การระบุตำแหน่งพร้อมกับการสร้างแผนที่ขนาดใหญ่ด้วยกล้องวิดีโอมุมกว้าง

นายยุทธนา สุทธสุภา

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรดุษฎีบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2557

LARGE SCALE SLAM USING WIDE-ANGLE CAMERA

Mr. Yuttana Suttasupa

A Dissertation Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy Program in Computer Engineering

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2014

หัวข้อวิทยานิพนธ์	การระบุตำแหน่งพร้อมกับการสร้างแผนที่ขนาดใหญ่ด้วยกล้องวิดีโอมุมกว้าง
โดย	นายยุทธนา สุทธสุภา
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ผู้ช่วยศาสตราจารย์ ดร. อรรถวิทย์ สุดแสง

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาดุษฎีบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์

(ศาสตราจารย์ ดร. บัณฑิต เอื้ออาภรณ์)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ

(ศาสตราจารย์ ดร. ประภาส จงสถิตย์วัฒนา)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

(ผู้ช่วยศาสตราจารย์ ดร. อรรถวิทย์ สุดแสง)

.....กรรมการ

(ผู้ช่วยศาสตราจารย์ ดร. นัทที นิภานันท์)

.....กรรมการ

(ดร. ณัฐพงศ์ ชินธเนศ)

.....กรรมการภายนอกมหาวิทยาลัย

(ผู้ช่วยศาสตราจารย์ ดร. ถวิดา มณีวรรณ์)

ยุทธนา สุทธสุภา : การระบุตำแหน่งพร้อมกับการสร้างแผนที่ขนาดใหญ่ด้วยกล้องวิดีโอมุมกว้าง (LARGE SCALE SLAM USING WIDE-ANGLE CAMERA) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ผศ. ดร. อรรถวิทย์ สุดแสง, 152 หน้า.

วิธีการระบุตำแหน่งพร้อมกับการสร้างแผนที่ เป็นกระบวนการที่หุ่นยนต์สามารถระบุตำแหน่งของตนว่าอยู่ ณ จุดใดในแผนที่ ในขณะเดียวกันหุ่นยนต์ก็จะทำการสร้างแผนที่ของสิ่งแวดล้อมในบริเวณที่หุ่นยนต์เคลื่อนที่ผ่านไปพร้อม ๆ กัน โดยที่หุ่นยนต์นั้นไม่มีข้อมูลของสิ่งแวดล้อมมาก่อน ในงานวิจัยนี้ได้นำเสนอวิธีการระบุตำแหน่งพร้อมกับการสร้างแผนที่ขนาดใหญ่ด้วยกล้องวิดีโอมุมกว้างโดยกล้องวิดีโอสามารถเคลื่อนที่ได้อย่างอิสระ ความท้าทายของการสร้างแผนที่ขนาดใหญ่ก็คือการจัดการประสิทธิภาพของอัลกอริทึมให้สามารถทำงานได้ทันการณ์ นอกจากนี้อัลกอริทึมยังต้องทนทานต่อข้อมูลรบกวน และต้องรับมือกับปัญหาการเบี่ยงเบนของขนาดแผนที่อันเกิดจากการใช้กล้องวิดีโอตัวเดียว

ผลลัพธ์ที่ได้จากการทำงานของอัลกอริทึม จะเป็นแผนที่ของสิ่งแวดล้อมในบริเวณที่กล้องเคลื่อนที่ผ่าน และสถานะของกล้องในสามมิติ โดยแผนที่จะถูกอธิบายด้วยตำแหน่งของจุดสังเกตในสามมิติจำนวนมาก ส่วนสถานะของกล้องจะประกอบด้วยตำแหน่งของกล้องและทิศทางการวางตัวของกล้องในสามมิติ และในตอนท้ายของงานวิจัยได้แสดงการทดลองการทำงานของการระบุตำแหน่งพร้อมกับการสร้างแผนที่ด้วยกล้องวิดีโอแบบเลนส์ตาปลาในสิ่งแวดล้อมจริง โดยอาศัยมนุษย์ในการถือกล้องและเคลื่อนที่กล้องไปมาในสามมิติ ซึ่งอัลกอริทึมที่ได้นำเสนอนั้นก็สามารถระบุตำแหน่งของกล้อง และสร้างแผนที่ของสิ่งแวดล้อมได้เป็นอย่างดี

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

ปีการศึกษา 2557

5271865821 : MAJOR COMPUTER ENGINEERING

KEYWORDS: SIMULTANEOUS LOCALIZATION AND MAPPING / SLAM / WIDE-ANGLE CAMERA

YUTTANA SUTTASUPA: LARGE SCALE SLAM USING WIDE-ANGLE CAMERA. ADVISOR: ASST. PROF. ATTAWITH SUDSANG, Ph.D., 152 pp.

Simultaneous localization and mapping (SLAM) is a technique for a robot to automatically determine its location and to build up an environment map simultaneously while traversing in an unknown environment. This work proposes a real-time SLAM method for a hand-held wide-angle camera, that is allowed to move freely in a large scale 3D environment. The challenge of the work lies on achieving the desired robustness while attaining the real-time performance as well as ability to cope with noises and the problem of scale drift in monocular localization and mapping.

Results produced by the algorithm consist of an environment map and the camera states along the traversed trajectory. The map is represented in the form of a massive group of 3D landmarks and the camera state is composed of the camera's position and direction. Several experiments are performed using a human carrying a fish-eye camera while traversing in a variety of 3D environments. The results confirm the efficiency and effectiveness of the proposed SLAM method.

Department:	Computer Engineering	Student's Signature	.....
Field of Study:	Computer Engineering	Advisor's Signature	.....
Academic Year:	2014		

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี เนื่องจากการสนับสนุนและส่งเสริมเป็นอย่างดีจาก ผศ.ดร.อรรถวิทย์ สุดแสง อาจารย์ที่ปรึกษาวิทยานิพนธ์ ซึ่งเป็นที่ปรึกษาทั้งในส่วนของแนวทางในการทำงานวิจัยชิ้นนี้ รวมถึงข้อแนะนำต่าง ๆ ของงานวิจัยและเรื่องอื่น ๆ นอกเหนือจากงานวิจัยข้าพเจ้าจึงขอขอบคุณอาจารย์เป็นอย่างสูง

ขอขอบคุณประธานกรรมการสอบวิทยานิพนธ์ ศาสตราจารย์ ดร.ประภาส จงสถิตย์วัฒนา ตลอดจนกรรมการสอบวิทยานิพนธ์ อาจารย์ ดร. นัทที นิภานันท์, อาจารย์ ดร. ณัฐพงศ์ ชินธเนศ และ ผู้ช่วยศาสตราจารย์ ดร. ถวิดา มณีวรรณ์ ที่ได้กรุณาสละเวลา ตรวจสอบและให้คำแนะนำเกี่ยวกับวิทยานิพนธ์ฉบับนี้จนเสร็จสมบูรณ์

ที่จะขาดเสียมิได้ คือขอขอบคุณเหล่าพี่ ๆ เพื่อน ๆ และ น้อง ๆ แห่งห้องปฏิบัติการวิจัยระบบอัจฉริยะ ISL2 ทุก ๆ คนที่ช่วยให้คำแนะนำต่าง ๆ ทั้งในเรื่องของงานวิจัยและเรื่องอื่น ๆ รวมถึงแนวทางในการแก้ปัญหาและอุปสรรคต่าง ๆ

ขอขอบคุณ บิดามารดา ผู้ให้กำเนิด รวมไปถึงญาติพี่น้องทุกคน ที่ให้กำลังใจและสนับสนุนผู้วิจัยตลอดมา และขอขอบคุณอีกหลาย ๆ ท่านที่ไม่สามารถเอ่ยนามได้ทั้งหมด ณ ที่นี้ด้วยใจจริง

สุดท้ายนี้ ขอขอบคุณทุนอุดหนุนการศึกษาระดับบัณฑิตศึกษา จุฬาลงกรณ์มหาวิทยาลัย เพื่อเฉลิมฉลองวโรกาสที่พระบาทสมเด็จพระเจ้าอยู่หัวทรงพระเจริญพระชนมายุครบ 72 พรรษา มา ณ ที่นี้

สารบัญ

หน้า

หน้า

หน้า

สารบัญตาราง

สารบัญรูปภาพ

บทที่ 1
บทนำ

การสร้างแผนที่ (Mapping) ของสิ่งแวดล้อม คือกระบวนการที่นำเอาข้อมูลการวัดที่ตรวจวัดได้จากสิ่งแวดล้อมจากอุปกรณ์วัดค่าต่าง ๆ (Sensors) มาใช้ประกอบกันเพื่อสร้างเป็นโครงสร้างข้อมูลสำหรับอธิบายสิ่งแวดล้อม ณ บริเวณนั้น ส่วนการระบุตำแหน่ง (Localization) คือการอธิบายการวางตัวของหุ่นยนต์ หรือวัตถุต่าง ๆ ที่เราสนใจในแผนที่ โดยแผนที่นั้นอาจเป็นแผนที่ที่ได้กำหนดไว้ล่วงหน้าแล้ว หรืออาจเป็นแผนที่ ซึ่งสร้างไปพร้อม ๆ กับการระบุตำแหน่งก็ได้

สำหรับการทำงานของหุ่นยนต์อัตโนมัติ การระบุตำแหน่งและการสร้างแผนที่ (Localization and Mapping) ถือเป็นงานที่มีความสำคัญสำหรับหุ่นยนต์เป็นอย่างมาก เนื่องจากว่าหุ่นยนต์ จำเป็นต้องใช้ข้อมูลแผนที่ ประกอบกับข้อมูลตำแหน่งของหุ่นยนต์ในการวางแผน เพื่อกระทำกิจกรรมต่าง ๆ ตอบสนองตามสถาพแวดล้อมนั้น อาทิเช่น การเคลื่อนที่สำหรับหุ่นยนต์สำรวจอัตโนมัติ, หุ่นยนต์กู้ภัย, หุ่นยนต์แม่บ้าน หรือ แม้แต่การหยิบจับสิ่งของ หุ่นยนต์ก็จำเป็นจะต้องรู้ตำแหน่งของ สิ่งของที่จะหยิบจับ และตำแหน่งของมือตนเองด้วย

แผนที่ที่ใช้สำหรับอธิบายสิ่งแวดล้อมของหุ่นยนต์นั้นมีได้หลายรูปแบบ ตามแต่วัตถุประสงค์ในการใช้งานของหุ่นยนต์ โดยอาจอยู่ในรูปแบบที่มนุษย์เข้าใจหรือไม่เข้าใจก็ได้ เช่นแผนที่ซึ่งอธิบายสิ่งแวดล้อมด้วยตำแหน่งของจุดสังเกตสำคัญต่าง ๆ, การอธิบายสิ่งแวดล้อมด้วยกลุ่มจุดจำนวนมาก (Point Cloud), การอธิบายสิ่งแวดล้อมด้วยตำแหน่งวัตถุในโลกจริง หรือการอธิบายสิ่งแวดล้อมด้วยโครงสร้างความสัมพันธ์ของสิ่งแวดล้อมก็เป็นได้ (Topology)

วิธีการระบุตำแหน่งพร้อมกับการสร้างแผนที่ (Simultaneous Localization and Mapping: SLAM) [1] เป็นกระบวนการที่หุ่นยนต์จะสร้างแผนที่ของสภาพแวดล้อมในขณะที่กำลังเคลื่อนที่ และระบุตำแหน่งของตัวเองในเวลาพร้อม ๆ กัน โดยที่หุ่นยนต์นั้นไม่มีข้อมูลของสิ่งแวดล้อมมาก่อน ซึ่ง SLAM นั้นมีความสำคัญเป็นอย่างมากสำหรับหุ่นยนต์ที่ต้องการการโต้ตอบแบบทันการณ์ การระบุตำแหน่งพร้อมกับการสร้างแผนที่ (SLAM) นั้นมีความท้าทายมากกว่าการระบุตำแหน่งเพียงอย่างเดียวหรือการสร้างแผนทีเพียงอย่างเดียว เนื่องจากว่าในระหว่างกระบวนการ หุ่นยนต์จะไม่มีข้อมูลล่วงหน้าของแผนที่ เพื่อใช้ในการอ้างอิงสำหรับระบุตำแหน่งหุ่นยนต์ หุ่นยนต์จำเป็นจะต้องสร้างแผนที่ไปพร้อม ๆ กับการระบุตำแหน่ง ซึ่งความคลาดเคลื่อนเล็ก ๆ ที่เกิดขึ้น ไม่ว่าจะเป็นในส่วนการสร้างแผนที่หรือการระบุตำแหน่ง อาจทำให้ส่งผลลุกลามจนทำให้การทำงานผิดพลาดได้ ดังนั้นสำหรับปัญหา SLAM จัดว่าเกี่ยวข้องกับการหาค่าเหมาะที่สุด (optimization) ที่ทำให้ค่าความคลาดเคลื่อนของการระบุตำแหน่งและสร้างแผนที่ต่ำสุด

การระบุตำแหน่งพร้อมกับการสร้างแผนที่ในสิ่งแวดล้อมขนาดใหญ่ (Large Scale SLAM) ที่นี้จะหมายถึงการระบุตำแหน่งพร้อมกับการสร้างแผนที่ในสิ่งแวดล้อมขนาดไม่จำกัด (สามารถขยายใหญ่ได้เรื่อย ๆ) โดยปัญหาท้าทายของ Large Scale SLAM มี 3 ประการคือ

- ปัญหาด้านเสถียรภาพ: เนื่องจากว่าในสิ่งแวดล้อมขนาดใหญ่นั้นเมื่อหุ่นยนต์เคลื่อนที่ไปข้างหน้าเรื่อย ๆ ความคลาดเคลื่อนของการระบุตำแหน่งและสร้างแผนที่ย่อมจะเพิ่มมากขึ้น ซึ่งอาจส่งผลทำให้ไม่สามารถหาค่าเหมาะที่สุด (Optimization) ของค่าความคลาดเคลื่อนได้
- ปัญหาประสิทธิภาพในการคำนวน: สำหรับวิธีการแก้ปัญหา SLAM โดยทั่วไปแล้ว เวลาในการคำนวนจะเป็นพหุนามกับขนาดของแผนที่ ทำให้การสร้างแผนที่ขนาดใหญ่นั้นมักจะเป็นไปได้ยากในการใช้งานแบบทันการณ์ (Real-time)
- ปัญหาการตรวจหา Loop Closure: เมื่อหุ่นยนต์เคลื่อนที่วนกลับมายังตำแหน่งเดิมที่เคยผ่านมาแล้ว การทำงานของ SLAM จำเป็นจะต้องตรวจหาการวนบรรจบของแผนที่ (Loop Closure) เพื่อปรับแก้แผนที่ให้มีความถูกต้อง ซึ่งในแผนที่ขนาดใหญ่นั้น การตรวจหา Loop Closure จะทำได้ยากเนื่องจากว่า เมื่อหุ่นยนต์เคลื่อนที่ไปเป็นระยะทางไกล หุ่นยนต์อาจมีความคลาดเคลื่อนของตำแหน่งมาก

ในงานวิจัยนี้จะนำเสนอ การระบุตำแหน่งพร้อมกับการสร้างแผนที่ในสิ่งแวดล้อมขนาดใหญ่สำหรับกล้องวิดีโอแบบมุมกว้าง ซึ่งกล้องวิดีโอแบบมุมกว้าง (Wide- Angle Camera) จะใช้แก้ปัญหามุมมองแคบของกล้องวิดีโอทั่วไป โดยกล้องวิดีโอแบบมุมกว้างนั้นอาจเป็น กล้องวิดีโอแบบออมนิ (Omni Directional Camera) หรืออาจจะเป็นกล้องวิดีโอแบบ Fish eye (Fish Eye Camera) โดยเป้าหมายของงานวิจัยเพื่อต้องการให้กล้องวิดีโอมีอิสระในการเคลื่อนที่มากขึ้นโดยที่ยังสามารถหาความสัมพันธ์ของจุดสังเกตในกล้องได้ ขณะทำการระบุตำแหน่งพร้อมกับการสร้างแผนที่ในสิ่งแวดล้อมขนาดใหญ่

1.1 งานวิจัยที่เกี่ยวข้อง

การระบุตำแหน่งพร้อมกับการสร้างแผนที่ (SLAM) นั้นสามารถทำงานได้กับอุปกรณ์วัดค่า (Sensor) หลากหลายประเภทขึ้นกับวัตถุประสงค์ในการใช้งาน ซึ่งอุปกรณ์ที่เป็นที่นิยมในการนำไปใช้งาน SLAM สามารถยกตัวอย่างได้แก่ อุปกรณ์วัดระยะด้วยเลเซอร์ (Laser Range Finder), กล้องวิดีโอ (Video Camera) และอาจจะมีอุปกรณ์เสริมอื่น ๆ ประกอบการทำงานเพื่อให้สามารถระบุตำแหน่งและสร้างแผนที่ได้แม่นยำมากขึ้น เช่น accelerometer, gyrometer, magnetic compass, GPS เป็นต้น

อุปกรณ์วัดระยะด้วยเลเซอร์ (รูปที่ 1.1 (ก)) เป็นอุปกรณ์การวัดที่ใช้หลักการสะท้อนของแสงและการวัดเวลาในการเดินทางของแสงมาคำนวนเป็นระยะทาง อุปกรณ์วัดระยะด้วยเลเซอร์เป็นอุปกรณ์ซึ่งเป็นที่นิยมกันมากสำหรับงานระบุตำแหน่งของหุ่นยนต์พร้อมกับการสร้างแผนที่ [2-4] เนื่องจากข้อมูลการวัดจากอุปกรณ์จะประกอบด้วยทิศทางและระยะทางจากอุปกรณ์ไปยังสิ่งแวดล้อม ทำให้มีความสะดวกในการสร้างแผนที่ ข้อเสียของอุปกรณ์วัดระยะด้วยเลเซอร์ก็คือ เป็นอุปกรณ์ที่มีราคาสูงและวัดค่าได้ช้า (เทียบกับกล้องวิดีโอ) นอกจากนี้การไม่มีข้อมูลภาพสีทำให้หาความสัมพันธ์ระหว่างเฟรมของข้อมูลเลเซอร์ได้ยาก ดังนั้นจึงมีหลายงานวิจัยได้พยายามนำเอากล้องวิดีโอมาใช้ร่วมกับอุปกรณ์วัดระยะด้วยเลเซอร์ [5, 6]

(ก) *(ข)* *(ค)*

(ง) *(จ)*

รูปที่ 1.1 อุปกรณ์วัดค่า (ก) อุปกรณ์วัดระยะด้วยเลเซอร์, (ข) กล้องวิดีโอ, (ค) กล้องวิดีโอแบบสเตอริโอ, (ง) กล้องวิดีโอแบบออมนิ, (จ) กล้องวิดีโอแบบ RGB-D

กล้องวิดีโอ (รูปที่ 1.1 (ข)) เป็นอุปกรณ์รับภาพจากสิ่งแวดล้อม ซึ่งมีข้อดีก็คือกล้องวิดีโอจะให้ข้อมูลของสิ่งแวดล้อมค่อนข้างมาก (ข้อมูลภาพ), การตอบสนองต่อข้อมูลสิ่งแวดล้อมรวดเร็วทันการณ์ (Real-time), ราคาถูก (เมื่อเทียบกับอุปกรณ์วัดระยะด้วยเลเซอร์) ส่วนข้อเสียของกล้องวิดีโอก็คือ กล้องวิดีโอต้องอาศัยแสงจากภายนอกช่วยในการรับภาพ จึงทำให้การเปลี่ยนแปลงสภาพแสงสภาพแวดล้อม ส่งผลกระทบต่อการทำงานของกล้องได้ง่าย นอกจากนี้ เนื่องจากกล้องวิดีโอไม่มีข้อมูลความลึก (ข้อมูลระยะจากกล้องไปยังวัตถุ) ทำให้ไม่สามารถสร้างแผนที่โดยอาศัยข้อมูลจากภาพเพียงภาพเดียวได้ (ต้องใช้หลายภาพประกอบกัน) ทำให้การนำข้อมูลมาใช้นั้นมีความยุ่งยาก

การใช้กล้องวิดีโอสำหรับการสร้างแผนที่นั้น อาศัยหลักการพื้นฐานคือ จะใช้ภาพสิ่งแวดล้อมในมุมมองที่ต่างกันเพื่อ triangulate หาระยะของสิ่งแวดล้อมและนำไปใช้ประกอบเป็นแผนที่ต่อไป แต่เนื่องจากว่ากล้องวิดีโอแบบทั่วไป มีมุมมองภาพที่แคบทำให้เป็นไปได้ยากที่จะได้ภาพสิ่งแวดล้อมเดิมในตำแหน่งที่ต่างกัน ทำให้ในการใช้งานจริง อาจต้องมีวิธีเคลื่อนกล้องแบบพิเศษ ไม่อาจเคลื่อนไปมาโดยอิสระได้ ในหลาย ๆ งานวิจัยจึงหลีกเลี่ยงปัญหาการไม่มีข้อมูลความลึกของกล้องวิดีโอด้วยการใช้กล้องแบบสเตอริโอ (Stereo Camera) [7-9] (รูปที่ 1.1 (ค)) ซึ่งข้อมูลความลึกของภาพ จะช่วยให้สามารถสร้างแผนที่และระบุตำแหน่งได้สะดวกขึ้น นอกจาก นี้ยังมีเทคนิคการใช้กล้องวิดีโอร่วมกับอุปกรณ์อื่น ๆ เช่น การใช้กล้องวิดีโอร่วมกับ IMU [10] เพื่อให้สามารถประมาณการเคลื่อนที่โดยคร่าวของกล้องได้ หรือการใช้กล้องร่วมกับ GPS และข้อมูลความสูงภูมิศาสตร์ (Digital Elevation Model (DEM)) [11] เพื่อจำกัดความคาดเคลื่อนของแผนที่ เป็นต้น

กล้องวิดีโอแบบออมนิ (รูปที่ 1.1 (ง)) เป็นอีกอุปกรณ์หนึ่งซึ่งถูกใช้ในงานระบุตำแหน่งพร้อมกับการสร้างแผนที่ [12-14] ความแตกต่างระหว่าง กล้องวิดีโอแบบออมนิและกล้องวิดีโอแบบทั่วไปก็คือ กล้องวิดีโอแบบออมนิจะให้มุมมองแบบรอบทิศทาง ลักษณะของตัวกล้องจะประกอบด้วยกล้องวิดีโอแบบทั่วไปและกระจกในการสะท้อนแสงจากสิ่งแวดล้อมเข้าไปยังอุปกรณ์รับภาพ การใช้กล้องวิดีโอแบบออมนิจะทำให้สามารถหาความสัมพันธ์ของสิ่งแวดล้อมในแต่ละเฟรมได้มาก ซึ่งจะทำให้การสร้างแผนที่มีความทนทานมากขึ้น ข้อเสียของกล้องวิดีโอแบบออมนิก็คือ เนื่องจากตัวกล้องสามารถรับภาพที่มีมุมมองกว้าง ดังนั้นภาพสิ่งแวดล้อมที่วัดได้จากกล้องจึงมีรายละเอียดลดลงเทียบกับกล้องวิดีโอแบบทั่วไปที่มีความละเอียดของภาพเท่ากัน

ในระยะหลังนี้งานระบุตำแหน่งของหุ่นยนต์พร้อมกับการสร้างแผนที่ด้วยกล้องวิดีโอแบบ RGB-D (RGB-D Camera) (รูปที่ 1.1 (จ)) กำลังได้รับความนิยมเป็นอย่างสูง [15-17] เนื่องจากว่ากล้องวิดีโอแบบ RGB-D มีราคาต่ำลงเป็นอย่างมาก การทำงานของกล้องวิดีโอแบบ RGB-D จะประกอบด้วยอุปกรณ์สามชนิด ได้แก่ IR projector มีหน้าที่ฉาย inferred pattern ลงบนสิ่งแวดล้อม, กล้อง IR (IR Camera) มีหน้าที่วัดแสง IR จากสิ่งแวดล้อมแล้วคำนวนเป็นข้อมูลความลึก และ กล้องวิดีโอทั่วไป (RGB Camera) ดังนั้นกล้องวิดีโอแบบ RGB-D จึงสามารถให้ข้อมูลเป็นภาพสี และภาพความลึกของสิ่งแวดล้อมพร้อม ๆ กัน ทำให้การใช้งานการระบุตำแหน่งของพร้อมกับการสร้างแผนที่ มีความสะดวกเนื่องจากสามารถบรรยายลักษณะสิ่งแวดล้อมในสามมิติได้เบ็จเสร็จด้วยข้อมูลในเฟรมเดียว และสามารถหาความสัมพันธ์ของข้อมูลการวัดได้ง่าย เนื่องจากมีข้อมูลภาพสีสำหรับการหาความสอดคล้องระหว่างภาพ อย่างไรก็ดีกล้องวิดีโอแบบ RGB-D ยังมีข้อจำกัดด้านระยะสูงสุดที่อุปกรณ์สามารถตรวจวัดได้ อีกทั้งข้อมูลความลึกที่วัดได้ยังมีสัญญาณรบกวนมาก และกล้องยังทำงานได้ไม่ดีในสภาพแวดล้อมนอกอาคาร เนื่องจากแสงแดดจะรบกวนการทำงานของกล้อง IR

1.1.1 **อัลกอริทึมของ** SLAM

อัลกอริทึมในการแก้ปัญหา SLAM นั้นมีมากมายหลายวิธี โดยอัลกอริทึมที่เป็นที่นิยมนั้นสามารถจำแนกได้เป็นสองประเภทด้วยกันด้วยกัน ได้แก่ Filter-based Approaches และ Optimization-based Approaches

- Filter-based Approaches จะใช้ในการแก้ปัญหา Online SLAM โดยที่สถานของระบบจะประกอบด้วยตำแหน่งของหุ่นยนต์ ณ เวลาปัจจุบันและแผนที่ การทำงานของระบบจะมีลักษณะคล้าย filter นั่นคือแต่ละช่วงเวลา ระบบจะรับข้อมูลการวัดที่ได้จากอุปกรณ์วัดค่าและคำสั่งควบคุมหุ่นยนต์ และนำข้อมูลมาปรับแก้สถานะให้มีความถูกต้องยิ่งขึ้นไปเรื่อย ๆ รวมถึงการเพิ่มสถานะใหม่ในระบบ เรียกอีกอย่างได้ว่าระบบมีการทำงานแบบ incremental smoothing [18] ตัวอย่างอัลกอริทึมที่มีลักษณะเป็น filter-based ได้แก่ Fast SLAM [19-21], EKF-SLAM [22-24], UKF SLAM[25, 26], Information-Based SLAM [27-31], H_∞ SLAM [32, 33] เป็นต้น

- Optimization-based Approaches จะทำการประมาณสถานะของหุ่นยนต์ทั้งหมดตั้งแต่เวลาเริ่มต้นจนถึงปัจจุบันและแผนที่ หรือพูดได้ว่าเป็นการประมาณเส้นทางการเคลื่อนที่ของหุ่นยนต์และแผนที่จากข้อมูลการ

วัดทั้งหมด ซึ่งก็คือการแก้ปัญหา Full SLAM นั้นเอง โดยปรกติแล้วการแก้ปัญหา Full SLAM จะอาศัยเทคนิค Least square Error Minimization มาใช้ในการประมาณสถานะที่เหมาะสมสุดที่ทำให้มีความคลาดเคลื่อนต่ำสุด ดังนั้นแนวทางนี้จึงถูกเรียกว่าวิธี Optimization-based การประมาณสถานะทั้งหมดด้วยวิธี optimization โดยอาศัยข้อมูลการวัดทั้งหมดนั้นมักจะใช้เวลานาน ดังนั้นการแก้ปัญหา Full SLAM มักจะไม่แบบทันการณ์ ตัวอย่างอัลกอริทึมได้แก่ Graph-based SLAM [34], TreeMap [35], TORO [36], iSAM [37] เป็นต้น

Fast SLAM [19-21] จะแก้ปัญหา SLAM โดยใช้ Particle Filter โดยแต่ละ particle จะเก็บค่าสถานะของแผนที่ โดยกลุ่ม particle จะถูกใช้อธิบายการกระจายความน่าจะเป็นของระบบ ทำให้สามารถประมาณการเปลี่ยนแปลงการกระจายความน่าจะเป็นได้รวดเร็ว อย่างไรก็ดีสำหรับสถานะระบบที่มีมิติสูงมาก อาจจำเป็นต้องใช้ particle จำนวนเยอะมากในการบรรยายการกระจายความน่าจะเป็น จนทำให้ไม่สามารถทำงานแบบทันการณ์ได้

EKF-SLAM [22-24] จะแก้ปัญหา SLAM โดยใช้ Extended Kalman Filter ซึ่งเป็นวิธีในการประมาณสถานะจากข้อมูลการวัดของระบบแบบ dynamic การแก้ปัญหาด้วย EKF-SLAM นั้นจะกำหนดให้แผนที่และตำแหน่งของหุ่นยนต์เป็นสถานะของระบบโดยจะอธิบายในรูป mean และ covariance ของการกระจายความน่าจะเป็นแบบ Gaussian จากนั้นเมื่อได้รับข้อมูลการวัดจากอุปกรณ์วัดค่า จะนำเอาข้อมูลการวัดมาปรับแก้สถานะของระบบ สำหรับ EKF-SLAM นั้นมักถูกนำมาใช้งานอย่างกว้างขวาง เนื่องจากมีประสิทธิภาพในการประมาณสถานะค่อนข้างดีและมีความเร็วสูงสำหรับสิ่งแวดล้อมที่มีขนาดไม่ใหญ่นัก แต่เนื่องจาก EKF-SLAM มีประสิทธิภาพเป็น $O(n^3)$ โดยที่ n เป็นขนาดของแผนที่ ทำให้ EKF-SLAM อาจจะเกิดปัญหาด้านเวลาในการคำนวนสำหรับสิ่งแวดล้อมขนาดใหญ่ได้

ในขั้นตอนการทำงานของ Extended Kalman Filter จะต้องทำ linearization โมเดลต่าง ๆ เช่นโมเดลการวัด หรือโมเดลการเคลื่อนที่ของหุ่นยนต์ ในทุก ๆ รอบการทำงาน ซึ่งสำหรับโมเดลที่มีความซับซ้อนการหา Jacobian ของระบบอาจทำได้ไม่สะดวกนัก UKF SLAM [25, 26] ถูกนำเสนอในการแก้ปัญหา linearization โดยใช้ unscented Kalman filter ในการประมาณสถานะของระบบ การทำงานของ UKF จะคล้ายคลึงกับ EKF แต่ในขั้นตอนการประมาณการกระจายความน่าจะเป็นจะใช้วิธี Sampling based แทนการ linearize โมเดลต่าง ๆ อย่างไรก็ดีทั้ง UKF และ EKF จะประมาณได้เพียงการกระจายความน่าจะเป็นแบบ Gaussian เท่านั้น ในปี 2010, H_∞ filter-based SLAM [32, 33] ได้ถูกนำเสนอ ซึ่งการใช้งาน H_∞ filter จะช่วยเพิ่มความแม่นยำและความทนทานของระบบในกรณี non-Gaussian

Information-based SLAM [30] จะมีขั้นตอนในการประมาณสถานะของระบบคล้ายคลึงกับ EKF-SLAM แต่จะอธิบายสถานะของระบบด้วย Information Vector และ Information Matrix ซึ่งเป็น dual ของ Covariance Form ซึ่งในระยะหลังนี้มีงานวิจัยปัญหา SLAM ที่อยู่ในรูป Information-based จำนวนมาก ทั้งนี้เนื่องจากข้อดีของ Information-Based ซึ่งเวลาในการประมาณและปรับแก้สถานะของ SLAM เป็น $O(m)$ เมื่อ

m เป็นขนาดของข้อมูลการวัด (measurement) อย่างไรก็ดีสถานะของระบบในรูป Information-based นั้นอยู่ในรูปที่หุ่นยนต์ไม่สามารถนำไปใช้ได้โดยตรงจึงต้องทำกระบวนการ State Recovery เพื่อแปลงสถานะให้อยู่ในรูป Covariance-based โดยเวลาที่ใช้ในการ Recovery สถานะระบบเป็น $O(n^3)$ ซึ่งเทียบเท่ากับเวลาในการทำงานของ EKF-SLAM อย่างไรก็ดี Thrun et al. [27] และ Hirzinger [29] ได้พบว่า สำหรับ Feature-based SLAM นั้น ค่าส่วนใหญ่ใน Information Matrix มีค่าเข้าใกล้ศูนย์และได้นำเสนอการแก้ปัญหา SLAM ด้วย Sparse Extended Information Filter (SEIF) นอกจากนี้สำหรับ View-based SLAM, Eustice [28] ได้พบว่า Information Matrix จะอยู่ในรูป Sparse Matrix พอดีจึงทำให้สามารถแก้ปัญหาได้ด้วย Iterative Conjugate Gradients (CG) ซึ่งใช้เวลาในการทำงานรวมเป็น $O(n^2)$ และถ้าใช้วิธี Partial State Recovery ก็จะสามารถลดเวลาในการคำนวนลงไปได้อีก ในปี 2011 X. Wang และ L. Ma [31] ได้นำเสนอวิธีแก้ปัญหา Feature-based SLAM ด้วย Sparse Extended Information Filter โดยใช้ minimal connected dominating set (CDS) ในการลดจำนวนความสัมพันธ์ของ Feature ในระหว่างที่แผนที่กำลังโต เพื่อสร้าง Information Matrix แบบ sparse ทำให้สามารถลดเวลาในการคำนวนขั้นตอนการ Recovery สถานะระบบด้วย sparse matrix solver

อีกแนวทางหนึ่งในการแก้ปัญหา SLAM ก็คือวิธีแบบ Optimization-based ซึ่งจะมองปัญหาการระบุตำแหน่งและสร้างแผนที่เป็นการแก้ปัญหาการหาค่าเหมาะสุด หลักการทำงานนั้นจะบรรยายสถานะของระบบอันประกอบด้วยเส้นทางการเคลื่อนที่ของหุ่นยนต์และแผนที่ในรูปแบบของโหนดในกราฟ โดยมี constraint เป็นเส้นเชื่อมระหว่างโหนด [34, 38] จากนั้นจึงหาค่าเหมาะสุดซึ่งเป็นคำตอบของ non-linear maximum likelihood ต่อไป การหาคำตอบของ Graph-based SLAM นั้นได้มีงานวิจัยซึ่งนำเสนอวิธีในการปรับปรุงประสิทธิภาพไว้เป็นจำนวนมาก ยกตัวอย่างเช่น การหาค่าเหมาะสุดบน Manifolds [39], การการหาค่าเหมาะสุดแบบต้นไม้ลำดับขั้น [40], การใช้ Stochastic Gradient Descent [36, 41] เป็นต้น การแก้ปัญหา Graph-based SLAM จะเป็นการประมาณคำตอบแบบ Full SLAM นั้นคือการประมาณหาสถานะของระบบทั้งหมดตั้งแต่เวลาเริ่มต้นทำให้ใช้เวลาในการประมวลผลนานไม่สามารถทำงานแบบทันการณ์ได้ จึงมักในการแก้ปัญหาแบบ off-line มากกว่า

ในระยะหลังนี้เริ่มมีการใช้วิธีแบบ Optimization-based ในการแก้ปัญหา Visual SLAM เป็นจำนวนมาก [42, 43] ทั้งนี้เนื่องจากวิธีแบบ Optimization-based นั้นมีความแม่นยำกว่าแบบ Filter-based การแก้ปัญหา Visual SLAM ด้วยวิธีแบบ Optimization-based จะถูกเรียกว่า Bundle Adjustment [44] ซึ่งแต่เดิมนั้นเป็นวิธีทาง Computer Vision ซึ่งใช้ในการแก้ปัญหาที่คล้ายคลึงกับ SLAM เรียกว่า “Structure from Motion (SFM)” ปัญหา SFM คือการประมาณโครงสร้างสิ่งแวดล้อมในสามมิติ โดยอาศัยข้อมูลภาพในหลาย ๆ มุมมองจำนวนมาก ความต่างระหว่าง SFM และ SLAM ก็คือ SFM นั้นจะต้องทำการเก็บรวบรวมข้อมูลภาพทุกมุมมองมาจนครบเสียก่อน จากนั้นจึงทำการคำนวนโครงสร้างสิ่งแวดล้อมและตำแหน่งของกล้องเพียงทั้งหมดพร้อม ๆ กัน เพื่อหาค่าเหมาะสุดของทั้งตำแหน่งของกล้องและแผนที่ ส่วน SLAM นั้นจะทำการระบุตำแหน่งไปพร้อมกับการสร้างแผนที่แบบทันทีในระหว่างที่หุ่นยนต์ทำงาน และแผนที่จะถูกปรับแก้เพิ่มเติมขึ้นเรื่อย ๆ

1.1.2 Monocular Vision-based SLAM

กล้องวิดีโอ (Video Camera) ถือเป็นอุปกรณ์วัดค่าที่เป็นที่นิยมใช้ในงาน SLAM มากที่สุดชนิดหนึ่ง เนื่องจากมีราคาถูก ใช้งานง่าย ให้ข้อมูลเป็นจำนวนมาก อีกทั้งมีความเร็วในการรับข้อมูลสูง (Real-time) แต่ปัญหาใหญ่สำหรับการใช้งานกล้องวิดีโอ คือข้อมูลที่ได้มาจากกล้องนั้นเป็นภาพในสองมิติ ดังนั้นการสร้างแผนที่สำหรับสิ่งแวดล้อมซึ่งต้องอาศัยข้อมูลระยะร่วมด้วยนั้นจึงมีความยุ่งยาก ในงานวิจัยหลาย ๆ งานจึงมีเทคนิคในการช่วยหาข้อมูลระยะ เช่น การใช้กล้องสเตอริโอ (Stereo Camera) [7-9], การใช้กล้องวิดีโอ ร่วมกับเซนเซอร์วัดระยะอื่นเช่นอุปกรณ์วัดระยะด้วยเลเซอร์ (Laser Range Finder) [2-4] อุปกรณ์ตรวจจับการเคลื่อนที่จากความเฉื่อย (Inertia Sensors)[10] หรืออุปกรณ์ตรวจวัดความเร็วจากล้อ (Wheel Encoders), แต่ถึงกระนั้นก็ยังมีงานวิจัยที่อาศัยข้อมูลรูปภาพจากกล้องเพียงอย่างเดียวในการระบุตำแหน่งพร้อมกับการสร้างแผนที่

งานระบุตำแหน่งพร้อมกับการสร้างแผนที่ โดยอาศัยกล้องวิดีโอเพียงตัวเดียวที่ประสบความสำเร็จงานแรก ๆ ถูกนำเสนอโดย Davison (MonoSLAM [45]) และ Eade และ Drummond [46] โดยในงานของ Davison นั้นได้ใช้วิธี EKF-SLAM ส่วน Eade และ Drummond ใช้วิธี Fast SLAM 2.0 [20] ในการระบุตำแหน่งพร้อมกับการสร้างแผนที่ ทั้งสองงานได้อธิบายแผนที่ในรูป Feature-based และสามารถระบุตำแหน่งของกล้องพร้อมกับการสร้างแผนที่ได้แบบทันการณ์ หลังจากนั้นจึงได้มีงานวิจัยการแก้ปัญหา Monocular SLAM แบบ filter-based เป็นจำนวนมาก ไม่ว่าจะเป็นการใช้ information filter ในการแก้ปัญหา [47] หรือการประยุกต์ใช้ Particle filter บน Lie Groups [48] เป็นต้น อย่างไรก็ดีข้อจำกัดของงานข้างต้นคือ อัลกอริทึมไม่สามารถทำงานในสิ่งแวดล้อมขนาดใหญ่ได้ เนื่องจากมีผลกระทบกับประสิทธิภาพเวลาการทำงาน

ในปี 2008, Klein และ Murray ได้นำเสนอ PTAM [49] ซึ่งเป็นวิธีในการติดตามตำแหน่งของกล้อง (Tracking) และสร้างแผนที่ (Mapping) โดยใช้ Local Bundle Adjustment ซึ่งการใช้ Bundle Adjustment นั้นถึงแม้จะสิ้นเปลืองประสิทธิภาพในการคำนวน แต่ก็ได้อัลกอริทึมที่มีความทนทานและได้คำตอบที่มีความแม่นยำสูง และเนื่องจากเป็นการทำ Bundle Adjustment แบบเฉพาะส่วนบนแผนที่ ทำให้ยังคงประสิทธิภาพแบบทันการณ์ได้ อย่างไรก็ดีในงาน PTAM ได้เน้นเฉพาะในการติดตามตำแหน่งของกล้อง (Tracking) เพื่อใช้ในงาน AR (Augmented Reality) เท่านั้น การสร้างแผนที่จึงไม่สมบูรณ์เพราะไม่มีการ Close Loop อย่างไรก็ดีก็มีงานวิจัยในภายหลัง อาศัยวิธีการที่คล้ายคลึงกับ PTAM ก็คือการนำเอา Visual Odometry มาใช้ร่วมกับ SLAM [50, 51] หรือการแก้ปัญหาแบบ Optimization-based สำหรับแผนที่ขนาดใหญ่ [43]

ในระยะหลังการใช้กล้องวิดีโอเพียงตัวเดียวในการสร้างแผนที่แบบหนาแน่น (Dense Mapping) เริ่มเป็นที่นิยม โดยในปี 2010 Newcombe และ Davison [52] ได้นำเสนอวิธีสร้างแผนที่แบบ Real-Time จากจุด Feature จำนวนมาก หลังจากนั้น New combe et al. ก็นำเสนอวิธีการติดตามตำแหน่งของกล้องและสร้างแผนที่แบบหนาแน่น (Dense Tracking and Mapping: DTAM [53]) ซึ่งอาศัยข้อมูลภาพเพียงอย่างเดียวในปี 2011 นอกจากนี้ Graber et al. [54] และ Pizzoli et al. [55] ได้นำเสนอการสร้างแผนที่แบบหนาแน่น (Dense Mapping) โดยใช้วิธีที่คล้ายคลึงกัน

การสร้างแผนที่แบบหนาแน่น (Dense Mapping) สามารถเรียกได้อีกชื่อว่า Direct Method เนื่องจากเป็นการสร้างแผนที่จากข้อมูลภาพโดยตรง เมื่อเทียบกับ Feature-Based Methods ซึ่งต้องทำการตรวจหาตำแหน่งจุดสังเกตในภาพเสียก่อน ข้อดีของ Dense Tracking and Mapping ก็คือการใช้ข้อมูลทั้งหมดของภาพในการการติดตามตำแหน่งของกล้องและสร้างแผนที่ แทนการใช้ข้อมูลจากจุดสังเกตเพียงบางจุด ทำให้แผนที่และตำแหน่งของกล้องมีความแม่นยำมากกว่า แต่ข้อเสียก็คือการสร้างแผนที่แบบหนาแน่นจะใช้เวลาในการคำนวนมาก ไม่สามารถทำงานในสิ่งแวดล้อมขนาดใหญ่ได้ และแผนที่ยังปรับแก้ได้ยาก

ข้อจำกัดของ Monocular SLAM ก็คือในการสร้างแผนที่ต้องอาศัยการเลื่อนตำแหน่งของกล้องเพื่อหามุม parallax ซึ่งใช้สำหรับการสร้างตำแหน่งจุดสังเกตในสามมิติ แต่ถ้าหากว่ากล้องมีการหมุนเพียงอย่างเดียวก็จะไม่เกิดมุม parallax ทำให้ไม่สามารถประมาณตำแหน่งของจุดสังเกตได้ และทำให้การระบุตำแหน่งของหุ่นยนต์ในขั้นตอนถัดไปล้มเหลว ในปี 2011 Zhao et al. [42] ได้เสนอวิธีในการอธิบายแผนที่รูปแบบใหม่สำหรับ Monocular SLAM โดยบรรยายตำแหน่งจุดสังเกตด้วย Parallax Angle แทนการบรรยายตำแหน่งใน Euclidian space แบบปรกติทำให้สามารถบรรยายสถานะของจุดสังเกตแม้ไม่สามารถคำนวนหาตำแหน่งได้ และในปี 2012 Gauglitz et al. [56] ได้แบ่ง keyframe ออกเป็นสองโหมดคือ แบบการหมุนเพียงอย่างเดียวกับแบบมีการเลื่อนตำแหน่งร่วมด้วย จากนั้นจะใช้ constraint ที่ต่างกันในการบรรยายความเชื่อมโยงของเฟรมในแต่ละโหมด ในปี 2013 Pirchheim et al. [57] ได้เสนอ infinite depth feature ซึ่งใช้บรรยายจุดสังเกตขณะที่เกิดการหมุนเพียงอย่างเดียว ทำให้สามารถระบุตำแหน่งและสร้างแผนที่ในขณะที่กล้องหมุนเพียงอย่างเดียวได้

ปัญหาสำคัญอีกประการสำหรับ Monocular SLAM ก็คือปัญหา Scale Drift ทั้งนี้เนื่องจากข้อมูลการวัดจากกล้องเพียงตัวเดียวไม่มีข้อมูลความลึกร่วมด้วย ทำให้ระบบไม่สามารถประมาณขนาดของแผนที่ที่ถูกต้องได้ และเมื่ออัลกอริทึมทำงานไปเรื่อย ๆ ความคลาดเคลื่อนของระบบสูงขึ้น ก็จะทำให้ขนาดของแผนที่มีการเบี่ยงเบน การแก้ปัญหา Scale Drift จะต้องอาศัยข้อมูลสิ่งแวดล้อมเพิ่มเติมให้สามารถประมาณขนาดของแผนที่ได้อย่างถูกต้อง ซึ่งมีงานวิจัยจำนวนมากนำเสนอวิธีการแก้ปัญหา SLAM โดยอาศัยข้อมูล semantic ร่วม ซึ่งส่วนมากจะใช้วิธีการรู้จำวัตถุในสิ่งแวดล้อมที่รู้ขนาดแน่นอน [58, 59] นอกจากนี้ยังมีจากวิจัยที่วิธีการแก้ปัญหา SLAM โดยใช้วัตถุต่าง ๆ ในสิ่งแวดล้อมเป็นจุดสังเกตโดยตรง [60] อย่างไรก็ดี การที่จะต้องเรียนรู้ Object Database ขนาดใหญ่สำหรับทำ Object Recognition ถือเป็นขั้นตอนที่ยุ่งยาก และไม่เหมาะกับสภาพแวดล้อมทั่วไป

1.1.3 Large Scale SLAM

ปัญหาใหญ่ของการระบุตำแหน่งพร้อมกับการสร้างแผนที่ในสิ่งแวดล้อมขนาดใหญ่ ก็คือปัญหาด้านเวลาในการคำนวนเนื่องจากการทำงานของ SLAM ในหนึ่งรอบจะใช้ค่าการวัดที่ได้จากอุปกรณ์วัดค่ามาปรับปรุงการกระจายความน่าจะเป็นของแผนที่ทั้งหมด ทำให้เมื่อแผนที่มีขนาดใหญ่ เวลาในการคำนวนก็มากขึ้นด้วย (ประสิทธิภาพการทำงานของอัลกอริทึม SLAM ทั่วไปจะเป็น $O(n^2)$ ต่อหนึ่ง time-step เมื่อ n เป็นขนาดของแผนที่) นอกจากนี้ยังมีปัญหาเรื่องความแม่นยำของการสร้างแผนที่ขนาดใหญ่ เนื่องจากว่าเมื่อแผนที่มีขนาดใหญ่ขึ้น ความคลาดเคลื่อนในการประมาณก็จะเพิ่มขึ้นเป็นเงาตามตัว

การแก้ปัญหาเวลาการคำนวนของการสร้างแผนที่ในสิ่งแวดล้อมขนาดใหญ่ สามารถทำได้ง่ายด้วยการแบ่งแผนที่ออกเป็น Submap ย่อย ๆ หลายชิ้นโดยเรียกว่า Local Map การระบุตำแหน่งพร้อมกับสร้างแผนที่ในแผนที่ย่อยเฉพาะส่วน จะใช้เวลาไม่เกินค่าคงที่ค่าหนึ่งเนื่องจากมีการจำกัดขนาดของ sub map ไว้ จากนั้นจึงนำเสนอวิธีในการเชื่อม Submap หลาย ๆ ส่วนเข้าด้วยกัน ยกตัวอย่างเช่นในงานวิจัยยุคเริ่มแรก Hierarchical SLAM [61] ได้เสนอการสร้างแผนที่สองลำดับขั้นได้แก่ Local Map และ Global Map การทำ SLAM ในแผนที่ย่อยจะสามารถรับประกันเวลาในการคำนวนได้ แต่เวลาที่ใช้ในการร่วมแผนที่ก็จะมากขึ้นตามจำนวน Submap อยู่ดี ถึงแม้งานวิจัยนี้สามารถลดเวลาในการคำนวนได้แต่ประสิทธิภาพของอัลกอริทึมยังคงเป็น $O(n^2)$

ในปี 2008 Paz et al. ได้นำเสนอ Divide and Conquer: EKF SLAM [62] ซึ่งจะสร้างแผนที่ย่อยจำนวนมาก (คล้าย Hierarchical SLAM) จากนั้นในขั้นตอนการรวมแผนที่จะใช้การร่วมแบบ Hierarchy โดยจะรวมแผนที่ย่อยที่อยู่ใกล้กันให้เป็นแผนที่ที่ใหญ่ขึ้น จากนั้นก็วนซ้ำเพื่อรวมแผนที่ให้ใหญ่ขึ้นเรื่อย ๆ จนกระทั้งได้แผนที่ใหญ่สุดเพียงอันเดียวทำให้เวลาในการคำนวนของ Divide and Conquer SLAM นั้นจะเป็น $O(n)$ ต่อหนึ่ง time-step ข้อเสียของการสร้างแผนที่แบบ Submap ก็คือความแม่นยำในการสร้างแผนที่จะลดลง ไม่เทียบเท่าการปรับแก้แผนที่แบบพร้อมกันทั้งหมด เนื่องจาก Submap แต่ในละส่วนนั้นไม่ขึ้นต่อกัน และในปีเดียวกัน, Piniés และ Tardós [63] ได้นำเสนอ Large Scale SLAM โดยใช้ Conditionally Independent Local Maps ซึ่งแนวคิดของงานวิจัยนี้จะคล้ายคลึงกับงานวิจัยอื่น ๆ คือแบ่งแผนที่เป็นแผนที่ย่อยเพื่อประหยัดเวลาในการคำนวน แต่ความต่างของงานวิจัยนี้เทียบกับงาน Hierarchical SLAM และ Divide and Conquer SLAM ก็คืองานวิจัยนี้จะสร้างแผนที่ย่อยแบบ Conditionally Independent นั่นคือระหว่าง Local Map จะไม่ขึ้นต่อกันอย่างมีเงื่อนไข ทำให้แผนที่ร่วมมีความถูกต้องมากขึ้น

การแก้ปัญหาการสร้างแผนที่ขนาดใหญ่ด้วย Submap ยังเป็นที่นิยมใช้อย่างแพร่หลาย โดยในปีหลัง ๆ ก็มีการประยุกต์ใช้ Submap ในหลาย ๆ งานยกตัวอย่างเช่น การใช้ Sparse Local Submap Joining Filter (SLSJF) [64] ในการรวม Submap เข้าด้วยกันซึ่งจะเพิ่มประสิทธิภาพในการคำนวน, การปรับปรุงความแม่นยำของ Submap ด้วย Local Bundle Adjustment [65], การตรวจหา Submap เพื่อการ close loop ด้วย spectral registration [66] ซึ่งทนทานต่อการเปลี่ยนแปลงของสภาพแวดล้อม เป็นต้น

อีกหนึ่งวิธีในการแก้ปัญหาแผนที่ขนาดใหญ่ ก็คือการอธิบายแผนที่ด้วย TreeMap โดยในปี 2006, Udo Frese [35] ได้นำเสนอแนวคิด ในการการแบ่งแผนที่ให้อยู่ในลักษณะของต้นไม้ ซึ่งโหนดล่างสุดก็จะเป็นแผนที่แบบละเอียดสุด และในโหนดลำดับถัดมาก็จะเป็นแผนที่ที่มีความหยาบขึ้น โดยโหนดพ่อก็จะเก็บข้อมูลที่จำเป็นของโหนดลูกไว้ และขั้นตอนในการปรับแก้แผนที่ก็จะใช้การ propagate ข้อมูลภายในต้นไม้ซึ่งรับประกันเวลาในการคำนวนเป็น $O(\log(n))$ อย่างไรก็ดี TreeMap นั้นมีความซับซ้อน ในการ implement มาก เพราะนอกเหนือจากการ propagate ข้อมูลในต้นไม้แล้ว ยังจะต้องจัดการ balance tree ให้มีความสมดุลทำให้สิ้นเปลืองเวลาในการคำนวน นอกจากนี้ TreeMap ยังมีปัญหาในแง่ของความเหมาะสมในการแบ่งต้นไม้

ในขณะที่งานวิจัยจำนวนหนึ่งได้มุ่งเน้นเพื่อแก้ปัญหาเวลาในการทำงานของการสร้างแผนที่ขนาดใหญ่ งานวิจัยอีกกลุ่มหนึ่งก็ได้พยายามนำเสนอวิธีในการแก้ปัญหาความแม่นยำในการสร้างแผนที่ขนาดใหญ่ ซึ่งจะเน้นการใช้อัลกอริทึมแบบ Optimization-based แทนอัลกอริทึมแบบ Filter-based เนื่องจากมีความแม่นยำในการประมาณค่าที่สูงกว่า เช่นในปี 2011, Beall et al. [67] ได้นำเสนอวิธีใช้งาน Bundle Adjustment ที่มีประสิทธิภาพสำหรับการสร้างแผนที่ขนาดใหญ่ของงานสำรวจใต้น้ำ หรือในปี 2013, Blanco et al. [68] ได้นำเสนอการใช้งาน Bundle Adjustment ร่วมกับการอธิบายแผนที่แบบ relative ซึ่งนอกจากจะช่วยเพิ่มความแม่นยำในการประมาณแล้ว ยังสามารถลดเวลาในการทำงานลงจนเหลือเวลาคงที่ได้

นอกจากนี้ยังมีงานวิจัยจำนวนหนึ่งมองปัญหาการสร้างแผนที่ขนาดใหญ่เป็นปัญหา Pose Graph Optimization ซึ่งจะอธิบายแผนที่ด้วยความสัมพันธ์ของมุมมองของหุ่นยนต์ในอดีต โดยแต่ละโหนดในกราฟจะแทนตำแหน่งของหุ่นยนต์ และเส้นเชื่อมระหว่างโหนดจะเป็นความสัมพันธ์ของตำแหน่ง การแก้ปัญหา Pose Graph จะเป็นการหาค่าเหมาะสุด (Optimization) ซึ่งทำให้ความคลาดเคลื่อนที่แต่ละความสัมพันธ์ต่ำสุด ตัวอย่างงานวิจัยในลักษณะนี้ได้แก่ HOG-Man [39], TORO [36], g^2o [69] ซึ่งนำเสนอ framework ทั่วไปในการแก้ปัญหา Graph Optimization ด้วยวิธีการประมาณค่าตอบของ non-linear maximum likelihood ในแบบต่าง ๆ ซึ่งจะมีเสถียรภาพและความแม่นยำมากกว่าวิธีแบบ Filter-based

iSAM (Incremental Smoothing and Mapping) [18] ถูกเสนอขึ้นโดย Kaess et al. ในปี 2008 โดยมีแนวคิดในการใช้ QR Factorization ของ Information Matrix เพื่ออธิบายแผนที่ การปรับแก้แผนที่ด้วย QR factorization จะมีประสิทธิภาพดีกว่าการปรับแก้บน Information Matrix โดยตรง เพราะขั้นตอน State Recovery นั้นสิ้นเปลืองเวลาในการทำงานมาก หลังจากนั้น iSAM2 (Incremental Smoothing and Mapping version 2) [37] ก็ถูกนำเสนอโดยมีการใช้ Bayes tree ร่วมในการอธิบายแผนที่ ทำให้ลดความซ้ำซ้อนและขั้นตอนที่ไม่จำเป็นของการคำนวนลงไปได้ นอกจากนี้คำตอบที่ได้ยังเป็น Exact Solution (เทียบเท่าการปรับแก้แผนที่พร้อมกันทั้งหมด)

การแก้ปัญหาการสร้างแผนที่ขนาดใหญ่ด้วยวิธี Pose Graph Optimization มีข้อดีในด้านเสถียรภาพและความแม่นยำในการทำงาน แต่ก็มีปัญหาด้านเวลาการทำงาน เพราะการอธิบายแผนที่แบบ pose graph จะทำให้ขนาดของแผนที่โตตามเวลาแม้แผนที่จะไม่ได้กินบริเวณกว้างขึ้น เนื่องจากจำนวน pose ในแผนที่เพิ่มขึ้นเรื่อย ๆ ในการแก้ปัญหานี้ได้มีงานวิจัยที่นำเสนอวิธีในการลด pose keyframe ที่มีความซ้ำซ้อนหรือมีความจำเป็นน้อยเพื่อลดขนาดของแผนที่ เช่น “Information-Based Compact Pose SLAM” [40] หรือ “Reduced State Representation in Delayed-State SLAM” [70] เป็นต้น

งานวิจัยส่วนใหญ่สำหรับปัญหา Large Scale SLAM จะนำเสนอในรูปของทฤษฎีซึ่งสามารถนำไปประยุกต์กับอุปกรณ์ต่าง ๆ ทั้งนี้ก็ยังมีการนำเสนอ Large Scale SLAM สำหรับการใช้งานจริง ไม่ว่าจะเป็นการใช้งานกับกล้องวิดีโอเพียงอย่างเดียว [43, 71] การใช้อุปกรณ์วัดระยะด้วยเลเซอร์เพียงอย่างเดียว [72], การใช้

อุปกรณ์วัดระยะด้วยเลเซอร์ร่วมกับกล้องวิดีโอสำหรับการสร้างแผนที่แบบ Outdoor [73], การใช้กล้องสเตอริโอ [74, 75] ซึ่งในแต่ละงานจะมีความเหมาะสมสำหรับแต่ละสถานการณ์แตกต่างกันไป

1.1.4 **การตรวจหา** Loop Closure

ปัญหาสำคัญอีกประการของการระบุตำแหน่งพร้อมกับการสร้างแผนที่ในสิ่งแวดล้อมขนาดใหญ่ ก็คือ ปัญหาการตรวจหา Loop Closure ซึ่งหมายความถึงการตรวจวัดว่าหุ่นยนต์ได้วิ่งวนซ้ำมายังตำแหน่งเดิมหรือไม่ สำหรับ Large Scale SLAM การตรวจหา Loop Closure ถือเป็นความยุ่งยากทั้งนี้เนื่องจากว่า เมื่อหุ่นยนต์เคลื่อนที่ในสิ่งแวดล้อมเป็นระยะทางไกล ความไม่แน่นอนของตำแหน่งหุ่นยนต์เทียบกับตำแหน่งตอนเริ่มต้นย่อมมีความคลาดเคลื่อนมาก ทำให้ไม่สามารถหาความสัมพันธ์ของ Feature ได้ด้วยวิธีการทางความน่าจะเป็นแบบปรกติ ต้องอาศัยวิธีพิเศษในการตรวจหา

ในปี 2009, Williams et al. [76] ได้จัดกลุ่มวิธีการตรวจหา Loop closure ออกเป็นสามกลุ่มได้แก่

- map-to-map เป็นการหาความสัมพันธ์โดยพิจารณาจากแผนที่เฉพาะส่วน (Submap) สองส่วน [66, 71]
- image-to-map เป็นการหาความสัมพันธ์ระหว่างภาพและแผนที่เฉพาะส่วน (Submap) [77-79]
- image-to-image เป็นการหาความสัมพันธ์ระหว่างคู่ภาพโดยตรง [80]

การตรวจหา Loop closure แบบ map-to-map จะเป็นตรวจหาความสัมพันธ์ระหว่าง Submap ล่าสุดกับ Submap ก่อนหน้าเพื่อหาโอกาสที่จะเกิด Loop closure โดนงานวิจัยของ Clemente et al. [71] ได้เสนอวิธีการใช้ Geometric Constraints Branch and Bound (GCBB) ในการตรวจหา Submap ที่มีความสัมพันธ์กัน ซึ่งเป็นการใช้ข้อมูลโครงสร้างของแผนที่ในการตรวจวัด หรือในงานวิจัยของ Oberländer et al. [66] ได้เสนอการตรวจหาความสัมพันธ์ โดยใช้ Spectral Submap Matching ซึ่งเป็นการเปรียบเทียบ 2D signals ของทั้งสอง Submap ซึ่งหาได้จาก Fourier-Mellin Transform (FMT) ของ gridmap ข้อดีของการตรวจหา Loop closure แบบ map-to-map ก็คือมีความผิดพลาดในการตรวจหาต่ำ เนื่องจากว่ามีการใช้ข้อมูลโครงสร้างแผนที่ซึ่งมีลักษณะเฉพาะในการพิจารณา แต่ข้อเสียก็คือการทำงานจะเหมาะกับแผนที่แบบ Submap และการบรรยายแผนที่บางประเภทเท่านั้น

การตรวจหา Loop closure แบบ image-to-map จะเป็นการหาความสัมพันธ์ระหว่างภาพปัจจุบันกับแผนที่ในอดีต โดยในการจับคู่ความสัมพันธ์ระหว่าง feature ในภาพและจุดสังเกตในแผนที่อาจใช้ Feature Descriptor [79, 81] หรือ RANSAC [77, 78] ในการค้นหาแต่หัวใจสำคัญของวิธีแบบ image-to-map ก็คือการทวนสอบความสัมพันธ์ด้วยข้อมูลเรขาคณิตของแผนที่ โดยกระบวนการจะทำการ back project จุดสังเกตในแผนที่ลงในภาพสองมิติเพื่อวัดความคลาดเคลื่อน วิธีแบบ image-to-map จึงมีความผิดพลาดในการตรวจหาต่ำใกล้เคียงกับแบบ map-to-map และมีข้อดีด้านการประยุกต์ใช้กับแผนที่ได้หลายประเภท และความสะดวกในการตรวจหา Loop closure เพราะเป็นการพิจารณาโดยอาศัยข้อมูลภาพล่าสุดเพียงภาพเดียว แต่ก็มีข้อเสียด้านเวลาในการทำงานโดยอาจจะขยายให้ใช้กับแผนที่ขนาดใหญ่ได้ยาก

การตรวจหา Loop closure แบบ image-to-image เป็นการหาความสัมพันธ์ระหว่างภาพ ณ ปัจจุบันเทียบกับภาพต่าง ๆ ในอดีต ซึ่งวิธีแบบ image-to-image นั้น เป็นวิธีการที่ได้รับความนิยมในการใช้งานมากสุด ทั้งนี้เนื่องจากว่าอัลกอริทึมสามารถทำงานได้อย่างอิสระโดยไม่ขึ้นกระบวนการระบุตำแหน่งและสร้างแผนที่ ทำให้สามารถควบคุมประสิทธิภาพในการทำงานได้ง่าย การทำงานการตรวจหา Loop closure มักจะนิยมใช้วิธี Bag-of-word (BoW) [82, 83] ในการสร้างกลุ่มของ visual words จาก Feature Descriptor เพื่อใช้ในการเปรียบเทียบคู่ภาพ โดยวิธีการเปรียบเทียบภาพ ณ เวลาปัจจุบัน กับภาพในอดีตทั้งหมดให้ได้ในเวลาอันรวดเร็วนั้น ในแต่ละงานวิจัยก็ได้เสนอเทคนิคที่แตกต่างกันออกไป

อัลกอริทึม FAB-MAP ซึ่งถูกในเสนอโดย Cummins และ Newman [84] เป็นอัลกอริทึมแรก ๆ ที่ประสบความสำเร็จอย่างมากในการทำ Appearance-based SLAM ซึ่งเป็นวิธีการระบุตำแหน่งของหุ่นยนต์โดยใช้การอธิบายความสัมพันธ์ของภาพในอดีตเพียงอย่างเดียว อัลกอริทึม FAB-MAP ได้เสนอวิธีการใช้ Chow-Liu tree ในการบรรยายความสัมพันธ์ร่วมของ visual words จากนั้นจึงนำเสนอวิธีการประมาณ likelihood ของแต่ละภาพจาก recursive Bayes estimation หลังจากนั้นในปี 2010 เขาก็ได้นำเสนอ FAB-MAP 2.0 [80] ซึ่งใช้ randomized kd-forest ในการเพิ่มความเร็วในการจัดกลุ่มสำหรับขั้นตอนการสร้าง visual words สำหรับแผนที่ขนาดใหญ่

การทำงานของอัลกอริทึม FAB-MAP นั้นจะใช้ SURF [85] ในการตรวจหา Features และ Feature Descriptors ซึ่งใช้เวลาในการทำงานมาก ดังนั้น Gálvez-Lopez และ Tárdos [86] จึงได้นำเสนอการใช้ FAST feature detector [87] ในการตรวจหา Feature และการใช้ BRIEF [88] สำหรับตรวจหา Feature Descriptors เพื่อลดเวลาในการทำงาน และในปี 2014, Mur-Artal และ Tárdos [89] ได้นำเสนอการใช้ ORB features ร่วมกับ Bag-of-word เพื่อลดเวลาในการทำงานเช่นกัน นอกจากนี้ยังได้นำเสนอวิธีในการ relocalization ตำแหน่งของกล้องสำหรับ keyframe-based SLAM อีกด้วย นอกจากนี้ยังมีอัลกอริทึมอื่น ๆ ที่พยายามเพิ่มประสิทธิภาพการทำงานโดยการนำเสนอ data structures แบบต่าง ๆ เช่นการใช้ locality sensitive hashing [90] หรือ kd-tree [80] ปัญหาของการวิธี Bag-of-word ก็คือก่อนเริ่มใช้งานต้องมีขั้นตอนในการสร้าง visual words ก่อนเสมอ โดยต้องอาศัย training data ซึ่งเป็นชุดภาพที่มีลักษณะคล้ายคลึงกับสิ่งแวดล้อมที่ต้องการใช้งาน ดังนั้นจึงมีหลายงานวิจัยนำเสนอวิธีในการสร้าง visual words แบบ online [91, 92] แต่ยังไม่เป็นที่นิยมนัก

อีกวิธีหนึ่งในการตรวจหา Loop closure แบบ image-to-image จะเป็นการสร้าง descriptor จากภาพโดยตรงโดยไม่ต้องตรวจหา feature ในภาพหรือการจัดกลุ่ม visual words ยกตัวอย่างเช่นการสร้าง compact image descriptor ซึ่งเสนอโดย Liu และ Zhang [93] หรือการสร้าง down-sampled binarized image (Wu, Zhang และ Guan [94]) เพื่อให้การเปรียบเทียบระหว่างคู่ภาพสามารถทำได้อย่างรวดเร็ว

วิธีการตรวจหา Loop closure แบบ image-to-image นั้นมีประสิทธิภาพสูงทำงานได้อย่างรวดเร็วแต่ก็มีโอกาสที่จะตรวจพบความผิดพลาดบ้าง (False Positive) ดังนั้นในการใช้งานจริงจึงต้องทีการทวนซ้ำคำตอบด้วยข้อมูลเรขาคณิตของแผนที่

1.2 สรุปขอบเขตปัญหา

สำหรับวิทยานิพนธ์ฉบับนี้ มุ่งเน้นที่จะเสนอวิธีการในการแก้ปัญหาการระบุตำแหน่งพร้อมกับการสร้างแผนที่ในสิ่งแวดล้อมขนาดใหญ่สำหรับกล้องวิดีโอแบบมุมกว้าง โดยอาศัยข้อดีของกล้องที่มีมุมมองที่กว้างกว่าปรกติทำให้สามารถหาความสัมพันธ์ระหว่างภาพได้ง่ายยิ่งขึ้น ในการทดลองนั้นจะมุ่งเน้นให้กล้องวิดีโอสามารถเคลื่อนที่ได้อย่างอิสระ โดยจะใช้มนุษย์ในการเดินถือกล้องเคลื่อนที่ นอกจากนี้อัลกอริทึมที่จะต้องทำงานได้ทันการณ์และมีเสถียรภาพในการทำงาน

เป้าหมายวิทยานิพนธ์สามารถสรุปได้ดังนี้

- ใช้กล้องวิดีโอแบบมุมกว้างเพื่อแก้ปัญหาข้อจำกัดเรื่องการเคลื่อนที่ของกล้องวิดีโอแบบทั่วไป
- ระบุตำแหน่งของกล้องวิดีโอพร้อมกับการสร้างแผนที่ในสิ่งแวดล้อมขนาดใหญ่ในสามมิติ โดยจะเน้นให้มีเสถียรภาพในการทำงาน)Robust)
- กล้องวิดีโอสามารถเคลื่อนที่ได้อย่างอิสระ โดยกล้องวิดีโอจะถูกถือและเคลื่อนที่โดยมนุษย์
- การทำงานของอัลกอริทึมจะเป็นแบบทันการณ์

1.3 ลำดับเนื้อหาวิทยานิพนธ์

ในบทที่ 2 จะนำเสนอทฤษฎีและหลักการทั่วไปที่ใช้ในการแก้ปัญหา การระบุตำแหน่งพร้อมกับการสร้างแผนที่ และได้ยกตัวอย่างอัลกอริทึมที่นิยมใช้ทั่วไปในการแก้ปัญหา SLAM ได้แก่ EKF SLAM, Information filters SLAM และ Graph-based SLAM

ในบทที่ 3 จะกล่าวถึงภาพรวมของการแก้ปัญหา การระบุตำแหน่งพร้อมกับการสร้างแผนที่ขนาดใหญ่ด้วยกล้องวิดีโอมุมกว้าง โดยจะกล่าวถึง อุปกรณ์ที่ใช้ในการทดลองพร้อมวิธีในการ calibrate และโมเดลการวัด รวมไปถึงวิธีอธิบายแผนที่ และภาพรวมของแนวทางการแก้ปัญหา

ในบทที่ 4, 5 และ 6 จะกล่าวถึงรายละเอียดในการแก้ปัญหา การระบุตำแหน่งพร้อมกับการสร้างแผนที่ขนาดใหญ่ด้วยกล้องวิดีโอมุมกว้าง โดยขั้นตอนการทำงานสามารถแบ่งออกได้เป็นสามส่วนด้วยกันได้แก่ การประมาณการเคลื่อนที่จากข้อมูลภาพ, การปรับแก้แผนที่เพื่อหาค่าเหมาะสุด และการตรวจหา Loop Closure โดยได้อธิบายในแต่ละหัวข้อในบทที่ 4, 5 และ 6 ตามลำดับ

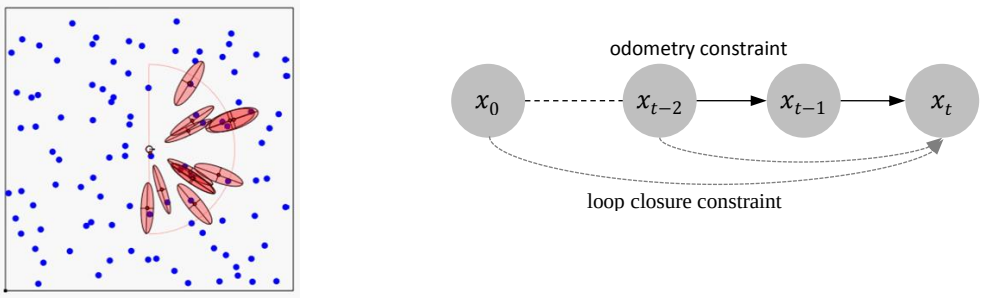
ในบทที่ 7 จะเป็นผลการทดลองการทำงานของอัลกอริทึมที่ได้นำเสนอกับสิ่งแวดล้อมจริง และบทที่ 8 จะเป็นบทสรุปของวิทยานิพนธ์ฉบับนี้

บทที่ 2
การระบุตำแหน่งพร้อมกับการสร้างแผนที่

การระบุตำแหน่งพร้อมกับการสร้างแผนที่ (SLAM) [1] คือการที่หุ่นยนต์ทำงานในสิ่งแวดล้อมที่หุ่นยนต์ไม่คุ้นเคย โดยหุ่นยนต์จะใช้อุปกรณ์วัดค่า (Sensor) ที่ติดตั้งอยู่บนตัวหุ่นในการตรวจวัดสิ่งแวดล้อม แล้วจึงนำข้อมูลการวัดที่ได้จากอุปกรณ์วัดค่ามาสร้างแผนที่ ในขณะเดียวกันหุ่นยนต์ก็ทำการระบุตำแหน่งของตัวเองในแผนที่นั้น ๆ หรือจะพูดได้ว่า เป็นกระบวนการที่หุ่นยนต์เคลื่อนที่ในสิ่งแวดล้อมที่ไม่คุ้นเคยได้โดยไม่หลงทางนั่นเอง (รู้ตัวว่าอยู่บริเวณไหนของสิ่งแวดล้อม)

ในกระบวนการสร้างแผนที่ของ SLAM นั้น แผนที่ที่หุ่นยนต์สร้างได้จะถูกใช้เพื่อการระบุตำแหน่งของหุ่นยนต์เป็นหลัก โดยสามารถแบ่งประเภทของแผนที่ออกเป็นประเภทใหญ่ ๆ ได้สองประเภทหลักด้วยกันคือ Feature-based SLAM และ View-based SLAM

- Feature-based SLAM [30, 45] จะอธิบายแผนที่ด้วยจุดสังเกตจำนวนมาก ซึ่งในที่นี้จุดสังเกตอาจแทนด้วย ตำแหน่งของวัตถุ จุดกลุ่มหมอก ,(Point Cloud), เส้นตรงในสิ่งแวดล้อมหรืออื่น ๆ ที่มีลักษณะเด่น แผนที่แบบ Feature-based ถือเป็นการอธิบายแผนที่ที่มีความนิยมสูง เนื่องจากความสามารถในการอธิบายสิ่งแวดล้อมได้อย่างตรงตัว (รูปที่ 2.1 (ก))
- View-based SLAM [28, 40] จะอธิบายแผนที่ด้วยความสัมพันธ์ของมุมมองของหุ่นยนต์ในอดีต ถึงแม้ว่าแผนที่แบบ View-based จะไม่สามารถอธิบายลักษณะของสิ่งแวดล้อมได้โดยตรงแต่เราก็สามารถสร้างแผนที่แบบ Feature-based ในบริเวณที่สนใจได้จากการแปลงค่าการวัดจากอุปกรณ์ ณ ขณะที่หุ่นยนต์อยู่ใน View นั้น ๆ ลงบนแผนที่ ข้อดีของแผนที่แบบ View-based ก็คือเป็นแผนที่มีความซับซ้อนต่ำสามารถปรับแก้แผนที่ได้อย่างรวดเร็ว (รูปที่ 2.1 (ข))



(ก) *(ข)*

รูปที่ 2.1 (ก) Feature-based representation (ข) view-based representation

2.1 การระบุตำแหน่งพร้อมกับการสร้างแผนที่ในเชิงความน่าจะเป็น

การแก้ปัญหา SLAM นั้นจะประกอบด้วย การประมาณตำแหน่งของหุ่นยนต์และการประมาณแผนที่ไปพร้อม ๆ กัน ซึ่งปัญหา SLAM จะเกี่ยวข้องกับการวัดค่าจากอุปกรณ์วัดค่าต่าง ๆ (Sensors) ซึ่งค่าที่วัดได้ นั้นย่อมมีความคลาดเคลื่อนของการวัดค่า ทำให้แผนที่และตำแหน่งของหุ่นยนต์ที่ประมาณได้ มีค่าความไม่แน่นอนเกิดขึ้น โดยทั่วไปแล้ว SLAM จะใช้วิธีการทางความน่าจะเป็น (Probabilistic Methods) มาจัดการความไม่แน่นอนเหล่านี้ให้มีค่าต่ำสุด

เมื่อรู้ข้อมูลการวัด (Measurement) จากอุปกรณ์วัดค่า (Sensor) และคำสั่งในการควบคุมหุ่นยนต์ (Control) ปัญหา SLAM จะแทนด้วยการประมาณการกระจายความน่าจะเป็นของตำแหน่งหุ่นยนต์และแผนที่

$$p(x_{1:t}, m | z_{1:t}, u_{1:t}, x_0)$$

โดยกำหนดให้

x_0	เป็นสถานะเริ่มต้นของหุ่นยนต์
$x_{1:t} = \{x_1, x_2, \dots, x_t\}$	เป็นเซตของสถานะของหุ่นยนต์ตั้งแต่เวลา 1 ถึงเวลา t
$m = \{m_1, m_2, \dots, m_n\}$	เป็นเซตของแผนที่ ซึ่ง m_i อาจแทนจุดสังเกตที่ใช้อธิบายแผนที่
$z_{1:t} = z^t = \{z_1, z_2, \dots, z_t\}$	เป็นเซตของค่าการวัด (Measurement) ตั้งแต่เวลา 1 ถึงเวลา t
$u_{1:t} = u^t = \{u_1, u_2, \dots, u_t\}$	เป็นเซตของคำสั่งควบคุมหุ่นยนต์ (Control) ตั้งแต่เวลา 1 ถึงเวลา t

สถานะของหุ่นยนต์ (x_i) และคำสั่งควบคุมหุ่นยนต์ (u_j) โดยปรกติแล้วจะอธิบายด้วยตำแหน่งของหุ่นยนต์และทิศทางของหุ่นยนต์ อาจเป็นค่าในสองมิติหรือสามมิติก็ได้ โดยสำหรับสถานะเริ่มต้นของหุ่นยนต์ (x_0) สามารถกำหนดเป็นค่าอะไรก็ได้แต่ปรกติแล้วจะกำหนดให้เป็นตำแหน่ง origin ของแผนที่ ส่วนแผนที่ (m) นั้นสามารถอธิบายได้หลายรูปแบบไม่ว่าจะเป็น เซตของตำแหน่งจุดสังเกต, occupancy grids, surface maps หรือ ข้อมูลดิบจากอุปกรณ์วัดค่า (Sensor) ซึ่งรูปแบบแผนที่จะขึ้นอยู่กับลักษณะของอุปกรณ์วัดค่า, ลักษณะสิ่งแวดล้อม และ อัลกอริทึมที่ใช้เป็นหลัก

การประมาณคำตอบของ SLAM แบ่งออกได้เป็นสองประเภทคือ Full SLAM และ Online SLAM

Full SLAM นั้นจะประมาณสถานะทั้งหมดของ SLAM ได้แก่แผนที่ทั้งหมดและตำแหน่งของหุ่นยนต์ตั้งแต่เวลาเริ่มต้นจนถึงปัจจุบัน โดยการกระจายความน่าจะเป็น สามารถเขียนได้ดังนี้

$$p(x_{1:t}, m | z_{1:t}, u_{1:t})$$

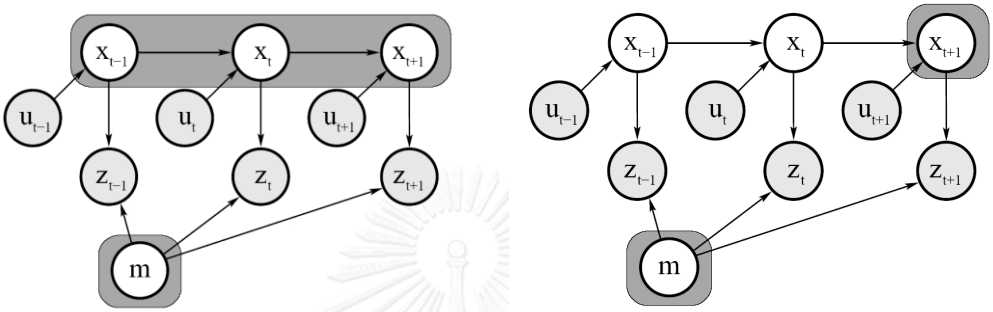
การประมาณคำตอบแบบ Full SLAM จะมีความถูกต้องของคำตอบสูงสุด เพราะเป็นการประมาณสถานะทั้งหมด แต่ก็ใช้เวลาในการประมวลผลนานเช่นกัน

Online SLAM นั้นจะประมาณสถานะของหุ่นยนต์เฉพาะเวลาปัจจุบันและสถานะของแผนที่ทั้งหมด ทำให้เวลาที่ใช้ในการประมวลผลไม่นานเกินไปนัก และสามารถทำงานแบบทันการณ์ได้ โดย ณ เวลาหนึ่ง ๆ การ

ประมาณการกระจายความน่าจะเป็นของ Online SLAM สามารถทำได้โดยการ Marginalize Out สถานะของหุ่นยนต์ก่อนหน้าทำให้ Online SLAM สามารถลดจำนวนสถานะลงได้

$$p(x_t, m|z_{1:t}, u_{1:t}) = \int \dots \int \int p(x_{1:t}, m|z_{1:t}, u_{1:t}) dx_1 dx_2 \dots dx_{t-1}$$

แผนภาพ Graphical Model ของ Full SLAM และ Online SLAM แสดงได้ดังรูปที่ 2.2 (ซ้าย) และรูปที่ 2.2 (ขวา) ตามลำดับ



รูปที่ 2.2 แผนภาพ Graphical Model ของ Full SLAM (ซ้าย) และ Online SLAM (ขวา)

การทำงานของ SLAM สามารถอธิบายได้ง่ายโดยใช้ dynamic Bayesian network (DBN) ตามรูปที่ 2.2 ซึ่งเป็น Graphical Model บรรยายกระบวนการสโทแคสติก (Stochastic Process) โดยใช้กราฟระบุทิศทาง (Directed Graph) จากกราฟ โหนดแต่ละโหนดจะแทน random variable ใน process ซึ่งโหนดที่เป็นสีขาวจะเป็น hidden variables ส่วนโหนดสีเทาจะเป็น observed variables สำหรับโหนดในกรอบสีเหลี่ยมสีเทาจะเป็นโหนด ที่ต้องการประมาณการกระจายความน่าจะเป็น โดยรูปที่ 2.2 (ซ้าย) สำหรับ Full SLAM ตัวแปรที่ต้องการประมาณค่าจะประกอบด้วยสถานะของหุ่นยนต์ทั้งหมด ($x_{1:t}$) และแผนที่ (m) ส่วนรูปที่ 2.2 (ขวา) Online SLAM จะประมาณค่าสถานะของหุ่นยนต์ ณ เวลาปัจจุบัน (x_t) และสถานะของแผนที่ (m) ส่วนเส้นเชื่อมระหว่างโหนดสองโหนดจะเป็น conditional dependence จะหว่างสองตัวแปร โดยในกระบวนการ SLAM จะใช้สองโมเดลในการบรรยายการเปลี่ยนแปลงของระบบ ประกอบด้วยโมเดลการเปลี่ยนแปลงสถานะของหุ่นยนต์ (state transition model) และ โมเดลการวัดจุดสังเกต (observation model)

โมเดลการเปลี่ยนแปลงของสถานะหุ่นยนต์ (state transition model) จะบรรยายการกระจายความน่าจะเป็นของสถานะหุ่นยนต์ (x_t) ที่เวลา t เมื่อรู้สถานะหุ่นยนต์ที่เวลา $t-1$ และคำสั่งควบคุมของหุ่นยนต์ (u_t) แสดงได้ดังนี้

$$P(x_t|x_{t-1}, u_t)$$

ส่วนโมเดลการวัดจุดสังเกต (observation model) จะบรรยายการกระจายความน่าจะเป็นของข้อมูลการวัดจุดสังเกตที่วัดได้จากหุ่นยนต์ (z_t) เมื่อรู้สถานะหุ่นยนต์ (x_t) และ ตำแหน่งของจุดสังเกต (m) ที่แน่นอน

$$P(z_t|x_t, m)$$

2.2 การประมาณคำตอบของ SLAM

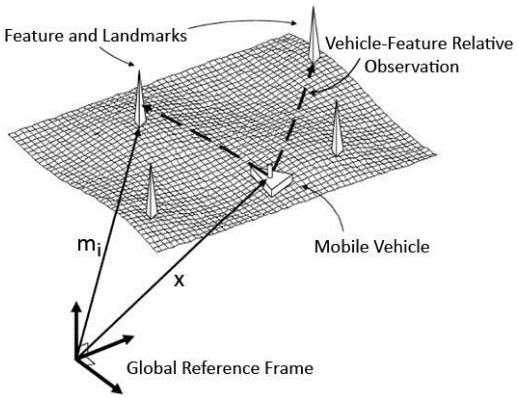
แนวทางในการแก้ปัญหา SLAM นั้น ได้มีงานวิจัยที่เสนอแนวทางในการแก้ปัญหาไว้อย่างหลากหลาย แต่แนวทางที่เป็นที่นิยมนั้นสามารถจำแนกได้เป็นสองแนวทางด้วยกัน ได้แก่ Filter-based Approaches และ Optimization-based Approaches

- Filter-based Approaches จะใช้ในการแก้ปัญหา Online SLAM โดยที่สถานของระบบจะประกอบด้วยตำแหน่งของหุ่นยนต์ ณ เวลาปัจจุบันและแผนที่ การทำงานของระบบจะมีลักษณะคล้าย filter นั่นคือระบบจะรับข้อมูลการวัดที่ได้จากอุปกรณ์วัดค่าและคำสั่งควบคุมหุ่นยนต์ โดยแต่ละช่วงเวลาที่ระบบได้รับข้อมูลเพิ่มนั้น ระบบจะนำข้อมูลมาปรับแก้สถานะให้มีความถูกต้องยิ่งขึ้นไปเรื่อย ๆ เรียกอีกอย่างได้ว่าระบบมีการทำงานแบบ incremental smoothing [18] ตัวอย่างอัลกอริทึมที่มีลักษณะเป็น filter-based ได้แก่ Kalman filters [22, 95, 96], Particle filters [19, 20], Information filters [28, 30] เป็นต้น
- Optimization-based Approaches จะทำการประมาณสถานะของหุ่นยนต์ทั้งหมดตั้งแต่เวลาเริ่มต้นจนถึงปัจจุบันและแผนที่ หรือพูดได้ว่าเป็นการประมาณเส้นทางการเคลื่อนที่ของหุ่นยนต์และแผนที่จากข้อมูลการวัดทั้งหมด ซึ่งก็คือการแก้ปัญหา Full SLAM นั้นเอง โดยปรกติแล้วการแก้ปัญหา Full SLAM จะอาศัยเทคนิค Least square Error Minimization มาใช้ในการประมาณสถานะที่เหมาะสมสุดที่ทำให้มีความคลาดเคลื่อนต่ำสุด ดังนั้นแนวทางนี้จึงถูกเรียกว่าวิธี Optimization-based การประมาณสถานะทั้งหมดด้วยวิธี optimization โดยอาศัยข้อมูลการวัดทั้งหมดนั้นมักจะใช้เวลานาน ดังนั้นการแก้ปัญหา Full SLAM มักจะไม่แบบทันการณ์ ตัวอย่างอัลกอริทึมได้แก่ Graph-based SLAM [34], TreeMap [35], TORO [36], iSAM [37] เป็นต้น

2.2.1 EKF SLAM

EKF SLAM เป็นการแก้ปัญหา SLAM ด้วย Extended Kalman filter (EKF) [22] ซึ่งสถานะของหุ่นยนต์และแผนที่ที่ต้องการประมาณจะถูกกำหนดให้เป็นสถานะของระบบ ส่วนโมเดลการเปลี่ยนแปลงของสถานะหุ่นยนต์ และโมเดลการวัดจุดสังเกต จะถูกเสนออยู่ในรูปโมเดลของ state-space ที่มีการกระจายความน่าจะเป็นแบบ Gaussian

เป้าหมายของ SLAM คือการระบุตำแหน่งของหุ่นยนต์ ในแผนที่ที่สร้างขึ้นโดยอาศัยข้อมูลจากการวัดจุดสังเกตจากอุปกรณ์วัดค่า (Sensor) โดยที่หุ่นยนต์นั้นจะติดตั้งอุปกรณ์วัดค่าบนตัวหุ่น ซึ่งอุปกรณ์วัดค่าจะมีความสามารถในการวัดค่าจุดสังเกต สัมพัทธ์กับตัวหุ่นยนต์ ดังรูปที่ 2.3



รูปที่ 2.3 เซนเซอร์วัดค่าจุดสังเกตในสิ่งแวดล้อม สัมพัทธ์กับตัวหุ่นยนต์

Process Model

สถานะของระบบจะประกอบด้วย ตำแหน่งและทิศทางของหุ่นยนต์ นอกจากนี้จะมีตำแหน่งของจุดสังเกตทั้งหมด (Landmark) โดยที่สถานะของหุ่นยนต์เขียนแทนด้วย x_t โมเดลการเปลี่ยนแปลงสถานะหุ่นยนต์เมื่อเวลาเปลี่ยนแปลงไป (state transition model) สามารถสร้างเขียนได้ดังนี้

$$x_t = f_x(x_{t-1}, u_t) + w_t$$

โดยที่ $f_x(x_t)$ เป็น ฟังก์ชันการเคลื่อนที่ของหุ่นยนต์, u_t เป็น คำสั่งการเคลื่อนที่ของหุ่นยนต์, w_t เป็น noise ที่ไม่ขึ้นกับโมเดลการเปลี่ยนแปลงสถานะมี mean เป็น 0 และ ความแปรปรวนร่วม (covariance) เป็น Q_t

ตำแหน่งของจุดสังเกต (Landmark) จุดที่ i จะแทนด้วย m_i โดยจะสมมุติว่าตำแหน่งที่แท้จริงของจุดสังเกตไม่เปลี่ยนแปลงตามเวลา จะได้โมเดลการเปลี่ยนแปลงสถานะจุดสังเกตสำหรับจุดสังเกตที่ i เป็น

$$m_{i,t} = m_{i,t-1} = m_i$$

เมื่อนำสถานะของหุ่นยนต์ และ ตำแหน่งของจุดสังเกตมาเขียนรวมกันจะได้

$$\mu_t = \begin{bmatrix} x_t \\ m_0 \\ \vdots \\ m_n \end{bmatrix} = \begin{bmatrix} f_x(x_{t-1}, u_t) \\ m_0 \\ \vdots \\ m_n \end{bmatrix} + \begin{bmatrix} w_t \\ 0 \\ 0 \\ 0 \end{bmatrix} = f(x_{t-1}, m, u_t) + W_t \qquad 2.1$$

โดยที่ $f(\cdot)$ เป็นโมเดลการเปลี่ยนแปลงของสถานะของระบบทั้งหมด (state transition model)

Observation Model

สำหรับอุปกรณ์วัดค่าที่ติดตั้งบนตัวหุ่นยนต์ อุปกรณ์จะมีความสามารถในการวัดค่าจุดสังเกตสัมพัทธ์กับตัวหุ่นยนต์ ดังนั้นจะได้ว่าโมเดลการวัดจุดสังเกตตัวที่ i (observation model) จะเขียนได้เป็น

$$z_{i,t} = h(x_t, m_i) + v_t \qquad 2.2$$

โดยที่ $h(x_t, m_i)$ เป็น ฟังก์ชันการวัดของเซนเซอร์, v_t เป็น noise ที่ไม่ขึ้นกับโมเดลการวัด มี mean เป็น 0 และความแปรปรวนร่วม (covariance) เป็น R_t โดยฟังก์ชัน $h(x_t, m_i)$ จะมีสถานะของหุ่นยนต์เป็นพารามิเตอร์ด้วย เนื่องจากการโมเดลวัดค่าจุดสังเกตจะขึ้นอยู่กับตำแหน่งของหุ่นยนต์ด้วย

The Estimation Process

Extended Kalman Filter จะประมาณสถานะของระบบ μ_t ด้วยวิธีเวียนบังเกิด (recursive) ซึ่งโมเดลการเปลี่ยนแปลงของระบบจะเป็นไปตามสมการ 2.1 และ โมเดลการวัดค่าจุดสังเกตจะเป็นไปตามสมการ 2.2 สถานะของ SLAM ที่ต้องการจะประมาณและความแปรปรวนร่วมของสถานะเขียนได้เป็น

$$\hat{\mu}_t = E[\mu_t \,|z_{0:t}]$$

$$\hat{e}_t = \hat{\mu}_t - \mu_t$$

$$P_t = E[\hat{e}_t \hat{e}_t^T \,|\, z_{0:t}]$$

เมื่อ $\hat{\mu}_t$ เป็นค่าประมาณสถานะหุ่นยนต์และแผนที่ โดย $E[\mu_t \,|z_{0:t}]$ คือ ค่าความคาดหวัง (Expected Value) ของ μ_t เมื่อรู้ข้อมูลการวัดจุดสังเกตทั้งหมดตั้งแต่เวลา 0 ถึง t ส่วน $\hat{e}_t$ เป็นความคลาดเคลื่อนการประมาณ และ P_t เป็นความแปรปรวนร่วมของสถานะ ซึ่งเมื่อสามารถประมาณสถานะหุ่นยนต์และความแปรปรวนร่วมได้หมายความว่าสามารถประมาณการกระจายความน่าจะเป็นของตำแหน่งของหุ่นยนต์และแผนที่ได้

EKF SLAM เป็นแนวทางแก้ปัญหาแบบ Filter-based Approaches ซึ่งการคำนวนผลลัพธ์ จะเป็นการคำนวนแบบเวียนบังเกิด (recursive solution) โดยแบ่งทำงานออกเป็นสามขั้นตอนคือ prediction, observation และupdate

- **Prediction**: กำหนดให้ $\hat{\mu}_{t-1}$ เป็นค่าประมาณสถานะของระบบ, P_{t-1} เป็นค่าประมาณความแปรปรวนร่วมของระบบ และกำหนดว่ารู้โมเดลตาม 2.1 และ 2.2 จะสามารถทำนายสถานะของระบบและค่าประมาณการวัดจุดสังเกตจุดที่ i รวมไปถึงความแปรปรวนร่วมที่เวลา t ได้เป็น

$$\hat{\mu}_t^- = f(\hat{\mu}_{t-1}, u_t)$$

$$\hat{z}_{i,t} = h_i(\hat{\mu}_t^-)$$

$$P_t^- = F_t P_{t-1} F_t^T + Q_t$$

โดยที่ F_t คือ Jacobian ของ $f(\hat{\mu}_{t-1}, u_t)$ เทียบกับการเปลี่ยนแปลงของ $\hat{\mu}_{t-1}$ และ u_t

- **Observation**: หลังจากการทำนายสถานะแล้ว เมื่อทราบค่าการวัดของจุดสังเกต $z_{i.t}$ จุดที่ i จะใช้ค่าการวัดจุดนี้ในการปรับแก้สถานะของระบบ โดยจะคำนวนค่าคลาดเคลื่อนการวัด (innovation) และ ความแปรปรวนร่วมค่าควาดเคลื่อน (innovation covariance matrix) ได้จาก

$$y_{i,t} = z_{i,t} - \hat{z}_{i,t}$$

$$S_t = H_t P_t^- H_t^T + R_t$$

ตามลำดับ โดยที่ H_t คือ Jacobian ของ $h(\hat{\mu}_t^-)$ เทียบกับการเปลี่ยนแปลงของ $\hat{\mu}_t^-$

- Update: สถานะประมาณของระบบ หลังจากการปรับแก้สถานะด้วยค่าการวัดจุดสังเกต และ ความแปรปรวนร่วมที่เวลา t หาได้จาก

$$\hat{\mu}_t = \hat{\mu}_t^- + K_t y_t$$

$$P_t = P_t^- - K_t S_t K_t^T$$

โดยที่ตัวคูณ K_t หาได้จาก

$$K_t = P_t^- H_t^T S_t^{-1}$$

2.2.2 Information filters SLAM

ในการแก้ปัญหา SLAM โดยทั่วไป การกระจายความน่าจะเป็นของสถานะของระบบจะถูกกำหนดให้เป็นแบบ Gaussian Distribution ซึ่งโดยปรกติแล้ว การอธิบายการกระจายความน่าจะเป็นแบบ Gaussian จะอธิบายได้ด้วย ค่าเฉลี่ย (mean) (μ_t) และ ค่าความแปรปรวน (covariance) (Σ_t)

$$p(x_t) = \mathcal{N}(x_t; \mu_t, \Sigma_t) = \frac{1}{\sqrt{|2\pi\Sigma_t|}} \exp\left\{-\frac{1}{2}(x_t - \mu_t)^T \Sigma_t^{-1}(x_t - \mu_t)\right\}$$

อย่างไรก็ดีการอธิบายการกระจายความน่าจะเป็นของสถานะระบบ ยังสามารถอธิบายในรูปแบบของ Information Vector (η_t) และ Information Matrix (Λ_t) โดยเราสามารถจัดรูปสมการการกระจายความน่าจะเป็นเสียใหม่ ได้เป็น

$$p(x_t) = \mathcal{N}^{-1}(x_t; \eta_t, \Lambda_t) = \frac{e^{-\frac{1}{2}\eta_t^T \Lambda_t^{-1} \eta_t}}{\sqrt{|2\pi\Lambda_t^{-1}|}} \exp\left\{-\frac{1}{2} x_t^T \Lambda_t x_t + \eta_t^T x_t\right\}$$

โดยที่

$$\Lambda_t = \Sigma_t^{-1} \quad \text{และ} \quad \eta_t = \Lambda_t \mu_t$$

Information Form มักถูกเรียกว่า Canonical หรือ Natural Representation ของการกระจายความน่าเป็นแบบ Gaussian โดยสาเหตุที่ถูกเรียกว่าเป็นการบรรยายแบบ Natural เนื่องจากส่วน exponential ของ Gaussian distribution สามารถเขียนได้เป็นผลบวกของเทอมสองเทอมแยกกัน ได้แก่เทอมของ Information Matrix และเทอมของ Information Vector

Operations ที่สำคัญของการกระจายความน่าเป็นแบบ Gaussian (Marginalization และ Conditioning) ที่อธิบายในรูป Covariance Form และ Information Form แสดงได้ใน*ตาราง 2.1*

ตาราง 2.1 สรุป Marginalization และ Conditioning Operations บนการกระจายแบบ Gaussian อธิบายในรูปของ Covariance Form และ Information Form

$$p(\alpha,\beta)=\mathcal{N}\left(\begin{bmatrix}\mu_\alpha\\ \mu_\beta\end{bmatrix},\begin{bmatrix}\Sigma_{\alpha\alpha} & \Sigma_{\alpha\beta}\\ \Sigma_{\beta\alpha} & \Sigma_{\beta\beta}\end{bmatrix}\right)=\mathcal{N}^{-1}\left(\begin{bmatrix}\eta_\alpha\\ \eta_\beta\end{bmatrix},\begin{bmatrix}\Lambda_{\alpha\alpha} & \Lambda_{\alpha\beta}\\ \Lambda_{\beta\alpha} & \Lambda_{\beta\beta}\end{bmatrix}\right)$$

	Marginalization $p(\alpha)=\int p(\alpha,\beta)d\beta$	Conditioning $p(\alpha\|\beta)=p(\alpha,\beta)/p(\beta)$
Cov. Form	$\mu=\mu_\alpha$ $\Sigma=\Sigma_{\alpha\alpha}$	$\mu'=\mu_\alpha+\Sigma_{\alpha\beta}\Sigma_{\beta\beta}^{-1}(\beta-\mu_\beta)$ $\Sigma'=\Sigma_{\alpha\alpha}-\Sigma_{\alpha\beta}\Sigma_{\beta\beta}^{-1}\Sigma_{\beta\alpha}$
Info. Form	$\eta=\eta_\alpha-\Lambda_{\alpha\beta}\Lambda_{\beta\beta}^{-1}\eta_\beta$ $\Lambda=\Lambda_{\alpha\alpha}-\Lambda_{\alpha\beta}\Lambda_{\beta\beta}^{-1}\Lambda_{\beta\alpha}$	$\eta'=\eta_\alpha-\Lambda_{\alpha\beta}\beta$ $\Lambda'=\Lambda_{\alpha\alpha}$

การทำงานของ Information Filter แบ่งได้เป็น 4 ขั้นตอนหลัก ๆ ด้วยกันได้แก่ State Augmentation, Measurement Updates, Motion Prediction และ State Recovery

- **State Augmentation** คือการเพิ่มสถานะเข้าไปในระบบ ซึ่งจะเกิดขึ้น เมื่อหุ่นยนต์เคลื่อนที่ไปข้างหน้า โดยเมื่อเพิ่มสถานะ x_{t+1} เข้าไปในระบบ การกระจายความน่าจะเป็นใหม่สามารถเขียนได้ดังนี้

$$p(x_{t+1},x_t,m|z^t,u^{t+1})=p(x_{t+1}|x_t,u_{t+1})p(x_t,m|z^t,u^t)$$

เมื่อโมเดลการเปลี่ยนแปลงของสถานะหุ่นยนต์ (state transition model) อยู่ในรูป

$$\begin{aligned}x_{t+1}&=f(x_t,u_{t+1})+w_t\\&\approx f\left(\mu_{x_t},u_{t+1}\right)+F(x_t-\mu_{x_t})+w_t\end{aligned}$$

กำหนดให้ F เป็น Jacobian ของฟังก์ชัน $f(x_t,u_{t+1})$ และ w_t เป็น white process noise จะหาสถานะประมาณของระบบได้เป็น

$$p(x_{t+1},x_t,m|z^t,u^{t+1})=\mathcal{N}^{-1}(\eta'_{t+1},\Lambda'_{t+1})$$

$$\eta'_{t+1}=\begin{bmatrix}Q^{-1}\left(f\left(\mu_{x_t},u_{t+1}\right)-F\mu_{x_t}\right)\\ \eta_{x_t}-F^TQ^{-1}\left(f\left(\mu_{x_t},u_{t+1}\right)-F\mu_{x_t}\right)\\ \eta_m\end{bmatrix}$$

$$\Lambda'_{t+1}=\begin{bmatrix}Q^{-1} & -Q^{-1}F & 0\\ -F^TQ^{-1} & \Lambda_{x_tx_t}+F^TQ^{-1}F & \Lambda_{x_tm}\\ 0 & \Lambda_{mx_t} & \Lambda_{mm}\end{bmatrix}$$

เมื่อ Q เป็น covariance ของ w_t

- **Measurement Updates** คือการปรับปรุงสถานะของระบบเมื่อได้รับค่าการวัด (Measurement) จากอุปกรณ์วัดค่า การกระจายความน่าจะเป็นใหม่สามารถเขียนได้ดังนี้

$$p(x_{t+1}, x_t, m|z^{t+1}, u^{t+1}) = p(z_{t+1}|x_{t+1}, x_i)p(x_{t+1}, x_t, m|z^t, u^{t+1})$$

เมื่อโมเดลการวัดจุดสังเกต (observation model) อยู่ในรูป

$$\begin{aligned} z_t &= h(x_t) + v_t \\ &\approx h(\bar{\mu}_t) + H(x_t - \bar{\mu}_t) + v_t \end{aligned}$$

กำหนดให้ H เป็น Jacobian ของฟังก์ชัน $h(x_t)$ และ v_t เป็น white measurement noise จะหาสถานะประมาณของระบบได้เป็น

$$\eta_t = \bar{\eta}_t + H^T R^{-1}(z_t - h(\bar{\mu}_t) + H\bar{\mu}_t)$$

$$\Lambda_t = \bar{\Lambda}_t + H^T R^{-1} H$$

เมื่อ R เป็น covariance ของ v_t

- **Motion Prediction** จะเป็นขั้นตอนที่จะลบสถานะของหุ่นยนต์ก่อนหน้าทิ้งไปซึ่งจะช่วยให้สถานะของระบบไม่ใหญ่เทอะทะ และสามารถทำงานได้แบบทันการณ์ (real time) การลดสถานะของระบบสามารถทำได้ด้วยการ Marginalization โดยในกรณีนี้ สมมุติว่าต้องการ Marginalize out สถานะ x_t

$$\begin{aligned} p(x_{t+1}, m|z^{t+1}, u^{t+1}) &= \int p(x_{t+1}, x_t, m|z^{t+1}, u^{t+1})\, dx_t \\ &= \mathcal{N}^{-1}(\bar{\eta}_{t+1}, \bar{\Lambda}_{t+1}) \end{aligned}$$

จะหาสถานะประมาณของระบบได้เป็น

$$\eta = \eta_\alpha - \Lambda_{\alpha\beta}\Lambda_{\beta\beta}^{-1}\eta_\beta$$

$$\Lambda = \Lambda_{\alpha\alpha} - \Lambda_{\alpha\beta}\Lambda_{\beta\beta}^{-1}\Lambda_{\beta\alpha}$$

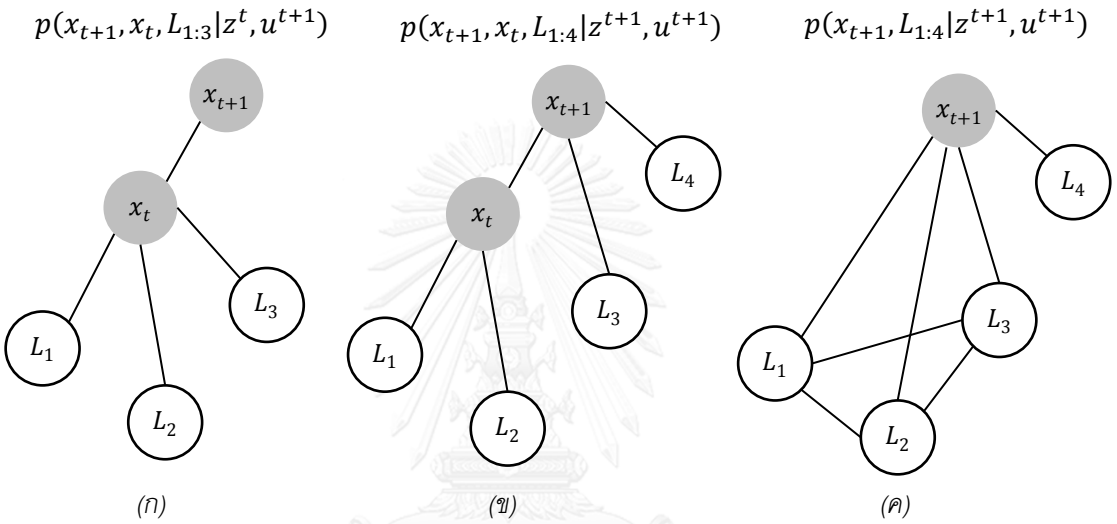
เมื่อ β เป็นสถานะที่ถูก Marginalize out และ α เป็นสถานะที่เหลืออยู่

- **State Recovery** เป็นการคำนวนเพื่อหาค่า mean และ covariance ของสถานะระบบทั้งนี้เนื่องจาก ในขั้นตอน State Augmentation และ Measurement Updates มีความจำเป็นต้องใช้ค่าสถานะในรูป Covariance Form โดย Full State Recovery สามารถคำนวนได้จาก

$$\Lambda_t \mu_t = \eta_t$$

อย่างไรก็ดี เนื่องจากในขั้นตอน State Augmentation และ Measurement Updates มีความต้องการใช้ค่า mean และ covariance เพียงบางส่วน จึงได้เกิดวิธี Partial State Recovery ซึ่งจะมีประสิทธิภาพในการทำงานที่ดีกว่า Full State Recovery ซึ่งในที่นี้จะไม่ขอกล่าวถึง

แผนภาพ Graphical Model สรุปขั้นตอนการทำงานของ Information Filter แสดงได้ดังรูปที่ 2.4 โดยในรูปที่ 2.4 (ก) ระบบได้มีการเพิ่มสถานะ x_{t+1} เข้าไปในระบบ, รูปที่ 2.4 (ข) ระบบได้ปรับปรุงสถานะของจุดสังเกต L_3 และ L_4 และ ในรูปที่ 2.4 (ค) ระบบได้ Marginalize out สถานะ x_t



รูปที่ 2.4 (ก) Augment state x_{t+1} (ข) Measurement updates (ค) Marginalize out state x_t

2.2.3 Graph-based SLAM

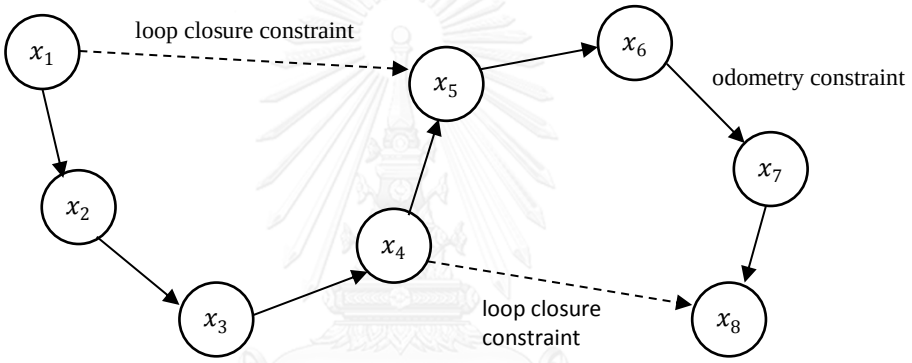
อีกหนึ่งแนวทางในการแก้ปัญหา SLAM คือการใช้ Graph-based ในการบรรยาย dynamic Bayesian network โดย Graph-based SLAM นั้นจะแทนปัญหาการระบุตำแหน่งและสร้างแผนที่ด้วยรูปแบบของกราฟที่มีโครงสร้างแบบ spatial

ในแต่ละโหนด (node) ของกราฟจะใช้แทนตำแหน่งของหุ่นยนต์ที่เวลาต่าง ๆ กัน โดยเส้นเชื่อม (edge) ระหว่างกราฟจะเป็น Spatial constraints ระหว่างตำแหน่งของหุ่นยนต์สองตำแหน่งดังแสดงได้ตามรูปที่ 2.5 ซึ่งคำนวนหาได้จาก ข้อมูลการวัดจากอุปกรณ์วัดค่า (sensor) ณ ช่วงเวลานั้น ๆ (z_t) หรือหาได้จากคำสั่งควบคุมหุ่นยนต์ (u_t) พูดอีกนัยหนึ่งก็คืออัลกอริทึม Graph-based SLAM จะทำการสร้างกราฟการเคลื่อนที่ของหุ่นยนต์จากข้อมูลการวัดที่ได้มาทั้งหมด โดยแต่ละโหนดจะเก็บข้อมูลตำแหน่งของหุ่นยนต์ และข้อมูลการวัด ณ เวลานั้น ๆ จากนั้นจึงทำการคำนวนหา configuration ของกราฟที่เหมาะสุด

การทำงานของอัลกอริทึมนั้น เมื่อหุ่นยนต์เคลื่อนที่ไปข้างหน้าเรื่อย ๆ อัลกอริทึมจะค่อย ๆ ทำการเพิ่มโหนดลงไปในกราฟที่ละโหนด โดยตำแหน่งของโหนดนั้น จะคำนวนหามาจากตำแหน่งของโหนดก่อนหน้าบวก

ด้วยการกระจัดของหุ่นยนต์ ซึ่งการกระจัดของหุ่นยนต์จะหาได้จากคำสั่งควบคุมหุ่นยนต์ หลังจากนั้นอัลกอริทึมจะเพิ่มเส้นเชื่อม ซึ่งเป็น odometry constraint สำหรับใช้บรรยายการกระจายความน่าจะเป็นของตำแหน่งสัมพัทธ์ (relative transformation) ระหว่างโหนดสองโหนด

เมื่อหุ่นยนต์เคลื่อนที่ไปเรื่อย ๆ จนวนบรรจบกลับมายังจุดเริ่มต้น แน่นอนว่าตำแหน่งของโหนดล่าสุดย่อมไม่สอดคล้องกับตำแหน่งของโหนด ณ จุดเริ่มต้น เนื่องจากความคลาดเคลื่อนจากการทำงานของหุ่นยนต์ที่ไม่สามารถเคลื่อนที่ได้ตรงตามคำสั่งควบคุมพอดี อัลกอริทึมจะตรวจหาจุดเริ่มต้นจากข้อมูลการวัดเพื่อหาความสอดคล้องระหว่างตำแหน่งปัจจุบันและตำแหน่งเริ่มต้น จากนั้นจึงคำนวนหาตำแหน่งสัมพัทธ์ระหว่างโหนดสองโหนดและเพิ่ม loop closure constraint ระหว่างโหนดล่าสุดและโหนดเริ่มต้น จากนั้นอัลกอริทึมจะทำ Graph Optimization ขึ้นกับ constraint ที่กำหนดไว้ เพื่อหาตำแหน่งของโหนดที่เหมาะสมสุดที่ทำให้ความคลาดเคลื่อนในแต่ละ constraint ต่ำสุด



รูปที่ 2.5 การบรรยาย SLAM ด้วย pose graph, โหนดแต่ละโหนดแสดงตำแหน่งของหุ่นยนต์, เส้นเชื่อมทึบแสดง odometry constraints ระหว่างสองโหนดที่อยู่ถัดกัน, เส้นเชื่อมประแสดง loop closure constraints จากข้อมูลการวัด

การทำงานของ Graph-based SLAM จะประกอบด้วยงานสองส่วนด้วยกันได้แก่ การสร้างกราฟจากข้อมูลการวัด (graph construction) และการหาค่าเหมาะสุดของตำแหน่งของโหนดในกราฟ โดยจะขึ้นกับ constraint ใน edge ที่กำหนดให้ (graph optimization) ขั้นตอน graph construction มักจะถูกเรียกว่าส่วน front-end ซึ่งจะเกี่ยวพันกับการประมวลผลข้อมูลดิบที่ได้จากอุปกรณ์วัดค่าส่วนขั้นตอน graph optimization มักจะถูกเรียกว่าส่วน back-end เพราะมีความเป็นอิสระจากข้อมูลการวัด

Problem Formulation

กำหนดให้ $x = (x_1, \dots, x_t)^T$ เป็นเวกเตอร์ของพารามิเตอร์ โดยที่ x_i เป็นตำแหน่งของโหนดที่ i เมื่อหุ่นยนต์เคลื่อนที่ไปข้างหน้าด้วยคำสั่งควบคุม u_i จะแสดงโมเดลของ odometry constraint ระหว่างโหนดสองโหนดที่อยู่ติดกันได้เป็น

$$x_{i+1} = f(x_i, u_i) + w_i$$

หลังจากหุ่นยนต์เคลื่อนที่วนกลับมายังจุดเดิมและส่วน front-end สามารถตรวจจับ loop closure ระหว่างโหนด x_i และ x_j ได้ loop closure constraint สามารถแสดงได้ดังนี้

$$x_j = f(x_i, u_{ij}) + w_{ij}$$

ในที่นี้กำหนดให้ u_{ij} เป็นตำแหน่งสัมพัทธ์ (relative transformation) ระหว่างโหนด x_i และโหนด x_j, $f(\cdot)$ เป็นโมเดลการเคลื่อนที่ของหุ่นยนต์ ซึ่งเป็น non-linear function และ w_k เป็น Gaussian error โดยมี mean เป็น 0 และ Covariance เป็น Σ_k

การกระจายความน่าจะเป็นอย่างมีเงื่อนไขของตัวแปรทั้งหมดทั้งตำแหน่งหุ่นยนต์ $x = \{x_i\}$ และ constraints $u = \{u_i \cap u_{ij}\}$ สามารถเขียนได้เป็น

$$P(x|u) \propto \underbrace{\prod_i P(x_{i+1}|x_i, u_i)}_{Odometry\ Constraints} \cdot \underbrace{\prod_{ij} P(x_j|x_i, u_{ij})}_{Loop\ Closures}$$

เป้าหมายของกระบวนการ graph optimization คือต้องการหา configuration ของตำแหน่งหุ่นยนต์ X^* ที่ maximize likelihood ข้างต้น หรือพูดอีกนัยหนึ่งได้ว่าต้องการจะ minimizes negative log likelihood ของ constraint ทั้งหมด

$$\begin{aligned} x^* &= \underset{x}{\operatorname{argmax}} P(x|u) = \underset{x}{\operatorname{argmin}} -\log P(x|u) \\ &= \underset{x}{\operatorname{argmin}} \sum_i e_i^T \Omega_i e_i + \sum_{ij} e_{ij}^T \Omega_{ij} e_{ij} \\ &= \underset{x}{\operatorname{argmin}} F(x) \end{aligned}$$

โดยที่ $\Omega_k = \Sigma_k^{-1}$ เป็น Information Matrix ของ constraints และ

$$e_i = f(x_i, u_i) - x_{i+1}$$

$$e_{ij} = f(x_i, u_{ij}) - x_j$$

การหาคำตอบของ likelihood ข้างต้นสามารถหาได้จากอัลกอริทึมสำหรับ optimization ทั่วไปเช่น Gradient Descent, Stochastic Gradient Descent, Gauss-Newton, Levenberg-Marquardt Algorithm

Nonlinear Least Squares Optimization

ถ้าหากว่ามี initial guess $\breve{x}$ ของตำแหน่งของหุ่นยนต์ที่ดีพอ เราสามารถใช้วิธีทาง numerical ในการหาคำตอบได้ เช่นวิธี Gauss-Newton หรือ Levenberg-Marquardt algorithms โดยแนวคิดก็คือจะประมาณ error function โดยใช้การกระจาย Taylor ลำดับที่ 1 รอบ ๆ ค่า initial guess $\breve{x}$

$$e_{ij}(\breve{x}_i + \Delta x_i, \breve{x}_j + \Delta x_j) = e_{ij}(\breve{x} + \Delta x)$$

$$\simeq e_{ij} + J_{ij}\Delta x$$

โดยที่ J_{ij} เป็น Jacobian ของ $e_{ij}(x)$ คำนวนที่ $\breve{x}$ และ $e_{ij} \overset{\text{def}}{=} e_{ij}(\breve{x})$

เมื่อแทนค่าประมาณ $e_{ij}(x)$ ในพจน์ error ของ function F_{ij} จะได้ว่า

$$F_{ij}(\breve{x} + \Delta x) = e_{ij}(\breve{x} + \Delta x)^T \Omega_{ij} e_{ij}(\breve{x} + \Delta x)$$

$$\simeq \left(e_{ij} + J_{ij}\Delta x\right)^T \Omega_{ij}(e_{ij} + J_{ij}\Delta x)$$

$$= \underbrace{e_{ij}^T \Omega_{ij} e_{ij}}_{c_{ij}} + 2\,\underbrace{e_{ij}^T \Omega_{ij} J_{ij}}_{b_{ij}}\Delta x + \Delta x^T \underbrace{J_{ij}^T \Omega_{ij} J_{ij}}_{H_{ij}}\Delta x$$

$$= c_{ij} + 2b_{ij}\Delta x + \Delta x^T H_{ij}\Delta x \qquad 2.3$$

โดยเราสามารถเขียนรวม Function $F(x)$ ได้เป็น

$$F(\breve{x} + \Delta x) = \sum_i F_i(\breve{x} + \Delta x) + \sum_{ij} F_j(\breve{x} + \Delta x)$$

$$\simeq \sum_i (c_i + 2b_i\Delta x + \Delta x^T H_i \Delta x) + \sum_{ij}\left(c_{ij} + 2b_{ij}\Delta x + \Delta x^T H_{ij}\Delta x\right) \qquad 2.4$$

$$= c + 2b^T\,\Delta x + \Delta x^T H \Delta x \qquad 2.5$$

Quadratic form ในสมการ 2.5 สามารถหาได้จากสมการ 2.4 โดยกำหนดให้ $c = \sum_i c_i + \sum_{ij} c_{ij}$, $b = \sum_i b_i + \sum_{ij} b_{ij}$ และ $H = \sum_i H_i + \sum_{ij} H_{ij}$

เราสามารถหาค่า Δx^* ที่ minimize Function $F(x)$ ข้างต้นได้จากการแก้ Linear System

$$H\Delta x^* = -b \qquad 2.6$$

Matrix H เป็น Information Matrix ของระบบซึ่งหาได้จากผลรวมของ measurement error ของตำแหน่งของหุ่นยนต์ผ่านทาง Jacobians ซึ่ง Matrix H จะเป็น sparse matrix เนื่องจากในขั้นตอนการสร้างกราฟ โหนดหนึ่งหนึ่งจะมี constraints ร่วมกับโหนดอื่น ๆ เป็นจำนวนค่อนข้างจำกัด ดังนั้นทำให้สามารถแก้ Linear System ข้างต้นได้อย่างมีประสิทธิภาพโดยใช้ sparse Cholesky factorization

คำตอบของระบบหาได้จากผลบวกของ initial guess กับค่า Δx^*

$$x^* = \breve{x} + \Delta x^* \qquad 2.7$$

วิธี Gauss-Newton จะใช้การคำนวนแบบวนซ้ำในการหา Information Matrix และ Information Vector จากสมการ 2.5, คำนวน Δx^* ในสมการ 2.6 และปรับแก้ค่าในสมการ 2.7 ไปเรื่อย ๆ จนผลลัพธ์ลู่เข้าสู่คำตอบที่ถูกต้อง

Least Squares on a Manifold

กระบวนการที่ได้อธิบายไปในหัวข้อ Nonlinear Least Squares Optimization เป็นวิธีการทั่วไปสำหรับการแก้ปัญหา minimization ฟังก์ชันหลายตัวแปร โดยได้สมมุติว่า space ของพารามิเตอร์ x เป็น Euclidean space อย่างไรก็ดีสำหรับปัญหา SLAM นั้น พารามิเตอร์ x ไม่ได้ span บน Euclidean space ดังนั้นการใช้ Least Squares แบบปรกติกับ SLAM อาจทำให้เกิดปัญหา sub-optimal solutions ก็ได้ วิธีที่ดีกว่าในการจัดการกับปัญหาที่เป็น non-Euclidean spaces ก็คือการ optimization บน manifold [97]

manifold เป็น Topological space ที่ประพฤติตัวคล้าย Euclidean space ใน local scale แต่อาจไม่ได้เป็น Euclidean ใน global scale ก็ได้ [98] ในปัญหา SLAM นั้น แต่ละตัวแปร x_i จะประกอบด้วยการเคลื่อนที่ (translation) และการหมุน (rotation) แน่นอนว่าการเคลื่อนที่ t_i นั้นอยู่บน Euclidean space แต่การหมุน r_i นั้น span บน rotation group $SO(2)$ หรือ $SO(3)$ ซึ่งเป็น non-Euclidean space การใช้ minimal representation ในการบรรยายการหมุนเช่นการใช้ Euler angles ในสามมิติ อาจนำไปสู่ปัญหา singularities ได้ ซึ่งการจะหลีกเลี่ยง singularities นั้นอาจจะต้องบรรยาย space ด้วยวิธีที่ over-parametrized เช่นการใช้ rotation matrices หรือ quaternions ซึ่งการใช้ตัวแปรที่ over-parametrized ในกระบวนการ optimization นั้นจะทำให้มี degrees of freedom มากเกินจนอาจจะทำให้เสีย constraints ในการหาค่าเหมาะสุดได้

วิธีที่ดีกว่าในการแก้ปัญหา SLAM คือการทำ optimization บน manifold ซึ่งจะเริ่มจากการนิยาม operator $\boxplus$ ที่แปลง local variation Δx บน Euclidean space ไปยัง variation บน manifold [99]

$$\Delta x \mapsto x \boxplus \Delta x$$

ด้วย operator ใหม่นี้เราสามารถนิยาม error ฟังก์ชันใหม่ได้เป็น

$$\begin{aligned}\check{e}_{ij}(\Delta\tilde{x}_i, \Delta\tilde{x}_j) &\stackrel{\text{def}}{=} e_{ij}(\check{x}_i \boxplus \Delta\tilde{x}_i, \check{x}_j \boxplus \Delta\tilde{x}_j) \\ &= e_{ij}(\check{x} \boxplus \Delta\tilde{x}) \simeq \check{e}_{ij} + \tilde{J}_{ij}\Delta\tilde{x}\end{aligned}$$

โดยที่ $\check{x}$ span บน over-parametrized space ดั้งเดิม โดยอาจจะเป็น quaternions หรือ rotation matrix ก็ได้ ส่วนเทอม $\Delta\tilde{x}$ เป็นค่าความเปลี่ยนแปลงปริมาณน้อย ๆ บริเวณรอบจุด $\check{x}$ และอธิบายด้วย minimal representation สำหรับในกรณีของ 3D SLAM เราจะแทนพารามิเตอร์ $\check{x}$ ด้วย transformation matrix และค่าความเปลี่ยนแปลงปริมาณน้อย ๆ จะเป็น vector 6 มิติ $\Delta\tilde{x} = [\Delta\tilde{t}^T \ \Delta\tilde{r}^T]^T$ โดย $\Delta\tilde{t}$ แทนการเลื่อนตำแหน่ง และ $\Delta\tilde{r}$ เป็น Rodrigues rotation ดังนั้น operator $\boxplus$ ก็คือการแปลง vector 6 มิติ $\Delta\tilde{x}$ ไปเป็น transformation matrix จากนั้นก็คูณเข้ากับพารามิเตอร์ $\check{x}$

ในขั้นตอน error minimization ในสมการที่ 2.3 เราจะแทนเครื่องหมายบวกปรกติใน error ฟังก์ชัน ด้วย operator ใหม่ $\boxplus$ ที่ได้นิยามขึ้นมา จะเขียน Jacobian $\tilde{J}_{ij}$ ใหม่ได้เป็น

$$\tilde{J}_{ij} = \left.\frac{\partial e_{ij}(\breve{x} \boxplus \Delta\tilde{x})}{\partial \Delta\tilde{x}}\right|_{\Delta\tilde{x}=0}$$

เนื่องจาก error ฟังก์ชัน e_{ij} จะขึ้นกับตัวแปร $\Delta\tilde{x}_i$ และ $\Delta\tilde{x}_j$ ดังนั้นเราจะกระจาย Jacobian ได้ดังนี้

$$\tilde{J}_{ij} = \begin{pmatrix} \cdots \underbrace{\left.\frac{\partial e_{ij}(\breve{x} \boxplus \Delta\tilde{x})}{\partial \Delta\tilde{x}_i}\right|_{\Delta\tilde{x}=0}}_{\tilde{A}_{ij}} \cdots \underbrace{\left.\frac{\partial e_{ij}(\breve{x} \boxplus \Delta\tilde{x})}{\partial \Delta\tilde{x}_j}\right|_{\Delta\tilde{x}=0}}_{\tilde{B}_{ij}} \cdots \end{pmatrix}$$

จากใช้ chain rule และอาศัยข้อเท็จจริงที่ว่า Jacobian นี้ประเมินค่าที่จุด $\Delta\tilde{x} = 0$ จะเขียน partial derivatives ได้เป็น

$$\frac{\partial e_{ij}(\breve{x} \boxplus \Delta\tilde{x}_i)}{\partial \Delta\tilde{x}_i} = \underbrace{\frac{\partial e_{ij}(\breve{x})}{\partial \breve{x}_i}}_{A_{ij}} \cdot \underbrace{\left.\frac{\breve{x}_i \boxplus \Delta\tilde{x}_i}{\partial \Delta\tilde{x}_i}\right|_{\Delta\tilde{x}=0}}_{M_i}$$

$$\frac{\partial e_{ij}(\breve{x} \boxplus \Delta\tilde{x}_j)}{\partial \Delta\tilde{x}_j} = \underbrace{\frac{\partial e_{ij}(\breve{x})}{\partial \breve{x}_j}}_{B_{ij}} \cdot \underbrace{\left.\frac{\breve{x}_j \boxplus \Delta\tilde{x}_j}{\partial \Delta\tilde{x}_j}\right|_{\Delta\tilde{x}=0}}_{M_j}$$

จะเห็นได้ว่า Jacobian สำหรับ optimize บน manifold สามารถหาได้จากการหา Jacobian บน Euclidean space ปรกติ แล้วคูณด้วยอนุพันธ์ของ operator $\boxplus$ เทียบกับค่า $\breve{x}_i$ และ $\breve{x}_j$

จากนั้นการหา $\Delta\tilde{x}^*$ จะหาได้จากการแก้ Linear System ด้วยอัลกอริทึมสำหรับ optimization ทั่วไป

$$\tilde{H}\Delta\tilde{x}^* = -\tilde{b}$$

การปรับแก้ค่า initial guess ทำได้โดยการ เพิ่มค่าความเปลี่ยนแปลงปริมาณน้อย ๆ $\Delta\tilde{x}^*$ ลงไปด้วย operator $\boxplus$

$$x^* = \breve{x} \boxplus \Delta\tilde{x}^*$$

บทที่ 3
การระบุตำแหน่งพร้อมกับการสร้างแผนที่ขนาดใหญ่ด้วยกล้องวิดีโอมุมกว้าง

การระบุตำแหน่งพร้อมกับการสร้างแผนที่ขนาดใหญ่ด้วยกล้องวิดีโอมุมกว้าง ในที่นี้จะหมายถึงการใช้กล้องวิดีโอมุมกว้างเพียงตัวเดียว (สมมุติให้เป็นอุปกรณ์วัดค่า (Sensor) สำหรับหุ่นยนต์) เคลื่อนที่ในสิ่งแวดล้อมขนาดใหญ่ โดยเป้าหมายคือการนำเอาข้อมูลลำดับภาพจากกล้องวิดีโอเพียงอย่างเดียวมาใช้ในการระบุตำแหน่งของกล้อง และสร้างแผนที่ของสิ่งแวดล้อมไปพร้อม ๆ กัน ปัญหาหลักของงานระบุตำแหน่งพร้อมกับสร้างแผนที่ในสิ่งแวดล้อมขนาดใหญ่ด้วยกล้องวิดีโอมุมกว้างสามารถแบ่งออกได้เป็นสามส่วนด้วยกันได้แก่ ปัญหาการระบุตำแหน่งพร้อมกับสร้างแผนทีด้วยกล้องวิดีโอเพียงตัวเดียว (Monocular Visual SLAM), ปัญหาการใช้กล้องวิดีโอมุมกว้าง และปัญหาการสร้างแผนที่สำหรับสิ่งแวดล้อมขนาดใหญ่ โดยแจกแจงปัญหาย่อยได้ดังนี้

ปัญหาการระบุตำแหน่งพร้อมกับสร้างแผนทีด้วยกล้องวิดีโอเพียงตัวเดียว

1. ข้อดีของการใช้กล้องวิดีโอก็คือ ข้อมูลที่ได้จากกล้องวิดีโอจะเป็นภาพสี ทำให้สามารถแยกแยะวัตถุและจุดสังเกตต่าง ๆ ในสิ่งแวดล้อมได้ดี แต่การใช้กล้องวิดีโอในงาน SLAM ก็มีข้อเสียก็คือ ข้อมูลที่ได้จากกล้องวิดีโอจะเป็นภาพในสองมิติทำให้ไม่สามารถ ระบุความลึกของวัตถุที่อยู่ในภาพได้ การจะนำข้อมูลภาพไปสร้างแผนที่ได้โดยตรง จะต้องใช้ภาพในหลาย ๆ มุมมองประกอบกันถึงสามารถสร้างแผนที่ของสิ่งแวดล้อมในสามมิติได้ แต่เนื่องจาก ภาพที่ได้จากกล้องวิดีโอทั่วไป จะมีมุมมอง (Field of View) ที่ค่อนข้างแคบ จึงทำให้มีโอกาสน้อยที่จะได้ภาพของสิ่งแวดล้อมที่สอดคล้องกันในมุมมองที่ต่างออกไปมาก ๆ หลายงานวิจัยจะแก้ปัญหานี้ด้วยการเปลี่ยนไปใช้อุปกรณ์ชนิดอื่น เช่นการเปลี่ยนไปใช้กล้องสเตอริโอ (Stereo Camera) ซึ่งจะทำให้ได้ข้อมูลความลึก จากการถ่ายภาพเพียงครั้งเดียวแต่ก็ไม่ละเอียดมากนัก หรือการใช้กล้องวิดีโอร่วมกับอุปกรณ์วัดระยะด้วยเลเซอร์ (Laser Range Finder) ก็จะทำให้ได้ข้อมูลความลึกของวัตถุเช่นกัน

ในวิทยานิพนธ์ฉบับนี้จะใช้กล้องวิดีโอมุมกว้าง (Wide-Angle Camera) ซึ่งจะช่วยแก้ปัญหามุมมองแคบของกล้องวิดีโอทั่วไป ทำให้มีโอกาสได้ภาพของสิ่งแวดล้อมเดิมในมุมมองที่ต่างออกไปมากขึ้น การใช้กล้องวิดีโอแบบมุมกว้าง จะมีข้อดีกว่าการใช้อุปกรณ์ชนิดอื่น ตรงที่มีความซับซ้อนของระบบน้อยกว่า จึงทำให้ใช้งานได้ง่ายกว่าและพกพาสะดวก อย่างไรก็ดีกล้องวิดีโอแบบมุมกว้างเพียงตัวเดียว ก็ไม่สามารถให้ข้อมูลความลึกได้ จึงทำให้การใช้กล้องวิดีโอแบบมุมกว้าง มีความท้าทายมากกว่าการใช้กล้องสเตอริโอ หรือกล้องวิดีโอร่วมกับอุปกรณ์วัดระยะด้วยเลเซอร์

2. วิทยานิพนธ์ฉบับนี้จะเน้นการระบุตำแหน่งพร้อมกับสร้างแผนที่โดยใช้กล้องวิดีโอทั่วไปแบบใช้คนถือ (Handheld Camera) ความต่างระหว่างกล้องวิดีโอแบบ handheld กับกล้องวิดีโอแบบติดตั้งบนหุ่นยนต์ก็คือ สำหรับกล้องวิดีโอแบบติดตั้งบนหุ่นยนต์ ระบบ SLAM จะสามารถคาดเดาการเคลื่อนที่ของกล้องได้ จากคำสั่งควบคุมการเคลื่อนที่ของหุ่นยนต์ (Robot Control Data) แต่ในกรณีของกล้องวิดีโอแบบ Handheld กล้องจะ

เคลื่อนที่ได้อย่างอิสระ จากการเคลื่อนโดยมนุษย์ ทำให้ระบบไม่สามารถคาดเดาการเคลื่อนที่ของกล้องได้ ดังนั้น การระบุตำแหน่งพร้อมกับสร้างแผนที่สำหรับกล้องวิดีโอแบบ Handheld จะมีความยุ่งยากมากขึ้น

3. การใช้กล้องวิดีโอแบบ handheld นั้น การเคลื่อนที่ของกล้องจะมีการสั่นมากกว่าปรกติ เนื่องจากการเคลื่อนที่ของมนุษย์จะไม่ราบเรียบเท่ากับการเคลื่อนที่ของหุ่นยนต์ ทำให้การประมาณโมเดลการเคลื่อนที่เป็นไปได้ยาก และข้อมูลภาพที่ได้รับจากกล้องอาจจะไม่คมชัด นอกจากนี้การเคลื่อนที่ของมนุษย์ จะมีการหมุน-ของกล้องมากกว่าปรกติซึ่งอาจจะทำให้สร้างแผนที่ของสิ่งแวดล้อมผิดพลาดได้ง่าย เนื่องจากมุมมองของกล้องเปลี่ยนไปมาบ่อย

4. ปัญหาสำคัญอีกประการของ Monocular Visual SLAM คือปัญหาการเบี่ยงเบนของขนาดแผนที่ (Scale Drift) ในการทำงานของ SLAM เมื่อหุ่นยนต์เคลื่อนที่ไปเรื่อย ๆ โดยไม่มีการ Close loop การเบี่ยงเบนของตำแหน่งประมาณของหุ่นยนต์จากค่าความเป็นจริงย่อมเกิดขึ้นได้ ซึ่งในที่นี้จะเรียกว่า Pose Drift แต่สำหรับกรณีของ Monocular Visual SLAM เนื่องจากข้อมูลการวัดไม่มีข้อมูลระยะทางร่วมด้วย ดังนั้น Monocular SLAM จึงมีโอกาสที่จะเกิดการเบี่ยงเบนของขนาดแผนที่ หรือในที่นี้เรียกว่า Scale Drift ซึ่งปัญหา Scale Drift นั้นนอกจากจะทำให้แผนที่สิ่งแวดล้อมไม่สอดคล้องกับความเป็นจริงแล้ว ยังทำให้การ Close loop ทำได้ยากขึ้นอีกด้วย ซึ่งแนวทางในการแก้ปัญหา Scale Drift จะได้นำเสนอต่อไป

ปัญหาการใช้กล้องวิดีโอมุมกว้าง

จุดประสงค์ของการใช้กล้องวิดีโอแบบมุมกว้างก็เพื่อเพิ่มโอกาสในการตรวจวัดจุดสังเกตในสิ่งแวดล้อม อย่างไรก็ดีการใช้กล้องวิดีโอแบบมุมกว้าง ก็มีความยุ่งยากมากกว่าการใช้กล้องวิดีโอทั่วไป นั่นคือภาพที่ได้จากกล้องวิดีโอแบบมุมกว้างนั้น จะไม่สามารถใช้โมเดลการวัดแบบ pinhole ตามปรกติได้ จะต้องใช้โมเดลการวัดแบบพิเศษ

ปัญหาการสร้างแผนที่สำหรับสิ่งแวดล้อมขนาดใหญ่

สิ่งแวดล้อมขนาดใหญ่ในที่นี้จะนิยามให้เป็นสิ่งแวดล้อมที่มีขนาดไม่จำกัด (สามารถใหญ่ขึ้นได้เรื่อย ๆ) ซึ่งปัญหาสำคัญของการสร้างแผนที่สำหรับสิ่งแวดล้อมขนาดใหญ่จะมีอยู่สามประการคือ ปัญหาประสิทธิภาพการคำนวน, ปัญหาความทนทานของระบบ และ ปัญหา loop closure

1. ปัญหาประสิทธิภาพการคำนวน: ขั้นตอนการทำงานของ SLAM จะประกอบด้วยขั้นตอนพื้นฐานสองขั้นตอนคือ การประมาณการเคลื่อนที่ของหุ่นยนต์ และการปรับแก้แผนที่และตำแหน่งของหุ่นยนต์โดยอาศัยข้อมูลการวัดที่ได้มาจากอุปกรณ์วัดค่า ในขั้นตอนการปรับแก้แผนที่นั้น ในช่วงหนึ่งการทำงานของ SLAM โดยทั่วไปจะทำการปรับแก้แผนที่แบบพร้อมกันทั้งหมด เนื่องจากแผนที่มีความขึ้นต่อกัน (dependency) และเมื่อแผนที่มีขนาดใหญ่เวลาในการคำนวนก็มากขึ้นด้วย จนเมื่อถึงจุดหนึ่ง SLAM ก็จะไม่สามารถทำงานแบบทันการณ์ (Real-Time) ได้ ดังนั้นจึงต้องมีกลวิธีอื่น ๆ เพื่อลดเวลาในการคำนวน

2. ปัญหาความทนทานของระบบ: วิธีแก้ปัญหา SLAM นั้นจะมุ่งเน้นในการจัดการความไม่แน่นอน (uncertainty) ของข้อมูลการวัดที่เกิดขึ้นจากอุปกรณ์วัดค่า เพื่อให้มีความคลาดเคลื่อนในการประมาณต่ำสุด อย่างไรก็ดีถ้าความไม่แน่นอนในระบบมีมากจนเกินไป จะมีผลกระทบกับความทนทานของระบบ หรืออาจจะทำให้ระบบล้มแหลวในที่สุด ยกตัวอย่างเช่น เมื่อหุ่นยนต์เคลื่อนที่ไปข้างหน้าเรื่อย ๆ โดยไม่มีการวกกลับมายังจุดเดิม ความคลาดเคลื่อนในการประมาณตำแหน่งของหุ่นยนต์และแผนที่ย่อมสูงขึ้นเรื่อย ๆ เนื่องจากมีการสะสมของความไม่แน่นอน ปัญหาที่จะเกิดตามมาก็คือ ในขั้นตอนการ close loop อาจจะเกิดความผิดพลาดในการหา data association ระหว่างแผนที่ข้อมูลการวัดได้ และบางครั้งอาจจะส่งผลทำให้การประมาณคำตอบของการปรับแก้แผนที่และสถานะของหุ่นยนต์ไม่ลู่เข้าสู่ค่าที่ต้องการ

3. ปัญหา loop closure: สืบเนื่องจากปัญหาความทนทานของระบบ เมื่อแผนที่มีขนาดใหญ่ขึ้น การตรวจหา Loop Closure จึงไม่สามารถใช้วิธีทางความความน่าจะเป็นแบบปรกติได้ เนื่องจากความคลาดเคลื่อนมีมากเกินไป จะต้องใช้ข้อมูล Vision มาใช้วิเคราะห์เพื่อหาความสอดคล้องระหว่างมุมมองของกล้องปัจจุบัน และ มุมมองของกล้องในอดีต ซึ่งปัญหาหลักของการตรวจหา Loop Closure ก็คือปัญหาเรื่องเวลาการคำนวณซึ่งจะเพิ่มมากขึ้นแปรผันตามขนาดของแผนที่

3.1 กล้องวิดีโอมุมกว้าง

กล้องวิดีโอมุมกว้างจะต่างจากกล้องวิดีโอแบบทั่วไปก็คือ มีมุมมอง (Field of View) ที่กว้างจนทำให้ภาพที่ได้มีความผิดเพียน (distort) ไปมาก และเนื่องจากมุมมองที่กว้าง จึงทำให้ไม่สามารถประมาณโมเดลของกล้องวิดีโอมุมกว้างด้วยโมเดลของกล้องรูเข็มได้ กล้องวิดีโอมุมกว้างสามารถแบ่งออกได้เป็นสองประเภทด้วยกันได้แก่ กล้องวิดีโอแบบออมนิ (Omni-directional Camera) และกล้องวิดีโอแบบเลนส์ตาปลา (Fish-eye Camera)

ในวิทยานิพนธ์ฉบับนี้จะใช้กล้องวิดีโอแบบเลนส์ตาปลา เป็นอุปกรณ์วัดค่าสำหรับการระบุตำแหน่งพร้อมกับการสร้างแผนที่ขนาดใหญ่ด้วยกล้องวิดีโอมุมกว้าง

กล้องวิดีโอแบบเลนส์ตาปลาสำหรับวิทยานิพนธ์ฉบับนี้จะเป็นกล้อง Canon รุ่น LEGRIA HF M300 โดยใช้คู่กับเลนส์ fisheye รุ่น opteka มีลักษณะดังรูปที่ 3.1 (ก) และภาพที่ได้จากกล้องดังกล่าวจะปรากฏดังรูปที่ 3.1 (ข) โดยตัวกล้องมีคุณสมบัติดังนี้

- ระบบภาพ NTSC หรือ PAL
- ความละเอียด 1920x1080 (Full HD)

ซึ่งภาพที่ได้จากกล้องจะมีมุมกว้างสุดประมาณ 170 องศา สามารถใช้งานได้ทั้งภาพ

(ก) *(ข)*

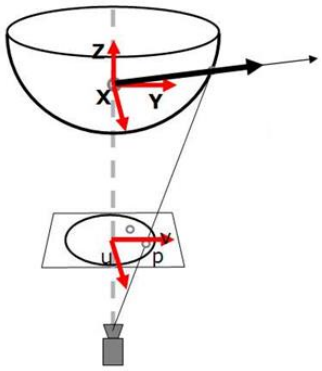
รูปที่ 3.1 (ก) กล้อง Canon ประกอบกับเลนส์ wide-angle opteka (ข) ภาพที่ได้จากกล้องมุมกว้าง

3.1.1 การหาพารามิเตอร์การเรียงตัวชิ้นส่วนของกล้อง

สำหรับกล้องวิดีโอทั่วไปแล้ว หลักการทำงานจะเป็นการการโปรเจคตำแหน่งวัตถุในพิกัดสามมิติให้มาอยู่ในพิกัดสองมิติของภาพ ซึ่งตำแหน่งที่โปรเจคในสองมิติของแต่ละกล้องก็จะแตกต่างกันไป

ในงานทางด้าน Robotic ที่มีการทำงานเกี่ยวข้องกับ Vision นั้น มีความจำเป็นอย่างยิ่งที่จะต้องทราบค่าพารามิเตอร์ และ Mapping Function ระหว่างตำแหน่งของวัตถุในสามสิติ และตำแหน่งในภาพ เพื่อใช้ในการคำนวน ดังนั้นสิ่งที่ขาดไปไม่ได้สำหรับงานทางด้าน Vision ก็คือการ Calibrate เพื่อหาค่าพารามิเตอร์ ของกล้อง

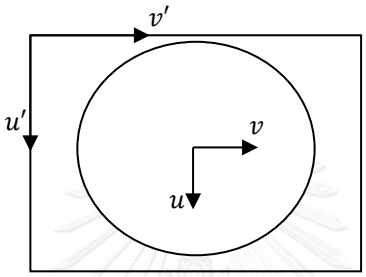
ในงานวิจัย [100] ได้นำเสนอหลักการในการประมาณพารามิเตอร์ของกล้องออมนิ (Omni Camera Calibration) ซึ่งจะเป็นการหาความสัมพันธ์ระหว่าง จุดสองมิติในภาพ (p) กับตำแหน่งในโลกสามมิติ (P) ที่สะท้อนมาจากกระจกดังรูปที่ 3.2 ซึ่งในความเป็นจริงแล้วกระบวนการนี้จะต้องหา intrinsic parameters ของกล้อง และ intrinsic parameters ของกระจก อย่างไรก็ตามโมเดลของกล้องออมนิ ในงานวิจัย [100] จะถูกมองเป็นระบบพิเศษระบบหนึ่ง ซึ่งจะไม่สนใจว่าจะใช้กระจก หรือ เลนส์ตาปลา ในกล้องออมนิ



รูปที่ 3.2 แสดงการ project ของแสงที่สะท้อนเข้ามายังกล้องออมนิ

ในงานวิจัย [100] ได้มีสมมุติฐานว่า กล้องจะเป็นแบบ central ดังนั้นจะต้องมีจุดจุดหนึ่งในกระจกซึ่งเป็นจุดตัดของ รังสีของแสงทั้งหมด ซึ่งจะกำหนดให้จุดจุดนั้นเป็นจุด origin ของระบบโคออนิเนตของกล้อง XYZ , กล้องและกระจกได้วางตัวอยู่ในแนวเดียวกัน, กระจกมีความสมมาตรในการหมุนรอบแกนของมัน, และ การ Distortion ของเลนส์ จะไม่ถูกพิจารณา ซึ่งเหตุผลที่ไม่สนใจการ Distortion ของเลนส์ เป็นเพราะกล้องออมนิที่ใช้กระจก โดยปรกติแล้วจะต้องการความยาวจุดโพกัสที่มากเพื่อที่จะโพกัสภาพได้บนกระจก ดังนั้นการ Distortion ของเลนส์สามารถละเลยได้ แต่ถ้าหากใช้เลนส์ตาปลา การ Distortion ของเลนส์จะรวมอยู่ใน Projection function อยู่แล้ว

พิจารณา โมเดลของกล้องออมนิ



รูปที่ 3.3 แสดงโคออดิเนตของพิกเซล (u', v') *และ โคออดิเนตที่พิจารณา distortion แล้ว* (u, v)

สมมุติให้กล้องและกระจกได้วางอยู่ในแนวเดียวกัน จาก*รูปที่ 3.3* กำหนดให้ (u', v') เป็นโคออดิเนตของพิกเซล ส่วน (u, v) เป็นโคออดิเนตที่พิจารณา distortion แล้ว จะเขียนการแปลง u' และ v' ได้ดังนี้

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} c & d \\ e & 1 \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} xc' \\ yc' \end{bmatrix}$$

จะได้ว่า

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1/det & -d/det \\ -e/det & c/det \end{bmatrix} \cdot \begin{bmatrix} u' - xc' \\ v' - yc' \end{bmatrix} \quad , det = c - e \cdot d$$

ซึ่ง c, d, e, xc' และ yc' เป็น parameter ที่ได้มาจากการ calibrate โดย xc' และ yc' จะเป็นจุดศูนย์กลางภาพ ส่วน c, d และ e จะเป็น affine transform matrix

จากรูปที่ 3.3 จะหา vector จากกล้องชี้ไปยังจุดในสามมิติได้เป็น

$$r = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} = \begin{bmatrix} u \\ v \\ f(\rho) \end{bmatrix} \qquad \begin{matrix} \rho = \sqrt{u^2 + v^2} \\ f(\rho) = a_0 + a_0\rho + a_0\rho^2 + a_0\rho^3 + a_0\rho^4 + \cdots \end{matrix} \qquad 3.1$$

จากสมการข้างต้นฟังก์ชัน $f(\rho)$ จะถูกประมาณด้วย Polynomial function ซึ่งสัมประสิทธิ์จะได้จากการ calibrate ซึ่งในที่นี้จะใช้ polynomial degree 4 ในการประมาณ

3.1.2 โมเดลการวัดของกล้อง

สำหรับกล้องวิดีโอแบบทั่วไป ตำแหน่ง pixel ที่ปรากฏในภาพ สามารถหาได้จากการ project ตำแหน่งวัตถุในสามมิติลงบนระนาบของภาพดัง*รูปที่ 3.4* (ก) โดยสามารถเขียนเป็นสมการได้ดังนี้

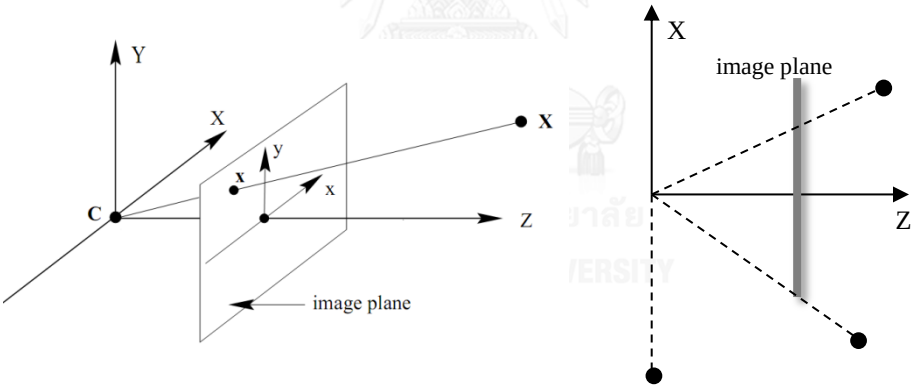
เมื่อจุด $p = [x\ y\ z]^T$ เป็นตำแหน่งในสามมิติเทียบกับแกนอ้างอิงของกล้อง ตำแหน่งของ pixel (u, v) ในภาพสามารถคำนวนได้จาก

$$x' = x/z, \quad y' = y/z$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x x' + c_x \\ f_y y' + c_y \end{bmatrix}$$

เมื่อ (f_x, f_y) เป็นความยาวโฟกัสใน pixel unit, (c_x, c_y) เป็น principal point ซึ่งโดยปรกติแล้วจะอยู่จุดศูนย์กลางของภาพ

โมเดลการวัดข้างต้นไม่สามารถใช้กับกล้องวิดีโอมุมกว้างได้ เนื่องจากสำหรับในมุมที่กว้างมาก ๆ จะทำให้ตำแหน่งที่ปรากฏบน image plane อยู่ห่างจากจุด principal point ออกไปมากและอาจจะไปปรากฏที่ระยะอนันต์ สำหรับจุดในสามมิติที่อยู่ตั้งฉากกับแกนอ้างอิงของกล้อง ดังแสดงให้เห็นในรูปที่ 3.4 (ข)



(ก) *(ข)*

รูปที่ 3.4 image plane projection สำหรับกล้องวิดีโอแบบปรกติ (ก) และ กล้องมุมกว้าง (ข)

สำหรับกล้องวิดีโอมุมกว้าง เราจะต้องอาศัยโมเดลพิเศษในการอธิบายการวัดตำแหน่งในสามมิติจากข้อมูลภาพ จากหัวข้อ Camera Calibration เราสามารถคำนวนหา vector ที่ชี้จากกล้องไปยังจุดในสามมิติได้จากสมการที่ 3.1

$$r = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} = \begin{bmatrix} u \\ v \\ f(\rho) \end{bmatrix} \qquad \rho = \sqrt{u^2 + v^2}$$

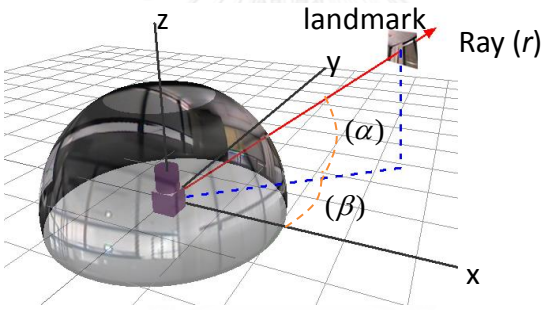
$$f(\rho) = a_0 + a_0\rho + a_0\rho^2 + a_0\rho^3 + a_0\rho^4 + \cdots$$

โมเดลการวัด (measurement model) จะหาได้การหา invert ของฟังก์ชันข้างต้นนี้ ซึ่งจะเห็นได้ว่าโมเดลการวัดนั้นมีความยุ่งยากเกินไปเพราะต้องหารากของ Polynomial function ดังนั้นเราจะไม่ใช้ตำแหน่ง pixel เป็นข้อมูลการวัดโดยตรง แต่จะแปลงให้อยู่ใน space อื่นเพื่อให้ง่ายในการใช้งาน

โมเดลการวัดของกล้องวิดีโอแบบมุมกว้าง จะใช้การ project ตำแหน่งวัตถุในสามมิติลงบน Sphere แทนที่จะ project ลงบน image plance (*รูปที่ 3.5*) โดยโคออดิเนตบน Sphere สามารถอธิบายได้ด้วย มุมอัลติจูด (α) และมุมอะซิมุท (β) ซึ่งสมการในการแปลง ตำแหน่งในสามมิติ $p = [x\ y\ z]^T$ ไปเป็นมุมอัลติจูด (α) และมุมอะซิมุท (β) จะแสดงได้ดังนี้

$$\begin{bmatrix}\alpha \\ \beta\end{bmatrix} = \begin{bmatrix}\arctan(z/\sqrt{x^2+y^2}) \\ \arctan(y/x)\end{bmatrix} \qquad 3.2$$

การหามุมอัลติจูด (α) และมุมอะซิมุท (β) จาก pixel บนภาพสามารถหาได้จากการคำนวน vector ที่ชี้จากกล้องไปยังจุดในสามมิติได้จากสมการที่ 3.1 แล้วจึงแทนค่าในสมการที่ 3.2 โดยกำหนดให้ $r = [x\ y\ z]^T$



รูปที่ 3.5 โมเดลการวัดของกล้องวิดีโอแบบมุมกว้าง

ถึงแม้ว่าการใช้มุมอัลติจูด (α) และมุมอะซิมุท (β) เป็นค่าการวัดที่ใช้ได้ดีกับกล้องวิดีโอแบบออมนิ แต่สำหรับกล้องวิดีโอแบบเลนส์ตาปลา ค่าการวัดจะเกิดปัญหา gimbal lock ตรงบริเวณส่วนบนของ sphere เมื่อมุมอัลติจูด (α) มีค่า 90 องศามุมอะซิมุท (β) จะมีค่าเป็นไปได้หลายค่า ดังนั้นการใช้มุมอัลติจูด (α) และมุมอะซิมุท (β) จึงไม่เหมาะสมกับกล้องวิดีโอแบบเลนส์ตาปลา

ในวิทยานิพนธ์นี้ จะใช้ค่าการวัดเป็น unit vector (r) ที่ชี้จากกล้องไปยังจุดในสามมิติโดยตรง โมเดลการวัดแสดงได้เป็น

$$r = [kx \quad ky \quad kz\ ]^T, \quad \rho = 1/\sqrt{x^2+y^2+z^2}$$

โมเดลการวัดนี้ ถึงแม้ว่าจะ over-parameterized เพราะว่ามี degrees of freedom เกินมา 1 degree แต่ก็ยังสามารถใช้งานได้ในกรณีที่ error จากการวัดไม่สูงนัก (น้อยกว่า 45 องศา) อย่างเช่นการใช้งานใน SLAM นั้นซึ่งมักจะใช้ข้อมูลภาพที่ต่อเนื่องกัน ทำให้ความต่างระหว่างภาพไม่มาก

3.2 การตรวจหาและวัดความสัมพันธ์ของจุดสังเกต

ข้อมูลการวัดที่ได้จากกล้องวิดีโอนั้นจะเป็นข้อมูลภาพสี ซึ่งประกอบด้วยค่าสีในแต่ละ pixel จำนวนมาก แน่นอนว่าข้อมูลมากมายเหล่านั้นย่อมไม่สามารถนำไปใช้งานได้ทั้งหมด เพราะมีข้อมูลฟุ่มเฟือยอยู่ด้วย ยกตัวอย่างเช่น pixels ส่วนที่เป็นท้องฟ้าสีขาวจำนวนมาก หรือ pixels ส่วนผนังปูนโล่ง ๆ การจะนำข้อมูลภาพมาใช้งาน จะต้องหาจุดเด่นในภาพ เช่น บริเวณ มุม, ขอบ, เส้นตรง หรือ พื้นผิวที่มีลวดลาย ในวิทยานิพนธ์ฉบับนี้จะใช้ features ที่มีลักษณะเป็นจุด ซึ่งอาจจะเป็น มุม หรือ จุดตัดในสิ่งแวดล้อม ซึ่งเป็นจุดที่มีการเปลี่ยนแปลงค่าสีในภาพ ซึ่งมีลักษณะเฉพาะแตกต่างกันในแต่ละจุด ดังแสดงได้ในรูปที่ 3.6 (ก)

มีอัลกอริทึมจำนวนมาก ที่สามารถใช้ในการตรวจหาจุดสังเกตในภาพ ยกตัวอย่างเช่น Fast corner detector [87], Harris corner detector [101], Star [102], SIFT [103], SURF [85], MSER [104] ในวิทยานิพนธ์นี้ ได้เลือกใช้ SIFT เนื่องจากว่า นอกจากตำแหน่งจุดสังเกตแล้ว SIFT ยังสามารถตรวจหาขนาด, ทิศทาง และ feature descriptors ของจุดสังเกตได้อีกด้วย ซึ่งมีความสำคัญมากในการหาความสัมพันธ์ระหว่างจุดสังเกต รายละเอียดการทำงานของอัลกอริทึม SIFT ได้อธิบายในภาคผนวก ก

(ก) *(ข)*

รูปที่ 3.6 (ก) จุดสังเกตที่ตรวจหาได้ในภาพ (ข) การตรวจหาความสัมพันธ์จุดสังเกต

Feature Matching

Feature matching คือการหาความสัมพันธ์ระหว่าง feature ปัจจุบันและ feature ที่เคยตรวจวัดได้ในอดีต สำหรับแต่ละ SIFT feature นอกจากค่าตำแหน่งในภาพแล้วยังมีข้อมูลสำคัญอื่น ๆ เช่น ขนาดของ feature, ทิศทางของ feature และ feature descriptors ซึ่งจะเป็น vector ใช้เก็บลักษณะเฉพาะของแต่ละ feature โดย features ที่มีความสอดคล้องกันจะมี descriptors ที่ใกล้เคียงกัน ดังนั้นการหาความสัมพันธ์ของ feature จะเป็นการค้นหาว่า feature ปัจจุบันมี descriptors ที่ใกล้เคียงกับ feature ที่เคยตรวจวัดได้ในอดีตอันใดบ้าง

การเปรียบเทียบ descriptors สอง descriptors จะใช้ Euclidean distance ในการเปรียบเทียบความต่าง แต่ในขั้นตอนการค้นหานั้น การวนซ้ำเปรียบเทียบทีละ feature จะทำให้เสียเวลาในการทำงานเป็นอย่างมาก ดังนั้นจึงใช้เทคนิค k-Nearest Neighbors Search เพื่อช่วยในการค้นหา (รูปที่ 3.6 (ข)) โดยในงานวิจัยนี้จะใช้อัลกอริึมจาก “Fast Library for Approximate Nearest Neighbors (FLANN)” [105]

การทำงานของ k-Nearest Neighbors Search จะเป็นการนำเอา features ในอดีตจำนวนมากมาสร้างเป็น k-d tree (เป็น binary tree ประเภทหนึ่ง) โดยแต่ละ node ใน k-d tree จะเลือกเงื่อนไขในการแบ่ง features ออกเป็นสองส่วน โดยพิจารณาจาก ข้อมูลมิติใดมิติหนึ่งใน descriptors ที่มีการกระจายของค่ามากสุด features จะถูกแบ่งไปเรื่อย ๆ จนท้ายสุดจะเหลือ feature เพียง 1 ตัวที่ leaf node ในขั้นตอนการค้นหานั้นจะค้นหา feature ที่มี descriptors สอดคล้องกันมากสุด โดยทำการค้นหาใน tree แบบ Branch and Bound ทำให้กรณีที่การกระจายของ features เป็นแบบสุ่มจะมีความเร็วในการค้นหาเป็น $O(\log(n))$

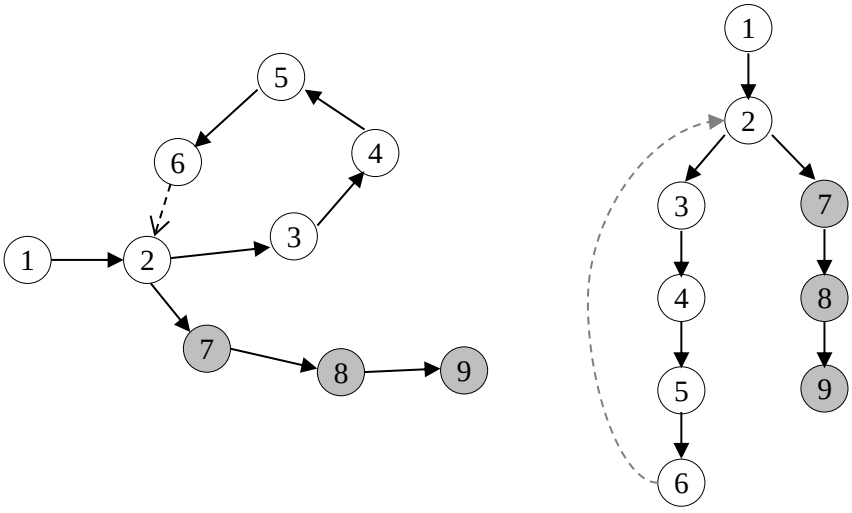
3.3 การบรรยายแผนที่

การบรรยายแผนที่สำหรับ SLAM สามารถแบ่งออกเป็นประเภทใหญ่ ๆ ได้สองประเภทคือ Feature-based Map และ View-based Map

Feature-based Map นั้นจะแทนแผนที่ด้วยตำแหน่งของจุดสังเกตจำนวนมาก ซึ่งการใช้ Feature-based Map นั้นอาจไม่เหมาะสมกับแผนที่ขนาดใหญ่เนื่องจากว่าขั้นตอนในการสร้างแผนที่นั้น ตำแหน่งของจุดสังเกตจะขึ้นกับตำแหน่งของหุ่นยนต์อย่างมีเงื่อนไข เมื่อหุ่นยนต์มีการเคลื่อนที่และมีการวัดค่าจุดสังเกต สถานะของหุ่นยนต์จะถูกประมาณตำแหน่งใหม่ และความน่าจะเป็นของตำแหน่งของจุดสังเกตจะถูกแพร่ไปยังตำแหน่งของจุดสังเกตอื่น ๆ ทำให้ตำแหน่งของจุดสังเกตมีความสัมพันธ์แบบขึ้นต่อกันทั้งหมด มีผลให้ Information Matrix ของแผนที่ไม่ sparse ดังนั้นในขั้นตอนการปรับแก้แผนที่ที่มีขนาดใหญ่นั้นจะทำงานได้ช้า

ในงานวิจัยนี้จะใช้วิธีบรรยายแผนที่แบบ View-based Map เป็นหลัก โดยแผนที่แบบ View-based จะอธิบายแผนที่ด้วยความสัมพันธ์ของมุมมองของหุ่นยนต์ในอดีต ข้อดีของแผนที่แบบ View-based ก็คือในแต่ละ view นั้นในยามปรกติจะมีความสัมพันธ์กับ view ก่อนหน้าและ view ถัดไปเท่านั้น ยกเว้นในกรณี Close Loop ซึ่ง view อาจจะมีความสัมพันธ์กับ view อื่นบ้างทำให้ Information Matrix ของแผนที่มีลักษณะ sparse ซึ่งเหมาะสมสำหรับแผนที่ขนาดใหญ่

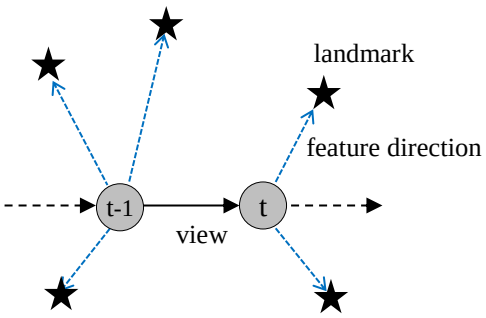
ในงานวิจัยนี้จะจัดเก็บข้อมูลของ view ในรูปแบบตำแหน่งแบบเชิงสัมพัทธ์ (Relative Transform) เทียบกับ view ก่อนหน้าเพื่อประโยชน์ในการปรับแก้แผนที่ โดยโครงสร้างความสัมพันธ์ของ View ในแผนที่จะอยู่ในลักษณะต้นไม้ดังแสดงได้ดังรูปที่ 3.7 (ข) โดยตำแหน่งของ Viewในโหนดลูก จะสัมพัทธ์กับตำแหน่งของ View ในโหนดพ่อ



รูปที่ 3.7 (ก) pose graph แสดงตำแหน่งของ view ในแผนที่ (ข) ต้นไม้ความสัมพันธ์ของ view

ตัวอย่างการสร้างความสัมพันธ์ของ view แบบต้นไม้ (tree parameterization of view) แสดงได้ดังรูปที่ 3.7 (ข) สำหรับการทำงานของขั้นตอนการสร้างต้นไม้ สามารถอธิบายได้โดยสมมุติให้หุ่นยนต์เคลื่อนที่ไปตามตำแหน่ง view 1, 2, 3, 4, 5, 6, 7, 8 และ 9 ตามลำดับ และที่ view 6 เกิด Close Loop กับตำแหน่ง view 2 ดังรูปที่ 3.7 (ก) เมื่อหุ่นยนต์เคลื่อนที่ไปยัง view ถัดไป โหนด 7 จะกลายเป็นลูกของโหนด 2 (แสดงได้ดังรูปที่ 3.7 (ข)) และเมื่อหุ่นยนต์เคลื่อนที่ต่อไป ก็จะเกิดโหนด 8 และ 9 เป็นอีกแขนงหนึ่งของต้นไม้

นอกจากนี้ในแต่ละ View ยังเก็บข้อมูลตำแหน่งจุดสังเกต โดยจะเก็บเป็น direction ของจุดสังเกตเทียบกับมุมมองของ View และข้อมูล invert depth ของระยะจาก view ถึงจุดสังเกต ดังรูปที่ 3.8 โดยข้อมูล invert depth นั้นจะถูกปรับแก้เมื่อได้รับข้อมูลการวัดเพิ่มเติม พูดอีกนัยหนึ่งก็คือ เป็นการเก็บตำแหน่งจุดสังเกตโดยใช้ view เป็น reference frame ซึ่งสาเหตุที่ต้องเก็บตำแหน่งจุดสังเกตเทียบกับ View นั้นเพราะต้องการใช้ประโยชน์ในการตรวจหา loop closure ในอนาคต และที่ต้องเก็บตำแหน่งสัมพัทธ์กับ view ก็เพื่อประโยชน์ในการปรับแก้แผนที่

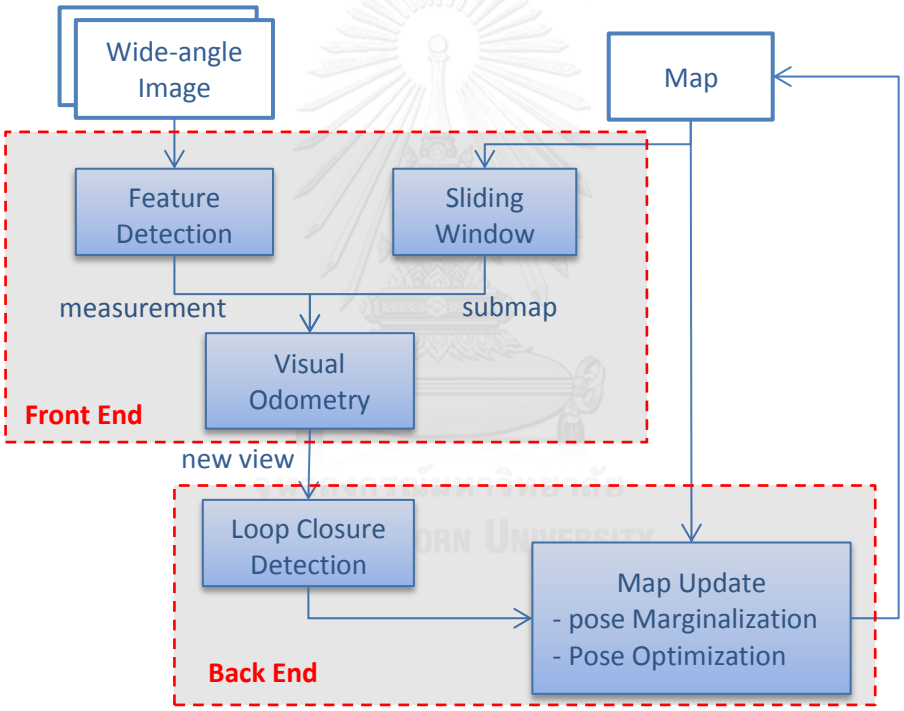


รูปที่ 3.8 ความสัมพันธ์ของ view และจุดสังเกต

3.4 การแก้ปัญหาการระบุตำแหน่งพร้อมกับการสร้างแผนที่ขนาดใหญ่ด้วยกล้องวิดีโอมุมกว้าง

อัลกอริทึมสำหรับแก้ปัญหาการระบุตำแหน่งพร้อมกับการสร้างแผนที่ขนาดใหญ่ด้วยกล้องวิดีโอมุมกว้าง ที่นำเสนอในวิทยานิพนธ์นี้จะเป็นแบบ optimization-based Approaches เนื่องจากเป็นแนวทางที่มีความ robust ในการใช้งาน เหมาะสมสำหรับการแก้ปัญหาแผนที่ขนาดใหญ่ ส่วนปัญหาด้านประสิทธิภาพการทำงานนั้น จะนำเสนอวิธีการในการแก้ปัญหาเป็นลำดับต่อไป

การทำงานของอัลกอริทึมที่นำเสนอจะแบ่งออกเป็นสองส่วนด้วยกันก็คือ ส่วน Front End ซึ่งทำหน้าที่ในการตรวจหาการเคลื่อนที่ของกล้องจากข้อมูลลำดับภาพ เพื่อนำมาสร้าง pose graph และเพิ่ม constraints ระหว่างโหนดในกราฟ โดยในส่วน Back End จะทำหน้าที่ optimize pose graph เมื่อเกิดการ close loop เพื่อหา configuration ที่เหมาะสมสุดสำหรับ pose ของแต่ละ view โดยไดอะแกรมการทำงานแสดงได้ดังรูปที่ 3.9



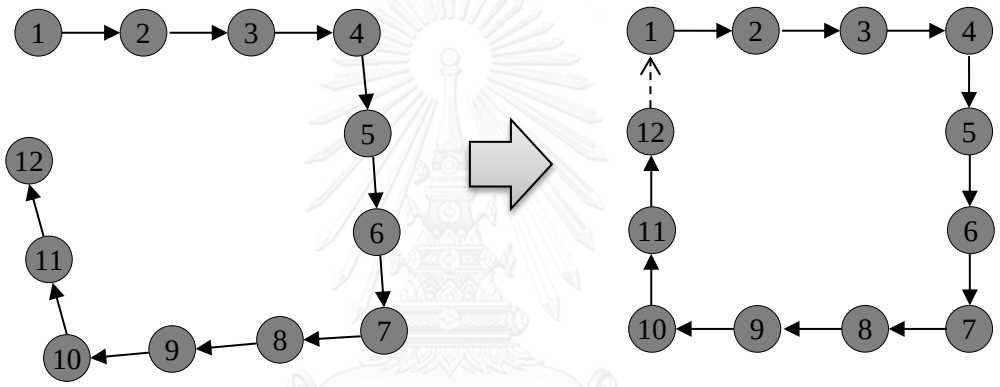
รูปที่ 3.9 ไดอะแกรมการทำงานของระบบ

Camera Tracking (Front End)

เมื่อหุ่นยนต์เคลื่อนที่ไปข้างหน้าเรื่อย ๆ อัลกอริทึมจะทำงานประมาณการเคลื่อนที่ของกล้องโดยอาศัยข้อมูลภาพเทียบกับข้อมูลภาพในอดีต (Visual Odometry) ในงานวิจัยนี้จะใช้ SIFT [103] ในการตรวจหาจุดสังเกตจากข้อมูลภาพ และหาความสัมพันธ์กับตำแหน่งจุดสังเกต ในอดีตด้วย k-Nearest Neighbors Search [102] ตามที่ได้กล่าวไว้ในหัวข้อ 3.2 ส่วนการประมาณการเลื่อนตำแหน่งนั้นจะใช้ Sliding-Window Bundle Adjustment [106] โดยอาศัยข้อมูลการวัดในอดีตหลาย ๆ view มาประมาณค่าเหมาะสุดของการเลื่อนตำแหน่ง รายละเอียดของการประมาณการเคลื่อนที่จะอธิบายในหัวข้อที่ 4

Map Optimization (Back End)

เนื่องจากความคลาดเคลื่อนในการประมาณการเคลื่อนที่จากข้อมูลภาพ ทำให้เมื่อหุ่นยนต์เคลื่อนที่ไปเรื่อย ๆ ความผิดพลาดในการประมาณตำแหน่งย่อมมากขึ้น เมื่อหุ่นยนต์ได้เคลื่อนที่กลับมายังตำแหน่งเดิมในอดีต ตำแหน่งหุ่นยนต์ย่อมไม่สอดคล้องกับตำแหน่งในอดีตพอดี กระบวนการปรับแก้แผนที่คือการหาค่าเหมาะสมสุดของแผนที่ในอดีตเมื่อมีการ close loop เพื่อให้ตำแหน่งหุ่นยนต์ปัจจุบันมีความสอดคล้องกับตำแหน่งหุ่นยนต์ในอดีต ตัวอย่างการปรับแก้แผนที่แสดงได้ดังรูปที่ 3.10 (ซ้าย) จะเห็นได้ว่าเมื่อหุ่นยนต์เคลื่อนที่เป็นวงรอบกลับมายังตำแหน่งเริ่มต้น และตรวจพบ loop closure ระหว่างโหนดที่ 1 กับโหนดที่ 12 ตำแหน่งหุ่นยนต์ที่ประมาณได้ คลาดเคลื่อนไปไกลจากความเป็นจริงมาก การ close loop จะทำให้โหนดทั้งหมดในกราฟถูกปรับแก้เพื่อหาตำแหน่งที่เหมาะสมสุดสำหรับทุกโหนดแสดงได้ดังรูปที่ 3.10 (ขวา)



รูปที่ 3.10 การปรับแก้แผนที่เพื่อ close loop

จะเห็นได้ว่ากระบวนการปรับแก้แผนที่ เป็นกระบวนการที่ใช้เวลาในการทำงานขึ้นกับขนาดของแผนที่ เนื่องจากการปรับแก้จะส่งผลกระทบกับโหนดทั้งหมดในแผนที่ โดยปรกติแล้วเวลาในการทำงานจะเป็น $O(n^2)$ เมื่อ n เป็นจำนวนโหนด ทำให้ไม่สามารถทำงานแบบทันการณ์ได้ในแผนที่ขนาดใหญ่

การปรับแก้แผนที่ในงานวิจัยนี้จะพยายามลดเวลาในการทำงานลงเพื่อให้สามารถทำงานได้ทันการณ์ โดยไม่ให้เสียคุณภาพของแผนที่ รายละเอียดของการปรับแก้แผนที่จะอธิบายในหัวข้อที่ 5

Loop Closure Detection

อีกหนึ่งกระบวนการที่มีความสำคัญมากสำหรับปัญหา Large Scale SLAM ก็คือการตรวจหา Loop Closure เนื่องจากว่าเมื่อหุ่นยนต์เคลื่อนที่ไปเป็นระยะทางไกลมาก ความคลาดเคลื่อนในการระบุตำแหน่งจะมากเกินกว่า จะหา data association ด้วยวิธีปรกติได้ ดังนั้นจึงต้องอาศัยข้อมูลอื่นช่วยเสริมในการทำ data association ซึ่งโดยปรกติแล้วการตรวจหา Loop Closure มักนิยมใช้ข้อมูล vision เนื่องจากมีข้อมูลภาพสีจำนวนมาก ทำให้สามารถแยกแยะความต่างของสิ่งแวดล้อมในบริเวณต่าง ๆ ได้ดี อย่างไรก็ดีปัญหาหลักของการตรวจหา Loop Closure สำหรับแผนที่ขนาดใหญ่ ก็คือการจัดการกับข้อมูลภาพในอดีตจำนวนมาก ซึ่งเมื่อมี

ภาพที่จะต้องค้นหามีจำนวนมากแล้ว นอกจากเวลาในการค้นหาจะเพิ่มขึ้น ยังจะส่งผลต่อความผิดพลาดในการค้นหา เพราะข้อมูลที่คล้ายคลึงกันจะมีมากขึ้น

ในวิทยานิพนธ์ฉบับนี้ก็จะใช้ข้อมูล vision ร่วมกับวิธีแบบ map-to-map ในการตรวจหา Loop Closure โดยจะอธิบายวิธีที่มีประสิทธิภาพในหัวข้อที่ 6

บทที่ 4
การประมาณการเคลื่อนที่จากข้อมูลภาพ (Front End)

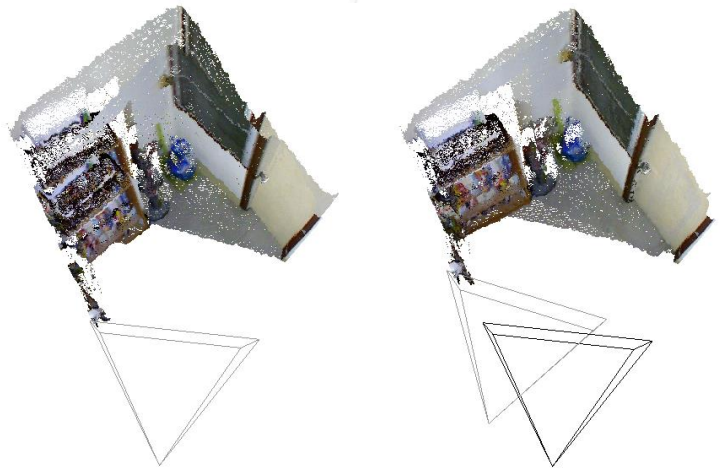
การประมาณการเคลื่อนที่จากข้อมูลภาพ (Visual Odometry) เป็นเทคนิคในการประมาณการเคลื่อนที่ของกล้องโดยอาศัยข้อมูลลำดับภาพที่ได้จากการวัดในแต่ละช่วงเวลา ซึ่งคำว่า Visual Odometry จะเป็นคำที่ล้อเลียนจากคำว่า Wheel Odometry อันเป็นการวัดการเคลื่อนที่ของหุ่นยนต์จากเซนเซอร์ encoder ที่ติดตั้งที่ล้อของหุ่นยนต์ กระบวนการประมาณการเคลื่อนที่จากข้อมูลภาพ สามารถจำแนกได้ 3 ประเภทใหญ่ ๆ โดยจะขึ้นกับอุปกรณ์วัดค่าได้แก่ Stereo Visual Odometry, Dense Visual Odometry และ Monocular Visual Odometry

ในยุคเริ่มแรกมักนิยมใช้กล้องแบบสเตอริโอ (Stereo Camera) ในการหา Visual Odometry [107-109] เนื่องจากข้อมูลการวัดที่ได้จากกล้องแบบสเตอริโอนั้นจะได้เป็นตำแหน่งจุดสังเกตในสามมิติ ขั้นตอนการทำงานของ Stereo Visual Odometry จะตรวจหา feature ในภาพโดยใช้อัลกอริทึมเช่น Fast Detector [87] หรือ Harris Detector [101] จากนั้นจึงตรวจหาความสัมพันธ์ของ feature ระหว่างภาพโดยใช้อัลกอริทึมเช่น SIFT [103], SURF [85] หรือ sum of absolute difference (SAD) จากนั้นการประมาณการเคลื่อนที่ของกล้องจะเป็นปัญหา 3D-to-3D point registration ซึ่งก็คือการหา rigid body transformation ที่ปรับให้กลุ่มจุดในสามมิติสองกลุ่มให้มีความสอดคล้องกัน โดยใช้ RANSAC ในการกรอง outliers

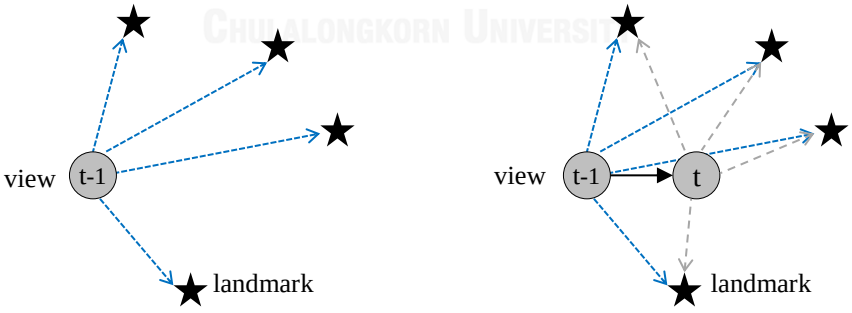
สำหรับการหา Visual Odometry โดยใช้อุปกรณ์วัดระยะด้วยเลเซอร์ (Laser Range Finder) ข้อมูลการวัดที่ได้จะเป็น point cloud ในสามมิติจำนวนมาก แต่การหาความสัมพันธ์ของแต่ละจุดใน point cloud กับข้อมูลก่อนหน้านั้นเป็นไปได้ยาก เนื่องจากไม่มีข้อมูลภาพสีในการหา feature descriptor การประมาณการเคลื่อนที่จึงนิยมใช้วิธี iterative closest point (ICP) [110] ซึ่งเป็นการทำงานแบบวนซ้ำโดยประมาณ rigid transformation จากคู่จุดที่อยู่ใกล้กันมากสุดจนกว่าจะลู่เข้าสู่คำตอบที่ต้องการ (แสดงได้ดังรูปที่ 4.1) อย่างไรก็ดีวิธี ICP จะทำงานได้ไม่ดีกับ point cloud ที่มีลักษณะราบเรียบ เนื่องจากไม่สามารถสวัดความเปลี่ยนแปลงอันเกิดมาจากการเลื่อนตำแหน่งได้ หรือในบางครั้ง point cloud ที่มีลักษณะซับซ้อน ก็อาจทำให้การประมาณไม่ลู่เข้าได้ เมื่อไม่นานมานี้กล้องวิดีโอแบบ RGB-D (RGB-D Camera) เริ่มเป็นที่นิยมในใช้หา Visual Odometry จึงเกิดอัลกอริทึมในการประยุกต์ใช้ ICP ร่วมกับจุด feature ในภาพ RGB [111, 112] หรืออัลกอริทึมแบบ direct motion estimation [113, 114] ซึ่งเป็นการประมาณการเลื่อนตำแหน่งจาก dense RGB-D image โดยตรงโดยไม่ต้องผ่านขั้นตอนการตรวจหา feature

การประมาณการเคลื่อนที่จากกล้องวิดีโอเพียงตัวเดียว (Monocular Visual Odometry) จะมีความซับซ้อนกว่าการประมาณการเคลื่อนที่จากกล้องแบบสเตอริโอ เนื่องจากกล้องวิดีโอเพียงตัวเดียวนั้นไม่รู้ข้อมูลความลึกของภาพ ดังนั้นจึงต้องประมาณการเคลื่อนที่และสร้างตำแหน่งจุดสังเกตในสามมิติไปพร้อม ๆ กันด้วยข้อมูลภาพในสองมิติเพียงอย่างเดียว (แสดงได้ดังรูปที่ 4.2) ปัญหา Monocular Visual Odometry นั้น

คล้ายคลึงกับปัญหา Monocular Visual SLAM เป็นอย่างมากโดยความต่างเพียงอย่างเดียวก็คือ อัลกอริทึม SLAM จะมีขั้นตอนการ close loop และการปรับแก้แผนที่เพิ่มขึ้นมา ในยุคแรกเริ่มนั้นการทำงานของ Monocular Visual Odometry จะใช้ RANSAC เพื่อการกรอง outliers ในขั้นตอนการประมาณ 3D-to-2D camera pose ซึ่งใช้อัลกอริทึม five-point relative pose [115] ในการคำนวนหาการเลื่อนตำแหน่งของกล้อง [116, 117] หลังจากนั้นจึงเริ่มมีการนำ Kalman Filter มาปรับใช้กับการหา Visual Odometry [118, 119] โดยอาศัยข้อมูลจาก Inertial measurement unit (IMU) ร่วมเพื่อเพิ่มความแม่นยำในการประมาณตำแหน่ง อย่างไรก็ดีการประมาณการเคลื่อนที่จากกล้องวิดีโอเพียงตัวเดียวนั้นถือเป็นปัญหายาก เนื่องการสะสมของความคลาดเคลื่อนที่ ระหว่างการประมาณการเคลื่อนที่จะก่อให้เกิดปัญหาการเบียงเบนขนาดของแผนที่ (Scale Drift) เมื่อคอมพิวเตอร์มีความเร็วในการทำงานมากขึ้น จึงมีงานวิจัยซึ่งเสนอทางแก้ปัญหาความแม่นยำการประมาณการเคลื่อนที่ด้วยวิธี “Sliding-Window Bundle Adjustment” [120, 121]



รูปที่ 4.1 แสดงการประมาณการเคลื่อนที่ด้วยข้อมูล Point Cloud



รูปที่ 4.2 แสดงการประมาณการเคลื่อนที่จากกลุ่มจุดสังเกต

วิธี Sliding-Window Bundle Adjustment จะประมาณการเคลื่อนที่โดยการทำ Bundle Adjustment เฉพาะส่วนโดยจะเลื่อนขอบเขตการทำงานของ Bundle Adjustment ไปเรื่อย ๆ ตามตำแหน่งของกล้องล่าสุด ซึ่งจะทำให้ประมาณการเคลื่อนที่ของกล้องมีความแม่นยำมาก แต่ก็ใช้เวลาในการทำงานพอสมควรเทียบกับวิธีอื่น นอกจากนี้ยังมีวิธีการประมาณการเคลื่อนที่จากกล้องวิดีโอเพียงตัวเดียวแบบ direct method ซึ่งจะติดตามการ

เคลื่อนที่โดยอาศัยข้อมูลทุก pixel ในภาพและสร้างแผนที่แบบหนาแน่น (dense) [53] แต่วิธีนี้ยังใช้พลังงานในการประมวลผลสูงมากและยังใช้กับแผนที่ขนาดใหญ่ไม่ได้

ในวิทยานิพนธ์ฉบับนี้จะประยุกต์ใช้วิธี Sliding-Window Bundle Adjustment ในการประมาณการเคลื่อนที่จากข้อมูลภาพโดยจะปรับแก้การทำงานให้มีความเหมาะสมสำหรับกล้องวิดีโอมุมกว้างและจะนำเสนอวิธีในการแก้ปัญหาการเบี่ยงเบนขนาดของแผนที่ (Scale Drift) โดยอาศัยข้อมูลสิ่งแวดล้อมเพิ่มเติม เพื่อให้สามารถคงขนาดของแผนที่เมื่อทำงานในสิ่งแวดล้อมขนาดใหญ่ได้

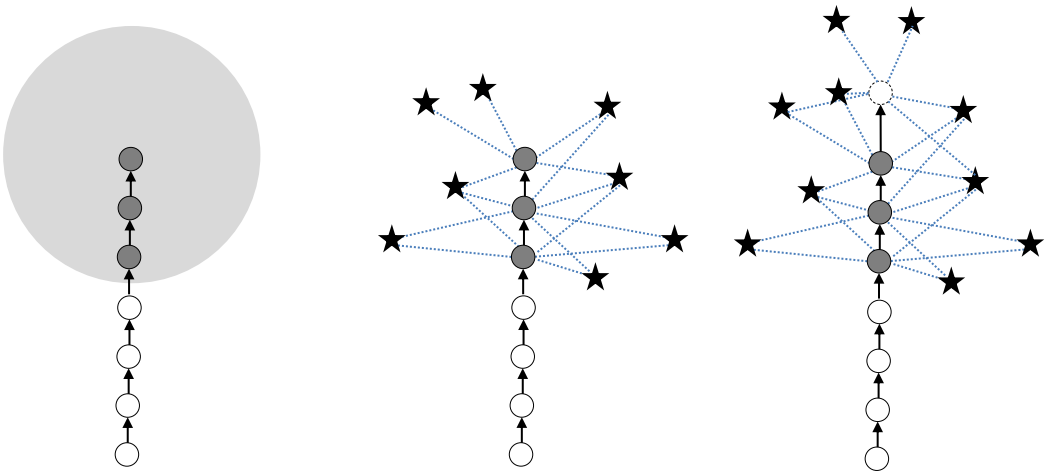
ขั้นตอนการประมาณการเคลื่อนที่จากข้อมูลภาพด้วยวิธี Sliding-Window Bundle Adjustment จะเริ่มจากการเลือก view จำนวน n views ที่อยู่ใกล้ view ปัจจุบันมากสุด จากนั้นจึงประมาณค่าคาดเดาเริ่มต้น (initial guess) การเลื่อนตำแหน่ง จากข้อมูลภาพที่ตรวจวัดได้และตำแหน่งจุดสังเกตในอดีต (อธิบายในหัวข้อที่ 4.1) แล้วจึงประมาณการเคลื่อนที่โดยละเอียดโดยทำหาค่าเหมาะสุด (optimization) ของกล้องและตำแหน่งของจุดสังเกต (อธิบายในหัวข้อที่ 4.2)

หลังจากการประมาณการเคลื่อนที่จากข้อมูลภาพแล้ว ข้อมูลการเคลื่อนที่ของกล้องที่ได้จะนำไปใช้ในการสร้าง pose graph และเพิ่ม odometry constraints ระหว่างโหนดในกราฟ ให้เกิดเป็นแผนที่ซึ่งใช้ในการบรรยายสิ่งแวดล้อม และเมื่อเกิดการ close loop การทำงานส่วน Back End (ในบทที่ 5) จะทำหน้าที่ในการ optimize pose graph ต่อไป

4.1 การหาความสัมพันธ์จุดสังเกตและการกำหนดค่าเริ่มต้น

จากวิธีบรรยายแผนที่ที่ได้นำเสนอ แผนที่จะถูกเก็บอยู่ในรูปแบบ relative view pose โดยแต่ละ view จะมีข้อมูลจุดสังเกตซึ่งตรวจวัดครั้งแรกที่ view นั้น ๆ ซึ่งข้อมูลจุดสังเกตจะอยู่ในรูป direction และ invert depth ตามที่ได้อธิบายไว้ในหัวข้อที่ 3.3 และเมื่อได้รับข้อมูลภาพใหม่จากกล้องระบบจะใช้ features ที่ตรวจหาได้ด้วย SIFT Algorithm เพื่อประมาณการเลื่อนตำแหน่งของกล้องเทียบกับจุดสังเกตในอดีต โดยมีขั้นตอนการทำงานดังต่อไปนี้

1. sliding-window จะถูกกำหนดโดยเลือก view จำนวน n views ที่อยู่ใกล้ view ปัจจุบันมากสุด เพื่อใช้เป็น view อ้างอิงในการประมาณการเลื่อนตำแหน่ง (รูปที่ 4.3 (ก))
2. เซตของจุดสังเกตจะสร้างจากจุดสังเกตที่ผูกกับ view ที่ใน sliding-window จากนั้นจะหาความสัมพันธ์ระหว่าง SIFT features จาก view ล่าสุดและเซตของจุดสังเกต.โดยใช้ k-Nearest Neighbors Search [105] ในการค้นหา (รูปที่ 4.3 (ข))
3. เซตของคู่ความสัมพันธ์ของจุดสังเกตในอดีตกับ feature ใน view ล่าสุด จะถูกใช้ในการประมาณการเลื่อนตำแหน่งของกล้อง โดยใช้ RANSAC และทำการกรองคู่ความสัมพันธ์ที่ความผิดพลาดทิ้ง (รูปที่ 4.3 (ค))



(ก) *(ข)* *(ค)*

รูปที่ 4.3 (ก) แสดงการเลือก view จำนวน k view ล่าสุด, (ข) แสดงจุดสังเกตที่ผูกกับ view ใน Sliding-window, (ค) แสดงการประมาณการเลื่อนตำแหน่งของ view ล่าสุด

Transformation Estimation using RANSAC

กำหนดให้ $u = \{u_0, u_1, \dots, u_m\}$ เป็น unit vector ของ features ที่ตรวจหาได้ในภาพล่าสุด และให้ $v = \{v_0, v_1, \dots, v_n\}$ เป็นตำแหน่งจุดสังเกตในอดีต จุดสังเกตุในอดีตจะแบ่งได้เป็นสองส่วนด้วยกันคือ จุดสังเกตที่รู้ค่าความลึก และจุดสังเกตที่ไม่รู้ค่าความลึก ซึ่งจุดสังเกตที่ไม่รู้ค่าความลึกนั้น จะเป็นจุดสังเกตที่เพิ่งตรวจวัดได้ในเฟรมก่อนหน้าทำให้ยังไม่สามารถประมาณค่าความลึกได้ สำหรับจุดสังเกตที่ไม่รู้ค่าความลึกจะกำหนดให้ v_j มีค่าเป็น measurement unit vector ของจุดสังเกต การประมาณตำแหน่งของกล้องนั้นเราจะใช้เฟรมก่อนหน้าเป็นเฟรมอ้างอิง โดยจะแปลงโคออดิเนตของจุดสังเกตที่รู้ค่าความลึกให้มาอยู่ในโคออดิเนตของเฟรมก่อนหน้า

จากการค้นหาความสัมพันธ์ระหว่าง features ใน view ล่าสุดกับจุดสังเกตในอดีตด้วย k-Nearest Neighbors Search จะได้คู่ features $p_{ij} = (u_i, v_j)$ จำนวนมากซึ่งจะใช้ในการหา hypothesis ในการคำนวนการเลื่อนตำแหน่งของกล้องด้วย RANSAC โดยขั้นตอนการทำงานประกอบด้วย

1. ทำการสุ่มคู่ features p_{ij} จำนวน 4 คู่ โดยเลือกเฉพาะคู่จุดสังเกต v_j ที่รู้ค่าความลึกหลังจากนั้นจึงประมาณตำแหน่งของกล้อง ($M = [R|t]$) ด้วยอัลกอริทึม perspective 3-point [122]

2. คำนวนหา error ของคู่จุดสังเกตทั้งหมดโดยแบ่งเป็นสองส่วนคือ error ของจุดสังเกตที่รู้ค่าความลึก และ error ของจุดสังเกตที่ไม่รู้ค่าความลึก

   - error ของจุดสังเกตที่รู้ค่าความลึก คำนวนได้จากการประมาณค่าการวัด ($\tilde{u}_i$) ด้วยโมเดลการวัดจากนั้นจึงหา error (e_{ij}) จากมุมความต่างระหว่างค่าประมาณการวัด ($\tilde{u}_i$) และค่าการวัดจริง (u_i)

$$\tilde{u}_i = \rho(M \oplus v_j), \quad \rho = 1/\left\|M \oplus v_j\right\|_2$$

$$e_{ij} = 1 - \langle \tilde{u}_i, u_i \rangle$$

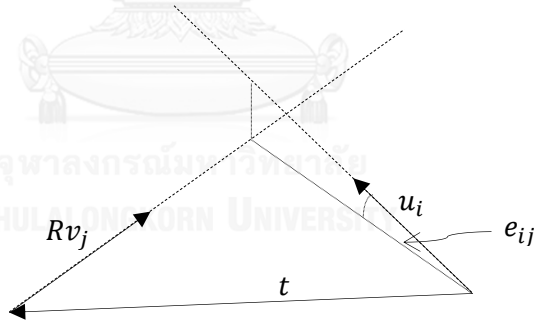
เมื่อ $\langle .,. \rangle$ แทน dot product และให้ error เท่ากับ $1 - \cos(\theta)$ เมื่อ θ เป็นมุมระหว่างค่าประมาณการวัด ($\tilde{u}_i$) และค่าการวัดจริง (u_i)

- error ของจุดสังเกตที่ไม่รู้ค่าความลึกคำนวนได้จากจาก*รูปที่ 4.4* โดยจากรูปแสดงภาพของ vector u_i และ v_j เนื่องจากความคาดเคลื่อนในการประมาณการเคลื่อนที่ จึงทำให้ vector ทั้งสองไม่ตัดพอดี ตำแหน่งของจุดสังเกตจะประมาณได้เป็นตำแหน่งที่ vector ทั้งสองเข้าใกล้กันมากสุด และ error (e_{ij}) จะหาจากมุมที่ vector u_i ทำกับจุดตัดบน vector v_j

$$n = \frac{t \times Rv_j}{\left\|t \times Rv_j\right\|_2}$$

$$e_{ij} = 1 - \|n \times u_i\|_2$$

ขั้นตอนการคำนวนจะเริ่มจาก หา normal ของ plane ที่เกิดจาก vector Rv_j และ t จากนั้นจึงหา error จากสูตร $\cos(\theta) = \sin(90 - \theta)$ ซึ่งขนาดของ cross product ระหว่าง normal (n) และ u_i ก็คือค่า $\sin(90 - \theta)$ นั่นเอง



รูปที่ 4.4 การหา error ของจุดสังเกตที่ไม่รู้ค่าความลึก

3. ตรวจหา inliner โดยเลือกเฉพาะ error ที่มีค่าต่ำกว่า threshold จากนั้นจะลบคู่จุดสังเกตที่มีความซ้ำซ้อน เช่น ในหนึ่ง feature อาจจับคู่ได้หลายจุดสังเกต โดยขั้นตอนการลบคู่จุดสังเกตจะเรียงลำดับ inliner โดยพิจารณาจากค่า error น้อยไปมาก จากนั้นสำหรับคู่จุดสังเกตที่มีความซ้ำซ้อนก็จะเลือกคู่จุดสังเกตที่มี error น้อยกว่า

4. ทำซ้ำขั้นตอนที่ 1 จนได้ hypothesis ที่มี inliner มากสุด
5. ประมาณตำแหน่งของกล้อง ($M = [R|t]$) อีกครั้งโดยใช้คู่จุดสังเกตใน inliner ทั้งหมดด้วยอัลกอริทึม perspective n-point [123]

หลังจากประมาณการเคลื่อนที่จากข้อมูลภาพได้แล้วจึงทำการเพิ่ม โหนดใหม่ลงใน graph โดยให้เป็นลูกของ โหนดก่อนหน้าและเพิ่มจุดสังเกตที่เกิดจาก feature ใหม่ที่ไม่ถูกจับคู่ลงใน view ล่าสุดเพื่อรอการจับคู่ในการทำงานรอบถัดไป นอกจากนี้จะลบจุดสังเกตที่ไม่รู้ค่าความลึกใน view ก่อนหน้าทิ้งไป

4.2 Sliding-Window Bundle Adjustment

การทำงานของ Sliding-Window Bundle Adjustment ในหัวข้อนี้ จะเป็นการ optimize ตำแหน่งของกล้องและตำแหน่งของจุดสังเกตไปพร้อมๆ กันเพื่อหาค่าเหมาะสมสุด โดยเนื่องจากเหตุผลด้านประสิทธิภาพการทำงาน เราจะไม่ optimize ตำแหน่งของกล้องและตำแหน่งของจุดสังเกตทั้งหมดในแผนที่ แต่จะ optimize แค่ตำแหน่งของกล้องใน n view ล่าสุด และจุดสังเกต ที่ผูกกับ view เท่านั้น (ขั้นตอนในการปรับแก้ view ทั้งหมดจะอธิบายในบทที่ 5)

กำหนดให้ $x = \{x_0, x_1, \dots, x_n\}$ เป็นเซตของตำแหน่งของกล้อง n view ล่าสุด โดยที่ $x_i = [R_i|t_i]$ เป็น transformation matrix ของ view ที่ i ประกอบด้วย translation vector (t_i) และ rotation matrix (R_i) กำหนดให้ $d = \{d_0, d_1, \dots, d_m\}$ เป็นเซตของค่า invert depth ของจุดสังเกตทั้งหมด และกำหนดให้ z_{ij} เป็นค่าการวัดของจุดสังเกตที่ j ในมุมมองของ view i ซึ่งมี reference view ที่ n โมเดลการวัดสามารถเขียนได้เป็น

$$p_j = x_n \oplus (r_j/d_j)$$

$$\hat{z}_{ij} = z(x_i, d_j) = \frac{p_j \ominus x_i}{\|p_j \ominus x_i\|_2}$$

เมื่อ r_j เป็น unit vector แสดง direction ของจุดสังเกตใน reference view ที่ n

โมเดลการวัดข้างต้นจะการคำนวนหาตำแหน่งของจุดสังเกตใน world coordinate (p_j) โดยคำนวนจาก feature direction (r_j) หารด้วย invert depth (d_j) แล้วแปลงไปยัง world coordinate จากนั้นจึงประมาณข้อมูลการวัดโดยการ แปลงกลับมายัง local coordinate ของ view i และ normalize ให้เป็น unit vector

Loglikelihood l_{ij} ของ measurement z_{ij} แสดงได้เป็น

$$l_{ij} \propto \left[z_{ij} - \hat{z}_{ij}\right]^T \Sigma_{ij}^{-1} [z_{ij} - \hat{z}_{ij}]$$

เมื่อ Σ_{ij} เป็น error covariance ของการวัดค่า

การหา maximum likelihood ของข้อมูลการวัดทั้งหมดในสมการข้างต้นจะสามารถหาค่าเหมาะสุดของ x^* และ d^* ได้ โดยสามารถเปลี่ยนเป็นการหา minimizes negative loglikelihood $F(x, d)$ ได้เป็น

$$F(x, d) = \sum_{\langle i,j \rangle \in \mathcal{C}} e_{ij}^T \Sigma_{ij}^{-1} e_{ij}$$

$$e_{ij} \stackrel{\text{def}}{=} e\left(x_i, d_j, z_{ij}\right) = z_{ij} - \hat{z}_{ij}$$

กำหนดให้ e_{ij} เป็น error function ของค่าการวัด, $\mathcal{C}$ เป็น set ของคู่ ลำดับการวัดค่าของจุดสังเกตที่ j ในมุมมองของ view i ซึ่งคำตอบที่ต้องการประมาณจะได้เป็น

$$(x^*, d^*) = \underset{(x,d)}{\operatorname{argmin}} F(x, d)$$

Optimization

ในการแก้ปัญหานี้จะใช้วิธี Gauss-Newton ในการแก้ปัญหา และเนื่องจากเซตของตำแหน่งของกล้อง (x) span บน non-Euclidean spaces ดังนั้นจะใช้การ optimization บน manifold ดังนี้

กำหนดให้นิยาม operator $\boxplus$ เป็น operator ที่แปลง local variation Δx บน Euclidean space ไปยัง variation บน manifold $\Delta x \longmapsto x \boxplus \Delta x$ เราจะประมาณ error function ใหม่ได้เป็น

$$\check{e}_{ij}(\Delta x_i, \Delta d_j) \stackrel{\text{def}}{=} e_{ij}(\breve{x}_i \boxplus \Delta x_i, \breve{d}_j + \Delta d_j)$$

$$\simeq \check{e}_{ij} + J_{ij}\left[\Delta x_i \;\; \Delta d_j\right]^T$$

$\Delta x_i, \Delta d_j$ คือค่าความเปลี่ยนแปลงปริมาณน้อย ๆ รอบ ๆ ค่า initial guess $\breve{x}_i$ และ $\breve{d}_j$ โดย Δx_i เป็น vector 6 มิติ $\Delta x_i = [\Delta t_i \; \Delta r_i]^T$ โดยที่ Δt_i แทนการเลื่อนตำแหน่งและ Δr_i เป็น Rodrigues rotation ดังนั้น operator $\boxplus$ ก็คือการแปลง vector 6 มิติ Δx_i ไปเป็น transformation matrix จากนั้นจึงคูณเข้ากับพารามิเตอร์ $\breve{x}_i$ ส่วนค่า invert depth (d_j) จะใช้ operator การบวกกับ Δd_j ตามปรกติ

สำหรับ Jacobian J_{ij} ของ error function $\check{e}_{ij}(\Delta x, \Delta d)$ จะหาได้จาก partial derivatives ของ error function เทียบ $\Delta x_i, \Delta d_j$ ที่จุด $\Delta x_i = 0$ และ $\Delta d_j = 0$

$$J_{ij} = \left(\left.\frac{\partial \check{e}_{ij}(\Delta x, \Delta d)}{\partial \Delta x_i}\right|_{\Delta x_i = 0} \quad \left.\frac{\partial \check{e}_{ij}(\Delta x, \Delta d)}{\partial \Delta d_j}\right|_{\Delta d_j = 0} \right)$$

รายละเอียดของการหา Jacobian นั้น อธิบายในภาคผนวก ค

เราสามารถหาค่า $\Delta x^*, \Delta d^*$ ที่ minimize Function $F(x, d)$ ได้จากการแก้ linear System

$$\Lambda \cdot [\Delta x^* \;\; \Delta d^*]^{\mathrm{T}} = \eta$$

เมื่อ

$$\Lambda = \sum_{\langle i,j \rangle \in \mathcal{C}} J_{ij}^T \Sigma_{ij}^{-1} J_{ij}$$

$$\eta = -\sum_{\langle i,j \rangle \in \mathcal{C}} e_{ij}^T \Sigma_{ij}^{-1} J_{ij}$$

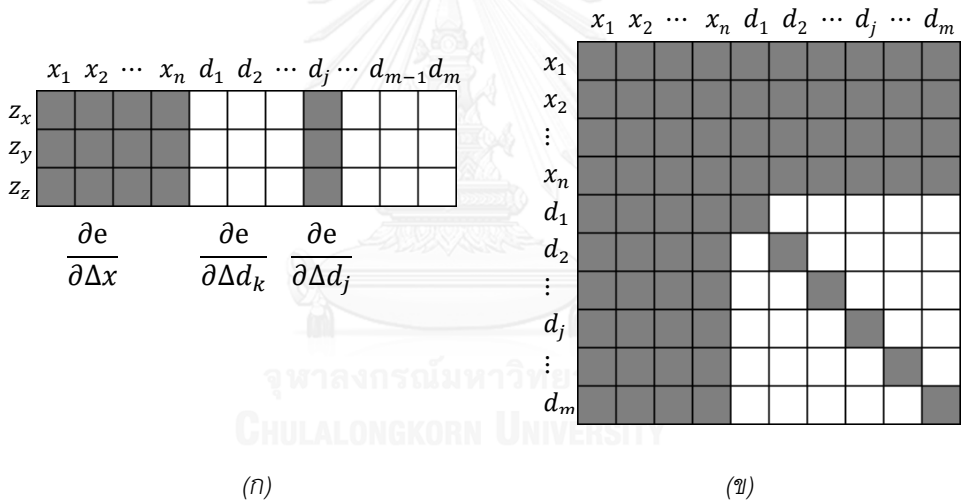
เมื่อคำนวนค่า $\Delta x^*, \Delta d^*$ ได้แล้ว จึงนำไปปรับแก้ initial guess และทำงานวนซ้ำจนกว่าค่าจะลู่เข้าสู่คำตอบที่ต้องการ

$$x^* = \breve{x} \boxplus \Delta x^*$$

$$d^* = \breve{d} + \Delta d^*$$

Marginalization

กระบวนการ optimization ข้างต้นจะหาคำตอบได้จากการแก้ linear system $\Lambda \cdot [\Delta x^* \ \Delta d^*]^{\mathrm{T}} = \eta$ โดยค่า Information Matrix Λ คำนวนจากผลคูณ Transposed ของ Jacobian J_{ij} ซึ่งโครงสร้างของ Jacobian Matrix J_{ij} แสดงได้ดัง*รูปที่ 4.5* (ก) จะมีลักษณะ sparse โดยจะประกอบด้วยค่า 0 จำนวนมากจากการหา partial derivatives ของ error function ณ คอลัมน์ที่ d_k เนื่องจากว่า error ของจุดสังเกต j ใด ๆ จะไม่ขึ้นกับตำแหน่งจุดสังเกต k เมื่อ $k \neq j$ ดังนั้นเมื่อนำมาคำนวนเป็น Information Matrix จะได้ Hessian Matrix ที่มีลักษณะ sparse ดัง*รูปที่ 4.5* (ข)



(ก) *(ข)*

รูปที่ 4.5 (ก) ตัวอย่าง Jacobian Matrix (ข) ตัวอย่าง information matrix ช่องสีเทาคือช่องที่มีค่า ส่วนช่องสีขาวคือช่องที่เป็นศูนย์

การแก้สมการ linear system ข้างต้น สามารถแก้ได้อย่างมีประสิทธิภาพโดยใช้ sparse linear solver อย่างไรก็ดีในทางปฏิบัตินั้น ยังมีวิธีแก้อีกวิธีหนึ่งที่มีประสิทธิภาพดีกว่า นั่นคือการลดขนาด Information Matrix Λ ด้วยการ marginalized out ตัวแปรสุ่ม (Random variable) d_j

$$p(x_i) = \int_y p(x_i, y) dy$$

โดยจะคำนวน Information Matrix Λ และ Information Vector η ใหม่ ได้ดังนี้

$$\Lambda' = \Lambda_x - \sum_j \Lambda_{x,d_j} \Lambda_{d_j}^{-1} \Lambda_{d_j,x}$$

$$\eta' = \eta_x - \sum_j \Lambda_{x,d_j} \Lambda_{d_j}^{-1} \eta_{d_j}$$

จากนั้นจึงแก้สมการ linear System เพื่อหาค่า Δx^*

$$\Lambda' \cdot \Delta x^{*\mathrm{T}} = \eta'$$

แล้วจึงกู้คืนค่าความเปลี่ยนแปลง invert depth (Δd_j) จากสมการ

$$\Delta d_j^T = \Lambda_{d_j}^{-1} (\eta_{d_j} - \Lambda_{d_j,x} \cdot \Delta x^{*\mathrm{T}})$$

Outlier Removal

กระบวนการ optimization ข้างต้นจะเป็นการหาค่าเหมาะสมสุดสำหรับตำแหน่งของกล้องจำนวน k views ล่าสุดและค่า invert ของความลึกของจุดสังเกตเทียบกับ reference view ของจุดสังเกตนั้น ๆ โดยอาศัยข้อมูลการวัด จาก n view ล่าสุด อย่างไรก็ดีการหา data association ระหว่างข้อมูลการวัดและจุดสังเกตที่หาได้จาก SIFT อัลกอริทึม ย่อมมีความผิดพลาดเกิดขึ้นได้ ซึ่งความผิดพลาดจะทำให้การประมาณตำแหน่งของกล้องและจุดสังเกตมีความคลาดเคลื่อน ดังนั้นจึงต้องมีขั้นตอนในการตรวจหาคู่จุดสังเกตที่มีความผิดพลาดและลบคู่จุดสังเกตนั้นทิ้งไป

สำหรับจุดสังเกตหนึ่งซึ่งถูกกำหนดด้วย unit vector ใช้บอกทิศทางของจุดสังเกตเทียบกับ reference view และค่า invert depth ตำแหน่งของจุดสังเกตในสามมิติ (p_j) หาได้จาก

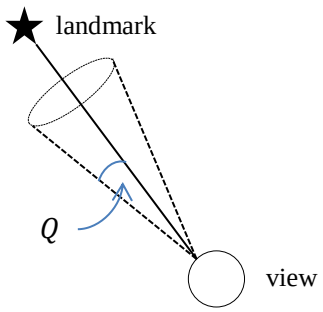
$$p_j = x_n \oplus (r_j / d_j)$$

เมื่อ $x_i = [R_i | t_i]$ เป็น transformation matrix ของ view ที่ i ประกอบด้วย translation (t_i) และ rotation (R_i), r_j เป็น unit vector แสดง direction ของจุดสังเกตใน reference view ที่ n และ d_j เป็นค่า invert depth

ค่าประมาณข้อมูลการวัดจุดสังเกต p_j จากมุมมองของ view ที่ i หาได้จาก

$$\hat{z}_{ij} = z(x_i, d_j) = \frac{p_j \ominus x_i}{\| p_j \ominus x_i \|_2}$$

error ในการวัดจุดสังเกต จะอธิบายอยู่ในรูปของคลาดเคลื่อนขององศาทิศทางการวัด โดยกำหนดให้เป็น Gaussian error มี mean เป็น 0 และ variance เป็น Q, error cone ของค่าการวัดแสดงได้ดัง*รูปที่ 4.6* (ก)



รูปที่ 4.6 error cone ของค่าการวัด

ขั้นตอนการตรวจหาคู่จุดสังเกตที่มีความผิดพลาด จะสมมุติว่าข้อมูลการวัดที่มีความผิดพลาดนั้น มีจำนวนน้อยมากเมื่อเทียบกับข้อมูลการวัดทั้งหมด ดังนั้นหลักการที่จะใช้ในการตรวจหาความผิดพลาด จะเทียบความคลาดเคลื่อน (คิดเป็นองศาการวัด) ขององศาระหว่างค่าประมาณข้อมูลการวัด และค่าการวัดจริงว่าอยู่ในขอบเขตที่กำหนดหรือไม่ ในกรณีที่ค่าการวัดในมีความคลาดเคลื่อนเกินขอบเขตที่กำหนดก็จะลบค่าการวัดนั้นทิ้งไป และถ้าหากจุดสังเกตมีค่าการวัดน้อยกว่าสองค่าก็จะลบจุดสังเกตนั้นทิ้งเสีย

ในขั้นตอนการทำ optimization ซึ่งจะทำงานแบบวนซ้ำจนกว่าค่าประมาณจะลู่เข้าสู่คำตอบ เราจะทำแทรกกระบวนการ Outlier Removal เข้าไปทุก iteration ซึ่งเมื่อข้อมูลการวัดที่ผิดพลาดถูกลบทิ้งไปในระหว่างกระบวนการ optimization ก็จะทำให้ได้คำตอบลู่เข้าสู่ค่าที่ถูกต้องในที่สุด

สำหรับกระบวนการ Outlier Removal นอกจากจะใช้ลบข้อมูลการวัดของคู่จุดสังเกตที่ผิดพลาดได้แล้ว ยังสามารถใช้ลบจุดสังเกตปลอมที่เกิดจากการตัดกันของวัตถุสองวัตถุที่อยู่ความลึกต่างระดับได้อีกด้วย

Management

เพื่อให้กระบวนการ optimization ข้างต้นทำงานได้อย่างมีประสิทธิภาพ และเพื่อให้ scale ของการประมาณตำแหน่งในแต่ละครั้งมีความสอดคล้องกัน อัลกอริทึมจะมีการตรึง view จำนวนหนึ่งไม่ให้มีการเลื่อนตำแหน่ง โดย view ที่ถูกตรึงนั้นจะพิจารณาจาก view ที่มีจำนวนจุดสังเกตที่สอดคล้องกับค่าการวัดล่าสุด น้อยกว่าเกณฑ์ที่กำหนด ทั้งนี้เพื่อให้การทำ Sliding-Window Bundle Adjustment มีความเสถียร และช่วยลดปัญหา Scale Drift

4.3 การจัดการปัญหาการเบี่ยงเบนขนาดของแผนที่

ปัญหา Scale Drift คืออาการเมื่อหุ่นยนต์เคลื่อนที่ไปข้างหน้าและสร้างแผนที่ แล้วแผนที่ ณ บริเวณต่าง ๆ มีขนาดเปลี่ยนแปลงไป ซึ่งปัญหา Scale Drift สามารถเกิดขึ้นได้กับ Monocular Visual SLAM ทั้งนี้เนื่องจากข้อมูลการวัดจากกล้องเพียงตัวเดียวไม่มีข้อมูลความลึกร่วมด้วย ทำให้ระบบไม่สามารถประมาณขนาดของแผนที่ที่ถูกต้องได้ โดยขนาดของแผนที่เริ่มต้นจะขึ้นกับค่า initial guess ของความลึกจุดสังเกตและสามารถเปลี่ยน โดยอาจจะใหญ่ขึ้นหรือเล็กลงก็ได้เมื่อหุ่นยนต์ทำงานไปเรื่อย ๆ

ปัญหา Scale Drift จัดเป็นปัญหาใหญ่สำหรับ Large Scale SLAM เนื่องจากว่าแผนที่ในแต่ละบริเวณมีโอกาสที่ขนาดจะแตกต่างกันได้มาก เพื่อจะแก้ไขปัญหานี้เราจึงต้องอาศัยข้อมูลสิ่งแวดล้อมเพิ่มเติมให้สามารถประมาณขนาดของแผนที่ได้อย่างถูกต้อง ยกตัวอย่างเช่น เราทราบว่ารถคันหนึ่งจะมีความสูงจากพื้นประมาณ 1.5 เมตร ดังนั้นเมื่อหุ่นยนต์เคลื่อนที่ไปพบเจอรถและสามารถตรวจหารถได้ หุ่นยนต์ก็จะอาศัยความรู้เกี่ยวกับความสูงรถมาใช้ในการปรับแก้ขนาดของแผนที่ให้มีความถูกต้อง

วิธีการในการปรับแก้ขนาดของแผนที่นั้น เราจะทำการเพิ่ม Constraint เข้าไปในขั้นตอน optimization โดยจาก error function ที่ใช้ในการประมาณค่าเหมาะสุด จะเป็น error function ของค่าการวัดเพียงอย่างเดียว ($e_{ij} = z_{ij} - \hat{z}_{ij}$) เราจะทำการเพิ่มอีกหนึ่ง error term ($e_{distance}$) เข้าไปใน error function โดยจะเป็น error ระหว่างระยะทางของ view สอง view ที่อยู่ไกลกันมากสุด กับระยะทางเป้าหมายซึ่งจะเป็นค่าที่คำนวนได้จากความรู้ในสิ่งแวดล้อม

$$e_{distance} = D - \|t_i - t_j\|_2$$

กำหนดให้ D เป็นระยะทางเป้าหมาย และ t_i, t_j เป็นตำแหน่งของ view ที่ i และ view ที่ j ตามลำดับ

ในการทดลองกับสิ่งแวดล้อมจริง เราจะใช้ข้อมูลความสูงระหว่างกล้องกับพื้น มาเป็นความรู้ที่จะใช้ในการกำหนดระยะเป้าหมาย เนื่องจากว่าในขั้นตอนการเก็บข้อมูลจะใช้วิธีให้มนุษย์เดินถือกล้องเพื่อเก็บข้อมูล ทำให้พอจะประมาณความสูงของกล้องได้จากระยะความสูงที่คนถือกล้อง

อัลกอริทึมในการตรวจหาพื้นจากจุดสังเกตในสามมิติ อธิบายรายละเอียดในภาคผนวก ข หลังจากคำนวนหาตำแหน่งพื้นได้แล้วก็สามารถประมาณระยะทางเป้าหมายได้ด้วยวิธีเทียบบัญญัติไตรยางค์

4.4 การทดลองการประมาณการเคลื่อนที่จากข้อมูลภาพ

การทดลองการประมาณการเคลื่อนที่จากข้อมูลภาพของกล้องมุมกว้าง จะใช้กล้องวิดีโอ Canon รุ่น LEGRIA HF M300 ใช้คู่กับเลนส์ fisheye รุ่น opteka ในการทดลอง โดยใช้คนเดินถือเคลื่อนที่ข้างหน้าไปตามสิ่งแวดล้อมและเก็บภาพมาประมวลผล ภาพที่ใช้ในการประมวลผลนั้นจะไม่ได้ใช้ภาพทั้งหมดจากวิดีโอ แต่จะวิเคราะห์การเปลี่ยนแปลงในภาพ แล้วจึงเลือกภาพที่มีการเปลี่ยนแปลงมากพอสมควรมาใช้ในการประมาณการเคลื่อนที่

สิ่งแวดล้อมที่ใช้ในการทดลองจะมีสองลักษณะได้แก่แบบ indoor และแบบ outdoor โดยแบบ indoor นั้นจะเป็นทางเดินภายในอาคาร ส่วนแบบ outdoor นั้นจะเป็นถนนระหว่างตึกในจุฬาลงกรณ์มหาวิทยาลัย ซึ่งภาพสิ่งแวดล้อมที่ใช้ในการทดลองจะแสดงได้ดังรูปที่ 4.7 (ก) และ (ข) ตามลำดับ

(ก)

(ข)

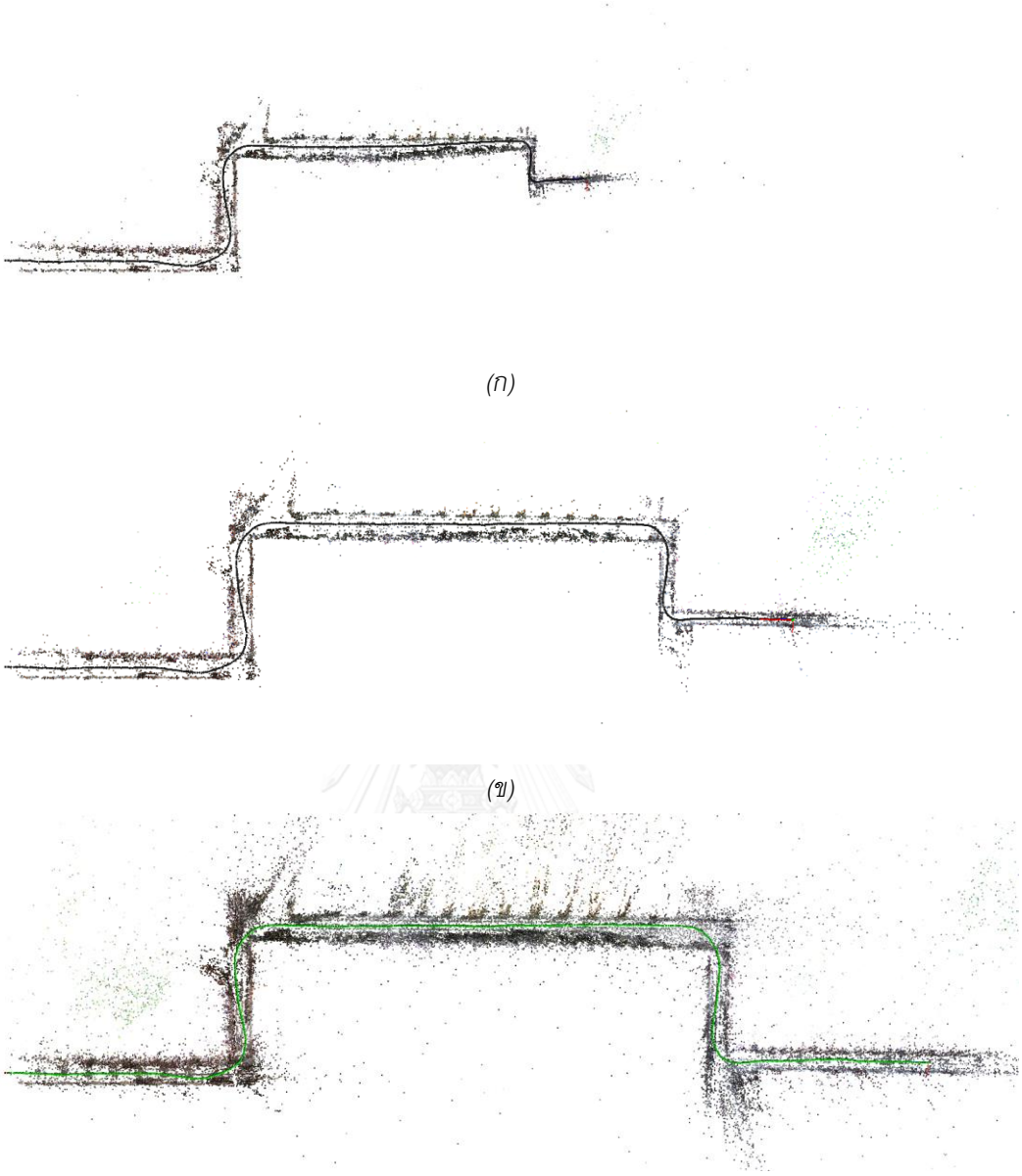
รูปที่ 4.7 ภาพสิ่งแวดล้อมที่ใช้ในการทดลอง (ก) ภาพภายในอาคาร (ข) ภาพภายนอกอาคาร

การเปรียบเทียบการประมาณการเคลื่อนที่นั้นจะเปรียบเทียบระหว่างสามอัลกอริทึม ได้แก่ EKF-base Visual Odometry, Sliding-Window Bundle Adjustment ซึ่งจะเป็นวิธีที่ใช้ในวิทยานิพนธ์นี้ และ Full Bundle Adjustment

ผลการทดลองสำหรับทางเดินภายในอาคารแสดงได้ดังรูปที่ 4.8 โดยรูปที่ 4.8 (ก) จะเป็นผลจาก EKF-base Visual Odometry, รูปที่ 4.8 (ข) จะเป็นผลจาก Sliding-Window Bundle Adjustment และรูปที่ 4.8 (ค) จะเป็นผลจาก Full Bundle Adjustment

สำหรับผลการทดลองสิ่งแวดล้อม outdoor แสดงได้ดัง*รูปที่ 4.9* โดยรูป (ก), (ข) และ (ค) จะเป็นผลการทดลองจาก EKF-based Visual Odometry, Sliding-Window Bundle Adjustment และ Full Bundle Adjustment ตามลำดับ

จากผลการทดลองในรูปที่ 4.8 และรูปที่ 4.9 นั้นจะเห็นได้ว่าการประมาณตำแหน่งของ EKF-base Visual Odometry และ Sliding-Window Bundle Adjustment จะมีอาการ Scale drift ซึ่งส่งผลในขนาดของแผนที่บริเวณท้ายๆ การเคลื่อนที่ โดยจะมีขนาดเล็กกว่าจุดเริ่มต้น เมื่อเทียบกับแผนที่ที่ประมาณได้จาก Full Bundle Adjustment ทั้งนี้เนื่องจากการไม่รู้ค่าความลึกของข้อมูลภาพ อย่างไรก็ดี วิธี Sliding-Window Bundle Adjustment นั้นมีความคลาดเคลื่อนในการประมาณตำแหน่งน้อยกว่าวิธี EKF-base Visual Odometry ซึ่งเป็นเพราะการทำ Local Bundle Adjustment จะช่วยให้การประมาณการเลื่อนตำแหน่งของกล้องมีความแม่นยำมากขึ้น

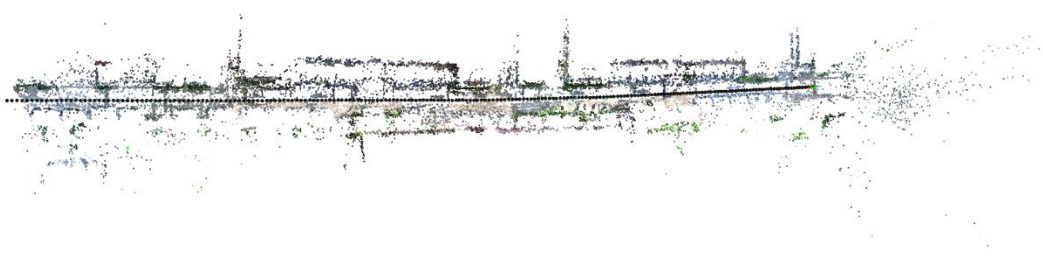


(ก)

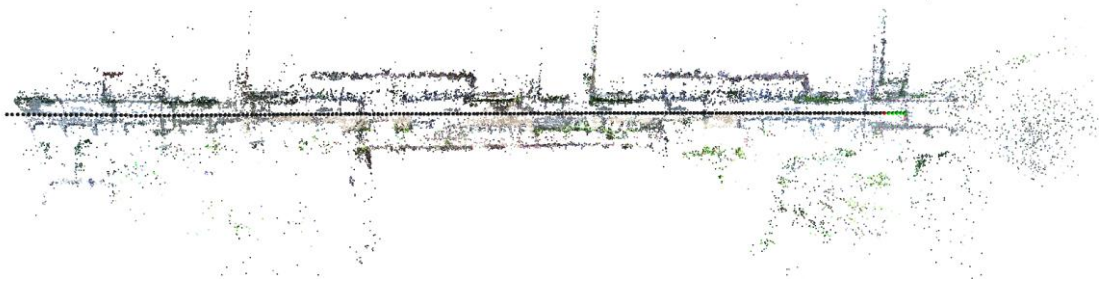
(ข)

(ค)

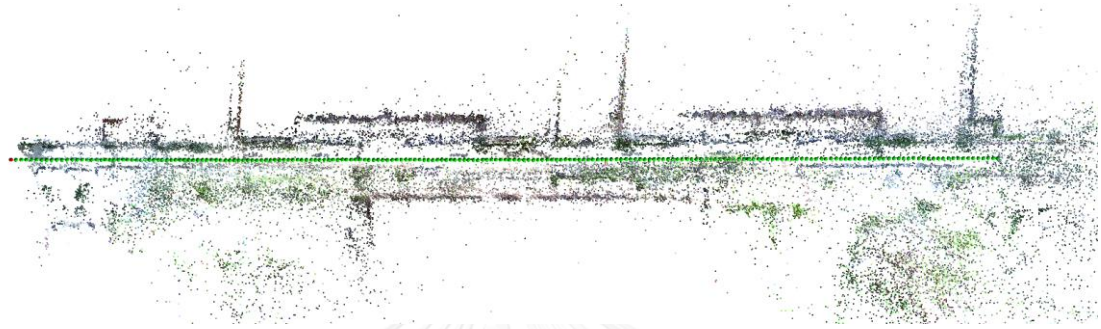
รูปที่ 4.8 ผลการทดลองการประมาณการเคลื่อนที่จากข้อมูลภาพสำหรับทางเดินภายในอาคาร



(ก)



(ข)

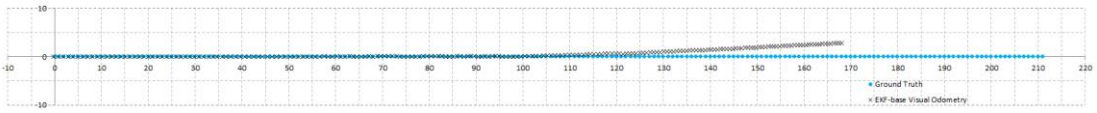


(ค)

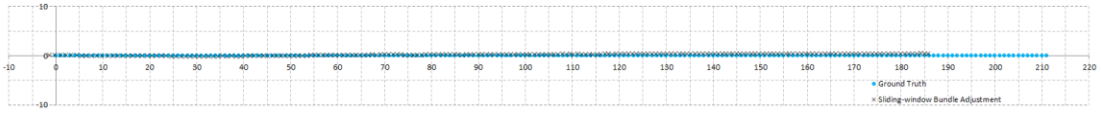
รูปที่ 4.9 ผลการทดลองการประมาณการเคลื่อนที่จากข้อมูลภาพสำหรับสิ่งแวดล้อมนอกอาคาร

4.4.1 การวัดผลความคลาดเคลื่อนการประมาณการเคลื่อนที่

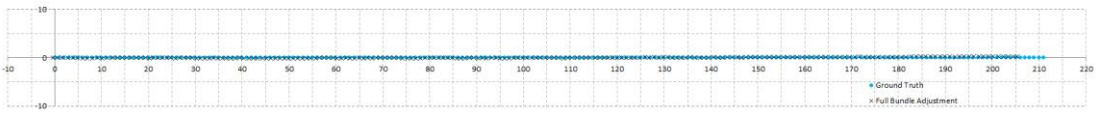
ในการวัดผลความถูกต้องการประมาณการเคลื่อนที่นั้นจะต้องใช้ข้อมูล ground truth ในการวัดเทียบ ซึ่งในที่นี้เราจะเทียบกับถนนซึ่งมีลักษณะเป็นเส้นตรงโดยข้อมูลภาพที่ใช้นั้น จะใช้การเคลื่อนกล้องไปทีละหนึ่งเมตรแล้วบันทึกภาพ จะได้การเคลื่อนที่เป็นเส้นตรงโดยมีระยะห่างระหว่าง view หนึ่งเมตร ผลลัพธ์จากการทำงานของทั้งสามอัลกอริทึมแสดงได้ดังรูปที่ 4.10 โดยรูป (ก), (ข) และ (ค) จะเป็นผลการทดลองจาก EKF-base Visual Odometry, Sliding-Window Bundle Adjustment และ Full Bundle Adjustment ตามลำดับ สำหรับตารางความคลาดเคลื่อนเทียบกับ ground truth แสดงได้ดังตาราง 4.1



(ก)



(ข)



(ค)

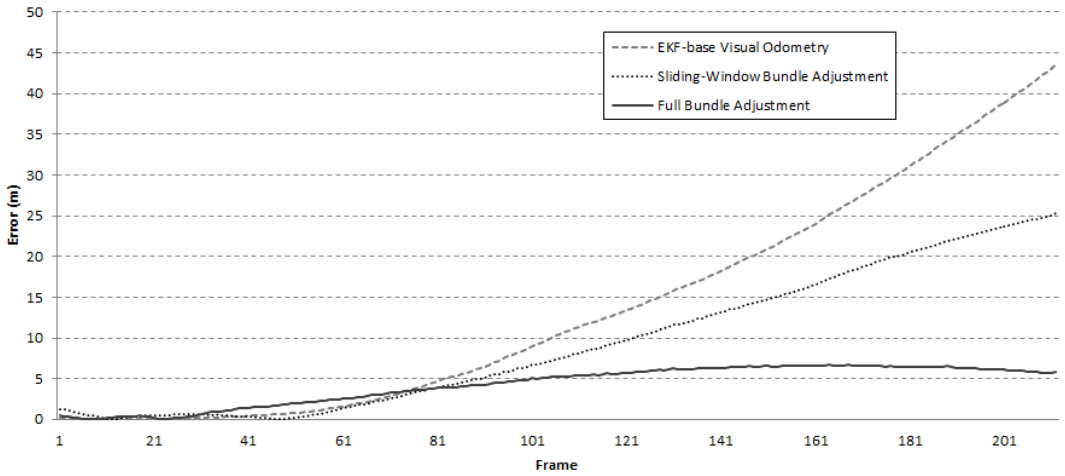
รูปที่ 4.10 กราฟเปรียบเทียบตำแหน่งของกล้องที่ประมาณได้จากอัลกอริทึมทั้งสามเทียบกับ ground truth

ตาราง 4.1 ตารางความคลาดเคลื่อนของการประมาณการเคลื่อนที่เฉลี่ยตลอดระยะเวลาการทำงานของทั้งสามอัลกอริทึมเทียบกับ ground truth

Method	Error (m)
EKF-base Visual Odometry	13.6875
Sliding-Window Bundle Adjustment	9.2735
Full Bundle Adjustment	4.2138

จากตารางความคลาดเคลื่อนแสดงให้เห็นว่าความคลาดเคลื่อนของ Full Bundle Adjustment นั้นมีความคลาดเคลื่อนน้อยสุดส่วน Sliding-Window Bundle Adjustment มีความคลาดเคลื่อนรองลงมา และ EKF-base Visual Odometry มีความคลาดเคลื่อนมากสุด อย่างไรก็ดีเมื่อพิจาณากราฟตำแหน่งผลลัพธ์ในรูปที่ 4.10 (ข) จะเห็นได้ว่าความคลาดเคลื่อนของ Sliding-Window Bundle Adjustment นั้นเกิดจาก Scale drift เสียเป็นส่วนใหญ่ ในขณะที่ความคลาดเคลื่อนของ EKF-base Visual Odometry จะมีความคลาดเคลื่อนของตำแหน่ง และทิศทางการเคลื่อนที่ร่วมด้วย

สำหรับกราฟแสดงความคลาดเคลื่อนของทั้งสามอัลกอริทึมต่อในแต่ละเฟรม แสดงได้ดังรูปที่ 4.11 โดยจะเห็นได้ว่าความคลาดเคลื่อนของ Sliding-Window Bundle Adjustment มีค่าสูงขึ้นเรื่อย ๆ ตามระยะทางที่กล้องเคลื่อนที่ ดังนั้นจึงต้องมีขั้นตอนการตรวจหา loop closure พร้อมกับการปรับแก้แผนที่ อันเป็นขั้นตอนสำคัญของ SLAM ต่อไป



รูปที่ 4.11 กราฟความคลาดเคลื่อนตำแหน่งของกล้องที่ประมาณได้จากอัลกอริทึมทั้งสามเทียบกับ ground truth

4.4.2 ประสิทธิภาพการทำงาน

คอมที่ใช้ในการทดลองนั้นจะใช้ CPU i7 3.40 GHz, RAM 8 GB และ GPU NVIDIA GTX 680 เวลาในการทำงานการประมาณการเคลื่อนที่นั้นแสดงได้ตามตาราง 4.2 โดยได้มีการแบ่งเวลาการทำงานออกเป็นสาม

ส่วนด้วยกันคือเวลาในการตรวจหา SIFT Features, เวลาในการทำ SIFT Feature Matching ด้วย k-Nearest Neighbors Search และเวลาในการ optimization สำหรับ Sliding-Window Bundle Adjustment

ตาราง 4.2 ตารางแสดงเวลาการทำงานในแต่ละขั้นตอนของการประมาณการเคลื่อนที่

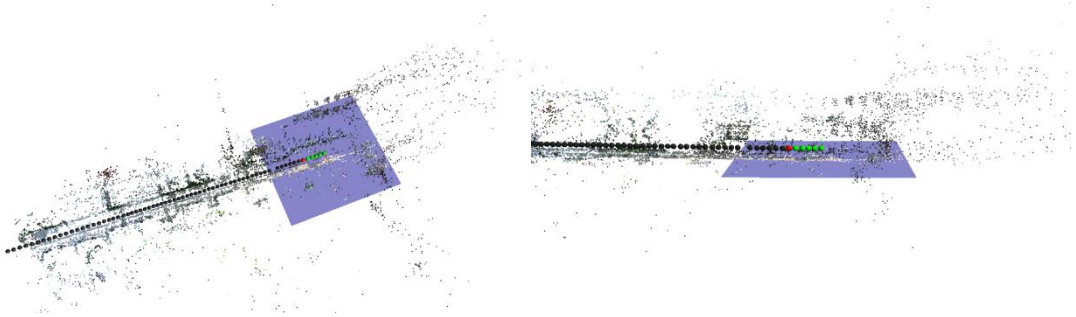
Process	Time Used (millisec)	
	Indoor Dataset	Outdoor Dataset
SIFT Feature Detector (GPU)	76.68	122.59
Feature Matching	42.21	134.77
Sliding-Window Bundle Adjustment	18.44	81.51

จะเห็นได้ว่าเวลาในการทำงานรวมต่อเฟรมจ ะอยู่ประมาณ 140-340 มิลลิวินาที ขึ้นอยู่กับจำนวน feature ในแต่ละเฟรม อย่างไรก็ดีขั้นตอนการตรวจหา SIFT Features นั้นเป็นการทำงานบน GPU ซึ่งสามารถแยกการทำงานได้อีก thread หนึ่ง ทำให้เวลาในการทำงานที่แท้จริงจะคิดจากเวลามากสุดที่ใช้ในแต่ละ thread เท่านั้นซึ่งจะอยู่ที่เฟรมละ 60-220 มิลลิวินาที

4.4.3 การวัดผลการแก้ปัญหา Scale Drift

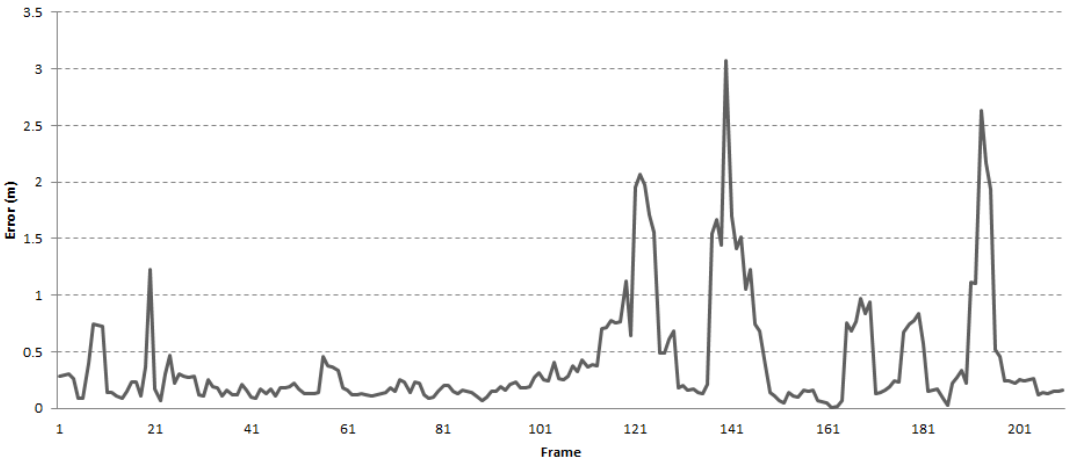
การแก้ปัญหา Scale Drift ในงานวิจัยนี้เราจะนำความรู้เกี่ยวกับความสูงของกล้องจากพื้นมาใช้ในการแก้ปัญหา โดยในการทดลองนั้นเราจะใช้มนุษย์ในการเดินถึงกล้องเพื่อถ่ายภาพบริเวณรอบ ๆ สิ่งแวดล้อม ดังนั้นเราจึงพอประมาณความสูงของกล้องจากพื้นได้ และในขั้นตอนการทำงานของอัลกอริทึม เราจะเพิ่มขั้นตอนการตรวจหาพื้นในกระบวนการทำงาน ทำให้เราสามารถประมาณอัตราส่วนระหว่างความสูงระหว่างกล้องกับพื้นในโลกจริง และความสูงของกล้องในแผนที่ที่สร้างได้ จากนั้นจึงใช้ข้อมูลอัตราส่วนนี้ในการปรับแก้แผนที่ให้มีความถูกต้อง

ผลลัพธ์การทำงานของอัลกอริทึมการตรวจหาพื้นจากชุดข้อมูลการเคลื่อนที่ outdoor แสดงได้ดังรูปที่ 4.12 โดยแผ่นสี่เหลี่ยมสีฟ้า แสดง plane ที่ตรวจวัดได้จากอัลกอริทึมที่ได้นำเสนอ สำหรับการวัดผลอัลกอริทึมการตรวจหาพื้นนั้น เราจะหา Ground Truth ของพื้นในแผนที่ได้โดยการใช้มนุษย์ในการสังเกต และกำหนดค่าตำแหน่งพื้นที่เหมาะสมในแผนที่ จากนั้นจึงเปรียบเทียบความต่างระหว่างพื้นที่ตรวจหาได้จากอัลกอริทึมและข้อมูล Ground Truth โดยในการเปรียบเทียบนั้นจะทำการสุ่มจุดจากพื้นซึ่งตรวจหาได้จากอัลกอริทึม จากนั้นจึงวัดระยะห่างระหว่างจุดถึง พื้น Ground Truth จะได้กราฟความคลาดเคลื่อนแสดงได้ดังรูปที่ 4.13 โดยมีค่าความคลาดเคลื่อนเฉลี่ยอยู่ที่ 0.4077 เมตร



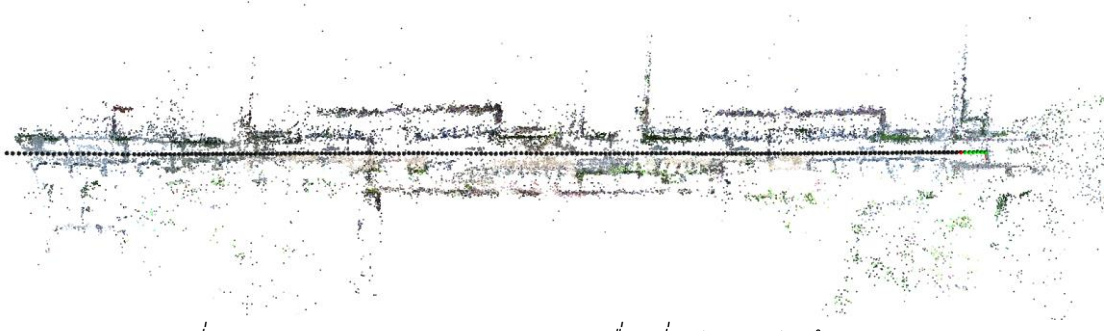
(ก) (ข)

รูปที่ 4.12 ผลลัพธ์การทำงานของอัลกอริทึมการตรวจหาพื้น



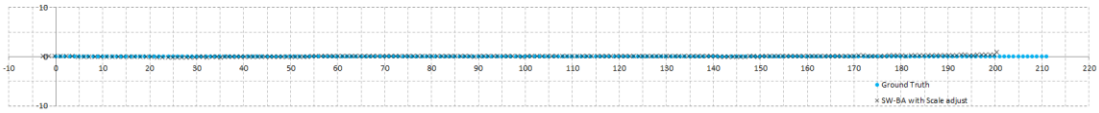
รูปที่ 4.13 กราฟความคลาดเคลื่อนอัลกอริทึมการตรวจหาพื้น

หลังการตรวจหาพื้นและคำนวนอัตราส่วนที่ใช้ในการปรับแก้แผนที่ได้แล้วก็จะนำค่าที่คำนวนได้ไปใช้ในการปรับ scale ในการประมาณการเคลื่อนที่ในรอบถัดไป ซึ่งจะทำให้สามารถลดปัญหา Scale Drift ได้ โดยผลการทดลองในชุดข้อมูล outdoor นั้นสามารถแสดงได้ในรูปที่ 4.14 ซึ่งจากรูปจะเห็นได้ว่า เมื่อนำข้อมูลความสูงของกล้องจากพื้นมาใช้ร่วม สามารถลดปัญหา Scale Drift ได้เมื่อเปรียบเทียบกับการทำงานของ Sliding-Window Bundle Adjustment เพียงอย่างเดียว (ในรูปที่ 4.9 (ข)) และได้ผลลัพธ์ใกล้เคียงกับการทำงานของ Full Bundle Adjustment มากขึ้น (เปรียบเทียบกับรูปที่ 4.9 (ค))

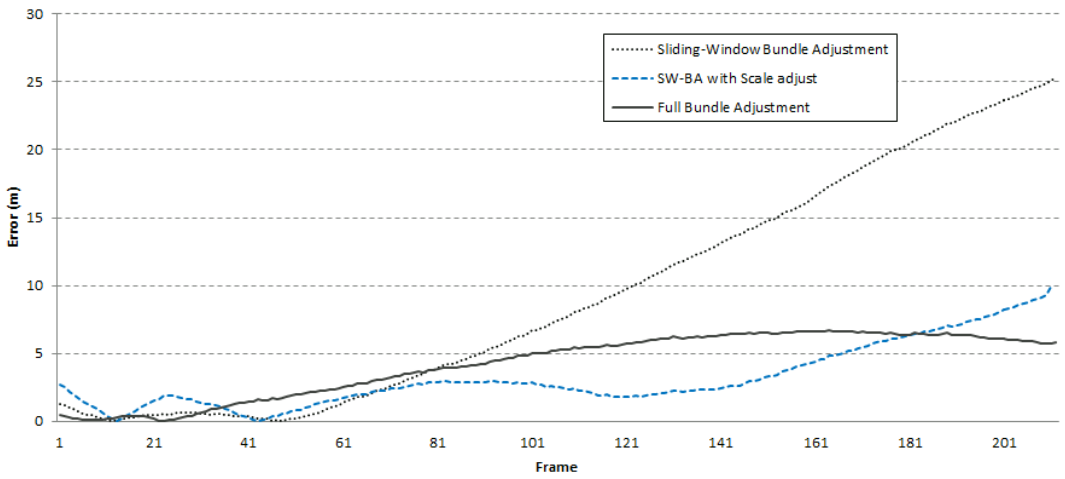


รูปที่ 4.14 ผลการทดลองการประมาณการเคลื่อนที่หลังการปรับแก้ Scale Drift

ในรูปที่ 4.15 จะแสดงตำแหน่งที่ของกล้องจากอัลกอริทึมซึ่งแก้ปัญหา Scale Drift แล้ว เทียบกับตำแหน่ง Ground Truth ซึ่งหาได้จากการวัดการเคลื่อนที่ในสิ่งแวดล้อมจริงไปทีละหนึ่งเมตร ความคลาดเคลื่อนของอัลกอริทึมแสดงได้ดังกราฟในรูปที่ 4.16 โดยจากกราฟจะสังเกตได้ว่าความคลาดเคลื่อนหลังจากแก้ปัญหา Scale Drift จะมีขนาดไล่เลี่ยกับความคลาดเคลื่อนของ Full Bundle Adjustment โดยลักษณะของกราฟจะไม่ได้มีลักษณะความคลาดเคลื่อนเพิ่มสูงขึ้นเรื่อย ๆ ตามเวลา แต่จะมีช่วงที่ความคลาดเคลื่อนลดต่ำลงมาเป็นระยะ ซึ่งเกิดจากการทำงานการปรับแก้ Scale Drift



รูปที่ 4.15 กราฟเปรียบเทียบตำแหน่งของกล้องหลังการปรับแก้ Scale Drift เทียบกับ ground truth



รูปที่ 4.16 กราฟความคลาดเคลื่อนตำแหน่งของกล้องที่ประมาณได้จากอัลกอริทึม Sliding-Window Bundle Adjustment (ก), Sliding-Window Visual Odometry with sacle adjust (ข) และ Full Bundle Adjustment (ค) เทียบกับ ground truth

บทที่ 5
การปรับแก้แผนที่เพื่อหาค่าเหมาะสุด (Back End)

หลังจากการประมาณการเคลื่อนที่จากข้อมูลภาพด้วยวิธี Sliding-Window Bundle Adjustment ในหัวข้อที่ 4 view ล่าสุดจะถูกเพิ่มเป็นโหนดในกราฟโดยกำหนดให้เป็นโหนดลูกของ view ก่อนหน้า รวมถึงได้มีการเพิ่ม edge ซึ่งเป็น odometry constraint สำหรับบรรยายการกระจายความน่าจะเป็นของตำแหน่งสัมพัทธ์ (relative transformation) ระหว่าง view ล่าสุดและ view ก่อนหน้า จะได้ต้นไม้ความสัมพันธ์ของโหนดซึ่งใช้บรรยายตำแหน่งของ view ต่าง ๆ เทียบกับ view พ่อ

จากนั้นเมื่อหุ่นยนต์ได้เคลื่อนที่กลับมายังจุดเดิมที่เคยผ่าน และสมมุติให้ระบบตรวจพบ loop closure ระหว่าง node x_i และ node x_j ระบบจะทำการเพิ่ม loop closure constraint ระหว่าง x_i และ x_j และการทำงานส่วน Back End จะทำหน้าที่ในการปรับแก้แผนที่ซึ่งในที่นี้จะเรียกว่า Pose Graph Optimization เพื่อหาค่าเหมาะสมสุดสำหรับแต่ละ View Pose

5.1 Pose Graph Optimization

Pose Graph Optimization เป็นกระบวนการในการปรับแก้ตำแหน่งของ view ในกราฟ โดยจะเป็นการหาค่าเหมาะสุดที่ทำให้ error สำหรับทุก constraint มีค่าต่ำสุด

กำหนดให้ $x = \{x_0, x_1, \dots, x_n\}$ เป็นเซตของตำแหน่งของกล้องที่ node ต่าง ๆ ใน world coordinate โดยตำแหน่งของกล้อง x_i จะบรรยายด้วย vector 7 มิติ $x_i = [t_i \;\; q_i]^T$ โดย t_i แทนการเลื่อนตำแหน่ง q_i เป็น quaternion rotation และ u_{ij} เป็น constraint ระหว่าง node i และ node j ซึ่งเป็น relative transformation ของ node j เทียบ node i ในที่นี้จะเรียกว่า virtual measurement โดยกำหนดให้ $\hat{u}_{ij}(x_i, x_j)$ เป็นค่า virtual measurement function

$$\hat{u}_{ij}(x_i, x_j) = x_j \ominus x_i$$

อธิบายได้ว่า ค่าประมาณ virtual measurement $\hat{u}_{ij}$ หาได้จาก invert pose x_i คูณกับ pose x_j

error ของ virtual measurement คือค่าความต่างระหว่าง virtual measurement กับค่าประมาณจาก virtual measurement function

$$e_{ij}(x_i, x_j) = u_{ij} - \hat{u}_{ij}(x_i, x_j)$$

และ Loglikelihood l_{ij} ของ virtual measurement u_{ij} แสดงได้เป็น

$$l_{ij} \propto e_{ij}(x_i, x_j)^T \Sigma_{ij}^{-1} e_{ij}(x_i, x_j)$$

เมื่อ Σ_{ij} เป็น error covariance ของ virtual measurement ซึ่งอธิบายวิธีในการคำนวนในหัวข้อ 5.1.1

เป้าหมายของการหา maximum likelihood ก็คือการหาค่า x^* ที่มีค่าเหมาะสมสุด ซึ่งจะเทียบได้กับการหา minimizes negative loglikelihood $F(x)$ ของข้อมูลการวัดทั้งหมด

$$F(x) = \sum_{\langle i,j \rangle \in \mathcal{C}} e_{ij}^T \Sigma_{ij}^{-1} e_{ij}$$

$$x^* = \underset{x}{\operatorname{argmin}} F(x)$$

กำหนดให้ $\mathcal{C}$ เป็น set ของคู่ indices สำหรับ constraint ทั้งหมด

การหาค่าเซตตำแหน่งของกล้อง x^* ที่ minimizes negative loglikelihood $F(x)$ หาได้จากการแก้ linear System

$$H\Delta x^* = -b$$

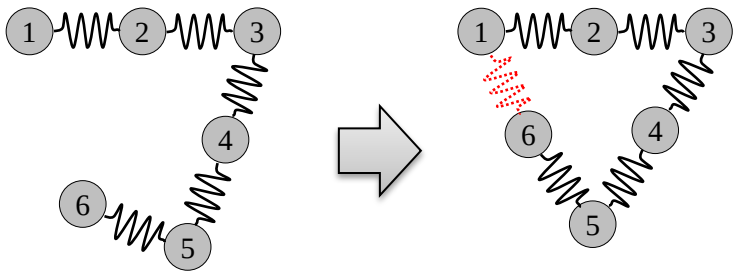
โดยที่ $\mathrm{H} = \sum_{\langle i,j \rangle \in \mathcal{C}} J_{ij}^T \Sigma_{ij}^{-1} J_{ij}$ และ $b = \sum_{\langle i,j \rangle \in \mathcal{C}} e_{ij}^T \Sigma_{ij}^{-1} J_{ij}$

เมื่อ J_{ij} เป็น Jacobian ของ error function e_{ij} รายละเอียดของการหา Jacobian นั้น อธิบายในภาคผนวก ง

สำหรับการแก้สมการ linear System ข้างต้นสามารถหาคำตอบได้โดยใช้วิธี Gauss-Newton และเนื่องจากว่าค่า error e_{ij} จะขึ้นกับตำแหน่งของ node i และ node j เท่านั้นทำให้ Jacobian J_{ij} เป็น sparse matrix ซึ่งในกรณีทั่วไปที่การเคลื่อนที่ของกล้องไม่ซับซ้อนมากนัก Hessian H จะเป็น sparse matrix ด้วย ทำให้มีวิธีที่มีประสิทธิภาพในการแก้ linear System เช่นการใช้ sparse Cholesky factorization จะลดเวลาในการคำนวนจากการแก้ linear System แบบปรกติจาก $O(n^3)$ เหลือ $O(n^2)$

5.1.1 Virtual Measurement Covariance

แนวคิดของการการปรับแก้แผนที่ด้วยวิธี Pose Graph Optimization เสมือนเป็นการเชื่อมตำแหน่ง view แต่ละ view ด้วยสปริง (ตามรูปที่ 5.1) กระบวนการ Optimization จะเป็นการหา configuration ของ view ทั้งหมดที่ทำให้สปริงเข้าสู่สถานะเสถียร ซึ่งสปริงชิ้นไหนจะถูกบิดตัวไปมากหรือน้อย ขึ้นกับค่านิจ (spring constant) ของสปริง



รูปที่ 5.1แบบจำลองแนวคิดของ Pose Graph Optimization ด้วยสปริง

สำหรับกรณีของ Pose Graph หลังจากการหาค่าเหมาะสุด ค่าความคลาดเคลื่อนของตำแหน่งสัมพัทธ์ (relative transformation) ของแต่ละ edge จะขึ้นอยู่กับ virtual measurement covariance ของ edge constraint นั้น ๆ โดยค่า covariance จะบ่งชี้ถึงค่าความมั่นใจของ edge ยกตัวอย่างเช่น virtual measurement ระหว่าง node i และ node j ที่มีค่า covariance ต่ำมาก หมายความว่า relative transformation ระหว่าง node i และ node j มีค่าความมั่นใจสูง ซึ่งจะทำให้หลังจากกระบวนการ Optimization แล้ว relative transformation ระหว่าง node i และ node j ก็จะมีค่าเปลี่ยนแปลงน้อยเมื่อเทียบกับ edge อื่น ๆ ที่มีค่าความมั่นใจต่ำกว่า

การคำนวนหา virtual measurement covariance จะหาได้จาก error การประมาณการเคลื่อนที่จากข้อมูลภาพในบทที่ 4

กำหนดให้ p_j เป็นตำแหน่งจุดสังเกตคำนวนได้จาก

$$p_j = x_n \oplus (r_j / d_j)$$

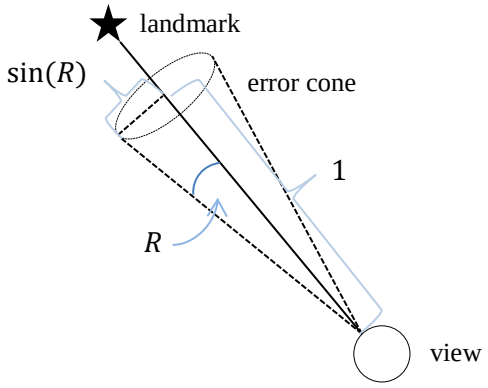
เมื่อ x_n เป็น reference view ของจุดสังเกต j, r_j เป็น unit vector แสดง direction ของจุดสังเกตใน reference view n และ d_j เป็นค่า invert depth ของจุดสังเกต โมเดลการวัดของจุดสังเกต j ที่ view i เขียนได้เป็น

$$\hat{z}_{ij} = z(x_i, d_j) = \frac{p_j \ominus x_i}{\|p_j \ominus x_i\|_2}$$

เมื่อกำหนดให้ error ในการวัดจุดสังเกต อธิบายอยู่ในรูปของคลาดเคลื่อนขององศาทิศทางการวัด มี mean เป็น 0 และค่าเบี่ยงเบนมาตรฐานเป็น R, covariance ของค่าการวัดจะประมาณให้เป็น

$$cov[z] \approx \begin{bmatrix} \sin^2 R & 0 & 0 \\ 0 & \sin^2 R & 0 \\ 0 & 0 & \sin^2 R \end{bmatrix}$$

การคำนวนข้างต้นเป็นการหาระยะความคลาดเคลื่อนจากองศาความคลาดเคลื่อน แสดงได้ดังรูปที่ 5.2



รูปที่ 5.2 error cone ของการวัด

สำหรับกรณีค่า R เข้าใกล้ 0 จะได้ว่า $\sin R \approx R$ ดังนั้นจะประมาณได้ว่า

$$cov[z] \approx \begin{bmatrix} R^2 & 0 & 0 \\ 0 & R^2 & 0 \\ 0 & 0 & R^2 \end{bmatrix}$$

โดยปรกติแล้ว virtual measurement covariance จะคำนวนได้จากจาก

$$\Sigma = \sum F_j \, cov[z] \, F_j^T$$

โดยที่ F_j เป็น Jacobian ของ Function การแปลงจากค่าการวัดเป็นการเคลื่อนที่ของกล้อง แต่เนื่องจากว่าค่าการวัดเพียงค่าเดียวไม่สามารถคำนวนหาการเคลื่อนที่ของกล้องได้ ดังนั้นจึงหา Jacobian F_j ไม่ได้ การคำนวน covariance จึงต้องหาจาก invert ของ Information Matrix การประมาณการเคลื่อนที่แทน $\Sigma = \Lambda^{-1}$ โดยที่

$$\Lambda = \sum_j H_j^T cov[z]^{-1} H_j$$

เมื่อ H_j เป็น Jacobian ของโมเดลการวัดของจุดสังเกต j

5.1.2 Optimizing Scale

กระบวนการ Pose Graph Optimization ในข้างต้นเป็นกระบวนการปรับแก้ตำแหน่งใน 6 มิติโดยประกอบด้วยการเลื่อนตำแหน่ง 3 มิติ และการหมุน 3 มิติ แต่สำหรับ Monocular Visual SLAM แล้วอาจจะมีปัญหา Scale Drift เกิดได้ ซึ่งเป็นปัญหาการเบี่ยงเบนของขนาดแผนที่ในบริเวณต่าง ดังนั้นกระบวนการปรับแก้ตำแหน่งเพียง 6 มิตินั้นอาจไม่เพียงพอ จึงต้องเพิ่มการปรับแก้ scale ขึ้นมาด้วย

ในงานวิจัย [124] ได้นำเสนอวิธีในการ Optimize Pose Graph ใน 7 มิติ โดยเพิ่มมิติ scale ขึ้นมาในสมการ โดย view pose จะเป็นสมาชิกของ similarity transformations group ($Sim(3)$)

$$S = \begin{bmatrix} sR & t \\ 0 & 1 \end{bmatrix}$$

เมื่อ R เป็น rotation matrix, t เป็น translation vector และ s เป็น scale factor

กระบวนการ Optimize Pose Graph บน 7 มิติจะทำให้ scale ของแผนที่ก่อนและหลังการ close loop มีขนาดเท่ากัน และทำให้ scale ของแผนที่ในส่วนอื่นสอดคล้องกันทั่วบริเวณอย่างไรก็ดี ก็ไม่อาจรับประกันได้ว่า scale ที่ได้หลังการ optimize มีความถูกต้องเทียบกับสิ่งแวดล้อมจริง ทั้งนี้เนื่องจากว่าการเบี่ยงเบนของขนาดของแผนที่มีการเบี่ยงเบนแบบสุ่ม

ในงานวิจัยนี้เราจะปรับแก้ scale ด้วยวิธีที่ interpolate ค่า scale factor โดยตรงบน log scale สำหรับ view ทั้งหมดที่อยู่ใน loop โดยในแต่ละ view นั้น เราจะเก็บค่า scale factor (s_i) ซึ่งใช้บ่งชี้ว่า view ที่ i มี scale เปลี่ยนไปจากปรกติเท่าใด

โมเดลการวัดของจุดสังเกต j ที่ view i สามารถคำนวนใหม่ได้เป็น

$$\hat{z}_{ij} = z(x_i, d_j) = \frac{p_j \ominus x_i}{\|p_j \ominus x_i\|_2}$$

เมื่อ

$$p_j = x_n \oplus (r_j \cdot s_n / d_j)$$

โดยที่ x_n เป็น reference view ของจุดสังเกต j, s_n เป็น scale factor ของ view ที่ n, r_j เป็น unit vector แสดง direction ของจุดสังเกตใน reference view n และ d_j เป็นค่า invert depth ของจุดสังเกต

สำหรับค่า scale factor (s_n) สามารถคำนวนได้ในขั้นตอนการ close loop โดยสมมุติว่าเมื่อตรวจพบ loop closure ระหว่าง node x_i และ node x_j เราสามารถหาได้ว่า view ที่ j มี scale factor (s_j) เป็นเท่าไร่เทียบกับ view ที่ i (รายละเอียดในการคำนวนหา relative transform และ scale factor ระหว่าง view i และ view j จะอธิบายโดยละเอียดในหัวข้อที่ 6.1) scale factor (s_n) สำหรับ view ที่ n สามารถหาได้จาก

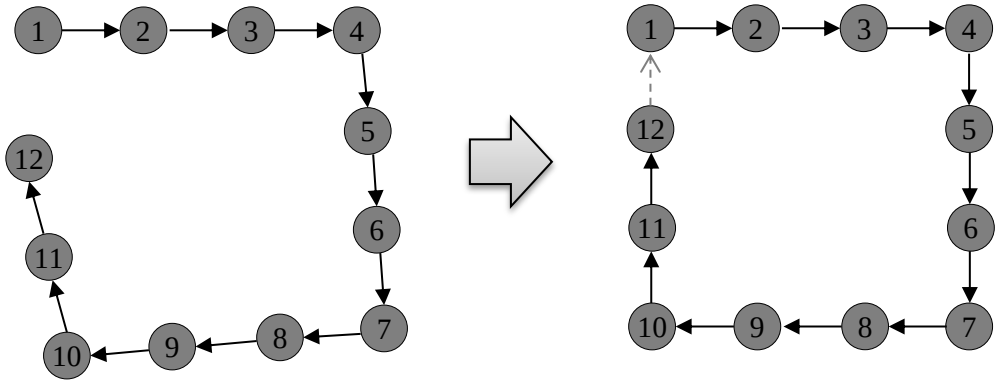
$$s_n = e^{(\log(s_j) - \log(s_i)) \cdot d_n / d_j}$$

เมื่อ d_j, d_n เป็นระยะทางจากจาก view i ไป view j และระยะทางจากจาก view i ไป view n ตามลำดับ

ในกระบวนการ Pose Graph Optimization ค่า s_n ที่คำนวนได้จะใช้ในการปรับ translation (t_k) ของ virtual measurement จะได้ว่า $x_n = [s_n t_n \ q_n]^T$ จากนั้นจึงใช้ virtual measurement ค่าใหม่นี้ในการคำนวนหาค่าเหมาะสุดตามปรกติ ผลลัพธ์ที่ได้จะได้แผนที่ซึ่ง scale ล่าสุดสอดคล้องกับแผนที่ก่อน close loop และ scale ของแผนที่บริเวณอื่น ๆ ก็จะปรับเปลี่ยนไปตาม log scale ระหว่าง view เริ่มต้นและ view ล่าสุด

5.2 การปรับแก้แผนที่อย่างมีประสิทธิภาพ

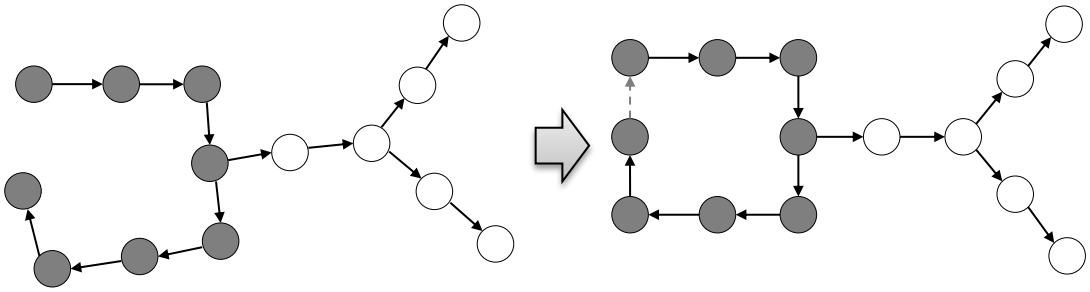
การทำงานของ Pose Graph Optimization ที่ได้อธิบายในหัวข้อที่ 5.1 นั้นใช้เวลาในการทำงานเป็น $O(n^2)$ เมื่อ n เป็นจำนวน view ซึ่งจะเห็นได้ว่าเมื่อขนาดของแผนที่ใหญ่ขึ้น เวลาที่ใช้ในการทำงานจะโตเป็น polynomial ตามไปด้วย เนื่องจากการปรับแก้แผนที่จะต้องมีการปรับปรุงตำแหน่งของ view ทั้งหมดเพื่อให้ได้ตำแหน่งเหมาะสุด ตัวอย่างการปรับแก้แผนที่แสดงได้ดังรูปที่ 5.3



รูปที่ 5.3 การปรับแก้แผนที่เพื่อ close loop

สำหรับแผนที่ที่มีขนาดใหญ่มาก ๆ การปรับแก้แผนที่อาจจะทำงานได้ไม่ทันการณ์ ดังนั้นในหลาย ๆ งานวิจัย จึงหาวิธีในการลดเวลาในการประมาณคำตอบ ไม่ว่าจะเป็นการลดจำนวน node [40, 125, 126] หรือการจัดโครงสร้างข้อมูลแบบต้นไม้ [127] ซึ่งก็จะช่วยให้ลดเวลาในการคำนวนได้ แต่ก็จะส่งผลให้ผลลัพธ์มีความคลาดเคลื่อนไปจาก Exact Solution

เมื่อพิจารณาการ Close Loop สำหรับกรณีทั่วไป เช่นในรูปที่ 5.4 เมื่อมีการ Close Loop เกิดขึ้น node ทั้งหมดในแผนที่จะถูกเลื่อนตำแหน่งเพื่อการปรับแก้ แต่เมื่อพิจารณาส่วนที่เป็นโหนดสีขาว จะเห็นได้ว่ารูปทรงของกลุ่มโหนดใน local space ไม่ได้มีการเปลี่ยนแปลงทั้งนี้เนื่องจากว่า constraint ในการ Close Loop นั้นไม่ได้ส่งผลกระทบกับโหนดบริเวณที่เป็นสีขาวด้วย ดังนั้นเพื่อลดเวลาในการทำงานเราจะสร้าง Pose Graph ที่บรรยายตำแหน่งแบบเชิงสัมพัทธ์ (Relative Transform) โดยโครงสร้างความสัมพันธ์ของ view จะเป็นแบบต้นไม้ ซึ่งที่โหนดลูกจะเก็บของมูล relative Transform เทียบกับโหนดพ่อ ตามที่ได้อธิบายไปในหัวข้อที่ 3.3 การจัดโครงสร้างข้อมูลเช่นนี้ จะทำให้สามารถลดจำนวนโหนดในการ optimize ได้โดยในการปรับแก้แผนที่จะปรับแก้แค่เพียงโหนดที่เป็นสีเทาเท่านั้น ส่วน node สีขาวจะมีการเปลี่ยนแปลงตำแหน่งสัมพัทธ์กับ node พ่อ



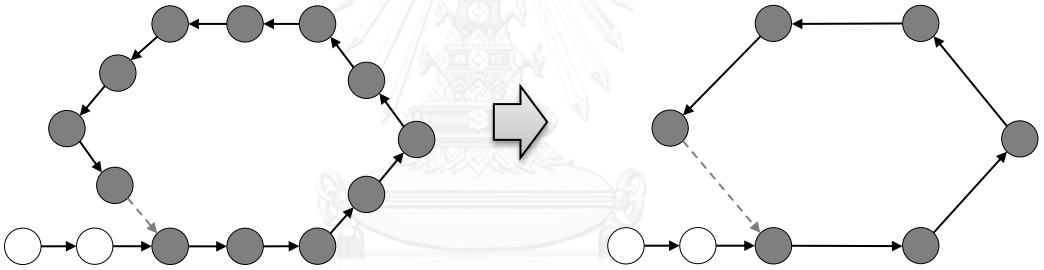
รูปที่ 5.4 การปรับแก้แผนที่เพื่อ close loop (2)

การจัดเก็บตำแหน่ง view ในรูปแบบ relative Transform จะช่วยลดเวลาในการปรับแก้แผนที่ โดยเมื่อมีการ close loop เกิดขึ้นการปรับแก้จะปรับแก้เฉพาะโหนดที่ก่อให้เกิด loop ทำให้สามารถลดเวลาในการคำนวนได้ อย่างไรก็ดีในกรณีที่หุ่นยนต์เคลื่อนที่เป็นระยะทางไกลมาก จะทำให้มี view จำนวนมากก่อนการ close loop ซึ่งทำให้เวลาในการทำงานแย่สุดเป็น $O(n^2)$ อยู่ดีดังนั้น ในวิทยานิพนธ์ฉบับนี้จะนำเสนอวิธีลด

เวลาในการปรับแก้แผนที่ตอน close loop โดยการลดโหนดในกราฟไม่ให้เกินค่าคงที่ค่าหนึ่งเพื่อให้สามารถทำงานได้ทันการณ์ แต่ยังคง information ไว้ซึ่งหมายความว่า ผลลัพธ์หลังจากทำ Pose Graph Optimization บนกราฟที่ลดโหนดแล้วจะยังมีคำตอบเทียบเท่ากับการปรับแก้กราฟแบบเต็ม ซึ่งในที่นี้จะเรียกว่าวิธี Pose Marginalization (อธิบายในหัวข้อ 5.2.1)

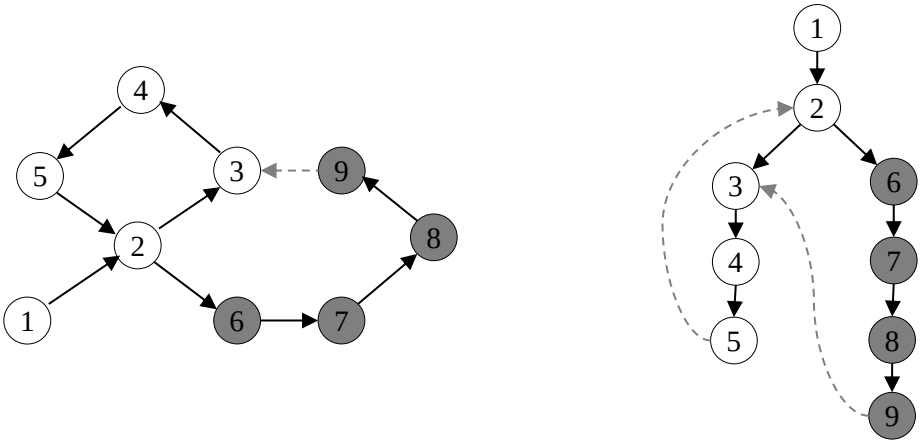
พิจารณาตัวอย่าง Pose Graph ใน*รูปที่ 5.5* (ซ้าย) สมมุติหุ่นยนต์เริ่มเคลื่อนที่จากจุดเริ่มต้น วิ่งวนรอบและกลับมาที่ยังตำแหน่งใกล้เคียงจุดเริ่มต้น เพื่อป้องกันไม่ให้ขั้นตอนในการปรับแก้แผนที่ใช้เวลามากเกินไป เราจะทำการเพิ่มขั้นตอนในการลดโหนดโดยจะทำงานทุกครั้งเมื่อมีการเพิ่มโหนดใหม่เข้ามา เงื่อนไขในการลดโหนดก็คือจะทำงานก็ต่อเมื่อจำนวนโหนดมีค่ามากเกินค่าคงที่ที่กำหนดไว้ ผลลัพธ์ของการลดโหนดจะแสดงได้ดังรูปที่ 5.5 (ขวา) สำหรับการปรับแก้แผนที่ จะทำเฉพาะโหนดที่อยู่ใน loop เท่านั้น ทำให้จะรับประกันได้ว่ามีจำนวนโหนดไม่เกินค่าคงที่ที่กำหนดไว้ ดังนั้นจะได้ว่า เวลาในการปรับแก้แผนที่เป็น $O(1)$

การทำ Pose Marginalization จะทำเพื่อลดโหนดบนกราฟเท่านั้นไม่ได้ลบ view นั้นทิ้งไปจากแผนที่ ดังนั้นหุ่นยนต์สามารถ observe ซ้ำ view นั้นได้โดยใช้ขั้นตอนการกู้คืนโหนด (Pose Restoration) ซึ่งจะอธิบายในหัวข้อ 5.2.2



รูปที่ 5.5 Pose Graph ก่อนและหลังการทำ Pose Marginalization

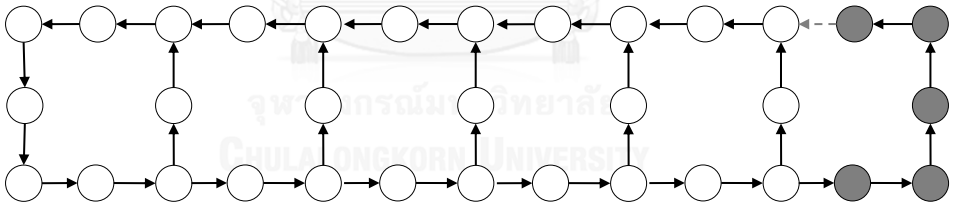
สำหรับ Pose Graphที่มีความซับซ้อนเป็นต้นว่ามี close loop ซ้อน close loop เช่นในกรณีรูปที่ 5.6 (ก) หุ่นยนต์เคลื่อนที่ไป close loop ในส่วน loop อื่น ต้นไม้ความสัมพันธ์ของ view แสดงได้ดังรูปที่ 5.6 (ข) ในขั้นตอนการลดโหนด จะพิจารณาเฉพาะโหนดในเป็นแขนงล่าสุดของต้นไม้เท่านั้นซึ่งในกรณีนี้ก็คือโหนดสีเทา ส่วนการปรับแก้แผนที่ก็จะทำการปรับแก้เฉพาะส่วนที่เป็นสีเทาเช่นกัน โดยโหนดสีขาวที่เชื่อมต่อกับโหนดสีเทาจะกำหนดให้มีตำแหน่งคงที่ในขั้นตอนการปรับแก้ ทำให้สามารถรับประกันเวลาในการทำงานได้



(ก) *(ข)*

รูปที่ 5.6 (ก) pose graph แสดงตำแหน่งของ vew ในแผนที่ (ข) ต้นไม้ความสัมพันธ์ของ view

อย่างไปก็ดี การปรับแก้แผนที่เฉพาะแขนงของต้นไม้นั้น ย่อมทำให้แผนที่หลังการปรับแก้นั้น ไม่ใช่ค่าเหมาะสุด เช่นในรูปที่ 5.6 ถ้าหากเกินการ close loop ขึ้นที่ตำแหน่งเส้นประ โหนดอื่น ๆ ที่ควรจะโดนผลกระทบด้วย ก็คือโหนดที่อยู่ใน loop สีขาวทั้งหมดยกเว้นโหนดเริ่มต้น ซึ่งถ้าหากปรับแก้โหนดทั้งหมดยกเว้นโหนดเริ่มต้น ก็จะได้ Pose Graph ที่มีค่า Optimal แต่เหตุผลที่เราไม่ปรับแก้โหนดที่ถูกผลกระทบทั้งหมด เนื่องจาก เราไม่สามารถรับประกันได้ว่าโหนดที่ถูกผลกระทบจะมีจำนวนมากน้อยเท่าไร่ ยกตัวอย่างเช่น ในรูปที่ 5.7 การ close loop เล็ก ๆ บริเวณท้ายแผนที่อาจส่งผลกระทบถึง view ทั้งหมดในแผนที่ก็ได้



รูปที่ 5.7 แสดงการ close loop ของ Pose Graph

เพื่อให้ได้แผนที่ที่ optimal เราจะแบ่งการทำงานการปรับแก้แผนที่ออกเป็น 2 Thread โดย Thread แรกจะทำ Local Map Update ซึ่งก็คือการปรับแก้แผนที่เฉพาะแขนงของต้นไม้ โดยการทำงานจะเป็นแบบทันการณ์ เพื่อให้ได้ข้อมูลแผนที่ในถูกต้องใน local scale สำหรับใช้ในการระบุตำแหน่งต่อไป ส่วน Thread ที่สองจะทำ Global Map Update เนื่องจากการทำ Local Map Update ใน thread แรกนั้นจะยังคงเหลือ view ที่ถูกผลกระทบแต่ยังไม่ถูกปรับแก้อยู่ ในที่นี้จะเรียกว่า dirty node การทำงานส่วน Global Map Update จะค้นหาส่วนที่เป็น dirty node ทั้งหมด (อธิบายรายละเอียดในหัวข้อ 5.2.3) และทำการปรับแก้ด้วยวิธี Pose Graph Optimization และทำงานวนซ้ำไปเรื่อย ๆ โดยไม่สนใจเวลาที่ใช้ในการทำงาน

การแบ่งปัญหา Map Optimization ออกเป็นสอง Thread เช่นนี้สามารถทำได้ เนื่องจากแผนที่ซึ่งจะถูกปรับแก้ใน Thread ทั้งสองนั้น สามารถแยกจากกันได้อย่างเด็ดขาดโดย Local Thread จะปรับแก้เฉพาะโหนดที่

เป็นแขนงล่าสุดเท่านั้น ส่วน Global Thread ก็จะปรับแก้เฉพาะโหนดที่ถูก fix ตำแหน่งแล้วทำให้ไม่มีความซ้ำซ้อนของข้อมูลในการทำงาน

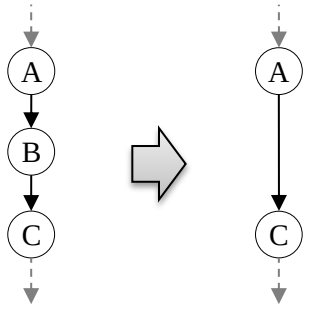
การปรับแก้แผนที่โดยแยกการทำงานออกเป็นสอง Thread ข้างต้น จะมีความสอดคล้องกับ พฤติกรรมของมนุษย์ในการรู้จำแผนที่ โดยเมื่อมุนษย์ตรวจพบ loop closure แล้ว ในตอนเริ่มแรกมนุษย์อาจจะยังมึนงงกับทิศทางอยู่บ้างแต่มนุษย์ก็เลือกที่จะปรับ topology ให้ถูกต้องก่อนเพื่อให้สามารถรู้จำแผนที่เฉพาะในบริเวณนั้นได้ จากนั้นมนุษย์จึงใช้เวลาพินิจในการปรับแก้แผนที่แบบ global เพื่อให้แผนที่ถูกต้องทั้งหมด

5.2.1 Pose Marginalization

จุดมุ่งหมายของการทำ Pose Marginalization นั้นก็เพื่อทำการลดตัวแปรที่จะถูกคำนวนในขั้นตอนการปรับแก้แผนที่ เพื่อให้การปรับแก้แผนที่สามารถทำงานได้ทันการณ์ นอกจากนี้ผลลัพธ์ที่ได้หลังการปรับแก้แผนที่นั้นต้องเทียบเท่าการปรับแก้แผนที่ที่ไม่ได้ถูกทำ Pose Marginalization ด้วย

กระบวนการ Pose Marginalization มีแนวคิดมาจาก Marginalization Operation ของ Information Filters ซึ่งเป็นขั้นตอนการแพร่ information ของตัวแปรสุ่มตัวหนึ่งไปยังตัวอื่น ๆ ปัญหาของ Information Filters ก็คือในขั้นตอนปรับแก้สถานะระบบนั้นจะทำการ linearize โมเดลการวัดเพียงครั้งเดียวและไม่มีการทำงานแบบวนซ้ำ จึงทำให้การประมาณสถานะของระบบไม่แม่นยำพอ และไม่เหมาะกับการปรับแก้แผนที่ขนาดใหญ่ซึ่งมีความคลาดเคลื่อนสูง

เมื่อเปรียบ Pose graph เป็นโมเดลมวลและสปริงในฟิสิกส์ คำตอบของการปรับแก้ Pose graph ก็เปรียบเสมือนการหา configuration ของสปริงที่ minimize energy โดยสปริงชิ้นไหนจะถูกบิดตัวไปมากน้อย จะขึ้นกับค่านิจ (spring constant) ของสปริง ดังนั้นการทำ Pose Marginalization ก็เปรียบเสมือนการลดมวลทิ้งไปบางโหนดและปรับค่านิจของสปริง เพื่อให้ส่งผลกระทบต่อโมเดลเหมือนเดิม สำหรับ Pose Marginalization จะทำการลดโหนดและปรับแก้ constraint ระหว่างโหนดให้ถูกต้อง



รูปที่ 5.8 (ซ้าย) ต้นไม้ความสัมพันธ์ของโหนด A, B และ C (ขวา) ต้นไม้ความสัมพันธ์หลังลดโหนด B

กำหนดให้โหนด A, B และ C มีความสัมพันธ์กันเชิงตำแหน่งในแผนภูมิต้นไม้ดังรูปที่ 5.8 (ซ้าย) โดยที่โหนด A เป็นพ่อของโหนด B และโหนด B เป็นพ่อของโหนด C เมื่อกำหนดให้ x_A^B และ x_B^C เป็น virtual measurementตำแหน่งสัมพัทธ์ของโหนด B เทียบกับโหนด A และ virtual measurement ตำแหน่งสัมพัทธ์ของ

โหนด C เทียบกับโหนด B ตามลำดับ การลดโหนด B ออกจากต้นไม้จะต้องคำนวน virtual measurement ใหม่สำหรับ constraint ระหว่างโหนด A และโหนด C โดย virtual measurementใหม่หาได้จาก

$$f_{AC}\left(x_A^B, x_B^C\right) = x_A^B \oplus x_B^C$$

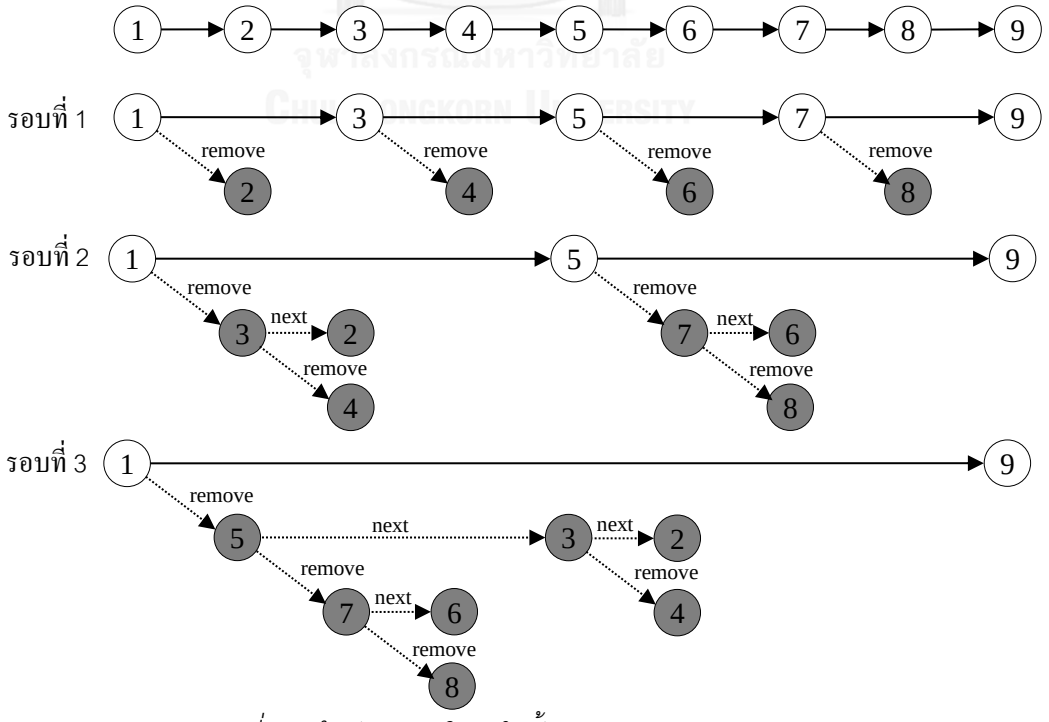
และ covariance ของ measurement ใหม่ หาได้จาก

$$\Sigma_{AC} = [F_{AB} \quad F_{BC}] \begin{bmatrix} \Sigma_{AB} & 0 \\ 0 & \Sigma_{BC} \end{bmatrix} [F_{AB} \quad F_{BC}]^T$$

เมื่อ Σ_{AB} เป็น covariance ของ virtual measurement ระหว่างโหนด A และ B, Σ_{BC} เป็น covariance ของ virtual measurement ระหว่างโหนด B และ C, ส่วน F_{AB} และ F_{BC} เป็น Jacobian ของฟังก์ชัน เทียบกับ x_A^B และ x_B^C ตามลำดับ

หลังจากทำการลดโหนดแล้ว จะต้องเก็บค่า virtual measurement x_B^C และ covariance ไว้ในโหนด B เพื่อรอการกู้คืน และจัดเก็บโหนด B ไว้ในรายการโหนดที่ถูกลบของโหนด A

หลักเกณฑ์ในการเลือกโหนดที่จะถูกลดนั้นจะลดเฉพาะโหนดที่เรียงกันเป็นเส้นตรงในต้นไม้เท่านั้นโดยโหนดจะต้องมีโหนดพ่อและโหนดลูกอย่างหนึ่ง การลดโหนดในแขนงจะเป็นแบบเว้นช่วงวนซ้ำไปเรื่อยๆ จนกว่าจะมีจำนวนโหนดในแขนงไม่เกินค่าคงที่ที่กำหนด ยกตัวอย่างเช่นในรูปที่ 5.9 ที่มีโหนดเรียงต่อกัน 9 โหนด การลดโหนดในรอบแรกจะเรียงลำดับโหนดที่ 2, 4, 6, 8 จากนั้นจึงวนซ้ำโดยลดโหนดที่ 3 และ 7 ในรอบที่สอง และในรอบที่สามโหนดที่ 5 จะถูกลดตามลำดับ ผลลัพธ์หลังการลดโหนดแสดงได้ดังรูปที่ 5.9 คอลัมน์สุดท้าย

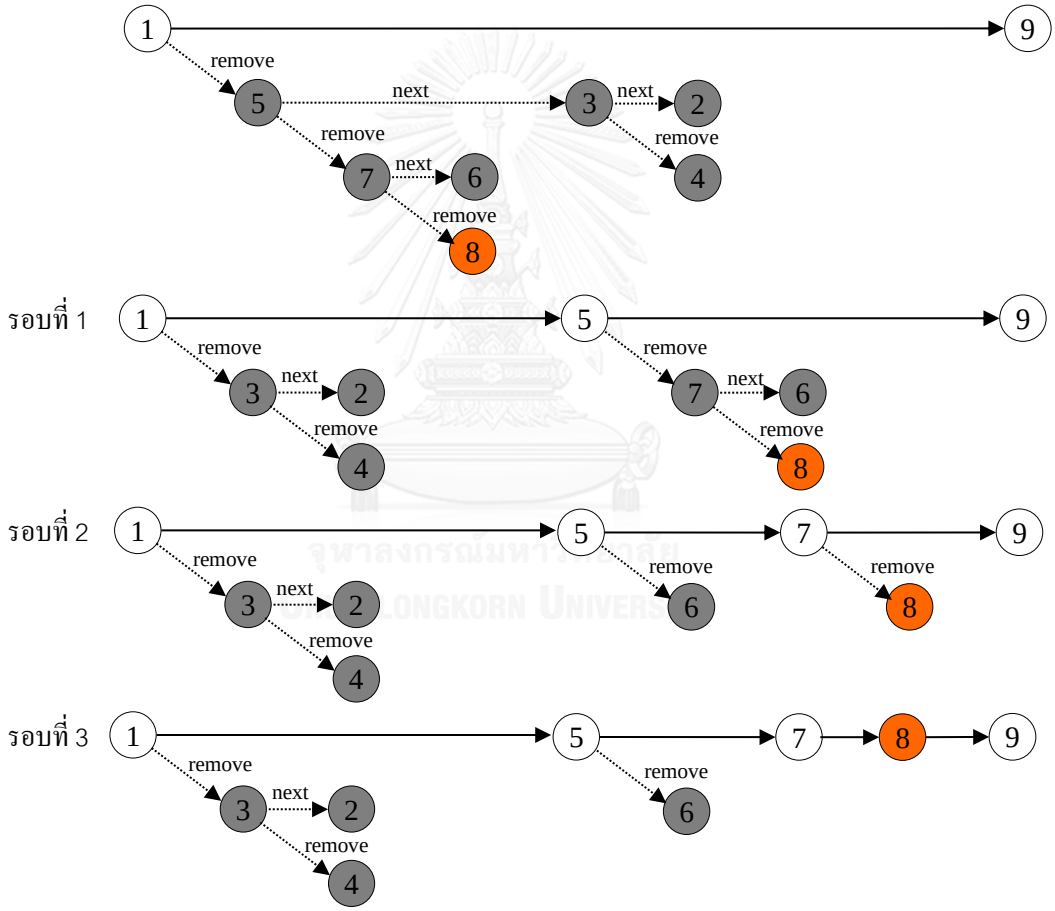


รูปที่ 5.9 ลำดับการลดโหนดในขั้นตอน Pose Marginalization

5.2.2 Pose Restoration

กระบวนการ Pose Marginalization ในหัวข้อที่ 5.2.1 จะช่วยลดโหนดในต้นไม้ เพื่อให้สามารถปรับแก้แผนที่ได้ในเวลาทันการณ์ ซึ่งในบางครั้งเมื่อหุ่นยนต์เคลื่อนที่วนซ้ำกลับมายังตำแหน่งเดิม จำเป็นจะต้องทำการกู้คืนโหนดบางโหนดในต้นไม้ เพื่อใช้ในการ close loop หรือการระบุตำแหน่งของหุ่นยต์

กระบวนการ Pose Restoration จะทำการกู้คืนโหนดในต้นไม้เป็นลำดับขั้น ยกตัวอย่างเช่นในรูปที่ 5.9 คอลัมน์สุดท้าย มีโหนดถูกลดไป 3 รอบ โดยรอบแรกนั้นโหนดที่ 2, 4, 6, 8 ถูกลด รอบที่สองโหนดที่ 3 และ 7 ถูกลด และในรอบสุดท้ายโหนดที่ 5 ถูกลดไป ดังนั้นถ้าหากต้องการกู้คืนโหนดที่ 8 จะต้องกู้คืนโหนดที่ 5 ก่อน จากนั้นกู้คืนโหนดที่ 7 และ 8 ตามลำดับ รูปตัวอย่างลำดับการกู้คืนโหนดแสดงได้ดังรูปที่ 5.10



รูปที่ 5.10 ลำดับการทำ Pose Restoration โหนดที่ 8

สำหรับการคำนวนตำแหน่งกู้คืนโหนดแต่ละโหนด ยกตัวอย่างเช่นในรูปที่ 5.11 เราต้องการกู้คืนโหนด B ซึ่งอยู่ระหว่างโหนด A และโหนด C จะต้องทำการเพิ่มโหนด B ระหว่างโหนด A และ C จากนั้นจะต้องทำการกู้คืน virtual measurement x_A^B และ x_B^C ซึ่งเป็นค่าการวัดตำแหน่งสัมพัทธ์ของโหนด B เทียบกับโหนด A และของโหนด C เทียบกับโหนด B ตามลำดับ พร้อมด้วย covariance ของ virtual measurement นั้น ซึ่งค่า virtual

measurement ทั้งสองนั้นได้ถูกเก็บค่าไว้แล้วในโหนด B ก่อนทำการลดโหนด ดังนั้นจึงสามารถคืนค่า virtual measurement ได้ทันที

รูปที่ 5.11 ต้นไม้ความสัมพันธ์ของ node A, B และ C (2)

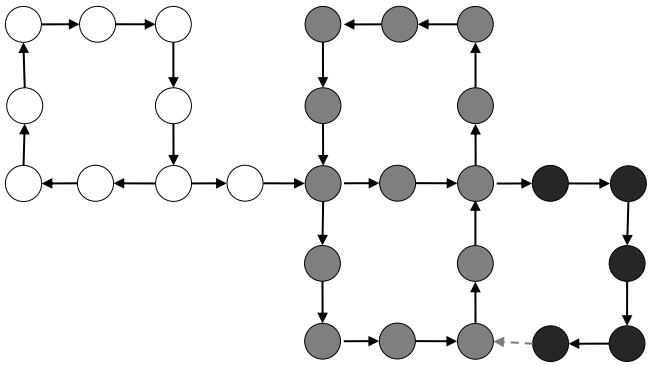
ส่วนตำแหน่งสัมพัทธ์ของโหนด B เทียบกับโหนด A สามารถคำนวนได้จากตำแหน่งสัมพัทธ์ของโหนด C เทียบกับโหนด A และค่า virtual measurement โดยจะต้องทำการ optimize ตำแหน่งของโหนด B ตามวิธีในหัวข้อ 5.1 โดยใช้ constraint จาก virtual measurement x_A^B และ x_B^C เมื่อกำหนดให้ตำแหน่งของโหนด A และโหนด C คงที่ ก็จะทำให้สามารถคำนวนตำแหน่งของโหนด B ได้

จะเห็นได้ว่าการกู้คืนโหนดเป็นลำดับขั้น จะใช้เวลาเป็น $O(log(n/k))$ เมื่อ n เป็นจำนวน view ในแขนง และ k เป็นจำนวนโหนดมากสุดที่ถูกจำกัดไว้เป็นค่าคงที่ในแขนง โดยเมื่อแขนงมีความยาวมาก จำนวนลำดับขั้นก็จะมีมากขึ้นเป็น log scale อย่างไรก็ดีในการใช้งานจริงพบว่า ขั้นตอนการกู้คืนนั้นใช้เวลาน้อยมากเพื่อเทียบกับเวลาในการ close loop ดังนั้นการทำงานของการกู้คืนโหนดจึงแทบไม่กระทบถึงประสิทธิภาพของอัลกอริทึม

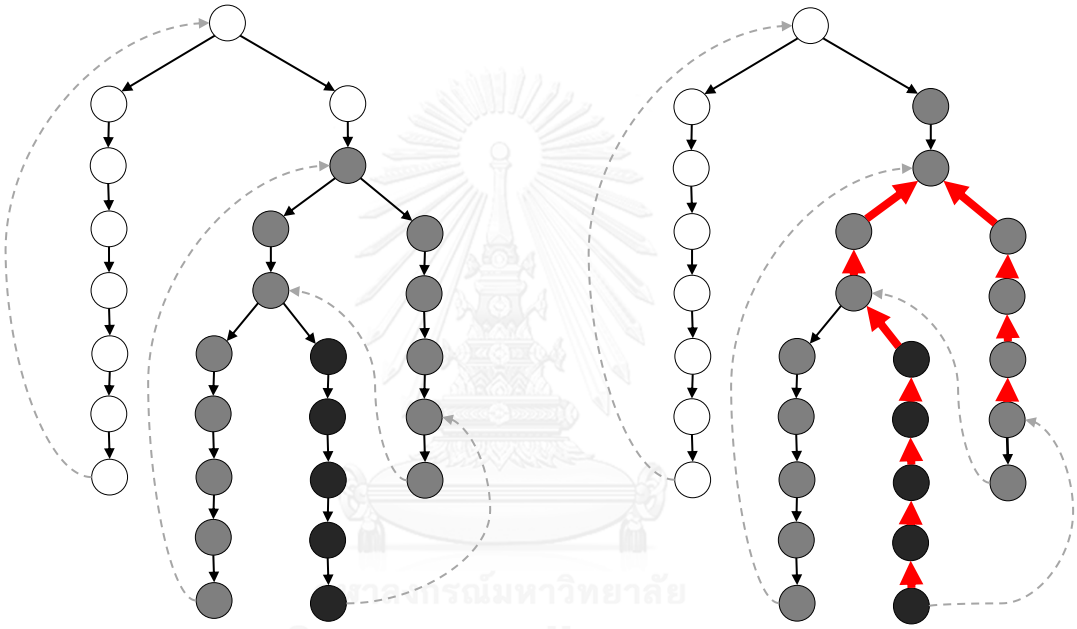
5.2.3 Dirty Loop Detection

การตรวจหา dirty loop จะเป็นการคำนวนหาโหนดที่จะถูกผลกระทบหลังจากการ close loop และต้องได้รับการปรับแก้ในขั้นตอน Global Map Optimization

พิจารณาจากตัวอย่างในรูปที่ 5.12 เมื่อหุ่นยนต์ทำการ close loop กลุ่ม dirty node ที่จะถูกผลกระทบและต้องถูกปรับแก้อธิบายได้ว่า เป็นโหนดใด ๆ ที่สามารถสร้าง loop ระหว่างโหนดและแขนงล่าสุดได้ (กลุ่มโหนดสีเทาในรูปที่ 5.12) การคำนวนหา dirty node สามารถทำได้ด้วยวิธีตรงไปตรงมาโดยการเก็บรายการของ loop ทั้งหมดไว้ โดยในแต่ละ loop จะเก็บรายการโหนดที่ loop นั้นวิ่งผ่าน ส่วนในแต่ละโหนดก็จะเก็บข้อมูลรายการ loop ไว้ว่ามี loop ไหนวิ่งผ่านบ้าง จากนั้นในขั้นตอนการตรวจหา dirty loop ก็สามารถหาได้จาก loop ทั้งหมดที่วิ่งผ่านโหนดใน loop ล่าสุด



รูปที่ 5.12 แสดงการ close loop ของ Pose Graph (2)



(ก) *(ข)*

รูปที่ 5.13 (ก) ต้นไม้ความสัมพันธ์ของ view (ข)การตรวจหา loop จากต้นไม้ความสัมพันธ์

จากต้นไม้ความสัมพันธ์จากรูปที่ 5.13 (ก) การตรวจหา loop จะทำการ backtrack โหนดในต้นไม้ขึ้นไปตามโหนดพ่อ จนกว่าจะพบโหนดพ่อร่วมกันดังรูปที่ 5.13 (ข) ก็จะสามารถสร้าง loop ได้ซึ่ง loop ที่หาได้นั้นก็จะถูกนำไปคำนวนหา dirty loop ต่อไป

5.3 การทดลองการปรับแก้แผนที่

การปรับแก้แผนที่สำหรับสิ่งแวดล้อมขนาดใหญ่จะมีปัญหาอยู่สองประการได้แก่ ปัญหาประสิทธิภาพในการคำนวน และปัญหาความทนทาน (Robustness) ในการทำงาน ผลการทดลองในที่นี้จะแสดงการวัดผล

อัลกอริทึมการปรับแก้แผนที่ที่ได้นำเสนอ โดยแสดงให้เห็นว่าอัลกอริทึมมีความสามารถในการทำงานแบบทันการณ์ และมีความทนทานต่อสภาพแวดล้อมขนาดใหญ่

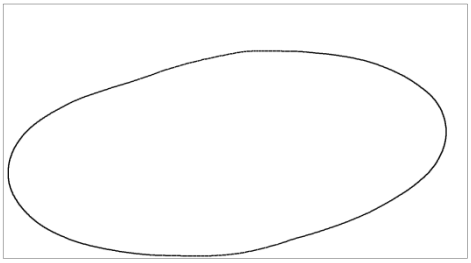
การทดลองการปรับแก้แผนที่จะทำงานทดลองบนข้อมูล simulation โดยจะจำลองการเคลื่อนที่ของหุ่นยนต์แบบต่าง ๆ ทั้งในกรณีทั่วไปและกรณีเลวร้ายสุด ในการจำลองข้อมูลนั้นจะจำลองตำแหน่งของหุ่นยนต์ในแต่ละโหนด จากนั้นจึงประมาณข้อมูลการวัดการเคลื่อนที่หุ่นยนต์จากผลต่างของตำแหน่งหุ่นยนต์ระหว่างสองโหนดที่ต่อเนื่องกัน จากนั้นจึงทำการสุ่มเพิ่มความคลาดเคลื่อนเข้าไปในข้อมูลการวัด ส่วนขั้นตอนการ close loop นั้นจะจำลองการตรวจหา loop closure ด้วยโดยวัดจากตำแหน่งของหุ่นยนต์เมื่อเคลื่อนที่เข้าใกล้กันระยะหนึ่ง

ในส่วนของการวัดผลนั้นจะทำการวัดเทียบกับอัลกอริทึม Tree-based netwORk Optimizer (TORO) [36] ซึ่งเป็นอัลกอริทึมสำหรับ optimize constraint-network โดยใช้ gradient descent-based ในการ minimize error ถูกนำเสนอโดย Giorgio Grisetti ในปี 2007

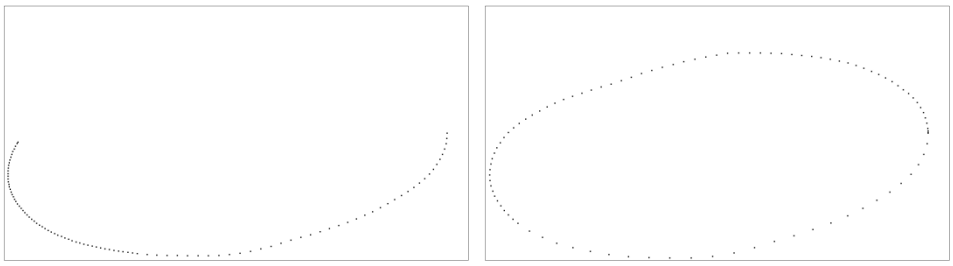
5.3.1 **ผลการทดลอง** Pose Marginalization

การทำ Pose Marginalization ระหว่างที่หุ่นยนต์กำลังเคลื่อนที่ คือการลดจำนวน node ในแผนที่โดยยังคง information ของแผนที่ไว้ ซึ่งหากจำกัดจำนวนโหนดที่จะถูกปรับแก้ไว้ไม่ให้เกินค่าค่าหนึ่ง ก็จะสามารถรับประกันเวลาในการทำงานของการปรับแก้แผนที่ในขั้นตอน close loop ได้

ในการทดลองนี้เราจะจำลองการเคลื่อนที่ของหุ่นยนต์เป็นวงกลมระยะทางไกลประมาณ 1000 เฟรม การทำ Pose Marginalization ในการทดลองนี้จะจำกัดจำนวนโหนดก่อนการ close loop ไว้ไม่เกิน 100 โหนด ผลลัพธ์ของการปรับแก้แผนที่ในกรณีที่ไม่ได้ทำ Pose Marginalization แสดงได้ดังรูปที่ 5.14 ส่วนผลลัพธ์ของการปรับแก้แผนที่โดยผ่านกระบวนการ Pose Marginalization แสดงได้ดังรูปที่ 5.15 (ก) และ (ข)



รูปที่ 5.14 ผลลัพธ์การปรับแก้แผนที่กรณีไม่ได้ทำ Pose Marginalization



(ก) *(ข)*

รูปที่ 5.15 ผลลัพธ์การปรับแก้แผนที่หลังผ่านกระบวนการ Pose Marginalization

จากรูปที่ 5.15 (ก) จะเห็นได้ว่า เมื่อจำนวนโหนดใน path ก่อนทำการ close loop มีจำนวนมากเกิน 100 โหนด จะมีบางโหนดถูกลดออกไป โดยโหนดที่ถูกลดจะกระจายเฉลี่ยตลอดทั้ง path และเมื่อหุ่นยนต์เคลื่อนที่ไปไกลมากขึ้น ก็จะมีโหนดถูกลดมากขึ้นโดยจะสังกตได้จากระยะห่างของช่องว่างระหว่างโหนด ตามรูปที่ 5.15 (ข)

ความแม่นยำในการทำงานจะแสดงได้ดังตาราง 6.1 โดยจะเป็นการเปรียบเทียบความคลาดเคลื่อนของตำแหน่งหลังการปรับแก้แผนที่ระหว่างแผนที่แบบปรกติกับแผนที่ที่ถูกลดโหนด โดยใช้ Euclidian distance (สำหรับแผนที่ที่ถูกลด node ด้วยกระบวนการ Pose Marginalization จะทำ Pose Restoration ก่อนการเปรียบเทียบ)

ตาราง 5.1 ตารางความคลาดเคลื่อนของการปรับแก้แผนที่เทียบกับแผนที่แบบ Full Optimization

Method	Error (m)
Full Optimization	0.00000
Pose Marginalization	0.06076
Node deletion	2.00077
TORO	2.64884

จากตารางความแม่นยำ จะเห็นได้ว่าความคลาดเคลื่อนของแผนที่แบบ Pose Marginalization จะมีความคลาดเคลื่อนน้อยมากซึ่งเกิดจาก numerical error โดยทางทฤษฎีแล้วผลลัพธ์ของแผนที่แบบ Pose Marginalization จะต้องเท่ากับแผนที่แบบปรกติ (Full Optimization) ส่วนแผนที่แบบการลบโหนดทิ้งไป (Node deletion) จะมีความคลาดเคลื่อนมากเนื่องจากสูญเสีย information ในโหนดที่ถูกลบ สำหรับการปรับแก้แผนที่ด้วย TORO จะมีความคลาดเคลื่อนจากการประมาณของอัลกอริทึม

สำหรับเวลาในการทำงานในขั้นตอนการ close loop จะแสดงได้ดังตาราง 6.2 ซึ่งจะเห็นได้ว่าเวลาในการทำงานของแผนที่แบบ Pose Marginalization จะใช้เวลาน้อยกว่าอัลกอริทึมอื่นมากเนื่องจากมีจำนวนโหนดน้อยกว่า

ตาราง 5.2 ตารางแสดงเวลาการทำงานของทั้งสามอัลกอริทึม

Method	Time used (sec)
Full Optimization	7.768
Pose Marginalization	0.055
Node deletion	0.019
TORO	0.578

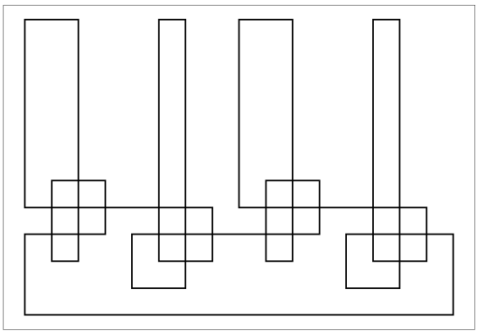
สาเหตุที่แผนที่แบบ Pose Marginalization ใช้เวลาการ close loop น้อยเนื่องจากหลักการทำงานของ Pose Marginalization ก็คือการกระจายเวลาในการทำงานจากขั้นตอน close loop ขั้นตอนเดียว ไปสู่ขั้นตอนการลดโหนดก่อนทำการ close loop ทำให้สามารถ close loop แบบทันการณ์ได้

5.3.2 ผลการทดลองการปรับแก้แผนที่แบบซับซ้อน

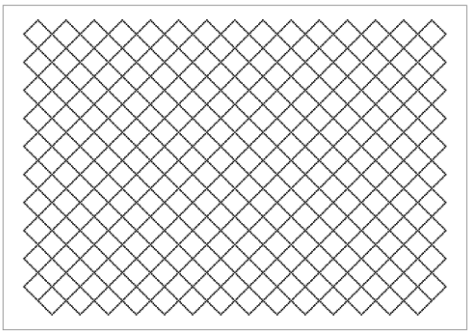
ในหัวข้อที่ 5.3.1 เป็นการทดลองการ close loop เพียง loop เดียวซึ่งในความเป็นจริง เมื่อหุ่นยนต์เคลื่อนที่แบบสุ่มย่อมมีโอกาสเกิด loop มากกว่าหนึ่งครั้ง ดังนั้นในการทดลองนี้จะเป็นการจำลองการเคลื่อนที่ของหุ่นยนต์แบบซับซ้อน เพื่อวัดประสิทธิ์ภาพในการทำงานและความทนทานของอัลกอริทึม

ชุดข้อมูลทดลองจะประกอบด้วยข้อมูล 4 ชุดได้แก่

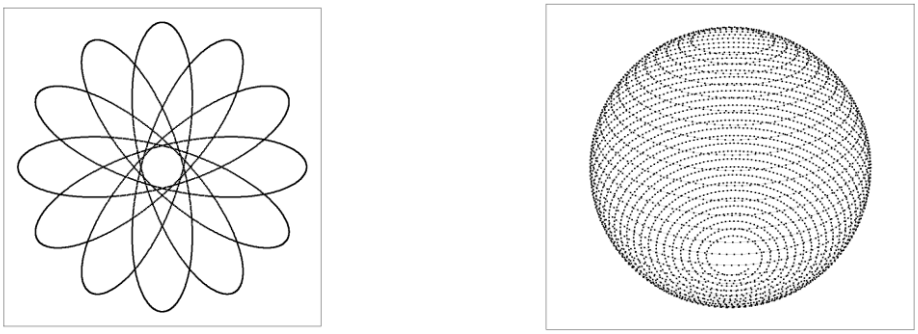
1. การเคลื่อนที่ของหุ่นยนต์แบบสุ่ม(รูปที่ 5.16 (ก)) เป็นการจำลองการเคลื่อนที่ของหุ่นยนต์ในกรณีทั่วไป
2. การเคลื่อนที่ของหุ่นยนต์ลักษณะตาข่าย (รูปที่ 5.16 (ข)) เป็นการจำลองการเคลื่อนที่ของหุ่นยนต์ในกรณีแล้วร้ายสุด โดยจะเป็นกรณีที่หุ่นยนต์ close loop แล้วทำให้ต้องปรับแก้แผนที่ทั้งหมด
3. การเคลื่อนที่ของหุ่นยนต์แบบ Hypotrochoid (รูปที่ 5.16 (ค)) เป็นการจำลองการเคลื่อนที่ของหุ่นยนต์ในกรณีที่มี loop ขนาดใหญ่หลาย loop
4. การเคลื่อนที่ของหุ่นยนต์รอบผิวทรงกลมในสามมิติ (รูปที่ 5.16 (ง)) เป็นการจำลองการเคลื่อนที่ของหุ่นยนต์ในสามมิติ



(ก)



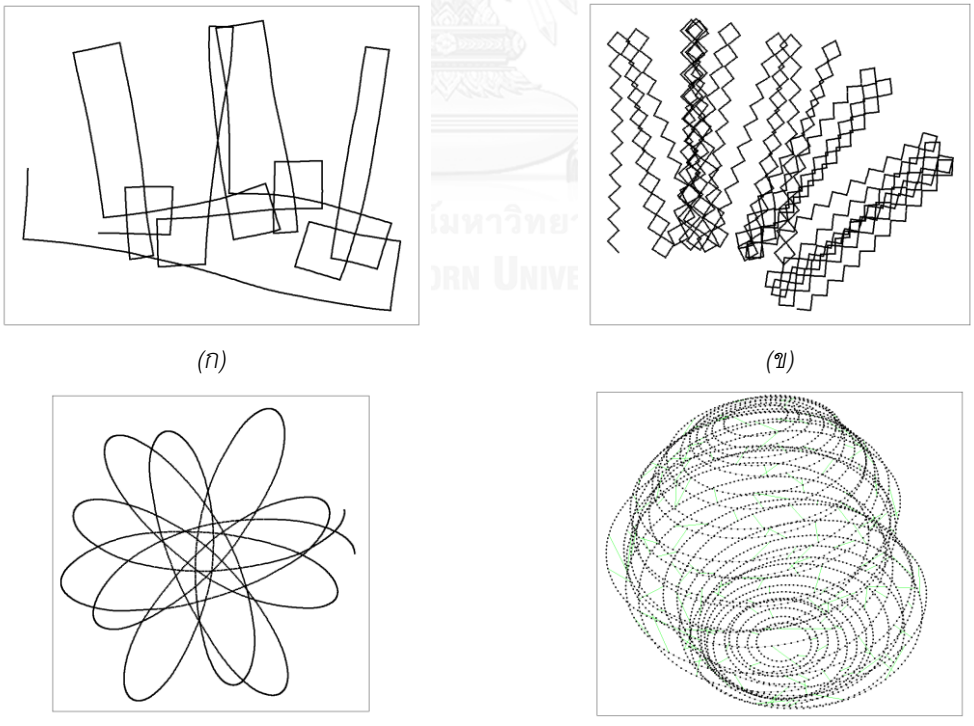
(ข)



(ค) (ง)

รูปที่ 5.16 ชุดข้อมูลทดลอง (ก) การเคลื่อนที่ของหุ่นยนต์แบบสุ่ม, (ข) การเคลื่อนที่ของหุ่นยนต์ลักษณะตาข่าย, (ค) การเคลื่อนที่ของหุ่นยนต์แบบ Hypotrochoid และ (ง) การเคลื่อนที่ของหุ่นยนต์รอบผิวทรงกลมในสามมิติ

การจำลองการเคลื่อนที่ของหุ่นยนต์ในรูปที่ 5.16 จะใช้เป็นข้อมูล ground truth ในการเปรียบเทียบความคลาดเคลื่อนในการปรับแก้แผนที่ เส้นทางการเคลื่อนที่ของหุ่นยนต์คำนวนจากผลบวกของ odometry เพียงอย่างเดียว (ไม่มีการปรับแก้แผนที่) แสดงได้ในรูปที่ 5.17 ซึ่งจะเห็นได้ว่าการประมาณตำแหน่งหุ่นยนต์จาก odometry เพียงอย่างเดียว นั้นมีความคลาดเคลื่อนค่อนข้างมาก โดยเส้นสีเขียวที่ลากเชื่อมระหว่างโหนดนั้นจะเป็น close loop constraint ของแผนที่ซึ่งจะถูกใช้ในการปรับแก้ต่อไป

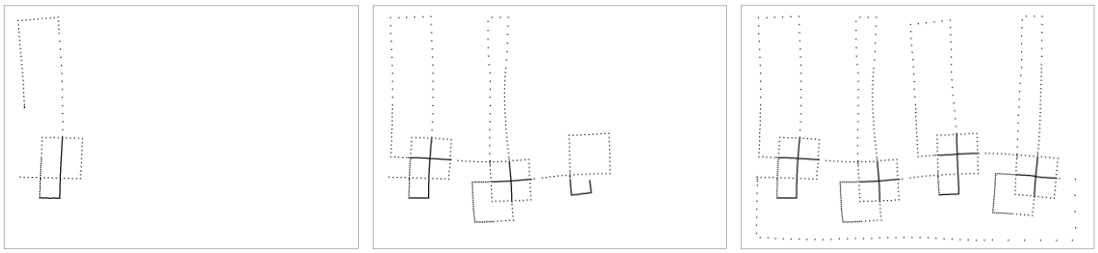


(ก) (ข)

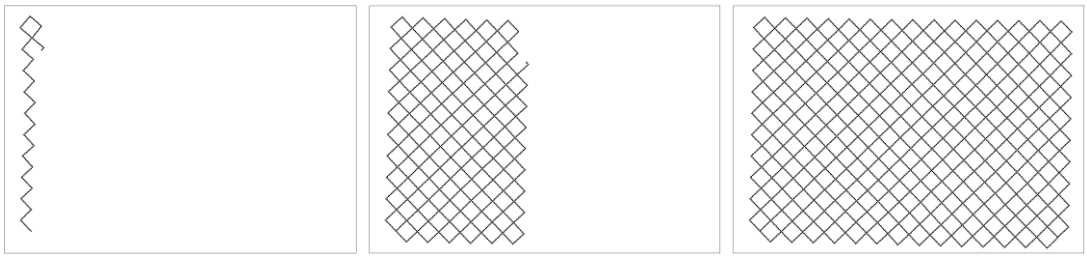
(ค) (ง)

รูปที่ 5.17 เส้นทางการเคลื่อนที่ของหุ่นยนต์คำนวนจาก odometry เพียงอย่างเดียว ของชุดข้อมูลทั้ง 4

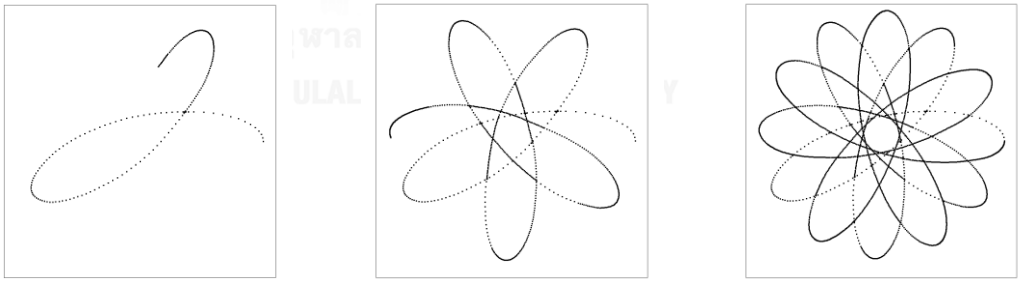
ผลลัพธ์ของการทำงานของอัลกอริทึมแสดงได้ดังรูปที่ 5.18 โดยในคอลัมน์แรกและคอลัมน์ที่สองจะเป็นผลลัพธ์ระหว่างการทำงาน ส่วนคอลัมน์ที่สามจะเป็นผลลัพธ์หลังการทำงานเสร็จสิ้นแล้ว โดยในรูปที่ 5.18 (ก) จะเป็นข้อมูลทดลองการเคลื่อนที่ของหุ่นยนต์แบบสุ่ม, รูปที่ 5.18 (ข) จะเป็นข้อมูลทดลองการเคลื่อนที่ของหุ่นยนต์ลักษณะตาข่าย, รูปที่ 5.18 (ค) จะเป็นข้อมูลทดลองการเคลื่อนที่ของหุ่นยนต์แบบ Hypotrochoid และรูปที่ 5.18 (ง) จะเป็นข้อมูลทดลองการเคลื่อนที่ของหุ่นยนต์รอบผิวทรงกลมในสามมิติ จะเห็นได้ว่าหลังการปรับแก้แผนที่แล้ว แผนที่ที่ได้จะค่อนข้างสอดคล้องกับข้อมูล ground truth มีเพียงชุดข้อมูลแรกที่เส้นทางการเคลื่อนที่จะบิดเบี้ยวเนื่องจากมีการเคลื่อนที่ระยะทางค่อยข้างไกล



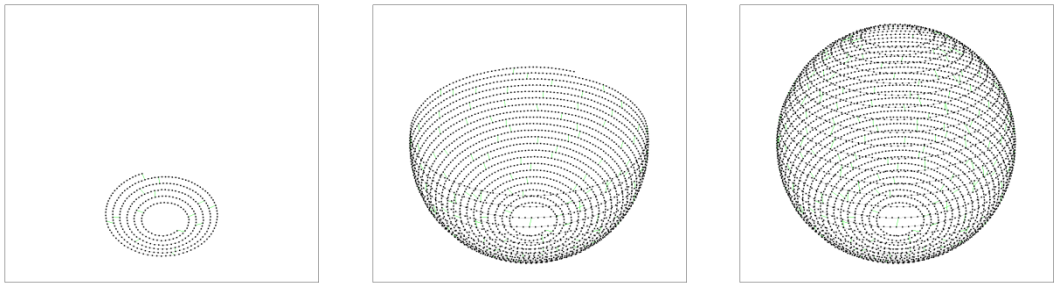
(ก)



(ข)



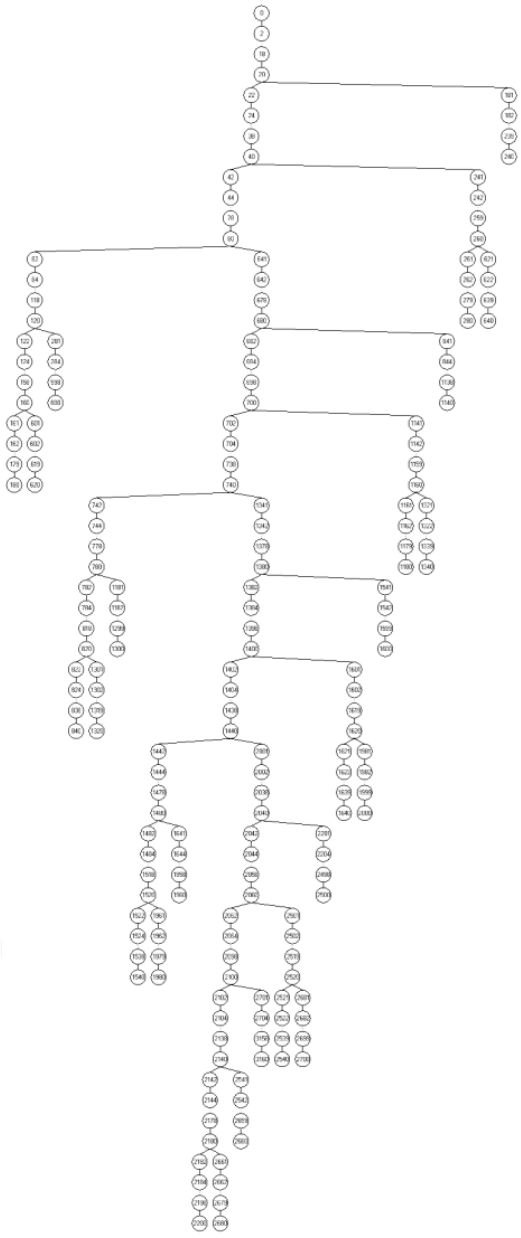
(ค)



(ง)

รูปที่ 5.18 ผลลัพธ์ของการทำงานของอัลกอริทึมการปรับแก้แผนที่ที่ได้นำเสนอ

ตัวอย่างต้นไม้ความสัมพันธ์ของโหนดจะแสดงตัวอย่างได้ดัง*รูปที่ 5.19* ซึ่งเป็นต้นไม้ความสัมพันธ์ของชุดข้อมูลที่ 1 จะเห็นได้ว่าตำแหน่งของหุ่นยนต์ในแต่ละ view ถูกบรรยายด้วยตำแหน่งสัมพัทธ์ (relative transform) เทียบกับโหนดพ่อ เป็นลำดับขั้นขึ้นไปเรื่อย ๆ



รูปที่ 5.19 ต้นไม้ความสัมพันธ์ของชุดข้อมูลที่ 1

สำหรับความคลาดเคลื่อนในการปรับแก้แผนที่เทียบกับข้อมูล ground truth เปรียบเทียบความแม่นยำระหว่างอัลกอริทึมที่ได้นำเสนอ และ TORO แสดงได้ดัง*ตาราง 5.3* โดยจากตารางจะเห็นได้ว่าในทั้งสี่ชุดข้อมูล อัลกอริทึมที่ได้นำเสนอ (หลังทำ Global optimization) จะมีความแม่นยำสูงสุด ส่วนอัลกอริทึมที่ได้นำเสนอที่ไม่ได้ทำ Global optimization จะมีความคลาดเคลื่อนใกล้เคียงกับ TORO อย่างไรก็ดี ความคลาดเคลื่อนหลังการปรับแก้นั้นของทั้งสามอัลกอริทึมนั้นอยู่ในช่วงที่ยอมรับได้

ตาราง 5.3 ตารางความคลาดเคลื่อนการปรับแก้แผนทีของอัลกอริทึมที่ได้นำเสนอเทียบกับ TORO

DataSet	Error (m)		
	Our Approach	Our Approach without Global optimization	TORO
Random walk	0.35694	0.49797	0.85093
net	0.08198	0.66482	0.17658
Hypotrochoid	0.05431	0.15880	0.14829
Sphere	0.03302	0.05301	0.16743

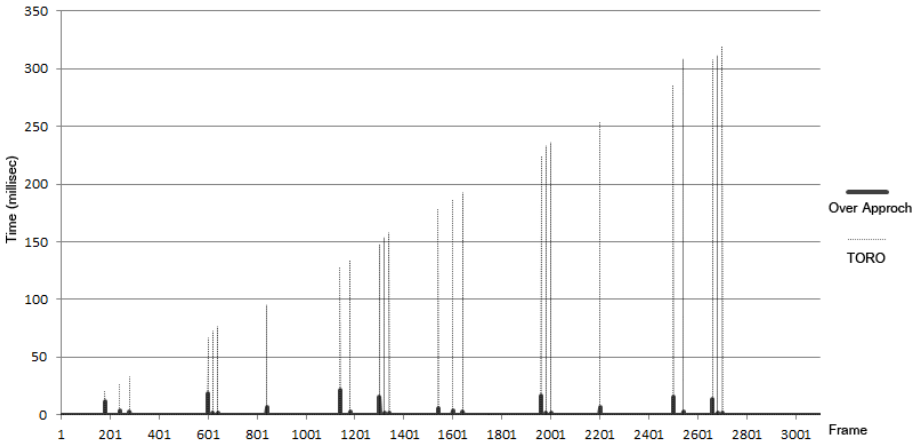
สำหรับเวลาในการทำงานชุดข้อมูลที่ 1 แสดงเป็นกราฟได้ตามรูปที่ 5.20 (ก) โดยกราฟเส้นทึบเป็นเวลาการทำงานของอัลกอริทึมที่ได้นำเสนอ ส่วนกราฟเส้นประเป็นเวลาการทำงานของ TORO จะเห็นได้ว่าช่วงกราฟที่มีเส้นโดดขึ้นมาจะเป็นช่วงที่มีการ close loop เกิดขึ้น ซึ่งสำหรับกราฟของอัลกอริทึมที่ได้นำเสนอเวลาในการ close loop จะค่อนข้างคงที่ เนื่องจากได้จำกัดจำนวน node ที่จะถูกปรับแก้ด้วยวิธี Pose Marginalization ส่วนในกรณีของ TORO จะเห็นได้ว่าทุกครั้งมีมีการ close loop เวลาในการทำงานจะเพิ่มขึ้น เพราะเป็นการปรับแก้ node ทุก node ในแผนที่ และแผนที่มีขนาดใหญ่ขึ้นเรื่อย ๆ

การเปรียบเทียบเวลาในกราฟรูปที่ 5.20 นั้นอาจจะไม่ค่อยยุติธรรมมากนัก เนื่องจากว่าการทำงานของอัลกอริทึมที่ได้นำเสนอ จะมีการสร้าง thread อีก thread ใช้สำหรับการปรับแก้แผนที่แบบ global โดยเฉพาะซึ่งไม่เกี่ยวข้องกับการทำงานหลัก และไม่ถูกนำมาคิดเวลาในการทำงานด้วย อย่างไรก็ดีในการใช้งานจริงจะเห็นได้ว่าอัลกอริทึมที่ได้นำเสนอนั้น สามารถทำงานได้อย่างทันการณ์และท้ายสุดก็จะได้แผนที่ที่มีความถูกต้องสูง ส่วนอัลกอริทึม TORO นั้นจะเหมาะกับการทำงานแบบ offline มากกว่าโดยจะเน้นเก็บข้อมูไว้ก่อนแล้วจึงค่อยปรับแก้แผนที่ครั้งเดียว

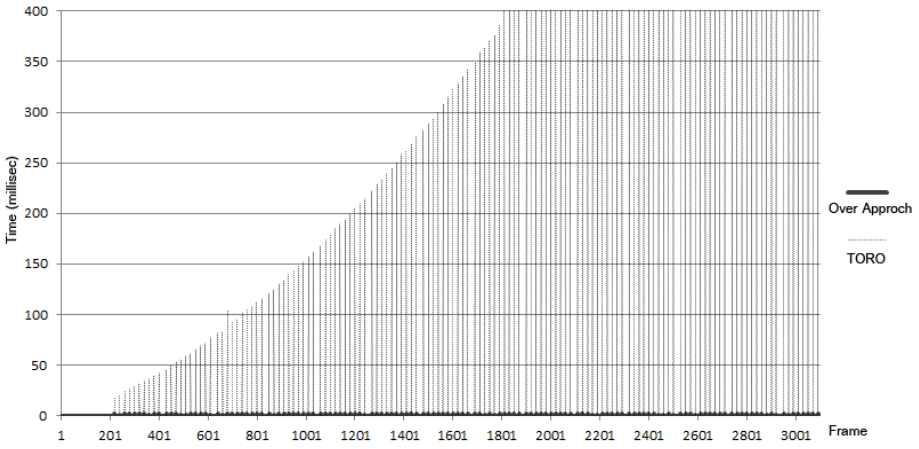
สำหรับกราฟเวลาการทำงานสำหรับชุดข้อมูลที่ 2, 3, และ 4 แสดงได้ดังรูปที่ 5.20 (ข), (ค) และ (ง) ตามลำดับโดยทุกผลการทดลองให้ผลสอดคล้องกันว่า อัลกอริทึมที่ได้นำเสนอใช้เวลาในการทำงานคงที่ ซึ่งเวลาเฉลี่ยของแต่อัลกอริทึมแสดงได้ดังตาราง 5.4

ตาราง 5.4 ตารางแสดงเวลาการทำงานเฉลี่ยต่อเฟรมของอัลกอริทึมที่ได้นำเสนอเทียบกับ TORO

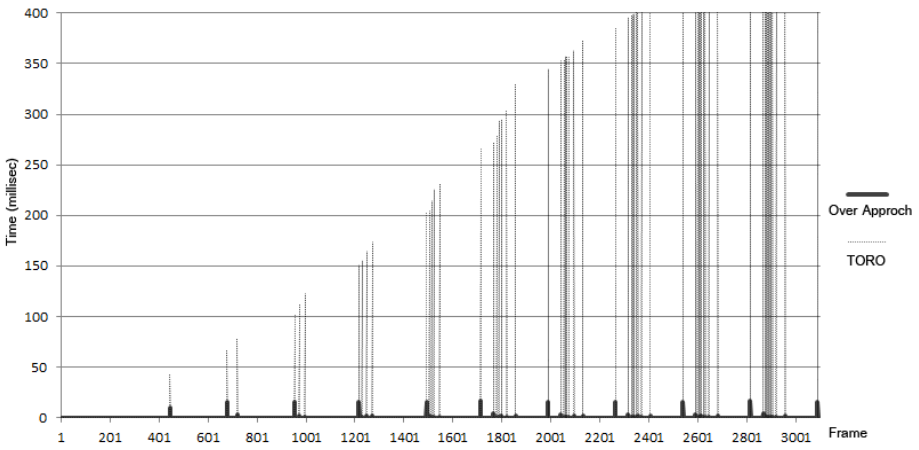
DataSet	Average time per frame (millisec)	
	Our Approach	TORO
Random walk	0.07049	1.43411
net	0.07759	60.08853
Hypotrochoid	0.09037	8.81194
Sphere	0.07054	35.49060


Time (millisec)
350
300
250
200
150
100
50
0
1
201
401
601
801
1001
1201
1401
1601
1801
2001
2201
2401
2601
2801
3001
Frame
Over Approch
TORO

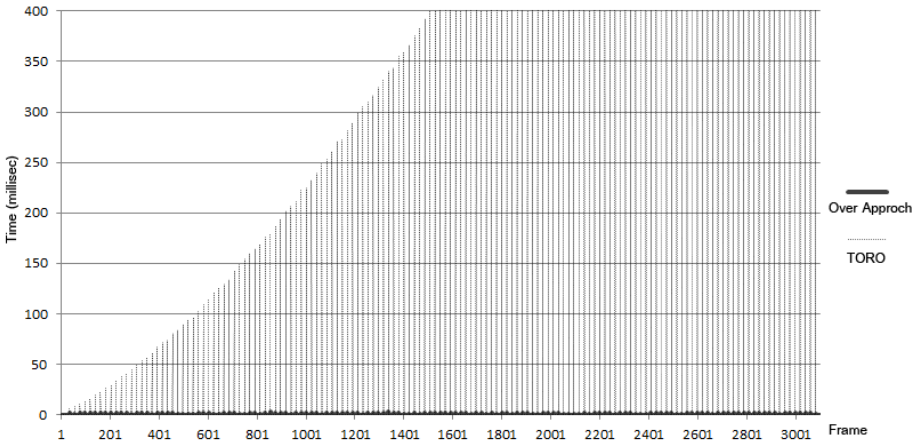
(ก)


Time (millisec)
400
350
300
250
200
150
100
50
0
1
201
401
601
801
1001
1201
1401
1601
1801
2001
2201
2401
2601
2801
3001
Frame
Over Approch
TORO

(ข)


Time (millisec)
400
350
300
250
200
150
100
50
0
1
201
401
601
801
1001
1201
1401
1601
1801
2001
2201
2401
2601
2801
3001
Frame
Over Approch
TORO

(ค)



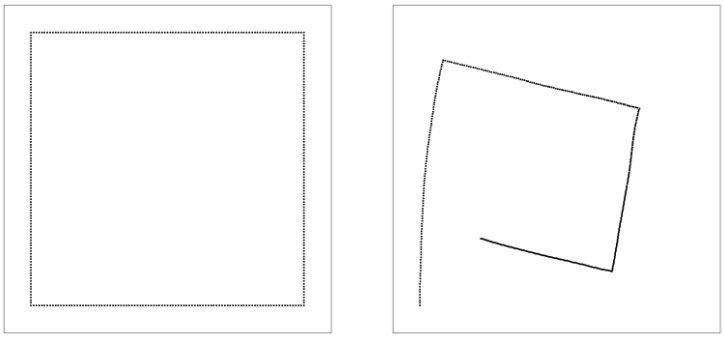
(ง)

รูปที่ 5.20 กราฟแสดงเวลาในการทำงานของอัลกอริทึมที่ได้นำเสนอเทียบกับ TORO

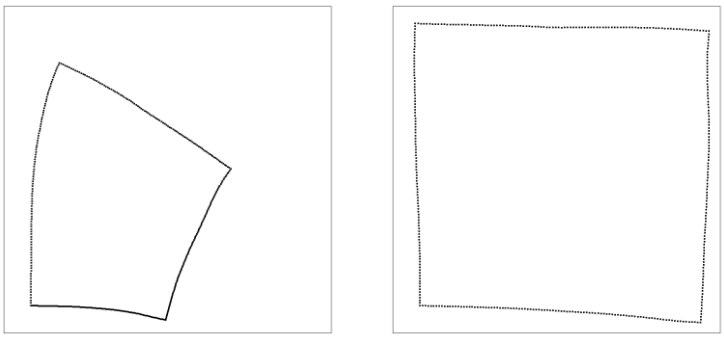
5.3.3 ผลการทดลองการปรับแก้แผนที่ที่มีการเบี่ยงเบนขนาด

สำหรับ Monocular Visual SLAM การประมาณการเคลื่อนที่ไปเรื่อย ๆอาจนำไปสู่ปัญหา Scale Drift ดังนั้นการปรับแก้แผนที่จึงต้องมีการปรับแก้ scale ให้ถูกต้องด้วย

ข้อมูลจำลองที่จะใช้ในการทดลองนี้จะใช้การเคลื่อนที่เป็นรูปสี่เหลี่ยมรูปที่ 5.21 (ก) โดยจะการสุ่มการเปลี่ยนแปลง scale เข้าไปด้วยซึ่งจะส่งผลให้ scale แผนที่ในช่วงต่าง ๆ ของแผนที่มีความคลาดเคลื่อนไปจาก scale จริง แผนที่ Scale Drift ก่อนการปรับแก้แสดงได้ตามรูปที่ 5.21 (ข), แผนที่หลังการปรับแก้ตำแหน่งแบบไม่ปรับแก้ scale แสดงได้ดังรูปที่ 5.21 (ค) และแผนที่หลังการปรับแก้ซึ่งมีการพิจารณาค่า scale ด้วยแสดงได้ดังรูปที่ 5.21 (ง) ซึ่งจะเห็นได้หากไม่มีการพิจารณา scale ในการปรับแก้แผนที่จะทำให้แผนที่มีความผิดเพี้ยนไปจากความเป็นจริงมาก สำหรับความคลาดเคลื่อนของแผนที่เปรียบเทียบระหว่างแบบที่ปรับแก้ scale และไม่ปรับแก้ scale แสดงได้ดังตาราง 5.5



(ก) *(ข)*



(ค) (ง)

รูปที่ 5.21 (ก) ชุดข้อมูลที่ใช้ทดลอง (ข) แผนที่ก่อนการปรับแก้ (ค) แผนที่หลังการปรับแก้แบบไม่พิจารณา scale (ค) แผนที่หลังการปรับแก้โดยพิจารณา scale ด้วย

ตาราง 5.5 ตารางความคลาดเคลื่อนการปรับแก้แผนที่เทียบระหว่างแบบไม่พิจารณา scale และแบบพิจารณา scale

Method	Error (m)
No Scale Adjust	1.05360
Scale Adjust	6.00337

บทที่ 6
การตรวจหา Loop Closure

ในหัวข้อที่ 5 ได้พูดถึงการปรับแก้แผนที่เพื่อหา configuration เหมาะสุดของตำแหน่ง view โดยสมมุติว่าระบบมีความสามารถในการตรวจพบ loop closure ระหว่าง node x_i และ node x_j ในหัวหัวข้อนี้จะนำเสนอขั้นตอนในการตรวจหา Loop Closure เพื่อนำไปสู่การปรับแก้แผนที่ต่อไป

ปัญหาการตรวจหา Loop Closure ถือเป็นปัญหาสำคัญที่สุดปัญหาหนึ่งของ SLAM เนื่องจากหัวใจสำคัญของ SLAM นั้นก็คือการที่หุ่นยนต์ต้องสามารถรู้ตัวว่ากำลังอยู่ส่วนไหนในแผนที่ และการตรวจหา Loop Closure คือการบ่งชี้ว่าหุ่นยนต์ได้กลับมายังตำแหน่งที่เดิมในอดีตแล้ว ปัญหาการตรวจหา Loop Closure ยังถือเป็นปัญหาที่ยุ่งยากที่สุดปัญหาหนึ่งสำหรับ SLAM เนื่องจากขั้นตอนการทำงานนั้นจะต้องเกี่ยวพันกับการค้นหาข้อมูลการวัดในอดีตทั้งหมดเพื่อตรวจหาว่าหุ่นยนต์ได้วนซ้ำกลับมายังตำแหน่งเดิมหรือไม่ ดังนั้นสำหรับแผนที่ที่มีขนาดใหญ่ขึ้น ย่อมจะส่งผลกระทบต่อประสิทธิภาพทางเวลาในการทำงานของการตรวจหา Loop Closure ด้วย ดังนั้นในงานวิจัยต่าง ๆ จะนิยมใช้ข้อมูล vision ในการตรวจหา Loop Closure เนื่องจากข้อมูล vision จะประกอบด้วยข้อมูลภาพสีจำนวนมาก ทำให้สามารถหา feature สำคัญสำหรับการแยกแยะตำแหน่ง view ต่าง ๆ ได้ และสามารถตรวจหาความสอดคล้องของ view ได้ง่าย

Williams et al. [76] ได้แบ่งประเภทการตรวจหา Loop closure ออกเป็นสามประเภทได้แก่ map-to-map, image-to-map และ image-to-image

- map-to-map [66, 71] จะเป็นการตรวจหา Loop closure โดยพิจารณาจากความสัมพันธ์ของแผนที่ย่อย (submap) ที่เวลาปัจจุบันกับแผนที่ย่อย (submap) ในอดีตโดยการตรวจหาจะใช้ลักษณะเฉพาะของโครงสร้างเรขาคณิตของแผนที่ย่อยในการเปรียบเทียบความสอดคล้อง ซึ่งแผนที่ที่จะใช้การตรวจหาแบบ map-to-map ได้ จะต้องอยู่ในรูป Local map จำนวนมาก

- image-to-map [77-79] จะเป็นการตรวจหา Loop closure โดยพิจารณาจากความสัมพันธ์ของแผนที่เฉพาะส่วนและภาพล่าสุด โดยการเปรียบเทียบนั้นจะใช้โครงสร้างเรขาคณิตของแผนที่ เทียบกับตำแหน่งจุดสังเกตในภาพล่าสุดที่ได้จากกล้อง ซึ่งการใช้ข้อมูล vision ร่วมกับข้อมูลเรขาคณิต จะทำให้สามารถตรวจหา Loop closure ได้แม่นยำมากขึ้น

- image-to-image [80] จะเป็นการตรวจหา Loop closure โดยพิจารณาจากความสัมพันธ์ระหว่างภาพ ณ เวลาปัจจุบัน เทียบกับภาพต่าง ๆ ในอดีตเพียงอย่างเดียว โดยไม่สนใจข้อมูลเรขาคณิตของแผนที่หรือตำแหน่งจุดสังเกตในภาพ การหาความสอดคล้องระหว่างภาพสองภาพจะอาศัย Bag of Words [128] เพื่อการช่วยในการจัดกลุ่ม SIFT Features, SUFT Features หรือ MSER Features ให้เป็น Visual Vocabulary จากนั้นก็พยายามอธิบายภาพด้วย vector ของ words เพื่อให้สามารถทำการเปรียบเทียบคู่ภาพได้อย่างรวดเร็ว

สาเหตุที่วิธีตรวจหา Loop Closure แบบ image-to-image เป็นที่นิยมนั้นเนื่องจาก อัลกอริทึมสามารถทำงานได้อย่างรวดเร็ว มีความซับซ้อนน้อยและสามารถทำงานได้เป็นเอกเทศ ไม่ต้องยุ่งกับการทำงานส่วนอื่นของ SLAM อย่างไรก็ดีวิธีตรวจหา Loop Closure แบบ image-to-image นั้นมีปัญหาใหญ่ซึ่งไม่เหมาะสมในการใช้งานร่วมกับกล้องวิดีโอมุมกว้าง โดยข้อจำกัดของอัลกอริทึมแบบ image-to-image ก็คือคู่ภาพที่สามารถตรวจหา Loop Closure ได้นั้นจะต้องมีมุมมองที่คล้ายคลึงกันมากระดับหนึ่ง ตัวอย่างของผลลัพธ์จากอัลกอริทึม FAB-MAP [80] แสดงได้ในรูปที่ 6.1 ซึ่งในความเป็นจริงแล้วการ close loop ของ SLAM นั้นมีได้หลายแบบไม่ว่าจะเป็นการเคลื่อนที่แบบวนบรรจบ การเคลื่อนที่แบบตัดกัน หรือการเคลื่อนที่สวนทางกัน โดยภาพที่ตรวจวัดจากกล้องวิดีโอมุมกว้างแสดงได้ดังรูปที่ 6.2 จะเห็นได้ว่ามุมมองของภาพมีความแตกต่างกันมากและไม่สามารถตรวจวัดได้ด้วยวิธี image-to-image

รูปที่ 6.1 ผลลัพธ์คู่ภาพจากอัลกอริทึม FAB-MAP [80] ซึ่งสามารถตรวจพบ loop closure

รูปที่ 6.2 คู่ภาพจากกล้องวิดีโอมุมกว้าง สำหรับกรณี close loop

นอกจากนี้ปัญหาอีกประการของอัลกอริทึมแบบ image-to-image คือเกิด False Detection ได้ง่ายเนื่องจากในบางครั้งสภาพแวดล้อมในการทำงานอาจมีความคล้ายคลึงกันในบางจุดทำให้เกิดการผิดพลาดในการตรวจวัด ดังนั้นการใช้งานจริงจึงต้องมีการทวนสอบด้วยวิธี image-to-map หรือ map-to-map

ในวิทยานิพนธ์ฉบับนี้จะนำเสนอวิธีตรวจหา Loop Closure แบบ map-to-map โดยจะอาศัยข้อมูล Vision ในการตรวจหาร่วมด้วย เพื่อให้การตรวจหา Loop Closure ไม่ต้องจำกัดมุมมองของภาพ การทำงานของวิธี map-to-map นั้นจะเป็นการหาความสัมพันธ์ระหว่าง Submap ณ เวลาปัจจุบันกับ Submap ในอดีต โดยข้อดีของการใช้วิธี map-to-map ก็คือ submap จะถูกสร้างขึ้นจาก View หลาย View ดังนั้นจึงมีข้อมูลสิ่งแวดล้อมมากกว่าภาพภาพเดียว ทำให้สามารถตรวจหา Loop Closure ได้ดีกว่าและมีโอกาสเกิด False Detection ต่ำ เนื่องจากมีการใช้ข้อมูล geometry จาก submap ร่วมด้วย นอกจากนี้ผลลัพธ์การตรวจหา Loop

Closure ด้วยวิธี map-to-map ยังได้ข้อมูลความสัมพันธ์ระหว่างจุดสังเกต ใน submap ก่อนหน้าและ submap ปัจจุบันด้วย ทำให้สามารถนำข้อมูลไปใช้ในการรวมแผนที่ได้

ในหัวข้อนี้จะอธิบายการทำงานของการตรวจหา Loop Closure โดยจะแบ่งเป็นสองส่วนด้วยกันคือ การเปรียบเทียบคู่ Submap พร้อมกับการการประมาณการเลื่อนตำแหน่งสัมพัทธ์ (Submap Comparison and Relative Transform Estimation) และการเลือก Candidate Submap (Candidate Submap Selection)

6.1 การเปรียบเทียบ Submap และการประมาณตำแหน่งสัมพัทธ์

สมมิตให้มีคู่ submap (S_u, S_v) ซึ่งมีความเป็นไปได้ที่จะเกิด close loop การทำงานในหัวข้อนี้จะเป็นการประเมินความเป็นไปได้ของความสอดคล้องของคู่ submap และการตรวจหาตำแหน่งสัมพัทธ์ของทั้งสอง submap

กำหนดให้ $u = \{u_0, u_1, \dots, u_m\}$ เป็นเซตของตำแหน่งจุดสังเกตใน world coordinate ของ submap ปัจจุบัน และ $v = \{v_0, v_1, \dots, v_n\}$ เป็นเซตของตำแหน่งจุดสังเกตใน world coordinate ของ submap ก่อนหน้า ขั้นตอนการเปรียบเทียบคู่ submap จะเป็นการหาเซตของคู่ความสัมพันธ์จุดสังเกต $p_{ij} = (u_i, v_j)$ ที่มี descriptor ใกล้เคียงกันซึ่งทำให้ตำแหน่งสัมพัทธ์ (relative transform) ระหว่างสอง submap มีความสอดคล้องกัน ขั้นตอนการทำงานจะเริ่มจากการคำนวนตำแหน่งของจุดสังเกตใน world coordinate ซึ่งสอดคล้องกับ view ใน submap นั้น ๆ จากนั้นจึงค้นหาคู่ความสัมพันธ์จุดสังเกตระหว่าง submap ปัจจุบัน และ submap ก่อนหน้า โดยใช้วิธี k-Nearest Neighbors Search ในการเปรียบเทียบ descriptor ของจุดสังเกต หลังจากนั้นจึงใช้ RANSAC ในการประมาณตำแหน่งสัมพัทธ์ของทั้งสอง submap และกรองคู่จุดที่ไม่สอดคล้องทางตำแหน่ง โดยมีขั้นตอนการทำงานของ RANSAC จะเป็นดังต่อไปนี้

1. ทำการสุ่มคู่จุดสังเกต p_{ij} จำนวน 3 คู่ จากนั้นจึงประมาณตำแหน่งสัมพัทธ์ (relative transform) ($M = [sR|t]$) ด้วยวิธี “Least-Squares Rigid Motion Using SVD” [129] กำหนดให้ R เป็น rotation matrix, t เป็น translation vector และ s เป็น scale factor

2. คำนวนหา error ของคู่จุดสังเกตทั้งหมด โดย error (e_{ij}) หาได้จากการวัดระยะทางด้วย Eclidian distance ระหว่างตำแหน่งจุดสังเกต u_i และตำแหน่งจุดสังเกต v_j โดยต้องแปลงตำแหน่งจุดสังเกต v_j ด้วย relative transform M

$$e_{ij} = \left\| u_i - M \oplus v_j \right\|_2$$

3. ตรวจหา inliner โดยเลือกเฉพาะ error ที่มีค่าต่ำกว่า threshold จากนั้นจะลบคู่จุดสังเกตที่มีความซ้ำซ้อน เช่น จุดสังเกตหนึ่งจุดอาจจับคู่กับอีก submap ได้หลายจุดสังเกต โดยการเรียงลำดับ inliner จากค่า error น้อยไปมาก จากนั้นสำหรับคู่จุดสังเกตที่มีความซ้ำซ้อน ก็ให้เลือกคู่จุดสังเกตที่มี error น้อยกว่า

4. ทำซ้ำขั้นตอนที่ 1 จนได้ hypothesis ที่มี inliner มากสุด

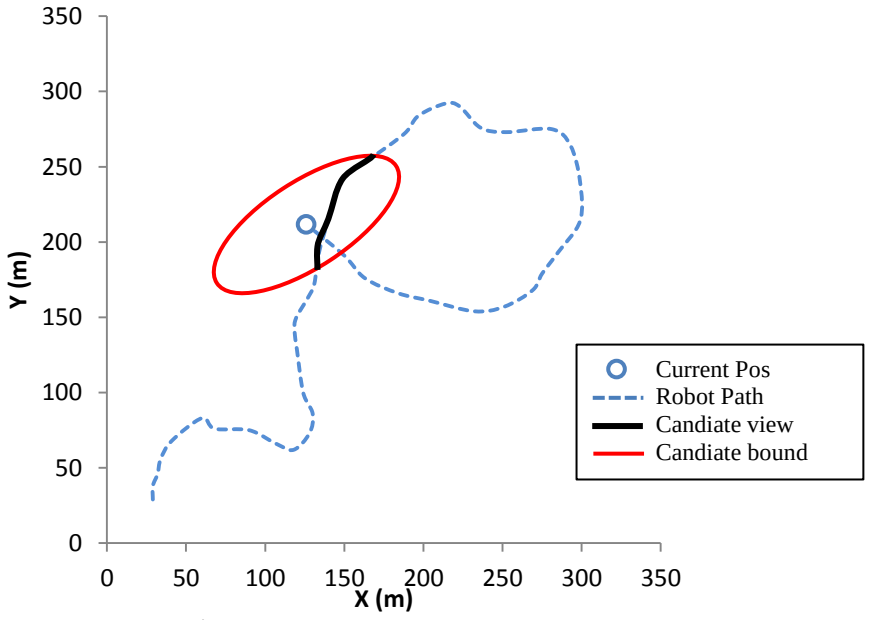
5. ประมาณ relative transform อีกครั้งโดยใช้คู่จุดสังเกตใน inliner ทั้งหมด

การพิจารณาว่า submap ทั้งสองมีความสัมพันธ์กันพอจะเกิด Loop Closure หรือไม่นั้น ดูได้จากจำนวน inliner ซึ่งถ้า inliner มีจำนวนมากกว่าค่าที่กำหนดก็ให้ถือว่า submap ทั้งสองมีความสัมพันธ์กันจากนั้นก็เข้ากระบวนการ close loop และการปรับแก้แผนที่ต่อไป ส่วนเซตของคู่ความสัมพันธ์จุดสังเกต $\{p_{ij}\}$ จะใช้ในการรวมแผนที่ซึ่งจะพูดถึงในหัวข้อที่ 6.3

6.2 การเลือก Candidate Submap

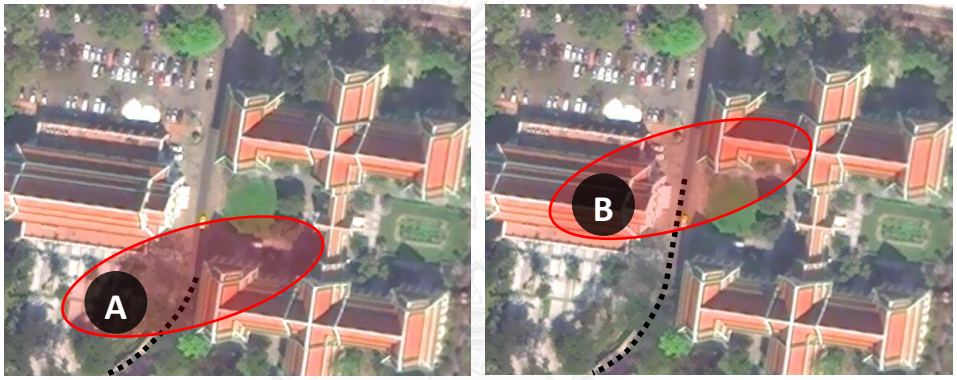
ในหัวข้อที่ 6.1 นั้นได้กล่าวถึงการเปรียบเทียบคู่ Submap ปัจจุบันและ Submap ในอดีตเพื่อหาความน่าจะเป็นในการเกิด loop closure สำหรับ Submap ปัจจุบันที่ใช้ในการเปรียบเทียบจะประกอบด้วย View ที่อยู่ใน Sliding Window ซึ่งถูกใช้ในการประมาณการเคลื่อนที่จากข้อมูลภาพในหัวข้อที่ 4.1 ส่วน Submap ในอดีตจะเลือกจากกลุ่มของ View ในอดีตที่อยู่ใกล้กัน

โดยปรกติแล้ว Candidate View ของ loop closure นั้นจะถูกเลือกโดยใช้ประโยชน์จากค่าประมาณ covariance ของตำแหน่งหุ่นยนต์ที่เวลาปัจจุบัน ดังแสดงได้รูปที่ 6.3 จากรูปวงรีรอบ ๆ ตำแหน่งของหุ่นยนต์แสดงการกระจายความน่าจะเป็นของตำแหน่งหุ่นยนต์ การเลือก Candidate นั้นจะเลือกโดยพิจารณาจากค่า Mahalanobis distance โดยเลือกตำแหน่งซึ่งมีค่า Mahalanobis distance น้อยกว่าค่าคงที่ค่าหนึ่งเช่นพื้นที่บริเวณข้างในวงรีเป็นตำแหน่งที่ Mahalanobis distance มีค่าน้อยกว่า 1 ในกรณีนี้ตำแหน่งหุ่นยนต์ล่าสุด k เฟรมจะไม่ถูกพิจารณาเนื่องจากอยู่ใกล้ตำแหน่งปัจจุบันมากไป

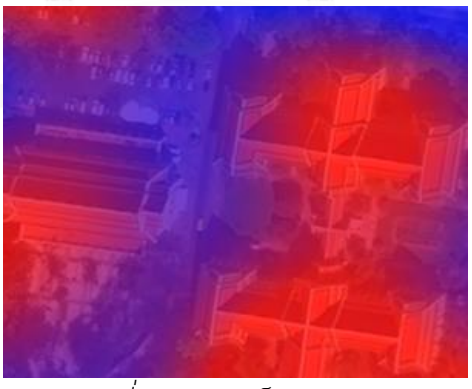


รูปที่ 6.3 แสดงวงรี covariance ของตำแหน่งหุ่นยนต์

การค้นหาโดยใช้ covariance นั้นจะช่วยจำกัดบริเวณความน่าจะเป็นที่จะตรวจพบ loop closure ทำให้ไม่ต้องทำการค้นหาทั้งแผนที่ อย่างไรก็ดีในกรณีที่หุ่นยนต์เคลื่อนที่เป็นระยะทางไกลมาก ๆ หุ่นยนต์จะมีความคลาดเคลื่อนของตำแหน่งสูง และจะทำให้ขอบเขตบริเวณในการค้นหากว้างขึ้นไปด้วย เมื่อพิจารณาวิธีค้นหาด้วย covariance จะพบความซ้ำซ้อนในการทำงาน ยกตัวอย่างเช่นจากรูปที่ 6.4 สมมุติว่าที่เวลา t ระบบได้ทำการค้นหา loop closure ภายในบริเวณ A ในวงรี แต่ไม่พบ loop closure จากนั้นที่เวลา $t+1$ เมื่อหุ่นยนต์เคลื่อนที่ไปทางเหนือเป็นระยะทาง 5 เมตร (สามารถทราบได้จากการประมาณการเคลื่อนทีของหุ่นยนต์) จะเห็นได้ว่าโอกาสที่บริเวณ B จะเกิด loop closure ย่อมเป็นไปได้น้อยกว่าบริเวณอื่น เนื่องจากที่เวลาก่อนหน้านั้นได้มีการค้นหาแล้วแต่ไม่พบความสัมพันธ์ ในทางตรงกันข้าม ถ้าหากว่าการค้นหาในบริเวณ A ที่เวลา t พบความน่าจะเป็นที่จะเกิดของ loop closure ที่เวลา $t+1$ โอกาสที่บริเวณ B จะเกิด loop closure ย่อมเป็นไปได้มากกว่าบริเวณอื่น



รูปที่ 6.4 แสดงวงรี covariance ของตำแหน่งหุ่นยนต์ที่เวลา t (ซ้าย) และ t + 1 (ขวา)



รูปที่ 6.5 แสดงแผนที่ความน่าจะเป็นของการเกิด loop closure

ในงานวิจัยนี้จะเสนอวิธีจัดการความน่าจะเป็นของพื้นที่ในการค้นหา เพื่อให้สามารถเลือก Candidate ได้อย่างเหมาะสม และลดความซ้ำซ้อนในการค้นหา loop closure แนวคิดในการเลือก Candidate นั้นจะเริ่มจากการสร้างแผนที่ความน่าจะเป็นของการเกิด loop closure จากนั้นในแต่ละรอบการทำงานจะเลือก Submap Candidate ตามความน่าจะเป็นในแต่ละบริเวณ ตัวอย่างสร้างแผนที่ความน่าจะเป็นแสดงได้ดังรูปที่ 6.5 โดยพื้นที่สีแดงจะเป็นบริเวณที่มีโอกาสเกิด loop closure ต่ำส่วนพื้นที่สีน้ำเงินจะมีโอกาสเกิด loop closure สูง

ดังนั้นในการสุ่มเลือก Candidate submap ก็จะมีโอกาสสุ่มเลือกพื้นที่บริเวณสีน้ำเงินมากกว่า หลังจากทำการประเมินความเป็นไปได้ในการเกิด loop closure ของคู่ submap แล้ว ก็จะทำการปรับแก้แผนที่ความน่าจะเป็นโดยเมื่อผลการประเมินเป็นลบ (คู่ submap ไม่สอดคล้องกัน) แผนที่บริเวณนั้นก็จะมีการกระจายความน่าจะเป็นต่ำลง แต่ถ้าผลการเปรียบเทียบเป็นบวก (คู่ submap มีความสอดคล้องกัน) แผนที่บริเวณนั้นก็จะมีการกระจายความน่าจะเป็นมากขึ้น นอกจากนี้เมื่อหุ่นยนต์มีการเคลื่อนที่ แผนที่ความน่าจะเป็นก็จะเปลี่ยนแปลงตามการเลื่อนตำแหน่งของหุ่นยนต์ด้วย

สำหรับวิธีในการบรรยายแผนที่ความน่าจะเป็นนั้น เนื่องจากว่าการกระจายความน่าจะเป็นไม่ได้มีการกระจายแบบ Gaussian ดังนั้นจึงไม่สามารถอธิบายด้วย mean และ covariance ได้ ในกรณีนี้จะใช้เซตของ particle มาใช้แทนการกระจายความน่าจะเป็น และใช้ particle filter มาจัดการการเปลี่ยนแปลงความน่าจะเป็นที่เกิดขึ้น

การบรรยายการกระจายความน่าจะเป็นด้วย particle นั้นจะแสดงค่าความเชื่อมั่นของสถานะระบบในรูปของ เซตตัวอย่างสถานะระบบ (Particles Set) ที่ทราบจำนวน โดยแต่ละตัวอย่างประกอบด้วยสถานะของระบบและ weight ของตัวอย่างนั้น สำหรับกรณีนี้สถานะของตัวอย่าง (particle) จะประกอบด้วยตำแหน่งและทิศทางของกล้อง ความเชื่อมั่นของการเกิด loop closure ณ บริเวณหนึ่ง ๆ พิจารณาได้จาก ความหนาแน่นของ particle ที่กระจายอยู่บริเวณนั้น

ขั้นตอนการทำงานของการหา Candidate submap ด้วย Particle Filter จะประกอบด้วย 4 ขั้นตอน ได้แก่ การประมาณการเปลี่ยนแปลงของระบบ (Prediction), การคัดเลือก Candidate submap (Candidate Selection), การปรับแก้สถานะระบบ (Update) และการปรับปรุงเซตตัวอย่าง (Resampling)

Initialization

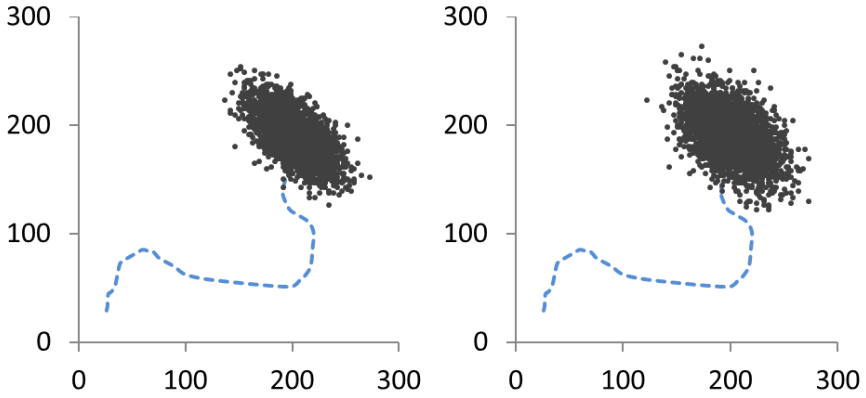
กำหนดให้ $x_t^i = \left[t_t^i \;\; q_t^i\right]^T$ เป็นสถานะของตัวอย่างที่ i ที่เวลา t สถานะของตัวอย่างจะประกอบด้วยตำแหน่งของกล้องในสามมิติ t_t^i และทิศทางของกล้อง q_t^i บรรยายด้วย quaternion rotation ณ เวลาเริ่มต้น ระบบจะสร้างตัวอย่างเป็นจำนวน I ตัวอย่าง โดยให้มีสถานะของระบบในแต่ละตัวอย่างเท่ากับค่าเริ่มต้นของตำแหน่งและทิศทางของกล้อง

Prediction

เมื่อกล้องมีการเคลื่อนที่ไป และระบบสามารถประมาณการเคลื่อนที่ของกล้อง (u_t) และความคลาดเคลื่อนในการประมาณ (covariance Q_t) ได้ตามวิธีที่อธิบายในหัวข้อที่ 4 ระบบจะนำเอาค่าประมาณการเคลื่อนที่ของกล้อง มาประมาณการเปลี่ยนแปลงสถานะของตัวอย่างในแต่ละตัวอย่าง โดยคำนวนได้จาก

$$x_{t+1}^i = f\left(x_t^i, u_t\right) + w_t^i$$

เมื่อ $f(x_t^i, u_t)$ เป็นฟังก์ชันการเคลื่อนที่ของหุ่นยนต์ และ u_t เป็นค่าประมาณการเคลื่อนที่ของกล้อง ส่วน w_t^i เป็น gaussian noise มี mean เป็น 0 และ covariance เป็น Q_t ซึ่งจะถูกสุ่มเพิ่มเข้าไปในแต่ละตัวอย่าง *รูปที่ 6.6* แสดงการกระจายตัวของตำแหน่งของ particle ก่อนและหลังการประมาณการเปลี่ยนแปลงของระบบ



รูปที่ 6.6 แสดงการกระจายตัวของตำแหน่งของ particle ก่อน (ซ้าย) และหลังการประมาณการเปลี่ยนแปลงของระบบ (ขวา)

Candidate Selection

การเลือก Candidate Submap สำหรับการตรวจหา loop closure นั้น จะสุ่มเลือกกลุ่ม View ตามการกระจายความน่าจะเป็นโดยมีขั้นตอนการทำงานดังต่อไปนี้

1. สำหรับ view แต่ละ view จะคำนวนความเชื่อมั่น โดยการวัดความหนาแน่นของ ตัวอย่างที่อยู่ในบริเวณรัศมีรอบ ๆ view นั้น
2. ทำการสุ่มเลือก view ตามค่าความเชื่อมั่นที่คำนวนได้ในข้อ 1 โดย view ที่มีความเชื่อมั่นมาก ก็จะมีโอกาสถูกสุ่มพบมาก
3. เลือก view อื่น ๆ ที่อยู่ใกล้เคียง view ที่ถูกสุ่มเลือกจำนวน k view เพื่อสร้างเป็น Candidate Submap สำหรับการตรวจหา loop closure

การเลือก Candidate Submap นั้นจะเลือกเป็นจำนวนคงที่ค่าหนึ่งเพื่อให้ขั้นตอนในการตรวจหา loop closure นั้นใช้เวลาคงที่

Update

หลังจากที่เลือก Candidate Submap ได้แล้ว ก็จะนำ submap นั้นไปหาความสัมพันธ์จุดสังเกตเทียบกับ submap ปัจจุบันโดยใช้วิธีที่ได้นำเสนอไปในหัวข้อที่ 6.1 ผลลัพธ์ในการค้นหาความสัมพันธ์จะเป็นไปได้ 3 กรณีคือ

1. ไม่พบความสัมพันธ์ของจุดสังเกตระหว่าง submap เลย
2. พบคู่ความสัมพันธ์บ้าง แต่ยังไม่มีความมั่นใจมากพอที่จะ close loop
3. พบคู่ความสัมพันธ์จำนวนมาก และเพียงพอสำหรับการ close loop

ในกรณีที่ 3 นั้นหลังจากระบบตรวจพบ loop closure แล้ว ระบบจะทำการปรับแก้แผนที่ตามที่ได้เสนอในหัวข้อที่ 5 และทำการ Initialize ค่าของเซตตัวอย่างใหม่ ส่วนในกรณีที่ 1 และ 2 นั้นจะทำการปรับแก้ weight ของตัวอย่าง และ Resampling กลุ่มตัวอย่างให้มีการกระจายอย่างเหมาะสมต่อไป

สำหรับกรณีที่ 1 ระบบไม่พบความสัมพันธ์ของจุดสังเกตระหว่าง submap การปรับแก้ weight จะต้องลดน้ำหนักของตัวอย่างที่อยู่ในบริเวณ Submap นั้น ซึ่ง weight ใหม่คำนวนได้จาก ระยะทางกำลังสองระหว่างจุดศูนย์กลาง Submap และ ตำแหน่งตัวอย่าง คูณด้วยค่าคงที่ค่าหนึ่ง

$$w^i = \min\left(1, \rho \cdot \text{distance}\left(t^i, c\right)\right)$$

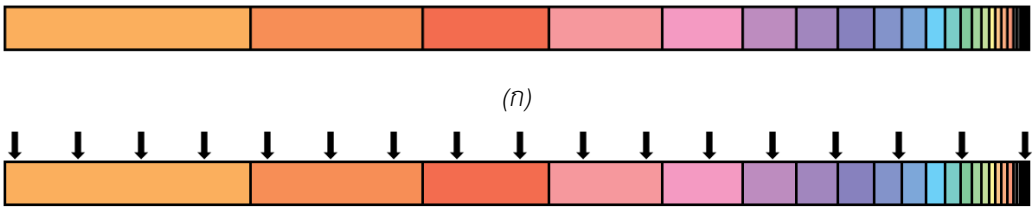
เมื่อ c คือจุดศูนย์กลาง Submap และ ρ คือค่าคงที่ค่าหนึ่งซึ่งใช้ปรับความกว้างของบริเวณ ที่กลุ่มตัวอย่างจะได้รับผลกระทบ

สำหรับกรณีที่ 2 ระบบพบคู่ความสัมพันธ์ระหว่าง submap บ้าง การปรับแก้ weight จะต้องเพิ่มน้ำหนักของตัวอย่างที่อยู่ในบริเวณ Submap นั้นเพื่อให้มีโอกาสที่จะถูกค้นหามากขึ้นในรอบถัดไป ซึ่ง weight ใหม่คำนวนได้จาก

$$w^i = 1 + \exp\left(-\rho \cdot \text{distance}\left(t^i, c\right)\right)$$

Resampling

กระบวนการ resampling จะเป็นการสุ่มเพื่อปรับรูปแบบการกระจายตัวของกลุ่มตัวอย่าง โดยโอกาสอยู่รอดของตัวอย่างหนึ่ง ๆ จะขึ้นอยู่กับค่า weight ของตัวอย่างนั้น ๆ สำหรับวิธีการ implement นั้นจะใช้การเรียงลำดับตัวอย่างเป็นช่วงโดยเปรียบเทียบค่า weight จากมากไปน้อย สำหรับตัวอย่างที่มีค่า weight มากก็จะกินช่วงพื้นที่มากดังแสดงได้ดังรูปที่ 6.7 (ก) จากนั้นจึงทำการสุ่มแบบยูนิฟอร์มบนช่วงตัวอย่างเพื่อเลือกเซตของกลุ่มตัวอย่างชุดต่อไป แสดงได้ดังรูปที่ 6.7 (ข)

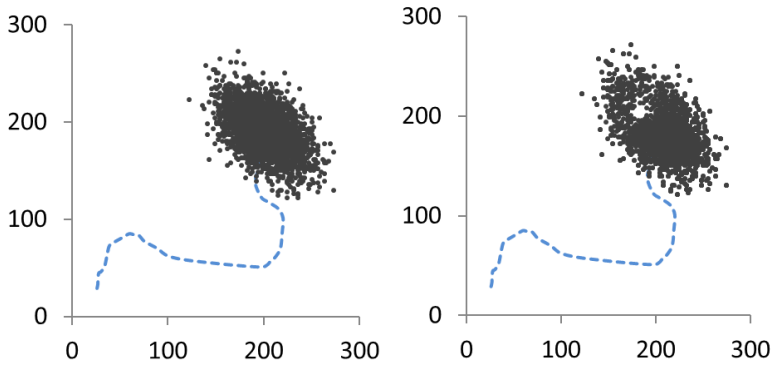


(ก)

(ข)

รูปที่ 6.7 วิธีการ implement กระบวนการ resampling

ผลลัพธ์ก่อนและหลังการ update และ resampling แสดงได้ดังรูปที่ 6.8 (ก) และ (ข) ตามลำดับ



รูปที่ 6.8 ผลลัพธ์ก่อน (ซ้าย) และหลัง (ขวา) การ update และ resampling

ความต่างระหว่าง Fast SLAM และการใช้ particle filter ในการบรรยายแผนที่ความน่าจะเป็น ก็คือ สำหรับ Fast SLAM นั้น ในหนึ่ง particle จะแทนสถานะของระบบทั้งหมดอันประกอบด้วยตำแหน่งของหุ่นยนต์ปัจจุบันและแผนที่ ในขณะที่การบรรยายแผนที่ความน่าจะเป็นในหัวข้อนี้ ในหนึ่ง particle จะแทนแค่ตำแหน่งและทิศทางของกล้องเท่านั้น ทำให้เวลาที่ใช้ในการทำงานคงที่ (ขึ้นกับจำนวน particle) ไม่ขึ้นกับขนาดแผนที่

6.3 การรวม Submap

หลังจากการตรวจพบ loop closure ระบบจะทำการรวมกลุ่มจุดสังเกตใน submap ปัจจุบันเข้ากับกลุ่มจุดสังเกตของ submap ในอดีตที่มีความสัมพันธ์กันโดยอาศัยเซตของคู่ความสัมพันธ์จุดสังเกต $\{p_{ij}\}$ ที่หาได้จากกระบวนการในหัวข้อที่ 6.1 โดยมีการทำงานดังต่อไปนี้

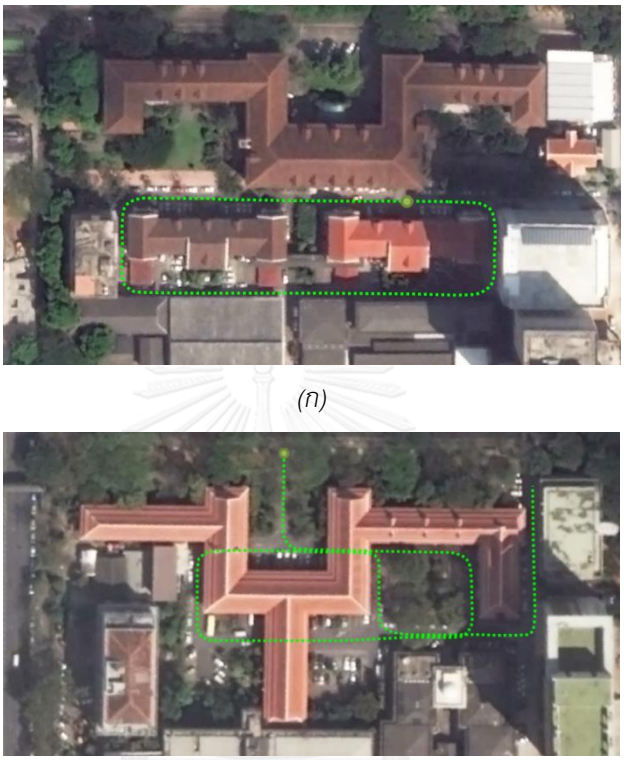
1. สำหรับคู่จุดสังเกต (u_i, v_j) ที่มีความซ้ำซ้อนกัน จะทำการรวมค่าการวัดของทั้งสองจุดสังเกตเข้าด้วยกัน จากนั้นจึงลบจุดสังเกตใน submap ปัจจุบันทิ้งไป
2. หลังจากการลบจุดสังเกตซ้ำซ้อนแล้ว ระบบจะรวม view ในสอง submap เข้าด้วยกัน แล้วจึงทำ bundle Adjustment เพื่อปรับแก้ตำแหน่งจุดสังเกตทั้งหมด

จากนั้นระบบก็จะทำการปรับแก้แผนที่ตามกระบวนการในหัวข้อที่ 5 ต่อไป

6.4 การทดลองการตรวจหา Loop Closure

การทดลองนี้ จะแสดงผลการตรวจหา Loop Closure เปรียบเทียบระหว่างอัลกอริทึมที่ได้นำเสนอ กับอัลกอริทึม Fast Appearance-based Mapping (FABMAP 2.0) [80, 84] ซึ่งเป็นอัลกอริทึมสำหรับตรวจหา Loop Closure แบบ Image-to-Image ที่นิยมชนิดหนึ่ง ถูกนำเสนอครั้งแรกโดย Mark Cummins และ Paul Newman ในปี 2008 นอกจากนี้จะเปรียบเทียบกับการตรวจหา Loop Closure แบบ Image-to-Image โดยใช้ SIFT Feature Detector [103] แบบ Brute Force ซึ่งจะวนซ้ำการเปรียบเทียบระหว่างคู่ภาพทุกภาพ

ข้อมูลใช้ในการทดลองจะมีสองชุด ได้แก่การเคลื่อนที่เป็น loop ขนาดใหญ่หนึ่ง loop แสดงได้ดังรูปที่ 6.9 (ก) และการเคลื่อนที่แบบซับซ้อนแสดงได้ดังรูปที่ 6.9 (ข) ซึ่งจะเกิดการ close loop 2 ตำแหน่ง โดยตำแหน่งแรกจะเป็นการ close loop แบบวิ่งตามเส้นทางเดิม ส่วนตำแหน่งที่สองจะเป็นการ close loop แบบวิ่งย้อนทางเดิม



(ก)

(ข)

รูปที่ 6.9 ภาพเส้นทางการเคลื่อนที่การทดลองการ close loop

เนื่องจากอัลกอริทึมการตรวจหา loop closure ที่ได้นำเสนอซึ่งเป็นแบบ Map-to-Map นั้นไม่สามารถทำงานได้เองเพียงลำพัง ต้องทำงานร่วมกับ SLAM ดังนั้นในการวัดผลเราจะประยุกต์อัลกอริทึม SLAM ที่ได้นำเสนอให้สามารถวิ่งทับซ้อนกับเส้นทางเดิม โดยไม่ต้องแตกแขนงต้นไม้ความสัมพันธ์ใหม่ (ปรับแก้ตำแหน่งในต้นไม้เมื่อมีการ close loop เพียงอย่างเดียว) เพื่อที่จะได้เปรียบเทียบจำนวนครั้งในการตรวจพบ loop closure

ผลลัพธ์ในการตรวจหา loop closure เปรียบเทียบระหว่างสามอัลกอริทึม สำหรับชุดข้อมูลที่ 1 และ 2 แสดงได้ดังรูปที่ 6.10 (ก) และรูปที่ 6.10 (ข) ตามลำดับ ในแถวแรกจะเป็นผลลัพธ์ของอัลกอริทึม SIFT Brute Force ในแถวที่สองเป็นผลลัพธ์ของอัลกอริทึม FABMAP และแถวที่สามแสดงผลลัพธ์ของอัลกอริทึมที่ได้นำเสนอ โดยตำแหน่งสีแดงนั้นจะเป็นตำแหน่งที่เกิด loop closure ส่วนเส้นสีแดงจะเชื่อมระหว่าง view สอง view ที่มีความสัมพันธ์กัน สำหรับอัลกอริทึมที่นำเสนอนั้น เนื่องจากเป็นแบบ Map-to-Map ดังนั้นจึงไม่สามารถบอกตำแหน่ง view ที่เกิด loop closure ได้โดยตรง จึงจะใช้แทนด้วย view ล่าสุด และ view ใกล้สุดใน map ที่มีความสัมพันธ์กันแทน

(ก) (ข)

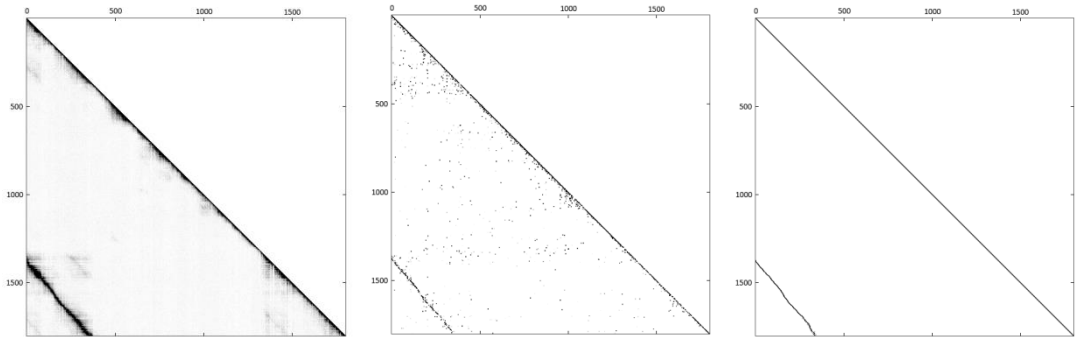
รูปที่ 6.10 ผลลัพธ์ในการตรวจหา loop closure เปรียบเทียบระหว่างสามอัลกอริทึมได้แก่ SIFT Brute Force, FABMAP และอัลกอริทึมที่ได้นำเสนอตามลำดับ สำหรับชุดข้อมูลที่ 1(ก) และชุดข้อมูลที่ 2 (ข)

จากในรูปที่ 6.10 (ก) จะเห็นได้ว่าอัลกอริทึมทั้งสามสามารถตรวจหา loop closure ได้ครบ เนื่องจากเป็นการเคลื่อนที่แบบวิ่งซ้ำเส้นทางเดิมทำให้อัลกอริทึมแบบ Image-to-Image ทำงานได้ แต่ในรูปที่ 6.10 (ข) จะเห็นได้ว่า ในส่วนเส้นทางที่เป็นการวิ่งย้อนเส้นทางเดิมนั้น อัลกอริทึม SIFT Brute Force และ FABMAP ไม่สามารถตรวจหา loop closure ได้ ในขณะที่อัลกอริทึมที่ได้นำเสนอ สามารถตรวจหาได้อย่างถูกต้อง

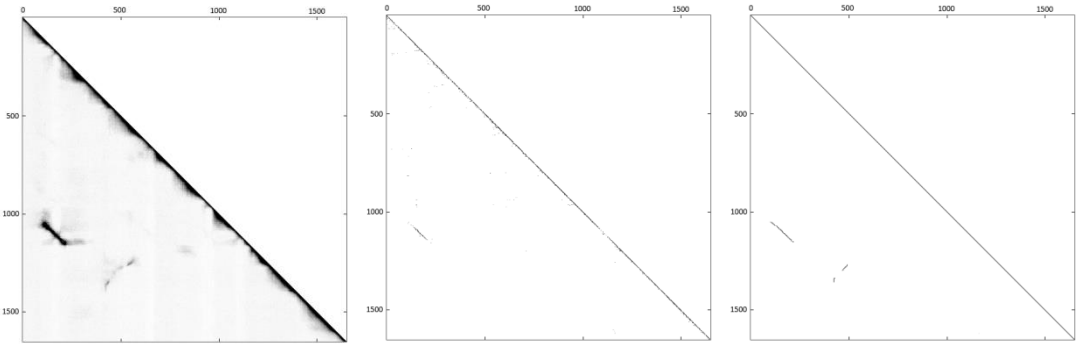
นอกจากนี้การตรวจหา loop closure ของอัลกอริทึม FABMAP ยังเกิด False Positive เป็นจำนวนมากเนื่องจากอัลกอริทึมไม่ได้พิจารณาข้อมูลแผนที่ในการทวนสอบ

ภาพ correspondence matrix แสดงได้ดังรูปที่ 6.11 และรูปที่ 6.12 สำหรับชุดข้อมูลที่ 1 และชุดข้อมูลที่ 2 ตามลำดับ โดยตารางจะใช้ในการบอกความสัมพันธ์แบบพบกันหมดของ view โดยในแต่ละช่องจะบอกความน่าจะเป็นที่จะเกิด loop closure ระหว่าง view ในแกน x และ view ในแกน y โดยความเข้มของจุดสีดำจะแสดงโอกาสที่เกิดการ close loop จะเห็นได้ว่า ในชุดข้อมูลที่ 2 นั้น อัลกอริทึม SIFT Brute Force และ

FABMAP ตรวจพบ loop closure เพียงบริเวณเดียว ส่วนอัลกอริทึมที่ได้นำเสนอ สามารถตรวจหาได้ทั้งสองบริเวณ



รูปที่ 6.11 correspondence matrix แสดงผลลพธ์การตรวจพบ loop closure ของอัลกอริทึม SIFT Brute Force, FABMAP และอัลกอริทึมที่ได้นำเสนอตามลำดับ สำหรับชุดข้อมูลที่ 1



รูปที่ 6.12 correspondence matrix แสดงผลลพธ์การตรวจพบ loop closure ของอัลกอริทึม SIFT Brute Force, FABMAP และอัลกอริทึมที่ได้นำเสนอตามลำดับ สำหรับชุดข้อมูลที่ 2

ตาราง 6.1แสดงเปอร์เซนต์ความถูกต้องในการตรวจหา loop closure และเปอร์เซนต์ในการเกิด False Positive โดยข้อมูล Ground Truth ที่ใช้ในการเปรียบเทียบจะใช้มนุษย์ในการพิจารณา จากตารางจะเห็นได้ว่า อัลกอริทึมที่ได้นำเสนอจะมีเปอร์เซนต์ในการตรวจพบ loop closure มากสุด และมี False Positive ต่ำสุด ส่วน SIFT Brute Force มีความสามารถในการตรวจหาได้ดีรองลงมา และ FABMAP มีโอกาสตรวจพบ loop closure ต่ำสุด และมีค่า False Positive มากสุด เนื่องจากภาพในการสิ่งแวดล้อมมีลักษณะคล้ายคลึงกัน จึงอาจสร้างความสับสนให้ FABMAP

อย่างไรก็ดีสำหรับเวลาในการทำงานนั้น อัลกอริทึม SIFT Brute Force จะใช้เวลามากสุดโดยเวลาจะขึ้นกับขนาดแผนที่กำลังสอง สำหรับอัลกอริทึม FABMAP นั้นทำงานเร็วสุด ส่วนอัลกอริทึมที่ได้นำเสนอนั้นมีความเร็วปานกลาง แต่เนื่องจากความเร็วในการทำงานไม่ขึ้นกับขนาดแผนที่ ทำให้สามารถใช้กับแผนที่ขนาดใหญ่ได้

ตาราง 6.1 ตารางแสดงเปอร์เซนต์ความถูกต้องในการตรวจหา loop closure และเปอร์เซนต์ในการเกิด False Positive

DataSet	SIFT Brute Force		FABMAP 2.0		Our Algorithm	
	Recall Rate	False Positive	Recall Rate	False Positive	Recall Rate	False Positive
DataSet 1	99.772 %	0.0879 %	34.396 %	8.571 %	97.039 %	0.000 %
DataSet 2	52.995 %	1.88153 %	19.355 %	0.5575%	69.585 %	0.209 %

6.4.1 ผลการทดลองการ Relocalization

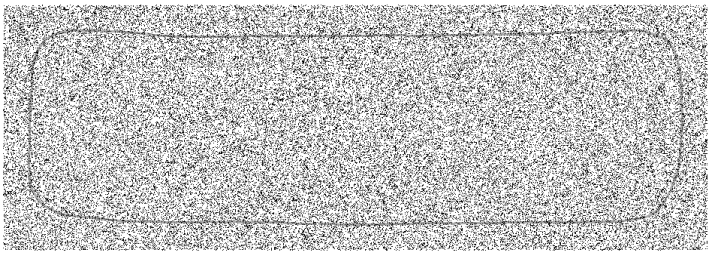
กระบวนการ relocalization เป็นกระบวนการที่หุ่นยนต์ทำการตรวจหาตำแหน่งของตัวเองซ้ำ ในแผนที่ที่กำหนดไว้แล้ว ซึ่งกระบวนการ relocalization จะถูกใช้ในกรณีที่หุ่นยนต์ถูกลักพาตัว หรือในกรณีที่หุ่นยนต์เริ่มทำงานใหม่ในแผนที่ที่กำหนดให้ เนื่องจากหุ่นยนต์ไม่รู้ตำแหน่งเริ่มต้นของตำแหน่งตนเองในแผนที่ กระบวนการ relocalization จะต้องทำการเปรียบข้อมูลการวัดจากอุปกรณ์วัดค่ากับตำแหน่งที่เป็นไปได้ทั้งหมดในแผนที่เพื่อหาตำแหน่งเริ่มต้นที่ถูกต้อง

ในการทดลองนี้จะให้หุ่นยนต์ทำการ relocalize ตำแหน่งตนเองโดยทำการหาความสัมพันธ์ระหว่างแผนที่ที่หุ่นยนต์สร้างได้กับแผนที่ที่กำหนดให้ และจะทำการเปรียบเทียบเวลาที่ใช้ในการทำงานระหว่างวิธีการเลือก Candidate Submap ตามอัลกอริทึมที่ได้นำเสนอ กับวิธีการเลือก Candidate Submap แบบสุ่ม

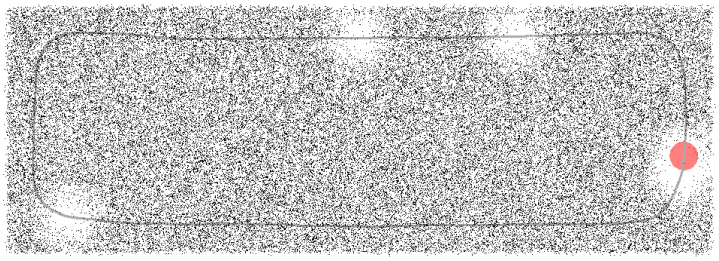
ในขั้นตอนเริ่มต้นนั้น เนื่องจากหุ่นยนต์ไม่รู้ตำแหน่งตนเองในแผนที่ที่กำหนดให้ จึงกำหนดให้แผนที่ความน่าจะเป็นเริ่มต้นมีการกระจายแบบ uniform โดย กลุ่ม particle ที่ใช้แทนการกระจายความน่าจะเป็นนั้นแสดงได้ดังรูปที่ 6.14 (ก) หลังจากนั้น เมื่ออัลกอริทึมเริ่มทำการค้นหา Candidate Submap แผนที่ความน่าจะเป็นจะถูกปรับแก้ไปตามโอกาสในการจะค้นเจอ Submap ที่ถูกต้อง ภาพแผนที่ความน่าจะเป็นการตรวจพบ loop closure ระหว่างการทำงานแสดงได้ดังรูปที่ 6.14 (ข) และ (ค) และท้ายสุดอัลกอริทึม ก็สามารถตรวจพบ Submap ที่ถูกต้องดังรูปที่ 6.14 (ง)

ภาพผลลัพธ์ของ view สอง view ที่ค้นเจอความสัมพันธ์กันในกระบวนการ relocalization แสดงได้ดังรูปที่ 6.13

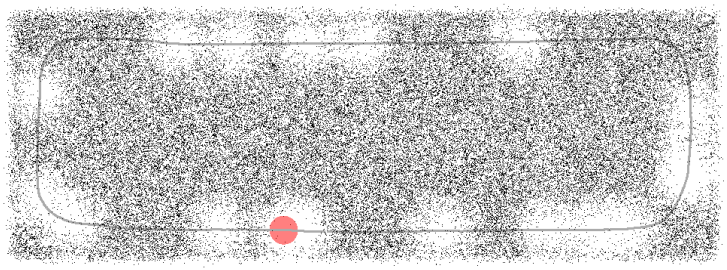
รูปที่ 6.13 ภาพคู่ความสัมพันธ์ของ view ที่เกิด loop closure



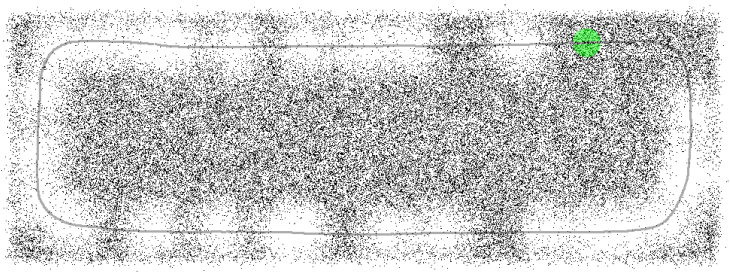
(ก)



(ข)



(ค)



(ง)

รูปที่ 6.14 ภาพแสดงการทำงานของภาพแผนที่ความน่าจะเป็นของการตรวจพบ loop closure

การวัดประสิทธิภาพการทำงาน จะทำการทดลองกระบวนการ relocalization จำนวน 10 ตำแหน่ง และในแต่ละตำแหน่งจะทดลองซ้ำ 20 ครั้งเพื่อวัดเวลาเฉลี่ยในการค้นพบ Submap จะได้ผลการทดลองดังตาราง 6.2 โดยจากตาราง เวลาในการทำงานของอัลกอริทึมที่นำเสนอจะอยู่ที่ 3 - 5 วินาที ทั้งนี้จะขึ้นอยู่กับขนาดของแผนที่ ส่วนวิธีค้นหาแบบสุ่มจะใช้เวลาในการทำงานประมาณ 8 - 12 วินาที จะเห็นได้ว่ามีความแตกต่างของเวลาอย่างมีนัยสำคัญ

ตาราง 6.2 ตารางแสดงเวลาเฉลี่ยในการทำงานของกระบวนการ relocalization

DataSet	Time used (second)	
	Probabilistic selection	Random selection
DataSet 1	5.6714	12.3916
DataSet 2	3.2185	8.2905

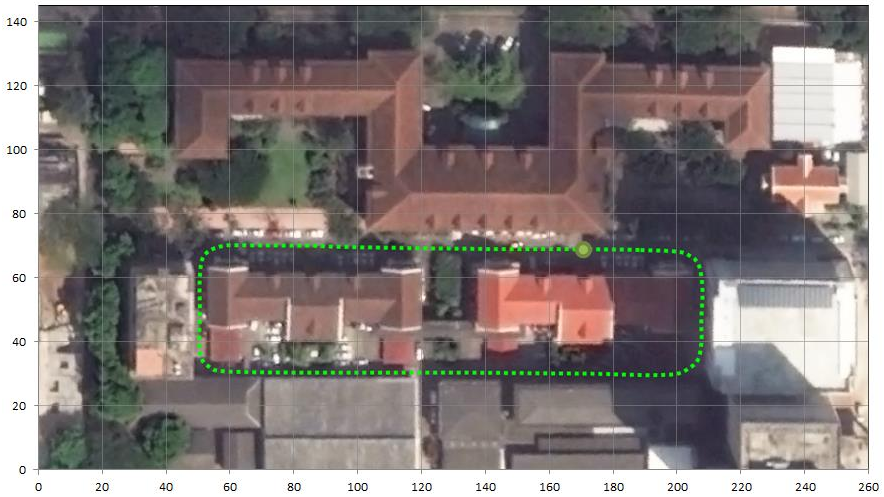
บทที่ 7
การทดลองอัลกอริทึมในสิ่งแวดล้อมจริง

ในหัวข้อนี้จะแสดงการทำงานโดยรวมของวิธีการระบุตำแหน่งพร้อมกับการสร้างแผนที่ในสิ่งแวดล้อมขนาดใหญ่ที่ได้นำเสนอ อันประกอบด้วยขั้นตอนการตรวจหาจุดสังเกตในภาพพร้อมการประมาณการเคลื่อนที่, การตรวจหา loop closure และการปรับแก้แผนที่หลัง close loop โดยในการทดสอบนั้นจะใช้มนุษย์ในการถือกล้องวิดีโอเคลื่อนที่ไปในสิ่งแวดล้อมขนาดใหญ่จากนั้นจึงเก็บภาพมาประมวลผล

สิ่งแวลดล้อมที่ใช้ในการทดสอบจะเป็นสภาพแวดล้อมภายในจุฬาลงกรณ์มหาวิทยาลัย ซึ่งการทดลองจะมีลักษณะหลายรูปแบบได้แก่ การทดลองการ close loop บริเวณรอบคณะวิศวกรรมศาสตร์, การทดลองการเคลื่อนที่แบบซับซ้อนบริเวณรอบคณะวิทยาศาสตร์, การทดลองการเคลื่อนกล้องแนวขนานกับการเคลื่อนที่เพื่อ reconstruct สิ่งปลูกสร้างบริเวณศูนย์วิทยทรัพยากร, การทดลองการสร้างแผนที่ขนาดใหญ่ภายในบริเวณจุฬาลงกรณ์มหาวิทยาลัยฝั่งตะวันตกของถนนพญาไท และ การทดลองการสร้างแผนที่ภาพถ่ายทางอากาศด้วย UAV

7.1 การทดลองการ close loop บริเวณรอบคณะวิศวกรรมศาสตร์

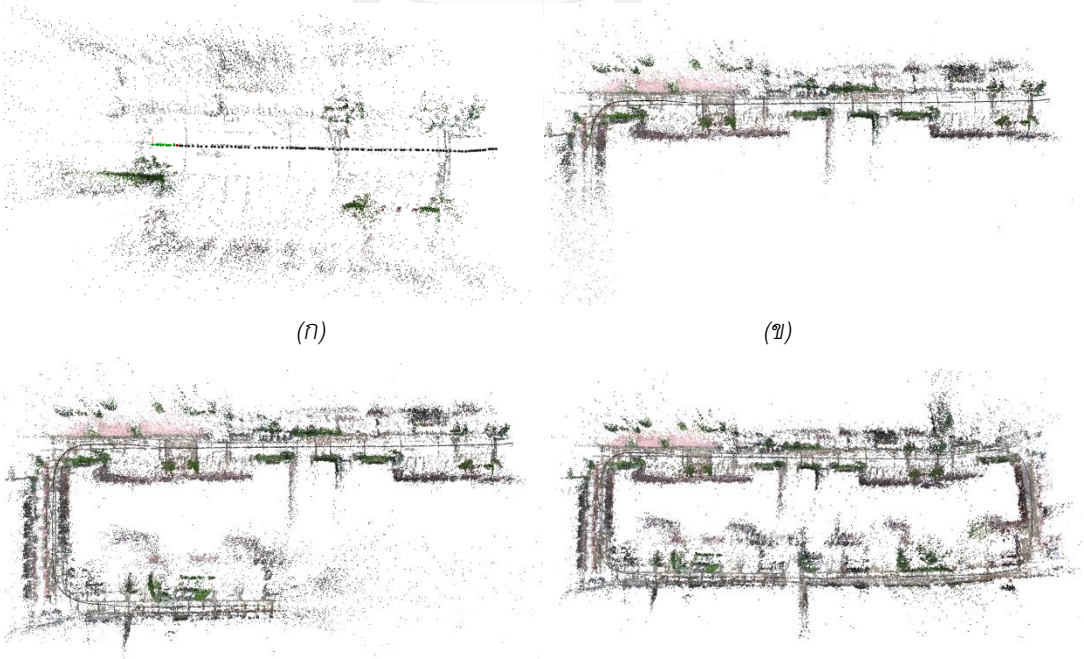
การทดลองนี้จะเคลื่อนที่กล้องวิดีโอรอบคณะวิศวกรรมศาสตร์หนึ่งรอบโดยวนกลับมายังจุดเริ่มต้น เป็นระยะทางประมาณ 330 เมตร เพื่อทดลองการ close loop ของอัลกอริทึมการระบุตำแหน่งพร้อมกับการสร้างแผนที่ที่ได้นำเสนอ ลักษณะสิ่งแวดล้อมจะเป็นถนนรอบคณะ ทั้งสองข้างทางประกอบไปด้วยตึกอาคารเรียนต่าง ๆ โดยการเคลื่อนที่จะมีลักษณะเป็นสี่เหลี่ยมผืนผ้าดังแสดงได้ดังรูปที่ 7.1 ซึ่งจะเหมาะสำหรับการวัดผลความแม่นยำของอัลกอริทึม SLAM ตัวอย่างภาพที่ใช้ในการทดลองแสดงได้ดังรูปที่ 7.2 โดยในการทดลองนี้จะใช้ลำดับภาพจำนวนทั้งสิ้น 1377 ภาพ



รูปที่ 7.1 ภาพเส้นทางการเคลื่อนที่การทดลองรอบคณะวิศวกรรมศาสตร์

รูปที่ 7.2 ตัวอย่างภาพที่ใช้ในการทดลองรอบคณะวิศวกรรมศาสตร์

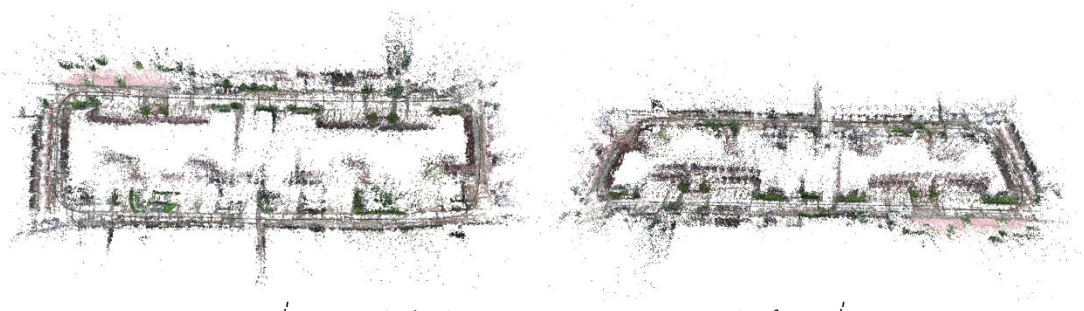
ผลการประมาณการเคลื่อนที่จากข้อมูลภาพก่อนการ close loop แสดงได้ดังรูปที่ 7.3 โดยจากภาพจุดทรงกลมแสดงเส้นทางการเคลื่อนทีของกล้อง จุดทรงกลมสีเขียวและสีแดงแสดงตำแหน่ง view ที่ถูกใช้ในการทำ Sliding-Window Bundle Adjustment โดยจุดสีแดงจะเป็น view ที่ถูก fix ตำแหน่งไว้ เมื่อกล้องเคลื่อนที่ไปเป็นระยะทางไกลขึ้นเรื่อย ๆ ตำแหน่งของกล้องที่ประมาณได้ย่อมมีความคลาดเคลื่อนมากขึ้น ดังแสดงในรูปที่ 7.3 (ค)(ง) จะเห็นได้ว่าตำแหน่งของกล้องคลาดเคลื่อนไปจากความเป็นจริงพอสมควรอันเนื่องมาจากปัญหา pose drift หลังนั้นเมื่อกล้องเคลื่อนที่กลับมายังตำแหน่งเริ่มต้น อัลกอริทึมสำหรับตรวจหา loop closure สามารถตรวจหา loop ได้จึงทำการ close loop และปรับแก้แผนที่ จะได้ผลลัพธ์หลังการปรับแก้ ดังแสดงในรูปที่ 7.4 โดยตำแหน่งของกล้องและแผนที่ จะมีควาสดคล้องกับตำแหน่งและแผนที่ ณ จุดเริ่มต้นมากขึ้น



(ก) *(ข)*

(ค) *(ง)*

รูปที่ 7.3 ผลลัพธ์การประมาณการเคลื่อนที่จากข้อมูลภาพก่อนการ close loop

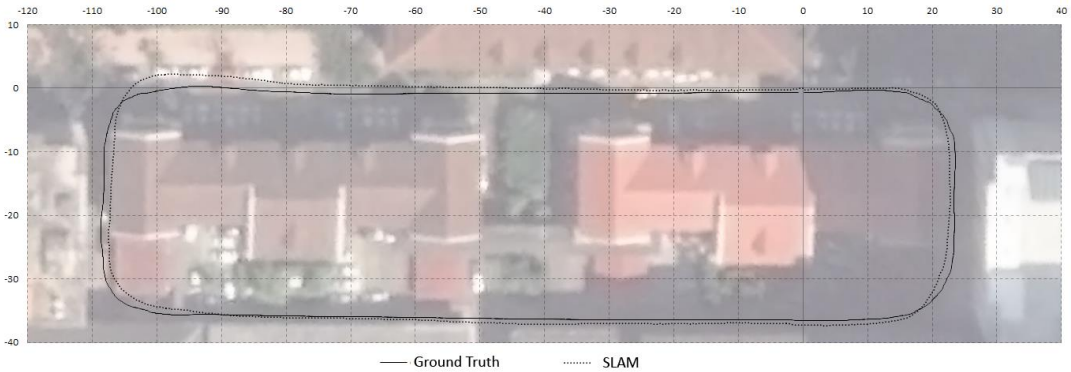


รูปที่ 7.4 ผลลัพธ์หลังการ close loop และการปรับแก้แผนที่

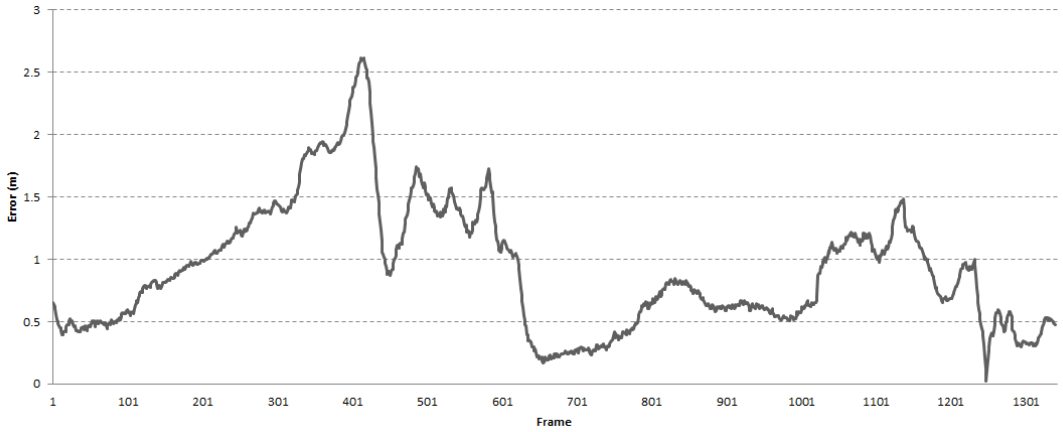
ในระหว่างที่กล้องเคลื่อนที่ไปเรื่อย ระบบจะทำการประมาณการเคลื่อนที่จากข้อมูลภาพและเพิ่มโหนด ซึ่งใช้สำหรับอธิบายตำแหน่งของ view เข้าไปในต้นไม้ และเมื่อโหนดในแขนงของต้นไม้มีจำนวนมากขึ้นเกินค่าคงที่ค่าหนึ่ง (ในที่นี้กำหนดไว้ 100 โหนด) กระบวนการ Pose Marginalization จะทำการลดโหนดไม่ให้เกินค่าคงที่ที่กำหนดไว้ ผลลัพธ์ของกระบวนการ Pose Marginalization แสดงได้ดังรูปที่ 7.5 โดยจะเห็นได้ว่า แม้จำนวน view ในแผนที่จะมีมากถึงพันกว่า view แต่จำนวนโหนดจะถูกจำกัดไว้ไม่ให้เกิน 100 โหนด

รูปที่ 7.5 ผลลัพธ์ของกระบวนการ Pose Marginalization

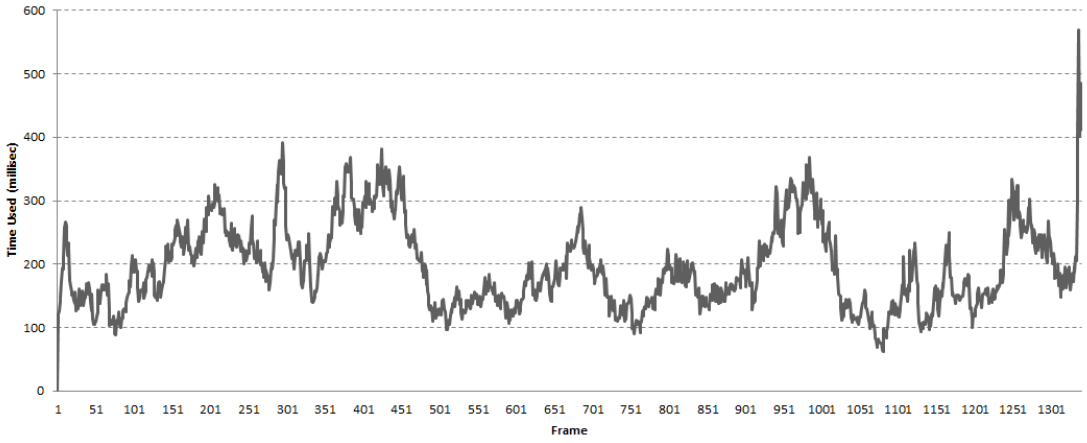
ในการทดลองนี้เราจะวัดความแม่นยำของการสร้างแผนที่ โดยการเปรียบเทียบเส้นทางการเคลื่อนของกล้องที่ประมาณได้จากอัลกอริทึม กับเส้นทางการเคลื่อนที่ซึ่งใช้มนุษย์ในการระบุโดยอ้างอิงแผนที่จาก Google Earth และเนื่องจากว่าแผนที่จาก SLAM นั้นไม่มีข้อมูล Scale ดังนั้นในการเปรียบเทียบ เราจะขยายขนาดแผนที่จาก SLAM ให้มีขนาดสอดคล้องกับแผนที่อ้างอิงมากสุด ความคลาดเคลื่อนของการระบุตำแหน่งแสดงได้ดังรูปที่ 7.6 โดยสามารถแสดงกราฟความคลาดเคลื่อนของตำแหน่งในแต่ละเฟรม ได้ดังรูปที่ 7.7 ซึ่งโดยเฉลี่ยแล้วความคลาดเคลื่อนในแต่ละเฟรม จะอยู่ประมาณ 0.92 เมตร



รูปที่ 7.6 กราฟเปรียบเทียบตำแหน่งของกล้องที่ประมาณได้จาก SLAM เทียบกับ Ground Truth



รูปที่ 7.7 กราฟความคลาดเคลื่อนตำแหน่งของกล้องที่ประมาณได้จาก SLAM เทียบกับ Ground Truth



รูปที่ 7.8 กราฟแสดงเวลาในการทำงานในแต่ละเฟรม

ตาราง 7.1 ตารางแสดงเวลาการทำงานในแต่ละขั้นตอนของ SLAM

Process	Time Used (millisec)
SIFT Feature Detector (GPU)	118.54
Feature Matching	124.30
Sliding-Window Bundle Adjustment	65.72
Pose Marginalization	0.78
Loop Closure Detection	235.68
Map Optimization	15.68

เวลาในการทำงานโดยเฉลี่ยต่อหนึ่งเฟรม แสดงได้ดังตาราง 7.1 โดยเวลารวมในทุกขั้นตอนต่อเฟรมจะอยู่ที่ 560.70 มิลลิวินาที อย่างไรก็ดีในการทำงานบางขั้นตอน เช่นขั้นตอน Pose Marginalization, Loop Closure Detection และ Map Optimization จะทำงานเพียงบางเฟรมเท่านั้น นอกจากนี้ขั้นตอนการตรวจหา SIFT Features จะเป็นการทำงานบน GPU ซึ่งสามารถแยกการทำงานได้เป็นอีก thread หนึ่งทำให้เวลาในการ

ทำงานที่แท้จริงจะคิดจากเวลามากสุดที่ใช้ในแต่ละ thread เท่านั้น ซึ่งโดยเฉลี่ยเวลาในการทำงานต่อเฟรม จะประมาณ 194.768 มิลลิวินาที โดยกราฟเวลาในการทำงานในแต่ละเฟรม แสดงได้ดังรูปที่ 7.8

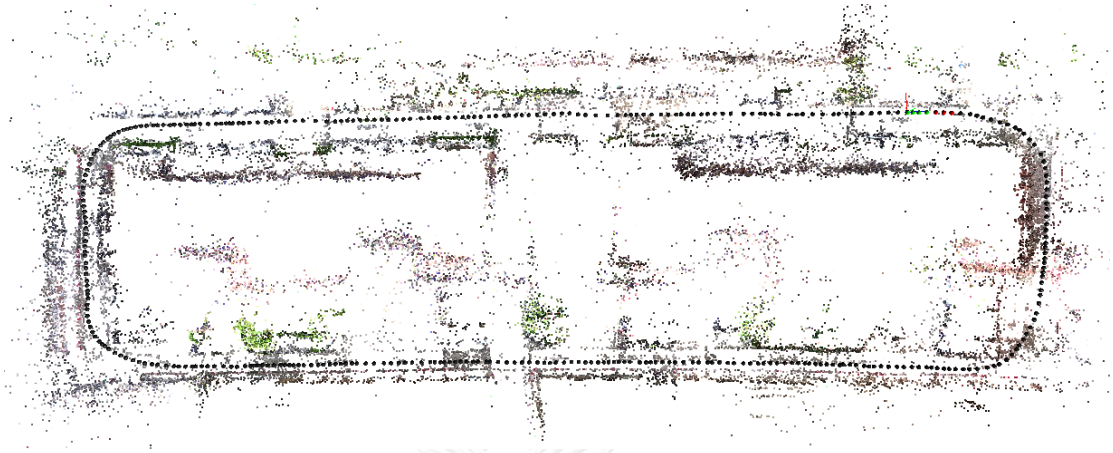
จากกราฟเวลาในการทำงานจะเห็นได้ว่าเวลาในการทำงานต่อในแต่ละเฟรม ค่อนข้างคงที่ ไม่ขึ้นกับขนาดของแผนที่ โดยเฟรมที่ใช้เวลาในการทำงานมากสุดจะเป็นเฟรมท้ายสุดซึ่งมีการ close loop แสดงให้เห็นว่าอัลกอริทึมที่ได้นำเสนอ สามารถรับมือกับแผนที่ขนาดใหญ่ได้อย่างมีประสิทธิภาพ

ในการทดลองการระบุตำแหน่งและสร้างแผนที่ด้วยกล้องวิดีโอมุมกว้าง เราได้เก็บภาพวิดีโอจากการเดินการถือกล้องวิดีโอเคลื่อนที่ไปในสิ่งแวดล้อม จากนั้นจึงนำข้อมูลมาประมวลผลในภายหลัง ทั้งนี้เพื่อความสะดวกในการทดลอง อย่างไรก็ดีอัลกอริทึมที่ได้นำเสนอมีการทำงานแบบ incremental นั่นคือทำการประมวลผลในแต่ละเฟรมที่มีข้อมูลรับเข้าและให้ผลลัพธ์เป็นค่าประมาณแผนที่และตำแหน่งของกล้องแบบทันที โดยเวลาที่ใช้โดยเฉลี่ยต่อเฟรมจะอยู่ที่ประมาณเฟรมละ 194.768 มิลลิวินาที เมื่อรวมเวลาในการประมวลผลทุกเฟรม (1377 เฟรม) แล้วเวลาในการทำงานรวมทั้งหมดจะประมาณ 268 วินาที และเมื่อเทียบกับเวลาที่มนุษย์ใช้ในการเดินซึ่งมีความเร็วประมาณ 5 กิโลเมตรต่อชั่วโมง เมื่อเดินเป็นระยะทาง 330 เมตรจะใช้เวลาประมาณ 240 วินาที จะเห็นได้ว่าเวลาที่ใช้ในการเคลื่อนที่จะใกล้เคียงกับเวลาที่ใช้ในการทำงาน ดังนั้นเมื่อสมมุติให้มีหุ่นยนต์เคลื่อนที่ไปในสิ่งแวดล้อมจริงและทำการประมวลผลทันทีที่ได้รับข้อมูลภาพ โดยให้เป็นการทำงานแบบต่อเนื่อง เมื่อหุ่นยนต์ประมวลผลภาพก่อนหน้าเสร็จก็นำข้อมูลภาพล่าสุดมาประมวลผลต่อทันที หุ่นยนต์ก็จะสามารถสร้างแผนที่และระบุตำแหน่งของตนเองในแผนที่แบบทันการณ์ได้

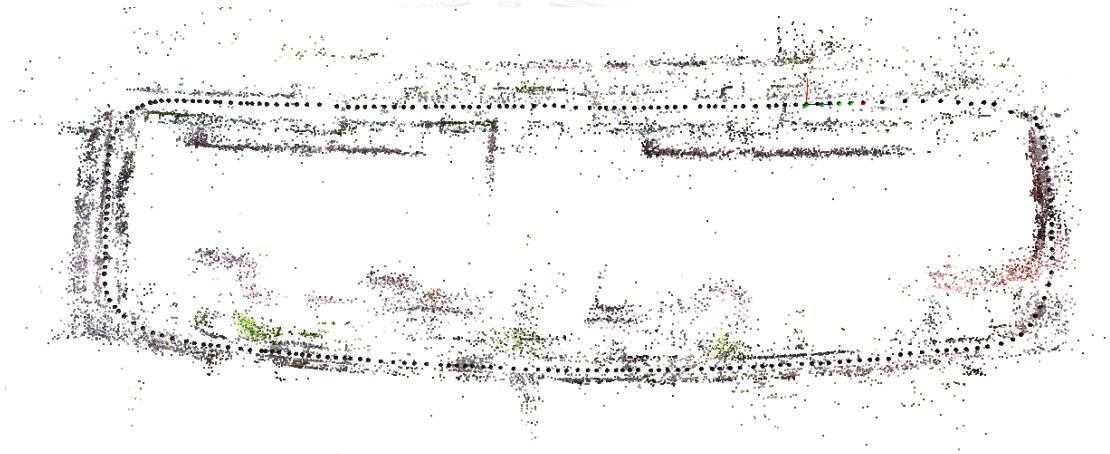
โดยปรกติแล้วกล้องวิดีโอจะมีความเร็วในการรับภาพประมาณ 30 ภาพต่อวินาที ซึ่งจะทำให้มีความต่างเวลาระหว่างภาพประมาณ 33.33 มิลลิวินาที ซึ่งไม่เพียงพอในการทำงานของอัลกอริทึมหนึ่งเฟรม อย่างไรก็ดี ในการทำงานจริงนั้นไม่ได้ประมวลผลภาพทุกเฟรม แต่ละเลือกประมวลผลเฉพาะเฟรมที่มีความต่างระหว่างภาพพอสมควรจึงทำให้สามารถทำงานแบบทันการณ์ได้

การทดลองในรูปที่ 7.9 (ก) และ (ข) เป็นการทดลองการระบุตำแหน่งและสร้างแผนที่บริเวณรอบคณะวิศวกรรมศาสตร์ โดยจะลดจำนวนเฟรมลง 1/4 เท่า และ 1/6 เท่าตามลำดับ ทำให้เหลือจำนวนภาพที่ใช้ทั้งสิ้น 344 ภาพ และ 229 ภาพ จากผลการทดลองจะเห็นว่าอัลกอริทึมยังสามารถทำงานได้โดยสามารถประมาณการเคลื่อนที่เป็นวงรอบและทำการ close loop ได้สำเร็จ แต่จะเห็นได้ว่าจำนวนจุดสังเกตในแผนที่จะมีจำนวนลดลงอย่างเห็นได้ชัด และจากการทดลองเพิ่มเติมโดยการลดจำนวนเฟรมลง 1/8 เท่า และ 1/10 เท่า พบว่าอัลกอริทึมไม่สามารถทำงานได้ โดยขั้นตอนการประมาณการเคลื่อนที่ของกล้องมีความคลาดเคลื่อนมากเกินไป ดังนั้นจึงสรุปได้ว่าสำหรับสิ่งแวดล้อมรอบคณะวิศวกรรมศาสตร์ ระยะห่างระหว่างเฟรมมากสุดที่อัลกอริทึมยังสามาถทำงานได้อย่างมีประสิทธิภาพจะอยู่ที่ประมาณ 1.44 เมตร และเมื่อวัดเวลาในการทำงานเฉลี่ยต่อเฟรม จะได้ประมาณ 138.47 มิลลิวินาที และ 120.84 มิลลิวินาที สำหรับการลดจำนวนเฟรม 1/4 เท่า และ 1/6 เท่าตามลำดับ ทั้งนี้จะเห็นได้ว่าเวลาในการทำงานเร็วขึ้นกว่าแบบปรกติ เนื่องจากระยะห่างระหว่างเฟรมมากขึ้นจึงทำให้จำนวนเฟรมใน Sliding Window สำหรับคำนวนหา Visual Odometry น้อยลง ดังนั้นเมื่อพิจารณา

ความเร็วในการเดินของมนุษย์ซึ่งอยู่ที่ 5 กิโลเมตรต่อชั่วโมง หรือ 1.38 เมตรต่อวินาที จะเห็นได้ว่ามีเวลาในการทำงานมากพอในการระบุตำแหน่งและสร้างแผนที่แบบทันการณ์



(ก)



(ข)

รูปที่ 7.9 ผลลัพธ์การระบุตำแหน่งพร้อมกับการสร้างแผนที่รอบคณะวิศวกรรมศาสตร์หลังการลดจำนวนเฟรม 1/4 เท่า (ก) และ 1/6 (ข)

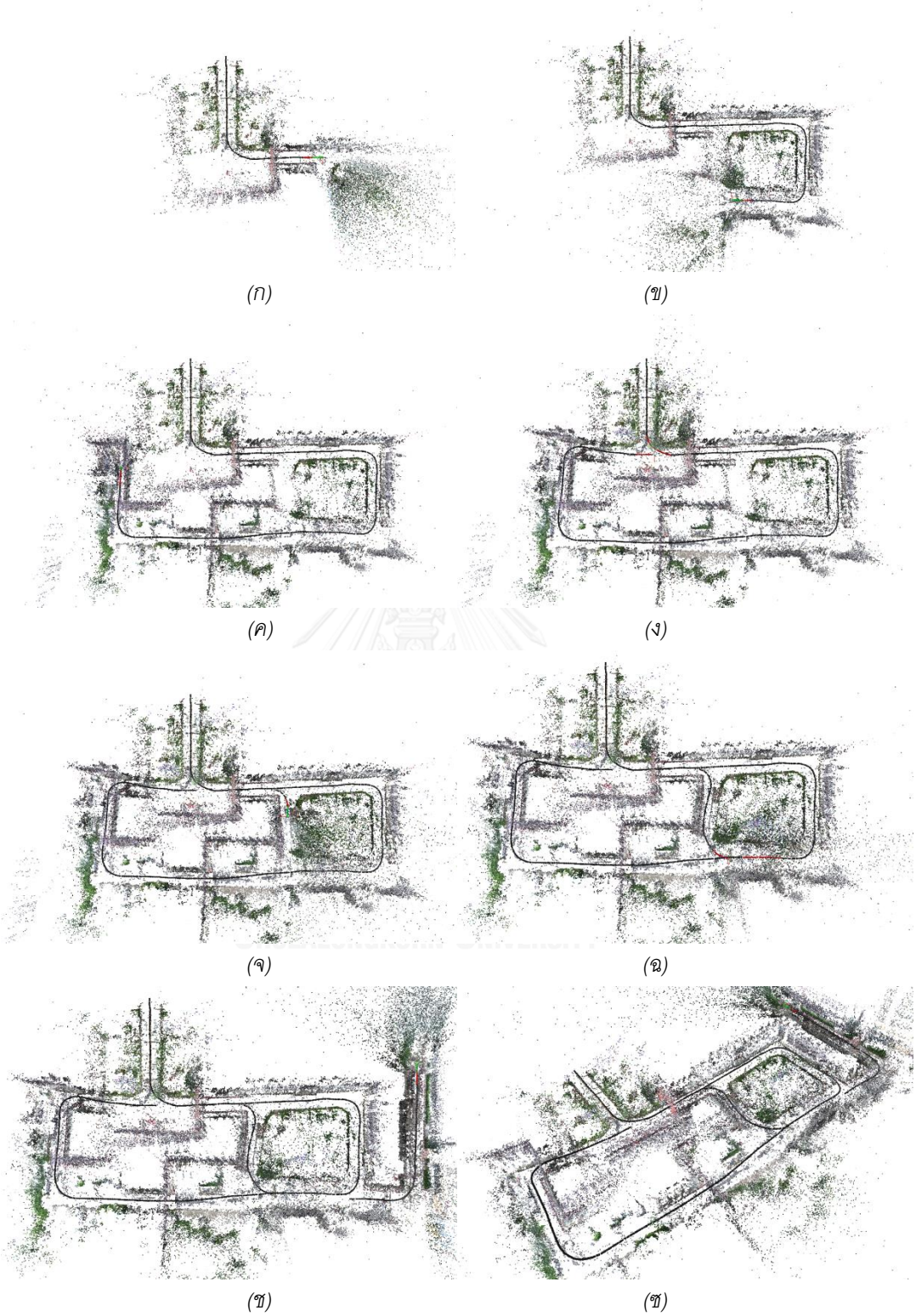
7.2 การทดลองการเคลื่อนที่แบบซับซ้อน บริเวณรอบคณะวิทยาศาสตร์

การทดลองนี้จะทดสอบอัลกอริทึมการระบุตำแหน่งพร้อมกับการสร้างแผนที่แบบซับซ้อน โดยผู้ทำการทดลองจะถือกล้องวิดีโอมุมกว้างเคลื่อนที่วนไปมาในบริเวณรอบคณะวิทยาศาสตร์แสดงได้ดังรูปที่ 7.10 ซึ่งการ close loop จะไม่ใช่การเคลื่อนที่แบบวนบรรจบหนึ่งรอบ แต่จะเป็นการวิ่งตัดกันของเส้นทางการเคลื่อนที่ หรือเป็นการวิ่งย้อนเส้นทางการเคลื่อนที่เดิม สำหรับลักษณะสิ่งแวดล้อมจะมีหลากหลายรูปแบบ ประกอบด้วยบริเวณต้นไม้จำนวนมาก, ทางลอดอุโมค์ขนาดเล็ก, ซอกตึกอาคารเรียน, พื้นที่โล่งกว้าง ตัวอย่างภาพที่ใช้ในการทดลองแสดงได้ดังรูปที่ 7.11 ซึ่งลักษณะสิ่งแวดล้อมอันหลากหลาย เหมาะสำหรับทดสอบความทนทานของอัลกอริทึมการระบุตำแหน่งและสร้างแผนที่ขนาดใหญ่ที่ได้นำเสนอ โดยการทดลองนี้จะประกอบด้วยลำดับภาพจำนวนทั้งสิ้น 1650 ภาพ 500 เมตร

รูปที่ 7.10 ภาพเส้นทางการเคลื่อนที่การทดลองรอบคณะวิทยาศาสตร์

รูปที่ 7.11 ตัวอย่างภาพที่ใช้ในการทดลองรอบคณะวิทยาศาสตร์

ผลการทดลองการระบุตำแหน่งและสร้างแผนที่แสดงได้ดังรูปที่ 7.12 โดยจุดทรงกลมแสดงเส้นทางการเคลื่อนทีของกล้อง จากรูปที่ 7.12 (ค) แสดงการระบุตำแหน่งและสร้างแผนที่ก่อนการ close loop ครั้งแรก โดยจะเห็นได้ว่าตำแหน่งของกล้องคลาดเคลื่อนจากความเป็นจริงไปมาก และเมื่อทำการ close loop พร้อมการปรับแก้แผนที่ก็จะได้แผนที่สักษณะครบรอบสมบูรณ์ตามรูปที่ 7.12 (ง) หลังจากนั้นกล้องจะเคลื่อนที่ซ้ำเส้นทางเดิม ซึ่งระบบสามารถตรวจวัดได้และจะไม่ทำการเพิ่ม view เข้าไปในแผนที่เพื่อเป็นการรักษาขนาดของแผนที่ไว้ไม่ให้โตตามเวลา จากนั้นกล้องมีการเคลื่อนที่แยกออกจากเส้นทางเดิมดังแสดงได้ดังรูปที่ 7.12 (จ) ระบบจึงทำการเพิ่ม view ใหม่เข้าในแผนที่อีกครั้ง หลังจากนั้นกล้องได้เคลื่อนที่กลับเข้าสู่เส้นทางเดิมอีกครั้ง แต่เป็นการเคลื่อนที่ย้อนทางเดิม ระบบก็สามารถตรวจวัด loop closure ได้และได้ทำการ close loop พร้อมปรับแก้แผนที่ดังแสดงได้ดังรูปที่ 7.12 (ฉ)



(ก) *(ข)*

(ค) *(ง)*

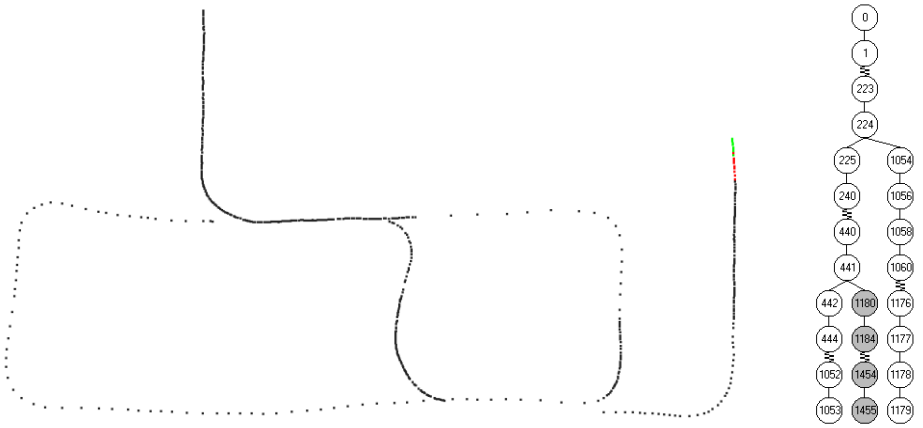
(จ) *(ฉ)*

(ช) *(ซ)*

รูปที่ 7.12 ผลลัพธ์การระบุตำแหน่งพร้อมกับการสร้างแผนที่รอบคณะวิทยาศาสตร์

จากรูปที่ 7.12 (ช)และ (ซ) จะเห็นได้ว่าในลักษณะแผนที่ที่ซับซ้อน อัลกอริทึมที่ได้นำเสนอก็สามารถสร้างแผนที่ได้อย่างถูกต้อง และระบุตำแหน่งของกล้องในแผนที่ได้อย่างแม่นยำ

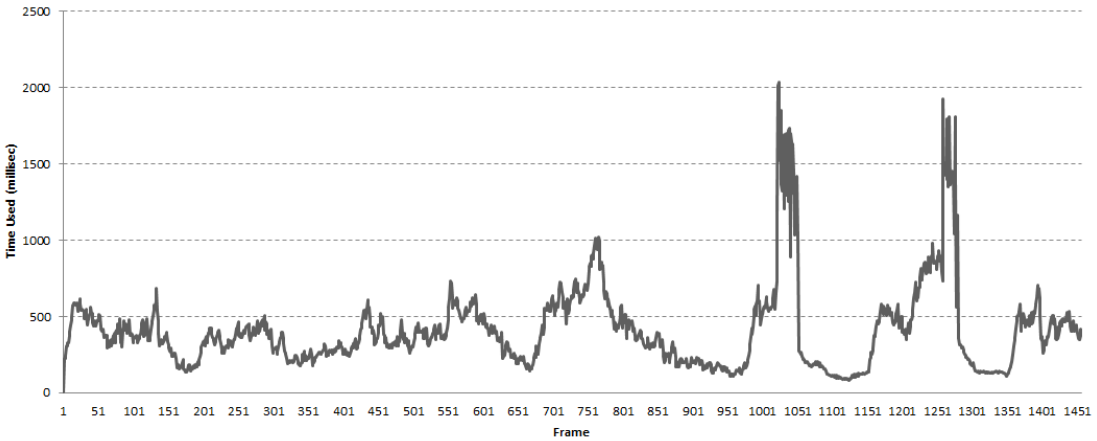
ผลลัพธ์ของการสร้างโหนด และการทำ Pose Marginalization แสดงได้ดังรูปที่ 7.13 (ก) โดยจะเห็นได้ว่ามีโหนดบางโหนดถูกลดออกไปเพื่อให้สามารถปรับแก้แผนที่ได้ทันการณ์ ซึ่งต้นไม้ความสัมพันธ์ของโหนดจะแสดงได้ดังรูปที่ 7.13 (ข)



รูปที่ 7.13 (ก) ผลลัพธ์ของกระบวนการ Pose Marginalization (ข) ต้นไม้ความความสัมพันธ์ของโหนด

เวลาในการทำงานโดยเฉลี่ยในหนึ่งเฟรม แสดงได้ดังตาราง 7.2 และกราฟเวลาในการทำงานแสดงได้ดังรูปที่ 7.14 โดยจะเห็นได้ว่าเวลาในการทำงานค่อนข้างคงที่ ถึงแม้ว่าแผนที่จะมีขนาดใหญ่ขึ้นก็ตาม แสดงให้เห็นว่าอัลกอริทึมที่ได้นำเสนอ สามารถรับมือกับแผนที่ขนาดใหญ่ได้อย่างมีประสิทธิภาพ โดยจากกราฟ SLAM จะใช้เวลาในการทำงานเฉลี่ยเฟรมละ 421.43 มิลลิวินาที สำหรับบริเวณที่แท่งกราฟมีค่าสูงสุดสองจุดจะเป็นบริเวณที่อัลกอริทึมกำลังตรวจหา loop closure

สาเหตุที่การทดลองนี้ใช้เวลาในการทำงานโดยเฉลี่ยต่อเฟรมมากกว่าการทดลองที่ 1 เนื่องจากมีการกำหนดขนาด Sliding Window ให้ใหญ่กว่าทำให้มีจำนวนเฟรมในขั้นตอน Local Bundle Adjustment มากกว่า ดังนั้นจึงใช้เวลาในการทำงานการประมาณการเคลื่อนที่จากภาพนานกว่า



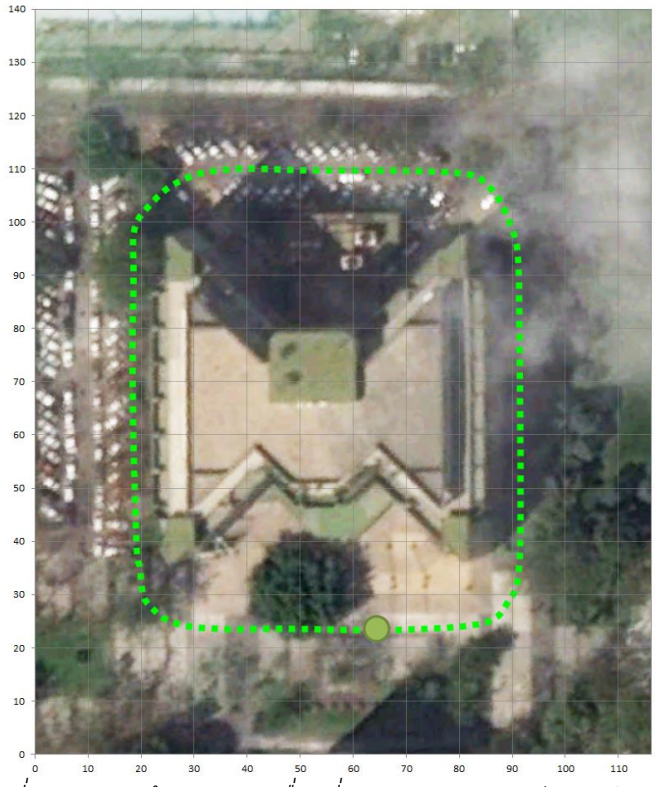
รูปที่ 7.14 กราฟแสดงเวลาในการทำงานในแต่ละเฟรม (2)

ตาราง 7.2 ตารางแสดงเวลาการทำงานในแต่ละขั้นตอนของ SLAM (2)

Process	Time Used (millisec)
SIFT Feature Detector (GPU)	123.21
Feature Matching	213.43
Sliding-Window Bundle Adjustment	191.29
Pose Marginalization	1.05
Loop Closure Detection	951.55
Map Optimization	51.58

7.3 การทดลองการสร้างโมเดลสิ่งปลูกสร้าง บริเวณศูนย์วิทยทรัพยากร

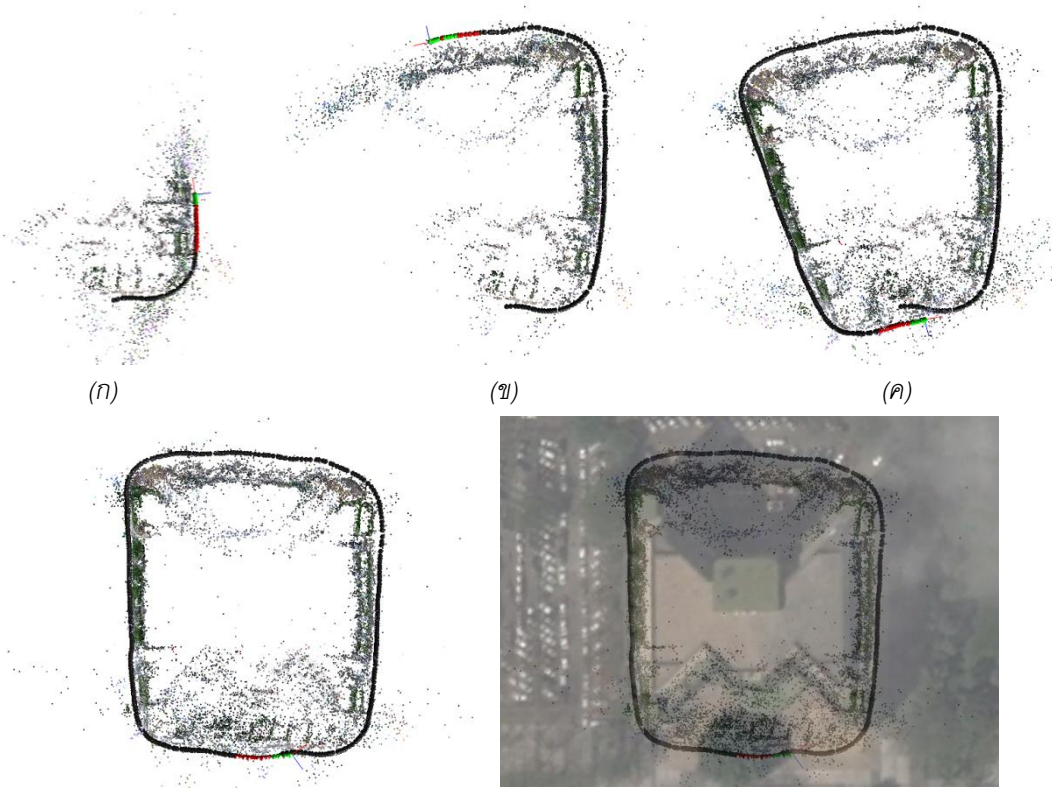
ในการทดลองที่ 7.1 และ 7.2 จะเป็นการเคลื่อนที่บริเวณรอบศูนย์วิทยทรัพยากร โดยการเคลื่อนที่ของกล้องจะเป็นการเคลื่อนที่ไปข้างหน้าเทียบกับมุมมองกล้อง ในการทดลองนี้จะนำเสนอการเคลื่อนที่ที่ต่างออกไป โดยจะถือกล้องเคลื่อนแทยงด้านข้าง ซึ่งการหันกล้องจะเป็นแนวขนานกับการเคลื่อนที่ ในการทดลองนี้เราจะทดลองเคลื่อนกล้องรอบอาคารศูนย์วิทยทรัพยากรดังรูปที่ 7.15 เพื่อจำลองโมเดลสิ่งปลูกสร้าง ภาพตัวอย่างซึ่งจะใช้ในการทดลองนี้แสดงได้ดังรูปที่ 7.16 โดยในการทดลองนี้จะใช้ลำดับภาพจำนวนทั้งสิ้น 373 ภาพ



รูปที่ 7.15 ภาพเส้นทางการเคลื่อนที่การทดลองรอบศูนย์วิทยทรัพยากร

รูปที่ 7.16 ตัวอย่างภาพที่ใช้ในการทดลองรอบศูนย์วิทยทรัพยากร

ผลการทดลองการระบุตำแหน่งและสร้างแผนที่แสดงได้ดังรูปที่ 7.17 จะเห็นได้ว่าขณะกล้องเคลื่อนที่ไป ระบบจะสร้างแผนที่จากข้อมูลภาพที่ได้รับเพิ่มมาแบบ incremental เมื่อกล้องเคลื่อนที่ได้ระยะหนึ่งแผนที่จะเริ่มเบี่ยงเบนดังรูปที่ 7.17 (ค) อันเนื่องมาจากความคลาดเคลื่อนในการประมาณการเคลื่อนที่ หลังจากนั้นเมื่อกล้องเคลื่อนที่กลับมายังตำแหน่งเดิมระบบจะทำการ close loop และปรับแก้แผนที่ จะได้แผนที่ที่ถูกต้องดังรูปที่ 7.17 (ง)

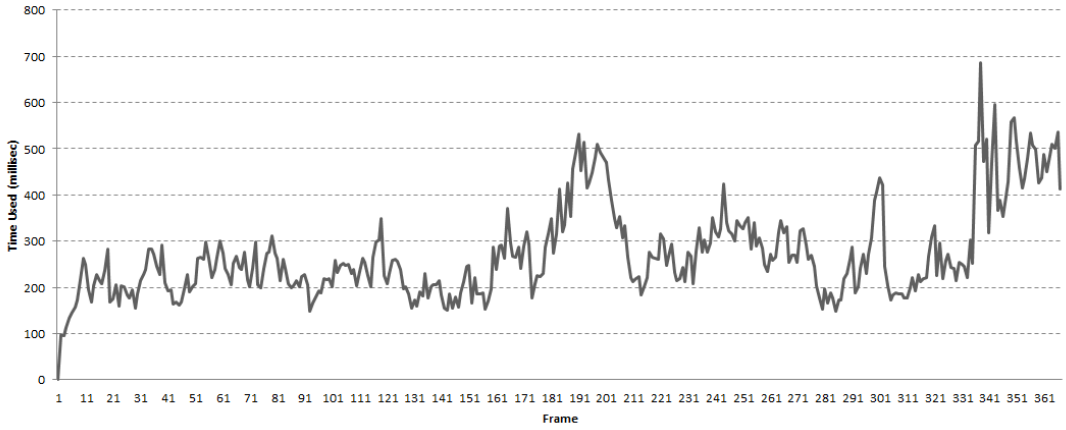


(ก) *(ข)* *(ค)*

(ง) *(จ)*

รูปที่ 7.17 ผลลัพธ์การระบุตำแหน่งพร้อมกับการสร้างแผนที่รอบศูนย์วิทยทรัพยากร

เวลาในการทำงานโดยเฉลี่ยในหนึ่งเฟรม แสดงได้ดัง*ตาราง 7.3* และกราฟเวลาในการทำงานแสดงได้ดังรูปที่ 7.18 โดยจะเห็นได้ว่าเวลาในการทำงานต่อเฟรมค่อนข้างคงที่ เวลาเฉลี่ยประมาณเฟรมละ 271.55 มิลลิวินาที



รูปที่ 7.18 กราฟแสดงเวลาในการทำงานในแต่ละเฟรม (3)

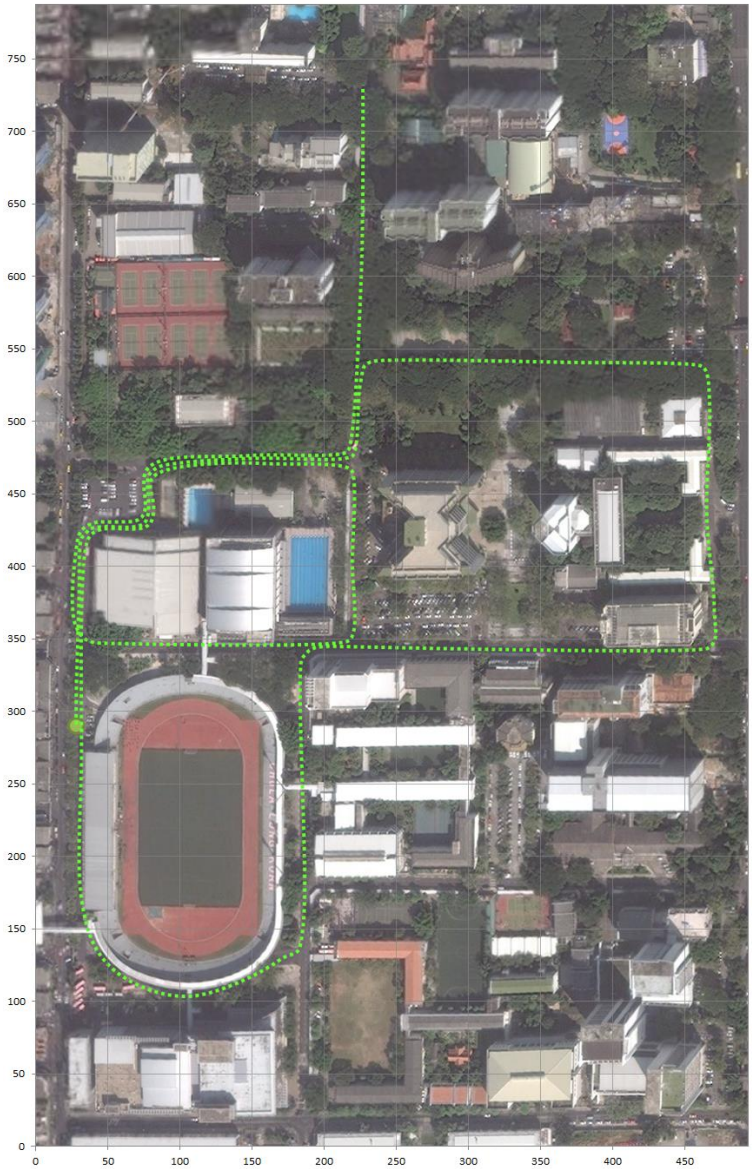
ตาราง 7.3 ตารางแสดงเวลาการทำงานในแต่ละขั้นตอนของ SLAM (3)

Process	Time Used (millisec)
SIFT Feature Detector (GPU)	120.44
Feature Matching	167.80
Sliding-Window Bundle Adjustment	64.62
Pose Marginalization	1.06
Loop Closure Detection	306.68
Map Optimization	21.00

7.4 การทดลองการสร้างแผนที่ขนาใหญ่ภายในบริเวณจุฬาลงกรณ์มหาวิทยาลัย

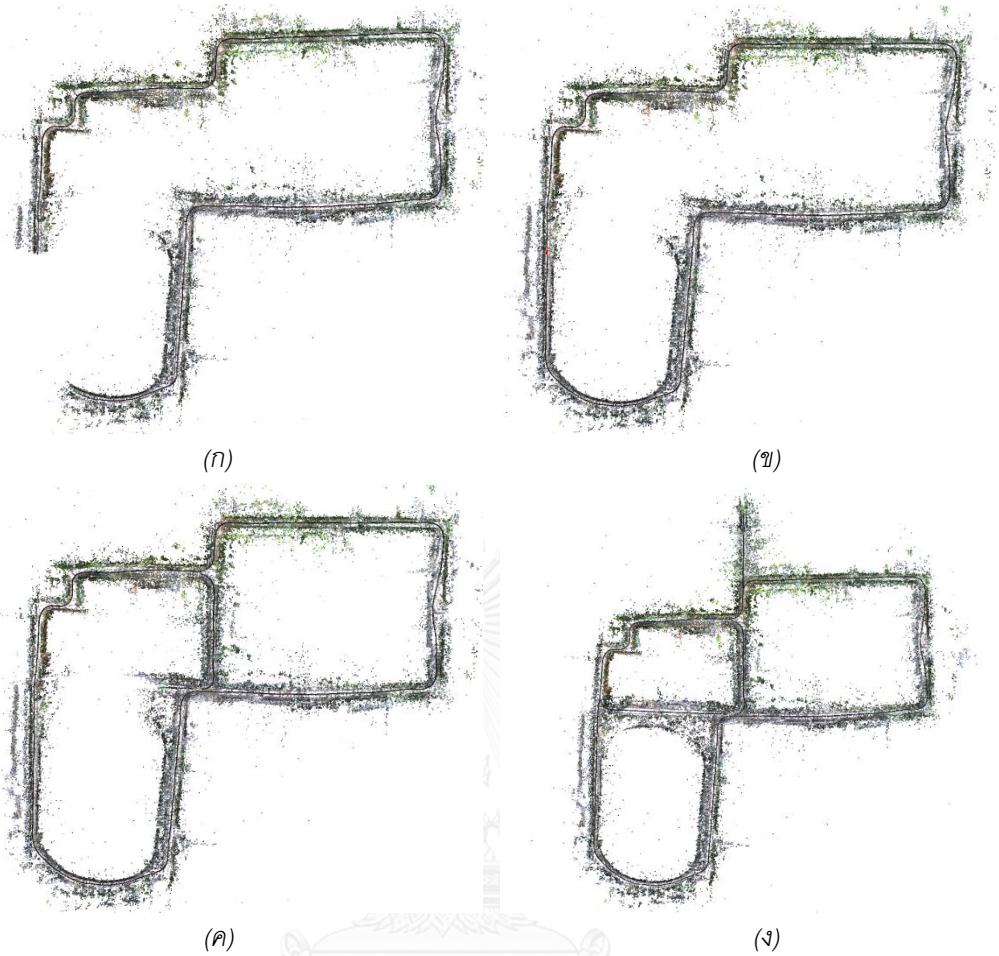
การทดลองนี้จะทดสอบอัลกอริทึมการระบุตำแหน่งพร้อมกับการสร้างแผนที่ในสิ่งแวดล้อมขนาดใหญ่ โดยสิ่งแวดล้อมที่จะใช้ในการทดลองนี้จะเป็นบริเวณภายในจุฬาลงกรณ์มหาวิทยาลัยฝั่งตะวันตกของถนนพญาไท โดยมีพื้นที่ในการทดลองขนาด 500 x 800 ตารางเมตร การเคลื่อนที่ของกล้องนั้นจะเคลื่อนที่ไปตามถนนโดยบริเวณรอบนอกหนึ่งรอบเป็นระยะทางประมาณ 1.6 กิโลเมตร จะนั้นจึงเคลื่อนที่วนซ้ำถนนด้านในเพื่อทดลองการ close loop และปรับแก้แผนที่โดยรวมแล้วเป็นระบะทางประมาณ 2.9 กิโลเมตร เส้นทางการเคลื่อนที่แสดงได้ดังรูปที่ 7.19

ลักษณะสิ่งแวดล้อมจะมีลักษณะเป็นกลุ่มอาคารข้างทาง, ลานกว้างสำหรับจอดรถ, แนวต้นไม้ ซึ่งตัวอย่างภาพที่ใช้ในการทดลองแสดงได้ดังรูปที่ 7.20 โดยการทดลองนี้จะประกอบด้วยลำดับภาพจำนวนทั้งสิ้น 4350 ภาพ



รูปที่ 7.19 ภาพเส้นทางการเคลื่อนที่การทดลองแผนที่ขนาดใหญ่ภายในจุฬาลงกรณ์มหาวิทยาลัย

รูปที่ 7.20 ตัวอย่างภาพที่ใช้ในการทดลองแผนที่ขนาดใหญ่ภายในจุฬาลงกรณ์มหาวิทยาลัย



(ก) (ข)

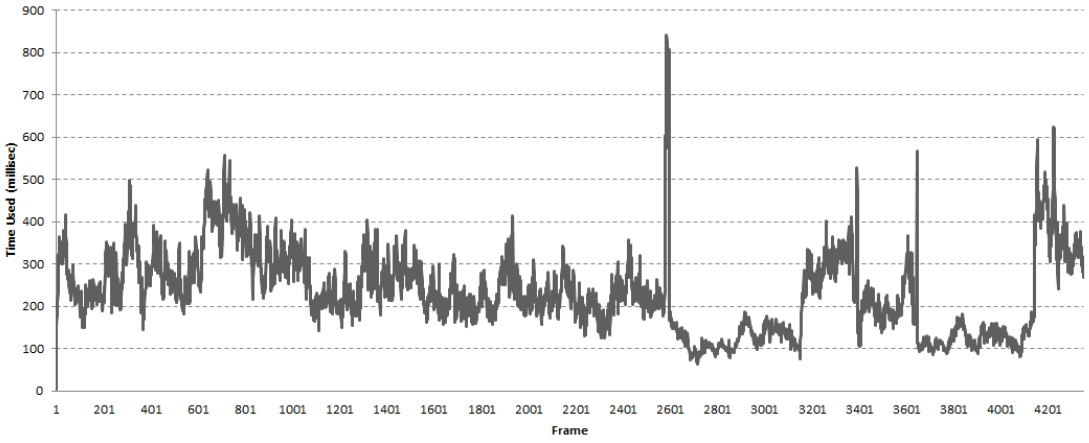
(ค) (ง)

(จ)

รูปที่ 7.21 ผลลัพธ์การระบุตำแหน่งพร้อมกับการสร้างแผนที่ขนาดใหญ่ภายในจุฬาลงกรณ์มหาวิทยาลัย

ผลการทดลองการระบุตำแหน่งและสร้างแผนที่แสดงได้ดังรูปที่ 7.21 โดยจะเห็นว่าอัลกอริทึมสามารถระบุตำแหน่งและสร้างแผนที่ที่มีขนาดใหญ่มากได้ โดยแผนที่ไม่ได้รับผลกระทบจากการเบี่ยงเบนของขนาดมากนัก และเมื่อนำแผนที่ที่สร้างได้จากอัลกอริทึมที่นำเสนอมาเทียบกับแผนที่ภาพถ่ายทางอากาศดังรูปที่ 7.21 (จ) จะเห็นได้ว่าแผนที่ที่สร้างได้นั้นมีความคลาดเคลื่อนจากแผนที่จริงพอสมควร ทั้งนี้เนื่องมาจากความคลาดเคลื่อนในการประมาณการเลื่อนตำแหน่งของกล้อง ซึ่งสำหรับแผนที่ที่มีขนาดใหญ่มากย่อมได้รับกระทบมาก อย่างไรก็ดี อัลกอริทึมที่นำเสนอยังสามารถตรวจเจอ Loop Closure และทำการ close loop พร้อมปรับแก้แผนที่ได้

เวลาในการทำงานโดยเฉลี่ยในหนึ่งเฟรม แสดงได้ดัง*ตาราง 7.4* และกราฟเวลาในการทำงานแสดงได้ดังรูปที่ 7.14 โดยจะเห็นได้ว่าเวลาในการทำงานค่อนข้างคงที่ ถึงแม้ว่าแผนที่จะมีขนาดใหญ่ขึ้นก็ตาม แสดงให้เห็นว่าอัลกอริทึมที่ได้นำเสนอ สามารถรับมือกับแผนที่ขนาดใหญ่ได้อย่างมีประสิทธิภาพ โดยจากกราฟ SLAM จะใช้เวลาในการทำงานเฉลี่ยเฟรมละ 233.63 มิลลิวินาที สำหรับบริเวณที่แท่งกราฟมีค่าสูงสุดสองจุดจะเป็นบริเวณที่อัลกอริทึมกำลังตรวจหา loop closure



รูปที่ 7.22 กราฟแสดงเวลาในการทำงานในแต่ละเฟรม (4)

ตาราง 7.4 ตารางแสดงเวลาการทำงานในแต่ละขั้นตอนของ SLAM (4)

Process	Time Used (millisec)
SIFT Feature Detector (GPU)	115.21
Feature Matching	142.56
Sliding-Window Bundle Adjustment	61.91
Pose Marginalization	0.82
Loop Closure Detection	448.07
Map Optimization	22.24

7.5 การทดลองการสร้างแผนที่ภาพถ่ายทางอากาศด้วย UAV

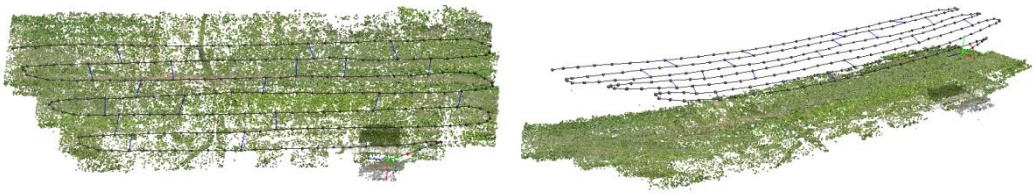
ในการทดลองก่อนหน้านี้จะเป็นการสร้างแผนที่ขนาดใหญ่แบบสามมิติ ซึ่งเป็นการเคลื่อนกล้องในแนวระนาบกับพื้นรอบ ๆ สิ่งแวดล้อม อย่างไรก็ดีอัลกอริทึม SLAM สามารถประยุกต์การใช้งานได้อย่างหลากหลายประเภท โดยในการทดลองนี้จะประยุกต์อัลกอริทึม SLAM มาใช้ในงานการสร้างแผนที่ภาพถ่ายทางอากาศด้วย UAV (Unmanned aerial vehicle) กล้องที่จะใช้จะเป็นกล้องแบบปรกติ ไม่ได้เป็นกล้องมุมกว้างแต่ก็สามารถทำงานได้กับอัลกอริทึมที่ได้นำเสนอ เนื่องจากโมเดลการวัดซึ่งอธิบายด้วย unit vector แสดงทิศทางของจุดสังเกตใน reference view นั้นครอบคลุมถึงกรณีกล้องแบบทั่วไป โดยกล้องที่ใช้จะเป็นกล้อง Sony NEX-5N, Focal length 16 mm ความละเอียดภาพ 4912 x 3264 pixels แต่ในการใช้งานจริงจะย่อรูปให้เหลือขนาด 2456 x 1632 pixels

การเคลื่อนที่ของ UAV จะเคลื่อนที่แบบ scan line เพื่อที่จะสามารถวิ่งได้ครอบคลุมพื้นที่ได้ทั้งหมด ภาพถ่ายที่ได้จากกล้องนั้นจะตั้งเวลาถ่ายประมาณภาพละสองวินาที โดยจะเป็นภาพมุมมองตั้งฉากกับพื้นดิน ภาพตัวอย่างซึ่งจะใช้ในการทดลองนี้แสดงได้ดังรูปที่ 7.23

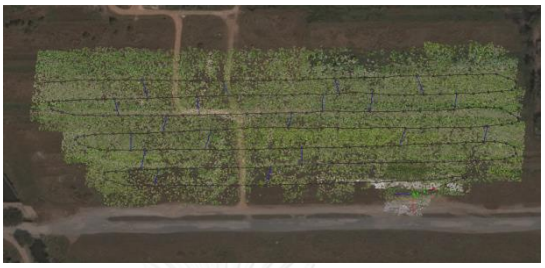
รูปที่ 7.23 ตัวอย่างภาพที่ใช้ในการทดลองการสร้างแผนที่ภาพถ่ายทางอากาศด้วย UAV

ผลลัพธ์การทดลองแสดงได้ดังรูปที่ 7.24 จะเห็นได้ว่าเมื่อ UAV มีการเคลื่อนที่วิ่งกลับไปกลับมาแบบ scan line อัลกอริทึมจะสร้างแผนที่ขึ้นมาแบบทันการณ์และใหญ่ขึ้นเรื่อย ๆ ตามเวลาที่ UAV วิ่งไป โดยการทำงานเมื่อได้รับภาพถ่ายเข้ามาใหม่ ระบบก็จะตรวจหาจุดสังเกตจากนั้นจึงประมาณการเคลื่อนที่ของ UAV แล้วจึงเพิ่ม view ใหม่เข้าในแผนที่ จากนั้นจึงทำการปรับแก้แผนที่เมื่อตรวจพบ loop closure โดยตำแหน่งที่เกิดการ close loop นั้นแสดงได้เป็นเส้นเชื่อมสีน้ำเงินระหว่างเส้นทางการเคลื่อนที่ของ UAV

(ก) *(ข)*



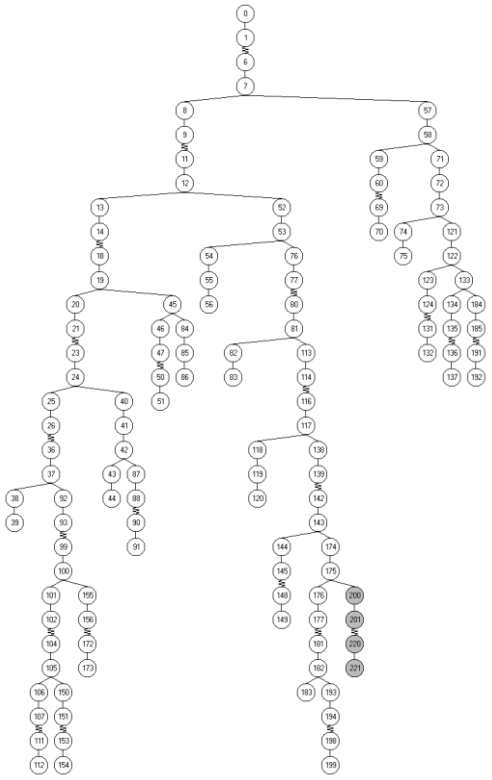
(ข) (ค)



(จ)

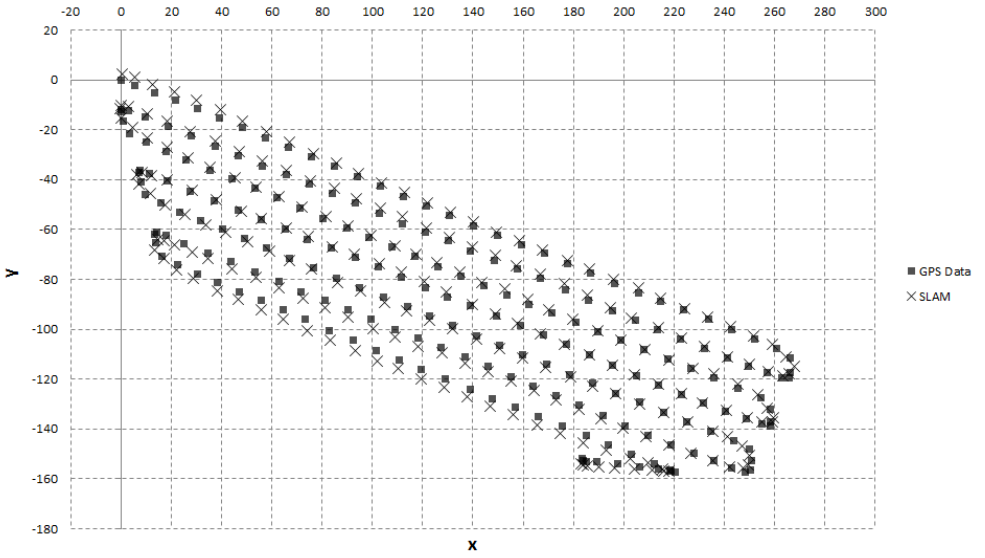
รูปที่ 7.24 ผลลัพธ์การระบุตำแหน่งพร้อมกับการสร้างแผนที่รอบศูนย์วิทยทรัพยากร

ต้นไม้ความสัมพันธ์ของโหนดแสดงได้ดังรูปที่ 7.25 โดยจะเห็นได้ว่าในแต่ละโหนดจะบรรยายตำแหน่งของ view แบบสัมพัทธ์เทียบกับโหนดพ่อ ซึ่งในการทดลองนี้ต้นไม้ได้แตกแขนงเป็นจำนวนมากเนื่องจากเกิดการ close loop หลายครั้ง

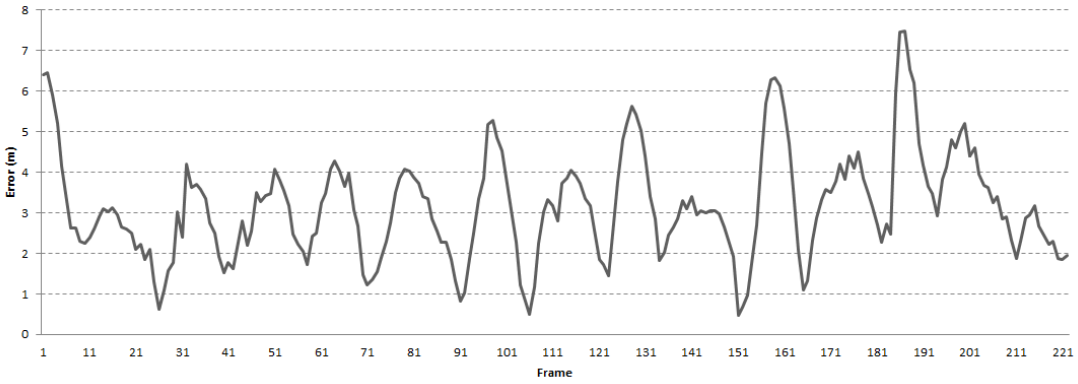


รูปที่ 7.25 ต้นไม้ความสัมพันธ์ของโหนด

ในการวัดความถูกต้องของการทำงานของอัลกอริทึมที่ได้นำเสนอ จะทำการวัดเทียบตำแหน่งของ UAV ที่ประมาณได้ในแต่ละเฟรม กับข้อมูลที่ได้จาก GPS ความคลาดเคลื่อนของการระบุตำแหน่งแสดงได้ดัง*รูปที่ 7.26* โดยสามารถแสดงกราฟความคลาดเคลื่อนของตำแหน่งในแต่ละเฟรม ได้ดังรูปที่ 7.27 ซึ่งโดยเฉลี่ยแล้วความคลาดเคลื่อนในแต่ละเฟรม จะอยู่ประมาณ 3.159 เมตร

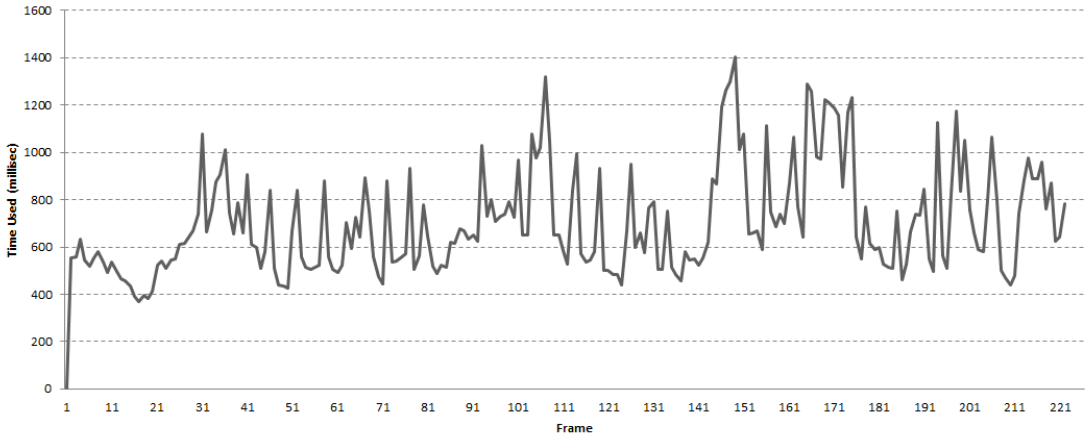


รูปที่ 7.26 กราฟเปรียบเทียบตำแหน่งของ UAV ที่ประมาณได้จาก SLAM เทียบกับ Ground Truth



รูปที่ 7.27 กราฟความคลาดเคลื่อนตำแหน่งของ UAV ที่ประมาณได้จาก SLAM เทียบกับ Ground Truth (2)

เวลาในการทำงานแสดงได้ดังรูปที่ 7.28 โดยจะเห็นได้ว่าเวลาในการทำงานของอัลกอริทึมจะค่อนข้างคงที่และไม่ขึ้นกับขนาดของแผนที่ มีเวลาเฉลี่ยประมาณเฟรมละ 699.49 มิลลิวินาที



รูปที่ 7.28 กราฟแสดงเวลาในการทำงานในแต่ละเฟรม (5)

บทที่ 8
สรุปการวิจัย

ในงานวิจัยนี้ได้เสนอวิธีการระบุตำแหน่งและสร้างแผนที่ขนาดใหญ่ด้วยกล้องวิดีโอมุมกว้าง โดยได้แก้ปัญหาข้อจำกัด ด้านความทนทานของอัลกอริมการระบุตำแหน่งและสร้างแผนที่สำหรับกล้องวิดีโอเพียงตัวเดียว และปัญหาประสิทธิภาพในการทำงานของการสร้างแผนที่ขนาดใหญ่ โดยขั้นตอนการทำงานจะแบ่งออกเป็นสามส่วนด้วยกันได้แก่ การประมาณการเคลื่อนที่จากข้อมูลภาพ, การปรับแก้แผนที่เพื่อหาค่าเหมาะสุด และการตรวจหา Loop Closure

การประมาณการเคลื่อนที่จากข้อมูลภาพของกล้องวิดีโอทั่วไปนั้น เนื่องจากว่าข้อมูลภาพที่ได้จากกล้องวิดีโอเพียงตัวเดียวนั้นไม่มีข้อมูลความลึกของภาพ ทำให้การสร้างแผนที่สามมิติขึ้นมาจากภาพนั้นเป็นไปได้ยากและไม่มีความทนทานมากพอสำหรับการเคลื่อนที่แบบอิสระ ในงานวิจัยนี้ได้นำเสนอการใช้กล้องวิดีโอมุมกว้างเพื่อลดปัญหาการขาดแคลนความสัมพันธ์ของจุดสังเกตในยามที่กล้องเปลี่ยนมุมมองกระทันหัน และได้เสนอวิธีใช้ Sliding-Window Bundle Adjustment ในการประมาณการเคลื่อนที่จากข้อมูลภาพของวิดีโอมุมกว้าง ซึ่งจะทำให้อัลกอริทึมประมาณการเคลื่อนที่มีความทนทานมากขึ้น นอกจากนี้ยังได้เสนอวิธีการจัดการปัญหาการเบี่ยงเบนขนาดของแผนที่ เพื่อให้ขนาดของแผนที่ไม่เปลี่ยนแปลงไปมากจนเกินไประหว่างการสร้างแผนที่ขนาดใหญ่

ในส่วนการปรับแก้แผนที่เพื่อหาค่าเหมาะสุด งานวิจัยนี้ได้นำเสนอวิธีในการลดเวลาในการทำงานโดยการปรับแก้แผนที่บนความสัมพันธ์ของตำแหน่งแบบต้นไม้ และการทำ Pose Marginalization รวมไปถึงการแยก thread การทำงานเพื่อให้สามารถประมาณคำตอบได้ทันการณ์ ผลลัพธ์จากการปรับแก้แผนที่ที่ได้นำเสนอจะเทียบเท่ากับการปรับแก้แผนที่แบบเต็ม โดยที่เวลาในการทำงานจะคงที่ไม่ขึ้นกับขนาดแผนที่

การตรวจหา Loop Closure ในงานวิจัยนี้ได้นำเสนอวิธีแบบ map-to-map โดยอาศัยข้อมูลการวัดจากภาพมาใช้ในการตรวจหาร่วมด้วย ซึ่งอัลกอริทึมสามารถทำงานได้ดีสำหรับ Loop Closure ทุกรูปแบบโดยมุมมองของภาพปัจจุบันไม่จำเป็นต้องสอดคล้องกับมุมมองภาพในอดีต นอกจากนี้ยังได้นำเสนอวิธีในการเลือก Candidate Submap โดยพิจารณาจากการกระจายความน่าจะเป็นที่จะเกิด Loop Closure เพื่อให้สามารถการตรวจหาได้ทันการณ์และลดความซับซ้อนการทำงาน

ผลการทดลองการระบุตำแหน่งและสร้างแผนที่ขนาดใหญ่ด้วยกล้องวิดีโอมุมกว้างในงานวิจัยนี้ จะใช้สิ่งแวดล้อมบริเวณรอบจุฬาลงกรณ์มหาวิทยาลัย โดยการทดลองจะมีทั้งการ Close Loop แบบทั่วไป, การ Close Loop หลายชั้น, การเคลื่อนกล้องแนวขนานกับการเคลื่อนที่ ซึ่งจะใช้กล้องวิดีโอแบบเลนส์ตาปลาในการทดลอง หรือการประยุกต์ใช้ในงานอื่น ๆ เช่น การทดลองการสร้างแผนที่ภาพถ่ายทางอากาศด้วย UAV ซึ่งผลการทดลองแสดงให้เห็นว่าอัลกอริทึมมีความสามารถในการระบุตำแหน่งและสร้างแผนที่สำหรับสิ่งแวดล้อมขนาดใหญ่ได้ มีประสิทธิภาพในการทำงานแบบทันการณ์ และเวลาในการทำงานไม่ขึ้นกับขนาดของแผนที่

8.1 ปัญหาที่พบในงานวิจัย

1. ปัญหา Camera Calibration: สำหรับงานระบุตำแหน่งและสร้างแผนที่ คุณภาพของ Camera Calibration นั้นมีผลกระทบต่อความแม่นยำของการทำงานของอัลกอริทึมเป็นอย่างมาก ทั้งนี้เนื่องจากว่าในขั้นตอนการคำนวนหาค่าการวัดจุดสังเกต (ซึ่งในงานวิจัยนี้จะบรรยายค่าการวัดด้วย unit vector (r) ที่ชี้จากกล้องไปยังจุดในสามมิติ) เราได้สมมุติให้มีการกระจายความน่าจะเป็นแบบ Guassian และมีค่าเฉลี่ยเป็นศูนย์ ถ้าหากว่า Camera Calibration มีความคลาดเคลื่อนแม้นเพียงหนึ่งองศา จะส่งผลให้การประมาณตำแหน่งจุดสังเกตในสามมิติผิดเพี้ยนไปมาก ซึ่งทำให้แผนที่ก่อนการ close loop มีการเบี่ยงเบนทั้งตำแหน่งและขนาดของแผนที่
2. ปัญหาคุณภาพของภาพจากกล้องวิดีโอทั่วไป: ในงานวิจัยนี้จะมุ่งเน้นให้สามารถใช้กล้องวิดีโอแบบทั่วไป (consumer grade) ในงานการระบุตำแหน่งและสร้างแผนที่ได้ โดยปัญหาที่พบก็คือ ภาพที่ได้จากกล้องจะมีความเบลอค่อนข้างมาก เมื่อกล้องมีการเปลี่ยนมุมมองอย่างรวดเร็ว นอกจากนี้การใช้เลนส์ตาปลาร่วมกับกล้องวิดีโอแบบทั่วไปยังมีผลให้ภาพที่ได้มีอาการ off focus ในบางบริเวณของภาพ ซึ่งปัญหาภาพเบลอนี้จะทำให้อัลกอริทึมในการตรวจหาจุดสังเกตทำงานได้ไม่ดีเท่าที่ควร
3. ปัญหาการเปลี่ยนแปลงของสิ่งแวดล้อม: การระบุตำแหน่งและสร้างแผนที่ในงานวิจัยนี้จะมีสมมุติฐานว่าสภาพสิ่งแวดล้อมต้องอยู่นิ่งขณะทำงาน โดยจะเห็นได้ว่าใน state transition model ที่นำเสนอจะไม่มีเทอมของการเปลี่ยนแปลงสิ่งแวดล้อม อย่างไรก็ดีในการทดลองจริง ย่อมต้องมีวัตถุเคลื่อนไหวบ้าง ไม่ว่าจะเป็นคนเดินผ่านหรือรถวิ่งผ่าน ซึ่งถ้าหากการเปลี่ยนแปลงของสิ่งแวดล้อมมีไม่มาก อัลกอริทึมที่นำเสนอจะมีความสามารถในการกรองข้อมูลการวัดที่ผิดพลาดทิ้งไปได้ แต่ถ้ามีการเปลี่ยนแปลงของสิ่งแวดล้อมมากเกินไป ก็จะส่งผลให้การทำงานของระบบล้มเหลว

8.2 แนวทางในการวิจัยขั้นถัดไป

1. การใช้งานการระบุตำแหน่งและสร้างแผนที่ในสิ่งแวดล้อมจริงนั้นย่อมหลีกเลี่ยงปัญหาการเปลี่ยนแปลงของสิ่งแวดล้อมไม่ได้ ซึ่งอัลกอริทึมที่ได้นำเสนอมีข้อจำกัดด้านการจัดการการเปลี่ยนแปลงของแผนที่ สำหรับอัลกอริทึมในขั้นตอนถัดไปควรจะต้องมีความสามารถในการแยกแยะระหว่างวัตถุอยู่นิ่งและวัตถุเคลื่อนไหวเพื่อให้สามารถระบุตำแหน่งและสร้างแผนที่ได้อย่างถูกต้อง นอกจากนี้อัลกอริทึมควรต้องมีความสามารถในการปรับแก้แผนที่เมื่อสภาพแวดล้อมมีการเปลี่ยนแปลงไปตามการณ์เวลา

2. อัลกอริทึมการตรวจหา Loop Closure ยังมีข้อจำกัดในการทำงานโดยเมื่อ แผนที่มีขนาดใหญ่ขึ้นและการกระจายความน่าจะเป็นของการเกิด Loop closure มีการกระจายมากขึ้น จะทำให้ตรวจหา Loop Closure ได้ยากขึ้น ทั้งนี้วิธีการแก้ปัญหาก็คือจะต้องขยายขนาด submap ที่ใช้ในการเปรียบเทียบให้มีขนาดใหญ่ขึ้นสัมพัทธ์กับขนาดของ loop
3. การทำงานของ Monocular Vision-based SLAM นั้นมีข้อจำกัดตรงที่ต้องใช้การเคลื่อนที่ของกล้องมาประมาณจุดสังเกตในสามมิติ ซึ่งถ้าหากว่ากล้องไม่มีการเคลื่อนที่ก็จะไม่สามารถประมาณแผนที่ได้ เมื่อเปรียบเทียบกับการทำงานของมนุษย์ถึงแม้มนุษย์จะมีข้อมูลเพียงภาพภาพเดียว มนุษย์ก็ยังสามารถอธิบายโครงสร้างของสิ่งแวดล้อมได้ ทั้งนี้เป็นเพราะมนุษย์มีความรู้เชิงความหมาย (Semantic) ของสิ่งแวดล้อม ซึ่งการจะทำให้หุ่นยนต์มีความสามารถใกล้เคียงกับมนุษย์มากขึ้นนั้น จะต้องเพิ่มความสามารถด้านเรียนรู้ความรู้เชิงความหมาย (Semantic) ให้กับหุ่นยนต์ด้วย

รายการอ้างอิง

1. Bailey, T. and H. Durrant-Whyte, *Simultaneous localization and mapping (SLAM): part I, II.* Robotics & Automation Magazine, IEEE, 2006. 13(3): p. 108-117.
2. Elseberg, J., D. Borrmann, and A. Nuchter. 6*DOF semi-rigid SLAM for mobile scanning*. in *Intelligent Robots and Systems (IROS),* 2012 *IEEE/RSJ International Conference on*. 2012.
3. Xilong, L., et al. *Robot pose estimation and navigation based on the understanding of laser landmarks in unknown environments*. in *Control & Automation (ICCA),* 11*th IEEE International Conference on*. 2014.
4. Steux, B. and O. El Hamzaoui. *tinySLAM: A SLAM algorithm in less than* 200 *lines C-language program*. in *Control Automation Robotics & Vision (ICARCV),* 2010 11*th International Conference on*. 2010.
5. Majdik, A.L., et al. *Laser and vision based map building techniques for mobile robot navigation*. in *Automation Quality and Testing Robotics (AQTR),* 2010 *IEEE International Conference on*. 2010.
6. Magree, D. and E.N. Johnson. *Combined laser and vision-aided inertial navigation for an indoor unmanned aerial vehicle*. in *American Control Conference (ACC),* 2014. 2014.
7. Kuen-Han, L. and W. Chieh-Chih. *Stereo-based simultaneous localization, mapping and moving object tracking*. in *Intelligent Robots and Systems (IROS),* 2010 *IEEE/RSJ International Conference on*. 2010.
8. Kitanov, A. and I. Petrovi. *Exactly Sparse Delayed State Filter based robust SLAM with Stereo Vision*. in *Robotics (ISR),* 2010 41*st International Symposium on and* 2010 6*th German Conference on Robotics (ROBOTIK)*. 2010.
9. Mei, C. *Robust and accurate pose estimation for vision-based localisation*. in *Intelligent Robots and Systems (IROS),* 2012 *IEEE/RSJ International Conference on*. 2012.
10. Mostofi, N., M. Elhabiby, and N. El-Sheimy. *Indoor localization and mapping using camera and inertial measurement unit (IMU)*. in *Position, Location and Navigation Symposium - PLANS* 2014*,* 2014 *IEEE/ION*. 2014.
11. Larnaout, D., et al. *Vehicle* 6*-DoF localization based on SLAM constrained by GPS and digital elevation model information*. in *Image Processing (ICIP),* 2013 20*th IEEE International Conference on*. 2013.
12. Pretto, A., E. Menegatti, and E. Pagello. *Omnidirectional dense large-scale mapping and navigation based on meaningful triangulation*. in *Robotics and Automation (ICRA),* 2011 *IEEE International Conference on*. 2011.

13. Guofeng, T., et al. *An Omni-directional vSLAM based on spherical camera model and 3D modeling*. in *Intelligent Control and Automation (WCICA),* 2012 10*th World Congress on*. 2012.

14. Kaiyu, W., et al. *The SLAM algorithm of mobile robot with omnidirectional vision based on EKF*. in *Information and Automation (ICIA),* 2012 *International Conference on*. 2012.

15. Henry, P., et al., *RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments.* Int. J. Rob. Res., 2012. 31(5): p. 647-663.

16. Taguchi, Y., et al. *Point-plane SLAM for hand-held 3D sensors*. in *Robotics and Automation (ICRA),* 2013 *IEEE International Conference on*. 2013.

17. Endres, F., et al., *3-D Mapping With an RGB-D Camera.* Robotics, IEEE Transactions on, 2014. 30(1): p. 177-187.

18. Kaess, M., A. Ranganathan, and F. Dellaert, *iSAM: Incremental Smoothing and Mapping*. Robotics, IEEE Transactions on, 2008. 24(6): p. 1365-1378.

19. Montemerlo, M., et al., *FastSLAM: a factored solution to the simultaneous localization and mapping problem*, in *Eighteenth national conference on Artificial intelligence*. 2002, American Association for Artificial Intelligence: Edmonton, Alberta, Canada. p. 593-598.

20. Montemerlo, M., et al., *FastSLAM 2.0: an improved particle filtering algorithm for simultaneous localization and mapping that provably converges*, in *Proceedings of the 18th international joint conference on Artificial intelligence*. 2003, Morgan Kaufmann Publishers Inc.: Acapulco, Mexico. p. 1151-1156.

21. Bailey, T., J. Nieto, and E. Nebot. *Consistency of the FastSLAM algorithm*. in *Robotics and Automation,* 2006. *ICRA* 2006. *Proceedings* 2006 *IEEE International Conference on*. 2006.

22. Weingarten, J. and R. Siegwart. *EKF-based 3D SLAM for structured environment reconstruction*. in *Intelligent Robots and Systems,* 2005. (*IROS* 2005). 2005 *IEEE/RSJ International Conference on*. 2005.

23. Jensfelt, P., et al. *A framework for vision based bearing only 3D SLAM*. in *Robotics and Automation,* 2006. *ICRA* 2006. *Proceedings* 2006 *IEEE International Conference on*. 2006.

24. Casarrubias-Vargas, H., et al. *EKF-SLAM and Machine Learning Techniques for Visual Robot Navigation*. in *Pattern Recognition (ICPR),* 2010 20*th International Conference on*. 2010.

25. Havangi, R., et al. *SLAM based on intelligent unscented Kalman filter*. in *Control, Instrumentation and Automation (ICCIA),* 2011 2*nd International Conference on*. 2011.

26. Martinez-Cantin, R. and J.A. Castellanos. *Unscented SLAM for large-scale outdoor environments*. in *Intelligent Robots and Systems,* 2005. (*IROS* 2005). 2005 *IEEE/RSJ International Conference on*. 2005.

27. Thrun, S., et al., *Simultaneous Localization and Mapping With Sparse Extended Information Filters.* International Journal of Robotics Research, 2004.

28. Eustice, R.M., H. Singh, and J.J. Leonard, *Exactly Sparse Delayed-State Filters for View-Based SLAM.* Robotics, IEEE Transactions on, 2006. 22(6): p. 1100-1114.

29. Frese, U., *A discussion of simultaneous localization and mapping.* Autonomous Robots, 2006. 20: p. 25-42.

30. Walter, M.R., R.M. Eustice, and J.J. Leonard, *Exactly Sparse Extended Information Filters for Feature-based SLAM.* Int. J. Rob. Res., 2007. 26(4): p. 335-359.

31. Xiaohua, W. and M. Liping. *Sparse extended information filter for feather-based SLAM*. in *Electronics, Communications and Control (ICECC),* 2011 *International Conference on*. 2011.

32. Okawa, Y. and T. Namerikawa. *Simultaneous localization and mapping problem via the H_infinity filter with a known landmark*. in *SICE Annual Conference (SICE),* 2013 *Proceedings of*. 2013.

33. Chandra, K.P.B., G. Da-wei, and I. Postlethwaite. *SLAM using EKF, EH_infinity and mixed EH_infinity^2 filter*. in *Intelligent Control (ISIC),* 2010 *IEEE International Symposium on*. 2010.

34. Grisetti, G., et al., *A Tutorial on Graph-Based SLAM.* Intelligent Transportation Systems Magazine, IEEE, 2010. 2(4): p. 31-43.

35. Frese, U., *Treemap: An O(log n) algorithm for indoor simultaneous localization and mapping.* Autonomous Robots, 2006. 21: p. 103-122.

36. Grisetti, G., C. Stachniss, and W. Burgard, *Non-linear constraint network optimization for efficient map learning.* IEEE Transactions on Intelligent Transportation Systems, 2009.

37. Kaess, M., et al. *iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering*. in *Robotics and Automation (ICRA),* 2011 *IEEE International Conference on*. 2011.

38. Dellaert, F. and M. Kaess, *Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing*. Int. J. Rob. Res., 2006. 25(12): p. 1181-1203.

39. Grisetti, G., et al. *Hierarchical optimization on manifolds for online 2D and 3D mapping*. in *Robotics and Automation (ICRA),* 2010 *IEEE International Conference on*. 2010.

40. Ila, V., J.M. Porta, and J. Andrade-Cetto, *Information-Based Compact Pose SLAM.* Robotics, IEEE Transactions on, 2010. 26(1): p. 78-93.

41. Wang, J. and E. Olson. *Robust pose graph optimization using stochastic gradient descent*. in *Robotics and Automation (ICRA),* 2014 *IEEE International Conference on*. 2014.

42. Liang, Z., et al. *Parallax angle parametrization for monocular SLAM*. in *Robotics and Automation (ICRA),* 2011 *IEEE International Conference on*. 2011.

43. Hyon, L., L. Jongwoo, and H.J. Kim. *Real-time 6-DOF monocular visual SLAM in a large-scale environment*. in *Robotics and Automation (ICRA),* 2014 *IEEE International Conference on*. 2014.

44. Lourakis, M.I.A. and A.A. Argyros, *SBA: A software package for generic sparse bundle adjustment.* ACM Trans. Math. Softw., 2009. 36(1): p. 1-30.

45. Davison, A.J., et al., *MonoSLAM: Real-Time Single Camera SLAM.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2007. 29(6): p. 1052-1067.

46. Eade, E. and T. Drummond. *Scalable Monocular SLAM*. in *Computer Vision and Pattern Recognition,* 2006 *IEEE Computer Society Conference on*. 2006.

47. Zhan, W. and G. Dissanayake. *Efficient Monocular SLAM using sparse information filters*. in *Information and Automation for Sustainability (ICIAFs),* 2010 5*th International Conference on*. 2010.

48. Junghyun, K. and L. Kyoung Mu. *Monocular SLAM with locally planar landmarks via geometric rao-blackwellized particle filtering on Lie groups*. in *Computer Vision and Pattern Recognition (CVPR),* 2010 *IEEE Conference on*. 2010.

49. Klein, G. and D.W. Murray, *Parallel Tracking and Mapping for Small AR Workspaces*., in *ISMAR*. 2007, IEEE. p. 225-234.

50. Williams, B. and I. Reid. *On combining visual SLAM and visual odometry*. in *Robotics and Automation (ICRA),* 2010 *IEEE International Conference on*. 2010.

51. Sünderhauf, N., et al., *Visual odometry using sparse bundle adjustment on an autonomous outdoor vehicle*, in *In Tagungsband Autonome Mobile Systeme*. 2005, Springer Verlag.

52. Newcombe, R.A. and A.J. Davison, *Live dense reconstruction with a single moving camera*, in *IEEE Conference on Computer Vision and pattern Recognition*. 2010.

53. Newcombe, R.A., S.J. Lovegrove, and A.J. Davison, *DTAM: Dense tracking and mapping in real-time*, in *Proceedings of the* 2011 *International Conference on Computer Vision*. 2011, IEEE Computer Society. p. 2320-2327.

54. Graber, G., T. Pock, and H. Bischof. *Online 3D reconstruction using convex optimization*. in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. 2011.

55. Pizzoli, M., C. Forster, and D. Scaramuzza. *REMODE: Probabilistic, monocular dense reconstruction in real time*. in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. 2014.

56. Pirchheim, C., D. Schmalstieg, and G. Reitmayr. *Handling pure camera rotation in keyframe-based SLAM*. in *Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on*. 2013.

57. Gauglitz, S., et al. *Live tracking and mapping from both general and rotation-only camera motion*. in *Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on*. 2012.

58. Civera, J., et al. *Towards semantic SLAM using a monocular camera*. in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. 2011.

59. Botterill, T., S. Mills, and R. Green, *Correcting Scale Drift by Object Recognition in Single-Camera SLAM.* Cybernetics, IEEE Transactions on, 2013. 43(6): p. 1767-1780.

60. Salas-Moreno, R.F., et al. *SLAM++: Simultaneous Localisation and Mapping at the Level of Objects*. in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. 2013.

61. Estrada, C., J. Neira, and J.D. Tardos, *Hierarchical SLAM: Real-Time Accurate Mapping of Large Environments.* Robotics, IEEE Transactions on, 2005. 21(4): p. 588-596.

62. Paz, L.M., J.D. Tardos, and J. Neira, *Divide and Conquer: EKF SLAM in O(n)*. Robotics, IEEE Transactions on, 2008. 24(5): p. 1107-1120.

63. Pinies, P. and J.D. Tardos, *Large-Scale SLAM Building Conditionally Independent Local Maps: Application to Monocular Vision.* Robotics, IEEE Transactions on, 2008. 24(5): p. 1094-1106.

64. Shoudong, H., W. Zhan, and G. Dissanayake, *Sparse Local Submap Joining Filter for Building Large-Scale Maps.* Robotics, IEEE Transactions on, 2008. 24(5): p. 1121-1130.

65. Liang, Z., et al. *Large-scale monocular SLAM by local bundle adjustment and map joining*. in *Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on*. 2010.

66. Oberlander, J., A. Roennau, and R. Dillmann. *Hierarchical SLAM using spectral submap matching with opportunities for long-term operation*. in *Advanced Robotics (ICAR), 2013 16th International Conference on*. 2013.

67. Beall, C., et al. *Bundle adjustment in large-scale 3D reconstructions based on underwater robotic surveys*. in *OCEANS, 2011 IEEE - Spain*. 2011.

68. Blanco, J.L., J. Gonzalez-Jimenez, and J.A. Fernandez-Madrigal. *Sparser Relative Bundle Adjustment (SRBA): Constant-time maintenance and local optimization of arbitrarily large maps*. in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. 2013.

69. Kummerle, R., et al. *G2o: A general framework for graph optimization*. in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. 2011.

70. Ila, V., J.M. Porta, and J. Andrade-Cetto. *Reduced state representation in delayed-state SLAM*. in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. 2009.

71. Clemente, L.A., et al., *Mapping large loops with a single hand-held camera*. Robotics Science and Systems, 2007.

72. Nuchter, A., et al., *6D SLAM - 3D mapping outdoor environments: Research Articles*. J. Field Robot., 2007. 24(8-9): p. 699-722.

73. Douillard, B., D. Fox, and F. Ramos, *Laser and Vision Based Outdoor Object Mapping*, in *Robotics: Science and Systems'08*. 2008. p. -1-1.

74. Agrawal, M. and K. Konolige. *Real-time Localization in Outdoor Environments using Stereo Vision and Inexpensive GPS*. in *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*. 2006.

75. Floros, G., B. van der Zander, and B. Leibe. *OpenStreetSLAM: Global vehicle localization using OpenStreetMaps*. in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. 2013.

76. Williams, B., et al., *A comparison of loop closing techniques in monocular SLAM*. Robot. Auton. Syst., 2009. 57(12): p. 1188-1197.

77. Williams, B., P. Smith, and I. Reid, *Automatic Relocalisation for a Single-Camera Simultaneous Localisation and Mapping System*, in *IEEE International Conference on Robotics and Automation*. 2007.

78. Williams, B., et al. *An image-to-map loop closing method for monocular SLAM*. in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. 2008.

79. Williams, B., G. Klein, and I. Reid, *Automatic Relocalization and Loop Closing for Real-Time Monocular SLAM.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2011. 33(9): p. 1699-1712.

80. Cummins, M. and P. Newman, *Appearance-only SLAM at large scale with FAB-MAP 2.0*. Int. J. Rob. Res., 2011. 30(9): p. 1100-1123.

81. Eade, E. and T. Drummond, *Unified loop closing and recovery for real time monocular slam*, in *In British Machine Vision Conference*. 2008.

82. Angeli, A., et al. *Real-time visual loop-closure detection*. in *Robotics and Automation,* 2008. *ICRA* 2008. *IEEE International Conference on*. 2008.

83. Angeli, A., et al., *Fast and Incremental Method for Loop-Closure Detection Using Bags of Visual Words.* Robotics, IEEE Transactions on, 2008. 24(5): p. 1027-1037.

84. Cummins, M. and P. Newman, *FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance.* Int. J. Rob. Res., 2008. 27(6): p. 647-665.

85. Bay, H., T. Tuytelaars, and L.V. Gool, *SURF: speeded up robust features*, in *Proceedings of the 9th European conference on Computer Vision - Volume Part I*. 2006, Springer-Verlag: Graz, Austria. p. 404-417.

86. Galvez, L., et al., *Bags of Binary Words for Fast Place Recognition in Image Sequences.* Robotics, IEEE Transactions on, 2012. 28(5): p. 1188-1197.

87. Rosten, E. and T. Drummond, *Machine learning for high-speed corner detection*, in *Computer Vision - ECCV*. 2006. p. 430-443.

88. Calonder, M., et al., *BRIEF: binary robust independent elementary features*, in *Proceedings of the 11th European conference on Computer vision: Part IV*. 2010, Springer-Verlag: Heraklion, Crete, Greece. p. 778-792.

89. Mur-Artal, R. and J.D. Tardos. *Fast relocalisation and loop closing in keyframe-based SLAM*. in *Robotics and Automation (ICRA),* 2014 *IEEE International Conference on*. 2014.

90. Shahbazi, H. and H. Zhang. *Application of Locality Sensitive Hashing to realtime loop closure detection*. in *Intelligent Robots and Systems (IROS),* 2011 *IEEE/RSJ International Conference on*. 2011.

91. Girdhar, Y. and G. Dudek. *Online Visual Vocabularies*. in *Computer and Robot Vision (CRV),* 2011 *Canadian Conference on*. 2011.

92. Nicosevici, T. and R. Garcia, *Automatic Visual Bag-of-Words for Online Robot Navigation and Mapping.* Robotics, IEEE Transactions on, 2012. 28(4): p. 886-898.

93. Yang, L. and Z. Hong. *Visual loop closure detection with a compact image descriptor*. in *Intelligent Robots and Systems (IROS),* 2012 *IEEE/RSJ International Conference on*. 2012.

94. Junjun, W., Z. Hong, and G. Yisheng. *An efficient visual loop closure detection method in a map of 20 million key locations*. in *Robotics and Automation (ICRA),* 2014 *IEEE International Conference on*. 2014.

95. Davison, A.J. and D.W. Murray, *Simultaneous localization and map-building using active vision.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2002. 24(7): p. 865-880.

96. Leonard, J. and P. Newman, *Consistent, convergent, and constant-time SLAM*, in *Proceedings of the 18th international joint conference on Artificial intelligence*. 2003, Morgan Kaufmann Publishers Inc.: Acapulco, Mexico. p. 1143-1150.

97. Taylor, C.J. and D.J. Kriegman, *Minimization on the Lie group SO(3) and related manifolds*, in *Technical Report* 9405. 1994, Yale University.

98. Lee, J.M., *Introduction to Smooth Manifolds*. Graduate Texts in Mathematics. Vol. 218. 2003: Springer Verlag.

99. Hertzberg, C., *A framework for sparse, non-linear least squares problems on manifolds*. 2008, Univ. Bremen.

100. D., S., *Omnidirectional Vision: from Calibration to Robot Motion Estimation*. 2008, ETH Zurich: Zurich.

101. Harris, C. and M. Stephens, *A combined corner and edge detector*, in *Proceedings of The Fourth Alvey Vision Conference*. 1988.

102. Agrawal, M., K. Konolige, and M. Rufus Blas, *CenSurE: Center Surround Extremas for Realtime Feature Detection and Matching*. Computer Vision - ECCV. Vol. 5305. 2008. 102-115.

103. Lowe, D.G., *Distinctive Image Features from Scale-Invariant Keypoints.* Int. J. Comput. Vision, 2004. 60(2): p. 91-110.

104. Forssen, P.E. and D.G. Lowe. *Shape Descriptors for Maximally Stable Extremal Regions*. in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. 2007.

105. Muja, M. and D.G. Lowe, *Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration*, in *International Conference on Computer Vision Theory and Application VISSAPP'09)*. 2009, INSTICC Press. p. 331-340.

106. Tue-Cuong, D.-S. and A.I. Mourikis. *Consistency analysis for sliding-window visual odometry*. in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. 2012.

107. Kitt, B., A. Geiger, and H. Lategahn. *Visual odometry based on stereo image sequences with RANSAC-based outlier rejection scheme*. in *Intelligent Vehicles Symposium (IV), 2010 IEEE*. 2010.

108. Howard, A. *Real-time stereo visual odometry for autonomous ground vehicles*. in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. 2008.

109. Wei, M., W. Han, and G. Seet. *Efficient visual odometry estimation using stereo camera*. in *Control & Automation (ICCA),* 11*th IEEE International Conference on*. 2014.

110. Segal, A.V., D. Haehnel, and S. Thrun, *Generalized-ICP.* Robotics: Science and Systems (RSS), 2005.

111. Dryanovski, I., R.G. Valenti, and X. Jizhong. *Fast visual odometry and mapping from RGB-D data*. in *Robotics and Automation (ICRA),* 2013 *IEEE International Conference on*. 2013.

112. Whelan, T., et al. *Robust real-time visual odometry for dense RGB-D mapping*. in *Robotics and Automation (ICRA),* 2013 *IEEE International Conference on*. 2013.

113. Kerl, C., J. Sturm, and D. Cremers. *Robust odometry estimation for RGB-D cameras*. in *Robotics and Automation (ICRA),* 2013 *IEEE International Conference on*. 2013.

114. Steinbrucker, F., J. Sturm, and D. Cremers. *Real-time visual odometry from dense RGB-D images*. in *Computer Vision Workshops (ICCV Workshops),* 2011 *IEEE International Conference on*. 2011.

115. Nister, D., *An efficient solution to the five-point relative pose problem.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2004. 26(6): p. 756-770.

116. Nister, D., O. Naroditsky, and J. Bergen, *Visual odometry for ground vehicle applications.* Journal of Field Robotics, 2006. 23: p. 2006.

117. Tardif, J.P., Y. Pavlidis, and K. Daniilidis. *Monocular visual odometry in urban environments using an omnidirectional camera*. in *Intelligent Robots and Systems,* 2008. *IROS* 2008. *IEEE/RSJ International Conference on*. 2008.

118. Jwu-Sheng, H. and C. Ming-Yuan. *A sliding-window visual-IMU odometer based on tri-focal tensor geometry*. in *Robotics and Automation (ICRA),* 2014 *IEEE International Conference on*. 2014.

119. Li, M. and A.I. Mourikis, *High-precision, consistent EKF-based visual-inertial odometry.* Int. J. Rob. Res., 2013. 32(6): p. 690-711.

120. Fraundorfer, F., D. Scaramuzza, and M. Pollefeys. *A constricted bundle adjustment parameterization for relative scale estimation in visual odometry*. in *Robotics and Automation (ICRA),* 2010 *IEEE International Conference on*. 2010.

121. Shiyu, S., M. Chandraker, and C.C. Guest. *Parallel, real-time monocular visual odometry*. in *Robotics and Automation (ICRA),* 2013 *IEEE International Conference on*. 2013.

122. Xiao-shan, G., et al., *Complete solution classification for the perspective-three-point problem.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2003. 25(8): p. 930-943.

123. Lepetit, V., F. Moreno-Noguer, and P. Fua, *EPnP: An Accurate O(n) Solution to the PnP Problem.* Int. J. Comput. Vision, 2009. 81(2): p. 155-166.

124. Strasdat, H., J. Montiel, and A.J. Davison, *Scale Drift-Aware Large Scale Monocular SLAM.* Robotics: Science and Systems (RSS), 2010.

125. Kretzschmar, H. and C. Stachniss, *Information-theoretic compression of pose graphs for laser-based SLAM.* Int. J. Rob. Res., 2012. 31(11): p. 1219-1230.

126. Johannsson, H., et al. *Temporally scalable visual SLAM using a reduced pose graph*. in *Robotics and Automation (ICRA),* 2013 *IEEE International Conference on*. 2013.

127. Callmer, J., et al., *Tree of Words for Visual Loop Closure Detection in Urban SLAM*, in *Proceedings of the* 2008 *Australasian Conference on Robotics & Automation*. 2008.

128. Csurka, G., et al., *Visual categorization with bags of keypoints.* Workshop on Statistical Learning in Computer Vision, ECCV, 2004: p. 1-22.

129. Arun, K.S., T.S. Huang, and S.D. Blostein, *Least-Squares Fitting of Two* 3*-D Point Sets.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, 1987. **PAMI**-9(5): p. 698-700.

ภาคผนวก

ภาคผนวก ก

Scale Invariance Feature Transform (SIFT)

ในการตรวจหาความคล้ายคลึงกันระหว่างภาพสองภาพ วิธีพื้นฐานที่สุดก็คือการนำเอาจุด pixel แต่ละจุดในภาพสองภาพมาหาค่าความต่างแล้วเปรียบเทียบว่ามีความสอดคล้องแค่ไหน แต่อย่างไรก็ดีวิธีนี้อาจใช้ไม่ได้ในความเป็นจริง เนื่องจากถ้าหากว่าภาพทั้งสอง ไม่ตรงกันพอดีเช่นมุมมองอาจจะเลื่อนหรือหมุนไป วิธีที่กล่าวมาก็จะใช้ไม่ได้

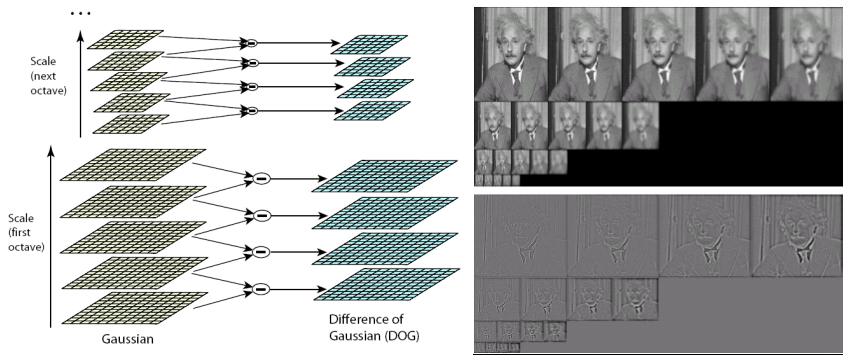
SIFT[103] เป็นวิธีในการตรวจหา Features และ Feature Descriptors ในภาพโดย Feature ที่ตรวจหาได้จาก SIFT จะรับประกันคุณสมบัติสองประการ คือ Scale Invariant และ Rotation Invariant หมายความว่าถึงแม้ภาพจะมีขนาดต่างกัน หรือภาพมีการหมุน อัลกอริทึม SIFT จะสามารถตรวจหา Feature เดิมได้ ส่วน Feature Descriptor จะใช้ในการอธิบายลักษณะเฉพาะของ Feature เพื่อจะนำไปหาความสัมพันธ์ของ Feature ระหว่างภาพสองภาพต่อไป จุดเด่นของ SIFT ก็คือ Features ที่ตรวจวัดได้จะมีความทนทานต่อ Image noise และการเปลี่ยนแปลงของค่าความสว่างของภาพ ขั้นตอนการตรวจหา Features และ Feature Descriptors ของ SIFT สามารถอธิบายได้ดังนี้

Scale-space extrema detection

ในขั้นตอนแรกก่อนที่จะตรวจหา Keypoint จะต้องนำภาพที่ต้องการตรวจหา Keypoint มา convolute ด้วย Gaussian filters ที่ Scales ต่าง ๆ กัน จากนั้นก็คำนวนหา Difference-of-Gaussian images (DoG) จากความต่างของภาพที่ทำ Gaussian blur แล้วดัง*รูปที่ 8.1* (ก) (ได้ผลลัพธ์ดัง*รูปที่ 8.1* (ข))

$$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

$$G(x, y, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

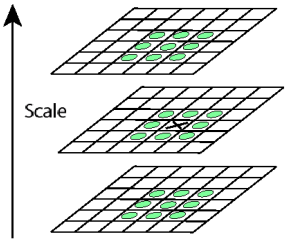


(ก) *(ข)*

รูปที่ 8.1 (ก) ขั้นตอนการหา Difference-of-Gaussian (ข) ภาพหลังจากทำ Gaussian blur ที่ Scales ต่าง ๆ กัน (ภาพบน) และภาพ Difference-of-Gaussian (ภาพล่าง)

Keypoint localization

SIFT keypoints หาได้จาก local maxima หรือ minima ในแต่ระหว่าง Scales โดยแต่ละ pixel ใน DoG image จะถูกเปรียบเทียบกับ pixel รอบข้าง 8 pixels ใน scale เดียวกัน และอีก 18 pixels รอบข้างใน DoG images ที่ scale ข้างเคียงถ้า pixel เป็น local maximum หรือ minimum ก็จะถูกเลือกเป็น candidate keypoints (*รูปที่ 8.2*) หลังจากนั้นในแต่ละ candidate keypoint จะทำ data Interpolation เพื่อประมาณตำแหน่ง pixel ที่แม่นยำขึ้น และลบ keypoint ที่ contrast ต่ำเกินไป หรือ keypoint ที่มีแนวโน้มจะเป็น edge

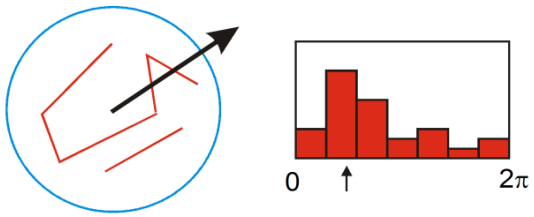


รูปที่ 8.2 ภาพการหา local maxima หรือ minima

Orientation assignment

ขั้นตอนการหา keypoint orientation อธิบายดังนี้

1. คำนวน Gradient Orientation Histogram จาก pixel รอบ ๆ ข้าง (รูปที่ 8.3)
2. สำหรับตำแหน่งใน Histogram ที่มีค่าสูงสุดเทียบกับรอบข้างให้ถือเป็น dominant orientations และถ้าหากมีตำแหน่งสูงสุดมากกว่า 1 ตำแหน่งให้เพิ่ม Feature ใหม่ที่ตำแหน่งเดิม
3. ในการหา Feature Descriptor จะคำนวนสัมพัทธ์กับ orientation และ scale ที่หาได้ ส่งผลให้ feature นั้นไม่ขึ้นกับ Scale และ Rotation

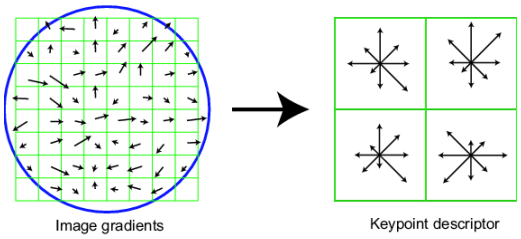


รูปที่ 8.3 แสดงการหา Gradient Orientation Histogram

Generation of keypoint descriptors

การหา keypoint descriptor ในขั้นตอนแรกจะต้องคำนวนขนาดและทิศทางของ gradient ในแต่ละ sample point ซึ่งอยู่ในบริเวณรอบ ๆ ตำแหน่ง keypoint ดังที่ได้แสดงใน*รูปที่ 8.4* (ซ้าย) ค่าในแต่ละ Sample เหล่านั้นจะถูกรวมกันเป็น orientation histograms ในบริเวณย่อยจำนวน 4x4 ช่อง ดังที่ได้แสดงใน*รูปที่ 8.4*

(ขวา) ซึ่งความยาวของลูกศรคือขนาดของ gradient ในทิศทางนั้น ๆ keypoint descriptors ก็คือ vector ที่เก็บค่าขนาดของ gradient ในแต่ละ histogram ทั้งหมด



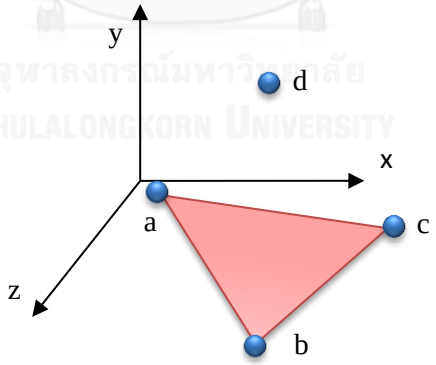
รูปที่ 8.4 แสดงการหา keypoint descriptor

ภาคผนวก ข
การตรวจหาพื้นจากกลุ่มจุดในสามมิติ

อัลกอริทึมการตรวจหาพื้นที่ใช้ในงานวิจัยนี้ จะสมมุติให้พื้นบริเวณที่สนใจเป็นพื้นเรียบ และอัลกอริทึมจะตรวจหา plane จากข้อมูลจุดสังเกตซึ่งใช้อธิบายพื้นในสามมิติ

โดยทั่วไปแล้วอัลกอริทึมในการตรวจหา plane จากข้อมูล point cloud มักจะนิยมใช้อัลกอริทึม RANSAC ซึ่งการทำงานจะเริ่มจากการสุ่มจุดสามจุดแล้วคำนวนหาตำแหน่ง plane จากจุดสามจุดนั้น หลังจากนั้นจึงทดสอบ plane กับชุดข้อมูลที่เหลือเพื่อเปรียบเทียบว่า plane ใดสอดคล้องกับจุดทั้งหมดใน point clound มากสุด อย่างไรก็ดีในงานวิจัยนี้เมื่อนำมาทดลองใช้กับข้อมูลตำแหน่งจุดสังเกตในสามมิติ ก็พบว่าอัลกอริทึม RANSAC ทำงานได้ไม่ดีนัก สาเหตุเนื่องจากกลุ่มจุดสังเกตที่สร้างได้จาก SLAM โดยใช้กล้องวิดีโอเพียงตัวเดียวนั้นมีความคลาดเคลื่อนค่อนข้างสูงและแปรผันตามระยะห่างจากกล้อง ทำให้เกิดความคลาดเคลื่อนในการประมาณตำแหน่ง plane ได้ง่าย นอกจากนี้ในบางครั้งจุดสังเกตที่อยู่บนพื้นนั้นมีไม่มากพอทำให้การตรวจหา plane ด้วย RANSAC ทำให้อาจตรวจพบ plane อื่นที่ไม่ใช่พื้นแทนก็เป็นได้ ดังนั้นจึงต้องใช้วิธีอื่นในการประมาณ plane แทน

สำหรับ plane ที่ใช้อธิบายพื้นในที่นี้ จะนิยามให้เป็น plane ที่อยู่สูงที่สุดที่ยังสามารถ support จุดสังเกตทั้งหมดได้ ยกตัวอย่างเช่นใน*รูปที่ 8.5* มีจุดสังเกตจำนวน 4 จุดได้แก่จุด a, b, c และ d เมื่อกำหนดให้แกน y เป็นทิศทางตั้งฉากกับพื้น โดยการประมาณพื้นซึ่งใช้อธิบายจุดทั้งสี่จุดนี้คือ plane ที่เกิดจากจุด a, b, และ c



รูปที่ 8.5 ตัวอย่างการสร้าง plane จากจุดสังเกต 4 จุด

อย่างไรก็ดีสำหรับตำแหน่งจุดสังเกตที่คำนวนได้จาก SLAM นั้นในบางครั้งอาจมีความคลาดเคลื่อนมากจนทำให้ตำแหน่งจุดสังเกตอยู่ต่ำกว่าตำแหน่งพื้นที่ควรจะเป็น ดังนั้นในการประมาณตำแหน่ง plane จะใช้วิธี optimization โดยยินยอมให้มีจุดสังเกตอยู่ต่ำกว่า plane ได้บ้าง

สำหรับ error function ที่จะใช้ในกระบวนการ optimization นั้น error จะเป็นฟังก์ชันกับระยะห่างระหว่างจุดสังเกตกับ plane โดยคุณลักษณะของฟังก์ชันที่ต้องการนั้น error เป็นศูนย์เมื่อจุดสังเกตอยู่บน plane

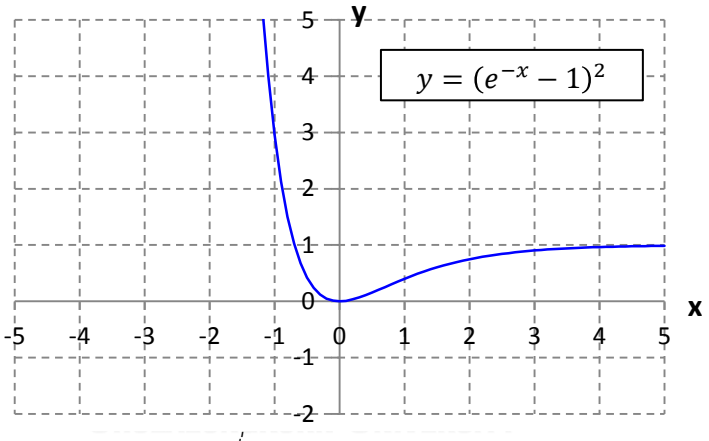
พอดี สำหรับจุดสังเกตที่อยู่เหนือ plane จะต้องมีค่า error ไม่สูงมากนักเพราะเราต้องการให้ plane support จุดสังเกตส่วนใหญ่ ส่วนจุดสังเกตที่อยู่ต่ำกว่า plane จะต้องมีค่า error สูงเพื่อป้องกันไม่ให้มีจุดสังเกตอยู่ใต้ plane มากจนเกินไป

กำหนดให้ $p_i \in P$ เป็นตำแหน่งจุดสังเกตใน point cloud และนิยาม plane จากเวกเตอร์ $V = [n \quad d]^T$ เมื่อ $\forall x \in V: \langle n, x \rangle + d = 0$ จะเขียน error function ได้เป็น

$$e_i = \left(k^{-f(p_i)} - 1\right)^2$$

เมื่อ k เป็นค่าคงที่ค่าหนึ่ง และ $f(x)$ เป็นฟังก์ชันคำนวนระยะทางจาก ตำแหน่งจุดสังเกตถึง plane $f(x) = \langle n, x \rangle + d$

กราฟฟังก์ชัน error จะมีลักษณะดัง*รูปที่ 8.6* โดยแกน y คือค่า error ส่วนแกน x จะเป็นระยะทางระหว่างจุดสังเกตกับ plane จากกราฟจะเห็นได้ว่าที่ระยะทางน้อยกว่าศูนย์จะมีค่า error สูงมาก ที่ระยะทางเท่ากับศูนย์จะมี error เป็นศูนย์พอดี และ error จะค่อย ๆ เพิ่มช้า ๆ เมื่อระยะทางมากกว่าศูนย์



รูปที่ 8.6 กราฟ error function

การหาค่าเหมาะสุดของ plane จะหาได้จากการ minimizes negative loglikelihood $F(V)$ ของตำแหน่งจุดสังเกตทั้งหมด (P)

$$F(V) = \sum_{p_i \in P} e_i^T \Sigma_i^{-1} e_i$$

ในที่นี้ Σ_i^{-1} จะเป็นค่า weight ของ error function โดยในแต่ละจุดสังเกตจะกำหนดให้ค่าเป็น Identity ค่าประมาณ plane ที่เหมาะสมสุด (V^*) จะหาได้จาก

$$V^* = \underset{V}{\operatorname{argmin}}\, F(V)$$

การ minimizes ฟังกชัน $F(V)$ ข้างต้นสามารถหาได้จากการแก้ linear System

$$H\Delta x^* = -b$$

เมื่อ

$$H = \sum_{p_i \in P} J_i^T \Sigma_i^{-1} J_i$$

$$b = \sum_{p_i \in P} e_i^T \Sigma_i^{-1} J_i$$

โดยที่ J_i เป็น Jacobian Matrix ของ error function ของจุดสังเกตที่ i การแก้ linear System ข้างต้นสามารถแก้ได้ด้วยวิธี Gauss-Newton

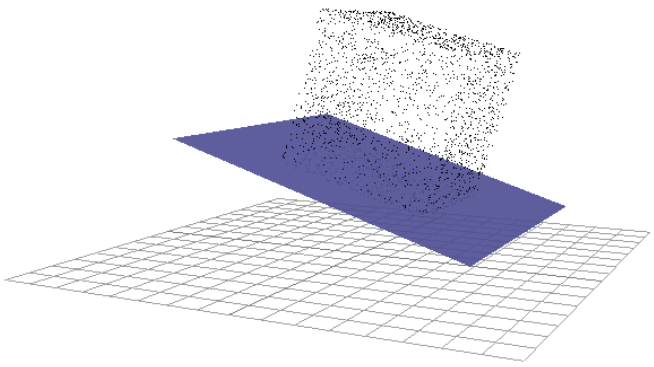
ทิศทางของ plane (n) ที่ได้จากการ optimization จะขึ้นอยู่กับค่า initial guess โดยในงานวิจัยนี้เราจะกำหนดทิศทางเริ่มต้นของ plane ให้มีทิศชี้ขึ้นเมื่อเทียบกับตำแหน่งกล้อง โดยมีสมมุติฐานว่ากล้องจะถ่ายภาพในลักษณะขนานกับพื้นโลกเสมอ นอกจากนี้เราจะกำหนด constraint เพิ่มเติมเพื่อบังคับไม่ให้ทิศทางของ plane ผิดเพี้ยนไปจากค่าเริ่มต้นมากเกินไป โดยการกำหนด constraint นั้นจะเป็นการเพิ่มเทอม error เข้าไปใน error function โดยเทอมที่เพิ่มขึ้นมาสามารถเขียนได้เป็น

$$e_{const} = \rho(1 - \langle n, n_0 \rangle)$$

Constraint error คำนวนได้จาก หนึ่งลบ dot product ของ plane normal (n) กับค่าทิศทาง initial guess (n_0) โดยมี ρ เป็นค่าคงที่สำหรับเพิ่มหรือลดน้ำหนักของ constraint จะเขียน error function ใหม่ได้เป็น

$$e_i = \left(k^{-f(p_i)} - 1\right)^2 + e_{const}$$

ตัวอย่างการทำงานของอัลกอริทึมการตรวจหาพื้นแสดงได้ดังรูปที่ 8.7 โดยตำแหน่งจุดสังเกตหาได้จากการสุ่มจุดรอบสี่เหลี่ยมลูกบาศก์และมีการเพิ่มความคลาดเคลื่อนเข้าไปในตำแหน่งจุดสังเกต เมื่อกำหนดให้ initial guess ของ plane เป็นทิศชี้ขึ้น อัลกอริทึมจะสามารถตรวจหา plane ได้โดยเลือกจาก plane ล่างสุดของลูกบาศก์



รูปที่ 8.7 ผลลัพธ์การตรวจหาพื้นจากกลุ่มจุดทรงสี่เหลี่ยมลูกบาศก์

ภาคผนวก ค
Jacobians สำหรับฟังก์ชันการวัด Visual Odometry

กำหนดให้ $x = \{x_0, x_1, \dots, x_k\}$ เป็นเซตของตำแหน่งของกล้องจำนวน k view ล่าสุด โดยมี Transformation Matrix เป็น $T(x_i) = [R_i|t_i]$ เมื่อ t_i เป็น translation vector และ R_i เป็น Rotation Matrix กำหนดให้ $d = \{d_0, d_1, \dots, d_m\}$ เป็นเซตของค่า invert depth ของจุดสังเกตทั้งหมด และ z_{ij} เป็นค่าการวัดของจุดสังเกตที่ j ในมุมมองของ view i ซึ่งมี reference view ที่ k

สำหรับ $x_i \in \mathrm{SE}(3)$ และ $d_j \in \mathbb{R}$ โมเดลการวัดสามารถเขียนได้เป็น

$$p_j = (r_j/d_j) \ominus x_k$$

$$\hat{z}_{ij} = z(x_i, d_j) = \frac{x_i \oplus p_j}{\|x_i \oplus p_j\|_2}$$

เมื่อ r_j เป็น unit vector แสดง direction ของจุดสังเกตใน reference view ที่ k

เราต้องการหาอนุพันธ์ของฟังก์ชัน $z(x_i \boxplus \Delta x_i, d_j + \Delta d_j)$ เทียบ Δx_i และ Δd_j ที่จุด $\Delta x_i = 0$ และ $\Delta d_j = 0$ ซึ่งสามารถหาได้จาก chain rule
นิยามให้ $x_i \boxplus \Delta x_i = e^{\Delta x_i} \oplus x_i$ และกำหนดให้ $A = e^{\Delta x_i} \oplus x_i \oplus p_j$ เราสามารถหาอนุพันธ์ของฟังก์ชันการวัดเทียบ Δx_i ได้จาก

$$\left.\frac{\partial z(x_i \boxplus \Delta x_i, d_j)}{\partial \Delta x_i}\right|_{\Delta x_i = 0} = \frac{1}{\|A\|}\frac{\partial \mathrm{A}}{\partial \Delta x_i} - \frac{A}{\|A\|^2}\frac{\partial \|A\|}{\partial \Delta x_i}$$

$$\frac{\partial \|A\|}{\partial \Delta x_i} = \frac{1}{\|A\|}\langle A, \frac{\partial A}{\partial \Delta x_i}\rangle$$

$$\frac{\partial \mathrm{A}}{\partial \Delta x_i} = \frac{\partial e^{\Delta x_i} \oplus x_i \oplus p_j}{\partial \Delta x_i} = \frac{\partial (e^{\Delta x_i}\, x_i) \oplus p_j}{\partial (e^{\Delta x_i}\, x_i)} \cdot \frac{\partial e^{\Delta x_i}\, x_i}{\partial \Delta x_i} = \left([p_j^T \;\; 1] \otimes I_3\right)\frac{\partial e^{\Delta x_i}\, x_i}{\partial \Delta x_i}$$

โดยที่

$$\frac{\partial e^{\Delta x_i}\, x_i}{\partial \Delta x_i} = \begin{bmatrix} 0_{3\times3} & -[R_{c1}]_\times \\ 0_{3\times3} & -[R_{c2}]_\times \\ 0_{3\times3} & -[R_{c2}]_\times \\ I_3 & -[t_i]_\times \end{bmatrix}$$

เมื่อ R_{c1}, R_{c2} และ R_{c2} เป็นค่าใน Rotation Matrix (R_i) คอลัมน์ที่ 1, 2 และ 3 ตามลำดับ

$\langle \cdot, \cdot \rangle$ เป็น inner product

$\otimes$ เป็น Kronecker operator หรือ matrix direct product

$[\cdot]_\times$ เป็น skew-symmetric matrix operator นิยามได้เป็น

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_\times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}$$

กำหนดให้ $A = x_i \oplus p'_j$ และ $p'_j = \left(r_j/(d_j + \Delta d_j)\right) \ominus x_k$ เราสามารถหาอนุพันธ์ของฟังก์ชันการวัดเทียบ Δd_j ได้จาก

$$\left.\frac{\partial z\left(x_i, d_j + \Delta d_j\right)}{\partial \Delta d_j}\right|_{\Delta d_j = 0} = \frac{1}{\|A\|}\frac{\partial \mathrm{A}}{\partial \Delta d_j} - \frac{A}{\|A\|^2}\frac{\partial \|A\|}{\partial \Delta d_j}$$

$$\frac{\partial \|A\|}{\partial \Delta d_j} = \frac{1}{\|A\|}\langle A, \frac{\partial A}{\partial \Delta d_j}\rangle$$

$$\frac{\partial A}{\partial \Delta d_j} = \frac{\partial x_i \oplus p'_j}{\partial \Delta d_j} = x_i \oplus \frac{\partial p'_j}{\partial \Delta d_j}$$

$$\frac{\partial p'_j}{\partial \Delta d_j} = -\frac{r_j}{d_j^2} \ominus x_k$$

ภาคผนวก ง
Jacobians สำหรับฟังก์ชันการแปลงตำแหน่งสัมพัทธ์

กำหนดให้ x_i เป็นตำแหน่งกล้องใน node พ่อ และ x_j เป็นตำแหน่งกล้องใน node ลูก บรรยายด้วย vector 7 มิติ $x_i = [t_i \;\; q_i]^T$ โดย t_i แทนการเลื่อนตำแหน่งและ q_i เป็น quaternion rotation โมเดลการวัด relative transformation ของ node j เทียบ node i หาได้จาก

$$\hat{u}_{ij}(x_i, x_j) = x_j \ominus x_i$$

เราสามารถเขียนโมเดลการวัดแบบแยก component ได้เป็น

$$\hat{u}_{ij}(x_i, x_j) = \begin{bmatrix} t \\ q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} = \begin{bmatrix} f_{qri}(q_i, t_j - t_i) \\ q_{wi}q_{wj} + q_{xi}q_{xj} + q_{yi}q_{yj} + q_{zi}q_{zj} \\ q_{wi}q_{xj} - q_{xi}q_{wj} - q_{yi}q_{zj} + q_{zi}q_{yj} \\ q_{wi}q_{yj} - q_{yi}q_{wj} - q_{zi}q_{xj} + q_{xi}q_{zj} \\ q_{wi}q_{zj} - q_{zi}q_{wj} - q_{xi}q_{yj} + q_{yi}q_{xj} \end{bmatrix}$$

เมื่อ

$$f_{qri}(q, a) = \begin{bmatrix} a_x + 2(-(q_y^2 + q_z^2)a_x + (q_x q_y + q_w q_z)a_y + (-q_w q_y + q_x q_z)a_z) \\ a_y + 2(\;(-q_w q_z + q_x q_y)a_x - (q_x^2 + q_z^2)a_y + (q_y q_z + q_w q_x)a_z) \\ a_y + 2(\;(q_x q_z + q_w q_y)a_x + (-q_w q_x + q_y q_z)a_y - (q_x^2 + q_y^2)a_z) \end{bmatrix}$$

อนุพันธ์ของฟังก์ชัน $u_{ij}(x_i, x_j)$ เทียบ x_j ได้เป็น

$$\frac{\partial u_{ij}(x_i, x_j)}{\partial x_j} = \left[\begin{array}{c|cccc} \dfrac{\partial f_{qri}(q_i, t_j - t_i)}{\partial t_j} & \multicolumn{4}{c}{0_{3\times 4}} \\ \hline 0_{4\times 3} & \begin{matrix} q_{wi} \\ -q_{xi} \\ -q_{yi} \\ -q_{zi} \end{matrix} & \begin{matrix} q_{xi} \\ q_{wi} \\ q_{zi} \\ q_{yi} \end{matrix} & \begin{matrix} q_{yi} \\ -q_{zi} \\ q_{wi} \\ -q_{xi} \end{matrix} & \begin{matrix} q_{zi} \\ -q_{yi} \\ q_{xi} \\ q_{wi} \end{matrix} \end{array}\right]$$

$$\frac{\partial f_{qri}(q_i, t_j - t_i)}{\partial t_j} = 2\begin{bmatrix} \frac{1}{2} - q_{yi}^2 - q_{zi}^2 & q_{xi}q_{yi} + q_{wi}q_{zi} & -q_{wi}q_{yi} + q_{xi}q_{zi} \\ -q_{wi}q_{zi} + q_{xi}q_{yi} & \frac{1}{2} - q_{xi}^2 - q_{zi}^2 & q_{yi}q_{zi} + q_{wi}q_{xi} \\ q_{xi}q_{zi} + q_{wi}q_{yi} & -q_{wi}q_{xi} + q_{yi}q_{zi} & \frac{1}{2} - q_{xi}^2 - q_{yi}^2 \end{bmatrix}$$

อนุพันธ์ของฟังก์ชัน $u_{ij}(x_i, x_j)$ เทียบ x_i ได้เป็น

$$\frac{\partial u_{ij}(x_i, x_j)}{\partial x_i} = \begin{bmatrix} \dfrac{\partial f_{qri}(q_i, t_j - t_i)}{\partial t_i} & \dfrac{\partial f_{qri}(q_i, t_j - t_i)}{\partial q_i} \\ 0_{4\times 3} & \begin{matrix} q_{wj} & q_{xj} & q_{yj} & q_{zj} \\ q_{xj} & -q_{wj} & -q_{zj} & q_{yj} \\ q_{yj} & q_{zj} & -q_{wj} & -q_{xj} \\ q_{zj} & -q_{yj} & q_{xj} & -q_{wj} \end{matrix} \end{bmatrix}$$

$$\frac{\partial f_{qri}(q_i, t_j - t_i)}{\partial t_i} = 2\begin{bmatrix} -\frac{1}{2} + q_{yi}^2 + q_{zi}^2 & -q_{xi}q_{yi} - q_{wi}q_{zi} & q_{wi}q_{yi} - q_{xi}q_{zi} \\ q_{wi}q_{zi} - q_{xi}q_{yi} & -\frac{1}{2} + q_{xi}^2 + q_{zi}^2 & -q_{yi}q_{zi} - q_{wi}q_{xi} \\ -q_{xi}q_{zi} - q_{wi}q_{yi} & q_{wi}q_{xi} - q_{yi}q_{zi} & -\frac{1}{2} + q_{xi}^2 + q_{yi}^2 \end{bmatrix}$$

$$\frac{\partial f_{qri}(q_i, a)}{\partial q_i} =$$

$$2\begin{bmatrix} q_{zi}a_y - q_{yi}a_z & -q_{yi}a_y - q_{zi}a_z & 2q_{yi}a_x - q_{xi}a_y + q_{wi}a_z & 2q_{zi}a_x - q_{wi}a_y - q_{xi}a_z \\ -q_{zi}a_x + q_{xi}a_z & -q_{yi}a_x + 2q_{xi}a_y - q_{wi}a_z & -q_{xi}a_x - q_{zi}a_z & q_{wi}a_x + 2q_{zi}a_y - q_{yi}a_z \\ q_{yi}a_x - q_{xi}a_y & -q_{zi}a_x + q_{wi}a_y + 2q_{xi}a_z & -q_{wi}a_x - q_{zi}a_y + 2q_{yi}a_z & -q_{xi}a_x - q_{yi}a_y \end{bmatrix}$$

ประวัติผู้เขียนวิทยานิพนธ์

นายยุทธนา สุทธสุภา เกิดเมื่อวันที่ 8 พฤษภาคม 2528 ที่จังหวัดเชียงใหม่ สำเร็จการศึกษาระดับปริญญาตรีวิศวกรรมศาสตรบัณฑิต (เกียรตินิยม อันดับหนึ่ง) สาขาวิศวกรรมคอมพิวเตอร์ จากคณะวิศวกรรมศาสตร์จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2549 สำเร็จการศึกษาระดับปริญญาโทวิศวกรรมศาสตรมหาบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์ จากคณะวิศวกรรมศาสตร์จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2551 และเข้าศึกษาในหลักสูตรวิศวกรรมศาสตรดุษฎีบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์ ณ ภาควิชาวิศวกรรมคอมพิวเตอร์คณะวิศวกรรมศาสตร์จุฬาลงกรณ์มหาวิทยาลัย เมื่อปีการศึกษา 2552 โดยได้รับทุนอุดหนุนการศึกษาระดับบัณฑิตศึกษา จุฬาลงกรณ์มหาวิทยาวัย เพื่อเฉลิมฉลองวโรกาสที่พระบาทสมเด็จพระเจ้าอยู่หัวทรงพระเจริญพระชนมายุครบ 72 พรรษา

มีความสนใจในงานวิจัยที่เกี่ยวข้องกับหุ่นยนต์และระบบอัตโนมัติ การประมวลเซนเซอร์หุ่นยนต์ และการระบุตำแหน่งพร้อมกับการสร้างแผนที่ของหุ่นยนต์