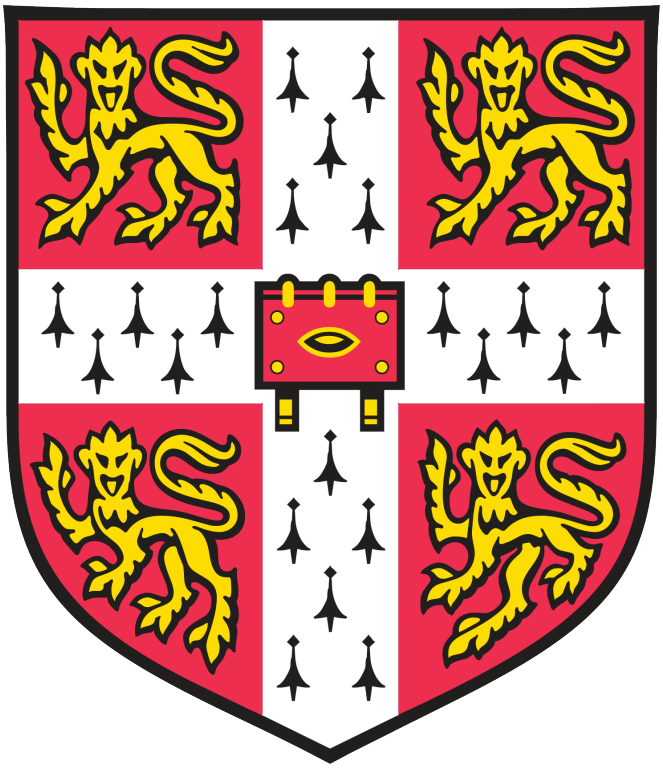
Joshua Bird

Distributed Visual Simultaneous Localization and Mapping



Computer Science Tripos – Part II
Queens' College

March 31, 2024

Declaration

I, Joshua Bird of Queens' College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose. In preparation of this dissertation I did not use text from AI-assisted platforms generating natural language answers to user queries, including but not limited to ChatGPT. I am content for my dissertation to be made available to the students and staff of the University.

Signed: Joshua Bird
Date: March 31, 2024

Proforma

Candidate number:	⟨CANDIDATE NUMBER⟩
Project Title:	**Distributed Visual Simultaneous Localization and Mapping**
Examination:	**Computer Science Tripos – Part II, 2024**
Word Count:	⟨WORD COUNT⟩[1]
Code Line Count:	⟨LINE COUNT⟩[2]
Project Originator:	Joshua Bird, Jan Blumenkamp
Project Supervisor:	Jan Blumenkamp

Original Aims of the Project

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Fusce ac turpis quis ligula lacinia aliquet. Mauris ipsum. Nulla metus metus, ullamcorper vel, tincidunt sed, euismod in, nibh.

Work Completed

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Fusce ac turpis quis ligula lacinia aliquet. Mauris ipsum. Nulla metus metus, ullamcorper vel, tincidunt sed, euismod in, nibh.

Special Difficulties

None.

[1] This word count was computed using `texcount`.

[2] This code line count was computed using `cloc` (excluding autogenerated test output).

Contents

Chapter 1

Introduction

Visual Simultaneous Localization and Mapping (visual SLAM) serves as the foundation of countless modern technologies, with self-driving cars, augmented reality devices and autonomous drones just being a few examples. By using purely visual inputs, visual SLAM is able to create a 3D map of the surroundings while also localizing the camera's position within this map in real time.

Unlike other SLAM systems which may use expensive and heavy sensor such as LIDAR, RGB-Depth cameras or Radar, visual SLAM only requires the ubiquitous camera sensor, allowing the technology to be used in many commercial applications such as Google's ARCore[1], Boston Dynamic's GraphNav system used on their robot Spot[2], and several models of DJI quadcopters[3] making this a particularly practical field of research.

1.1 Relevant Work

While there have been many advanced implementations of visual SLAM over the last decade, very few of them have focused on distributed multi-agent systems, instead focusing on single-agent or centralized multi-agent systems. ORB-SLAM3 [1] is perhaps the most popular single-agent SLAM system, and often ranks at the top of benchmarks in a variety of environments [2], however it is fundamentally a single-agent system with no considerations in place for collaboration among peers. Out of the few multi-agent systems that do exist, the majority of them require a centralized server to manage communications between the agents and perform map merging, such as CCM-SLAM(2019) [3] and COVINS(2021) [4] among many others.

todo: add comparison table?

1.2 Motivation

Multi-robot systems are only becoming more common as automation in numerous fields continues to grow, such as self-driving cars, drone swarms and warehouse robots. These systems require the agents to understand the world around them, as know each other's locations within that world for collision avoidance. This task is often achieved with technologies such as GPS or motion capture setups, however we can not assume that all environments will have access to these systems. A few emerging examples include:

[1] https://developers.google.com/ar/develop/fundamentals
[2] https://support.bostondynamics.com/s/article/GraphNav-Technical-Summary
[3] DOI: 10.1016/j.vrih.2019.09.002

- Search and rescue operations in large indoor systems, assisted by drone swarms.
- Self-driving cars in underground road networks.
- Multi-agent cave/subsea exploration.

These are scenarios where multi-agent SLAM is able to assist, as it enables us to build a map of an unknown environment and keep agents aware of their relative poses. However, as noted in section 1.1, the majority of existing multi-agent visual SLAM implementations are centralized systems, requiring the agents to maintain reliable communications with the central server in order to operate. This is somewhat counterintuitive, as environments which don't have access to GPS or motion capture systems are more than likely to also have very poor communication channels – greatly limiting the use cases of these centralized multi-agent SLAM systems.

Naturally, this leaves us with distributed multi-agent visual SLAM systems which do not rely on a centralized management server, allowing the agents to be used in environments where network infrastructure may be lacking. Instead of a central node, the agents are able to communicate peer-to-peer when they come within close proximity with one another.

It is easy to see the broad reaching uses cases this opens up. Agents will be able to explore sections of the world independently or in small teams, sharing new world locations with their peers as they come into communication range using an ad-hoc network. Agents will only know their peer's locations when they are within communication range, but this is sufficient if we only need the relative position for collision avoidance (as is common in multi-robot systems).

1.3 Project Overview

In this project, I:

1. Design and implement a distributed multi-agent visual SLAM system that is capable of localization, relative pose estimation and collaborative mapping, all while being tolerant to degraded network conditions and not reliant on any single leader agent.

2. Create a simulation environment for testing and evaluating my system locally.

3. Evaluate the performance of my system on standardized datasets, **demonstrating that its performace is superior to comparable state-of-the-art systems**.

4. Deploy my system on physical robots, demonstrating the practical use cases of this system and benchmarking real world performance.

5. Develop *Multi-Agent EVO* – an evaluation library for multi-agent SLAM systems based on *EVO* [7].

Chapter 2

Preparation

2.1 Starting Point

As noted in the *Relevant Work* section, visual SLAM systems are a mature and well-researched subfield of Computer Science with many advanced implementations. To avoid spending the majority of my effort re-implementing a visual SLAM system from scratch, I instead used a **single-agent** visual SLAM implementation as the starting point for my project. The thinking behind this decision was that it would allow me to focus my efforts on the distributed multi-agent aspect of this project, which I believe is a novel and under-researched aspect of the field.

I chose ORB-SLAM 3 as the single agent SLAM system to base my system on top of, as its researchers released code alongside their paper. I primarily utilized the system's visual odometry (VO) front end and helper functions from the backend to perform operations such as bundle adjustment.

However, as I will expand on in the *ORB-SLAM 3* section, a significant amount of time and effort was still required to understand its extremely large undocumented codebase, especially since an almost complete understanding of its inner workings were needed to both extract and inject map information from the system, as it was never designed to work in a multi-agent context. In retrospect, using an existing single-agent SLAM system as a foundation was not as much of a time-saver as I had initially hoped.

Nevertheless, using a cutting-edge single-agent SLAM system as a foundation for my project has allowed me to create a distributed SLAM system that is accurate and performant enough to have real-world use cases.

At the time of submitting my project proposal, I had forked the ORB-SLAM3 [1] git repository[1] and explored the codebase. ORB-SLAM3 is licensed under GPL-3.0, and as such, I have open-sourced my code under the same license[2].

I had no prior experience working with SLAM systems, but I had researched the current state of multi-agent visual SLAM systems to evaluate the feasibility of my project and to prevent it from being a duplication of prior work.

[1] `https://github.com/UZ-SLAMLab/ORB_SLAM3`

[2] `https://github.com/jyjblrd/part_II_project` TODO: Redact

2.2 Visual SLAM Background

Before developing a distributed multi-agent SLAM system, we must first understand the basics of a visual SLAM. This is a topic on which numerous books [5] and research papers [6] have discussed in depth, which I will attempt to summarize here.

2.2.1 Problem Statement

2.2.2 Visual Odometry

2.2.3 Feature Descriptors

2.2.4 Loop Closure

2.3 ORB-SLAM 3

ORB-SLAM 3 is a cutting-edge single-agent SLAM system, often ranking at the top of visual SLAM comparison papers. As noted in *Starting Point* section, I used ORB-SLAM as my starting point, primarily for its mature visual odometry (VO) front end and backend helper functions.

2.4 Development Tools & Frameworks

From the start, I knew that a well-structured development plan would be essential to the successful implementation of this project. An entire suite of infrastructure had to be implemented to aid the development of my distributed SLAM system, including:

- Simulation software
- Saving and loading test cases
- Motion control systems
- Evaluation libraries

2.4.1 Webots Simulation Software

Robotics projects work in the physical domain, however testing in the real world requires a large amount of setup and infrastructure. To ensure fast iteration, I decided to use simulations for the majority of my development, allowing me to easily test my system in various environments and scenarios before committing to deploying it on physical robots. Additionally, it allowed me to record numerous test cases which I used as regression tests and benchmarks for my system throughout development.

TODO: ...

2.4.2 Robot Operating System 2 Communication Middleware

Robot Operating System (ROS) 2 is the glue holding everything together, allowing independent software processes and hardware to communicate through an abstracted messaging interface.

ROS has long been the industry standard, being almost ubiquitous in both robotics research and the commercial sector. Confusingly, ROS is not an operating system at all, but instead a cross-platform development framework that provides a middleware to facilitate reliable communication between independent processes called *nodes*. These nodes can be on the same device or on a device within the local area network and may be written in C++ or Python. Nodes communicate by *publishing* and *subscribing* to different *topics*, allowing both peer-to-peer communication and broadcasting.

This is best illustrated with an example. Below is a toy distributed SLAM system. Given agents $\{\texttt{agent}_n \mid n \in \{1, 2\}\}$, each agent has a camera which publishes to the $\texttt{/agent}_n\texttt{/camera}$ topic. The $\texttt{SLAM\_Processor}_n$ node subscribes to the $\texttt{/agent}_n\texttt{/camera}$ topic, and performs simultaneous localization and mapping using the image stream. The $\texttt{SLAM\_Processor}_n$ node then publishes to the $\texttt{/agent}_n\texttt{/new\_map\_data}$ topic, which the other agent can subscribe to and use to improve their local map.

todo: add diagram

Since every node is abstracted away behind the interface provided by the various topics, we can easily swap out nodes in this system. For example, we can substitute the real camera for a simulated camera to test our system in a virtual environment without having to change any other part of our system. This makes transitioning between the real and simulated world almost seamless, which I knew would be essential for this project as I planned to test my system in simulations before running it on physical robots.

Furthermore, using the ROS framework allows my code to be far more portable, as anyone can download my nodes, link the camera topics up to their robot's camera, and run my SLAM system with minimal effort. This turns my project from simply being a nice codebase to something that anyone can take and run on their own robots.

There are two versions of ROS: ROS 1 and ROS 2. ROS 2 has slightly less software support than ROS 1, but I chose to use it due to its better decentralized properties, which align with the goals of this project. ROS 2 conforms to the Data Distribution Service (DDS)[1] specification, which guarantees a reliable broadcast and, unlike ROS 1, it does not require a leader node when used in a multi-agent setup.

[1] `https://en.wikipedia.org/wiki/Data_Distribution_Service`

2.4.3 Testing Infrastructure

2.4.4 Docker Containers

2.4.5 Visualization Tools

2.5 Datasets

2.6 Requirements Analysis

2.6.1 Development Model

2.6.2 Version Control and Testing

Chapter 3

Implementation

3.1 Architectural Overview

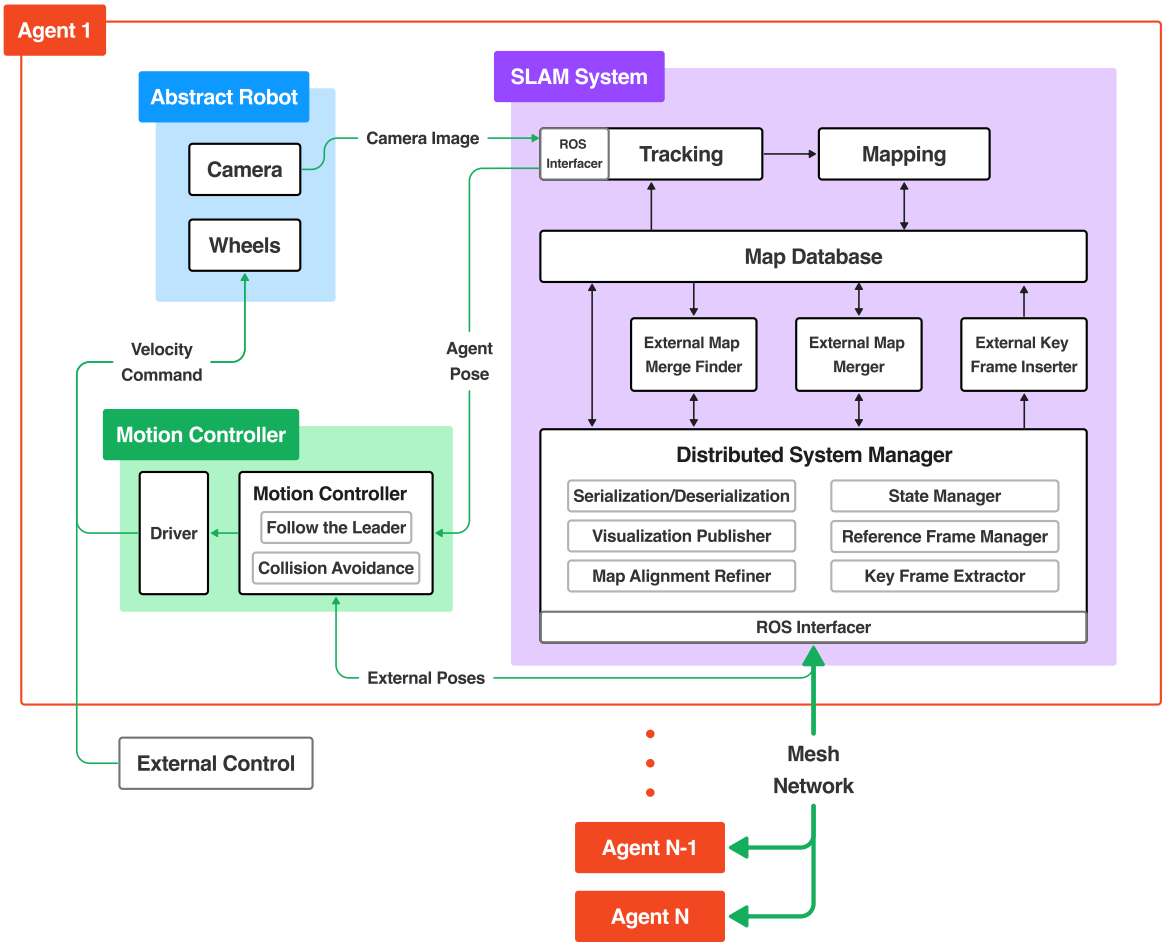


Figure 3.1: Agent diagram.

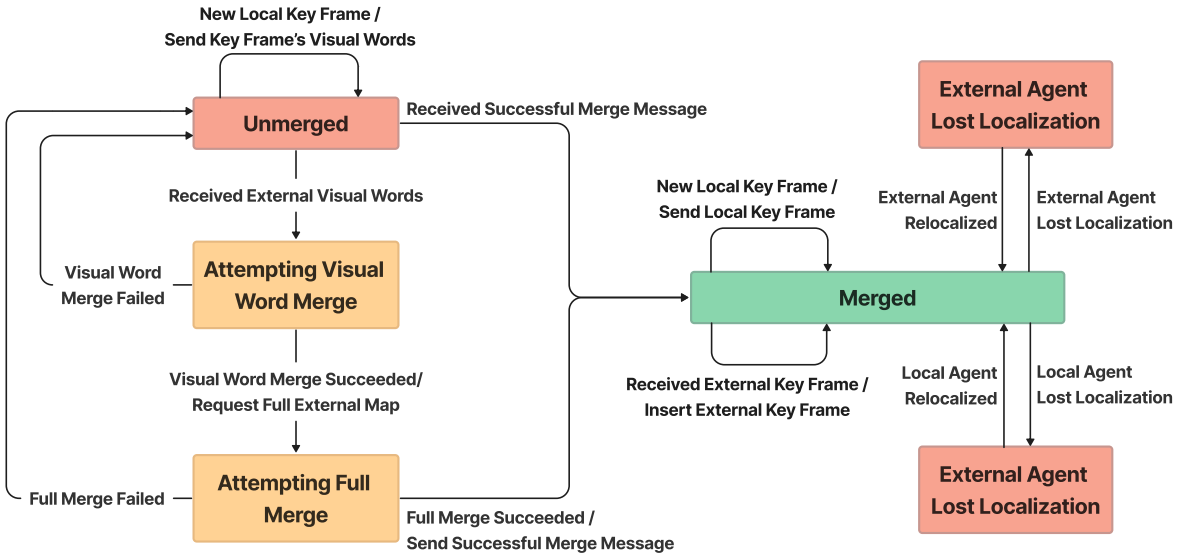


Figure 3.2: SLAM system state machine for a single peer.

3.1.1 Repository Overview

3.2 Simulation Environment

3.3 SLAM System

3.3.1 Distributed System Manager

3.3.1.1 State Manager

3.3.1.2 Map Alignment Refiner

3.3.1.3 Reference Frame Manager

3.3.1.4 Visualization Publisher

3.3.2 Map Serialization and Deserialization

3.3.3 External Map Merge Finder

3.3.4 External Map Merger

3.3.5 External Key Frame Inserter

3.3.6 Keeping Coordinate Frames Consistant

3.4 Motion Controller

3.4.1 Follow The Leader

3.4.2 Multi-Agent Collision Avoidance

an open-source multi-agent SLAM evaluation tool: *Multi Agent EVO*, based on the popular single agent SLAM evaluation tool *EVO* [7].

Besides the simple data structure and data ingestion modifications needed to allow EVO to process multi-agent SLAM data, there is some additional nuance to evaluating data from multiple agents.

Initially, all agents will be in separate reference frames until they explore an area previously seen by another agent, allowing them to merge their maps and share the same coordinate frame. We may also have cases where two independent groups of agents meet and merge maps, which requires multiple agents to simultaneously change coordinate frames. We also must note that these coordinate frames are part of the SIM(3) transformation group, which is composed of rotation, translation, and uniform scale in 3-dimensional space (scale being necessitated by the scale ambiguity of monocular visual SLAM).

Therefore, I have created a new data format to capture these changes in coordinate frames over time within our trajectory data, which Multi-Agent EVO is able to ingest. This allows us to properly compare the multi-agent SLAM trajectories to the ground truth data, giving us insights on how long it takes for agents to successfully merge maps, the accuracy of relative pose estimation, and much more.

TODO: add graph illustrating this coordinate frame stuff TODO: perhaps list out all capabilities added

3.6 Real World Implementation

Chapter 4

Evaluation

4.1 Review of Success Criteria

4.2 Benchmarking

4.3 Comparison to Related Work

4.4 Real World Experiments

Chapter 5

Conclusions

5.1 Lessons learned

5.2 Future Work

Bibliography

[1] Carlos Campos et al. "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM". In: *IEEE Transactions on Robotics* 37.6 (2021), pp. 1874–1890.

[2] Dinar Sharafutdinov et al. "Comparison of modern open-source visual SLAM approaches". In: *CoRR* abs/2108.01654 (2021). arXiv: 2108.01654. URL: https://arxiv.org/abs/2108.01654.

[3] Patrik Schmuck and Margarita Chli. "CCM-SLAM: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams". In: *Journal of Field Robotics* 36.4 (2019), pp. 763–781.

[4] Patrik Schmuck et al. *COVINS: Visual-Inertial SLAM for Centralized Collaboration*. 2021. arXiv: 2108.05756 [cs.RO].

[5] Xiang Gao and Tao Zhang. *Introduction to visual SLAM: from theory to practice*. Springer Nature, 2021.

[6] Hugh Durrant-Whyte and Tim Bailey. "Simultaneous localization and mapping: part I". In: *IEEE robotics & automation magazine* 13.2 (2006), pp. 99–110.

[7] Michael Grupp. *evo: Python package for the evaluation of odometry and SLAM.* https://github.com/MichaelGrupp/evo. 2017.

Appendix A

⟨APPENDIX A NAME⟩

Appendix A...

Appendix B

⟨APPENDIX B NAME⟩

Appendix B...

Appendix C

Proposal

Proposal...