_2 l_1, l_2 **Epipoline**

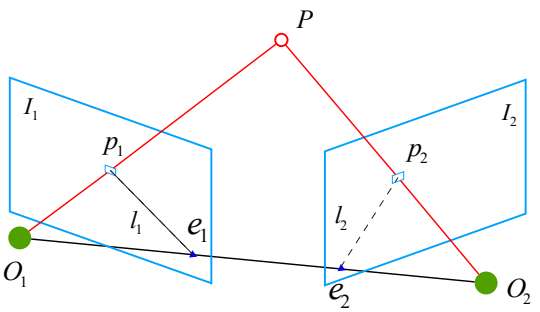


Figure 7-9:

$\overrightarrow{O_1p_1}$ —— P $\overrightarrow{e_2p_2}$ P $\overrightarrow{O_1p_1}$ P

$$\boldsymbol{P} = [X, Y, Z]^{\mathrm{T}}.$$

5 $\boldsymbol{p}_1, \boldsymbol{p}_2$

$$s_1\boldsymbol{p}_1 = \boldsymbol{K}\boldsymbol{P}, \quad s_2\boldsymbol{p}_2 = \boldsymbol{K}\left(\boldsymbol{R}\boldsymbol{P} + \boldsymbol{t}\right). \tag{7.1}$$

$\boldsymbol{K}$ $\boldsymbol{R}, \boldsymbol{t}$ $\boldsymbol{R}_{21}$ $\boldsymbol{t}_{21}$

$s_1\boldsymbol{p}_1$ $\boldsymbol{p}_1$ equal up to a scale

$$s\boldsymbol{p} \simeq \boldsymbol{p}. \tag{7.2}$$

$$\boldsymbol{p}_1 \simeq \boldsymbol{K}\boldsymbol{P}, \quad \boldsymbol{p}_2 \simeq \boldsymbol{K}\left(\boldsymbol{R}\boldsymbol{P} + \boldsymbol{t}\right). \tag{7.3}$$

$$\boldsymbol{x}_1 = \boldsymbol{K}^{-1}\boldsymbol{p}_1, \quad \boldsymbol{x}_2 = \boldsymbol{K}^{-1}\boldsymbol{p}_2. \tag{7.4}$$

$\boldsymbol{x}_1, \boldsymbol{x}_2$

$$\boldsymbol{x}_2 \simeq \boldsymbol{R}\boldsymbol{x}_1 + \boldsymbol{t}. \tag{7.5}$$

$\boldsymbol{t}^\wedge$ $\wedge$ $\boldsymbol{t}$

$$\boldsymbol{t}^\wedge\boldsymbol{x}_2 \simeq \boldsymbol{t}^\wedge\boldsymbol{R}\boldsymbol{x}_1. \tag{7.6}$$

$\boldsymbol{x}_2^{\mathrm{T}}$

$$\boldsymbol{x}_2^{\mathrm{T}}\boldsymbol{t}^\wedge\boldsymbol{x}_2 \simeq \boldsymbol{x}_2^{\mathrm{T}}\boldsymbol{t}^\wedge\boldsymbol{R}\boldsymbol{x}_1. \tag{7.7}$$

$\boldsymbol{t}^\wedge\boldsymbol{x}_2$ $\boldsymbol{t}$ $\boldsymbol{x}_2$ $\boldsymbol{x}_2$ 0 $\simeq$

$$\boldsymbol{x}_2^{\mathrm{T}}\boldsymbol{t}^\wedge\boldsymbol{R}\boldsymbol{x}_1 = 0. \tag{7.8}$$

$\boldsymbol{p}_1, \boldsymbol{p}_2$

$$\boldsymbol{p}_2^{\mathrm{T}}\boldsymbol{K}^{-\mathrm{T}}\boldsymbol{t}^\wedge\boldsymbol{R}\boldsymbol{K}^{-1}\boldsymbol{p}_1 = 0. \tag{7.9}$$

O_1, P, O_2 Fundamental Matrix $\boldsymbol{F}$ Essential Matrix $\boldsymbol{E}$

$$\boldsymbol{E} = \boldsymbol{t}^\wedge\boldsymbol{R}, \quad \boldsymbol{F} = \boldsymbol{K}^{-\mathrm{T}}\boldsymbol{E}\boldsymbol{K}^{-1}, \quad \boldsymbol{x}_2^{\mathrm{T}}\boldsymbol{E}\boldsymbol{x}_1 = \boldsymbol{p}_2^{\mathrm{T}}\boldsymbol{F}\boldsymbol{p}_1 = 0. \tag{7.10}$$

1. $\boldsymbol{E}$ $\boldsymbol{F}$

2. $\boldsymbol{E}$ $\boldsymbol{F}$ $\boldsymbol{R}, \boldsymbol{t}$

$\boldsymbol{E}$ $\boldsymbol{F}$ SLAM * $\boldsymbol{E}$ $\boldsymbol{E}$

7.3.2

$\boldsymbol{E} = \boldsymbol{t}^\wedge \boldsymbol{R}$ 3×3 9 3×3 $\boldsymbol{E}$

- $\boldsymbol{E}$ $\boldsymbol{E}$
- $\boldsymbol{E} = \boldsymbol{t}^\wedge \boldsymbol{R}$ [3] $\boldsymbol{E}$ $[\sigma, \sigma, 0]^\mathrm{T}$
- 3 $\boldsymbol{t}^\wedge \boldsymbol{R}$ 6 $\boldsymbol{E}$ 5

$\boldsymbol{E}$ 5 5 $\boldsymbol{E}$ $\boldsymbol{E}$ 8 $\boldsymbol{E}$—— **Eight-point-algorithm** [48, 49] $\boldsymbol{E}$

$$\boldsymbol{x}_1 = [u_1, v_1, 1]^\mathrm{T},\ \boldsymbol{x}_2 = [u_2, v_2, 1]^\mathrm{T}$$

$$\begin{pmatrix} u_2, v_2, 1 \end{pmatrix} \begin{pmatrix} e_1 & e_2 & e_3 \\ e_4 & e_5 & e_6 \\ e_7 & e_8 & e_9 \end{pmatrix} \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} = 0. \tag{7.11}$$

$\boldsymbol{E}$

$$\boldsymbol{e} = [e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9]^\mathrm{T},$$

$\boldsymbol{e}$

$$[u_2 u_1, u_2 v_1, u_2, v_2 u_1, v_2 v_1, v_2, u_1, v_1, 1] \cdot \boldsymbol{e} = 0. \tag{7.12}$$

u^i, v^i i

$$\begin{pmatrix} u_2^1 u_1^1 & u_2^1 v_1^1 & u_2^1 & v_2^1 u_1^1 & v_2^1 v_1^1 & v_2^1 & u_1^1 & v_1^1 & 1 \\ u_2^2 u_1^2 & u_2^2 v_1^2 & u_2^2 & v_2^2 u_1^2 & v_2^2 v_1^2 & v_2^2 & u_1^2 & v_1^2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ u_2^8 u_1^8 & u_2^8 v_1^8 & u_2^8 & v_2^8 u_1^8 & v_2^8 v_1^8 & v_2^8 & u_1^8 & v_1^8 & 1 \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \\ e_8 \\ e_9 \end{pmatrix} = 0. \tag{7.13}$$

8 8×9 $\boldsymbol{e}$ 8 1 $\boldsymbol{e}$ $\boldsymbol{e}$ 8 8 $\boldsymbol{E}$ $\boldsymbol{E}$ $\boldsymbol{R}, \boldsymbol{t}$ SVD $\boldsymbol{E}$ SVD

$$\boldsymbol{E} = \boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{V}^\mathrm{T}, \tag{7.14}$$

$\boldsymbol{U}, \boldsymbol{V}$ $\boldsymbol{\Sigma}$ $\boldsymbol{E}$ $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma, \sigma, 0)$ SVD $\boldsymbol{E}$ $\boldsymbol{t}, \boldsymbol{R}$

$$\begin{aligned} \boldsymbol{t}_1^\wedge &= \boldsymbol{U} \boldsymbol{R}_Z(\tfrac{}{2}) \boldsymbol{\Sigma} \boldsymbol{U}^\mathrm{T}, \quad \boldsymbol{R}_1 = \boldsymbol{U} \boldsymbol{R}_Z^\mathrm{T}(\tfrac{}{2}) \boldsymbol{V}^\mathrm{T} \\ \boldsymbol{t}_2^\wedge &= \boldsymbol{U} \boldsymbol{R}_Z(-\tfrac{}{2}) \boldsymbol{\Sigma} \boldsymbol{U}^\mathrm{T}, \quad \boldsymbol{R}_2 = \boldsymbol{U} \boldsymbol{R}_Z^\mathrm{T}(-\tfrac{}{2}) \boldsymbol{V}^\mathrm{T}. \end{aligned} \tag{7.15}$$

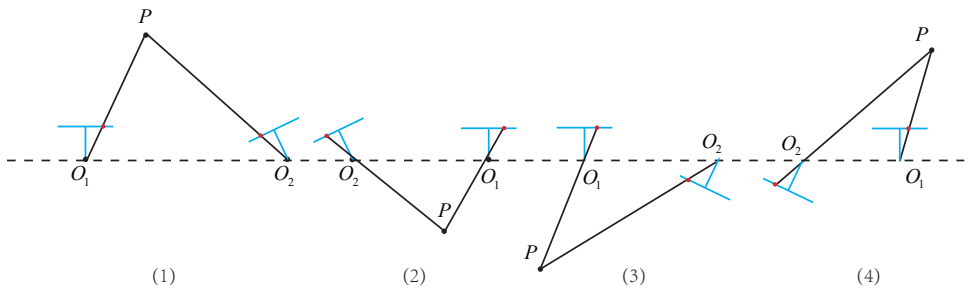


Figure 7-10: 4 4

$\boldsymbol{R}_Z(\frac{}{2})$ Z 90° $-\boldsymbol{E}$ $\boldsymbol{E}$ $\boldsymbol{t}$ $\boldsymbol{E}$ $\boldsymbol{t}, \boldsymbol{R}$ **4**
Figure 7-10 4 4 P 4
$\boldsymbol{E}$ 5 5 [50, 51] 8 5
$\boldsymbol{E}$ $\boldsymbol{E}$ —— $\sigma, \sigma, 0$ $\boldsymbol{\Sigma}$ $\boldsymbol{E}$ SVD $\boldsymbol{\Sigma} =$ $\mathrm{diag}(\sigma_1, \sigma_2, \sigma_3)$ $\sigma_1 \geqslant \sigma_2 \geqslant \sigma_3$

$$\boldsymbol{E} = \boldsymbol{U}\mathrm{diag}(\frac{\sigma_1 + \sigma_2}{2}, \frac{\sigma_1 + \sigma_2}{2}, 0)\boldsymbol{V}^{\mathrm{T}}. \tag{7.16}$$

$\boldsymbol{E}$ $\mathrm{diag}(1, 1, 0)$ $\boldsymbol{E}$

7.3.3

Homography $\boldsymbol{H}$
I_1 I_2 p_1 p_2 P

$$\boldsymbol{n}^{\mathrm{T}}\boldsymbol{P} + d = 0. \tag{7.17}$$

$$-\frac{\boldsymbol{n}^{\mathrm{T}}\boldsymbol{P}}{d} = 1. \tag{7.18}$$

(7.1)

$$\begin{aligned}
\boldsymbol{p}_2 &\simeq \boldsymbol{K}(\boldsymbol{R}\boldsymbol{P} + \boldsymbol{t}) \\
&\simeq \boldsymbol{K}\left(\boldsymbol{R}\boldsymbol{P} + \boldsymbol{t}\cdot(-\frac{\boldsymbol{n}^{\mathrm{T}}\boldsymbol{P}}{d})\right) \\
&\simeq \boldsymbol{K}\left(\boldsymbol{R} - \frac{\boldsymbol{t}\boldsymbol{n}^{\mathrm{T}}}{d}\right)\boldsymbol{P} \\
&\simeq \boldsymbol{K}\left(\boldsymbol{R} - \frac{\boldsymbol{t}\boldsymbol{n}^{\mathrm{T}}}{d}\right)\boldsymbol{K}^{-1}\boldsymbol{p}_1.
\end{aligned}$$

$\boldsymbol{p}_1$ $\boldsymbol{p}_2$ $\boldsymbol{H}$

$$\boldsymbol{p}_2 \simeq \boldsymbol{H}\boldsymbol{p}_1. \tag{7.19}$$

---

* SfM

$\boldsymbol{F}$ $\boldsymbol{H}$ 3×3 $\boldsymbol{F}$ $\boldsymbol{H}$

$$\begin{pmatrix} u_2 \\ v_2 \\ 1 \end{pmatrix} \simeq \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix} \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix}. \tag{7.20}$$

$\simeq$ $\boldsymbol{H}$ $h_9 = 1$ 3

$$u_2 = \frac{h_1 u_1 + h_2 v_1 + h_3}{h_7 u_1 + h_8 v_1 + h_9}$$
$$v_2 = \frac{h_4 u_1 + h_5 v_1 + h_6}{h_7 u_1 + h_8 v_1 + h_9}.$$

$$h_1 u_1 + h_2 v_1 + h_3 - h_7 u_1 u_2 - h_8 v_1 u_2 = u_2$$
$$h_4 u_1 + h_5 v_1 + h_6 - h_7 u_1 v_2 - h_8 v_1 v_2 = v_2.$$

8 4 $h_9 = 0$

$$\begin{pmatrix} u_1^1 & v_1^1 & 1 & 0 & 0 & 0 & -u_1^1 u_2^1 & -v_1^1 u_2^1 \\ 0 & 0 & 0 & u_1^1 & v_1^1 & 1 & -u_1^1 v_2^1 & -v_1^1 v_2^1 \\ u_1^2 & v_1^2 & 1 & 0 & 0 & 0 & -u_1^2 u_2^2 & -v_1^2 u_2^2 \\ 0 & 0 & 0 & u_1^2 & v_1^2 & 1 & -u_1^2 v_2^2 & -v_1^2 v_2^2 \\ u_1^3 & v_1^3 & 1 & 0 & 0 & 0 & -u_1^3 u_2^3 & -v_1^3 u_2^3 \\ 0 & 0 & 0 & u_1^3 & v_1^3 & 1 & -u_1^3 v_2^3 & -v_1^3 v_2^3 \\ u_1^4 & v_1^4 & 1 & 0 & 0 & 0 & -u_1^4 u_2^4 & -v_1^4 u_2^4 \\ 0 & 0 & 0 & u_1^4 & v_1^4 & 1 & -u_1^4 v_2^4 & -v_1^4 v_2^4 \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \end{pmatrix} = \begin{pmatrix} u_2^1 \\ v_2^1 \\ u_2^2 \\ v_2^2 \\ u_2^3 \\ v_2^3 \\ u_2^4 \\ v_2^4 \end{pmatrix}. \tag{7.21}$$

$\boldsymbol{H}$ $\boldsymbol{H}$ Direct Linear Transform $\boldsymbol{R}$ $\boldsymbol{t}$ [52, 53]
SLAM degenerate

7.4

$\boldsymbol{E}, \boldsymbol{F}$ $\boldsymbol{H}$ $\boldsymbol{E}$ $\boldsymbol{R}, \boldsymbol{t}$ OpenCV

Listing 7.5: slambook2/ch7/pose_estimation_2d2d.cpp

```
void pose_estimation_2d2d(std::vector<KeyPoint> keypoints_1,
    std::vector<KeyPoint> keypoints_2,
    std::vector<DMatch> matches,
    Mat &R, Mat &t) {
    // □□□□,TUM Freiburg2
    Mat K = (Mat_<double>(3, 3) << 520.9, 0, 325.1, 0, 521.0, 249.7, 0, 0, 1);

    //— □□□□□□□□vector<Point2f□□□>
    vector<Point2f> points1;
    vector<Point2f> points2;

    for (int i = 0; i < (int) matches.size(); i++) {
        points1.push_back(keypoints_1[matches[i].queryIdx].pt);
        points2.push_back(keypoints_2[matches[i].trainIdx].pt);
    }

    //— □□□□□□
    Mat fundamental_matrix;
    fundamental_matrix = findFundamentalMat(points1, points2, CV_FM_8POINT);
    cout << "fundamental_matrix is " << endl << fundamental_matrix << endl;
```

```
21
22      //-- 
23      Point2d principal_point(325.1, 249.7);  //, TUM dataset
24      double focal_length = 521;      //, TUM dataset
25      Mat essential_matrix;
26      essential_matrix = findEssentialMat(points1, points2, focal_length,
            principal_point);
27      cout << "essential_matrix is " << endl << essential_matrix << endl;
28
29      //-- 
30      //-- 
31      Mat homography_matrix;
32      homography_matrix = findHomography(points1, points2, RANSAC, 3);
33      cout << "homography_matrix is " << endl << homography_matrix << endl;
34
35      //-- .
36      recoverPose(essential_matrix, points1, points2, R, t, focal_length,
            principal_point);
37      cout << "R is " << endl << R << endl;
38      cout << "t is " << endl << t << endl;
39  }
```

Listing 7.6: slambook2/ch7/pose_estimation_2d2d.cpp

```
1   int main( int argc, char** argv ){
2       if (argc != 3) {
3           cout << "usage: pose_estimation_2d2d img1 img2" << endl;
4           return 1;
5       }
6       //-- 
7       Mat img_1 = imread(argv[1], CV_LOAD_IMAGE_COLOR);
8       Mat img_2 = imread(argv[2], CV_LOAD_IMAGE_COLOR);
9       assert(img_1.data && img_2.data && "Can not load images!");
10
11      vector<KeyPoint> keypoints_1, keypoints_2;
12      vector<DMatch> matches;
13      find_feature_matches(img_1, img_2, keypoints_1, keypoints_2, matches);
14      cout << "" << matches.size() << "" << endl;
15
16      //-- 
17      Mat R, t;
18      pose_estimation_2d2d(keypoints_1, keypoints_2, matches, R, t);
19
20      //-- E=t^R*scale
21      Mat t_x =
22          (Mat_<double>(3, 3) << 0, -t.at<double>(2, 0), t.at<double>(1, 0),
23          t.at<double>(2, 0), 0, -t.at<double>(0, 0),
24          -t.at<double>(1, 0), t.at<double>(0, 0), 0);
25      cout << "t^R=" << endl << t_x * R << endl;
26
27      //-- 
28      Mat K = (Mat_<double>(3, 3) << 520.9, 0, 325.1, 0, 521.0, 249.7, 0, 0, 1);
29      for (DMatch m: matches) {
30          Point2d pt1 = pixel2cam(keypoints_1[m.queryIdx].pt, K);
31          Mat y1 = (Mat_<double>(3, 1) << pt1.x, pt1.y, 1);
32          Point2d pt2 = pixel2cam(keypoints_2[m.trainIdx].pt, K);
33          Mat y2 = (Mat_<double>(3, 1) << pt2.x, pt2.y, 1);
34          Mat d = y2.t() * t_x * R * y1;
35          cout << "epipolar constraint = " << d << endl;
36      }
37      return 0;
38  }
```

$\boldsymbol{E}, \boldsymbol{F}\ \boldsymbol{H}$ $t\hat{}\boldsymbol{R}\ \boldsymbol{E}$

Listing 7.7:

```
1   % build/pose_estimation_2d2d 1.png 2.png
2   -- Max dist : 95.000000
```

```
-- Min dist : 4.000000□□□□□
 79 □□□□
fundamental_matrix is
[4.844484382466111e-06, 0.0001222601840188731, -0.01786737827487386;
-0.0001174326832719333, 2.122888800459598e-05, -0.01775877156212593;
0.01799658210895528, 0.008143605989020664, 1]
essential_matrix is
[-0.0203618550523477, -0.4007110038118445, -0.03324074249824097;
0.3939270778216369, -0.03506401846698079, 0.5857110303721015;
-0.006788487241438284, -0.5815434272915686, -0.01438258684486258]
homography_matrix is
[0.9497129583105288, -0.143556453147626, 31.20121878625771;
0.04154536627445031, 0.9715568969832015, 5.306887618807696;
-2.81813676978796e-05, 4.353702039810921e-05, 1]
R is
[0.9985961798781875, -0.05169917220143662, 0.01152671359827873;
0.05139607508976055, 0.9983603445075083, 0.02520051547522442;
-0.01281065954813571, -0.02457271064688495, 0.9996159607036126]
t is
[-0.8220841067933337;
-0.03269742706405412;
0.5684264241053522]

t^R=
[0.02879601157010516, 0.5666909361828478, 0.04700950886436416;
-0.5570970160413605, 0.0495880104673049, -0.8283204827837456;
0.009600370724838804, 0.8224266019846683, 0.02034004937801349]
epipolar constraint = [0.002528128704106625]
epipolar constraint = [-0.001663727901710724]
epipolar constraint = [-0.0008009088410884102]
......
```

10^{-3} $\boldsymbol{R}, \boldsymbol{t}$ 4 OpenCV

$\boldsymbol{E}$ $\boldsymbol{F}$ $\boldsymbol{E}, \boldsymbol{F}$ $\boldsymbol{H}$ $\boldsymbol{H}$ $\boldsymbol{E}$
$\boldsymbol{E}$ $\boldsymbol{t}, \boldsymbol{R}$ $\boldsymbol{R} \in \mathrm{SO}(3)$ $\boldsymbol{t}$ $\boldsymbol{t}$ $\boldsymbol{t}$ 1

$\boldsymbol{t}$ **Scale Ambiguity** $\boldsymbol{t}$ 0.822 0.822 0.822 0.822 $\boldsymbol{t}$
$\boldsymbol{t}$ $\boldsymbol{t}$ 1 3D SLAM 3D−2D
$\boldsymbol{t}$ 1 $\boldsymbol{t}$ 1

$\boldsymbol{E}$ $\boldsymbol{R}, \boldsymbol{t}$ $\boldsymbol{t}$ $\boldsymbol{E}$ $\boldsymbol{R}$ $\boldsymbol{H}$

8

8 79 (7.13) $\boldsymbol{A}$

$$\boldsymbol{A}\boldsymbol{e} = \boldsymbol{0}. \tag{7.22}$$

$\boldsymbol{A}$ 8×9 8 $\boldsymbol{e}$

$$\min_{\boldsymbol{e}} \|\boldsymbol{A}\boldsymbol{e}\|_2^2 = \min_{\boldsymbol{e}} \boldsymbol{e}^{\mathrm{T}} \boldsymbol{A}^{\mathrm{T}} \boldsymbol{A}\boldsymbol{e}. \tag{7.23}$$

$\boldsymbol{E}$ **Random Sample Concensus RANSAC** RANSAC

7.5

SLAM **Triangulation** P
11

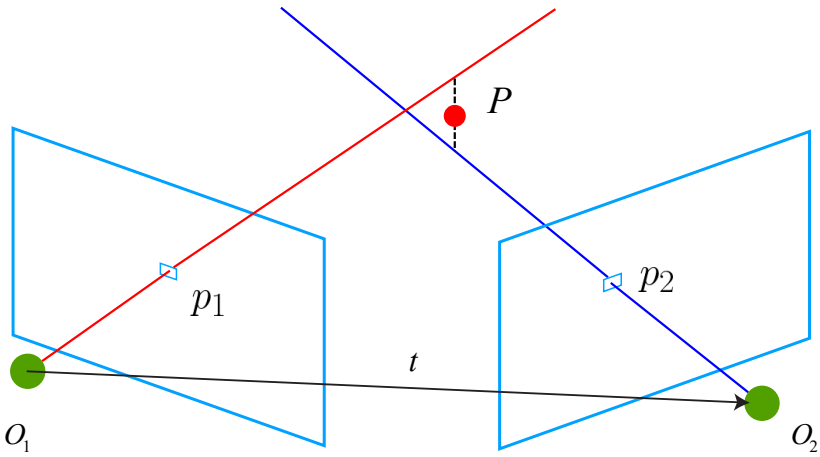


Figure 7-11:

SLAM
I_1 I_2 $\boldsymbol{T}$ O_1 O_2 I_1 p_1 I_2 p_2 O_1p_1 O_2p_2 P
$\boldsymbol{x}_1, \boldsymbol{x}_2$

$$s_2\boldsymbol{x}_2 = s_1\boldsymbol{R}\boldsymbol{x}_1 + \boldsymbol{t}. \tag{7.24}$$

$\boldsymbol{R}, \boldsymbol{t}$ s_1, s_2 O_1p_1 3D $\boldsymbol{p}_2$ O_2p_2 s_1

$$s_2\boldsymbol{x}_2^\wedge\boldsymbol{x}_2 = 0 = s_1\boldsymbol{x}_2^\wedge\boldsymbol{R}\boldsymbol{x}_1 + \boldsymbol{x}_2^\wedge\boldsymbol{t}. \tag{7.25}$$

s_2 s_2 s_2 s_1 $\boldsymbol{R}, \boldsymbol{t}$ (7.25)

7.6

7.6.1

OpenCV triangulation

Listing 7.8: slambook2/ch7/triangulation.cpp

```
void triangulation(
  const vector<KeyPoint> &keypoint_1,
  const vector<KeyPoint> &keypoint_2,
  const std::vector<DMatch> &matches,
  const Mat &R, const Mat &t,
  vector<Point3d> &points) {
  Mat T1 = (Mat_<float>(3, 4) <<
    1, 0, 0, 0,
    0, 1, 0, 0,
    0, 0, 1, 0);
  Mat T2 = (Mat_<float>(3, 4) <<
    R.at<double>(0, 0), R.at<double>(0, 1), R.at<double>(0, 2), t.at<double>(0, 0),
    R.at<double>(1, 0), R.at<double>(1, 1), R.at<double>(1, 2), t.at<double>(1, 0),
```

```
    R.at<double>(2, 0), R.at<double>(2, 1), R.at<double>(2, 2), t.at<double>(2, 0)
  );

  Mat K = (Mat_<double>(3, 3) << 520.9, 0, 325.1, 0, 521.0, 249.7, 0, 0, 1);
  vector<Point2f> pts_1, pts_2;
  for (DMatch m:matches) {
    // 
    pts_1.push_back(pixel2cam(keypoint_1[m.queryIdx].pt, K));
    pts_2.push_back(pixel2cam(keypoint_2[m.trainIdx].pt, K));
  }

  Mat pts_4d;
  cv::triangulatePoints(T1, T2, pts_1, pts_2, pts_4d);

  // 
  for (int i = 0; i < pts_4d.cols; i++) {
    Mat x = pts_4d.col(i);
    x /= x.at<float>(3, 0); // 
    Point3d p(
      x.at<float>(0, 0),
      x.at<float>(1, 0),
      x.at<float>(2, 0)
    );
    points.push_back(p);
  }
}
```

main

7.6.2

Figure 7-12 δx $\delta\theta$ δd $\boldsymbol{t}$ δd

— “ ” parallax

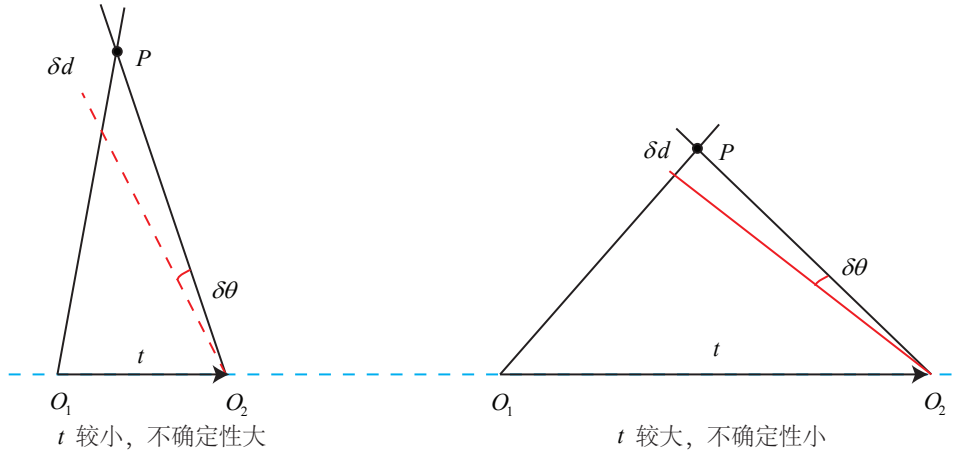


Figure 7-12:

[55]

Depth

Filter 3D−2D 3D−3D

7.7 3D—2D PnP

PnP Perspective-n-Point 3D 2D n 3D 2D—2D 8 8
D RGB-D PnP PnP 3D—2D
PnP 3 P3P[56] DLT EPnP Efficient PnP [57] UPnP[58] E
Adjustment DLT Bundle Adjustment

7.7.1

3D 3D
P $\boldsymbol{P} = (X, Y, Z, 1)^{\mathrm{T}}$ I_1 $\boldsymbol{x}_1 = (u_1, v_1, 1)^{\mathrm{T}}$ $\boldsymbol{R}, \boldsymbol{t}$ $[\boldsymbol{R}|\boldsymbol{t}]$ 3×
4 *

$$s \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} = \begin{pmatrix} t_1 & t_2 & t_3 & t_4 \\ t_5 & t_6 & t_7 & t_8 \\ t_9 & t_{10} & t_{11} & t_{12} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{7.26}$$

s

$$u_1 = \frac{t_1 X + t_2 Y + t_3 Z + t_4}{t_9 X + t_{10} Y + t_{11} Z + t_{12}}, \quad v_1 = \frac{t_5 X + t_6 Y + t_7 Z + t_8}{t_9 X + t_{10} Y + t_{11} Z + t_{12}}.$$

$\boldsymbol{T}$

$$\boldsymbol{t}_1 = (t_1, t_2, t_3, t_4)^{\mathrm{T}}, \boldsymbol{t}_2 = (t_5, t_6, t_7, t_8)^{\mathrm{T}}, \boldsymbol{t}_3 = (t_9, t_{10}, t_{11}, t_{12})^{\mathrm{T}},$$

$$\boldsymbol{t}_1^{\mathrm{T}} \boldsymbol{P} - \boldsymbol{t}_3^{\mathrm{T}} \boldsymbol{P} u_1 = 0,$$

$$\boldsymbol{t}_2^{\mathrm{T}} \boldsymbol{P} - \boldsymbol{t}_3^{\mathrm{T}} \boldsymbol{P} v_1 = 0.$$

$\boldsymbol{t}$ $\boldsymbol{t}$ N

$$\begin{pmatrix} \boldsymbol{P}_1^{\mathrm{T}} & 0 & -u_1 \boldsymbol{P}_1^{\mathrm{T}} \\ 0 & \boldsymbol{P}_1^{\mathrm{T}} & -v_1 \boldsymbol{P}_1^{\mathrm{T}} \\ \vdots & \vdots & \vdots \\ \boldsymbol{P}_N^{\mathrm{T}} & 0 & -u_N \boldsymbol{P}_N^{\mathrm{T}} \\ 0 & \boldsymbol{P}_N^{\mathrm{T}} & -v_N \boldsymbol{P}_N^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} \boldsymbol{t}_1 \\ \boldsymbol{t}_2 \\ \boldsymbol{t}_3 \end{pmatrix} = 0. \tag{7.27}$$

$\boldsymbol{t}$ 12 6 $\boldsymbol{T}$ Direct Linear Transform DLT 6 SVD
DLT $\boldsymbol{T}$ 12 $\boldsymbol{R} \in \mathrm{SO}(3)$ DLT $\boldsymbol{R}$ DLT $\boldsymbol{T}$ 3×
3 QR [3, 59] [6, 60]

$$\boldsymbol{R} \leftarrow \left(\boldsymbol{R}\boldsymbol{R}^{\mathrm{T}}\right)^{-\frac{1}{2}} \boldsymbol{R}. \tag{7.28}$$

SE(3)
$\boldsymbol{x}_1$ $\boldsymbol{K}$ —— $\boldsymbol{K}$ SLAM PnP $\boldsymbol{K}, \boldsymbol{R}, \boldsymbol{t}$

* SE(3) $\boldsymbol{T}$

7.7.2 P3P

P3P PnP 3 [61]
P3P 3 3 3D−2D 3D A, B, C 2D a, b, c Figure 7-
13 P3P $D-d$ O A, B, C 3D 3D

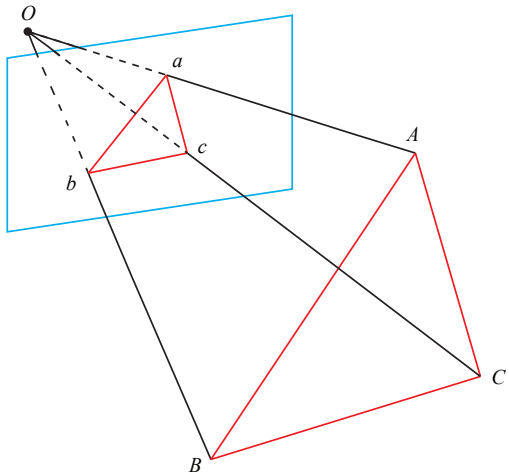


Figure 7-13: P3P

$$\Delta Oab - \Delta OAB, \quad \Delta Obc - \Delta OBC, \quad \Delta Oac - \Delta OAC. \tag{7.29}$$

Oab OAB

$$OA^2 + OB^2 - 2OA \cdot OB \cdot \cos\langle a, b\rangle = AB^2. \tag{7.30}$$

$$\begin{aligned} OA^2 + OB^2 - 2OA \cdot OB \cdot \cos\langle a, b\rangle &= AB^2 \\ OB^2 + OC^2 - 2OB \cdot OC \cdot \cos\langle b, c\rangle &= BC^2 \\ OA^2 + OC^2 - 2OA \cdot OC \cdot \cos\langle a, c\rangle &= AC^2. \end{aligned} \tag{7.31}$$

OC^2 $x = OA/OC, y = OB/OC$

$$\begin{aligned} x^2 + y^2 - 2xy \cos\langle a, b\rangle &= AB^2/OC^2 \\ y^2 + 1^2 - 2y \cos\langle b, c\rangle &= BC^2/OC^2 \\ x^2 + 1^2 - 2x \cos\langle a, c\rangle &= AC^2/OC^2. \end{aligned} \tag{7.32}$$

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

$$\begin{aligned} x^2 + y^2 - 2xy \cos\langle a, b\rangle - v &= 0 \\ y^2 + 1^2 - 2y \cos\langle b, c\rangle - uv &= 0 \\ x^2 + 1^2 - 2x \cos\langle a, c\rangle - wv &= 0. \end{aligned} \tag{7.33}$$

v

$$\begin{aligned} (1-u)\, y^2 - ux^2 - \cos\langle b, c\rangle\, y + 2uxy \cos\langle a, b\rangle + 1 &= 0 \\ (1-w)\, x^2 - wy^2 - \cos\langle a, c\rangle\, x + 2wxy \cos\langle a, b\rangle + 1 &= 0. \end{aligned} \tag{7.34}$$

2D 3 $\cos\langle a,b\rangle,\ \cos\langle b,c\rangle,\ \cos\langle a,c\rangle$ $u = BC^2/AB^2, w = AC^2/AB^2$ A,B,C x,y x,y
P3P PnP a,b,c 3D 3D 3D 3D−3D

1. P3P 3 3
2. 3D 2D

EPnP UPnP P3P PnP SLAM Adjustment PnP

7.7.3 PnP

PnP 4 5 PnP ICP Adjustment BA*

PnP Bundle Adjustment SLAM BA 9 BA n P p $\boldsymbol{R},\boldsymbol{t}$ $\boldsymbol{T}$ $\boldsymbol{P}_i = [X_i, Y_i, Z_i]^{\mathrm{T}}$ $\boldsymbol{u}_i = [u_i, v_i]^{\mathrm{T}}$ 5

$$s_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \boldsymbol{K}\boldsymbol{T} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}. \tag{7.35}$$

$$s_i \boldsymbol{u}_i = \boldsymbol{K}\boldsymbol{T}\boldsymbol{P}_i.$$

†

$$\boldsymbol{T}^* = \arg\min_{\boldsymbol{T}} \frac{1}{2} \sum_{i=1}^{n} \left\| \boldsymbol{u}_i - \frac{1}{s_i} \boldsymbol{K}\boldsymbol{T}\boldsymbol{P}_i \right\|_2^2. \tag{7.36}$$

3D 3 $\boldsymbol{u}$ 1 2 Figure 7-14 p_1 p_2 P P $\hat{p}_2$ p_2
6 — —

$$\boldsymbol{e}(\boldsymbol{x} + \Delta\boldsymbol{x}) \approx \boldsymbol{e}(\boldsymbol{x}) + \boldsymbol{J}^{\mathrm{T}} \Delta\boldsymbol{x}. \tag{7.37}$$

$\boldsymbol{J}^{\mathrm{T}}$ $\boldsymbol{e}$ 2 $\boldsymbol{x}$ 6 $\boldsymbol{J}^{\mathrm{T}}$ 2×6 $\boldsymbol{J}^{\mathrm{T}}$
$\boldsymbol{P}'$ 3

$$\boldsymbol{P}' = (\boldsymbol{T}\boldsymbol{P})_{1:3} = [X', Y', Z']^{\mathrm{T}}. \tag{7.38}$$

$\boldsymbol{P}'$

$$s\boldsymbol{u} = \boldsymbol{K}\boldsymbol{P}'. \tag{7.39}$$

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix}. \tag{7.40}$$

---

* BA BA BA BA BA BA PnP BA
† $\boldsymbol{T}\boldsymbol{P}_i$ 4×1 $\boldsymbol{K}$ 3×3 $\boldsymbol{T}\boldsymbol{P}_i$ $\boldsymbol{R}\boldsymbol{P}+\boldsymbol{t}$

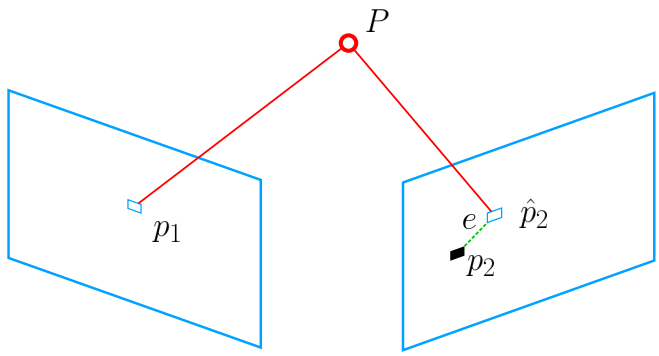


Figure 7-14:

3 s $\boldsymbol{P}'$

$$u = f_x \frac{X'}{Z'} + c_x, \quad v = f_y \frac{Y'}{Z'} + c_y. \tag{7.41}$$

5 u, v $\boldsymbol{T}$ $\delta\boldsymbol{\xi}$ $\boldsymbol{e}$

$$\frac{\partial \boldsymbol{e}}{\partial \delta \boldsymbol{\xi}} = \lim_{\delta \boldsymbol{\xi} \to 0} \frac{\boldsymbol{e}\left(\delta \boldsymbol{\xi} \oplus \boldsymbol{\xi}\right) - \boldsymbol{e}(\boldsymbol{\xi})}{\delta \boldsymbol{\xi}} = \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{P}'} \frac{\partial \boldsymbol{P}'}{\partial \delta \boldsymbol{\xi}}. \tag{7.42}$$

$\oplus$ (7.41)

$$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{P}'} = -\begin{bmatrix} \frac{\partial u}{\partial X'} & \frac{\partial u}{\partial Y'} & \frac{\partial u}{\partial Z'} \\ \frac{\partial v}{\partial X'} & \frac{\partial v}{\partial Y'} & \frac{\partial v}{\partial Z'} \end{bmatrix} = -\begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} \end{bmatrix}. \tag{7.43}$$

4.3.5

$$\frac{\partial (\boldsymbol{T}\boldsymbol{P})}{\partial \delta \boldsymbol{\xi}} = (\boldsymbol{T}\boldsymbol{P})^{\odot} = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{P}'^{\wedge} \\ \boldsymbol{0}^{\mathrm{T}} & \boldsymbol{0}^{\mathrm{T}} \end{bmatrix}. \tag{7.44}$$

$\boldsymbol{P}'$ 3

$$\frac{\partial \boldsymbol{P}'}{\partial \delta \boldsymbol{\xi}} = [\boldsymbol{I}, -\boldsymbol{P}'^{\wedge}]. \tag{7.45}$$

2×6

$$\frac{\partial \boldsymbol{e}}{\partial \delta \boldsymbol{\xi}} = -\begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} & -\frac{f_x X' Y'}{Z'^2} & f_x + \frac{f_x X'^2}{Z'^2} & -\frac{f_x Y'}{Z'} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} & -f_y - \frac{f_y Y'^2}{Z'^2} & \frac{f_y X' Y'}{Z'^2} & \frac{f_y X'}{Z'} \end{bmatrix}. \tag{7.46}$$

“ ” $\mathfrak{se}(3)$ 3 3

$\boldsymbol{e}$ $\boldsymbol{P}$

$$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{P}} = \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{P}'} \frac{\partial \boldsymbol{P}'}{\partial \boldsymbol{P}}. \tag{7.47}$$

$$\boldsymbol{P}' = (\boldsymbol{T}\boldsymbol{P})_{1:3} = \boldsymbol{R}\boldsymbol{P} + \boldsymbol{t},$$

$\boldsymbol{P}'$ $\boldsymbol{P}$ $\boldsymbol{R}$

$$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{P}} = -\begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} \end{bmatrix} \boldsymbol{R}. \tag{7.48}$$

7.8 PnP

7.8.1 EPnP

PnP OpenCV EPnP PnP PnP 3D RGB-D 1_depth.png 3D OpenCV PnP

Listing 7.9: slambook2/ch7/pose_estimation_3d2d.cpp

```
int main( int argc, char** argv ) {
    Mat r, t;
   solvePnP(pts_3d, pts_2d, K, Mat(), r, t, false); // OpenCV  PnP EPNPDLS
   Mat R;
   cv::Rodrigues(r, R); // rRodrigues
   cout << "R=" << endl << R << endl;
   cout << "t=" << endl << t << endl;
}
```

3D 2D EPnP PnP

Listing 7.10:

```
% build/pose_estimation_3d2d 1.png 2.png d1.png d2.png
-- Max dist : 95.000000
-- Min dist : 4.000000
79
3d-2d pairs: 76
R=
[0.9978662025826269, -0.05167241613316376, 0.03991244360207524;
0.0505958915956335, 0.998339762771668, 0.02752769192381471;
-0.04126860182960625, -0.025449547736074, 0.998823919929363]
t=
[-0.1272259656955879;
-0.007507297652615337;
0.06138584177157709]
```

2D−2D $\boldsymbol{R}, \boldsymbol{t}$ 3D $\boldsymbol{R}$ $\boldsymbol{t}$ Kinect 3D BA

7.8.2

PnP g2o

Listing 7.11: slambook2/ch7/pose_estimation_3d2d.cpp

```
void bundleAdjustmentGaussNewton(
const VecVector3d &points_3d,
const VecVector2d &points_2d,
const Mat &K,
Sophus::SE3d &pose) {
  typedef Eigen::Matrix<double, 6, 1> Vector6d;
  const int iterations = 10;
  double cost = 0, lastCost = 0;
  double fx = K.at<double>(0, 0);
  double fy = K.at<double>(1, 1);
  double cx = K.at<double>(0, 2);
  double cy = K.at<double>(1, 2);

  for (int iter = 0; iter < iterations; iter++) {
    Eigen::Matrix<double, 6, 6> H = Eigen::Matrix<double, 6, 6>::Zero();
    Vector6d b = Vector6d::Zero();

    cost = 0;
    // compute cost
    for (int i = 0; i < points_3d.size(); i++) {
      Eigen::Vector3d pc = pose * points_3d[i];
      double inv_z = 1.0 / pc[2];
      double inv_z2 = inv_z * inv_z;
      Eigen::Vector2d proj(fx * pc[0] / pc[2] + cx, fy * pc[1] / pc[2] + cy);
```

```
      Eigen::Vector2d e = points_2d[i] - proj;
      cost += e.squaredNorm();
      Eigen::Matrix<double, 2, 6> J;
      J << -fx * inv_z,
      0,
      fx * pc[0] * inv_z2,
      fx * pc[0] * pc[1] * inv_z2,
      -fx - fx * pc[0] * pc[0] * inv_z2,
      fx * pc[1] * inv_z,
      0,
      -fy * inv_z,
      fy * pc[1] * inv_z,
      fy + fy * pc[1] * pc[1] * inv_z2,
      -fy * pc[0] * pc[1] * inv_z2,
      -fy * pc[0] * inv_z;

      H += J.transpose() * J;
      b += -J.transpose() * e;
    }

    Vector6d dx;
    dx = H.ldlt().solve(b);

    if (isnan(dx[0])) {
      cout << "result is nan!" << endl;
      break;
    }

    if (iter > 0 && cost >= lastCost) {
      // cost increase, update is not good
      cout << "cost: " << cost << ", last cost: " << lastCost << endl;
      break;
    }

    // update your estimation
    pose = Sophus::SE3d::exp(dx) * pose;
    lastCost = cost;

    cout << "iteration " << iter << " cost=" << cout.precision(12) << cost << endl;
    if (dx.norm() < 1e-6) {
      // converge
      break;
    }
  }

  cout << "pose by g-n: \n" << pose.matrix() << endl;
}
```

OpenCV g2o

7.8.3 g2o BA

g2o Ceres g2o 6 g2o Figure 7-15

1. $\boldsymbol{T} \in \mathrm{SE}(3)$
2. 3D

$$\boldsymbol{z}_j = h(\boldsymbol{T}, \boldsymbol{P}_j).$$

3D 2D

g2o BA g2o/ types/sba/types_six_dof_expmap.h VertexPose EdgeProjection

Listing 7.12: slambook2/ch7/pose_estimation_3d2d.cpp

```
/// vertex and edges used in g2o ba
class VertexPose : public g2o::BaseVertex<6, Sophus::SE3d> {
```

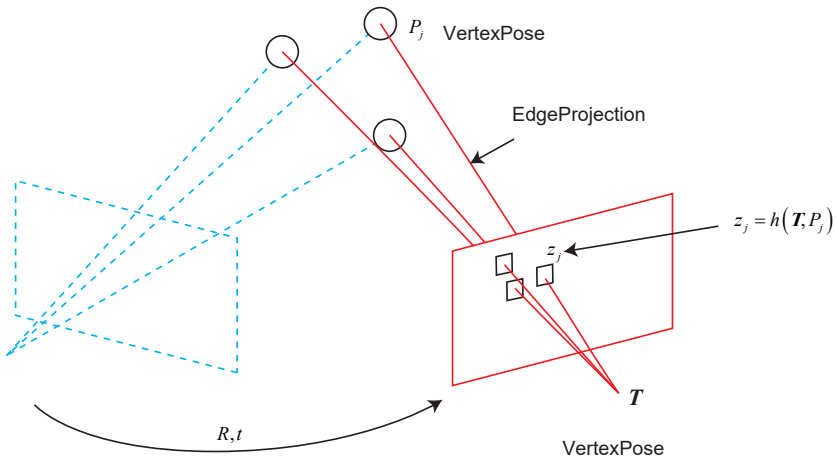


Figure 7-15: PnP Bundle Adjustment

```
  public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW;

  virtual void setToOriginImpl() override {
    _estimate = Sophus::SE3d();
  }

  /// left multiplication on SE3
  virtual void oplusImpl(const double *update) override {
    Eigen::Matrix<double, 6, 1> update_eigen;
    update_eigen << update[0], update[1], update[2], update[3], update[4], update[5];
    _estimate = Sophus::SE3d::exp(update_eigen) * _estimate;
  }

  virtual bool read(istream &in) override {}

  virtual bool write(ostream &out) const override {}
};

class EdgeProjection : public g2o::BaseUnaryEdge<2, Eigen::Vector2d, VertexPose> {
  public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW;

  EdgeProjection(const Eigen::Vector3d &pos, const Eigen::Matrix3d &K) : _pos3d(pos),
      _K(K) {}

  virtual void computeError() override {
    const VertexPose *v = static_cast<VertexPose *> (_vertices[0]);
    Sophus::SE3d T = v->estimate();
    Eigen::Vector3d pos_pixel = _K * (T * _pos3d);
    pos_pixel /= pos_pixel[2];
    _error = _measurement - pos_pixel.head<2>();
  }

  virtual void linearizeOplus() override {
    const VertexPose *v = static_cast<VertexPose *> (_vertices[0]);
    Sophus::SE3d T = v->estimate();
    Eigen::Vector3d pos_cam = T * _pos3d;
    double fx = _K(0, 0);
    double fy = _K(1, 1);
    double cx = _K(0, 2);
    double cy = _K(1, 2);
    double X = pos_cam[0];
    double Y = pos_cam[1];
    double Z = pos_cam[2];
    double Z2 = Z * Z;
```

```
    _jacobianOplusXi
    << -fx / Z, 0, fx * X / Z2, fx * X * Y / Z2, -fx - fx * X * X / Z2, fx * Y / Z,
    0, -fy / Z, fy * Y / (Z * Z), fy + fy * Y * Y / Z2, -fy * X * Y / Z2, -fy * X / Z;
  }

  virtual bool read(istream &in) override {}

  virtual bool write(ostream &out) const override {}

  private:
  Eigen::Vector3d _pos3d;
  Eigen::Matrix3d _K;
};
```

Listing 7.13: slambook2/ch7/pose_estimation_3d2d.cpp

```
void bundleAdjustmentG2O(
const VecVector3d &points_3d,
const VecVector2d &points_2d,
const Mat &K,
Sophus::SE3d &pose) {
  // □□□□□□□□□g2o
  typedef g2o::BlockSolver<g2o::BlockSolverTraits<6, 3>> BlockSolverType;  // pose is
      6, landmark is 3
  typedef g2o::LinearSolverDense<BlockSolverType::PoseMatrixType> LinearSolverType; //
       □□□□□□□□
  // □□□□□□□□□□□GN, LM, DogLeg □□
  auto solver = new g2o::OptimizationAlgorithmsGaussNewton(
    g2o::make_unique<BlockSolverType>(g2o::make_unique<LinearSolverType>()));
  g2o::SparseOptimizer optimizer;     // □□□
  optimizer.setAlgorithm(solver);   // □□□□□
  optimizer.setVerbose(true);       // □□□□□□

  // vertex
  VertexPose *vertex_pose = new VertexPose(); // camera vertex_pose
  vertex_pose->setId(0);
  vertex_pose->setEstimate(Sophus::SE3d());
  optimizer.addVertex(vertex_pose);

  // K
  Eigen::Matrix3d K_eigen;
  K_eigen <<
  K.at<double>(0, 0), K.at<double>(0, 1), K.at<double>(0, 2),
  K.at<double>(1, 0), K.at<double>(1, 1), K.at<double>(1, 2),
  K.at<double>(2, 0), K.at<double>(2, 1), K.at<double>(2, 2);

  // edges
  int index = 1;
  for (size_t i = 0; i < points_2d.size(); ++i) {
    auto p2d = points_2d[i];
    auto p3d = points_3d[i];
    EdgeProjection *edge = new EdgeProjection(p3d, K_eigen);
    edge->setId(index);
    edge->setVertex(0, vertex_pose);
    edge->setMeasurement(p2d);
    edge->setInformation(Eigen::Matrix2d::Identity());
    optimizer.addEdge(edge);
    index++;
  }

  chrono::steady_clock::time_point t1 = chrono::steady_clock::now();
  optimizer.setVerbose(true);
  optimizer.initializeOptimization();
  optimizer.optimize(10);
  chrono::steady_clock::time_point t2 = chrono::steady_clock::now();
  chrono::duration<double> time_used = chrono::duration_cast<chrono::duration<double
      >>(t2 - t1);
  cout << "optimization costs time: " << time_used.count() << " seconds." << endl;
  cout << "pose estimated by g2o =\n" << vertex_pose->estimate().matrix() << endl;
  pose = vertex_pose->estimate();
}
```

6 g2o g2o

Listing 7.14:

```
./build/pose_estimation_3d2d 1.png 2.png 1_depth.png 2_depth.png
-- Max dist : 95.000000
-- Min dist : 4.000000□□□□□□□□□
79
3d-2d pairs: 76
solve pnp in opencv cost time: 0.000332991 seconds.
R=
[0.9978662025826269, -0.05167241613316376, 0.03991244360207524;
0.0505958915956335, 0.998339762771668, 0.02752769192381471;
-0.04126860182960625, -0.025449547736074, 0.998823919929363]
t=
[-0.1272259656955879;
-0.007507297652615337;
0.06138584177157709]
calling bundle adjustment by gauss newton
iteration 0 cost=645538.1857253
iteration 1 cost=12750.239874896
iteration 2 cost=12301.774589343
iteration 3 cost=12301.427574651
iteration 4 cost=12301.426806652
pose by g-n:
0.99786618832  -0.0516873580423    0.039893448423   -0.127218696289
0.0506143671126    0.998340854865   0.0274540224544 -0.00738695798083
-0.0412462852904  -0.0253762590968    0.998826706403   0.0617019263823
0                 0                  0                1
solve pnp by gauss newton cost time: 0.000159492 seconds.
calling bundle adjustment by g2o
iteration= 0   chi2= 413.390599  time= 2.7291e-05  cumTime= 2.7291e-05   edges= 76
     schur= 0  lambda= 79.000412   levenbergIter= 1
iteration= 1   chi2= 301.367030  time= 1.47e-05  cumTime= 4.1991e-05   edges= 76
     schur= 0  lambda= 26.333471   levenbergIter= 1
iteration= 2   chi2= 301.365779  time= 1.7794e-05  cumTime= 5.9785e-05   edges= 76
     schur= 0  lambda= 17.555647   levenbergIter= 1
iteration= 3   chi2= 301.365779  time= 1.4875e-05  cumTime= 7.466e-05  edges= 76
     schur= 0  lambda= 11.703765   levenbergIter= 1
iteration= 4   chi2= 301.365779  time= 1.3132e-05  cumTime= 8.7792e-05   edges= 76
     schur= 0  lambda= 7.802510  levenbergIter= 1
iteration= 5   chi2= 301.365779  time= 2.0379e-05  cumTime= 0.000108171  edges= 76
     schur= 0  lambda= 41.613386   levenbergIter= 3
iteration= 6   chi2= 301.365779  time= 3.4186e-05  cumTime= 0.000142357  edges= 76
     schur= 0  lambda= 2859650082279.672363  levenbergIter= 8
optimization costs time: 0.000763649 seconds.
pose estimated by g2o =
0.997866202583  -0.0516724161336   0.0399124436024   -0.127225965696
0.050595891596    0.998339762772   0.0275276919261 -0.00750729765631
-0.04126860183  -0.0254495477384    0.998823919929   0.0613858417711
0                 0                  0                1
solve pnp by g2o cost time: 0.000923095 seconds.
```

0.15 OpenCV PnP g2o 1

Bundle Adjustment SLAM 10 Adjustment

7.9 3D−3D ICP

3D−3D 3D RGB-D

$$\boldsymbol{P} = \{\boldsymbol{p}_1, \cdots, \boldsymbol{p}_n\}, \quad \boldsymbol{P}' = \{\boldsymbol{p}'_1, \cdots, \boldsymbol{p}'_n\},$$

$\boldsymbol{R}, \boldsymbol{t}$ *

$$\forall i, \boldsymbol{p}_i = \boldsymbol{R}\boldsymbol{p}'_i + \boldsymbol{t}.$$

* $\boldsymbol{p}_i$ $\boldsymbol{p}'_i$ $\boldsymbol{R}, \boldsymbol{t}$

Iterative Closest Point ICP 3D−3D 3D SLAM IC
D SLAM ICP
PnP ICP SVD Bundle Adjustment

7.9.1 SVD

SVD ICP i

$$\boldsymbol{e}_i = \boldsymbol{p}_i - (\boldsymbol{R}\boldsymbol{p}_i' + \boldsymbol{t}). \tag{7.49}$$

$\boldsymbol{R}, \boldsymbol{t}$

$$\min_{\boldsymbol{R},\boldsymbol{t}} \frac{1}{2}\sum_{i=1}^{n} \|(\boldsymbol{p}_i - (\boldsymbol{R}\boldsymbol{p}_i{}' + \boldsymbol{t}))\|_2^2. \tag{7.50}$$

$$\boldsymbol{p} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{p}_i), \quad \boldsymbol{p}' = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{p}_i'). \tag{7.51}$$

$$\begin{aligned}
\tfrac{1}{2}\sum_{i=1}^{n} \|\boldsymbol{p}_i - (\boldsymbol{R}\boldsymbol{p}_i{}' + \boldsymbol{t})\|^2 &= \tfrac{1}{2}\sum_{i=1}^{n} \|\boldsymbol{p}_i - \boldsymbol{R}\boldsymbol{p}_i{}' - \boldsymbol{t} - \boldsymbol{p} + \boldsymbol{R}\boldsymbol{p}' + \boldsymbol{p} - \boldsymbol{R}\boldsymbol{p}'\|^2 \\
&= \tfrac{1}{2}\sum_{i=1}^{n} \|(\boldsymbol{p}_i - \boldsymbol{p} - \boldsymbol{R}(\boldsymbol{p}_i{}' - \boldsymbol{p}')) + (\boldsymbol{p} - \boldsymbol{R}\boldsymbol{p}' - \boldsymbol{t})\|^2 \\
&= \tfrac{1}{2}\sum_{i=1}^{n} (\|\boldsymbol{p}_i - \boldsymbol{p} - \boldsymbol{R}(\boldsymbol{p}_i{}' - \boldsymbol{p}')\|^2 + \|\boldsymbol{p} - \boldsymbol{R}\boldsymbol{p}' - \boldsymbol{t}\|^2 + \\
&\quad 2(\boldsymbol{p}_i - \boldsymbol{p} - \boldsymbol{R}(\boldsymbol{p}_i{}' - \boldsymbol{p}'))^T (\boldsymbol{p} - \boldsymbol{R}\boldsymbol{p}' - \boldsymbol{t})).
\end{aligned}$$

$(\boldsymbol{p}_i - \boldsymbol{p} - \boldsymbol{R}(\boldsymbol{p}_i{}' - \boldsymbol{p}'))$

$$\min_{\boldsymbol{R},\boldsymbol{t}} J = \frac{1}{2}\sum_{i=1}^{n} \|\boldsymbol{p}_i - \boldsymbol{p} - \boldsymbol{R}(\boldsymbol{p}_i{}' - \boldsymbol{p}')\|^2 + \|\boldsymbol{p} - \boldsymbol{R}\boldsymbol{p}' - \boldsymbol{t}\|^2. \tag{7.52}$$

$\boldsymbol{R}$ $\boldsymbol{R}$ $\boldsymbol{t}$ $\boldsymbol{R}$ $\boldsymbol{t}$ ICP

1. $\boldsymbol{p}, \boldsymbol{p}'$

$$\boldsymbol{q}_i = \boldsymbol{p}_i - \boldsymbol{p}, \quad \boldsymbol{q}_i' = \boldsymbol{p}_i' - \boldsymbol{p}'.$$

2.

$$\boldsymbol{R}^* = \arg\min_{\boldsymbol{R}} \frac{1}{2}\sum_{i=1}^{n} \|\boldsymbol{q}_i - \boldsymbol{R}\boldsymbol{q}_i'\|^2. \tag{7.53}$$

3. 2 $\boldsymbol{R}$ $\boldsymbol{t}$

$$\boldsymbol{t}^* = \boldsymbol{p} - \boldsymbol{R}\boldsymbol{p}'. \tag{7.54}$$

$\boldsymbol{R}$ $\boldsymbol{R}$

$$\frac{1}{2}\sum_{i=1}^{n} \|\boldsymbol{q}_i - \boldsymbol{R}\boldsymbol{q}_i'\|^2 = \frac{1}{2}\sum_{i=1}^{n} \boldsymbol{q}_i^{\mathrm{T}}\boldsymbol{q}_i + \boldsymbol{q}_i'^{\mathrm{T}}\boldsymbol{R}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{q}_i' - 2\boldsymbol{q}_i^{\mathrm{T}}\boldsymbol{R}\boldsymbol{q}_i'. \tag{7.55}$$

$\boldsymbol{R}$ $\boldsymbol{R}^{\mathrm{T}}\boldsymbol{R} = \boldsymbol{I}$ $\boldsymbol{R}$

$$\sum_{i=1}^{n} -\boldsymbol{q}_i^{\mathrm{T}}\boldsymbol{R}\boldsymbol{q}_i' = \sum_{i=1}^{n} -\mathrm{tr}\left(\boldsymbol{R}\boldsymbol{q}_i'\boldsymbol{q}_i^{\mathrm{T}}\right) = -\mathrm{tr}\left(\boldsymbol{R}\sum_{i=1}^{n}\boldsymbol{q}_i'\boldsymbol{q}_i^{\mathrm{T}}\right). \tag{7.56}$$

SVD $\boldsymbol{R}$ [62, 63] $\boldsymbol{R}$

$$\boldsymbol{W} = \sum_{i=1}^{n}\boldsymbol{q}_i\boldsymbol{q}_i'^{\mathrm{T}}. \tag{7.57}$$

$\boldsymbol{W}$ 3×3 $\boldsymbol{W}$ SVD

$$\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}. \tag{7.58}$$

$\boldsymbol{\Sigma}$ $\boldsymbol{U}$ $\boldsymbol{V}$ $\boldsymbol{W}$ $\boldsymbol{R}$

$$\boldsymbol{R} = \boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}. \tag{7.59}$$

$\boldsymbol{R}$ (7.54) $\boldsymbol{t}$ $\boldsymbol{R}$ $-\boldsymbol{R}$

7.9.2

ICP PnP

$$\min_{\boldsymbol{\xi}} = \frac{1}{2}\sum_{i=1}^{n}\left\|\left(\boldsymbol{p}_i - \exp\left(\boldsymbol{\xi}^{\wedge}\right)\boldsymbol{p}_i'\right)\right\|_2^2. \tag{7.60}$$

$$\frac{\partial \boldsymbol{e}}{\partial \delta\boldsymbol{\xi}} = -\left(\exp\left(\boldsymbol{\xi}^{\wedge}\right)\boldsymbol{p}_i'\right)^{\odot}. \tag{7.61}$$

[6] ICP —— ICP I

ICP [64–66] ICP ICP

D SLAM PnP ICP 3D−3D 3D−2D

7.10 ICP

7.10.1 SVD

SVD ICP RGB-D 3D ICP OpenCV ICP IC

Listing 7.15: slambook2/ch7/pose_estimation_3d3d.cpp

```
void pose_estimation_3d3d(
const vector<Point3f> &pts1,
const vector<Point3f> &pts2,
Mat &R, Mat &t) {
  Point3f p1, p2;     // center of mass
  int N = pts1.size();
  for (int i = 0; i < N; i++) {
    p1 += pts1[i];
    p2 += pts2[i];
  }
  p1 = Point3f(Vec3f(p1) / N);
  p2 = Point3f(Vec3f(p2) / N);
  vector<Point3f> q1(N), q2(N); // remove the center
  for (int i = 0; i < N; i++) {
    q1[i] = pts1[i] - p1;
    q2[i] = pts2[i] - p2;
```

```
  }

  // compute q1*q2^T
  Eigen::Matrix3d W = Eigen::Matrix3d::Zero();
  for (int i = 0; i < N; i++) {
    W += Eigen::Vector3d(q1[i].x, q1[i].y, q1[i].z) * Eigen::Vector3d(q2[i].x, q2[i].y
        , q2[i].z).transpose();
  }
  cout << "W=" << W << endl;

  // SVD on W
  Eigen::JacobiSVD<Eigen::Matrix3d> svd(W, Eigen::ComputeFullU | Eigen::ComputeFullV);
  Eigen::Matrix3d U = svd.matrixU();
  Eigen::Matrix3d V = svd.matrixV();

  cout << "U=" << U << endl;
  cout << "V=" << V << endl;

  Eigen::Matrix3d R_ = U * (V.transpose());
  if (R_.determinant() < 0) {
    R_ = -R_;
  }
  Eigen::Vector3d t_ = Eigen::Vector3d(p1.x, p1.y, p1.z) - R_ * Eigen::Vector3d(p2.x,
      p2.y, p2.z);

  // convert to cv::Mat
  R = (Mat_<double>(3, 3) <<
    R_(0, 0), R_(0, 1), R_(0, 2),
    R_(1, 0), R_(1, 1), R_(1, 2),
    R_(2, 0), R_(2, 1), R_(2, 2)
  );
  t = (Mat_<double>(3, 1) << t_(0, 0), t_(1, 0), t_(2, 0));
}
```

ICP Eigen SVD $\boldsymbol{R}, \boldsymbol{t}$ $\boldsymbol{p}_i = \boldsymbol{R}\boldsymbol{p}'_i + \boldsymbol{t}$ $\boldsymbol{R}, \boldsymbol{t}$ PnP

Listing 7.16:

```
./build/pose_estimation_3d3d 1.png 2.png 1_depth.png 2_depth.png
-- Max dist : 95.000000
-- Min dist : 4.000000
79
3d-3d pairs: 74
W=  11.9404 -0.567258   1.64182
-1.79283   4.31299  -6.57615
3.12791  -6.55815   10.8576
U=  0.474144  -0.880373 -0.0114952
-0.460275  -0.258979   0.849163
0.750556   0.397334   0.528006
V=  0.535211  -0.844064 -0.0332488
-0.434767  -0.309001    0.84587
0.724242   0.438263   0.532352
ICP via SVD results:
R = [0.9972395977366739, 0.05617039856770099, -0.04855997354553433;
-0.05598345194682017, 0.9984181427731508, 0.005202431117423125;
0.0487753812298326, -0.002469515369266572, 0.9988067198811421]
t = [0.1417248739257469;
-0.05551033302525193;
-0.03119093188273858]
R_inv = [0.9972395977366739, -0.05598345194682017, 0.0487753812298326;
0.05617039856770099, 0.9984181427731508, -0.002469515369266572;
-0.04855997354553433, 0.005202431117423125, 0.9988067198811421]
t_inv = [-0.1429199667309695;
0.04738475446275858;
0.03832465717628181]
```

ICP PnP — — Kinect

7.10.2

ICP RGB-D 3D VertexPose 3D-3D

Listing 7.17: slambook2/ch7/pose_estimation_3d3d.cpp

```
1
2  /// g2o edge
3  class EdgeProjectXYZRGBDPoseOnly : public g2o::BaseUnaryEdge<3, Eigen::Vector3d,
       VertexPose> {
4    public:
5    EIGEN_MAKE_ALIGNED_OPERATOR_NEW;
6
7    EdgeProjectXYZRGBDPoseOnly(const Eigen::Vector3d &point) : _point(point) {}
8
9    virtual void computeError() override {
10     const VertexPose *pose = static_cast<const VertexPose *> ( _vertices[0] );
11     _error = _measurement - pose->estimate() * _point;
12   }
13
14   virtual void linearizeOplus() override {
15     VertexPose *pose = static_cast<VertexPose *>(_vertices[0]);
16     Sophus::SE3d T = pose->estimate();
17     Eigen::Vector3d xyz_trans = T * _point;
18     _jacobianOplusXi.block<3, 3>(0, 0) = -Eigen::Matrix3d::Identity();
19     _jacobianOplusXi.block<3, 3>(0, 3) = Sophus::SO3d::hat(xyz_trans);
20   }
21
22   bool read(istream &in) {}
23
24   bool write(ostream &out) const {}
25
26   protected:
27   Eigen::Vector3d _point;
28 };
```

g2o::EdgeSE3ProjectXYZ 2 3 3×6

g2o

Listing 7.18:

```
1  iteration= 0   chi2= 1.811539  time= 1.7046e-05  cumTime= 1.7046e-05   edges= 74
       schur= 0
2  iteration= 1   chi2= 1.811051  time= 1.0422e-05  cumTime= 2.7468e-05   edges= 74
       schur= 0
3  iteration= 2   chi2= 1.811050  time= 9.589e-06   cumTime= 3.7057e-05   edges= 74
       schur= 0□□□
4  ...
5  iteration= 9   chi2= 1.811050  time= 9.113e-06   cumTime= 0.000100604  edges= 74
       schur= 0
6  optimization costs time: 0.000559208 seconds.
7
8  after optimization:
9  T=
10 0.99724  0.0561704   -0.04856   0.141725
11 -0.0559834   0.998418 0.00520242 -0.0555103
12 0.0487754 -0.0024695   0.998807 -0.0311913
13 0          0          0          1
```

SVD SVD SVD ICP PnP Adjustment

7.11

1.

2. 2D−2D

3. 2D−2D

4. 3D−2D PnP Bundle Adjustment

5. 3D−3D ICP Bundle Adjustment

$\boldsymbol{H}$ PnP ICP

1. ORB SIFT SURF ORB

2. OpenCV 1000

3.* OpenCV ORB

4. FLANN FLANN

5. EPnP PnP

6. PnP

7. ICP

8.* PnP ICP

9.* Sophus SE3 g2o PnP ICP

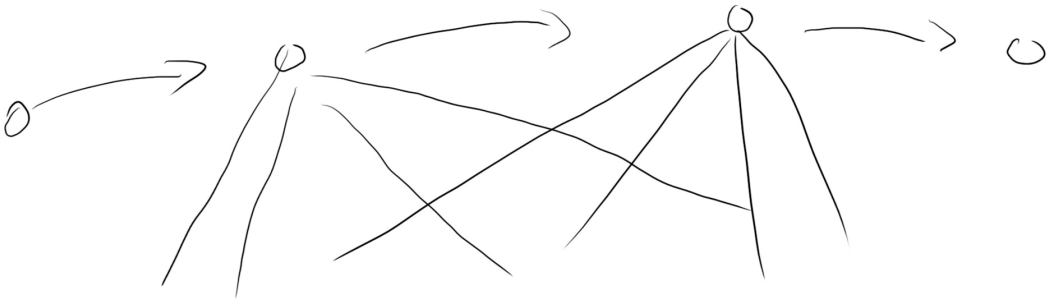
10.* Ceres PnP ICP

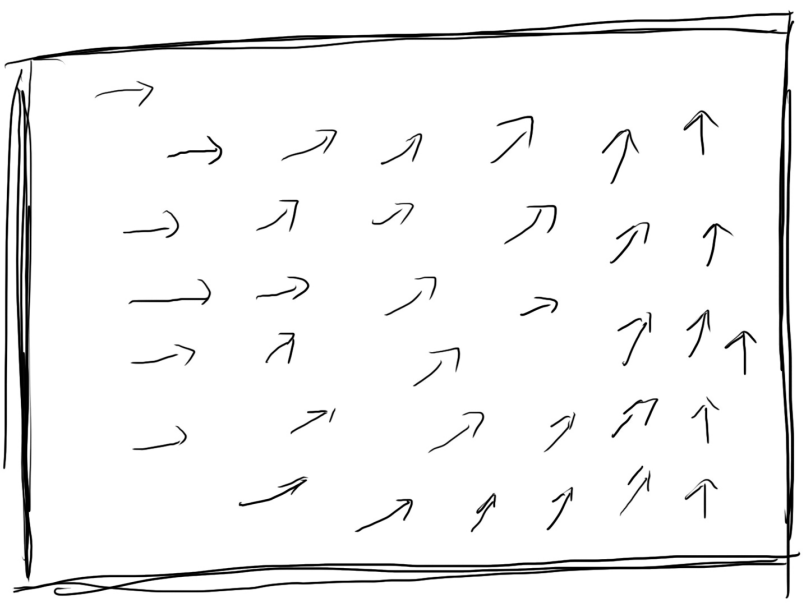
Chapter 8

2

1.
2.
3.

VO





$$\min \| I_1(p) - I_2(K(Rp+t)) \|_2$$

$$J = \frac{\partial I}{\partial u} \frac{\partial u}{\partial \xi} \frac{\partial \xi}{\partial \xi}$$

8.1

1. SIFT CPU ORB 20ms SLAM 30 /
2.
3.

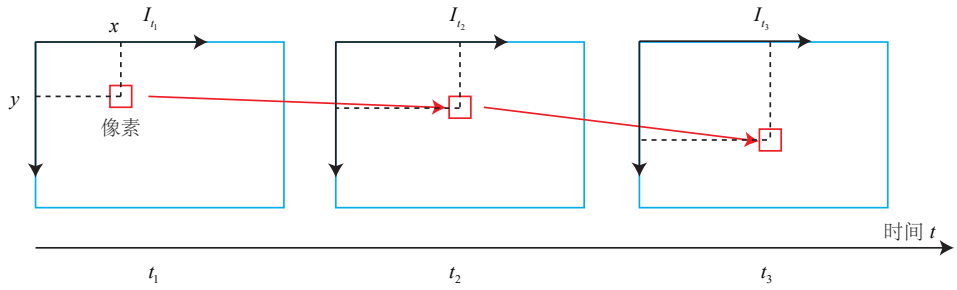
- Optical Flow
- Direct Method

PnP ICP
Reprojection error —
— Photometric error

[67] SVO[68] LSD-SLAM[69] DSO[70]

8.2 2D Optical Flow

Figure 8-1
Kanade [71] SLAM Horn-Schunck [72] Lucas-Kanade Lucas-
LK



灰度不变假设： $I(x_1,y_1,t_1)=I(x_2,y_2,t_2)=I(x_3,y_3,t_3)$

Figure 8-1: LK

Lucas-Kanade

LK $\boldsymbol{I}(t)$ t (x,y)

$$\boldsymbol{I}(x,y,t).$$

t x,y

t (x, y) $t + \mathrm{d}t$ $(x + \mathrm{d}x, y + \mathrm{d}y)$

$$\boldsymbol{I}(x + \mathrm{d}x, y + \mathrm{d}y, t + \mathrm{d}t) = \boldsymbol{I}(x, y, t). \tag{8.1}$$

$$\boldsymbol{I}\left(x + \mathrm{d}x, y + \mathrm{d}y, t + \mathrm{d}t\right) \approx \boldsymbol{I}\left(x, y, t\right) + \frac{\partial \boldsymbol{I}}{\partial x}\mathrm{d}x + \frac{\partial \boldsymbol{I}}{\partial y}\mathrm{d}y + \frac{\partial \boldsymbol{I}}{\partial t}\mathrm{d}t. \tag{8.2}$$

$$\frac{\partial \boldsymbol{I}}{\partial x}\mathrm{d}x + \frac{\partial \boldsymbol{I}}{\partial y}\mathrm{d}y + \frac{\partial \boldsymbol{I}}{\partial t}\mathrm{d}t = 0. \tag{8.3}$$

$\mathrm{d}t$

$$\frac{\partial \boldsymbol{I}}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial \boldsymbol{I}}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t} = -\frac{\partial \boldsymbol{I}}{\partial t}. \tag{8.4}$$

$\mathrm{d}x/\mathrm{d}t$ x $\mathrm{d}y/\mathrm{d}t$ y u, v $\partial \boldsymbol{I}/\partial x$ x y $\boldsymbol{I}_x, \boldsymbol{I}_y$ $\boldsymbol{I}_t$

$$\begin{bmatrix} \boldsymbol{I}_x & \boldsymbol{I}_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\boldsymbol{I}_t. \tag{8.5}$$

u, v u, v u, v LK
$w \times w$ w^2 w^2

$$\begin{bmatrix} \boldsymbol{I}_x & \boldsymbol{I}_y \end{bmatrix}_k \begin{bmatrix} u \\ v \end{bmatrix} = -\boldsymbol{I}_{tk}, \quad k = 1, \ldots, w^2. \tag{8.6}$$

$$\boldsymbol{A} = \begin{bmatrix} [\boldsymbol{I}_x, \boldsymbol{I}_y]_1 \\ \vdots \\ [\boldsymbol{I}_x, \boldsymbol{I}_y]_k \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} \boldsymbol{I}_{t1} \\ \vdots \\ \boldsymbol{I}_{tk} \end{bmatrix}. \tag{8.7}$$

$$\boldsymbol{A} \begin{bmatrix} u \\ v \end{bmatrix} = -\boldsymbol{b}. \tag{8.8}$$

u, v

$$\begin{bmatrix} u \\ v \end{bmatrix}^* = -\left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\right)^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}. \tag{8.9}$$

u, v t SLAM LK

8.3 LK

8.3.1 LK

OpenCV LK Euroc OpenCV L

Listing 8.1: slambook2/ch8/optical_flow.cpp

```
// use opencv's flow for validation
vector<Point2f> pt1, pt2;
for (auto &kp: kp1) pt1.push_back(kp.pt);
vector<uchar> status;
vector<float> error;
cv::calcOpticalFlowPyrLK(img1, img2, pt1, pt2, status, error);
```

OpenCV cv::calcOpticalFlowPyrLK status 1

8.3.2

Listing 8.2: slambook2/ch8/optical_flow.cpp

```
class OpticalFlowTracker {
public:
  OpticalFlowTracker(
    const Mat &img1_,
    const Mat &img2_,
    const vector<KeyPoint> &kp1_,
    vector<KeyPoint> &kp2_,
    vector<bool> &success_,
    bool inverse_ = true, bool has_initial_ = false) :
    img1(img1_), img2(img2_), kp1(kp1_), kp2(kp2_), success(success_), inverse(
        inverse_),
    has_initial(has_initial_) {}

  void calculateOpticalFlow(const Range &range);

private:
  const Mat &img1;
  const Mat &img2;
  const vector<KeyPoint> &kp1;
  vector<KeyPoint> &kp2;
  vector<bool> &success;
  bool inverse = true;
  bool has_initial = false;
};

void OpticalFlowSingleLevel(
  const Mat &img1,
  const Mat &img2,
  const vector<KeyPoint> &kp1,
  vector<KeyPoint> &kp2,
  vector<bool> &success,
  bool inverse, bool has_initial) {
  kp2.resize(kp1.size());
  success.resize(kp1.size());
  OpticalFlowTracker tracker(img1, img2, kp1, kp2, success, inverse, has_initial);
  parallel_for_(Range(0, kp1.size()),
    std::bind(&OpticalFlowTracker::calculateOpticalFlow, &tracker, placeholders::_1));
}

void OpticalFlowTracker::calculateOpticalFlow(const Range &range) {
  // parameters
  int half_patch_size = 4;
  int iterations = 10;
  for (size_t i = range.start; i < range.end; i++) {
    auto kp = kp1[i];
    double dx = 0, dy = 0; // dx,dy need to be estimated
    if (has_initial) {
      dx = kp2[i].pt.x - kp.pt.x;
      dy = kp2[i].pt.y - kp.pt.y;
    }

    double cost = 0, lastCost = 0;
    bool succ = true; // indicate if this point succeeded

    // Gauss-Newton iterations
    Eigen::Matrix2d H = Eigen::Matrix2d::Zero();    // hessian
    Eigen::Vector2d b = Eigen::Vector2d::Zero();    // bias
    Eigen::Vector2d J;  // jacobian
    for (int iter = 0; iter < iterations; iter++) {
      if (inverse == false) {
        H = Eigen::Matrix2d::Zero();
        b = Eigen::Vector2d::Zero();
      } else {
        // only reset b
        b = Eigen::Vector2d::Zero();
      }
```

```
66
67        cost = 0;
68
69        // compute cost and jacobian
70        for (int x = -half_patch_size; x < half_patch_size; x++)
71        for (int y = -half_patch_size; y < half_patch_size; y++) {
72          double error = GetPixelValue(img1, kp.pt.x + x, kp.pt.y + y) -
73            GetPixelValue(img2, kp.pt.x + x + dx, kp.pt.y + y + dy);;  // Jacobian
74          if (inverse == false) {
75            J = -1.0 * Eigen::Vector2d(
76              0.5 * (GetPixelValue(img2, kp.pt.x + dx + x + 1, kp.pt.y + dy + y) -
77                GetPixelValue(img2, kp.pt.x + dx + x - 1, kp.pt.y + dy + y)),
78              0.5 * (GetPixelValue(img2, kp.pt.x + dx + x, kp.pt.y + dy + y + 1) -
79                GetPixelValue(img2, kp.pt.x + dx + x, kp.pt.y + dy + y - 1))
80            );
81          } else if (iter == 0) {
82            // in inverse mode, J keeps same for all iterations
83            // NOTE this J does not change when dx, dy is updated, so we can store it
                  and only compute error
84            J = -1.0 * Eigen::Vector2d(
85              0.5 * (GetPixelValue(img1, kp.pt.x + x + 1, kp.pt.y + y) -
86                GetPixelValue(img1, kp.pt.x + x - 1, kp.pt.y + y)),
87              0.5 * (GetPixelValue(img1, kp.pt.x + x, kp.pt.y + y + 1) -
88                GetPixelValue(img1, kp.pt.x + x, kp.pt.y + y - 1))
89            );
90          }
91          // compute H, b and set cost;
92          b += -error * J;
93          cost += error * error;
94          if (inverse == false || iter == 0) {
95            // also update H
96            H += J * J.transpose();
97          }
98        }
99
100       // compute update
101       Eigen::Vector2d update = H.ldlt().solve(b);
102
103       if (std::isnan(update[0])) {
104         // sometimes occurred when we have a black or white patch and H is
                irreversible
105         cout << "update is nan" << endl;
106         succ = false;
107         break;
108       }
109
110       if (iter > 0 && cost > lastCost) {
111         break;
112       }
113
114       // update dx, dy
115       dx += update[0];
116       dy += update[1];
117       lastCost = cost;
118       succ = true;
119
120       if (update.norm() < 1e-2) {
121         // converge
122         break;
123       }
124     }
125
126     success[i] = succ;
127
128     // set kp2
129     kp2[i].pt = kp.pt + Point2f(dx, dy);
130   }
131 }
```

OpticalFlowSingleLevel cv::parallel_for_ OpticalFlowTracker::calculateOpticalFlow tbb std::function

calculateOpticalFlow

$$\min_{\Delta x,\Delta y} \|\boldsymbol{I}_1(x,y) - \boldsymbol{I}_2(x+\Delta x, y+\Delta y)\|_2^2 . \tag{8.10}$$

$x+\Delta x, y+\Delta y$ [73] $\boldsymbol{I}_1(x,y)$ Inverse $\boldsymbol{I}_1(x,y)$

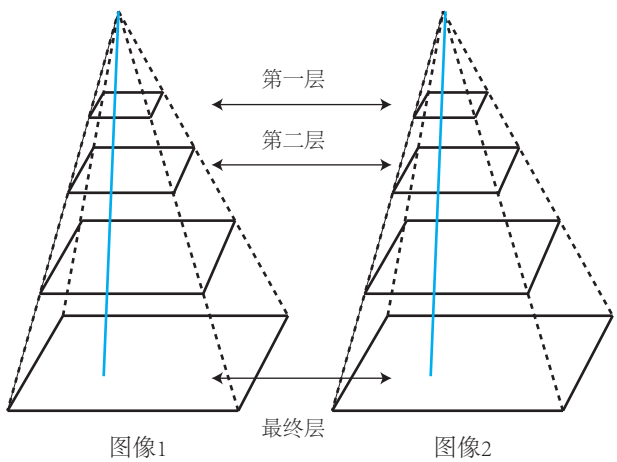


Figure 8-2: Coarse-to-fine

Figure 8-2

to-fine

20 0.5 5

Listing 8.3: slambook2/ch8/optical_flow.cpp

```
void OpticalFlowMultiLevel(
  const Mat &img1,
  const Mat &img2,
  const vector<KeyPoint> &kp1,
  vector<KeyPoint> &kp2,
  vector<bool> &success,
  bool inverse) {

  // parameters
  int pyramids = 4;
  double pyramid_scale = 0.5;
  double scales[] = {1.0, 0.5, 0.25, 0.125};

  // create pyramids
  vector<Mat> pyr1, pyr2; // image pyramids
  for (int i = 0; i < pyramids; i++) {
    if (i == 0) {
      pyr1.push_back(img1);
      pyr2.push_back(img2);
    } else {
      Mat img1_pyr, img2_pyr;
      cv::resize(pyr1[i - 1], img1_pyr,
      cv::Size(pyr1[i - 1].cols * pyramid_scale, pyr1[i - 1].rows * pyramid_scale));
      cv::resize(pyr2[i - 1], img2_pyr,
      cv::Size(pyr2[i - 1].cols * pyramid_scale, pyr2[i - 1].rows * pyramid_scale));
```

```
      pyr1.push_back(img1_pyr);
      pyr2.push_back(img2_pyr);
    }
  }

  // coarse-to-fine LK tracking in pyramids
  vector<KeyPoint> kp1_pyr, kp2_pyr;
  for (auto &kp:kp1) {
    auto kp_top = kp;
    kp_top.pt *= scales[pyramids - 1];
    kp1_pyr.push_back(kp_top);
    kp2_pyr.push_back(kp_top);
  }

  for (int level = pyramids - 1; level >= 0; level--) {
    // from coarse to fine
    success.clear();
    OpticalFlowSingleLevel(pyr1[level], pyr2[level], kp1_pyr, kp2_pyr, success,
        inverse, true);

    if (level > 0) {
      for (auto &kp: kp1_pyr)
      kp.pt /= pyramid_scale;
      for (auto &kp: kp2_pyr)
      kp.pt /= pyramid_scale;
    }
  }

  for (auto &kp: kp2_pyr)
    kp2.push_back(kp);
}
```

0.5 OpenCV

Listing 8.4:

```
./build/optical_flow
build pyramid time: 0.000150349
track pyr 3 cost time: 0.000304633
track pyr 2 cost time: 0.000392889
track pyr 1 cost time: 0.000382347
track pyr 0 cost time: 0.000375099
optical flow by gauss-newton: 0.00189268
optical flow by opencv: 0.00220134
```

OpenCV Figure 8-3 OpenCV

8.3.3

LK PnP ICP

230 OpenCV 2 CPU Intel I7-8550U FAST 5

8.4 Direct Method

8.4.1

15 15

Figure 8-4 P P $[X, Y, Z]$ $\boldsymbol{p}_1, \boldsymbol{p}_2$

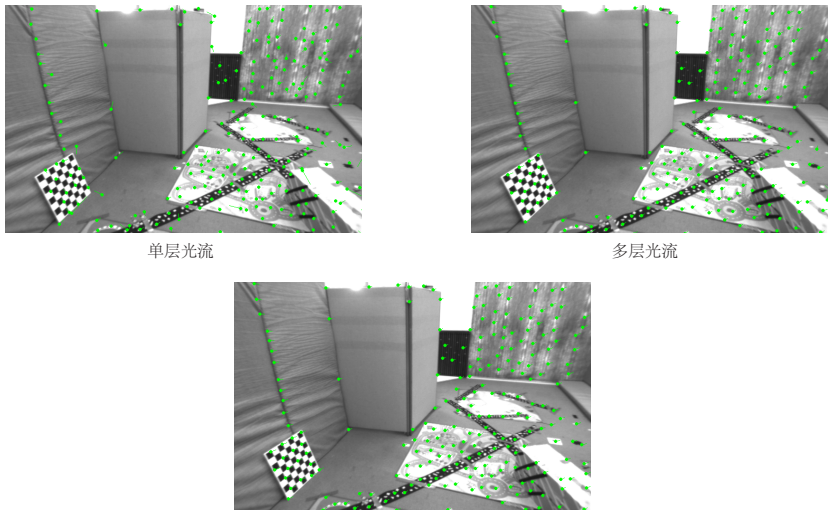


Figure 8-3:

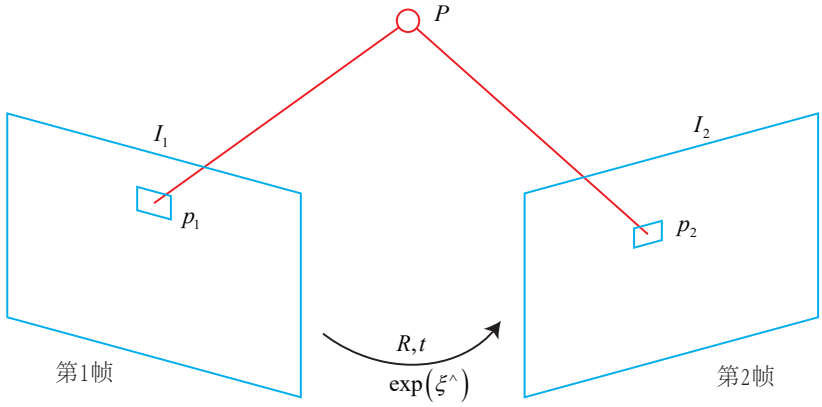


Figure 8-4:

$\boldsymbol{R}, \boldsymbol{t}$ $\boldsymbol{T}$ $\boldsymbol{K}$

$$\boldsymbol{p}_1 = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_1 = \frac{1}{Z_1}\boldsymbol{K}\boldsymbol{P},$$
$$\boldsymbol{p}_2 = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_2 = \frac{1}{Z_2}\boldsymbol{K}\left(\boldsymbol{R}\boldsymbol{P} + \boldsymbol{t}\right) = \frac{1}{Z_2}\boldsymbol{K}\left(\boldsymbol{T}\boldsymbol{P}\right)_{1:3}.$$

Z_1 P Z_2 P $\boldsymbol{R}\boldsymbol{P} + \boldsymbol{t}$ 3 $\boldsymbol{T}$ 3 5

$\boldsymbol{p}_1, \boldsymbol{p}_2$ $\boldsymbol{p}_2$ $\boldsymbol{p}_1$ $\boldsymbol{p}_2$ $\boldsymbol{p}_2$ $\boldsymbol{p}_1$

Error P

$$e = \boldsymbol{I}_1\left(\boldsymbol{p}_1\right) - \boldsymbol{I}_2\left(\boldsymbol{p}_2\right). \tag{8.11}$$

e

$$\min_{\boldsymbol{T}} J\left(\boldsymbol{T}\right) = \|e\|^2. \tag{8.12}$$

N P_i

$$\min_{\boldsymbol{T}} J\left(\boldsymbol{T}\right) = \sum_{i=1}^{N} e_i^{\mathrm{T}} e_i, \quad e_i = \boldsymbol{I}_1\left(\boldsymbol{p}_{1,i}\right) - \boldsymbol{I}_2\left(\boldsymbol{p}_{2,i}\right). \tag{8.13}$$

$\boldsymbol{T}$ e $\boldsymbol{T}$

$$\boldsymbol{q} = \boldsymbol{T}\boldsymbol{P},$$
$$\boldsymbol{u} = \frac{1}{Z_2}\boldsymbol{K}\boldsymbol{q}.$$

$\boldsymbol{q}$ P $\boldsymbol{u}$ $\boldsymbol{q}$ $\boldsymbol{T}$ $\boldsymbol{u}$ $\boldsymbol{q}$ $\boldsymbol{T}$

$$e(\boldsymbol{T}) = \boldsymbol{I}_1(\boldsymbol{p}_1) - \boldsymbol{I}_2(\boldsymbol{u}), \tag{8.14}$$

$$\frac{\partial e}{\partial \boldsymbol{T}} = \frac{\partial \boldsymbol{I}_2}{\partial \boldsymbol{u}} \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{q}} \frac{\partial \boldsymbol{q}}{\partial \delta \boldsymbol{\xi}} \delta \boldsymbol{\xi}, \tag{8.15}$$

$\delta \boldsymbol{\xi}$ $\boldsymbol{T}$ 3 3

1. $\partial \boldsymbol{I}_2 / \partial \boldsymbol{u}$ $\boldsymbol{u}$
2. $\partial \boldsymbol{u} / \partial \boldsymbol{q}$ $\boldsymbol{q} = [X, Y, Z]^{\mathrm{T}}$ 7

$$\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{q}} = \begin{bmatrix} \frac{\partial u}{\partial X} & \frac{\partial u}{\partial Y} & \frac{\partial u}{\partial Z} \\ \frac{\partial v}{\partial X} & \frac{\partial v}{\partial Y} & \frac{\partial v}{\partial Z} \end{bmatrix} = \begin{bmatrix} \frac{f_x}{\mathrm{Z}} & 0 & -\frac{f_x X}{Z^2} \\ 0 & \frac{f_y}{Z} & -\frac{f_y Y}{Z^2} \end{bmatrix}. \tag{8.16}$$

3. $\partial \boldsymbol{q} / \partial \delta \boldsymbol{\xi}$

$$\frac{\partial \boldsymbol{q}}{\partial \delta \boldsymbol{\xi}} = \left[\boldsymbol{I}, -\boldsymbol{q}^{\wedge}\right]. \tag{8.17}$$

$\boldsymbol{q}$

$$\frac{\partial \boldsymbol{u}}{\partial \delta \boldsymbol{\xi}} = \begin{bmatrix} \frac{f_x}{Z} & 0 & -\frac{f_x X}{Z^2} & -\frac{f_x XY}{Z^2} & f_x + \frac{f_x X^2}{Z^2} & -\frac{f_x Y}{Z} \\ 0 & \frac{f_y}{Z} & -\frac{f_y Y}{Z^2} & -f_y - \frac{f_y Y^2}{Z^2} & \frac{f_y XY}{Z^2} & \frac{f_y X}{Z} \end{bmatrix}. \tag{8.18}$$

2×6

$$\boldsymbol{J} = -\frac{\partial \boldsymbol{I}_2}{\partial \boldsymbol{u}} \frac{\partial \boldsymbol{u}}{\partial \delta \boldsymbol{\xi}}. \tag{8.19}$$

N —

8.4.2

P RGB-D P 13
P

1. P L-K
2. P (8.19) Semi-Dense
3. P CPU GPU

[70]

8.5

8.5.1

GPU
g2o Ceres

Listing 8.5: slambook2/ch8/direct_method.cpp

```
/// class for accumulator jacobians in parallel
class JacobianAccumulator {
public:
  JacobianAccumulator(
    const cv::Mat &img1_,
    const cv::Mat &img2_,
    const VecVector2d &px_ref_,
    const vector<double> depth_ref_,
    Sophus::SE3d &T21_) :
  img1(img1_), img2(img2_), px_ref(px_ref_), depth_ref(depth_ref_), T21(T21_) {
    projection = VecVector2d(px_ref.size(), Eigen::Vector2d(0, 0));
  }

  /// accumulate jacobians in a range
  void accumulate_jacobian(const cv::Range &range);

  /// get hessian matrix
  Matrix6d hessian() const { return H; }

  /// get bias
  Vector6d bias() const { return b; }

  /// get total cost
  double cost_func() const { return cost; }

```

```
  /// get projected points
  VecVector2d projected_points() const { return projection; }

  /// reset h, b, cost to zero
  void reset() {
    H = Matrix6d::Zero();
    b = Vector6d::Zero();
    cost = 0;
  }

private:
  const cv::Mat &img1;
  const cv::Mat &img2;
  const VecVector2d &px_ref;
  const vector<double> depth_ref;
  Sophus::SE3d &T21;
  VecVector2d projection; // projected points

  std::mutex hessian_mutex;
  Matrix6d H = Matrix6d::Zero();
  Vector6d b = Vector6d::Zero();
  double cost = 0;
};

void JacobianAccumulator::accumulate_jacobian(const cv::Range &range) {

  // parameters
  const int half_patch_size = 1;
  int cnt_good = 0;
  Matrix6d hessian = Matrix6d::Zero();
  Vector6d bias = Vector6d::Zero();
  double cost_tmp = 0;

  for (size_t i = range.start; i < range.end; i++) {
    // compute the projection in the second image
    Eigen::Vector3d point_ref =
    depth_ref[i] * Eigen::Vector3d((px_ref[i][0] - cx) / fx, (px_ref[i][1] - cy) / fy,
          1);
    Eigen::Vector3d point_cur = T21 * point_ref;
    if (point_cur[2] < 0)   // depth invalid
      continue;

    float u = fx * point_cur[0] / point_cur[2] + cx, v = fy * point_cur[1] / point_cur
         [2] + cy;
    if (u < half_patch_size || u > img2.cols - half_patch_size || v < half_patch_size
          ||
    v > img2.rows - half_patch_size)
      continue;

    projection[i] = Eigen::Vector2d(u, v);
    double X = point_cur[0], Y = point_cur[1], Z = point_cur[2],
    Z2 = Z * Z, Z_inv = 1.0 / Z, Z2_inv = Z_inv * Z_inv;
    cnt_good++;

    // and compute error and jacobian
    for (int x = -half_patch_size; x <= half_patch_size; x++)
    for (int y = -half_patch_size; y <= half_patch_size; y++) {
      double error = GetPixelValue(img1, px_ref[i][0] + x, px_ref[i][1] + y) -
        GetPixelValue(img2, u + x, v + y);
      Matrix26d J_pixel_xi;
      Eigen::Vector2d J_img_pixel;

      J_pixel_xi(0, 0) = fx * Z_inv;
      J_pixel_xi(0, 1) = 0;
      J_pixel_xi(0, 2) = -fx * X * Z2_inv;
      J_pixel_xi(0, 3) = -fx * X * Y * Z2_inv;
      J_pixel_xi(0, 4) = fx + fx * X * X * Z2_inv;
      J_pixel_xi(0, 5) = -fx * Y * Z_inv;

      J_pixel_xi(1, 0) = 0;
      J_pixel_xi(1, 1) = fy * Z_inv;
      J_pixel_xi(1, 2) = -fy * Y * Z2_inv;
      J_pixel_xi(1, 3) = -fy - fy * Y * Y * Z2_inv;
      J_pixel_xi(1, 4) = fy * X * Y * Z2_inv;
```

```
97        J_pixel_xi(1, 5) = fy * X * Z_inv;
98
99        J_img_pixel = Eigen::Vector2d(
100         0.5 * (GetPixelValue(img2, u + 1 + x, v + y) - GetPixelValue(img2, u - 1 + x,
                v + y)),
101         0.5 * (GetPixelValue(img2, u + x, v + 1 + y) - GetPixelValue(img2, u + x, v -
                1 + y))
102       );
103
104       // total jacobian
105       Vector6d J = -1.0 * (J_img_pixel.transpose() * J_pixel_xi).transpose();
106       hessian += J * J.transpose();
107       bias += -error * J;
108       cost_tmp += error * error;
109     }
110   }
111
112   if (cnt_good) {
113     // set hessian, bias and cost
114     unique_lock<mutex> lck(hessian_mutex);
115     H += hessian;
116     b += bias;
117     cost += cost_tmp / cnt_good;
118   }
119 }
```

accumulate_jacobian $\boldsymbol{H}$

Listing 8.6: slambook2/ch8/direct_method.cpp

```
1  void DirectPoseEstimationSingleLayer(
2    const cv::Mat &img1,
3    const cv::Mat &img2,
4    const VecVector2d &px_ref,
5    const vector<double> depth_ref,
6    Sophus::SE3d &T21) {
7    const int iterations = 10;
8    double cost = 0, lastCost = 0;
9    JacobianAccumulator jaco_accu(img1, img2, px_ref, depth_ref, T21);
10
11   for (int iter = 0; iter < iterations; iter++) {
12     jaco_accu.reset();
13     cv::parallel_for_(cv::Range(0, px_ref.size()),
14       std::bind(&JacobianAccumulator::accumulate_jacobian, &jaco_accu, std::
             placeholders::_1));
15     Matrix6d H = jaco_accu.hessian();
16     Vector6d b = jaco_accu.bias();
17
18     // solve update and put it into estimation
19     Vector6d update = H.ldlt().solve(b);;
20     T21 = Sophus::SE3d::exp(update) * T21;
21     cost = jaco_accu.cost_func();
22
23     if (std::isnan(update[0])) {
24       // sometimes occurred when we have a black or white patch and H is irreversible
25       cout << "update is nan" << endl;
26       break;
27     }
28     if (iter > 0 && cost > lastCost) {
29       cout << "cost increased: " << cost << ", " << lastCost << endl;
30       break;
31     }
32     if (update.norm() < 1e-3) {
33       // converge
34       break;
35     }
36
37     lastCost = cost;
38     cout << "iteration: " << iter << ", cost: " << cost << endl;
39   }
40 }
```

$\boldsymbol{H}$ $\boldsymbol{b}$

8.5.2

Coarse-to-fine

Listing 8.7: slambook2/ch8/direct_method.cpp

```
void DirectPoseEstimationMultiLayer(
  const cv::Mat &img1,
  const cv::Mat &img2,
  const VecVector2d &px_ref,
  const vector<double> depth_ref,
  Sophus::SE3d &T21) {
  // parameters
  int pyramids = 4;
  double pyramid_scale = 0.5;
  double scales[] = {1.0, 0.5, 0.25, 0.125};

  // create pyramids
  vector<cv::Mat> pyr1, pyr2; // image pyramids
  for (int i = 0; i < pyramids; i++) {
    if (i == 0) {
      pyr1.push_back(img1);
      pyr2.push_back(img2);
    } else {
      cv::Mat img1_pyr, img2_pyr;
      cv::resize(pyr1[i - 1], img1_pyr,
        cv::Size(pyr1[i - 1].cols * pyramid_scale, pyr1[i - 1].rows * pyramid_scale));
      cv::resize(pyr2[i - 1], img2_pyr,
        cv::Size(pyr2[i - 1].cols * pyramid_scale, pyr2[i - 1].rows * pyramid_scale));
      pyr1.push_back(img1_pyr);
      pyr2.push_back(img2_pyr);
    }
  }

  double fxG = fx, fyG = fy, cxG = cx, cyG = cy;  // backup the old values
  for (int level = pyramids - 1; level >= 0; level--) {
    VecVector2d px_ref_pyr; // set the keypoints in this pyramid level
    for (auto &px: px_ref) {
      px_ref_pyr.push_back(scales[level] * px);
    }

    // scale fx, fy, cx, cy in different pyramid levels
    fx = fxG * scales[level];
    fy = fyG * scales[level];
    cx = cxG * scales[level];
    cy = cyG * scales[level];
    DirectPoseEstimationSingleLayer(pyr1[level], pyr2[level], px_ref_pyr, depth_ref,
        T21);
  }
}
```

8.5.3

Kitti[74] left.png disparity.png 000001.png-000005.png

Listing 8.8: slambook2/ch8/direct_method.cpp

```
int main(int argc, char **argv) {

  cv::Mat left_img = cv::imread(left_file, 0);
  cv::Mat disparity_img = cv::imread(disparity_file, 0);

  // let's randomly pick pixels in the first image and generate some 3d points in the
      first image's frame
  cv::RNG rng;
  int nPoints = 2000;
  int boarder = 20;
  VecVector2d pixels_ref;
```

```
  vector<double> depth_ref;

  // generate pixels in ref and load depth data
  for (int i = 0; i < nPoints; i++) {
    int x = rng.uniform(boarder, left_img.cols - boarder);  // don't pick pixels close to boarder
    int y = rng.uniform(boarder, left_img.rows - boarder);  // don't pick pixels close to boarder
    int disparity = disparity_img.at<uchar>(y, x);
    double depth = fx * baseline / disparity; // you know this is disparity to depth
    depth_ref.push_back(depth);
    pixels_ref.push_back(Eigen::Vector2d(x, y));
  }

  // estimates 01~05.png's pose using this information
  Sophus::SE3d T_cur_ref;

  for (int i = 1; i < 6; i++) {  // 1~10
    cv::Mat img = cv::imread((fmt_others % i).str(), 0);
    DirectPoseEstimationMultiLayer(left_img, img, pixels_ref, depth_ref, T_cur_ref);
  }
  return 0;
}
```

Figure 8-5 3.8

2 8 2000

初始图像

初始视差图

追踪图像

直接法追踪点

Figure 8-5:

(8.19)

229 P Figure 8-6 I_1

P 126 $229-126=103$

u 123 3 v 18 108 $[-3,-18]$

* $\exp(\boldsymbol{\xi}^\wedge)$ I_2 I_2'

Figure 8-7

patch Normalized Cross Correlation NCC

8.5.4

•

*

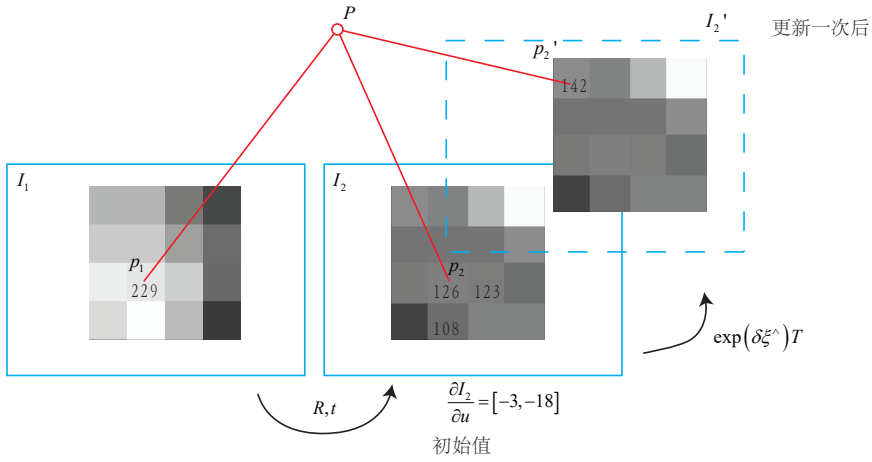
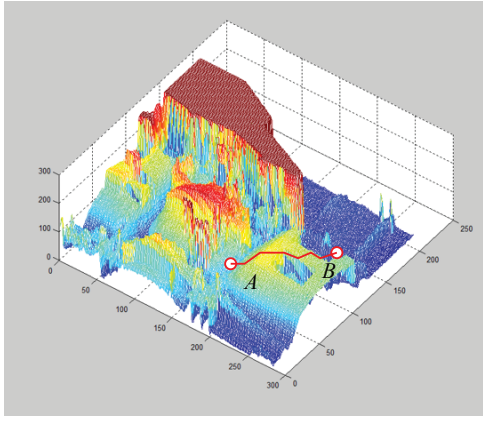


Figure 8-6:



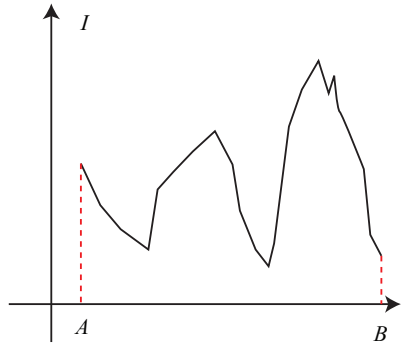


Figure 8-7: “ ” “ ”

-
-

-
- “ ” 500
- [70]

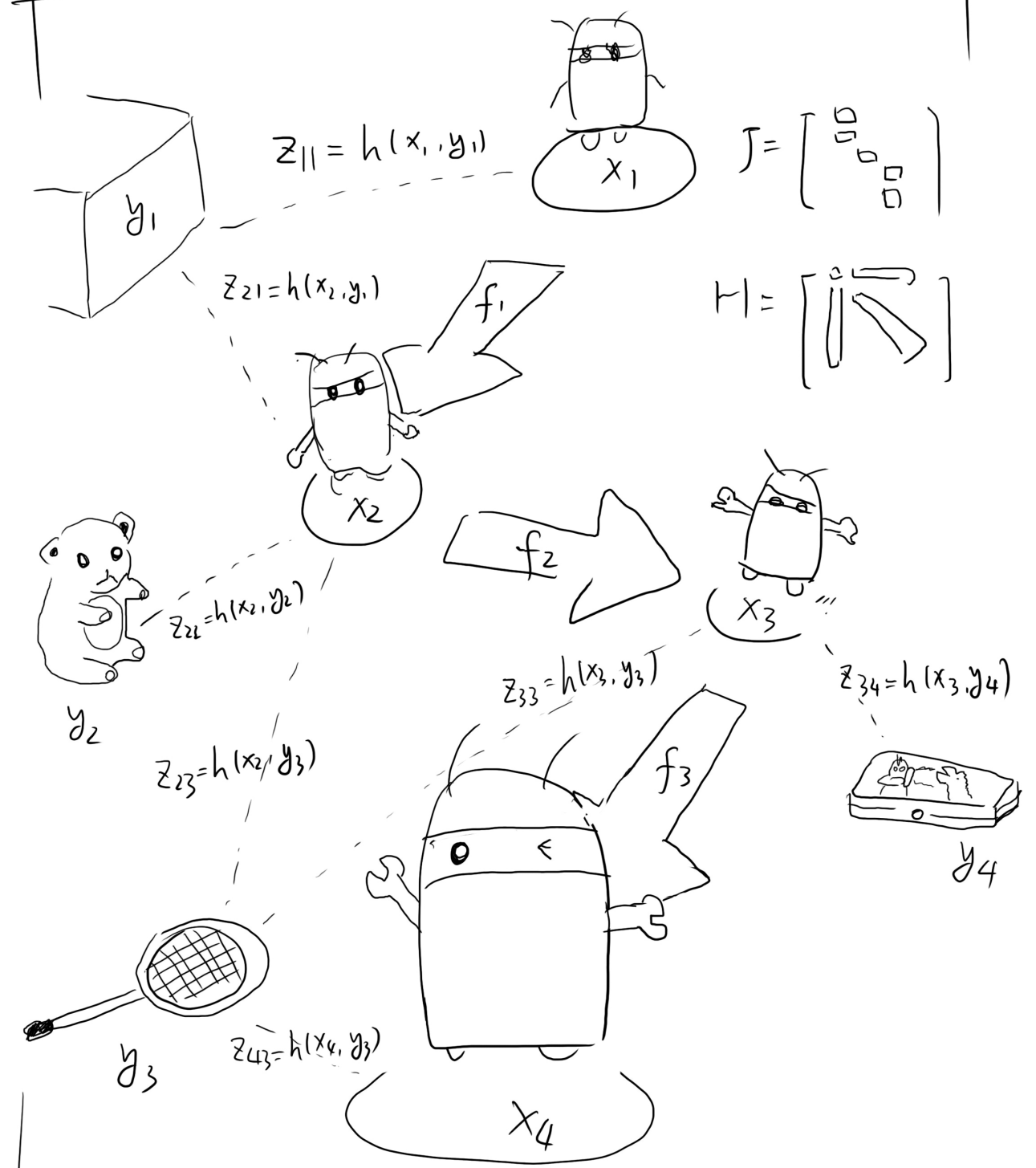
1. LK
2. $u+1\ u-1$ 2 u
3. “ ”
4.* Ceres g2o
5. RGB-D [69, 75]

Chapter 9

1

1.
2. EKF
3.
4. g2o Ceres

SLAM


$z_{11} = h(x_1, y_1)$
y_1
x_1
$f =$
$z_{21} = h(x_2, y_1)$
f_1
$H =$
x_2
f_2
x_3
$z_{22} = h(x_2, y_2)$
y_2
$z_{33} = h(x_3, y_3)$
$z_{34} = h(x_3, y_4)$
$z_{23} = h(x_2, y_3)$
f_3
y_4
y_3
$z_{43} = h(x_4, y_3)$
x_4

9.1

9.1.1

2 —— “ ” “ ”
SLAM $t = 0\ t = N$ $\boldsymbol{x}_0\ \boldsymbol{x}_N$ $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_M$

$$\begin{cases} \boldsymbol{x}_k = f\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k\right) + \boldsymbol{w}_k \\ \boldsymbol{z}_{k,j} = h\left(\boldsymbol{y}_j, \boldsymbol{x}_k\right) + \boldsymbol{v}_{k,j} \end{cases} \quad k = 1, \ldots, N,\ j = 1, \ldots, M. \tag{9.1}$$

1. $\boldsymbol{x}_k \quad \boldsymbol{y}_j$ SLAM

2. SfM Structure from Motion SfM SLAM SfM

$\boldsymbol{x} \quad \boldsymbol{y}$ $\boldsymbol{u} \quad \boldsymbol{z} \quad \boldsymbol{x}, \boldsymbol{y}$

—— ——

1

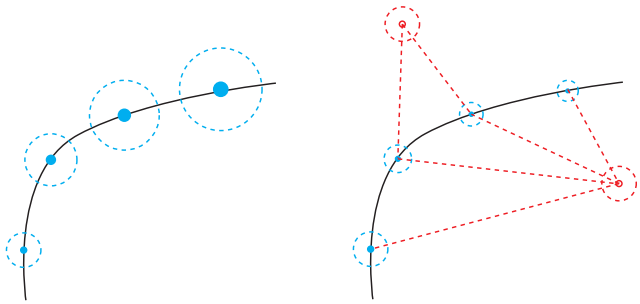


Figure 9-1:

6 SLAM
$\boldsymbol{x}_k\ k$ m

$$\boldsymbol{x}_k \triangleq \{\boldsymbol{x}_k, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_m\}. \tag{9.2}$$

k $\boldsymbol{z}_k$ $\boldsymbol{y}$ $\boldsymbol{x}$ $\boldsymbol{y}$

$$\begin{cases} \boldsymbol{x}_k = f\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k\right) + \boldsymbol{w}_k \\ \boldsymbol{z}_k = h\left(\boldsymbol{x}_k\right) + \boldsymbol{v}_k \end{cases} \quad k = 1, \ldots, N. \tag{9.3}$$

k $0\ k$

$$P(\boldsymbol{x}_k | \boldsymbol{x}_0, \boldsymbol{u}_{1:k}, \boldsymbol{z}_{1:k}). \tag{9.4}$$

$0 : k \quad 0 \quad k$ $\boldsymbol{z}_k \quad k$ $\boldsymbol{x}_k \quad \boldsymbol{x}_{k-1}, \boldsymbol{x}_{k-2}$
$\boldsymbol{z}_k\ \boldsymbol{x}_k$

$$P\left(\boldsymbol{x}_k | \boldsymbol{x}_0, \boldsymbol{u}_{1:k}, \boldsymbol{z}_{1:k}\right) \propto P\left(\boldsymbol{z}_k | \boldsymbol{x}_k\right) P\left(\boldsymbol{x}_k | \boldsymbol{x}_0, \boldsymbol{u}_{1:k}, \boldsymbol{z}_{1:k-1}\right). \tag{9.5}$$

$\boldsymbol{x}_k$ $\boldsymbol{x}_{k-1}$ $\boldsymbol{x}_{k-1}$

$$P\left(\boldsymbol{x}_k|\boldsymbol{x}_0,\boldsymbol{u}_{1:k},\boldsymbol{z}_{1:k-1}\right)=\int P\left(\boldsymbol{x}_k|\boldsymbol{x}_{k-1},\boldsymbol{x}_0,\boldsymbol{u}_{1:k},\boldsymbol{z}_{1:k-1}\right)P\left(\boldsymbol{x}_{k-1}|\boldsymbol{x}_0,\boldsymbol{u}_{1:k},\boldsymbol{z}_{1:k-1}\right)\mathrm{d}\boldsymbol{x}_{k-1}. \tag{9.6}$$

k $k-1$ k

1 EKF k

9.1.2 KF

(9.6)

$$P\left(\boldsymbol{x}_k|\boldsymbol{x}_{k-1},\boldsymbol{x}_0,\boldsymbol{u}_{1:k},\boldsymbol{z}_{1:k-1}\right)=P\left(\boldsymbol{x}_k|\boldsymbol{x}_{k-1},\boldsymbol{u}_k\right). \tag{9.7}$$

k $k-1$ $\boldsymbol{x}_{k-1}$ $\boldsymbol{u}_k$ k

$$P\left(\boldsymbol{x}_{k-1}|\boldsymbol{x}_0,\boldsymbol{u}_{1:k},\boldsymbol{z}_{1:k-1}\right)=P\left(\boldsymbol{x}_{k-1}|\boldsymbol{x}_0,\boldsymbol{u}_{1:k-1},\boldsymbol{z}_{1:k-1}\right). \tag{9.8}$$

k $\boldsymbol{u}_k$ $k-1$ $\boldsymbol{u}_k$ $k-1$ “ $k-1$ k ”

$$\begin{cases}\boldsymbol{x}_k=\boldsymbol{A}_k\boldsymbol{x}_{k-1}+\boldsymbol{u}_k+\boldsymbol{w}_k\\ \boldsymbol{z}_k=\boldsymbol{C}_k\boldsymbol{x}_k+\boldsymbol{v}_k\end{cases}\quad k=1,\ldots,N. \tag{9.9}$$

$$\boldsymbol{w}_k\sim N(\boldsymbol{0},\boldsymbol{R}).\quad \boldsymbol{v}_k\sim N(\boldsymbol{0},\boldsymbol{Q}). \tag{9.10}$$

$\boldsymbol{R}$ $\boldsymbol{Q}$ $k-1$ $k-1$ $\hat{\boldsymbol{x}}_{k-1}$ $\hat{\boldsymbol{P}}_{k-1}$ k $\boldsymbol{x}_k$ $\hat{\boldsymbol{x}}_k$ $\boldsymbol{x}_k$

$$P\left(\boldsymbol{x}_k|\boldsymbol{x}_0,\boldsymbol{u}_{1:k},\boldsymbol{z}_{1:k-1}\right)=N\left(\boldsymbol{A}_k\hat{\boldsymbol{x}}_{k-1}+\boldsymbol{u}_k,\boldsymbol{A}_k\hat{\boldsymbol{P}}_{k-1}\boldsymbol{A}_k^{\mathrm{T}}+\boldsymbol{R}\right). \tag{9.11}$$

A.3

$$\check{\boldsymbol{x}}_k=\boldsymbol{A}_k\hat{\boldsymbol{x}}_{k-1}+\boldsymbol{u}_k,\quad \check{\boldsymbol{P}}_k=\boldsymbol{A}_k\hat{\boldsymbol{P}}_{k-1}\boldsymbol{A}_k^{\mathrm{T}}+\boldsymbol{R}. \tag{9.12}$$

$$P\left(\boldsymbol{z}_k|\boldsymbol{x}_k\right)=N\left(\boldsymbol{C}_k\boldsymbol{x}_k,\boldsymbol{Q}\right). \tag{9.13}$$

(9.5) $\boldsymbol{x}_k$ $\boldsymbol{x}_k\sim N(\hat{\boldsymbol{x}}_k,\hat{\boldsymbol{P}}_k)$

$$N(\hat{\boldsymbol{x}}_k,\hat{\boldsymbol{P}}_k)=\eta N\left(\boldsymbol{C}_k\boldsymbol{x}_k,\boldsymbol{Q}\right)\cdot N(\check{\boldsymbol{x}}_k,\check{\boldsymbol{P}}_k). \tag{9.14}$$

*

$$\left(\boldsymbol{x}_k-\hat{\boldsymbol{x}}_k\right)^{\mathrm{T}}\hat{\boldsymbol{P}}_k^{-1}\left(\boldsymbol{x}_k-\hat{\boldsymbol{x}}_k\right)=\left(\boldsymbol{z}_k-\boldsymbol{C}_k\boldsymbol{x}_k\right)^{\mathrm{T}}\boldsymbol{Q}^{-1}\left(\boldsymbol{z}_k-\boldsymbol{C}_k\boldsymbol{x}_k\right)+\left(\boldsymbol{x}_k-\check{\boldsymbol{x}}_k\right)^{\mathrm{T}}\check{\boldsymbol{P}}_k^{-1}\left(\boldsymbol{x}_k-\check{\boldsymbol{x}}_k\right). \tag{9.15}$$

$\hat{\boldsymbol{x}}_k$ $\hat{\boldsymbol{P}}_k$ $\boldsymbol{x}_k$

$$\hat{\boldsymbol{P}}_k^{-1}=\boldsymbol{C}_k^{\mathrm{T}}\boldsymbol{Q}^{-1}\boldsymbol{C}_k+\check{\boldsymbol{P}}_k^{-1}. \tag{9.16}$$

* $\boldsymbol{x}_k$

$$\boldsymbol{K} = \hat{\boldsymbol{P}}_k \boldsymbol{C}_k^{\mathrm{T}} \boldsymbol{Q}^{-1}. \tag{9.17}$$

(9.16) $\hat{\boldsymbol{P}}_k$

$$\boldsymbol{I} = \hat{\boldsymbol{P}}_k \boldsymbol{C}_k^{\mathrm{T}} \boldsymbol{Q}^{-1} \boldsymbol{C}_k + \hat{\boldsymbol{P}}_k \check{\boldsymbol{P}}_k^{-1} = \boldsymbol{K}\boldsymbol{C}_k + \hat{\boldsymbol{P}}_k \check{\boldsymbol{P}}_k^{-1}. \tag{9.18}$$

*

$$\hat{\boldsymbol{P}}_k = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{C}_k)\check{\boldsymbol{P}}_k. \tag{9.19}$$

$$-2\hat{\boldsymbol{x}}_k^{\mathrm{T}} \hat{\boldsymbol{P}}_k^{-1} \boldsymbol{x}_k = -2\boldsymbol{z}_k^{\mathrm{T}} \boldsymbol{Q}^{-1} \boldsymbol{C}_k \boldsymbol{x}_k - 2\check{\boldsymbol{x}}_k^{\mathrm{T}} \check{\boldsymbol{P}}_k^{-1} \boldsymbol{x}_k. \tag{9.20}$$

$$\hat{\boldsymbol{P}}_k^{-1} \hat{\boldsymbol{x}}_k = \boldsymbol{C}_k^{\mathrm{T}} \boldsymbol{Q}^{-1} \boldsymbol{z}_k + \check{\boldsymbol{P}}_k^{-1} \check{\boldsymbol{x}}_k. \tag{9.21}$$

$\hat{\boldsymbol{P}}_k$ (9.17)

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{P}}_k \boldsymbol{C}_k^{\mathrm{T}} \boldsymbol{Q}^{-1} \boldsymbol{z}_k + \hat{\boldsymbol{P}}_k \check{\boldsymbol{P}}_k^{-1} \check{\boldsymbol{x}}_k \tag{9.22}$$

$$= \boldsymbol{K}\boldsymbol{z}_k + (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{C}_k)\,\check{\boldsymbol{x}}_k = \check{\boldsymbol{x}}_k + \boldsymbol{K}\left(\boldsymbol{z}_k - \boldsymbol{C}_k \check{\boldsymbol{x}}_k\right). \tag{9.23}$$

“ ” Predict “ ” Update

1.

$$\check{\boldsymbol{x}}_k = \boldsymbol{A}_k \hat{\boldsymbol{x}}_{k-1} + \boldsymbol{u}_k, \quad \check{\boldsymbol{P}}_k = \boldsymbol{A}_k \hat{\boldsymbol{P}}_{k-1} \boldsymbol{A}_k^{\mathrm{T}} + \boldsymbol{R}. \tag{9.24}$$

2. $\boldsymbol{K}$

$$\boldsymbol{K} = \check{\boldsymbol{P}}_k \boldsymbol{C}_k^{\mathrm{T}} \left(\boldsymbol{C}_k \check{\boldsymbol{P}}_k \boldsymbol{C}_k^{\mathrm{T}} + \boldsymbol{Q}_k\right)^{-1}. \tag{9.25}$$

$$\begin{aligned} \hat{\boldsymbol{x}}_k &= \check{\boldsymbol{x}}_k + \boldsymbol{K}\left(\boldsymbol{z}_k - \boldsymbol{C}_k \check{\boldsymbol{x}}_k\right) \\ \hat{\boldsymbol{P}}_k &= (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{C}_k)\,\check{\boldsymbol{P}}_k. \end{aligned} \quad . \tag{9.26}$$

9.1.3 EKF

SLAM SLAM

Extended Kalman Filter EKF

1 $\hat{\boldsymbol{x}}_{k-1}, \hat{\boldsymbol{P}}_{k-1}$ k $\hat{\boldsymbol{x}}_{k-1}, \hat{\boldsymbol{P}}_{k-1}$ $k-$

$$\boldsymbol{x}_k \approx f\left(\hat{\boldsymbol{x}}_{k-1}, \boldsymbol{u}_k\right) + \left.\frac{\partial f}{\partial \boldsymbol{x}_{k-1}}\right|_{\hat{\boldsymbol{x}}_{k-1}} \left(\boldsymbol{x}_{k-1} - \hat{\boldsymbol{x}}_{k-1}\right) + \boldsymbol{w}_k. \tag{9.27}$$

$$\boldsymbol{F} = \left.\frac{\partial f}{\partial \boldsymbol{x}_{k-1}}\right|_{\hat{\boldsymbol{x}}_{k-1}}. \tag{9.28}$$

$$\boldsymbol{z}_k \approx h\left(\check{\boldsymbol{x}}_k\right) + \left.\frac{\partial h}{\partial \boldsymbol{x}_k}\right|_{\check{\boldsymbol{x}}_k} \left(\boldsymbol{x}_k - \check{\boldsymbol{x}}_k\right) + \boldsymbol{n}_k. \tag{9.29}$$

$$\boldsymbol{H} = \left.\frac{\partial h}{\partial \boldsymbol{x}_k}\right|_{\check{\boldsymbol{x}}_k}. \tag{9.30}$$

* $\hat{\boldsymbol{P}}_k$ $\boldsymbol{K}$ $\hat{\boldsymbol{P}}_k$ $\boldsymbol{K}$ $\boldsymbol{K}$ $\hat{\boldsymbol{P}}_k$ Sherman-Morrison-Woodbury [76]

$$P\left(\boldsymbol{x}_k|\boldsymbol{x}_0,\boldsymbol{u}_{1:k},\boldsymbol{z}_{0:k-1}\right)=N(f\left(\hat{\boldsymbol{x}}_{k-1},\boldsymbol{u}_k\right),\boldsymbol{F}\hat{\boldsymbol{P}}_{k-1}\boldsymbol{F}^{\mathrm{T}}+\boldsymbol{R}_k). \tag{9.31}$$

$$\check{\boldsymbol{x}}_k=f\left(\hat{\boldsymbol{x}}_{k-1},\boldsymbol{u}_k\right),\quad \check{\boldsymbol{P}}_k=\boldsymbol{F}\hat{\boldsymbol{P}}_{k-1}\boldsymbol{F}^{\mathrm{T}}+\boldsymbol{R}_k. \tag{9.32}$$

$$P\left(\boldsymbol{z}_k|\boldsymbol{x}_k\right)=N(h\left(\check{\boldsymbol{x}}_k\right)+\boldsymbol{H}\left(\boldsymbol{x}_k-\check{\boldsymbol{x}}_k\right),\boldsymbol{Q}_k). \tag{9.33}$$

$\boldsymbol{x}_k$ EKF $\boldsymbol{K}_k$

$$\boldsymbol{K}_k=\check{\boldsymbol{P}}_k\boldsymbol{H}^{\mathrm{T}}\left(\boldsymbol{H}\check{\boldsymbol{P}}_k\boldsymbol{H}^{\mathrm{T}}+\boldsymbol{Q}_k\right)^{-1}. \tag{9.34}$$

$$\hat{\boldsymbol{x}}_k=\check{\boldsymbol{x}}_k+\boldsymbol{K}_k\left(\boldsymbol{z}_k-h\left(\check{\boldsymbol{x}}_k\right)\right),\hat{\boldsymbol{P}}_k=\left(\boldsymbol{I}-\boldsymbol{K}_k\boldsymbol{H}\right)\check{\boldsymbol{P}}_k. \tag{9.35}$$

SLAM MAP

9.1.4 EKF

EKF EKF SLAM EKF SLAM IF [77] IKF[78] It
KF UKF[79] Unscented KF [80–82] SWF Sliding Window Filter [83] [17]* EKF [84, 85]
EKF

1. k $k-1$ $k-1$
   SLAM Full-SLAM
2. 6 EKF $\hat{\boldsymbol{x}}_{k-1}$
   — EKF
3. EKF SLAM EKF
   SLAM
4. EKF SLAM

EKF [13] “ ” g2o Ceres

9.2 BA

Bundle Adjustment‡ 3D bundles
of light rays adjustment [34] BA
5 7 BA

---

*
†
‡ Bundle Adjustment

9.2.1 BA

$\boldsymbol{p}$

1. $(\boldsymbol{R}, \boldsymbol{t})$

$$\boldsymbol{P}' = \boldsymbol{R}\boldsymbol{p} + \boldsymbol{t} = [X', Y', Z']^{\mathrm{T}}. \tag{9.36}$$

2. $\boldsymbol{P}'$

$$\boldsymbol{P}_c = [u_c, v_c, 1]^{\mathrm{T}} = [X'/Z', Y'/Z', 1]^{\mathrm{T}}. \tag{9.37}$$

3.

$$\begin{cases} u_c' = u_c\left(1 + k_1 r_c^2 + k_2 r_c^4\right) \\ v_c' = v_c\left(1 + k_1 r_c^2 + k_2 r_c^4\right) \end{cases}. \tag{9.38}$$

4.

$$\begin{cases} u_s = f_x u_c' + c_x \\ v_s = f_y v_c' + c_y \end{cases}. \tag{9.39}$$

Figure 9-2

$$\boldsymbol{z} = h(\boldsymbol{x}, \boldsymbol{y}). \tag{9.40}$$

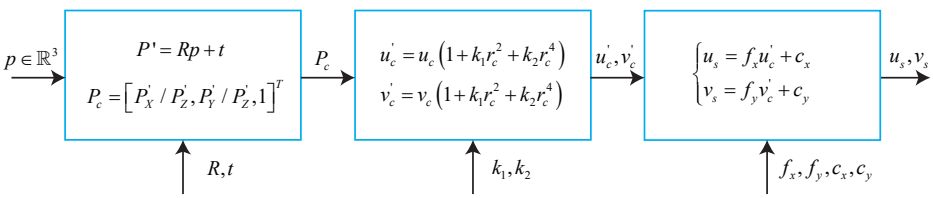


Figure 9-2: $\boldsymbol{p}$ u_s, v_s $r_c^2 = u_c^2 + v_c^2$

$\boldsymbol{x}$ $\boldsymbol{R}, \boldsymbol{t}$ $\boldsymbol{T}$ $\boldsymbol{\xi}$ $\boldsymbol{y}$ $\boldsymbol{p}$ $\boldsymbol{z} \triangleq [u_s, v_s]^T$

$$\boldsymbol{e} = \boldsymbol{z} - h(\boldsymbol{T}, \boldsymbol{p}). \tag{9.41}$$

$\boldsymbol{z}_{ij}$ $\boldsymbol{T}_i$ $\boldsymbol{p}_j$ Cost Function

$$\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}\|\boldsymbol{e}_{ij}\|^2 = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}\|\boldsymbol{z}_{ij} - h(\boldsymbol{T}_i, \boldsymbol{p}_j)\|^2. \tag{9.42}$$

BA 6

9.2.2 BA

$h(\boldsymbol{T},\boldsymbol{p})$ 6.2 $\Delta\boldsymbol{x}$ (6.33) $\Delta\boldsymbol{x}$

$$\boldsymbol{x}=[\boldsymbol{T}_1,\ldots,\boldsymbol{T}_m,\boldsymbol{p}_1,\ldots,\boldsymbol{p}_n]^{\mathrm{T}}. \tag{9.43}$$

$\Delta\boldsymbol{x}$

$$\frac{1}{2}\left\|f(\boldsymbol{x}+\Delta\boldsymbol{x})\right\|^2\approx\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}\left\|\boldsymbol{e}_{ij}+\boldsymbol{F}_{ij}\Delta\boldsymbol{\xi}_i+\boldsymbol{E}_{ij}\Delta\boldsymbol{p}_j\right\|^2. \tag{9.44}$$

$\boldsymbol{F}_{ij}$ $\boldsymbol{E}_{ij}$ 7.7.3

$$\boldsymbol{x}_c=[\boldsymbol{\xi}_1,\boldsymbol{\xi}_2,\ldots,\boldsymbol{\xi}_m]^{\mathrm{T}}\in\mathbb{R}^{6m}, \tag{9.45}$$

$$\boldsymbol{x}_p=[\boldsymbol{p}_1,\boldsymbol{p}_2,\ldots,\boldsymbol{p}_n]^{\mathrm{T}}\in\mathbb{R}^{3n}, \tag{9.46}$$

(9.44)

$$\frac{1}{2}\left\|f(\boldsymbol{x}+\Delta\boldsymbol{x})\right\|^2=\frac{1}{2}\left\|\boldsymbol{e}+\boldsymbol{F}\Delta\boldsymbol{x}_c+\boldsymbol{E}\Delta\boldsymbol{x}_p\right\|^2. \tag{9.47}$$

$\boldsymbol{E}$ $\boldsymbol{F}$ $\boldsymbol{F}_{ij}$ $\boldsymbol{E}_{ij}$“ ” —

$$\boldsymbol{H}\Delta\boldsymbol{x}=\boldsymbol{g}. \tag{9.48}$$

6 — $\boldsymbol{H}$ $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}$ $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}+\lambda\boldsymbol{I}$

$$\boldsymbol{J}=[\boldsymbol{F}\ \boldsymbol{E}]. \tag{9.49}$$

$\boldsymbol{H}$

$$\boldsymbol{H}=\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}=\begin{bmatrix}\boldsymbol{F}^{\mathrm{T}}\boldsymbol{F} & \boldsymbol{F}^{\mathrm{T}}\boldsymbol{E}\\ \boldsymbol{E}^{\mathrm{T}}\boldsymbol{F} & \boldsymbol{E}^{\mathrm{T}}\boldsymbol{E}\end{bmatrix}. \tag{9.50}$$

— SLAM $\boldsymbol{H}$

9.2.3

21 SLAM $\boldsymbol{H}$ [36, 87]
$\boldsymbol{H}$ $\boldsymbol{J}(\boldsymbol{x})$ $\boldsymbol{e}_{ij}$ $\boldsymbol{T}_i$ $\boldsymbol{p}_j$ i j 0

$$\boldsymbol{J}_{ij}(\boldsymbol{x})=\left(\boldsymbol{0}_{2\times6},\ldots\boldsymbol{0}_{2\times6},\frac{\partial\boldsymbol{e}_{ij}}{\partial\boldsymbol{T}_i},\boldsymbol{0}_{2\times6},\ldots\boldsymbol{0}_{2\times3},\ldots\boldsymbol{0}_{2\times3},\frac{\partial\boldsymbol{e}_{ij}}{\partial\boldsymbol{p}_j},\boldsymbol{0}_{2\times3},\ldots\boldsymbol{0}_{2\times3}\right). \tag{9.51}$$

$\boldsymbol{0}_{2\times6}$ 2×6 $\boldsymbol{0}$ $\boldsymbol{0}_{2\times3}$ $\partial\boldsymbol{e}_{ij}/\partial\boldsymbol{\xi}_i$ 2×6 $\partial\boldsymbol{e}_{ij}/\partial\boldsymbol{p}_j$ 2×3
Figure 9-3 $\boldsymbol{J}_{ij}$ i,j $\boldsymbol{H}$ $\boldsymbol{J}_{ij}^{\mathrm{T}}\boldsymbol{J}_{ij}$ $\boldsymbol{J}_{ij}^{\mathrm{T}}\boldsymbol{J}_{ij}$ 4 $(i,i),(i,j),(j,i),(j,j)$ $\boldsymbol{H}$

$$\boldsymbol{H}=\sum_{i,j}\boldsymbol{J}_{ij}^{\mathrm{T}}\boldsymbol{J}_{ij}, \tag{9.52}$$

i j $\boldsymbol{H}$

$$\boldsymbol{H}=\begin{bmatrix}\boldsymbol{H}_{11} & \boldsymbol{H}_{12}\\ \boldsymbol{H}_{21} & \boldsymbol{H}_{22}\end{bmatrix}. \tag{9.53}$$

$\boldsymbol{H}_{11}$ $\boldsymbol{H}_{22}$ i,j

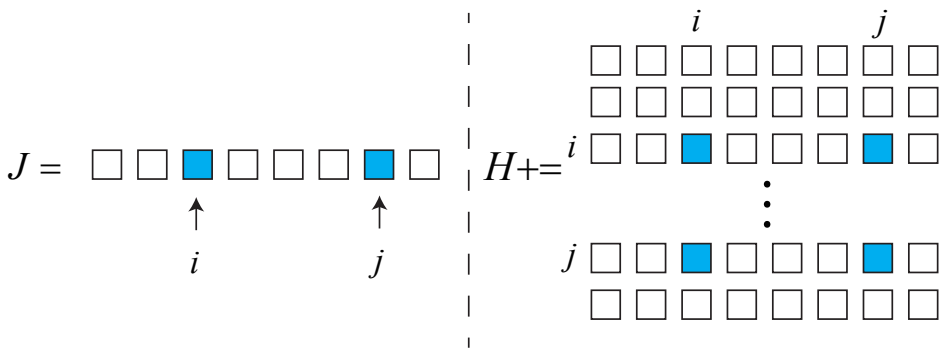


Figure 9-3: $\boldsymbol{J}$ $\boldsymbol{H}$

1. i, j $\boldsymbol{H}_{11}$ $\boldsymbol{H}_{i,i}$
2. $\boldsymbol{H}_{22}$ $\boldsymbol{H}_{j,j}$
3. $\boldsymbol{H}_{12}$ $\boldsymbol{H}_{21}$

$\boldsymbol{H}$ 2 C_1, C_2 6 $P_1, P_2, P_3, P_4, P_5, P_6$ 1, 2 $\boldsymbol{p}_j, j = 1, \cdots, 6$ C_1 P_1, P_2, P_3, P_4 C_2 P_3, P_4, P_5, P_6 Figure 9-4 i j

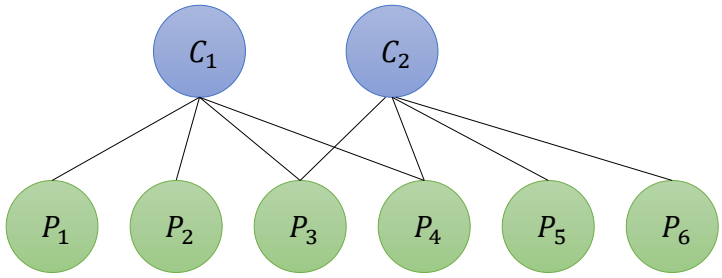


Figure 9-4: C_1 P_1, P_2, P_3, P_4 C_2 P_3 P_6

BA

$$\frac{1}{2}\left(\|\boldsymbol{e}_{11}\|^2 + \|\boldsymbol{e}_{12}\|^2 + \|\boldsymbol{e}_{13}\|^2 + \|\boldsymbol{e}_{14}\|^2 + \|\boldsymbol{e}_{23}\|^2 + \|\boldsymbol{e}_{24}\|^2 + \|\boldsymbol{e}_{25}\|^2 + \|\boldsymbol{e}_{26}\|^2\right). \tag{9.54}$$

$\boldsymbol{e}_{ij}$ (9.42) $\boldsymbol{e}_{11}$ C_1 P_1 $\boldsymbol{J}_{11}$ $\boldsymbol{e}_{11}$ $\boldsymbol{e}_{11}$ $\boldsymbol{\xi}_2$ $\boldsymbol{p}_2, \cdots, \boldsymbol{p}_6$ 0 $\boldsymbol{x}$ $(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \boldsymbol{p}_1, \cdots, \boldsymbol{p}_2)^{\mathrm{T}}$

$$\boldsymbol{J}_{11} = \frac{\partial \boldsymbol{e}_{11}}{\partial \boldsymbol{x}} = \left(\frac{\partial \boldsymbol{e}_{11}}{\partial \boldsymbol{\xi}_1}, \boldsymbol{0}_{2\times 6}, \frac{\partial \boldsymbol{e}_{11}}{\partial \boldsymbol{p}_1}, \boldsymbol{0}_{2\times 3}, \boldsymbol{0}_{2\times 3}, \boldsymbol{0}_{2\times 3}, \boldsymbol{0}_{2\times 3}\right). \tag{9.55}$$

0 $\boldsymbol{J}_{11}$ Figure 9-5

$\boldsymbol{J}_{ij}$ $\boldsymbol{H}$ Figure 9-6

Figure 9-4 Adjacency Matrix * $\boldsymbol{H}$ $\boldsymbol{H}$ 8×8 $\boldsymbol{H}$ Figure 9-7 $\boldsymbol{H}$

* i, j i j 1 0

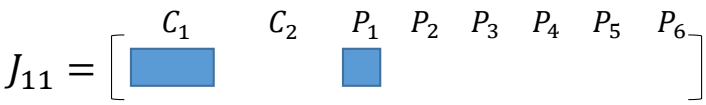


Figure 9-5: $\boldsymbol{J}_{11}$ C_1 P_1

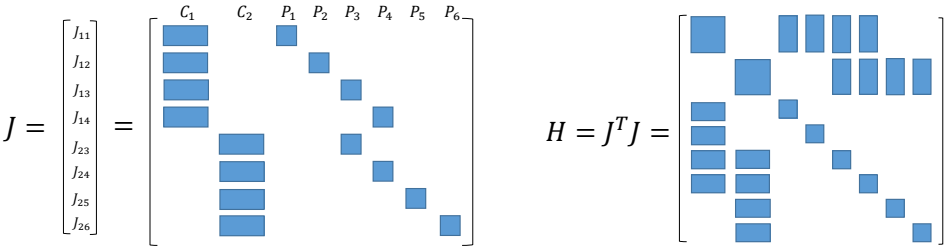


Figure 9-6: $\boldsymbol{H}$ 0

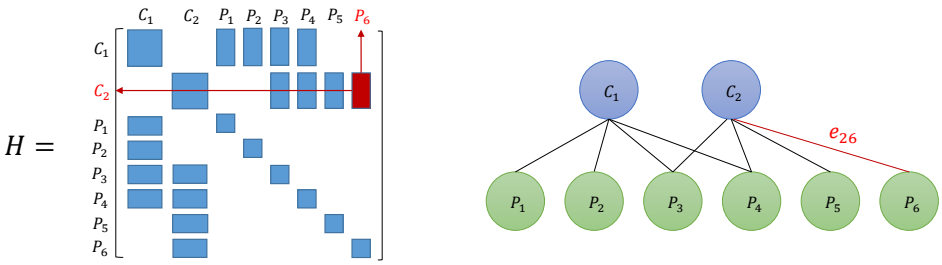


Figure 9-7: $\boldsymbol{H}$ $\boldsymbol{H}$ C_2 P_6 $\boldsymbol{e}_{26}$

m n $n \gg m$ $\boldsymbol{H}$ Figure 9-8 like [6] *

$\boldsymbol{H}$ $\boldsymbol{H}\Delta\boldsymbol{x} = \boldsymbol{g}$ $\boldsymbol{H}$ SLAM Schur SLAM Marginalizatio
Figure 9-8 4 (9.53) F
9 4 (9.50) 4 4 $\boldsymbol{B}, \boldsymbol{E}, \boldsymbol{E}^{\mathrm{T}}, \boldsymbol{C}$
$\boldsymbol{H}\boldsymbol{\Delta x} = \boldsymbol{g}$

$$\begin{bmatrix} \boldsymbol{B} & \boldsymbol{E} \\ \boldsymbol{E}^{\mathbf{T}} & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x}_c \\ \Delta \boldsymbol{x}_p \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{bmatrix}. \tag{9.56}$$

$\boldsymbol{B}$ $\boldsymbol{C}$ $\boldsymbol{B}$ $\boldsymbol{C}$ 3×3

$$\begin{bmatrix} \boldsymbol{I} & -\boldsymbol{E}\boldsymbol{C}^{-1} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{B} & \boldsymbol{E} \\ \boldsymbol{E}^{\mathbf{T}} & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x}_c \\ \Delta \boldsymbol{x}_p \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{E}\boldsymbol{C}^{-1} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{bmatrix}. \tag{9.57}$$

*

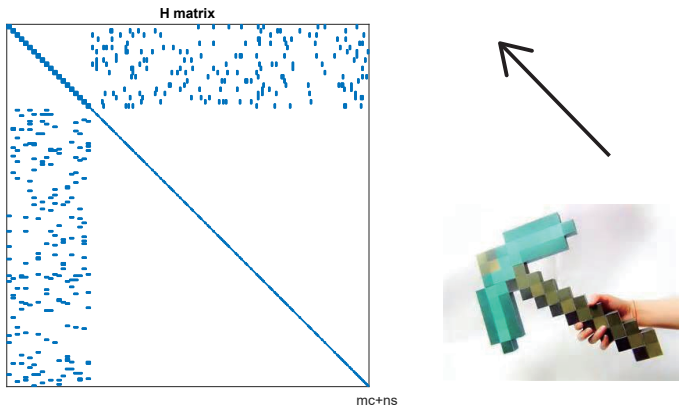


Figure 9-8: $\boldsymbol{H}$

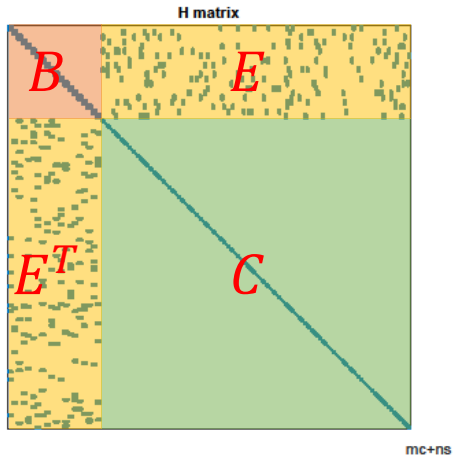


Figure 9-9: $\boldsymbol{H}$

$$\begin{bmatrix} \boldsymbol{B}-\boldsymbol{E}\boldsymbol{C}^{-1}\boldsymbol{E}^{\mathrm{T}} & \boldsymbol{0} \\ \boldsymbol{E}^{\mathrm{T}} & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{x}_c \\ \Delta\boldsymbol{x}_p \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}-\boldsymbol{E}\boldsymbol{C}^{-1}\boldsymbol{w} \\ \boldsymbol{w} \end{bmatrix}. \tag{9.58}$$

$\Delta\boldsymbol{x}_p$

$$\left[\boldsymbol{B}-\boldsymbol{E}\boldsymbol{C}^{-1}\boldsymbol{E}^{\mathrm{T}}\right]\Delta\boldsymbol{x}_c = \boldsymbol{v}-\boldsymbol{E}\boldsymbol{C}^{-1}\boldsymbol{w}. \tag{9.59}$$

$\boldsymbol{B}$ $\Delta\boldsymbol{x}_c$ $\Delta\boldsymbol{x}_p$ **Marginalization**[83] **Schur** Schur Elimination

1. $\boldsymbol{C}$ $\boldsymbol{C}^{-1}$
2. $\Delta\boldsymbol{x}_c$ $\Delta\boldsymbol{x}_p = \boldsymbol{C}^{-1}(\boldsymbol{w}-\boldsymbol{E}^{\mathrm{T}}\Delta\boldsymbol{x}_c)$ $\boldsymbol{C}^{-1}$

(9.59) $\boldsymbol{S}$ Figure 9-10 Schur $\boldsymbol{S}$ $\boldsymbol{H}$ Schur $\boldsymbol{S}$ $\boldsymbol{S}$ Co-visibility Figure 9-11 4×4 C_1, C_2, C_3, C_4

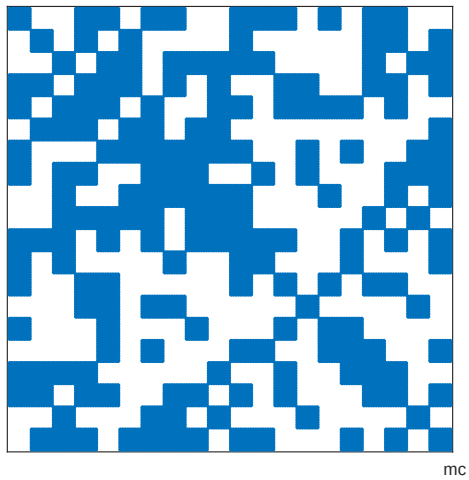


Figure 9-10: $\boldsymbol{H}$ Schur $\boldsymbol{S}$

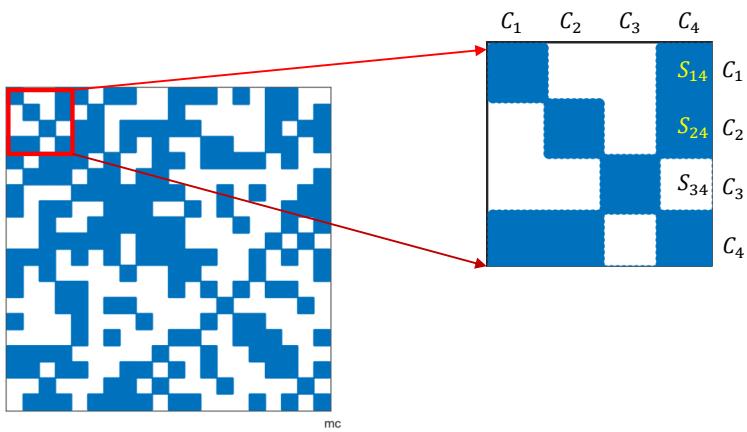


Figure 9-11: $\boldsymbol{S}$ 4×4 $\boldsymbol{S}_{14}, \boldsymbol{S}_{24}$ C_4 C_1 C_2 S_{34} C_3 C_4

$\boldsymbol{S}$ ORB-SLAM[88] Local Mapping BA Schur $\boldsymbol{S}$ Window BA $\boldsymbol{S}$ $(\Delta\boldsymbol{x}_c, \Delta\boldsymbol{x}_p)$ $\Delta\boldsymbol{x}_p$ $\Delta\boldsymbol{x}_c$ $\Delta\boldsymbol{x}_p$

$$P(\boldsymbol{x}_c, \boldsymbol{x}_p) = P(\boldsymbol{x}_c|\boldsymbol{x}_p)P(\boldsymbol{x}_p), \tag{9.60}$$

$\boldsymbol{x}_p$ Schur Cholesky Schur (9.59) $\boldsymbol{H}$ Schur

9.2.4

BA

Ro

Kernel

Huber

$$H(e) = \begin{cases} \frac{1}{2}e^2 & |e| \leqslant \delta, \\ \delta\left(|e| - \frac{1}{2}\delta\right) & \end{cases} \tag{9.61}$$

e δ Huber Figure 9-12 Huber Huber

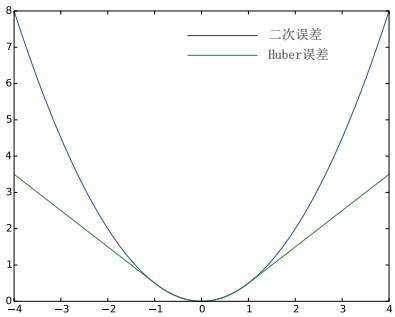


Figure 9-12: Huber

Huber Cauchy Tukey g2o Ceres

9.2.5

BA Bundle Adjustment Schur BA Ceres g2o Bundle Adjustment BAL[90]

9.3 Ceres BA

9.3.1 BAL

BAL BA BAL problem-16-22106-pre.txt BA `http://grail.cs.washington.edu/projects/bal` common.h BALProblem Ceres g2o BAL

1. BAL f k_1, k_2 f f_x f_y f_x f_y c_x, c_y
2. BAL -1

BALProblem Normalize Perturb
BALProblem BA ply meshlab meshlab
get

9.3.2 Ceres BA

bundle_adjustment_ceres.cpp Ceres BA Ceres SnavelyReprojectionError.h

Listing 9.1: slambook2/ch9/SnavelyReprojectionError.cpp

```
class SnavelyReprojectionError {
public:
  SnavelyReprojectionError(double observation_x, double observation_y) : observed_x(
      observation_x),
  observed_y(observation_y) {}

  template<typename T>
  bool operator()(const T *const camera,
    const T *const point,
    T *residuals) const {
    // camera[0,1,2] are the angle-axis rotation
    T predictions[2];
    CamProjectionWithDistortion(camera, point, predictions);
    residuals[0] = predictions[0] - T(observed_x);
    residuals[1] = predictions[1] - T(observed_y);

    return true;
  }

  // camera : 9 dims array
  // [0-2] : angle-axis rotation
  // [3-5] : translation
  // [6-8] : camera parameter, [6] focal length, [7-8] second and forth order radial
      distortion
  // point : 3D location.
  // predictions : 2D predictions with center of the image plane.
  template<typename T>
  static inline bool CamProjectionWithDistortion(const T *camera, const T *point, T *
      predictions) {
    // Rodrigues' formula
    T p[3];
    AngleAxisRotatePoint(camera, point, p);
    // camera[3,4,5] are the translation
    p[0] += camera[3];
    p[1] += camera[4];
    p[2] += camera[5];

    // Compute the center fo distortion
    T xp = -p[0] / p[2];
    T yp = -p[1] / p[2];

    // Apply second and fourth order radial distortion
    const T &l1 = camera[7];
    const T &l2 = camera[8];

    T r2 = xp * xp + yp * yp;
    T distortion = T(1.0) + r2 * (l1 + l2 * r2);

    const T &focal = camera[6];
    predictions[0] = focal * distortion * xp;
    predictions[1] = focal * distortion * yp;

    return true;
  }

  static ceres::CostFunction *Create(const double observed_x, const double observed_y)
      {
```

```
    return (new ceres::AutoDiffCostFunction<SnavelyReprojectionError, 2, 9, 3>(
      new SnavelyReprojectionError(observed_x, observed_y)));
  }

private:
  double observed_x;
  double observed_y;
};
```

Ceres CamProjectionWithDistortion Ceres double 6 1 2 9 BA

Listing 9.2: slambook2/ch9/SnavelyReprojectionError.cpp

```
void SolveBA(BALProblem &bal_problem) {
  const int point_block_size = bal_problem.point_block_size();
  const int camera_block_size = bal_problem.camera_block_size();
  double *points = bal_problem.mutable_points();
  double *cameras = bal_problem.mutable_cameras();

  // Observations is 2 * num_observations long array observations
  // [u_1, u_2, ... u_n], where each u_i is two dimensional, the x
  // and y position of the observation.
  const double *observations = bal_problem.observations();
  ceres::Problem problem;

  for (int i = 0; i < bal_problem.num_observations(); ++i) {
    ceres::CostFunction *cost_function;

    // Each Residual block takes a point and a camera as input
    // and outputs a 2 dimensional Residual
    cost_function =
      SnavelyReprojectionError::Create(observations[2 * i + 0], observations[2 * i +
          1]);

    // If enabled use Huber's loss function.
    ceres::LossFunction *loss_function = new ceres::HuberLoss(1.0);

    // Each observation corresponds to a pair of a camera and a point
    // which are identified by camera_index()[i] and point_index()[i]
    // respectively.
    double *camera = cameras + camera_block_size * bal_problem.camera_index()[i];
    double *point = points + point_block_size * bal_problem.point_index()[i];

    problem.AddResidualBlock(cost_function, loss_function, camera, point);
  }

  std::cout << "Solving ceres BA ... " << endl;
  ceres::Solver::Options options;
  options.linear_solver_type = ceres::LinearSolverType::SPARSE_SCHUR;
  options.minimizer_progress_to_stdout = true;
  ceres::Solver::Summary summary;
  ceres::Solve(options, &problem, &summary);
  std::cout << summary.FullReport() << "\n";
}
```

ceres::Solver::Options SPARSE_SCHUR Ceres Ceres BA

Listing 9.3:

```
./build/bundle_adjustment_ceres  problem-16-22106-pre.txt
Header: 16 22106 83718bal problem file loaded...
bal problem have 16 cameras and 22106 points.
Forming 83718 observations.
Solving ceres BA ...
iter      cost      cost_change  |gradient|   |step|    tr_ratio  tr_radius  ls_iter
     iter_time  total_time
0  1.842900e+07    0.00e+00    2.04e+06   0.00e+00   0.00e+00  1.00e+04        0
     6.10e-02    2.24e-01
1  1.449093e+06    1.70e+07    1.75e+06   2.16e+03   1.84e+00  3.00e+04        1
     1.79e-01    4.03e-01
```

```
2  5.848543e+04    1.39e+06    1.30e+06   1.55e+03   1.87e+00  9.00e+04        1
     1.56e-01    5.59e-01
3  1.581483e+04    4.27e+04    4.98e+05   4.98e+02   1.29e+00  2.70e+05        1
     1.51e-01    7.10e-01
......
```

initial.ply final.ply meshlab Figure 9-13

9.4 g2o BA

g2o BA g2o override

Listing 9.4: slambook2/ch9/bundle_adjustment_g2o.cpp

```
/// 
struct PoseAndIntrinsics {
  PoseAndIntrinsics() {}

  /// set from given data address
  explicit PoseAndIntrinsics(double *data_addr) {
    rotation = SO3d::exp(Vector3d(data_addr[0], data_addr[1], data_addr[2]));
    translation = Vector3d(data_addr[3], data_addr[4], data_addr[5]);
    focal = data_addr[6];
    k1 = data_addr[7];
    k2 = data_addr[8];
  }

  /// 
  void set_to(double *data_addr) {
    auto r = rotation.log();
    for (int i = 0; i < 3; ++i) data_addr[i] = r[i];
    for (int i = 0; i < 3; ++i) data_addr[i + 3] = translation[i];
    data_addr[6] = focal;
    data_addr[7] = k1;
    data_addr[8] = k2;
  }

  SO3d rotation;
  Vector3d translation = Vector3d::Zero();
  double focal = 0;
  double k1 = 0, k2 = 0;
};

/// 9so3t, f, k1, k2
class VertexPoseAndIntrinsics : public g2o::BaseVertex<9, PoseAndIntrinsics> {
public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW;

  VertexPoseAndIntrinsics() {}

  virtual void setToOriginImpl() override {
    _estimate = PoseAndIntrinsics();
  }

  virtual void oplusImpl(const double *update) override {
    _estimate.rotation = SO3d::exp(Vector3d(update[0], update[1], update[2])) *
        _estimate.rotation;
    _estimate.translation += Vector3d(update[3], update[4], update[5]);
    _estimate.focal += update[6];
    _estimate.k1 += update[7];
    _estimate.k2 += update[8];
  }

  /// 
  Vector2d project(const Vector3d &point) {
    Vector3d pc = _estimate.rotation * point + _estimate.translation;
    pc = -pc / pc[2];
    double r2 = pc.squaredNorm();
    double distortion = 1.0 + r2 * (_estimate.k1 + _estimate.k2 * r2);
    return Vector2d(_estimate.focal * distortion * pc[0],
```

```
    _estimate.focal * distortion * pc[1]);
  }

  virtual bool read(istream &in) {}

  virtual bool write(ostream &out) const {}
};

class VertexPoint : public g2o::BaseVertex<3, Vector3d> {
public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW;

  VertexPoint() {}

  virtual void setToOriginImpl() override {
    _estimate = Vector3d(0, 0, 0);
  }

  virtual void oplusImpl(const double *update) override {
    _estimate += Vector3d(update[0], update[1], update[2]);
  }

  virtual bool read(istream &in) {}

  virtual bool write(ostream &out) const {}
};

class EdgeProjection :
public g2o::BaseBinaryEdge<2, Vector2d, VertexPoseAndIntrinsics, VertexPoint> {
public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW;

  virtual void computeError() override {
    auto v0 = (VertexPoseAndIntrinsics *) _vertices[0];
    auto v1 = (VertexPoint *) _vertices[1];
    auto proj = v0->project(v1->estimate());
    _error = proj - _measurement;
  }

  // use numeric derivatives
  virtual bool read(istream &in) {}

  virtual bool write(ostream &out) const {}
};
```

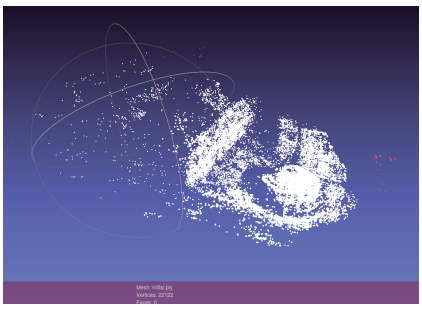
g2o BAL g2o

Listing 9.5: slambook2/ch9/bundle_adjustment_g2o.cpp

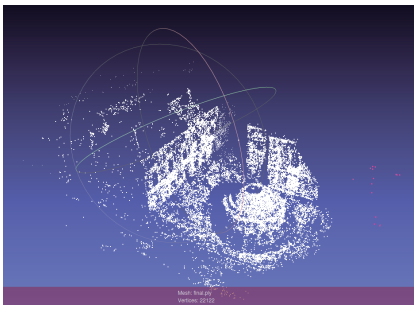
```
void SolveBA(BALProblem &bal_problem) {
  const int point_block_size = bal_problem.point_block_size();
  const int camera_block_size = bal_problem.camera_block_size();
  double *points = bal_problem.mutable_points();
  double *cameras = bal_problem.mutable_cameras();

  // pose dimension 9, landmark is 3
  typedef g2o::BlockSolver<g2o::BlockSolverTraits<9, 3>> BlockSolverType;
  typedef g2o::LinearSolverCSparse<BlockSolverType::PoseMatrixType> LinearSolverType;
  // use LM
  auto solver = new g2o::OptimizationAlgorithmLevenberg(
  g2o::make_unique<BlockSolverType>(g2o::make_unique<LinearSolverType>()));
  g2o::SparseOptimizer optimizer;
  optimizer.setAlgorithm(solver);
  optimizer.setVerbose(true);

  /// build g2o problem
  const double *observations = bal_problem.observations();
  // vertex
  vector<VertexPoseAndIntrinsics *> vertex_pose_intrinsics;
  vector<VertexPoint *> vertex_points;
  for (int i = 0; i < bal_problem.num_cameras(); ++i) {
    VertexPoseAndIntrinsics *v = new VertexPoseAndIntrinsics();
    double *camera = cameras + camera_block_size * i;
    v->setId(i);
```

```
    v->setEstimate(PoseAndIntrinsics(camera));
    optimizer.addVertex(v);
    vertex_pose_intrinsics.push_back(v);
  }
  for (int i = 0; i < bal_problem.num_points(); ++i) {
    VertexPoint *v = new VertexPoint();
    double *point = points + point_block_size * i;
    v->setId(i + bal_problem.num_cameras());
    v->setEstimate(Vector3d(point[0], point[1], point[2]));
    // □□□□□□□□□□□□g2oBAMarg
    v->setMarginalized(true);
    optimizer.addVertex(v);
    vertex_points.push_back(v);
  }

  // edge
  for (int i = 0; i < bal_problem.num_observations(); ++i) {
    EdgeProjection *edge = new EdgeProjection;
    edge->setVertex(0, vertex_pose_intrinsics[bal_problem.camera_index()[i]]);
    edge->setVertex(1, vertex_points[bal_problem.point_index()[i]]);
    edge->setMeasurement(Vector2d(observations[2 * i + 0], observations[2 * i + 1]));
    edge->setInformation(Matrix2d::Identity());
    edge->setRobustKernel(new g2o::RobustKernelHuber());
    optimizer.addEdge(edge);
  }

  optimizer.initializeOptimization();
  optimizer.optimize(40);

  // set to bal problem
  for (int i = 0; i < bal_problem.num_cameras(); ++i) {
    double *camera = cameras + camera_block_size * i;
    auto vertex = vertex_pose_intrinsics[i];
    auto estimate = vertex->estimate();
    estimate.set_to(camera);
  }
  for (int i = 0; i < bal_problem.num_points(); ++i) {
    double *point = points + point_block_size * i;
    auto vertex = vertex_points[i];
    for (int k = 0; k < 3; ++k) point[k] = vertex->estimate()[k];
  }
}
```

BALProblem g2o BA setMarginalized true g2o C
>setMarginalized(true) Ceres g2o



初始值



优化值

Figure 9-13:

9.5

SLAM EKF BA

1. (9.25) SMW Sherman-Morrison-Woodbury [6, 76]
2. g2o Ceres Meshlab
3. Ceres Schur
4. $\boldsymbol{S}$
5. [36] g2o Ceres Loss function
6.* f, k_1, k_2 5 $f_x, f_y, p_1, p_2, k_1, k_2$ Ceres g2o

Chapter 10

2

1. Sliding Window
2. Pose Graph
3. IMU
4. g2o Pose Graph

BA BA BA

10.1

10.1.1 BA

BA SfM SLAM BA BA—
— BA 1 20 0.5 SfM SLAM
[89] BA BA
BA N BA Sliding Window [91] N
SLAM2[88] “ ” Covisibility graph Figure 10-1 “ BA

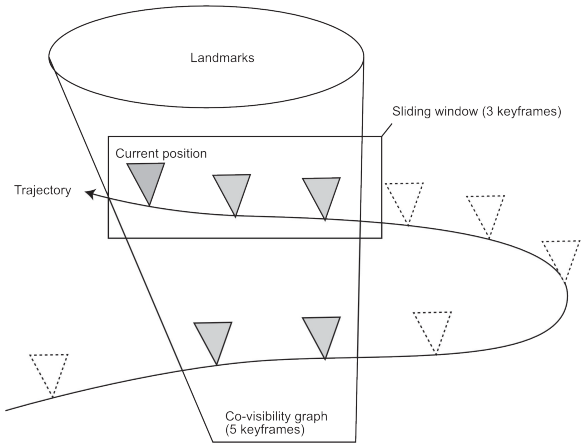


Figure 10-1:

“ ” “ ” “ ” “ ” “ ”

10.1.2

N

$$\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N,$$

M $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$ N
Adjustment Hessian

$$[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^{\mathrm{T}} \sim N([\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_N]^{\mathrm{T}}, \boldsymbol{\Sigma}).$$

$\boldsymbol{\mu}_k$ k $\boldsymbol{\Sigma}$ BA $\boldsymbol{\Sigma}$ BA $\boldsymbol{H}$ $\boldsymbol{S}$

1.

2.

BA BA BA

N $\boldsymbol{x}_{N+1}$ N+1 BA N-
1

$\boldsymbol{x}_1$ $\boldsymbol{x}_1$ $\boldsymbol{x}_1$ Figure 10-2

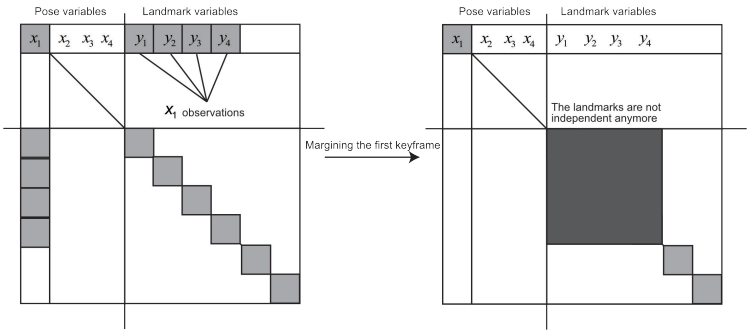


Figure 10-2:

$\boldsymbol{x}_1$ $\boldsymbol{y}_1$ $\boldsymbol{y}_4$ BA Hessian $\boldsymbol{x}_1$ $\boldsymbol{y}_1$ $\boldsymbol{y}_4$ $\boldsymbol{x}_1$ $\boldsymbol{x}_1$ Schur in [91]

Fill-in BA BA BA
BA SLAM [70, 89]

SWF

“ ” “ ”

$$p\left(\boldsymbol{x}_1, \ldots \boldsymbol{x}_4, \boldsymbol{y}_1, \ldots \boldsymbol{y}_6\right) = p\left(\boldsymbol{x}_2, \ldots, \boldsymbol{x}_4, \boldsymbol{y}_1, \ldots \boldsymbol{y}_6 | \boldsymbol{x}_1\right) \underbrace{p\left(\boldsymbol{x}_1\right)}. \tag{10.1}$$

VO
g2o Ceres * SLAM

10.2 Pose Graph

10.2.1 Pose Graph

Graph Figure 10-3
BA BA CPU SLAM
Graph Pose Pose Graph[93–95] Pose Pose Graph Pose

* g2o Ceres

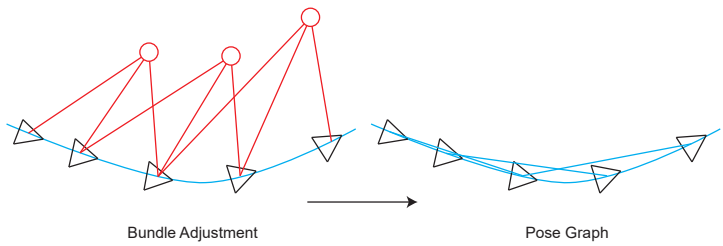


Figure 10-3: Pose Graph Bundle Adjustment Pose Graph

10.2.2 Pose Graph

Pose Graph $\boldsymbol{T}_1, \cdots, \boldsymbol{T}_n$ GPS IMU $\boldsymbol{T}_i$ $\boldsymbol{T}_j$

$$\Delta\boldsymbol{\xi}_{ij} = \boldsymbol{\xi}_i^{-1} \circ \boldsymbol{\xi}_j = \ln\left(\boldsymbol{T}_i^{-1}\boldsymbol{T}_j\right)^\vee, \tag{10.2}$$

$$\boldsymbol{T}_{ij} = \boldsymbol{T}_i^{-1}\boldsymbol{T}_j. \tag{10.3}$$

$\boldsymbol{T}_{ij}$ $\boldsymbol{e}_{ij}$

$$\boldsymbol{e}_{ij} = \Delta\boldsymbol{\xi}_{ij} \ln\left(\boldsymbol{T}_{ij}^{-1}\boldsymbol{T}_i^{-1}\boldsymbol{T}_j\right)^\vee \tag{10.4}$$

$\boldsymbol{\xi}_i$ $\boldsymbol{\xi}_j$ $\boldsymbol{e}_{ij}$ $\boldsymbol{\xi}_i$ $\boldsymbol{\xi}_j$ $\boldsymbol{\delta\xi}_i$ $\boldsymbol{\delta\xi}_j$

$$\hat{\boldsymbol{e}}_{ij} = \ln\left(\boldsymbol{T}_{ij}^{-1}\boldsymbol{T}_i^{-1}\exp((-\boldsymbol{\delta\xi}_i)^\wedge)\exp(\delta\boldsymbol{\xi}_j^\wedge)\boldsymbol{T}_j\right)^\vee. \tag{10.5}$$

BCH 4 (4.55)

$$\exp\left((\mathrm{Ad}(\boldsymbol{T})\boldsymbol{\xi})^\wedge\right) = \boldsymbol{T}\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{T}^{-1}. \tag{10.6}$$

$$\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{T} = \boldsymbol{T}\exp\left(\left(\mathrm{Ad}(\boldsymbol{T}^{-1})\boldsymbol{\xi}\right)^\wedge\right). \tag{10.7}$$

“ ” $\boldsymbol{T}$

$$\begin{aligned}
\hat{\boldsymbol{e}}_{ij} &= \ln\left(\boldsymbol{T}_{ij}^{-1}\boldsymbol{T}_i^{-1}\exp((-\boldsymbol{\delta\xi}_i)^\wedge)\exp(\delta\boldsymbol{\xi}_j^\wedge)\boldsymbol{T}_j\right)^\vee \\
&= \ln\left(\boldsymbol{T}_{ij}^{-1}\boldsymbol{T}_i^{-1}\boldsymbol{T}_j\exp\left(\left(-\mathrm{Ad}(\boldsymbol{T}_j^{-1})\boldsymbol{\delta\xi}_i\right)^\wedge\right)\exp\left(\left(\mathrm{Ad}(\boldsymbol{T}_j^{-1})\boldsymbol{\delta\xi}_j\right)^\wedge\right)\right)^\vee \\
&\approx \ln\left(\boldsymbol{T}_{ij}^{-1}\boldsymbol{T}_i^{-1}\boldsymbol{T}_j\left[\boldsymbol{I} - (\mathrm{Ad}(\boldsymbol{T}_j^{-1})\boldsymbol{\delta\xi}_i)^\wedge + (\mathrm{Ad}(\boldsymbol{T}_j^{-1})\boldsymbol{\delta\xi}_j)^\wedge\right]\right)^\vee \\
&\approx \boldsymbol{e}_{ij} + \frac{\partial\boldsymbol{e}_{ij}}{\partial\boldsymbol{\delta\xi}_i}\boldsymbol{\delta\xi}_i + \frac{\partial\boldsymbol{e}_{ij}}{\partial\boldsymbol{\delta\xi}_j}\boldsymbol{\delta\xi}_j
\end{aligned}. \tag{10.8}$$

$\boldsymbol{T}_i$

$$\frac{\partial\boldsymbol{e}_{ij}}{\partial\boldsymbol{\delta\xi}_i} = -\boldsymbol{\mathcal{J}}_r^{-1}(\boldsymbol{e}_{ij})\mathrm{Ad}(\boldsymbol{T}_j^{-1}). \tag{10.9}$$

$\boldsymbol{T}_j$

$$\frac{\partial\boldsymbol{e}_{ij}}{\partial\boldsymbol{\delta\xi}_j} = \boldsymbol{\mathcal{J}}_r^{-1}(\boldsymbol{e}_{ij})\mathrm{Ad}(\boldsymbol{T}_j^{-1}). \tag{10.10}$$

4 $\mathfrak{se}(3)$ $\mathcal{J}_r$ $\boldsymbol{I}$

$$\mathcal{J}_r^{-1}(\boldsymbol{e}_{ij}) \approx \boldsymbol{I} + \frac{1}{2}\begin{bmatrix} \boldsymbol{\phi}_{\boldsymbol{e}}^{\wedge} & \boldsymbol{\rho}_{\boldsymbol{e}}^{\wedge} \\ \mathbf{0} & \boldsymbol{\phi}_{\boldsymbol{e}}^{\wedge} \end{bmatrix}. \tag{10.11}$$

$\mathcal{J}_r$ $\boldsymbol{I}$

—— $\mathcal{E}$

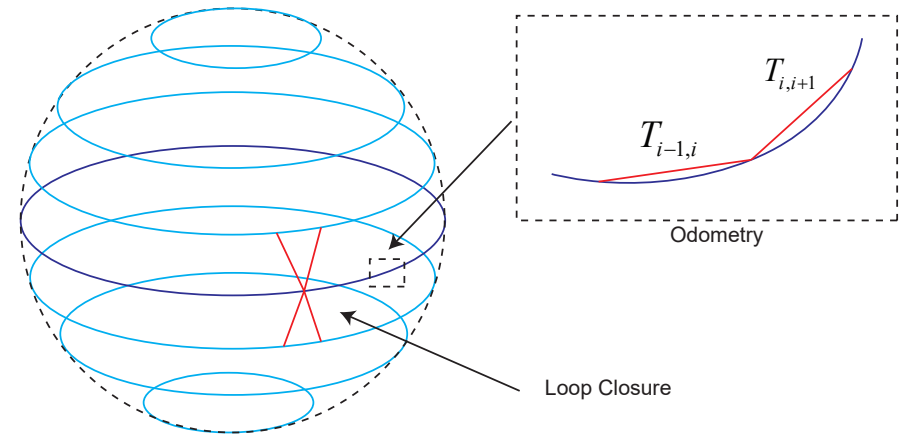
$$\min \frac{1}{2}\sum_{i,j\in\mathcal{E}} \boldsymbol{e}_{ij}^{\mathrm{T}}\boldsymbol{\Sigma}_{ij}^{-1}\boldsymbol{e}_{ij}. \tag{10.12}$$

— Ceres g2o

10.3

10.3.1 g2o

g2o g2o_viewer slambook2/ch10/sphere.g2o Figure 10-4



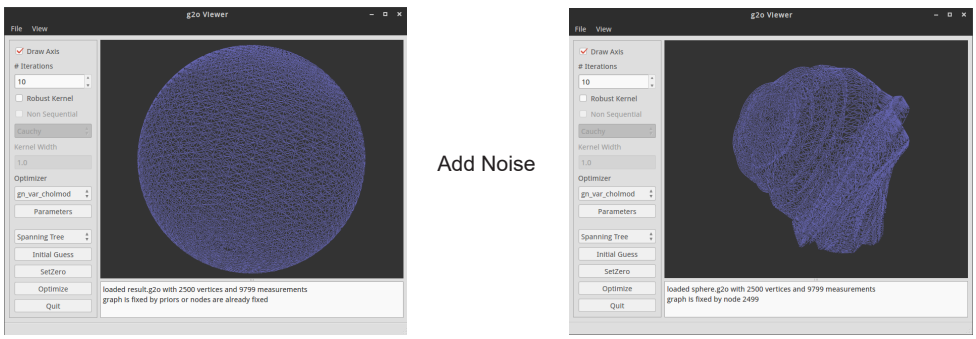


Figure 10-4: g2o

g2o create sphere 2500 Figure 10-4 $t-1$ t odometry loop closure 4

g2o_viewer optimize s

```
VERTEX_SE3:QUAT 0 -0.125664 -1.53894e-17 99.9999 0.706662 4.32706e-17 0.707551 -4.3325e-17
......
EDGE_SE3:QUAT 1524 1574 -0.210399 -0.0101193 -6.28806 -0.00122939 0.0375067 -2.85291e-05 0.999296 10000 0 0 0 0 0 10000 0 0 0 0 10000 0 0 0 40000 0 0 40000 0 40000
```

VERTEX_SE3 g2o ID $t_x, t_y, t_z, q_x, q_y, q_z, q_w$ 3 4 g2o g2o g2o slambook2/ch10/pose_graph_g2o_SE3.cpp — result.g2o

Listing 10.1: slambook2/ch10/pose_graph_g2o_SE3.cpp

```cpp
#include <iostream>
#include <fstream>
#include <string>

#include <g2o/types/slam3d/types_slam3d.h>
#include <g2o/core/block_solver.h>
#include <g2o/core/optimization_algorithm_levenberg.h>
#include <g2o/solvers/eigen/linear_solver_eigen.h>

using namespace std;

/**********************************************
* □□□□□□□□g2o □□□□□□□solver
* sphere.□□□□□□□□g2oPose □□□□□□□□graph
* □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□load
* □□□□g2o/types/slam3d□□/□□□□□□□□□□□□□□□□□□□SE3.
* **********************************************/

int main(int argc, char **argv) {
    if (argc != 2) {
        cout << "Usage: pose_graph_g2o_SE3 sphere.g2o" << endl;
        return 1;
    }
    ifstream fin(argv[1]);
    if (!fin) {
        cout << "file " << argv[1] << " does not exist." << endl;
        return 1;
    }

    // □□g2o
    typedef g2o::BlockSolver<g2o::BlockSolverTraits<6, 6>> BlockSolverType;
    typedef g2o::LinearSolverEigen<BlockSolverType::PoseMatrixType> LinearSolverType;
    auto solver = new g2o::OptimizationAlgorithmLevenberg(
    g2o::make_unique<BlockSolverType>(g2o::make_unique<LinearSolverType>()));
    g2o::SparseOptimizer optimizer;     // □□□
    optimizer.setAlgorithm(solver);   // □□□□□
    optimizer.setVerbose(true);       // □□□□□□

    int vertexCnt = 0, edgeCnt = 0; // □□□□□□□
    while (!fin.eof()) {
        string name;
        fin >> name;
        if (name == "VERTEX_SE3:QUAT") {
            // SE3 □□
            g2o::VertexSE3 *v = new g2o::VertexSE3();
            int index = 0;
            fin >> index;
            v->setId(index);
            v->read(fin);
            optimizer.addVertex(v);
            vertexCnt++;
            if (index == 0)
            v->setFixed(true);
        } else if (name == "EDGE_SE3:QUAT") {
            // SE3-SE3 □
            g2o::EdgeSE3 *e = new g2o::EdgeSE3();
            int idx1, idx2;     // □□□□□□□
            fin >> idx1 >> idx2;
```

```
            e->setId(edgeCnt++);
            e->setVertex(0, optimizer.vertices()[idx1]);
            e->setVertex(1, optimizer.vertices()[idx2]);
            e->read(fin);
            optimizer.addEdge(e);
        }
        if (!fin.good()) break;
    }

    cout << "read total " << vertexCnt << " vertices, " << edgeCnt << " edges." <<
         endl;

    cout << "optimizing ..." << endl;
    optimizer.initializeOptimization();
    optimizer.optimize(30);

    cout << "saving optimization results ..." << endl;
    optimizer.save("result.g2o");

    return 0;
}
```

6×6 — 30

Listing 10.2:

```
$ build/pose_graph_g2o_SE3 sphere.g2o
read total 2500 vertices, 9799 edges.
optimizing ...
iteration= 0  chi2= 1023011093.851879 edges= 9799 schur= 0 lambda= 805.622433
     levenbergIter= 1
iteration= 1  chi2= 385118688.233188  time= 0.863567 cumTime= 1.71545  edges= 9799
     schur= 0 lambda= 537.081622 levenbergIter= 1
iteration= 2  chi2= 166223726.693659  time= 0.861235 cumTime= 2.57668  edges= 9799
     schur= 0 lambda= 358.054415 levenbergIter= 1
iteration= 3  chi2= 86610874.269316   time= 0.844105 cumTime= 3.42079  edges= 9799
     schur= 0 lambda= 238.702943 levenbergIter= 1
iteration= 4  chi2= 40582782.710190   time= 0.862221 cumTime= 4.28301  edges= 9799
     schur= 0 lambda= 159.135295 levenbergIter= 1
......
iteration= 28 chi2= 45095.174398 time= 0.869451 cumTime= 30.0809 edges= 9799 schur= 0
     lambda= 0.003127 levenbergIter= 1
iteration= 29 chi2= 44811.248504 time= 1.76326  cumTime= 31.8442 edges= 9799 schur= 0
     lambda= 0.003785 levenbergIter= 2
saving optimization results ...
```

g2o_viewer result.g2o Figure 10-5
g2o_viewer Optimize

10.3.2

Sophus Sophus g2o

Listing 10.3: slambook2/ch10/pose_graph_g2o_lie_algebra.cpp

```
typedef Matrix<double, 6, 6> Matrix6d;

// □□□□□J_R□□□^{-1}
Matrix6d JRInv(const SE3d &e) {
    Matrix6d J;
    J.block(0, 0, 3, 3) = SO3d::hat(e.so3().log());
    J.block(0, 3, 3, 3) = SO3d::hat(e.translation());
    J.block(3, 0, 3, 3) = Matrix3d::Zero(3, 3);
    J.block(3, 3, 3, 3) = SO3d::hat(e.so3().log());
    J = J * 0.5 + Matrix6d::Identity();
    return J;
}

// □□□□□
typedef Matrix<double, 6, 1> Vector6d;
```

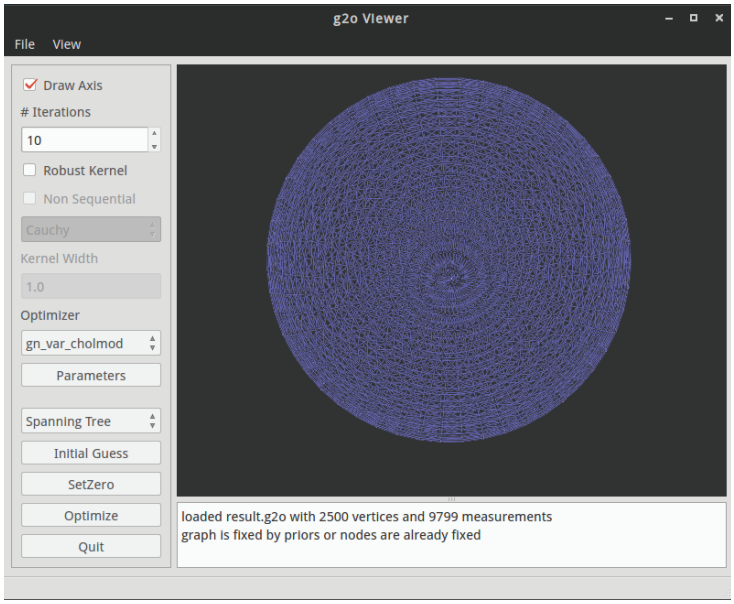


Figure 10-5: g2o

```
16
17  class VertexSE3LieAlgebra : public g2o::BaseVertex<6, SE3d> {
18      public:
19      EIGEN_MAKE_ALIGNED_OPERATOR_NEW
20
21      virtual bool read(istream &is) override {
22          double data[7];
23          for (int i = 0; i < 7; i++)
24          is >> data[i];
25          setEstimate(SE3d(
26          Quaterniond(data[6], data[3], data[4], data[5]),
27          Vector3d(data[0], data[1], data[2])
28          ));
29      }
30
31      virtual bool write(ostream &os) const override {
32          os << id() << " ";
33          Quaterniond q = _estimate.unit_quaternion();
34          os << _estimate.translation().transpose() << " ";
35          os << q.coeffs()[0] << " " << q.coeffs()[1] << " " << q.coeffs()[2] << " " <<
                q.coeffs()[3] << endl;
36          return true;
37      }
38
39      virtual void setToOriginImpl() override {
40          _estimate = SE3d();
41      }
42
43      // □□□□
44      virtual void oplusImpl(const double *update) override {
45          Vector6d upd;
46          upd << update[0], update[1], update[2], update[3], update[4], update[5];
47          _estimate = SE3d::exp(upd) * _estimate;
48      }
49  };
50
51  // □□□□□□□□□□
52  class EdgeSE3LieAlgebra : public g2o::BaseBinaryEdge<6, SE3d, VertexSE3LieAlgebra,
        VertexSE3LieAlgebra> {
53      public:
54      EIGEN_MAKE_ALIGNED_OPERATOR_NEW
```

```
55
56      virtual bool read(istream &is) override {
57          double data[7];
58          for (int i = 0; i < 7; i++)
59          is >> data[i];
60          Quaterniond q(data[6], data[3], data[4], data[5]);
61          q.normalize();
62          setMeasurement(SE3d(q, Vector3d(data[0], data[1], data[2])));
63          for (int i = 0; i < information().rows() && is.good(); i++)
64          for (int j = i; j < information().cols() && is.good(); j++) {
65              is >> information()(i, j);
66              if (i != j)
67              information()(j, i) = information()(i, j);
68          }
69          return true;
70      }
71
72      virtual bool write(ostream &os) const override {
73          VertexSE3LieAlgebra *v1 = static_cast<VertexSE3LieAlgebra *> (_vertices[0]);
74          VertexSE3LieAlgebra *v2 = static_cast<VertexSE3LieAlgebra *> (_vertices[1]);
75          os << v1->id() << " " << v2->id() << " ";
76          SE3d m = _measurement;
77          Eigen::Quaterniond q = m.unit_quaternion();
78          os << m.translation().transpose() << " ";
79          os << q.coeffs()[0] << " " << q.coeffs()[1] << " " << q.coeffs()[2] << " " <<
                q.coeffs()[3] << " ";
80
81          // information matrix
82          for (int i = 0; i < information().rows(); i++)
83          for (int j = i; j < information().cols(); j++) {
84              os << information()(i, j) << " ";
85          }
86          os << endl;
87          return true;
88      }
89
90      // □□□□□□□□□□□□
91      virtual void computeError() override {
92          SE3d v1 = (static_cast<VertexSE3LieAlgebra *> (_vertices[0]))->estimate();
93          SE3d v2 = (static_cast<VertexSE3LieAlgebra *> (_vertices[1]))->estimate();
94          _error = (_measurement.inverse() * v1.inverse() * v2).log();
95      }
96
97      // □□□□□
98      virtual void linearizeOplus() override {
99          SE3d v1 = (static_cast<VertexSE3LieAlgebra *> (_vertices[0]))->estimate();
100         SE3d v2 = (static_cast<VertexSE3LieAlgebra *> (_vertices[1]))->estimate();
101         Matrix6d J = JRInv(SE3d::exp(_error));
102         // □□□□□□□JI
103         _jacobianOplusXi = -J * v2.inverse().Adj();
104         _jacobianOplusXj = J * v2.inverse().Adj();
105     }
106 };
```

g2o read write “ ” g2o SE3 g2o_viewer Sophus g2o JRInv() $\mathcal{J}_r^{-1}$ $\boldsymbol{I}$ oplusImpl g2o

Listing 10.4:

```
1 $ build/pose_graph_g2o_lie sphere.g2o
2 read total 2500 vertices, 9799 edges.
3 optimizing ...
4 iteration= 0   chi2= 626657736.014949  time= 0.549125  cumTime= 0.549125   edges= 9799
        schur= 0  lambda= 6706.585223   levenbergIter= 1
5 iteration= 1   chi2= 233236853.521434  time= 0.510685  cumTime= 1.05981  edges= 9799
      schur= 0  lambda= 2235.528408   levenbergIter= 1
6 iteration= 2   chi2= 142629876.750105  time= 0.557893  cumTime= 1.6177   edges= 9799
      schur= 0  lambda= 745.176136  levenbergIter= 1
7 iteration= 3   chi2= 84218288.615592   time= 0.525079  cumTime= 2.14278  edges= 9799
      schur= 0  lambda= 248.392045  levenbergIter= 1
8 ......
```

23 30 * result_lie.g2o Figure 10-6

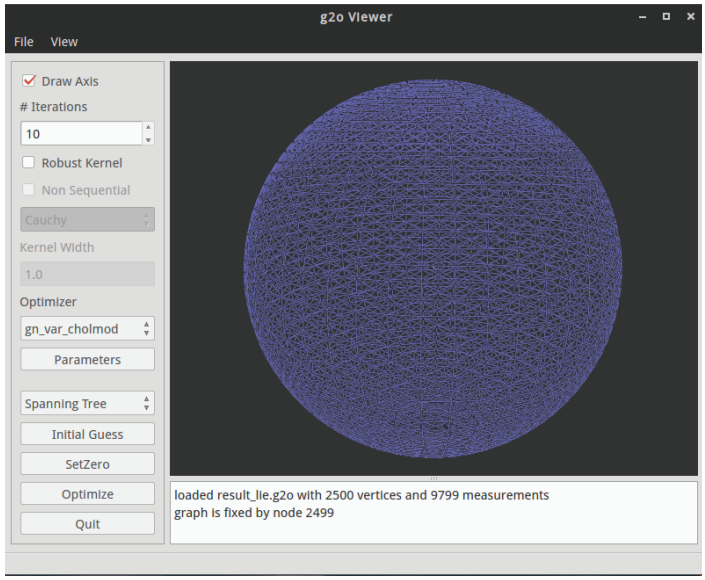


Figure 10-6:

g2o_viewer Optimize g2o SE3

```
loaded result_lie.g2o with 2500 vertices and 9799 measurements
graph is fixed by node 2499
# Using CHOLMOD poseDim -1 landMarkDim -1 blockordering 0
Preparing (no marginalization of Landmarks)
iteration= 0 chi2= 44360.509723 time= 0.567504 cumTime= 0.567504 edges= 9799 schur= 0
iteration= 1 chi2= 44360.471110 time= 0.595993 cumTime= 1.1635   edges= 9799 schur= 0
iteration= 2 chi2= 44360.471110 time= 0.582909 cumTime= 1.74641  edges= 9799 schur= 0
```

SE3 44360 30 44811 † $\mathcal{J}_r^{-1}$

10.3.3

Odometry Loop Closure SLAM “ ” 2,500 10,000
— “ ” Loopy motion “ ”
PTAM[96] Tracking Mapping ——

1. $\Delta\boldsymbol{\xi}_{ij} = \boldsymbol{\xi}_i \circ \boldsymbol{\xi}_j^{-1}$
2.
3. g2o Ceres “ ”
4. “ ” g2o gtsam

---

*
† “ ”

5.* iSAM

Chapter 11

1.
2.
3. DBoW3

SLAM SLAM SLAM

11.1

11.1.1

VO SLAM

Pose Graph Figure 11-1(a)

$\boldsymbol{x}_2, \boldsymbol{x}_2 - \boldsymbol{x}_3$ $\boldsymbol{x}_1$ $\boldsymbol{x}_2$ $\boldsymbol{x}_1$ $\boldsymbol{x}_3$ $\boldsymbol{x}_2$ Figure 11-1 (b)

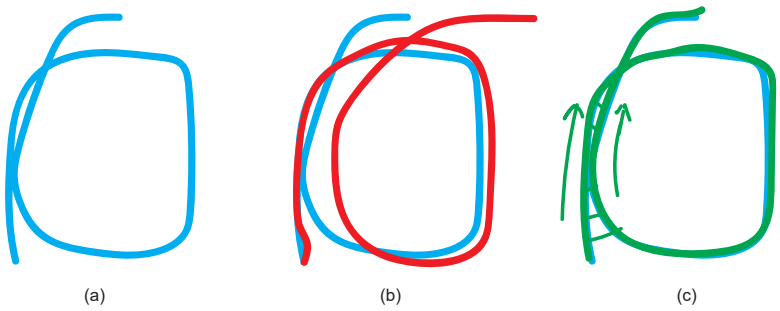


Figure 11-1: a b Pose Graph c Pose Graph

“ ”

$\boldsymbol{x}_1 - \boldsymbol{x}_{100}$

Graph Pose Graph —— “ ” —

—

SLAM

11.1.2

—— “ ” N C_N^2 $O(N^2)$

[97] “ ” Odometry based Appearance based [98]—— “ ”

SLAM 21 SL

GPS GPS

$\boldsymbol{A}$ $\boldsymbol{B}$ $s(\boldsymbol{A}, \boldsymbol{B})$

$$s(\boldsymbol{A}, \boldsymbol{B}) = \|\boldsymbol{A} - \boldsymbol{B}\|. \tag{11.1}$$

1.

2.

$\boldsymbol{A} - \boldsymbol{B}$ “ ” “ ”

Aliasing Perceptual Variability

11.1.3

“ ” “ ”

1 4

Table 11-1:

\		
	True Positive False Negative	False Positive True Negative

/ False Positive False Negative Figure 11-2 TP True Positive TP TN FP FN TP TN FP FN Precisio & Recall

$$\text{Precision} = \text{TP}/(\text{TP} + \text{FP}), \quad \text{Recall} = \text{TP}/(\text{TP} + \text{FN}). \tag{11.2}$$

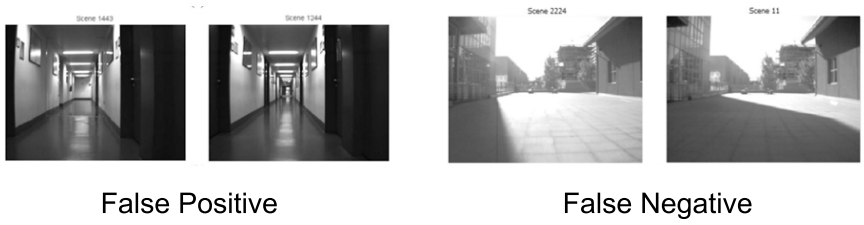


Figure 11-2:

“ ”____

P R Precision-Recall Figure 11-3 100% 50%

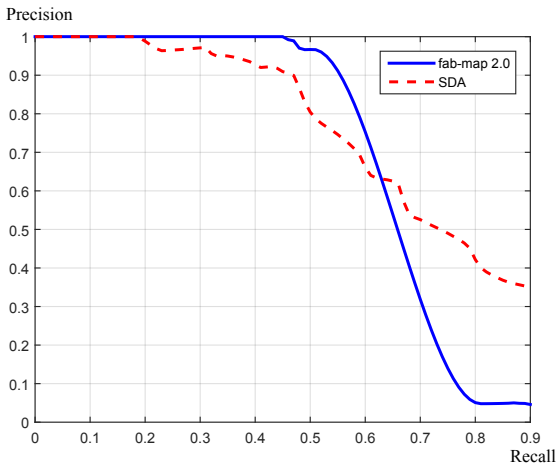


Figure 11-3: — [100]

*

SLAM Pose Graph SLAM

—

$\boldsymbol{A}-\boldsymbol{B}$ False Positive False Negative “ ”

11.2

VO VO

Bag-of-Words BoW “ ”

1. “ ”“ ”“ ” —— BoW “ ” Word “ ” Dictionary
2. ——
3.

“ ” “ ”“ ”“ ” w_1, w_2, w_3 A

$$A = 1 \cdot w_1 + 1 \cdot w_2 + 0 \cdot w_3. \tag{11.3}$$

$[1,1,0]^{\mathrm{T}}$ A “ ” “ ” “ ”

Bag-of-Words List-of-Words Words

$[2,0,1]^{\mathrm{T}}$ B “ ” $[1,0,1]^{\mathrm{T}}$ $\boldsymbol{a}, \boldsymbol{b} \in$

$\mathbb{R}^W$

$$s(\boldsymbol{a}, \boldsymbol{b}) = 1 - \frac{1}{W} \|\boldsymbol{a} - \boldsymbol{b}\|_1 . \tag{11.4}$$

L_1 1 $\boldsymbol{a}$ 0 $\boldsymbol{b}$ 1 0

1.
2.

11.3

11.3.1

Clustering

Unsupervised ML BoW N k

means K [101]

K-means N k K-means

1. k $c_1, \cdots, c_k$
2.
3.

*

4. 2

K-means K-means++[102]
means k

——— *

$O(n)$ Fabmap[103–105] Chou-
Liu tree[106] [107]
[107] k Figure 11-4 k-means N d k †

1. k-means k k-means++
2. k
3. Words

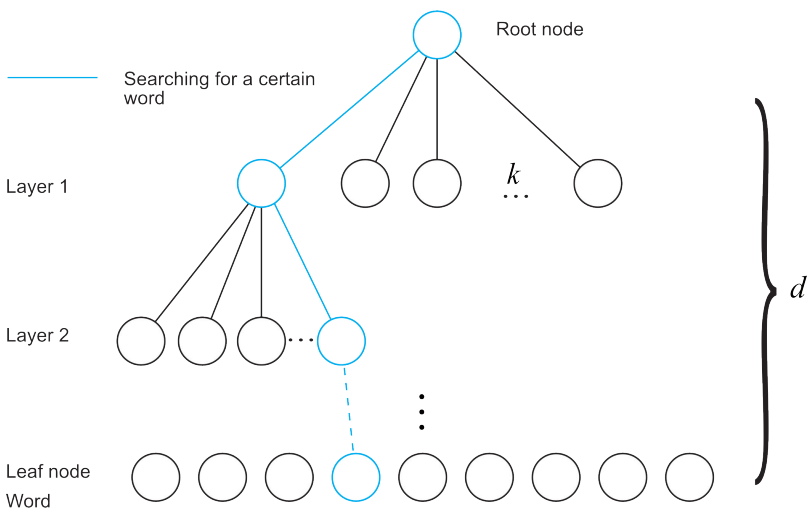


Figure 11-4: K K-means

k d k^d d

11.3.2

VO ORB ORB
TUM 10 slambook2/ch11/data Figure 11-5
BoW ——— GitHub
BoW DBoW3‡ `https://github.com/rmsalinas/DBow3` 3rdparty

Listing 11.1: slambook2/ch11/feature_training.cpp

*
† k d k-d [108]
‡ OpenCV3

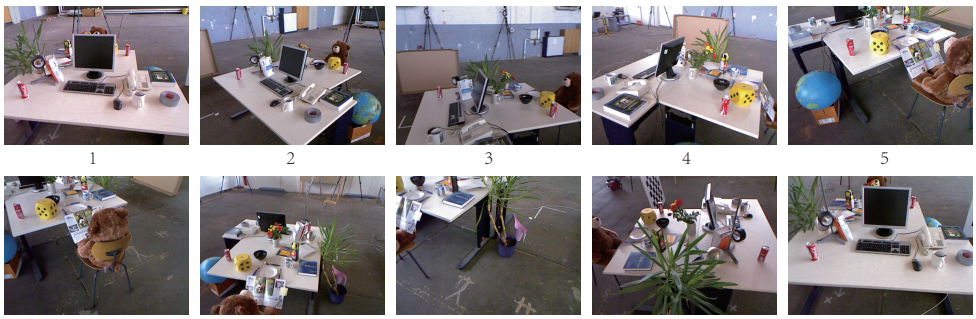


Figure 11-5: 10

```
int main(int argc, char **argv) {
  // read the image
  cout << "reading images... " << endl;
  vector<Mat> images;
  for (int i = 0; i < 10; i++) {
    string path = "./data/" + to_string(i + 1) + ".png";
    images.push_back(imread(path));
  }
  // detect ORB features
  cout << "detecting ORB features ... " << endl;
  Ptr<Feature2D> detector = ORB::create();
  vector<Mat> descriptors;
  for (Mat &image:images) {
    vector<KeyPoint> keypoints;
    Mat descriptor;
    detector->detectAndCompute(image, Mat(), keypoints, descriptor);
    descriptors.push_back(descriptor);
  }

  // create vocabulary
  cout << "creating vocabulary ... " << endl;
  DBoW3::Vocabulary vocab;
  vocab.create(descriptors);
  cout << "vocabulary info: " << vocab << endl;
  vocab.save("vocabulary.yml.gz");
  cout << "done" << endl;

  return 0;
}
```

DBoW3 10 ORB vector DBoW3 DBoW3::Vocabulary
10, $d = 5$ 100,000 500

```
$ build/feature_training
reading images...
detecting ORB features ...
creating vocabulary ...
vocabulary info: Vocabulary: k = 10, L = 5, Weighting = tf-idf, Scoring = L1-norm,
     Number of words = 4983
done
```

k 10 L 5* 4983 Weighting Scoring Weighting Scoring

* L d

11.4

11.4.1

f_i w_j—— f_i w_j N N
of-Words Bag
“ ” —— “ ”“ ” “ ”“ ”

TF-IDF Term Frequency–Inverse Document Frequency [109, 110] — * BoW TF
IDF w_i IDF n w_i n_i IDF

$$\mathrm{IDF}_i = \log \frac{n}{n_i}. \tag{11.5}$$

TF A w_i n_i n TF

$$\mathrm{TF}_i = \frac{n_i}{n}. \tag{11.6}$$

w_i TF IDF

$$\eta_i = \mathrm{TF}_i \times \mathrm{IDF}_i. \tag{11.7}$$

A Bag-of-Words

$$A = \{(w_1, \eta_1), (w_2, \eta_2), \ldots, (w_N, \eta_N)\} \triangleq \boldsymbol{v}_A. \tag{11.8}$$

$\boldsymbol{v}_A$ $\boldsymbol{v}_A$ A $\boldsymbol{v}_A$ A TF-IDF
$\boldsymbol{v}_A$ $\boldsymbol{v}_B$ [111] L_1

$$s\left(\boldsymbol{v}_A - \boldsymbol{v}_B\right) = 2\sum_{i=1}^{N} |\boldsymbol{v}_{Ai}| + |\boldsymbol{v}_{Bi}| - |\boldsymbol{v}_{Ai} - \boldsymbol{v}_{Bi}|. \tag{11.9}$$

11.4.2

Bag-of-Words

Listing 11.2: slambook/ch12/loop_closure.cpp

```
int main(int argc, char **argv) {
  // read the images and database
  cout << "reading database" << endl;
  DBoW3::Vocabulary vocab("./vocabulary.yml.gz");
  // DBoW3::Vocabulary vocab("./vocab_larger.yml.gz");  // use large vocab if you want
      :
  if (vocab.empty()) {
    cerr << "Vocabulary does not exist." << endl;
    return 1;
  }
  cout << "reading images... " << endl;
  vector<Mat> images;
  for (int i = 0; i < 10; i++) {
    string path = "./data/" + to_string(i + 1) + ".png";
    images.push_back(imread(path));
  }

  // NOTE: in this case we are comparing images with a vocabulary generated by
       themselves, this may lead to overfit.
```

* TF-IDF

```
  // detect ORB features
  cout << "detecting ORB features ... " << endl;
  Ptr<Feature2D> detector = ORB::create();
  vector<Mat> descriptors;
  for (Mat &image:images) {
    vector<KeyPoint> keypoints;
    Mat descriptor;
    detector->detectAndCompute(image, Mat(), keypoints, descriptor);
    descriptors.push_back(descriptor);
  }

  // we can compare the images directly or we can compare one image to a database
  // images :
  cout << "comparing images with images " << endl;
  for (int i = 0; i < images.size(); i++) {
    DBoW3::BowVector v1;
    vocab.transform(descriptors[i], v1);
    for (int j = i; j < images.size(); j++) {
      DBoW3::BowVector v2;
      vocab.transform(descriptors[j], v2);
      double score = vocab.score(v1, v2);
      cout << "image " << i << " vs image " << j << " : " << score << endl;
    }
    cout << endl;
  }

  // or with database
  cout << "comparing images with database " << endl;
  DBoW3::Database db(vocab, false, 0);
  for (int i = 0; i < descriptors.size(); i++)
  db.add(descriptors[i]);
  cout << "database info: " << db << endl;
  for (int i = 0; i < descriptors.size(); i++) {
    DBoW3::QueryResults ret;
    db.query(descriptors[i], ret, 4);      // max result=4
    cout << "searching for image " << i << " returns " << ret << endl << endl;
  }
  cout << "done." << endl;
}
```

—— Bag-of-Words

Listing 11.3:

```
$ build/feature_training
reading database
reading images...
detecting ORB features ...
comparing images with images
desp 0 size: 500
transform image 0 into BoW vector: size = 455
key value pair = <1, 0.00155622>, <3, 0.00222645>, <12, 0.00222645>, <13, 0.00222645>,
      <14, 0.00222645>, <22, 0.00222645>, <33, 0.00222645>, <37, 0.00155622>, <38,
     0.00222645>, <39, 0.00222645>, <43, 0.00222645>, <57, 0.00155622> ......
```

BoW ID DBoW3

Listing 11.4:

```
image 0 vs image 0 : 1
image 0 vs image 1 : 0.0234552
image 0 vs image 2 : 0.0225237
image 0 vs image 3 : 0.0254611
image 0 vs image 4 : 0.0253451
image 0 vs image 5 : 0.0272257
image 0 vs image 6 : 0.0217745
image 0 vs image 7 : 0.0231948
image 0 vs image 8 : 0.0311284
image 0 vs image 9 : 0.0525447
```

DBoW

Listing 11.5:

```
searching for image 0 returns 4 results:
<EntryId: 0, Score: 1>
<EntryId: 9, Score: 0.0525447>
<EntryId: 8, Score: 0.0311284>
<EntryId: 5, Score: 0.0272257>

searching for image 1 returns 4 results:
<EntryId: 1, Score: 1>
<EntryId: 2, Score: 0.0339641>
<EntryId: 8, Score: 0.0299387>
<EntryId: 3, Score: 0.0256668>

searching for image 2 returns 4 results:
<EntryId: 2, Score: 1>
<EntryId: 7, Score: 0.036092>
<EntryId: 9, Score: 0.0348702>
<EntryId: 1, Score: 0.0339641>

searching for image 3 returns 4 results:
<EntryId: 3, Score: 1>
<EntryId: 9, Score: 0.0357317>
<EntryId: 8, Score: 0.0278496>
<EntryId: 5, Score: 0.0270168>

searching for image 4 returns 4 results:
<EntryId: 4, Score: 1>
<EntryId: 5, Score: 0.0493492>
<EntryId: 0, Score: 0.0253451>
<EntryId: 6, Score: 0.0253017>

searching for image 5 returns 4 results:
<EntryId: 5, Score: 1>
<EntryId: 4, Score: 0.0493492>
<EntryId: 9, Score: 0.028996>
<EntryId: 6, Score: 0.0277584>

searching for image 6 returns 4 results:
<EntryId: 6, Score: 1>
<EntryId: 8, Score: 0.0306241>
<EntryId: 5, Score: 0.0277584>
<EntryId: 3, Score: 0.0267135>

searching for image 7 returns 4 results:
<EntryId: 7, Score: 1>
<EntryId: 2, Score: 0.036092>
<EntryId: 1, Score: 0.0239091>
<EntryId: 0, Score: 0.0231948>

searching for image 8 returns 4 results:
<EntryId: 8, Score: 1>
<EntryId: 9, Score: 0.0329149>
<EntryId: 0, Score: 0.0311284>
<EntryId: 6, Score: 0.0306241>

searching for image 9 returns 4 results:
<EntryId: 9, Score: 1>
<EntryId: 0, Score: 0.0525447>
<EntryId: 3, Score: 0.0357317>
<EntryId: 2, Score: 0.0348702>
```

1 10 C++ 0 9 0.0525 0.02
1 10 100% 1 10 2%

11.5

11.5.1

“ ” “ ” SLAM

slambook2/ch11/vocab_larger.yml.gz —— 2,900 $k = 10, d = 5$ gen_vocab_large.cpp

Listing 11.6:

```
comparing images with database
database info: Database: Entries = 10, Using direct index = no. Vocabulary: k = 10, L = 5, Weighting = tf-idf, Scoring = L1-norm, Number of words = 99566
searching for image 0 returns 4 results:
<EntryId: 0, Score: 1>
<EntryId: 9, Score: 0.0320906>
<EntryId: 8, Score: 0.0103268>
<EntryId: 4, Score: 0.0066729>

searching for image 1 returns 4 results:
<EntryId: 1, Score: 1>
<EntryId: 2, Score: 0.0238409>
<EntryId: 8, Score: 0.00814409>
<EntryId: 3, Score: 0.00697527>

searching for image 2 returns 4 results:
<EntryId: 2, Score: 1>
<EntryId: 1, Score: 0.0238409>
<EntryId: 5, Score: 0.00897928>
<EntryId: 8, Score: 0.00893477>

searching for image 3 returns 4 results:
<EntryId: 3, Score: 1>
<EntryId: 5, Score: 0.0107005>
<EntryId: 8, Score: 0.00870392>
<EntryId: 6, Score: 0.00720695>

searching for image 4 returns 4 results:
<EntryId: 4, Score: 1>
<EntryId: 6, Score: 0.0069998>
<EntryId: 0, Score: 0.0066729>
<EntryId: 5, Score: 0.0062834>

searching for image 5 returns 4 results:
<EntryId: 5, Score: 1>
<EntryId: 3, Score: 0.0107005>
<EntryId: 2, Score: 0.00897928>
<EntryId: 4, Score: 0.0062834>

searching for image 6 returns 4 results:
<EntryId: 6, Score: 1>
<EntryId: 7, Score: 0.00915307>
<EntryId: 3, Score: 0.00720695>
<EntryId: 4, Score: 0.0069998>

searching for image 7 returns 4 results:
<EntryId: 7, Score: 1>
<EntryId: 6, Score: 0.00915307>
<EntryId: 8, Score: 0.00814517>
<EntryId: 1, Score: 0.00538609>

searching for image 8 returns 4 results:
<EntryId: 8, Score: 1>
<EntryId: 0, Score: 0.0103268>
<EntryId: 2, Score: 0.00893477>
<EntryId: 3, Score: 0.00870392>

searching for image 9 returns 4 results:
<EntryId: 9, Score: 1>
<EntryId: 0, Score: 0.0320906>
<EntryId: 8, Score: 0.00636511>
<EntryId: 1, Score: 0.00587605>
```

11.5.2

$s(\boldsymbol{v}_t, \boldsymbol{v}_{t-\Delta t})$

$$s(\boldsymbol{v}_t, \boldsymbol{v}_{t_j})' = s(\boldsymbol{v}_t, \boldsymbol{v}_{t_j}) / s(\boldsymbol{v}_t, \boldsymbol{v}_{t-\Delta t}). \tag{11.10}$$

3

11.5.3

n $n-2$ $n-3$

1 n $n+1$ $n+2$ 1 1 n $n+1$ $n+$
2 1 “ ”

11.5.4

[95, 112]

Pose
Graph

11.5.5

——

——

1. SURF ORB
2. K-means

“ ” [113, 114]
116] [117]

1. PR MATLAB Python
2. [103] TUM
3. DBoW3 DBoW2
4.
5. Chow-Liu
6. [118]

Chapter 12

1. SLAM
2.
3. RGB-D

SLAM SLAM

12.1

Mapping SLAM —— SLAM SLAM
SLAM Bundle Adjustment
SLAM SLAM ——
“ ” SLAM SLAM “ ” “ ” “ ”

1.
2.
3.
4. SLAM ——
5. ——
—

Figure 12-1 “ ”

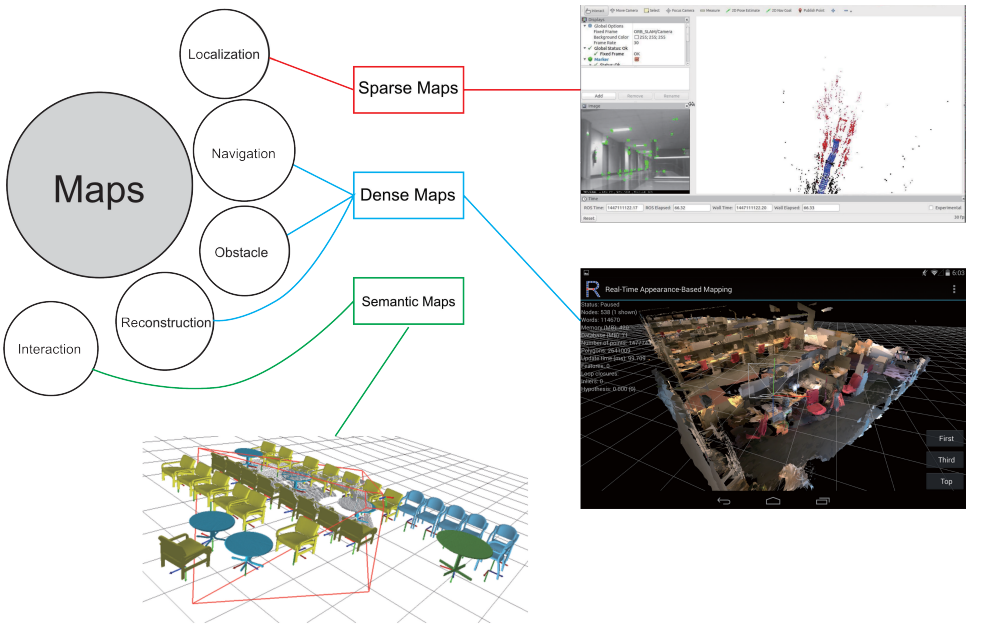


Figure 12-1: [88, 119, 120]

SLAM

12.2

12.2.1

SLAM Bearing only Range

1.

2.

3. RGB-D

Stereo Vision Moving View Stereo MVS RGB-D “ ” —
— * RGB-D RGB-D RGB-
D
SLAM

1.

2.

[121]

12.2.2

7.3 Figure 12-2 $\boldsymbol{p}_1$ $(d_{\min}, +\infty)$

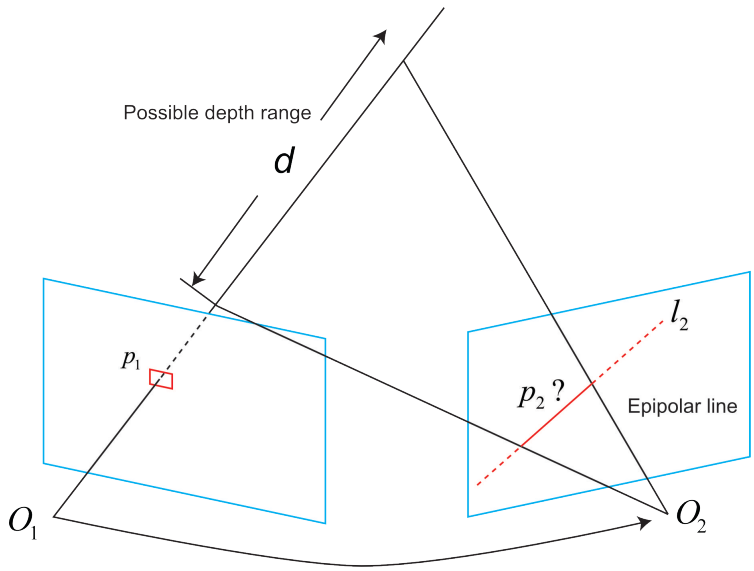


Figure 12-2:

$\boldsymbol{p}_2$ $\boldsymbol{p}_1$ $\boldsymbol{p}_1$
$\boldsymbol{p}_1$
$\boldsymbol{p}_1$ $w \times w$
—
$\boldsymbol{p}_1$ $\boldsymbol{p}_1$ $\boldsymbol{A} \in \mathbb{R}^{w \times w}$ n $\boldsymbol{B}_i, i = 1, \cdots, n$

1. SAD Sum of Absolute Difference

$$S(\boldsymbol{A}, \boldsymbol{B})_{\text{SAD}} = \sum_{i,j} |\boldsymbol{A}(i,j) - \boldsymbol{B}(i,j)|. \tag{12.1}$$

* RGB-D Fragile
†

2. SSD SSD Sum of Squared Distance

$$S(\boldsymbol{A}, \boldsymbol{B})_{\mathrm{SSD}} = \sum_{i,j} \left(\boldsymbol{A}(i,j) - \boldsymbol{B}(i,j)\right)^2. \tag{12.2}$$

3. NCC Normalized Cross Correlation

$$S(\boldsymbol{A}, \boldsymbol{B})_{\mathrm{NCC}} = \frac{\sum_{i,j} \boldsymbol{A}(i,j)\boldsymbol{B}(i,j)}{\sqrt{\sum_{i,j} \boldsymbol{A}(i,j)^2 \sum_{i,j} \boldsymbol{B}(i,j)^2}}. \tag{12.3}$$

0 1 0

– SSD NCC

123]

$\boldsymbol{A}$ $\boldsymbol{B}_i$ NCC NCC Figure 12-3

—

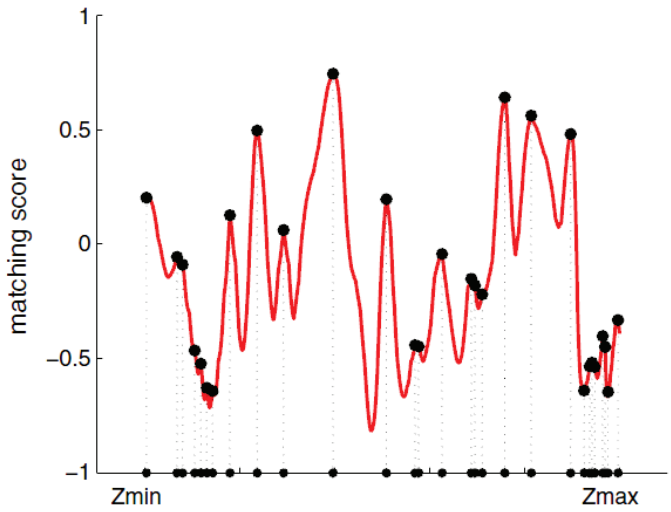


Figure 12-3: [124]

12.2.3

SLAM

[68, 124] –

d

$$P(d) = N(\mu, \sigma^2). \tag{12.4}$$

$$P(d_{\mathrm{obs}}) = N(\mu_{\mathrm{obs}}, \sigma^2_{\mathrm{obs}}). \tag{12.5}$$

*

d A d $N(\mu_{\text{fuse}}, \sigma^2_{\text{fuse}})$

$$\mu_{\text{fuse}} = \frac{\sigma^2_{\text{obs}}\mu + \sigma^2\mu_{\text{obs}}}{\sigma^2 + \sigma^2_{\text{obs}}}, \quad \sigma^2_{\text{fuse}} = \frac{\sigma^2\sigma^2_{\text{obs}}}{\sigma^2 + \sigma^2_{\text{obs}}}. \tag{12.6}$$

“ ” $\mu_{\text{obs}}, \sigma_{\text{obs}}$

$\mu_{\text{obs}}, \sigma_{\text{obs}}$ [75] [124]

Figure 12-4 $\boldsymbol{p}_1$ $\boldsymbol{p}_2$ $\boldsymbol{p}_1$ $\boldsymbol{p}_1$ $\boldsymbol{P}$ $\boldsymbol{O}_1\boldsymbol{P}$ $\boldsymbol{p}$ $\boldsymbol{O}_1\boldsymbol{O}_2$ $\boldsymbol{t}$ $\boldsymbol{O}_2\boldsymbol{P}$ $\boldsymbol{a}$ α, β $\boldsymbol{p}'$ $\boldsymbol{p}$

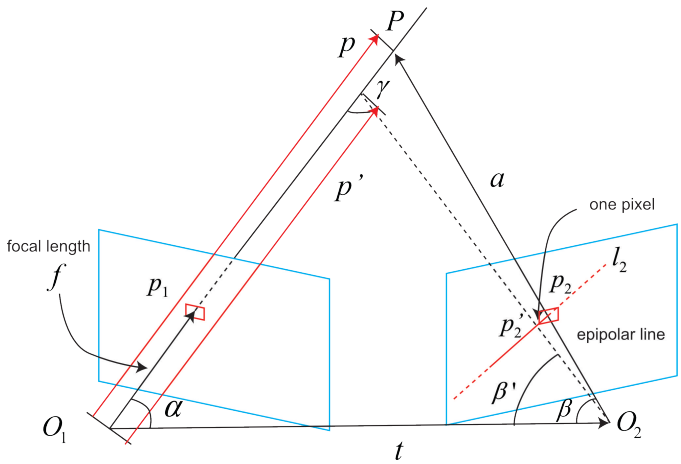


Figure 12-4:

$$\begin{aligned} \boldsymbol{a} &= \boldsymbol{p} - \boldsymbol{t} \\ \alpha &= \arccos\langle \boldsymbol{p}, \boldsymbol{t} \rangle \\ \beta &= \arccos\langle \boldsymbol{a}, -\boldsymbol{t} \rangle . \end{aligned} \tag{12.7}$$

$\boldsymbol{p}_2$ β β'

$$\begin{aligned} \beta' &= \arccos\langle \boldsymbol{O}_2\boldsymbol{p}'_2, -\boldsymbol{t} \rangle \\ \gamma &= -\alpha - \beta'. \end{aligned} \tag{12.8}$$

$\boldsymbol{p}'$

$$\|\boldsymbol{p}'\| = \|\boldsymbol{t}\| \frac{\sin\beta'}{\sin\gamma}. \tag{12.9}$$

$$\sigma_{\text{obs}} = \|\boldsymbol{p}\| - \|\boldsymbol{p}'\|. \tag{12.10}$$

1.

2.

3.

4. 2

O_1P “ ” —— z

12.3

REMODE[121, 125] 200

http://rpg.ifi.uzh.ch/datasets/remode_test_data.zip test_

data/Images 0 200 test_data

```
scene_000.png 1.086410 4.766730 −1.449960 0.789455 0.051299 −0.000779 0.611661
scene_001.png 1.086390 4.766370 −1.449530 0.789180 0.051881 −0.001131 0.611966
scene_002.png 1.086120 4.765520 −1.449090 0.788982 0.052159 −0.000735 0.612198
......
```

Figure 12-5 C

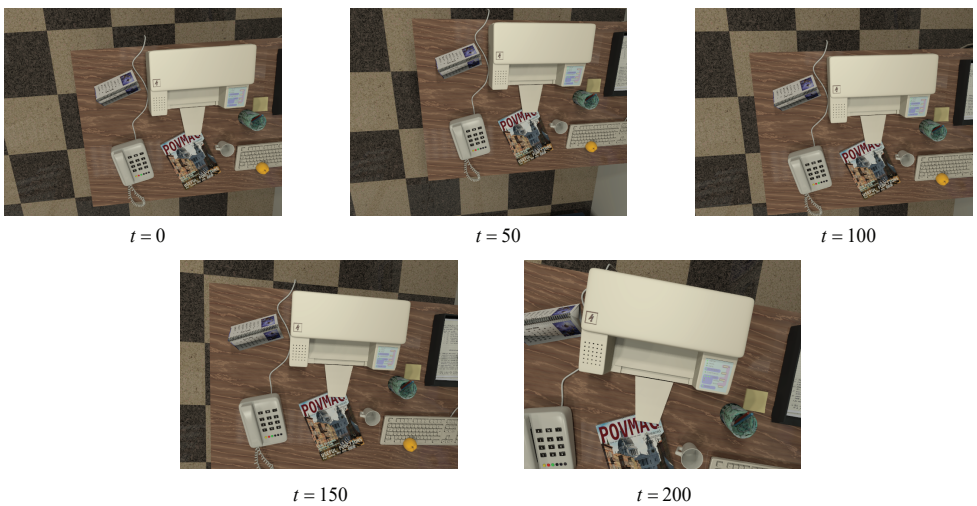


Figure 12-5:

Listing 12.1: slambook2/ch12/dense_monocular/dense_mapping.cpp

```
/**********************************************
* 
* + NCC  12.2 
* ----()
**********************************************/

// ------------------------------------------------------------------
// parameters
const int boarder = 20;         //
const int width = 640;          //
const int height = 480;         //
const double fx = 481.2f;       //
const double fy = -480.0f;
const double cx = 319.5f;
const double cy = 239.5f;
const int ncc_window_size = 3;    // NCC 
const int ncc_area = (2 * ncc_window_size + 1) * (2 * ncc_window_size + 1); // NCC
const double min_cov = 0.1;     //
const double max_cov = 10;      //
```

```
20
21 // ------------------------------------------------------------------
22 // □□□□□
23 /**
24     * □□□□□□□□□□□□□□□
25     * @param ref □□□□
26     * @param curr □□□□
27     * @param T_C_R □□□□□□□□□□□□□□□
28     * @param depth □□
29     * @param depth_cov □□□□
30     * @return □□□□
31     */
32 bool update(
33     const Mat &ref, const Mat &curr, const SE3d &T_C_R,
34     Mat &depth, Mat &depth_cov2);
35
36 /**
37     * □□□□
38     * @param ref □□□□
39     * @param curr □□□□
40     * @param T_C_R □□
41     * @param pt_ref □□□□□□□□□□
42     * @param depth_mu □□□□
43     * @param depth_cov □□□□
44     * @param pt_curr □□□
45     * @param epipolar_direction □□□□
46     * @return □□□□
47     */
48 bool epipolarSearch(
49     const Mat &ref, const Mat &curr, const SE3d &T_C_R,
50     const Vector2d &pt_ref, const double &depth_mu, const double &depth_cov,
51     Vector2d &pt_curr, Vector2d &epipolar_direction);
52
53 /**
54     * □□□□□□□□
55     * @param pt_ref □□□□□
56     * @param pt_curr □□□□□
57     * @param T_C_R □□
58     * @param epipolar_direction □□□□
59     * @param depth □□□□
60     * @param depth_cov2 □□□□
61     * @return □□□□
62     */
63 bool updateDepthFilter(
64     const Vector2d &pt_ref, const Vector2d &pt_curr, const SE3d &T_C_R,
65     const Vector2d &epipolar_direction, Mat &depth, Mat &depth_cov2);
66
67 /**
68     * □□ NCC □□
69     * @param ref □□□□
70     * @param curr □□□□
71     * @param pt_ref □□□
72     * @param pt_curr □□□
73     * @return □□         NCC
74     */
75 double NCC(const Mat &ref, const Mat &curr, const Vector2d &pt_ref, const Vector2d &
       pt_curr);
76
77 // □□□□□□□□
78 inline double getBilinearInterpolatedValue(const Mat &img, const Vector2d &pt) {
79     uchar *d = &img.data[int(pt(1, 0)) * img.step + int(pt(0, 0))];
80     double xx = pt(0, 0) - floor(pt(0, 0));
81     double yy = pt(1, 0) - floor(pt(1, 0));
82     return ((1 - xx) * (1 - yy) * double(d[0]) +
83     xx * (1 - yy) * double(d[1]) +
84     (1 - xx) * yy * double(d[img.step]) +
85     xx * yy * double(d[img.step + 1])) / 255.0;
86 }
87
```

```
int main(int argc, char **argv) {
    if (argc != 2) {
        cout << "Usage: dense_mapping path_to_test_dataset" << endl;
        return -1;
    }

    // □□□□□□□□□
    vector<string> color_image_files;
    vector<SE3d> poses_TWC;
    Mat ref_depth;
    bool ret = readDatasetFiles(argv[1], color_image_files, poses_TWC, ref_depth);
    if (ret == false) {
        cout << "Reading image files failed!" << endl;
        return -1;
    }
    cout << "read total " << color_image_files.size() << " files." << endl;

    // □□□□
    Mat ref = imread(color_image_files[0], 0);                // gray-scale image
    SE3d pose_ref_TWC = poses_TWC[0];
    double init_depth = 3.0;    // □□□□□
    double init_cov2 = 3.0;     // □□□□□
    Mat depth(height, width, CV_64F, init_depth);             // □□□
    Mat depth_cov2(height, width, CV_64F, init_cov2);         // □□□□□

    for (int index = 1; index < color_image_files.size(); index++) {
        cout << "*** loop " << index << " ***" << endl;
        Mat curr = imread(color_image_files[index], 0);
        if (curr.data == nullptr) continue;
        SE3d pose_curr_TWC = poses_TWC[index];
        SE3d pose_T_C_R = pose_curr_TWC.inverse() * pose_ref_TWC;   // T_C_W * T_W_R =
             T_C_R
        update(ref, curr, pose_T_C_R, depth, depth_cov2);
        evaludateDepth(ref_depth, depth);
        plotDepth(ref_depth, depth);
        imshow("image", curr);
        waitKey(1);
    }

    cout << "estimation returns, saving depth map ..." << endl;
    imwrite("depth.png", depth);
    cout << "done." << endl;

    return 0;
}


bool update(const Mat &ref, const Mat &curr, const SE3d &T_C_R, Mat &depth, Mat &
     depth_cov2) {
    for (int x = boarder; x < width - boarder; x++)
    for (int y = boarder; y < height - boarder; y++) {
        // □□□□□□
        if (depth_cov2.ptr<double>(y)[x] < min_cov || depth_cov2.ptr<double>(y)[x] >
             max_cov) // □□□□□□□□
        continue;
        // □□□□□□ (x,y) □□□
        Vector2d pt_curr;
        Vector2d epipolar_direction;
        bool ret = epipolarSearch(
            ref, curr, T_C_R, Vector2d(x, y), depth.ptr<double>(y)[x],           sqrt(
                 depth_cov2.ptr<double>(y)[x]), pt_curr, epipolar_direction);

        if (ret == false) // □□□□
            continue;

        // □□□□□□□□□□
        // showEpipolarMatch(ref, curr, Vector2d(x, y), pt_curr);

        // □□□□□□□□□□
        updateDepthFilter(Vector2d(x, y), pt_curr, T_C_R, epipolar_direction, depth,
             depth_cov2);
    }
```

```
155 }
156
157 // □□□□
158 // □□□□ 122 12.3 □□
159 bool epipolarSearch(
160     const Mat &ref, const Mat &curr,
161     const SE3d &T_C_R, const Vector2d &pt_ref,
162     const double &depth_mu, const double &depth_cov,
163     Vector2d &pt_curr, Vector2d &epipolar_direction) {
164     Vector3d f_ref = px2cam(pt_ref);
165     f_ref.normalize();
166     Vector3d P_ref = f_ref * depth_mu;    // □□□□ P □□
167
168     Vector2d px_mean_curr = cam2px(T_C_R * P_ref); // □□□□□□□□□□□□
169     double d_min = depth_mu - 3 * depth_cov, d_max = depth_mu + 3 * depth_cov;
170     if (d_min < 0.1) d_min = 0.1;
171     Vector2d px_min_curr = cam2px(T_C_R * (f_ref * d_min));    // □□□□□□□□□□□□
172     Vector2d px_max_curr = cam2px(T_C_R * (f_ref * d_max));    // □□□□□□□□□□□□
173
174     Vector2d epipolar_line = px_max_curr - px_min_curr;    // □□□□□□□□□
175     epipolar_direction = epipolar_line;        // □□□□
176     epipolar_direction.normalize();
177     double half_length = 0.5 * epipolar_line.norm();    // □□□□□□□□□
178     if (half_length > 100) half_length = 100;   // □□□□□□□□□□□□□
179
180     // □□□□□□□□□□□□□□□□□□
181     // showEpipolarLine( ref, curr, pt_ref, px_min_curr, px_max_curr );
182
183     // □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
184     double best_ncc = -1.0;
185     Vector2d best_px_curr;
186     for (double l = -half_length; l <= half_length; l += 0.7) { // l+=sqrt(2)
187         Vector2d px_curr = px_mean_curr + l * epipolar_direction;  // □□□□
188         if (!inside(px_curr))
189         continue;
190         // □□□□□□□□□□□□ NCC
191         double ncc = NCC(ref, curr, pt_ref, px_curr);
192         if (ncc > best_ncc) {
193             best_ncc = ncc;
194             best_px_curr = px_curr;
195         }
196     }
197     if (best_ncc < 0.85f)      // □□□ NCC □□□□□
198         return false;
199     pt_curr = best_px_curr;
200     return true;
201 }
202
203 double NCC(
204     const Mat &ref, const Mat &curr,
205     const Vector2d &pt_ref, const Vector2d &pt_curr) {
206     // □□□□□□□□□□-
207     // □□□□
208     double mean_ref = 0, mean_curr = 0;
209     vector<double> values_ref, values_curr; // □□□□□□□□□□□□
210     for (int x = -ncc_window_size; x <= ncc_window_size; x++)
211     for (int y = -ncc_window_size; y <= ncc_window_size; y++) {
212         double value_ref = double(ref.ptr<uchar>(int(y + pt_ref(1, 0)))[int(x + pt_ref
                (0, 0))]) / 255.0;
213         mean_ref += value_ref;
214
215         double value_curr = getBilinearInterpolatedValue(curr, pt_curr + Vector2d(x, y
                ));
216         mean_curr += value_curr;
217
218         values_ref.push_back(value_ref);
219         values_curr.push_back(value_curr);
220     }
221
222     mean_ref /= ncc_area;
223     mean_curr /= ncc_area;
```

```
224
225     // □□ Zero mean NCC
226     double numerator = 0, demoniator1 = 0, demoniator2 = 0;
227     for (int i = 0; i < values_ref.size(); i++) {
228         double n = (values_ref[i] - mean_ref) * (values_curr[i] - mean_curr);
229         numerator += n;
230         demoniator1 += (values_ref[i] - mean_ref) * (values_ref[i] - mean_ref);
231         demoniator2 += (values_curr[i] - mean_curr) * (values_curr[i] - mean_curr);
232     }
233     return numerator / sqrt(demoniator1 * demoniator2 + 1e-10);   // □□□□□□□□
234 }
235
236 bool updateDepthFilter(
237     const Vector2d &pt_ref, const Vector2d &pt_curr, const SE3d &T_C_R,
238     const Vector2d &epipolar_direction, Mat &depth, Mat &depth_cov2) {
239     // □□□□□□□□□□□□
240     // □□□□□□□□□
241     SE3d T_R_C = T_C_R.inverse();
242     Vector3d f_ref = px2cam(pt_ref);
243     f_ref.normalize();
244     Vector3d f_curr = px2cam(pt_curr);
245     f_curr.normalize();
246
247     // □□
248     // d_ref * f_ref = d_cur * ( R_RC * f_cur ) + t_RC
249     // f2 = R_RC * f_cur
250     // □□□□□□□□□□□□□
251     // => [ f_ref^T f_ref, -f_ref^T f2 ] [d_ref]   [f_ref^T t]
252     //    [ f_cur^T f_ref, -f2^T f2    ] [d_cur] = [f2^T t   ]
253     Vector3d t = T_R_C.translation();
254     Vector3d f2 = T_R_C.so3() * f_curr;
255     Vector2d b = Vector2d(t.dot(f_ref), t.dot(f2));
256     Matrix2d A;
257     A(0, 0) = f_ref.dot(f_ref);
258     A(0, 1) = -f_ref.dot(f2);
259     A(1, 0) = -A(0, 1);
260     A(1, 1) = -f2.dot(f2);
261     Vector2d ans = A.inverse() * b;
262     Vector3d xm = ans[0] * f_ref;           // ref □□□□
263     Vector3d xn = t + ans[1] * f2;          // cur □□
264     Vector3d p_esti = (xm + xn) / 2.0;      // □□□□□□□□□□□P
265     double depth_estimation = p_esti.norm();   // □□□
266
267     // □□□□□□□□□□□□□□□□□□□
268     Vector3d p = f_ref * depth_estimation;
269     Vector3d a = p - t;
270     double t_norm = t.norm();
271     double a_norm = a.norm();
272     double alpha = acos(f_ref.dot(t) / t_norm);
273     double beta = acos(-a.dot(t) / (a_norm * t_norm));
274     Vector3d f_curr_prime = px2cam(pt_curr + epipolar_direction);
275     f_curr_prime.normalize();
276     double beta_prime = acos(f_curr_prime.dot(-t) / t_norm);
277     double gamma = M_PI - alpha - beta_prime;
278     double p_prime = t_norm * sin(beta_prime) / sin(gamma);
279     double d_cov = p_prime - depth_estimation;
280     double d_cov2 = d_cov * d_cov;
281
282     // □□□□
283     double mu = depth.ptr<double>(int(pt_ref(1, 0)))[int(pt_ref(0, 0))];
284     double sigma2 = depth_cov2.ptr<double>(int(pt_ref(1, 0)))[int(pt_ref(0, 0))];
285
286     double mu_fuse = (d_cov2 * mu + sigma2 * depth_estimation) / (sigma2 + d_cov2);
287     double sigma_fuse2 = (sigma2 * d_cov2) / (sigma2 + d_cov2);
288
289     depth.ptr<double>(int(pt_ref(1, 0)))[int(pt_ref(0, 0))] = mu_fuse;
290     depth_cov2.ptr<double>(int(pt_ref(1, 0)))[int(pt_ref(0, 0))] = sigma_fuse2;
291
292     return true;
293 }
```

1. main update

2. update

3. $\pm 3\sigma$ $\sqrt{2}/2$ 0.7 NCC NC

4. NCC $\boldsymbol{A}, \boldsymbol{B}$

$$\mathrm{NCC}_z(\boldsymbol{A}, \boldsymbol{B}) = \frac{\sum_{i,j} \left(\boldsymbol{A}(i,j) - \bar{\boldsymbol{A}}(i,j)\right)\left(\boldsymbol{B}(i,j) - \bar{\boldsymbol{B}}(i,j)\right)}{\sqrt{\sum_{i,j} \left(\boldsymbol{A}(i,j) - \bar{\boldsymbol{A}}(i,j)\right)^2 \sum_{i,j} \left(\boldsymbol{B}(i,j) - \bar{\boldsymbol{B}}(i,j)\right)^2}}. \quad (12.11)$$

5. 7.5

*

```
$ build/dense_mapping ~/dataset/test_data
read total 202 files.
*** loop 1 ***
*** loop 2 ***
......
```

0.4 —— 1.0 2.5 —

— 3.0

Figure 12-6

12.3.1

trick ——

——

12.3.2

NCC

—

NCC

—

—

[75]

Figure 12-7

12.3.3

Parameterization

*

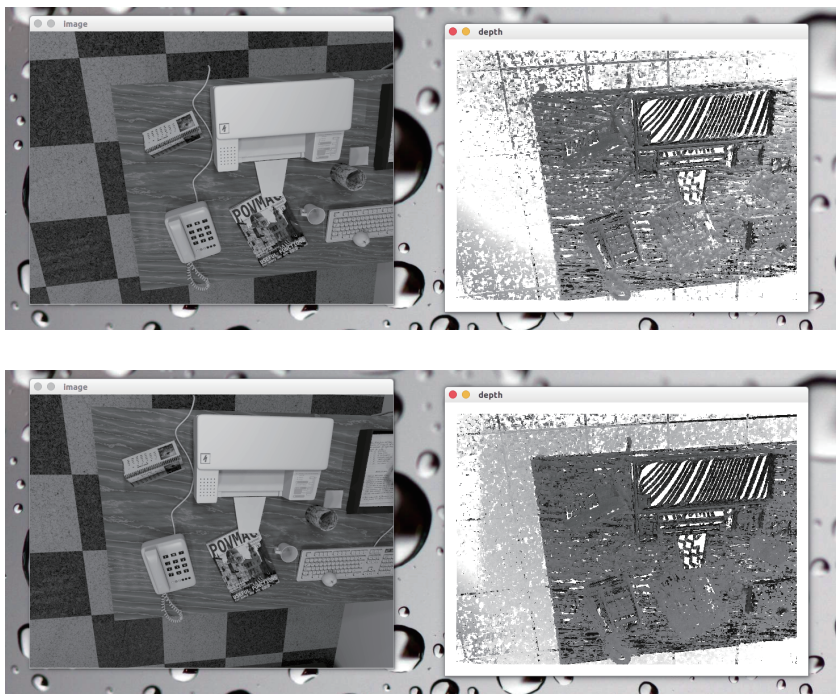


Figure 12-6: 10 30

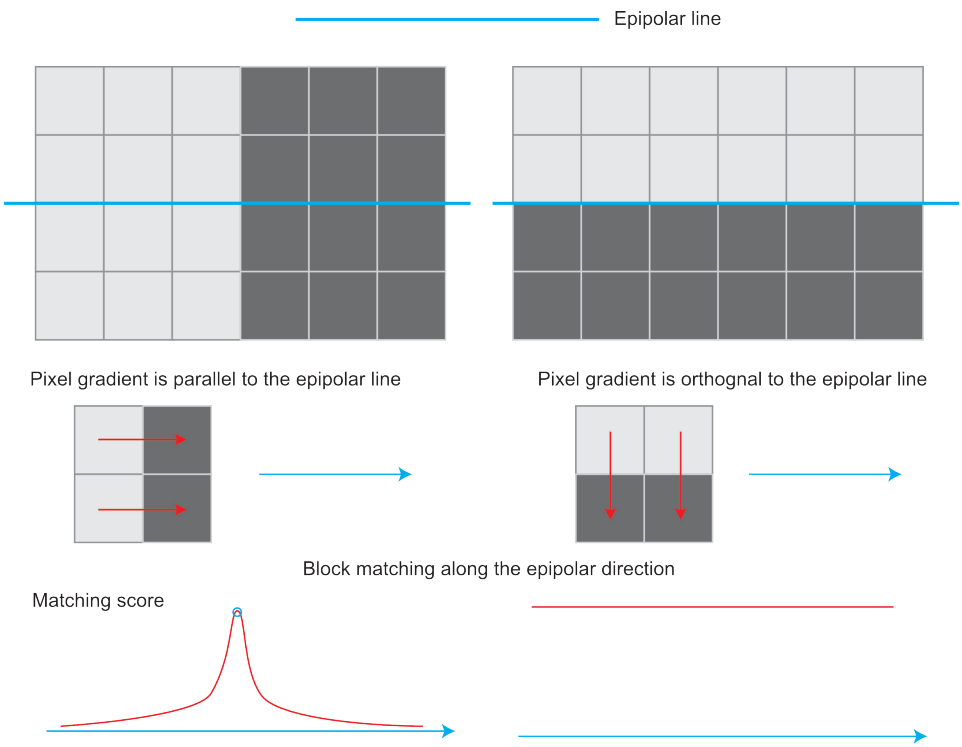

Epipolar line
Pixel gradient is parallel to the epipolar line
Pixel gradient is orthognal to the epipolar line
Block matching along the epipolar direction
Matching score

Figure 12-7:

x, y, z x, y, z u, v d u, v d
u, v * d x, y, z x, y, z

—

Inverse depth SLAM [126, 127] $d \sim N(\mu, \sigma^2)$

1. 5~10 0
2.

[127] SLAM [68, 69, 88]
d d^{-1}

12.3.4

$\boldsymbol{P}_R$ $\boldsymbol{P}_W$

$$d_R \boldsymbol{P}_R = \boldsymbol{K} \left(\boldsymbol{R}_{\mathrm{RW}} \boldsymbol{P}_W + \boldsymbol{t}_{\mathrm{RW}} \right). \tag{12.12}$$

$\boldsymbol{P}_W$ $\boldsymbol{P}_C$

$$d_C \boldsymbol{P}_C = \boldsymbol{K} \left(\boldsymbol{R}_{\mathrm{CW}} \boldsymbol{P}_W + \boldsymbol{t}_{\mathrm{CW}} \right). \tag{12.13}$$

$\boldsymbol{P}_W$

$$d_C \boldsymbol{P}_C = d_R \boldsymbol{K} \boldsymbol{R}_{\mathrm{CW}} \boldsymbol{R}_{\mathrm{RW}}^{\mathrm{T}} \boldsymbol{K}^{-1} \boldsymbol{P}_R + \boldsymbol{K} \boldsymbol{t}_{\mathrm{CW}} - \boldsymbol{K} \boldsymbol{R}_{\mathrm{CW}} \boldsymbol{R}_{\mathrm{RW}}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{t}_{\mathrm{RW}}. \tag{12.14}$$

$d_R, \boldsymbol{P}_R$ $\boldsymbol{P}_C$ $\boldsymbol{P}_R$ $\mathrm{d}u, \mathrm{d}v$ $\boldsymbol{P}_C$ $\mathrm{d}u_c, \mathrm{d}v_c$

$$\begin{bmatrix} \mathrm{d}u_c \\ \mathrm{d}v_c \end{bmatrix} = \begin{bmatrix} \frac{\mathrm{d}u_c}{\mathrm{d}u} & \frac{\mathrm{d}u_c}{\mathrm{d}v} \\ \frac{\mathrm{d}v_c}{\mathrm{d}u} & \frac{\mathrm{d}v_c}{\mathrm{d}v} \end{bmatrix} \begin{bmatrix} \mathrm{d}u \\ \mathrm{d}v \end{bmatrix} \tag{12.15}$$

12.3.5

CPU
CPU
GPU GPU GPU GPU GPU

12.3.6

1.
2. Outlier NCC
[124] —

* u, v

12.4 RGB-D

RGB-D RGB-D RGB-D
D
RGB-D RGB-D Point Cloud Poin
Cloud Map Mesh Surfel Voxel Occupancy
Map
RGB-D GPU

12.4.1

x, y, z r, g, b RGB-D RGB-D
grid filter

Listing 12.2: slambook/ch12/dense_RGBD/pointcloud_mapping.cpp

```
int main(int argc, char **argv) {
    vector<cv::Mat> colorImgs, depthImgs;    // □□□□□□□
    vector<Eigen::Isometry3d> poses;         // □□□□
    ifstream fin("./data/pose.txt");
    if (!fin) {
        cerr << "cannot find pose file" << endl;
        return 1;
    }
    for (int i = 0; i < 5; i++) {
        boost::format fmt("./data/%s/%d.%s"); //□□□□□□
        colorImgs.push_back(cv::imread((fmt % "color" % (i + 1) % "png").str()));
        depthImgs.push_back(cv::imread((fmt % "depth" % (i + 1) % "png").str(), -1));
            // □□□□□□□□-1
        double data[7] = {0};
        for (int i = 0; i < 7; i++) {
            fin >> data[i];
        }
        Eigen::Quaterniond q(data[6], data[3], data[4], data[5]);
        Eigen::Isometry3d T(q);
        T.pretranslate(Eigen::Vector3d(data[0], data[1], data[2]));
        poses.push_back(T);
    }
    // □□□□□□□
    // □□□□
    double cx = 319.5;
    double cy = 239.5;
    double fx = 481.2;
    double fy = -480.0;
    double depthScale = 5000.0;
    cout << "□□□□□□□□□□□..." << endl;
    // □□□□□□□□□□□□□□□□XYZRGB
    typedef pcl::PointXYZRGB PointT;
    typedef pcl::PointCloud<PointT> PointCloud;
    // □□□□□□
    PointCloud::Ptr pointCloud(new PointCloud);
    for (int i = 0; i < 5; i++) {
        PointCloud::Ptr current(new PointCloud);
        cout << "□□□□□: " << i + 1 << endl;
        cv::Mat color = colorImgs[i];
        cv::Mat depth = depthImgs[i];
        Eigen::Isometry3d T = poses[i];
        for (int v = 0; v < color.rows; v++)
        for (int u = 0; u < color.cols; u++) {
```

```
            unsigned int d = depth.ptr<unsigned short>(v)[u]; // □□□
            if (d == 0) continue; // □□□□□□□□0
            Eigen::Vector3d point;
            point[2] = double(d) / depthScale;
            point[0] = (u − cx) * point[2] / fx;
            point[1] = (v − cy) * point[2] / fy;
            Eigen::Vector3d pointWorld = T * point;

            PointT p;
            p.x = pointWorld[0];
            p.y = pointWorld[1];
            p.z = pointWorld[2];
            p.b = color.data[v * color.step + u * color.channels()];
            p.g = color.data[v * color.step + u * color.channels() + 1];
            p.r = color.data[v * color.step + u * color.channels() + 2];
            current->points.push_back(p);
        }
        // depth filter and statistical removal
        PointCloud::Ptr tmp(new PointCloud);
        pcl::StatisticalOutlierRemoval<PointT> statistical_filter;
        statistical_filter.setMeanK(50);
        statistical_filter.setStddevMulThresh(1.0);
        statistical_filter.setInputCloud(current);
        statistical_filter.filter(*tmp);
        (*pointCloud) += *tmp;
    }

    pointCloud->is_dense = false;
    cout << "□□□□" << pointCloud->size() << "□□." << endl;

    // voxel filter
    pcl::VoxelGrid<PointT> voxel_filter;
    double resolution = 0.03;
    voxel_filter.setLeafSize(resolution, resolution, resolution);       // resolution
    PointCloud::Ptr tmp(new PointCloud);
    voxel_filter.setInputCloud(pointCloud);
    voxel_filter.filter(*tmp);
    tmp->swap(*pointCloud);

    cout << "□□□□□□□□□" << pointCloud->size() << "□□." << endl;

    pcl::io::savePCDFileBinary("map.pcd", *pointCloud);
    return 0;
}
```

pcl Ubuntu 18.04

Listing 12.3:

```
sudo apt-get install libpcl-dev pcl-tools
```

pcl

1. Kinect
2. N “ ”
3.

ICL-NUIM RGB-D data/ 0.03 0.03
dense_RGBD

Listing 12.4:

```
./build/pointcloud_mapping
```

map.pcd pcl_viewer pcd Figure 12-8
RGB-D

Figure 12-8: ICL-NUIM

1. ICP ICP
2. “ ” “ ”
3.

“ ” “ ” Occupan

Grid SfM [128] Surfel [129]

Figure 12-9 Surfel PCL PCL

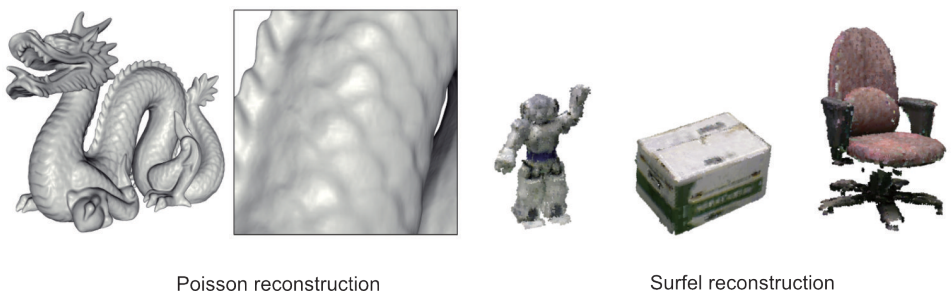


Figure 12-9: Surfel

12.4.2

Listing 12.5: slambook2/ch12/dense_RGBD/surfel_mapping.cpp

```
#include <pcl/point_cloud.h>
#include <pcl/point_types.h>
#include <pcl/io/pcd_io.h>
#include <pcl/visualization/pcl_visualizer.h>
#include <pcl/kdtree/kdtree_flann.h>
#include <pcl/surface/surfel_smoothing.h>
```

```
#include <pcl/surface/mls.h>
#include <pcl/surface/gp3.h>
#include <pcl/surface/impl/mls.hpp>

// typedefs
typedef pcl::PointXYZRGB PointT;
typedef pcl::PointCloud<PointT> PointCloud;
typedef pcl::PointCloud<PointT>::Ptr PointCloudPtr;
typedef pcl::PointXYZRGBNormal SurfelT;
typedef pcl::PointCloud<SurfelT> SurfelCloud;
typedef pcl::PointCloud<SurfelT>::Ptr SurfelCloudPtr;

SurfelCloudPtr reconstructSurface(
const PointCloudPtr &input, float radius, int polynomial_order) {
    pcl::MovingLeastSquares<PointT, SurfelT> mls;
    pcl::search::KdTree<PointT>::Ptr tree(new pcl::search::KdTree<PointT>);
    mls.setSearchMethod(tree);
    mls.setSearchRadius(radius);
    mls.setComputeNormals(true);
    mls.setSqrGaussParam(radius * radius);
    mls.setPolynomialFit(polynomial_order > 1);
    mls.setPolynomialOrder(polynomial_order);
    mls.setInputCloud(input);
    SurfelCloudPtr output(new SurfelCloud);
    mls.process(*output);
    return (output);
}

pcl::PolygonMeshPtr triangulateMesh(const SurfelCloudPtr &surfels) {
    // Create search tree*
    pcl::search::KdTree<SurfelT>::Ptr tree(new pcl::search::KdTree<SurfelT>);
    tree->setInputCloud(surfels);

    // Initialize objects
    pcl::GreedyProjectionTriangulation<SurfelT> gp3;
    pcl::PolygonMeshPtr triangles(new pcl::PolygonMesh);

    // Set the maximum distance between connected points (maximum edge length)
    gp3.setSearchRadius(0.05);

    // Set typical values for the parameters
    gp3.setMu(2.5);
    gp3.setMaximumNearestNeighbors(100);
    gp3.setMaximumSurfaceAngle(M_PI / 4); // 45 degrees
    gp3.setMinimumAngle(M_PI / 18); // 10 degrees
    gp3.setMaximumAngle(2 * M_PI / 3); // 120 degrees
    gp3.setNormalConsistency(true);

    // Get result
    gp3.setInputCloud(surfels);
    gp3.setSearchMethod(tree);
    gp3.reconstruct(*triangles);

    return triangles;
}

int main(int argc, char **argv) {
    // Load the points
    PointCloudPtr cloud(new PointCloud);
    if (argc == 0 || pcl::io::loadPCDFile(argv[1], *cloud)) {
        cout << "failed to load point cloud!";
        return 1;
    }
    cout << "point cloud loaded, points: " << cloud->points.size() << endl;

    // Compute surface elements
    cout << "computing normals ... " << endl;
    double mls_radius = 0.05, polynomial_order = 2;
    auto surfels = reconstructSurface(cloud, mls_radius, polynomial_order);

    // Compute a greedy surface triangulation
    cout << "computing mesh ... " << endl;
    pcl::PolygonMeshPtr mesh = triangulateMesh(surfels);

```

```
81      cout << "display mesh ... " << endl;
82      pcl::visualization::PCLVisualizer vis;
83      vis.addPolylineFromPolygonMesh(*mesh, "mesh frame");
84      vis.addPolygonMesh(*mesh, "mesh");
85      vis.resetCamera();
86      vis.spin();
87  }
```

Listing 12.6:

```
1  ./build/surfel_mapping map.pcd
```

Figure 12-10 Moving Least Square - Greedy Projection [?] [?]

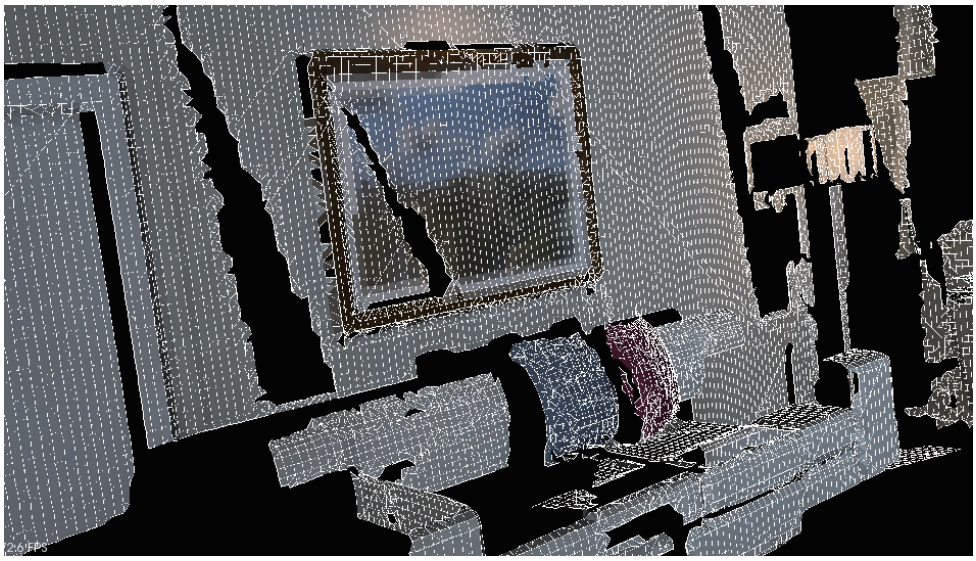


Figure 12-10:

12.4.3

- pcd 640 ×480 30 pcd “ ”
- “ ” “ ”

Octo-map [130]

“ ” “ tree

Figure 12-11 “ ” cm^3 10 8^{10}cm$^3 = 1,073$m^3

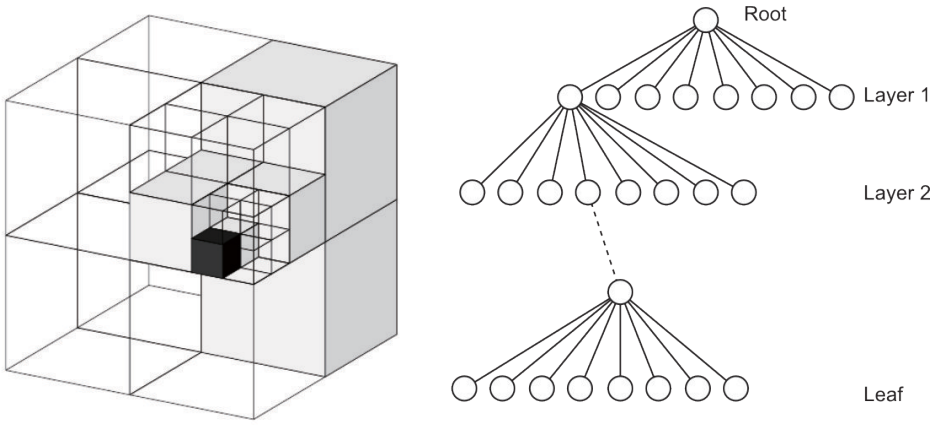


Figure 12-11:

0 1 0−1 0 1 0 1 “ ”“ $[0,1]$ x 0.5

x $[0,1]$ Log-odds $y \in \mathbb{R}$ x 0~1 logit

$$y = \text{logit}(x) = \log\left(\frac{x}{1-x}\right). \tag{12.16}$$

$$x = \text{logit}^{-1}(y) = \frac{\exp(y)}{\exp(y)+1}. \tag{12.17}$$

y $-\infty$ $+\infty$ x 0 1 y 0 x 0.5 y “ ” y y logit y 1

$$L(n|z_{1:t+1}) = L(n|z_{1:t-1}) + L(n|z_t). \tag{12.18}$$

$$P(n|z_{1:T}) = \left[1 + \frac{1-P(n|z_T)}{P(n|z_T)}\frac{1-P(n|z_{1:T-1})}{P(n|z_{1:T-1})}\frac{P(n)}{1-P(n)}\right]^{-1}. \tag{12.19}$$

RGB-D RGB-D d

12.4.4

octomap octomap 18.04 octomap octovis

Listing 12.7:

```
sudo apt-get install liboctomap-dev octovis
```

5

Listing 12.8: slambook/ch13/dense_RGBD/octomap_mapping.cpp

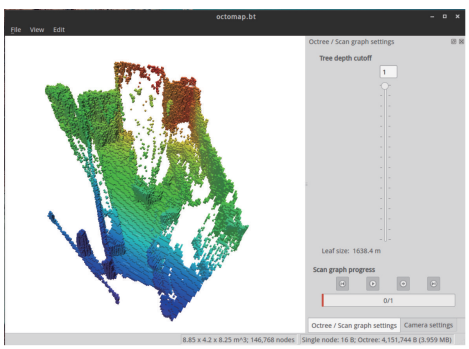
```
// octomap tree
octomap::OcTree tree(0.01); // 
```

```
for (int i = 0; i < 5; i++) {
    cout << "□□□□□: " << i + 1 << endl;
    cv::Mat color = colorImgs[i];
    cv::Mat depth = depthImgs[i];
    Eigen::Isometry3d T = poses[i];

    octomap::Pointcloud cloud;  // the point cloud in octomap

    for (int v = 0; v < color.rows; v++)
    for (int u = 0; u < color.cols; u++) {
        unsigned int d = depth.ptr<unsigned short>(v)[u]; // □□□
        if (d == 0) continue; // □□□□□□□□0
        Eigen::Vector3d point;
        point[2] = double(d) / depthScale;
        point[0] = (u - cx) * point[2] / fx;
        point[1] = (v - cy) * point[2] / fy;
        Eigen::Vector3d pointWorld = T * point;
        // □□□□□□□□□□□□
        cloud.push_back(pointWorld[0], pointWorld[1], pointWorld[2]);
    }

    // □□□□□□□□□□□□□□□□□□□□□□□□□□□□
    tree.insertPointCloud(cloud, octomap::point3d(T(0, 3), T(1, 3), T(2, 3)));
}

// □□□□□□□□□□□□□□□□□□
tree.updateInnerOccupancy();
cout << "saving octomap ... " << endl;
tree.writeBinary("octomap.bt");
```

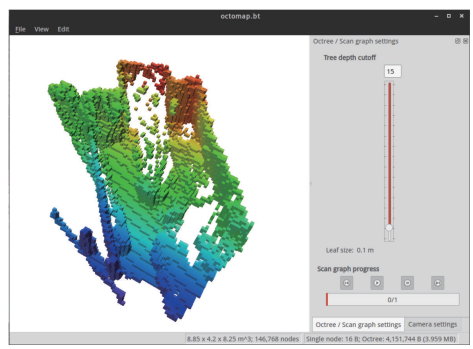
octomap::OcTree octomap

Ocotmap PCL RGB-D octomap octomap

Figure 12-12 “1” octovis



octomap (resolution=0.05)



octomap (resolution=0.1)

Figure 12-12:

16 16 0.05 0.1

Octomap [131] 6.9MB octomap 56KB

12.5 * TSDF Fusion

SLAM GPU

SLAM

__ “ ” __

“ ” GPU GPU

RGB-D RGB-D Kinect Fusion[132] Dynamic Fusion[133] Elastic Fusion[134] Fusion4D[135] Volumn Deform[136] Kinect Fusion 13

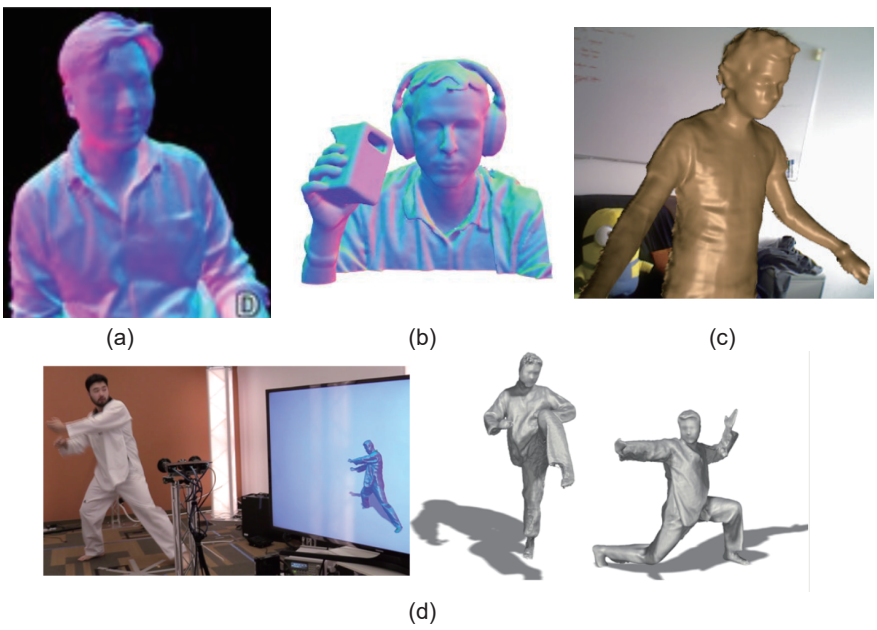


Figure 12-13: a Kinect Fusion b Dynamic Fusion c Volumn Deform d Fusion4D e Elastic Fusion

TSDF TSDF Truncated Signed Distance Function “ ” “ ” TSDF Figure 12-14 $3\times3\times3m^3$ TSDF TSDF 1 −1 TSDF — TSDF Figure 12-14 TSDF TSDF “ ” “ ” SLAM RGB-D GPU TSDF T

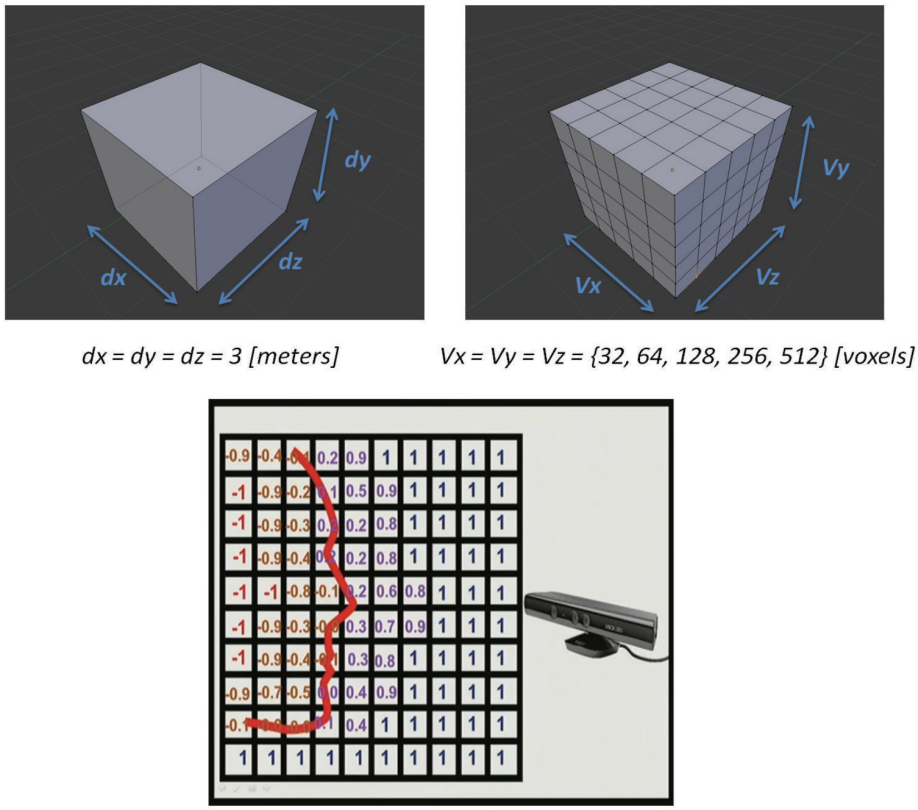


Figure 12-14: TSDF

D
TSDF ICP GPU TSDF ICP TSDF
D * TSDF GPU

12.6

RGB-D SLAM

1. (12.6)
2.
3.*

*

4.

5. [132] TSDF

6.* —

Chapter 13

SLAM

1.
2. SLAM
3. VO

BA

13.1

Figure

1 *

Figure 13-1:

SLAM

SLAM Trick SLAM “ ”“

* Epicwork Epicwork

SLAM SLAM “
slambook2/ch13 VO Kitti VO BA VO

13.2

Linux

1. bin
2. include/myslam SLAM .h include include "myslam/xxx.h"
3. src .cpp
4. test .cpp
5. config
6. cmake_modules cmake g2o

VO

13.2.1

\+ VO VO

-
- 2D
- 3D

Figure 13-2 3D

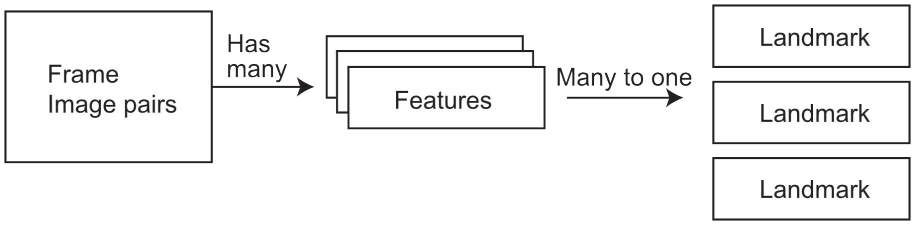


Figure 13-2:

SLAM

-
-

Figure 13-3

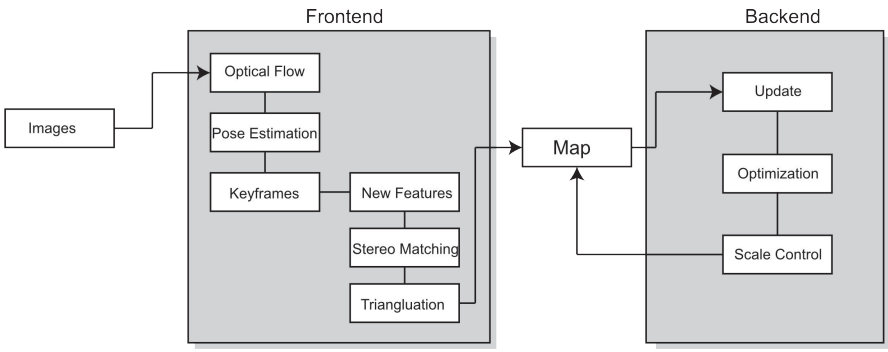


Figure 13-3:

-
-
- Kitti Kitti
-

13.3

13.3.1

struct

Frame

Listing 13.1: slambook2/ch13/include/myslam/frame.h

```
struct Frame {
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW;
    typedef std::shared_ptr<Frame> Ptr;

    unsigned long id_ = 0;           // id of this frame
    unsigned long keyframe_id_ = 0;  // id of key frame
    bool is_keyframe_ = false;       // □□□□□□
    double time_stamp_;              // □□□□□□□□
    SE3 pose_;                       // Tcw □□Pose
    std::mutex pose_mutex_;          // □□□Pose
    cv::Mat left_img_, right_img_;   // stereo images

    // extracted features in left image
    std::vector<std::shared_ptr<Feature>> features_left_;
    // corresponding features in right image, set to nullptr if no corresponding
    std::vector<std::shared_ptr<Feature>> features_right_;

public:  // data members
    Frame() {}

    Frame(long id, double time_stamp, const SE3 &pose, const Mat &left,
    const Mat &right);

    // set and get pose, thread safe
```

```
    SE3 Pose() {
        std::unique_lock<std::mutex> lck(pose_mutex_);
        return pose_;
    }

    void SetPose(const SE3 &pose) {
        std::unique_lock<std::mutex> lck(pose_mutex_);
        pose_ = pose;
    }

    /// □□□□□□□□□□□□id
    void SetKeyFrame();

    /// □□□□□□□□□□id
    static std::shared_ptr<Frame> CreateFrame();
};
```

Frame id Pose Pose Set Get Frame id Feature

Listing 13.2: slambook2/ch13/include/myslam/feature.h

```
struct Feature {
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW;
    typedef std::shared_ptr<Feature> Ptr;

    std::weak_ptr<Frame> frame_;         // □□□□featureframe
    cv::KeyPoint position_;              // 2□□□□D
    std::weak_ptr<MapPoint> map_point_;  // □□□□□

    bool is_outlier_ = false;       // □□□□□□
    bool is_on_left_image_ = true;  // □□□□□□□□□□□□□false

public:
    Feature() {}

    Feature(std::shared_ptr<Frame> frame, const cv::KeyPoint &kp)
    : frame_(frame), position_(kp) {}
};
```

Feature 2D Feature Frame Frame MapPoint sh ptr * weak_ptr MapPoint

Listing 13.3: slambook2/ch13/include/myslam/mappoint.h

```
struct MapPoint {
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW;
    typedef std::shared_ptr<MapPoint> Ptr;
    unsigned long id_ = 0;  // ID
    bool is_outlier_ = false;
    Vec3 pos_ = Vec3::Zero();  // Position in world
    std::mutex data_mutex_;
    int observed_times_ = 0;  // being observed by feature matching algo.
    std::list<std::weak_ptr<Feature>> observations_;

    MapPoint() {}

    MapPoint(long id, Vec3 position);

    Vec3 Pos() {
        std::unique_lock<std::mutex> lck(data_mutex_);
        return pos_;
    }

```

* Frame Feature shared_ptr Feature Frame shared_ptr

```
    void SetPos(const Vec3 &pos) {
        std::unique_lock<std::mutex> lck(data_mutex_);
        pos_ = pos;
    };

    void AddObservation(std::shared_ptr<Feature> feature) {
        std::unique_lock<std::mutex> lck(data_mutex_);
        observations_.push_back(feature);
        observed_times_++;
    }

    void RemoveObservation(std::shared_ptr<Feature> feat);

    std::list<std::weak_ptr<Feature>> GetObs() {
        std::unique_lock<std::mutex> lck(data_mutex_);
        return observations_;
    }

    // factory function
    static MapPoint::Ptr CreateNewMappoint();
};
```

MapPoint 3D pos_ observation_ Feature Feature outlier observation Frame MapPoint

Listing 13.4: slambook2/ch13/include/myslam/map.h

```
class Map {
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW;
    typedef std::shared_ptr<Map> Ptr;
    typedef std::unordered_map<unsigned long, MapPoint::Ptr> LandmarksType;
    typedef std::unordered_map<unsigned long, Frame::Ptr> KeyframesType;

    Map() {}

    /// □□□□□□□□
    void InsertKeyFrame(Frame::Ptr frame);
    /// □□□□□□□□□
    void InsertMapPoint(MapPoint::Ptr map_point);

    /// □□□□□□□□
    LandmarksType GetAllMapPoints() {
        std::unique_lock<std::mutex> lck(data_mutex_);
        return landmarks_;
    }
    /// □□□□□□□□
    KeyframesType GetAllKeyFrames() {
        std::unique_lock<std::mutex> lck(data_mutex_);
        return keyframes_;
    }

    /// □□□□□□□□
    LandmarksType GetActiveMapPoints() {
        std::unique_lock<std::mutex> lck(data_mutex_);
        return active_landmarks_;
    }

    /// □□□□□□□□
    KeyframesType GetActiveKeyFrames() {
        std::unique_lock<std::mutex> lck(data_mutex_);
        return active_keyframes_;
    }

    /// □□□□□□□□□□□□□map
    void CleanMap();

private:
    // □□□□□□□□□□□□□□□
    void RemoveOldKeyframe();

    std::mutex data_mutex_;
```

```
    LandmarksType landmarks_;         // all landmarks
    LandmarksType active_landmarks_;  // active landmarks
    KeyframesType keyframes_;         // all key-frames
    KeyframesType active_keyframes_;  // all key-frames

    Frame::Ptr current_frame_ = nullptr;

    // settings
    int num_active_keyframes_ = 7;  // □□□□□□□□
};
```

13.3.2

1.
2.
3.
4.
   -
   -
   -
5.

Listing 13.5: slambook2/ch13/src/frontend.cpp

```
bool Frontend::AddFrame(myslam::Frame::Ptr frame) {
    current_frame_ = frame;
    switch (status_) {
        case FrontendStatus::INITING:
            StereoInit();
            break;
        case FrontendStatus::TRACKING_GOOD:
        case FrontendStatus::TRACKING_BAD:
            Track();
            break;
        case FrontendStatus::LOST:
            Reset();
            break;
    }

    last_frame_ = current_frame_;
    return true;
}
```

Track

Listing 13.6: slambook2/ch13/src/frontend.cpp

```
bool Frontend::Track() {
    if (last_frame_) {
        current_frame_->SetPose(relative_motion_ * last_frame_->Pose());
    }

    int num_track_last = TrackLastFrame();
```

```
    tracking_inliers_ = EstimateCurrentPose();

    if (tracking_inliers_ > num_features_tracking_) {
        // tracking good
        status_ = FrontendStatus::TRACKING_GOOD;
    } else if (tracking_inliers_ > num_features_tracking_bad_) {
        // tracking bad
        status_ = FrontendStatus::TRACKING_BAD;
    } else {
        // lost
        status_ = FrontendStatus::LOST;
    }

    InsertKeyframe();
    relative_motion_ = current_frame_->Pose() * last_frame_->Pose().inverse();

    if (viewer_) viewer_->AddCurrentFrame(current_frame_);
    return true;
}
```

TrackLastFrame OpenCV

Listing 13.7: slambook2/ch13/src/frontend.cpp

```
int Frontend::TrackLastFrame() {
    // use LK flow to estimate points in the right image
    std::vector<cv::Point2f> kps_last, kps_current;
    for (auto &kp : last_frame_->features_left_) {
        if (kp->map_point_.lock()) {
            // use project point
            auto mp = kp->map_point_.lock();
            auto px =
                camera_left_->world2pixel(mp->pos_, current_frame_->Pose());
                kps_last.push_back(kp->position_.pt);
            kps_current.push_back(cv::Point2f(px[0], px[1]));
        } else {
            kps_last.push_back(kp->position_.pt);
            kps_current.push_back(kp->position_.pt);
        }
    }

    std::vector<uchar> status;
    Mat error;
    cv::calcOpticalFlowPyrLK(
        last_frame_->left_img_, current_frame_->left_img_, kps_last,
        kps_current, status, error, cv::Size(21, 21), 3,
        cv::TermCriteria(cv::TermCriteria::COUNT + cv::TermCriteria::EPS, 30, 0.01),
        cv::OPTFLOW_USE_INITIAL_FLOW);

    int num_good_pts = 0;

    for (size_t i = 0; i < status.size(); ++i) {
        if (status[i]) {
            cv::KeyPoint kp(kps_current[i], 7);
            Feature::Ptr feature(new Feature(current_frame_, kp));
            feature->map_point_ = last_frame_->features_left_[i]->map_point_;
            current_frame_->features_left_.push_back(feature);
            num_good_pts++;
        }
    }

    LOG(INFO) << "Find " << num_good_pts << " in the last image.";
    return num_good_pts;
}
```

OpenCV g2o 400

13.3.3

Listing 13.8: slambook2/ch13/include/myslam/backend.h

```
class Backend {
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW;
    typedef std::shared_ptr<Backend> Ptr;

    /// □□□□□□□□□□□□□□□□
    Backend();

    // □□□□□□□□□□□□□□□□□□□
    void SetCameras(Camera::Ptr left, Camera::Ptr right) {
        cam_left_ = left;
        cam_right_ = right;
    }

    /// □□□□
    void SetMap(std::shared_ptr<Map> map) { map_ = map; }

    /// □□□□□□□□□□□□
    void UpdateMap();

    /// □□□□□□□
    void Stop();

private:
    /// □□□□
    void BackendLoop();

    /// □□□□□□□□□□□□□□□□
    void Optimize(Map::KeyframesType& keyframes, Map::LandmarksType& landmarks);

    std::shared_ptr<Map> map_;
    std::thread backend_thread_;
    std::mutex data_mutex_;

    std::condition_variable map_update_;
    std::atomic<bool> backend_running_;

    Camera::Ptr cam_left_ = nullptr, cam_right_ = nullptr;
};
```

map_update_

Listing 13.9: slambook2/ch13/src/backend.cpp

```
void Backend::BackendLoop() {
    while (backend_running_.load()) {
        std::unique_lock<std::mutex> lock(data_mutex_);
        map_update_.wait(lock);

        /// □□□□□□□□□FramesLandmarks
        Map::KeyframesType active_kfs = map_->GetActiveKeyFrames();
        Map::LandmarksType active_landmarks = map_->GetActiveMapPoints();
        Optimize(active_kfs, active_landmarks);
    }
}
```

BA Frame MapPoint

Listing 13.10: slambook2/ch13/src/backend.cpp

```
void Backend::Optimize(Map::KeyframesType &keyframes,
    Map::LandmarksType &landmarks) {
    // setup g2o
    typedef g2o::BlockSolver_6_3 BlockSolverType;
    typedef g2o::LinearSolverCSparse<BlockSolverType::PoseMatrixType>
    LinearSolverType;
    auto solver = new g2o::OptimizationAlgorithmLevenberg(
    g2o::make_unique<BlockSolverType>(
    g2o::make_unique<LinearSolverType>()));
    g2o::SparseOptimizer optimizer;
```

```
    optimizer.setAlgorithm(solver);

    // pose □□□□□Keyframe id
    std::map<unsigned long, VertexPose *> vertices;
    unsigned long max_kf_id = 0;
    for (auto &keyframe : keyframes) {
        auto kf = keyframe.second;
        VertexPose *vertex_pose = new VertexPose();  // camera vertex_pose
        vertex_pose->setId(kf->keyframe_id_);
        vertex_pose->setEstimate(kf->Pose());
        optimizer.addVertex(vertex_pose);
        if (kf->keyframe_id_ > max_kf_id) {
            max_kf_id = kf->keyframe_id_;
        }

        vertices.insert({kf->keyframe_id_, vertex_pose});
    }

    // □□□□□□□□□□□□id
    std::map<unsigned long, VertexXYZ *> vertices_landmarks;

    // K □□□□□
    Mat33 K = cam_left_->K();
    SE3 left_ext = cam_left_->pose();
    SE3 right_ext = cam_right_->pose();

    // edges
    int index = 1;
    double chi2_th = 5.991;  // robust kernel □□
    std::map<EdgeProjection *, Feature::Ptr> edges_and_features;

    for (auto &landmark : landmarks) {
        if (landmark.second->is_outlier_) continue;
        unsigned long landmark_id = landmark.second->id_;
        auto observations = landmark.second->GetObs();
        for (auto &obs : observations) {
            if (obs.lock() == nullptr) continue;
            auto feat = obs.lock();
            if (feat->is_outlier_ || feat->frame_.lock() == nullptr) continue;

            auto frame = feat->frame_.lock();
            EdgeProjection *edge = nullptr;
            if (feat->is_on_left_image_) {
                edge = new EdgeProjection(K, left_ext);
            } else {
                edge = new EdgeProjection(K, right_ext);
            }

            // □□□□□□□□□□□□□□□□□□□□□□□□landmark
            if (vertices_landmarks.find(landmark_id) ==
            vertices_landmarks.end()) {
                VertexXYZ *v = new VertexXYZ;
                v->setEstimate(landmark.second->Pos());
                v->setId(landmark_id + max_kf_id + 1);
                v->setMarginalized(true);
                vertices_landmarks.insert({landmark_id, v});
                optimizer.addVertex(v);
            }

            edge->setId(index);
            edge->setVertex(0, vertices.at(frame->keyframe_id_));    // pose
            edge->setVertex(1, vertices_landmarks.at(landmark_id));  // landmark
            edge->setMeasurement(toVec2(feat->position_.pt));
            edge->setInformation(Mat22::Identity());
            auto rk = new g2o::RobustKernelHuber();
            rk->setDelta(chi2_th);
            edge->setRobustKernel(rk);
            edges_and_features.insert({edge, feat});

            optimizer.addEdge(edge);

            index++;
        }
    }
```

```
85
86      // do optimization and eliminate the outliers
87      optimizer.initializeOptimization();
88      optimizer.optimize(10);
89
90      int cnt_outlier = 0, cnt_inlier = 0;
91      int iteration = 0;
92      while (iteration < 5) {
93          cnt_outlier = 0;
94          cnt_inlier = 0;
95          // determine if we want to adjust the outlier threshold
96          for (auto &ef : edges_and_features) {
97              if (ef.first->chi2() > chi2_th) {
98                  cnt_outlier++;
99              } else {
100                 cnt_inlier++;
101             }
102         }
103         double inlier_ratio = cnt_inlier / double(cnt_inlier + cnt_outlier);
104         if (inlier_ratio > 0.5) {
105             break;
106         } else {
107             chi2_th *= 2;
108             iteration++;
109         }
110     }
111
112     for (auto &ef : edges_and_features) {
113         if (ef.first->chi2() > chi2_th) {
114             ef.second->is_outlier_ = true;
115             // remove the observation
116             ef.second->map_point_.lock()->RemoveObservation(ef.second);
117         } else {
118             ef.second->is_outlier_ = false;
119         }
120     }
121
122     LOG(INFO) << "Outlier/Inlier in optimization: " << cnt_outlier << "/"
123     << cnt_inlier;
124
125     // Set pose and lanrmark position
126     for (auto &v : vertices) {
127         keyframes.at(v.first)->SetPose(v.second->estimate());
128     }
129     for (auto &v : vertices_landmarks) {
130         landmarks.at(v.first)->SetPos(v.second->estimate());
131     }
132 }
```

13.4

Kitti `http://www.cvlibs.net/datasets/kitti/eval_odometry.php` odometry 22GB 0 config/default.yaml

dataset_dir: /media/xiang/Data/Dataset/Kitti/dataset/sequences/00

Listing 13.11:

```
1 bin/run_kitti_stereo
```

Figure 13-4

16ms

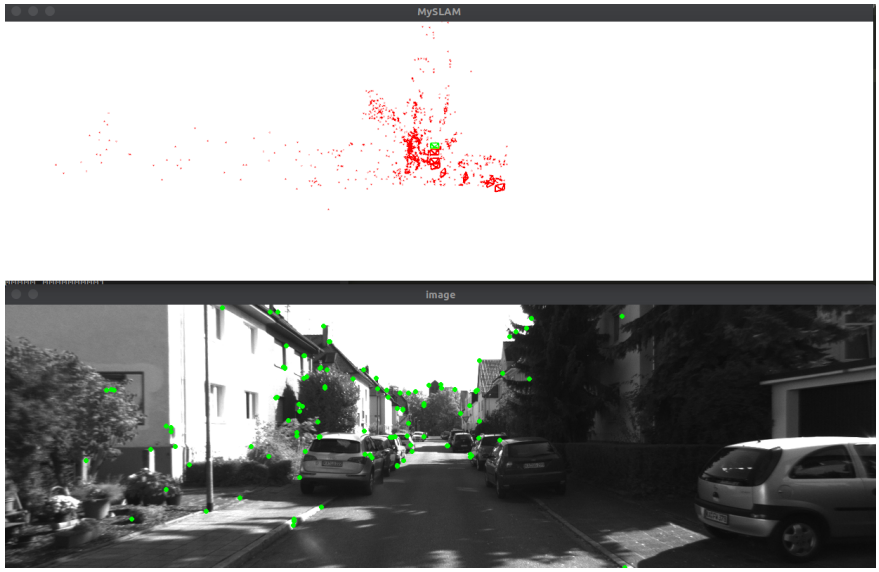


Figure 13-4:

1. C++ for
2. GFTT 1
3. * Pose Graph

Chapter 14

SLAM

1. SLAM
2. SLAM
3. SLAM

SLAM SLAM

14.1

SLAM SLAM 20% 80%
SLAM Table 14-1 SLAM

Table 14-1: SLAM

MonoSLAM		https://github.com/hanmekim/SceneLib2
PTAM		http://www.robots.ox.ac.uk/~gk/PTAM/
ORB-SLAM		http://webdiis.unizar.es/~raulmur/orbslam/
LSD-SLAM		http://vision.in.tum.de/research/vslam/lsdslam
SVO		https://github.com/uzh-rpg/rpg_svo
DTAM	RGB-D	https://github.com/anuranbaka/OpenDTAM
DVO	RGB-D	https://github.com/tum-vision/dvo_slam
DSO		https://github.com/JakobEngel/dso
VINS	+IMU	https://github.com/HKUST-Aerial-Robotics/VINS-Mono
RTAB-MAP	/RGB-D	https://github.com/introlab/rtabmap
RGBD-SLAM-V2	RGB-D	https://github.com/felixendres/rgbdslam_v2
Elastic Fusion	RGB-D	https://github.com/mp3guy/ElasticFusion
Hector SLAM		http://wiki.ros.org/hector_slam
GMapping		http://wiki.ros.org/gmapping
OKVIS	+IMU	https://github.com/ethz-asl/okvis
ROVIO	+IMU	https://github.com/ethz-asl/rovio

14.1.1 MonoSLAM

SLAM A. J. Davison SLAM [2, 55] Davison SLAM 2007 MonoSLAM SLAM
Figure 14-1 MonoSLAM EKF
SLAM SLAM M

14.1.2 PTAM

2007 Klein PTAM Parallel Tracking and Mapping [96] SLAM PTAM

1. PTAM SLAM SLAM
2. PTAM SLAM EKF PTAM
   PTAM AR Figure 14-2 PTAM

* SLAM

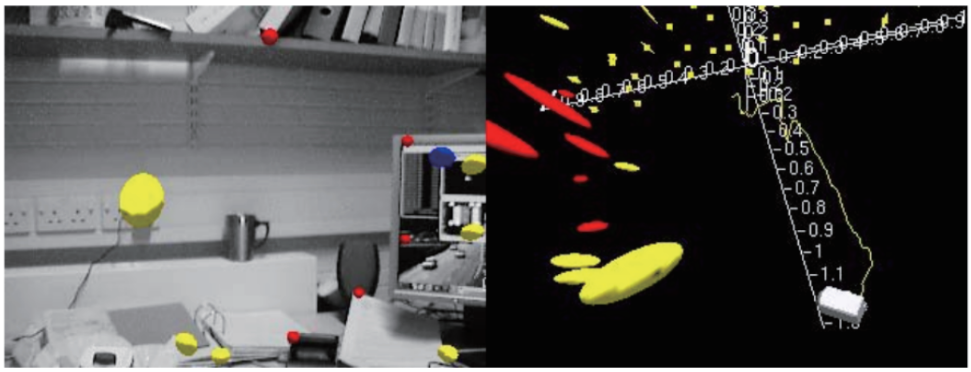


Figure 14-1: MonoSLAM

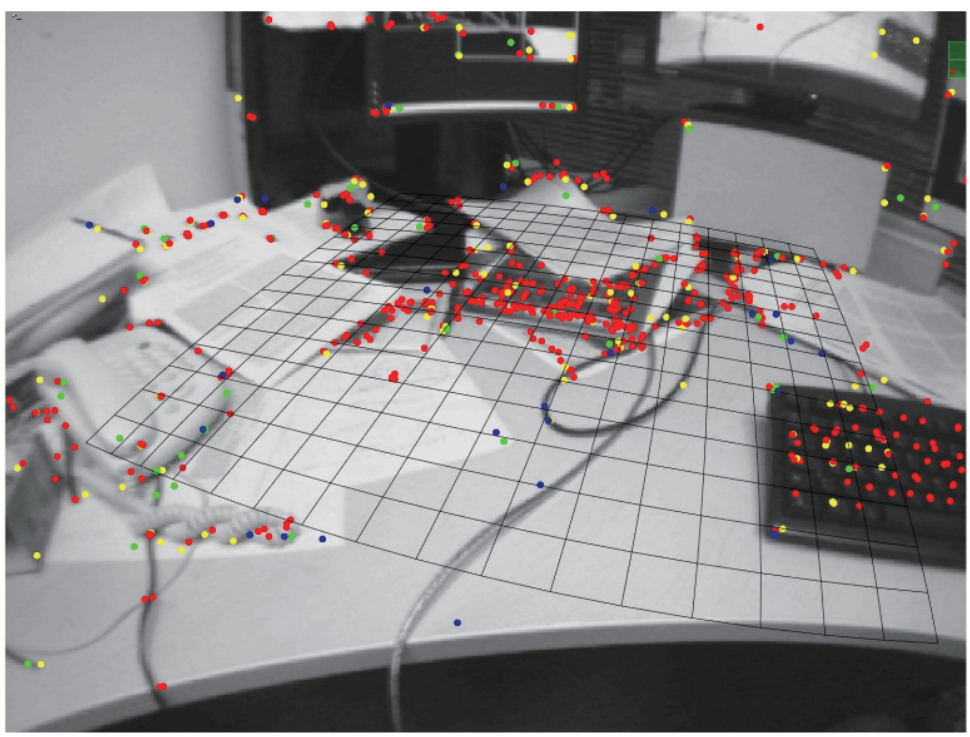


Figure 14-2: PTAM

PTAM AR SLAM

14.1.3 ORB-SLAM

SLAM ORB-SLAM PTAM [88] Figure 14-3 2015 SLAM SLAM SLAM ORB-SLAM

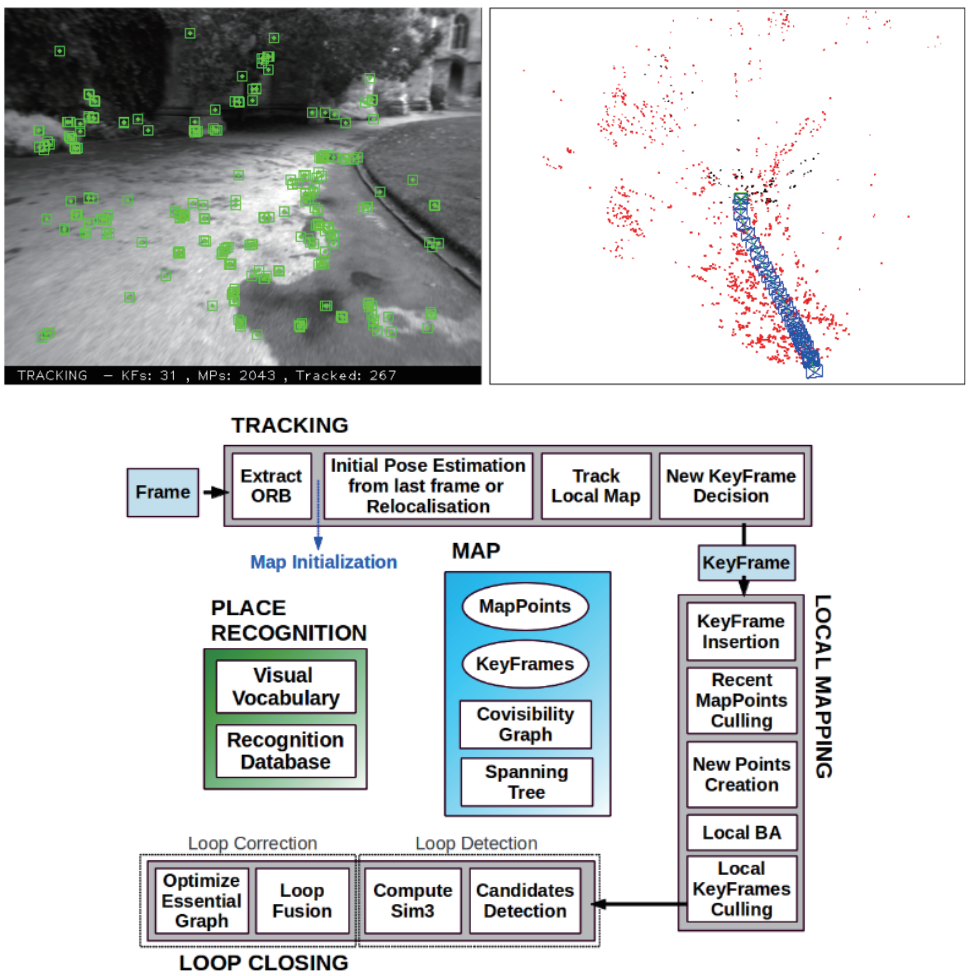


Figure 14-3: ORB-SLAM

1. RGB-D ORB-SLAM
2. ORB ORB ORB ORB SIFT SURF CPU Harris
3. ORB ORB-SLAM SLAM ORB-SLAM ORB *
4. ORB-SLAM SLAM Tracking Bundle Adjustment Co-visibility Graph Pose Graph Essential Graph Tracking ORB Adjustment

* ORB-SLAM

PTAM ORB-SLAM

5. ORB-SLAM OpenCV 4 PTAM ORB-SLAM ORB-SLAM

ORB-SLAM SLAM ORB-SLAM
ORB-SLAM SLAM ORB ORB-SLAM CPU PC SLAM ORB-SLAM

14.1.4 LSD-SLAM

LSD-SLAM Large Scale Direct monocular SLAM J. Engel 2014 SLAM [69, 75] ORB-SLAM LSD-SLAM SLAM LSD-SLAM SLAM — —

1. LSD-SLAM 8 13 LSD-SLAM

2. LSD-SLAM CPU RGB-D GPU [137] TUM

3. LSD-SLAM LSD-SLAM 5 SSD LSD-SLAM $\zeta \in \mathfrak{sim}(3)$

Figure 14-4 LSD LSD-SLAM CPU

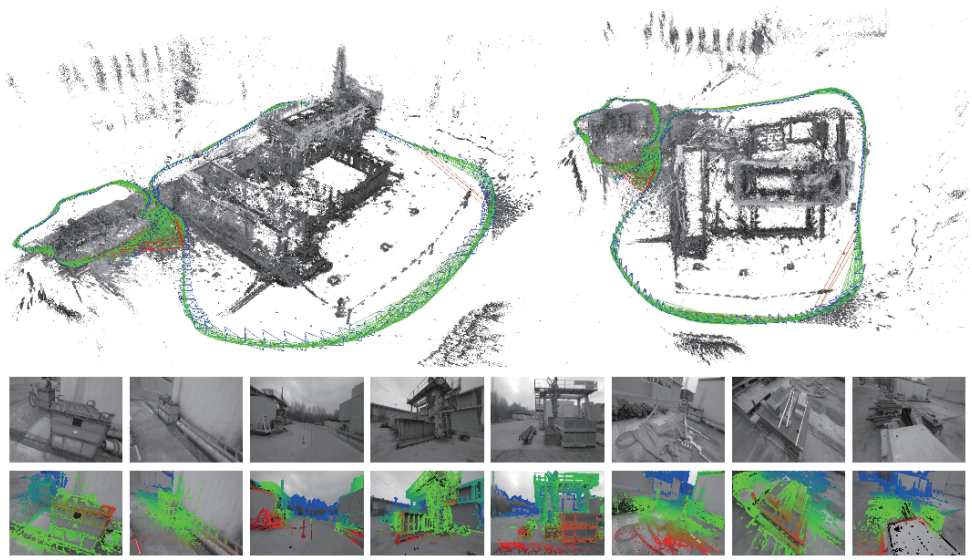


Figure 14-4: LSD-SLAM

LSD-SLAM LSD-SLAM L SLAM

14.1.5 SVO

SVO Semi-direct Visual Odoemtry [68] Forster 2014 “ ” “ ”
4 SVO 4 × 4
SVO PC 100 SVO
2.0 400 SVO AR/VR SVO

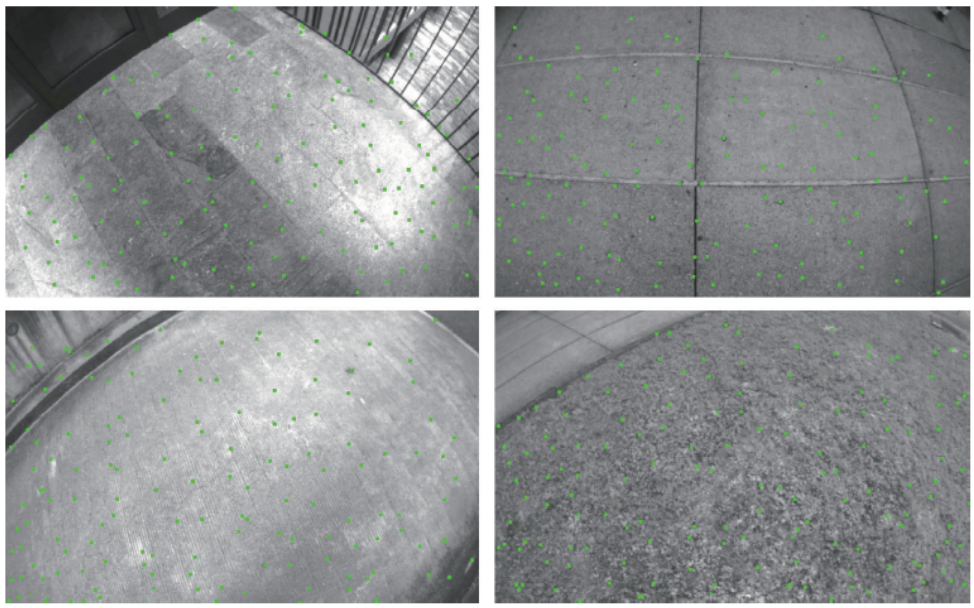


Figure 14-5: SVO

SVO – 13 SVO
SVO SLAM SVO

1. SVO SVO ***H*** ***F*** ***E***
2. SVO SVO VO SLAM

14.1.6 RTAB-MAP

SLAM RGB-D SLAM RGB-D SLAM CPU
RTAB-MAP Real Time Appearance-Based Mapping [119] RGB-D SLAM RGB-D SLAM RTAB-MAP RGB-D SLAM ROS Project Tango App Figure 14-6
RTAB-MAP RGB-D Kinect Xtion RTAB-MAP SLAM

14.1.7

`openslam.org` DVO-SLAM[138] RGBD-SLAM-V2[97] DSO[70] Kinect Fusion SLAM

Figure 14-6: RTAB-MAP Google Project Tango

14.2 SLAM

* SLAM SLAM AR/VR

14.2.1 + SLAM

– SLAM “ ” Big Clean Problem SLAM SLAM IMU SLAM [139]

1. IMU Drift IMU IMU

*

SLAM IMU SLAM

2. IMU SLAM IMU

3. SLAM IMU

IMU IMU ——

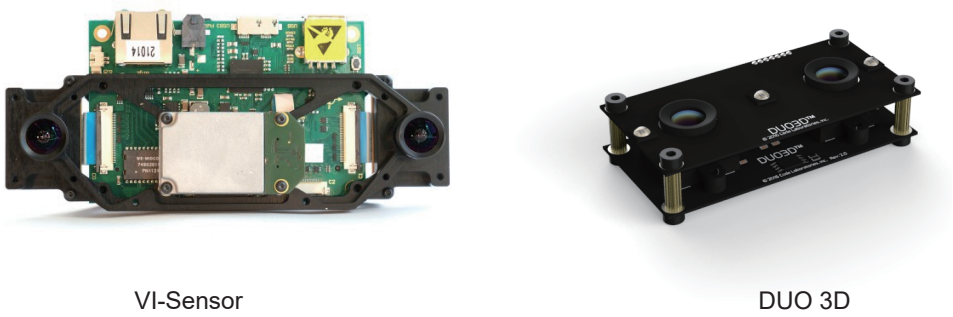


Figure 14-7: IMU

VIO Visual Inertial Odometry IMU VIO Loo
Coupled Tightly Coupled [140] IMU IMU —
— EKF[141] MSCKF Multi-State Constraint KF [142] EK
VIO SLAM SLAM VO

14.2.2 SLAM

SLAM SLAM SLAM
SLAM [147–150] SLAM BA [151]

1. SLAM SLAM BA

2. SLAM SLAM

Figure 14-8: SLAM [150, 152]

[120, 152] [150, 153–155]

14.2.3 SLAM

/ SLAM[165–167] SLAM[168–170] SLAM[82, 171, 172] [9] SLAM
SLAM SLAM

1. SLAM
2. SLAM

Appendix A

A.1

x $N(\mu, \sigma^2)$

$$p(x) = \frac{1}{\sqrt{2}\sigma} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right). \tag{A.1}$$

$$p(x) = \frac{1}{\sqrt{(2)^N \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right). \tag{A.2}$$

A.2

A.2.1

$$\boldsymbol{x} \sim N(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_{xx}), \quad \boldsymbol{y} \sim N(\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_{yy}),$$

$$\boldsymbol{x} + \boldsymbol{y} \sim N(\boldsymbol{\mu}_x + \boldsymbol{\mu}_y, \boldsymbol{\Sigma}_{xx} + \boldsymbol{\Sigma}_{yy}). \tag{A.3}$$

a $\boldsymbol{x}$ $a\boldsymbol{x}$

$$a\boldsymbol{x} \sim N(a\boldsymbol{\mu}_x, a^2\boldsymbol{\Sigma}_{xx}). \tag{A.4}$$

$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ $\boldsymbol{y}$

$$\boldsymbol{y} \sim N(\boldsymbol{A}\boldsymbol{\mu}_x, \boldsymbol{A}\boldsymbol{\Sigma}_{xx}\boldsymbol{A}^{\mathrm{T}}). \tag{A.5}$$

A.2.2

$p(\boldsymbol{x}\boldsymbol{y}) = N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$

$$\begin{aligned} \boldsymbol{\Sigma}^{-1} &= \boldsymbol{\Sigma}_{xx}^{-1} + \boldsymbol{\Sigma}_{yy}^{-1} \\ \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} &= \boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\mu}_x + \boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\mu}_y. \end{aligned} \tag{A.6}$$

A.2.3

$\boldsymbol{x}$ $\boldsymbol{y}$

$$p(\boldsymbol{x}, \boldsymbol{y}) = N\left(\left[\begin{array}{c} \boldsymbol{\mu}_x \\ \boldsymbol{\mu}_y \end{array}\right], \left[\begin{array}{cc} \boldsymbol{\Sigma}_{xx} & \boldsymbol{\Sigma}_{xy} \\ \boldsymbol{\Sigma}_{yx} & \boldsymbol{\Sigma}_{yy} \end{array}\right]\right). \tag{A.7}$$

$p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{x}|\boldsymbol{y})\, p(\boldsymbol{y})$ $p(\boldsymbol{x}|\boldsymbol{y})$

$$p(\boldsymbol{x}|\boldsymbol{y}) = N\left(\boldsymbol{\mu}_x + \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}(\boldsymbol{y} - \boldsymbol{\mu}_y), \boldsymbol{\Sigma}_{xx} - \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\Sigma}_{yx}\right). \tag{A.8}$$

A.3

$\boldsymbol{x} \sim N(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_{xx})$ $\boldsymbol{y}$

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} + \boldsymbol{w} \tag{A.9}$$

$\boldsymbol{A}, \boldsymbol{b}$ $\boldsymbol{w}$ $\boldsymbol{w} \sim N(\boldsymbol{0}, \boldsymbol{R})$

$\boldsymbol{y}$

$$p(\boldsymbol{y}) = N\left(\boldsymbol{A}\boldsymbol{\mu}_x + \boldsymbol{b}, \boldsymbol{R} + \boldsymbol{A}\boldsymbol{\Sigma}_{xx}\boldsymbol{A}^{\mathrm{T}}\right). \tag{A.10}$$

Appendix B

$f(x)$ x $\frac{\mathrm{d}f}{\mathrm{d}x}$ x f

B.1

$\boldsymbol{x}$ $\boldsymbol{x} \in \mathbb{R}^m$

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \left[\frac{\mathrm{d}f}{\mathrm{d}x_1}, \cdots, \frac{\mathrm{d}f}{\mathrm{d}x_m}\right]^{\mathrm{T}} \in \mathbb{R}^m. \tag{B.1}$$

$m \times 1$ $\boldsymbol{x}^{\mathrm{T}}$

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}^{\mathrm{T}}} = \left[\frac{\mathrm{d}f}{\mathrm{d}x_1}, \cdots, \frac{\mathrm{d}f}{\mathrm{d}x_m}\right]. \tag{B.2}$$

$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}$ Jacobian

B.2

$\boldsymbol{F}(\boldsymbol{x})$

$$\boldsymbol{F}(\boldsymbol{x}) = [f_1(\boldsymbol{x}), \cdots, f_n(\boldsymbol{x})]^{\mathrm{T}},$$

f_k $\boldsymbol{x}$

$$\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}^{\mathrm{T}}} = \begin{bmatrix} \frac{\partial f_1}{\partial \boldsymbol{x}^{\mathrm{T}}} \\ \vdots \\ \frac{\partial f_n}{\partial \boldsymbol{x}^{\mathrm{T}}} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_m} \end{bmatrix} \in \mathbb{R}^{n \times m} \tag{B.3}$$

$n \times m$

$$\frac{\partial \boldsymbol{A}\boldsymbol{x}}{\partial \boldsymbol{x}^{\mathrm{T}}} = \boldsymbol{A}. \tag{B.4}$$

$$\frac{\partial \boldsymbol{F}^{\mathrm{T}}}{\partial \boldsymbol{x}} = \left(\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}^{\mathrm{T}}}\right)^{\mathrm{T}}. \tag{B.5}$$

$$\frac{\partial \boldsymbol{A}\boldsymbol{x}}{\partial \boldsymbol{x}} = \boldsymbol{A}. \tag{B.6}$$

Appendix C

ROS

ROS ROS ROS

C.1 ROS

ROS Robot Operating System Willow Garage 2007 ROS
ROS Ubuntu ROS Ubuntu ROS Box Turtle 2016 Kinetic Kame Figure C-1 ROS 2.0

Figure C-1: ROS

ROS Ubuntu Kubuntu Linux Mint Ubuntu GNOME Linux Windows
ROS C++ Python

C.2 ROS

ROS

ROS ROS
14 SLAM ORB-SLAM ORB-SLAM2 LSD-SLAM SVO DVO RTAB-MAP RGBD-SLAM-V2 Hector SLAM Gmapping ROVIO ROS ROS
SLAM 3 bag SLAM ROS bag topic SLAM topi

C.3 ROS

ROS ROS http://wiki.ros.org/ROS/Installation ROS ROS //www.aicrobo.com/ubuntu_for_ros.html
ROS

1. rqt rqt ROS GUI rqt ROS
2. rosbag rosbag ROS topic SLAM bag
3. rviz rviz ROS ROS

ROS ROS 1.x demo ROS2 ROS
ROS ROS SLAM ROS ROS SLAM

Bibliography

[1] L. Haomin, Z. Guofeng, and B. Hujun, "A survey of monocular simultaneous localization and mapping," *Journal of Computer-Aided Design and Compute Graphics*, vol. 28, no. 6, pp. 855–868, 2016. in Chinese.

[2] A. Davison, I. Reid, N. Molton, and O. Stasse, "Monoslam: Real-time single camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[3] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge university press, 2003.

[4] R. C. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *International Journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, 1986.

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

[6] T. Barfoot, "State estimation for robotics: A matrix lie group approach," 2016.

[7] A. Pretto, E. Menegatti, and E. Pagello, "Omnidirectional dense large-scale mapping and navigation based on meaningful triangulation," *2011 IEEE International Conference on Robotics and Automation (ICRA 2011)*, pp. 3289–96, 2011.

[8] B. Rueckauer and T. Delbruck, "Evaluation of event-based algorithms for optical flow with ground-truth from inertial measurement sensor," *Frontiers in neuroscience*, vol. 10, 2016.

[9] C. Cesar, L. Carlone, H. C., Y. Latif, D. Scaramuzza, J. Neira, I. D. Reid, and L. John J., "Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age," *arXiv preprint arXiv:1606.05830*, 2016.

[10] P. Newman and K. Ho, "Slam-loop closing with visually salient features," in *proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 635–642, IEEE, 2005.

[11] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," in *Autonomous robot vehicles*, pp. 167–193, Springer, 1990.

[12] P. Beeson, J. Modayil, and B. Kuipers, "Factoring the mapping problem: Mobile robot map-building in the hybrid spatial semantic hierarchy," *International Journal of Robotics Research*, vol. 29, no. 4, pp. 428–459, 2010.

[13] H. Strasdat, J. M. Montiel, and A. J. Davison, "Visual slam: Why filter?," *Image and Vision Computing*, vol. 30, no. 2, pp. 65–77, 2012.

[14] M. Liang, H. Min, and R. Luo, "Graph-based slam: A survey," *ROBOT*, vol. 35, no. 4, pp. 500–512, 2013. in Chinese.

[15] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, "Visual simultaneous localization and mapping: a survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 55–81, 2015.

[16] J. Boal, Á. Sánchez-Miralles, and Á. Arranz, "Topological simultaneous localization and mapping: a survey," *Robotica*, vol. 32, pp. 803–821, 2014.

[17] S. Y. Chen, "Kalman filter for robot vision: A survey," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 11, pp. 4409–4420, 2012.

[18] Z. Chen, J. Samarabandu, and R. Rodrigo, "Recent advances in simultaneous localization and map-building using computer vision," *Advanced Robotics*, vol. 21, no. 3-4, pp. 233–265, 2007.

[19] J. Stuelpnagel, "On the parametrization of the three-dimensional rotation group," *SIAM Review*, vol. 6, no. 4, pp. 422–430, 1964.

[20] T. Barfoot, J. R. Forbes, and P. T. Furgale, "Pose estimation using linearized rotations and quaternion algebra," *Acta Astronautica*, vol. 68, no. 1-2, pp. 101–112, 2011.

[21] V. S. Varadarajan, *Lie groups, Lie algebras, and their representations*, vol. 102. Springer Science & Business Media, 2013.

[22] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An invitation to 3-d vision: from images to geometric models*, vol. 26. Springer Science & Business Media, 2012.

[23] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d SLAM systems," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 573–580, IEEE, 2012.

[24] H. Strasdat, *Local accuracy and global consistency for efficient visual slam*. PhD thesis, Citeseer, 2012.

[25] Z. Zhang, "Flexible camera calibration by viewing a plane from unknown orientations," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 1, pp. 666–673, Ieee, 1999.

[26] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 30, no. 2, pp. 328–341, 2008.

[27] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International journal of computer vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[28] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A comparison and evaluation of multi-view stereo reconstruction algorithms," in *null*, pp. 519–528, IEEE, 2006.

[29] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski, "Building rome in a day," in *2009 IEEE 12th international conference on computer vision*, pp. 72–79, IEEE, 2009.

[30] P. Wolfe, "Convergence conditions for ascent methods," *SIAM review*, vol. 11, no. 2, pp. 226–235, 1969.

[31] J. Nocedal and S. Wright, *Numerical Optimization*. Springer Science & Business Media, 2006.

[32] M. I. Lourakis and A. A. Argyros, "Sba: A software package for generic sparse bundle adjustment," *ACM Transactions on Mathematical Software (TOMS)*, vol. 36, no. 1, p. 2, 2009.

[33] G. Sibley, "Relative bundle adjustment," *Department of Engineering Science, Oxford University, Tech. Rep*, vol. 2307, no. 09, 2009.

[34] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment: a modern synthesis," in *Vision algorithms: theory and practice*, pp. 298–372, Springer, 2000.

[35] S. Agarwal, K. Mierle, and Others, "Ceres solver." `http://ceres-solver.org`.

[36] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: a general framework for graph optimization," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3607–3613, IEEE, 2011.

[37] Wikipedia, "Feature (computer vision)." `"https://en.wikipedia.org/wiki/Feature_(computer_vision)"`, 2016. [Online; accessed 09-July-2016].

[38] C. Harris and M. Stephens, "A combined corner and edge detector.," in *Alvey vision conference*, vol. 15, pp. 10–5244, Citeseer, 1988.

[39] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision–ECCV 2006*, pp. 430–443, Springer, 2006.

[40] J. Shi and C. Tomasi, "Good features to track," in *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pp. 593–600, IEEE, 1994.

[41] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[42] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision–ECCV 2006*, pp. 404–417, Springer, 2006.

[43] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: an efficient alternative to sift or surf," in *2011 IEEE International Conference on Computer Vision (ICCV)*, pp. 2564–2571, IEEE, 2011.

[44] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *European conference on computer vision*, pp. 778–792, Springer, 2010.

[45] M. Nixon and A. S. Aguado, *Feature extraction and image processing for computer vision*. Academic Press, 2012.

[46] P. L. Rosin, "Measuring corner properties," *Computer Vision and Image Understanding*, vol. 73, no. 2, pp. 291–307, 1999.

[47] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration.," in *VISAPP (1)*, pp. 331–340, 2009.

[48] R. I. Hartley, "In defense of the eight-point algorithm," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 19, no. 6, pp. 580–593, 1997.

[49] H. C. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms, MA Fischler and O. Firschein, eds*, pp. 61–62, 1987.

[50] H. Li and R. Hartley, "Five-point motion estimation made easy," in *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 1, pp. 630–633, IEEE, 2006.

[51] D. Nistér, "An efficient solution to the five-point relative pose problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 756–770, 2004.

[52] O. D. Faugeras and F. Lustman, "Motion and structure from motion in a piecewise planar environment," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 2, no. 03, pp. 485–508, 1988.

[53] Z. Zhang and A. R. Hanson, "3d reconstruction based on homography mapping," *ARPA Image Understanding Workshop*, pp. 1007–1012, 1996.

[54] E. Malis and M. Vargas, *Deeper understanding of the homography decomposition for vision-based control*. PhD thesis, INRIA, 2007.

[55] A. J. Davison, "Real-time simultaneous localisation and mapping with a single camera," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pp. 1403–1410, IEEE, 2003.

[56] X.-S. Gao, X.-R. Hou, J. Tang, and H.-F. Cheng, "Complete solution classification for the perspective-three-point problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, pp. 930–943, Aug 2003.

[57] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Epnp: An accurate o(n) solution to the pnp problem," *International Journal of Computer Vision*, vol. 81, no. 2, pp. 155–166, 2008.

[58] A. Penate-Sanchez, J. Andrade-Cetto, and F. Moreno-Noguer, "Exhaustive linearization for robust camera pose and focal length estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 10, pp. 2387–2400, 2013.

[59] L. Chen, C. W. Armstrong, and D. D. Raftopoulos, "An investigation on the accuracy of three-dimensional space reconstruction using the direct linear transformation technique," *Journal of Biomechanics*, vol. 27, no. 4, pp. 493–500, 1994.

[60] B. F. Green, "The orthogonal approximation of an oblique structure in factor analysis," *Psychometrika*, vol. 17, no. 4, pp. 429–440, 1952.

[61] iplimage, "P3p(blog)." `"http://iplimage.com/blog/p3p-perspective-point-overview/"`, 2016.

[62] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-d point sets," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 5, pp. 698–700, 1987.

[63] F. Pomerleau, F. Colas, and R. Siegwart, "A review of point cloud registration algorithms for mobile robotics," *Foundations and Trends in Robotics (FnTROB)*, vol. 4, no. 1, pp. 1–104, 2015.

[64] O. D. Faugeras and M. Hebert, "The representation, recognition, and locating of 3-d objects," *The International Journal of Robotics Research*, vol. 5, no. 3, pp. 27–52, 1986.

[65] B. K. Horn, "Closed-form solution of absolute orientation using unit quaternions," *JOSA A*, vol. 4, no. 4, pp. 629–642, 1987.

[66] G. C. Sharp, S. W. Lee, and D. K. Wehe, "Icp registration using invariant features," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 1, pp. 90–102, 2002.

[67] G. Silveira, E. Malis, and P. Rives, "An efficient direct approach to visual slam," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 969–979, 2008.

[68] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on* (rs, ed.), pp. 15–22, IEEE, 2014.

[69] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *Computer Vision–ECCV 2014*, pp. 834–849, Springer, 2014.

[70] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *arXiv preprint arXiv:1607.02565*, 2016.

[71] B. D. Lucas, T. Kanade, *et al.*, "An iterative image registration technique with an application to stereo vision," 1981.

[72] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial intelligence*, vol. 17, no. 1-3, pp. 185–203, 1981.

[73] S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework," *International journal of computer vision*, vol. 56, no. 3, pp. 221–255, 2004.

[74] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, 2013.

[75] J. Engel, J. Sturm, and D. Cremers, "Semi-dense visual odometry for a monocular camera," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1449–1456, 2013.

[76] J. Sherman and W. J. Morrison, "Adjustment of an inverse matrix corresponding to a change in one element of a given matrix," *The Annals of Mathematical Statistics*, vol. 21, no. 1, pp. 124–127, 1950.

[77] V. Sujan and S. Dubowsky, "Efficient information-based visual robotic mapping in unstructured environments," *International Journal of Robotics Research*, vol. 24, no. 4, pp. 275–293, 2005.

[78] F. Janabi-Sharifi and M. Marey, "A kalman-filter-based method for pose estimation in visual servoing," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 939–947, 2010.

[79] S. Li and P. Ni, "Square-root unscented kalman filter based simultaneous localization and mapping," in *Information and Automation (ICIA), 2010 IEEE International Conference on*, pp. 2384–2388, IEEE, 2010.

[80] R. Sim, P. Elinas, and J. Little, "A study of the rao-blackwellised particle filter for efficient and accurate vision-based slam," *International Journal of Computer Vision*, vol. 74, no. 3, pp. 303–318, 2007.

[81] J. S. Lee, S. Y. Nam, and W. K. Chung, "Robust rbpf-slam for indoor mobile robots using sonar sensors in non-static environments," *Advanced Robotics*, vol. 25, no. 9-10, pp. 1227–1248, 2011.

[82] A. Gil, O. Reinoso, M. Ballesta, and M. Julia, "Multi-robot visual slam using a rao-blackwellized particle filter," *Robotics and Autonomous Systems*, vol. 58, no. 1, pp. 68–80, 2010.

[83] G. Sibley, L. Matthies, and G. Sukhatme, "Sliding window filter with application to planetary landing," *Journal of Field Robotics*, vol. 27, no. 5, pp. 587–608, 2010.

[84] L. M. Paz, J. D. Tardós, and J. Neira, "Divide and conquer: Ekf slam in o(n)," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1107–1120, 2008.

[85] O. G. Grasa, J. Civera, and J. Montiel, "Ekf monocular slam with relocalization for laparoscopic sequences," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 4816–4821, IEEE, 2011.

[86] E. Süli and D. F. Mayers, *An Introduction to Numerical Analysis*. Cambridge university press, 2003.

[87] L. Polok, V. Ila, M. Solony, P. Smrz, and P. Zemcik, "Incremental block cholesky factorization for nonlinear least squares in robotics.," in *Robotics: Science and Systems*, 2013.

[88] R. Mur-Artal, J. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *arXiv preprint arXiv:1502.00956*, 2015.

[89] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual–inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015.

[90] "Bundle adjustment in the large." `http://grail.cs.washington.edu/projects/bal/`.

[91] G. Sibley, L. Matthies, and G. Sukhatme, "A sliding window filter for incremental slam," in *Unifying perspectives in computational and robot vision*, pp. 103–112, Springer, 2008.

[92] H. Strasdat, A. J. Davison, J. M. M. Montiel, and K. Konolige, "Double window optimisation for constant time visual SLAM," *2011 IEEE International Conference On Computer Vision (ICCV)*, pp. 2352–2359, 2011.

[93] G. Dubbelman and B. Browning, "Cop-slam: Closed-form online pose-chain optimization for visual slam," *Robotics, IEEE Transactions on*, vol. 31, pp. 1194–1213, Oct 2015.

[94] D. Lee and H. Myung, "Solution to the slam problem in low dynamic environments using a pose graph and an rgb-d sensor," *Sensors*, vol. 14, no. 7, pp. 12467–12496, 2014.

[95] Y. Latif, C. Cadena, and J. Neira, "Robust loop closing over time for pose graph slam," *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1611–1626, 2013.

[96] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pp. 225–234, IEEE, 2007.

[97] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-d mapping with an rgb-d camera," *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 177–187, 2014.

[98] D. Hahnel, W. Burgard, D. Fox, and S. Thrun, "An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements," in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 1, pp. 206–211, IEEE, 2003.

[99] I. Ulrich and I. Nourbakhsh, "Appearance-based place recognition for topological localization," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2, pp. 1023–1029, Ieee, 2000.

[100] X. Gao and T. Zhang, "Robust rgb-d simultaneous localization and mapping using planar point features," *Robotics and Autonomous Systems*, vol. 72, pp. 1–14, 2015.

[101] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[102] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027–1035, Society for Industrial and Applied Mathematics, 2007.

[103] M. Cummins and P. Newman, "Fab-map: Probabilistic localization and mapping in the space of appearance," *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 647–665, 2008.

[104] M. Cummins and P. Newman, "Accelerating fab-MAP with concentration inequalities," *IEEE Transactions On Robotics*, vol. 26, no. 6, pp. 1042–1050, 2010.

[105] M. Cummins and P. Newman, "Appearance-only slam at large scale with fab-map 2.0," *International Journal of Robotics Research*, vol. 30, no. 9, pp. 1100–1123, 2011.

[106] C. Chow and C. Liu, "Approximating discrete probability distributions with dependence trees," *IEEE transactions on Information Theory*, vol. 14, no. 3, pp. 462–467, 1968.

[107] D. Galvez-Lopez and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions On Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[108] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[109] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pp. 1470–1477, IEEE, 2003.

[110] S. Robertson, "Understanding inverse document frequency: on theoretical arguments for idf," *Journal of documentation*, vol. 60, no. 5, pp. 503–520, 2004.

[111] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, pp. 2161–2168, IEEE, 2006.

[112] C. Cadena, D. Galvez-Lopez, J. D. Tardos, and J. Neira, "Robust place recognition with stereo sequences," *IEEE Transactions on Robotics*, vol. 28, no. 4, pp. 871–885, 2012.

[113] X. Gao and T. Zhang, "Loop closure detection for visual slam systems using deep neural networks," in *Control Conference (CCC), 2015 34th Chinese*, pp. 5851–5856, IEEE, 2015.

[114] X. Gao and T. Zhang, "Unsupervised learning to detect loops using deep neural networks for visual slam system," *Autonomous Robots*, pp. 1–18, 2015.

[115] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "Netvlad: Cnn architecture for weakly supervised place recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5297–5307, 2016.

[116] M. Angelina Uy and G. Hee Lee, "Pointnetvlad: Deep point cloud based retrieval for large-scale place recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4470–4479, 2018.

[117] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2938–2946, 2015.

[118] B. Williams, M. Cummins, J. Neira, P. Newman, I. Reid, and J. Tardós, "A comparison of loop closing techniques in monocular slam," *Robotics and Autonomous Systems*, vol. 57, no. 12, pp. 1188–1197, 2009.

[119] M. Labbé and F. Michaud, "Online global loop closure detection for large-scale multi-session graph-based slam," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2661–2666, IEEE, 2014.

[120] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. J. Kelly, and A. J. Davison, "Slam++: Simultaneous localisation and mapping at the level of objects," *2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1352–9, 2013.

[121] M. Pizzoli, C. Forster, and D. Scaramuzza, "Remode: Probabilistic, monocular dense reconstruction in real time," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2609–2616, IEEE, 2014.

[122] "Correlation based similarity measure-summary." `https://siddhantahuja.wordpress.com/tag/stereo-matching/`.

[123] H. Hirschmuller and D. Scharstein, "Evaluation of cost functions for stereo matching," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE, 2007.

[124] G. Vogiatzis and C. Hernández, "Video-based, real-time multi-view stereo," *Image and Vision Computing*, vol. 29, no. 7, pp. 434–441, 2011.

[125] A. Handa, R. A. Newcombe, A. Angeli, and A. J. Davison, "Real-time camera tracking: When is high frame-rate best?," in *European Conference on Computer Vision*, pp. 222–235, Springer, 2012.

[126] J. Montiel, J. Civera, and A. J. Davison, "Unified inverse depth parametrization for monocular slam," *analysis*, vol. 9, p. 1, 2006.

[127] J. Civera, A. J. Davison, and J. M. Montiel, "Inverse depth parametrization for monocular slam," *IEEE transactions on robotics*, vol. 24, no. 5, pp. 932–945, 2008.

[128] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, vol. 7, 2006.

[129] J. Stuckler and S. Behnke, "Multi-resolution surfel maps for efficient dense 3d modeling and tracking," *Journal of Visual Communication and Image Representation*, vol. 25, no. 1, pp. 137–147, 2014.

[130] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[131] M. Burri, H. Oleynikova, M. W. Achtelik, and R. Siegwart, "Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pp. 1872–1878, IEEE, 2015.

[132] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE international symposium on Mixed and augmented reality (ISMAR)*, pp. 127–136, IEEE, 2011.

[133] R. A. Newcombe, D. Fox, and S. M. Seitz, "Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 343–352, 2015.

[134] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison, "Elasticfusion: Dense slam without a pose graph," *Proc. Robotics: Science and Systems, Rome, Italy*, 2015.

[135] M. Dou, S. Khamis, Y. Degtyarev, P. Davidson, S. R. Fanello, A. Kowdle, S. O. Escolano, C. Rhemann, D. Kim, J. Taylor, *et al.*, "Fusion4d: real-time performance capture of challenging scenes," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 114, 2016.

[136] M. Innmann, M. Zollhöfer, M. Nießner, C. Theobalt, and M. Stamminger, "Volumedeform: Real-time volumetric non-rigid reconstruction," *arXiv preprint arXiv:1603.08161*, 2016.

[137] C. Kerl, J. Sturm, and D. Cremers, "Robust odometry estimation for rgb-d cameras," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 3748–3754, IEEE, 2013.

[138] C. Kerl, J. Sturm, and D. Cremers, "Dense visual slam for rgb-d cameras," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2100–2106, IEEE, 2013.

[139] J. Gui, D. Gu, S. Wang, and H. Hu, "A review of visual inertial odometry from filtering and optimisation perspectives," *Advanced Robotics*, vol. 29, pp. 1289–1301, Oct 18 2015.

[140] A. Martinelli, "Closed-form solution of visual-inertial structure from motion," *International Journal of Computer Vision*, vol. 106, no. 2, pp. 138–152, 2014.

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

[142] M. Li and A. I. Mourikis, "High-precision, consistent ekf-based visual-inertial odometry," *International Journal of Robotics Research*, vol. 32, pp. 690–711, MAY 2013.

[143] G. Huang, M. Kaess, and J. J. Leonard, "Towards consistent visual-inertial navigation," in *2014 IEEE International Conference on Robotics and Automation (icra)*, IEEE International Conference on Robotics and Automation ICRA, pp. 4926–4933, 2014. IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, PEOPLES R CHINA, MAY 31-JUN 07, 2014.

[144] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation," in *Robotics: Science and Systems XI*, no. EPFL-CONF-214687, 2015.

[145] M. Tkocz and K. Janschek, "Towards consistent state and covariance initialization for monocular slam filters," *Journal of Intelligent & Robotic Systems*, vol. 80, pp. 475–489, DEC 2015.

[146] V. Usenko, J. Engel, J. Stueckler, and D. Cremers, "Direct visual-inertial odometry with stereo cameras," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2016.

[147] A. Nüchter and J. Hertzberg, "Towards semantic maps for mobile robots," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 915–926, 2008.

[148] J. Civera, D. Gálvez-López, L. Riazuelo, J. D. Tardós, and J. Montiel, "Towards semantic slam using a monocular camera," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 1277–1284, IEEE, 2011.

[149] H. S. Koppula, A. Anand, T. Joachims, and A. Saxena, "Semantic labeling of 3d point clouds for indoor scenes," in *Advances in Neural Information Processing Systems*, pp. 244–252, 2011.

[150] A. Anand, H. S. Koppula, T. Joachims, and A. Saxena, "Contextually guided semantic labeling and search for three-dimensional point clouds," *The International Journal of Robotics Research*, p. 0278364912461538, 2012.

[151] N. Fioraio and L. Di Stefano, "Joint detection, tracking and mapping by semantic bundle adjustment," *2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1538–45, 2013.

[152] R. F. Salas-Moreno, B. Glocken, P. H. Kelly, and A. J. Davison, "Dense planar slam," in *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pp. 157–164, IEEE, 2014.

[153] J. Stückler, N. Biresev, and S. Behnke, "Semantic mapping using object-class segmentation of rgb-d images," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3005–3010, IEEE, 2012.

[154] I. Kostavelis and A. Gasteratos, "Learning spatially semantic representations for cognitive robot navigation," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1460–1475, 2013.

[155] C. Couprie, C. Farabet, L. Najman, and Y. LeCun, "Indoor semantic segmentation using depth information," *arXiv preprint arXiv:1301.3572*, 2013.

[156] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR09*, 2009.

[157] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[158] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[159] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[160] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *arXiv preprint arXiv:1411.4038*, 2014.

[161] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. Torr, "Conditional random fields as recurrent neural networks," in *International Conference on Computer Vision (ICCV)*, 2015.

[162] S. Gupta, P. Arbeláez, R. Girshick, and J. Malik, "Indoor scene understanding with rgb-d images: Bottom-up segmentation, object detection and semantic segmentation," *International Journal of Computer Vision*, pp. 1–17, 2014.

[163] K. Konda and R. Memisevic, "Learning visual odometry with a convolutional network," in *International Conference on Computer Vision Theory and Applications*, 2015.

[164] Y. Hou, H. Zhang, and S. Zhou, "Convolutional neural network-based image representation for visual loop closure detection," *arXiv preprint arXiv:1504.05241*, 2015.

[165] S. Y. An, J. G. Kang, L. K. Lee, and S. Y. Oh, "Line segment-based indoor mapping with salient line feature extraction," *Advanced Robotics*, vol. 26, no. 5-6, pp. 437–460, 2012.

[166] H. Zhou, D. Zou, L. Pei, R. Ying, P. Liu, and W. Yu, "Structslam: Visual slam with building structure lines," *Vehicular Technology, IEEE Transactions on*, vol. 64, pp. 1364–1375, April 2015.

[167] D. Benedettelli, A. Garulli, and A. Giannitrapani, "Cooperative slam using m-space representation of linear features," *Robotics and Autonomous Systems*, vol. 60, no. 10, pp. 1267–1278, 2012.

[168] J. P. Saarinen, H. Andreasson, T. Stoyanov, and A. J. Lilienthal, "3d normal distributions transform occupancy maps: An efficient representation for mapping in dynamic environments," *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1627–1644, 2013.

[169] W. Maddern, M. Milford, and G. Wyeth, "Cat-slam: probabilistic localisation and mapping using a continuous appearance-based trajectory," *International Journal of Robotics Research*, vol. 31, no. 4SI, pp. 429–451, 2012.

[170] H. Wang, Z.-G. Hou, L. Cheng, and M. Tan, "Online mapping with a mobile robot in dynamic and unknown environments," *International Journal of Modelling, Identification and Control*, vol. 4, no. 4, pp. 415–423, 2008.

[171] D. Zou and P. Tan, "Coslam: Collaborative visual SLAM in dynamic environments," *IEEE Transactions On Pattern Analysis And Machine Intelligence*, vol. 35, no. 2, pp. 354–366, 2013.

[172] T. A. Vidal-Calleja, C. Berger, J. Sola, and S. Lacroix, "Large scale multiple robot visual mapping with heterogeneous landmarks in semi-structured terrain," *Robotics and Autonomous Systems*, vol. 59, no. 9, pp. 654–674, 2011.