REGISTRATION AND LOCALIZATION OF UNKNOWN MOVING OBJECTS IN MARKERLESS MONOCULAR SLAM

by

Blake Troutman

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

Department of Computer and Information Science

Indianapolis, Indiana

May 2023

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Mihran Tuceryan, Chair

Department of Computer and Information Science

Dr. Shiaofen Fang

Department of Computer and Information Science

Dr. Gavriil Tsechpenakis

Department of Computer and Information Science

Dr. Qin Hu

Department of Computer and Information Science

Approved by:

Dr. Shiaofen Fang

To my late father, Danny Lee Troutman

ACKNOWLEDGMENTS

As I reflect on my achievement of this tremendous milestone, I am overwhelmed with gratitude to all of the people in my life that have helped me get here. Thus, it is only fitting that I take a moment to acknowledge those for whom I am particularly thankful.

First, I would like to thank my advisor, Dr. Mihran Tuceryan, whose vast experience has provided me with access to an invaluable repository of knowledge and perspective on the field of computer vision, the value of which cannot be overstated. Additionally, both the support and freedom that Dr. Tuceryan has afforded me through his advisement over the past several years has allowed me to develop competency in my research area organically and efficiently; for this, I am certainly grateful. I would also like to thank Dr. Shiaofen Fang, Dr. Gavriil Tsechpenakis, and Dr. George Mohler for the constructive and insightful feedback they provided during my preliminary exam, which has helped me improve the quality of my work. I would also like to thank Dr. Qin Hu for graciously taking on the responsibility of joining my examining committee in Dr. Mohler's absence.

I certainly want to take a moment to also thank my parents, whose unyielding love and support throughout the years has not only helped shape me into the person I am today, but has also provided me with the opportunities and privileges necessary for me to get to this point in life. I would also like to thank my amazing wife for the loving companionship we share, which has given me the strength to persevere through my studies even in my research's most challenging moments. I also want to extend gratitude to my friends for staying in touch with me throughout my studies, despite my extremely scarce availability.

Finally, I would like to thank all of the staff and faculty of the Department of Computer and Information Science for providing students with all of the resources necessary to obtain a high-quality computer science education. In addition to the department, I would also like to broadly thank everyone involved at Indiana University-Purdue University Indianapolis, as IUPUI has been a beacon of opportunity, accessibility, and practicality in this day and age where the real-world costs of pursuing a college education is otherwise increasingly-difficult for the average person to justify.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

AR	augmented reality
VR	virtual reality
2D	2-dimensions/2-dimensional
3D	3-dimensions/3-dimensional
HMD	head-mounted display
SLAM	simultaneous localization and mapping
PnP	Perspective-n-Point
SfM	structure from motion
SVD	singular value decomposition
RANSAC	Random Sample Consensus
BA	bundle adjustment
ReLU	rectified linear units
CPU	central processing unit
GPU	graphics processing unit
6DoF	six degree-of-freedom
BoW	bag-of-words
ATE	absolute trajectory error
RMSE	root mean square error
7DoF	seven degrees of freedom

ABSTRACT

Simultaneous localization and mapping (SLAM) is a general device localization technique that uses realtime sensor measurements to develop a virtualization of the sensor's environment while also using this growing virtualization to determine the position and orientation of the sensor. This is useful for augmented reality (AR), in which a user looks through a head-mounted display (HMD) or viewfinder to see virtual components integrated into the real world. Visual SLAM (i.e., SLAM in which the sensor is an optical camera) is used in AR to determine the exact device/headset movement so that the virtual components can be accurately redrawn to the screen, matching the perceived motion of the world around the user as the user moves the device/headset. However, many potential AR applications may need access to more than device localization data in order to be useful; they may need to leverage environment data as well. Additionally, most SLAM solutions make the naive assumption that the environment surrounding the system is completely static (non-moving). Given these circumstances, it is clear that AR may benefit substantially from utilizing a SLAM solution that detects objects that move in the scene and ultimately provides localization data for each of these objects. This problem is known as the *dynamic* SLAM problem. Current attempts to address the dynamic SLAM problem often use machine learning to develop models that identify the parts of the camera image that belong to one of many classes of potentially-moving objects. The limitation with these approaches is that it is impractical to train models to identify every possible object that moves; additionally, some potentially-moving objects may be static in the scene, which these approaches often do not account for. Some other attempts to address the dynamic SLAM problem also localize the moving objects they detect, but these systems almost always rely on depth sensors or stereo camera configurations, which have significant limitations in real-world use cases. This dissertation presents a novel approach for registering and localizing unknown moving objects in the context of markerless, monocular, keyframe-based SLAM with no required prior information about object structure, appearance, or existence. This work also details a novel deep learning solution for determining SLAM map initialization suitability in structure-from-motion-based initialization approaches. This dissertation goes on to validate these approaches by implementing

them in a markerless, monocular SLAM system called LUMO-SLAM, which is built from the ground up to demonstrate this approach to unknown moving object registration and localization. Results are collected for the LUMO-SLAM system, which address the accuracy of its camera localization estimates, the accuracy of its moving object localization estimates, and the consistency with which it registers moving objects in the scene. These results show that this solution to the dynamic SLAM problem, though it does not act as a practical solution for all use cases, has an ability to accurately register and localize unknown moving objects in such a way that makes it useful for some applications of AR without thwarting the system's ability to also perform accurate camera localization.

1. INTRODUCTION

Augmented reality (AR) has been the focus of increasing interest in recent years due to both broad advancements in technology as well as the recent commercialization of virtual reality (VR). AR and VR are emerging interfaces that enable users to experience computing with a heightened sense of immersion. In VR, a user is completely immersed in a 3D virtual environment; whereas in AR, a user is able to perceive the real world with the addition of virtual elements integrated into their perception of the 3D space. The functionality afforded by integrating real and virtual worlds can be useful in basic applications, such as one that enables a user to cast multiple dynamic computer screens into their environment at will; or, it can be useful in applications that are significantly more complex, such as applications which provide precise, 3D instructions for performing realtime maintenance on a piece of complex machinery.

In order to implement any AR or VR system, the system developer must first implement a realtime approach for localizing a device, such as a head-mounted display (HMD). This is so that virtual objects can be rendered to the end-user's view in a way that makes them appear spatially-consistent with the real world, even as the device changes its viewpoint. Increasingly, various forms of visual simultaneous localization and mapping (SLAM) have been used to provide this underlying functionality in AR and VR systems. SLAM is a process that uses a sensor (such as a camera in the case of visual SLAM) to continually generate a virtual mapping of the environment while also using this map in conjunction with the sensor measurements to localize the sensor in realtime. Though many approaches to device localization may lend themselves well to VR, the potential interaction between real and virtual scene elements implied by AR makes SLAM particularly well-suited for device localization in this context, as many applications of AR may utilize SLAM's virtual mapping to support some degree of scene understanding.

SLAM, and in particular *visual* SLAM, has been studied extensively and its research has led to the development of several general solutions to the basic SLAM problem. At this point in time, it seems there is little new theoretical ground to be covered in regards to solving the basic SLAM problem; however, since the typical SLAM problem formulation

adopts the constraint that the scene elements are assumed to be still/static, this constraint can be relaxed to expose a new and useful derivative of the SLAM problem that provides expansive opportunities for fresh discovery. This expanded view of the SLAM problem (in which the scene may contain moving elements) is often referred to as the *dynamic* SLAM problem. Dynamic SLAM is particularly relevant to AR systems, as these systems may often operate in environments that contain moving elements such as people, vehicles, and any other moving object that is common in day-to-day life. This makes a basic SLAM solution a less-appropriate choice for facilitating an AR application than a dynamic SLAM solution. Additionally, a solution to the dynamic SLAM problem that can localize moving objects (rather than just registering them to make the system more robust) could facilitate a plethora of AR applications that have yet to be seen. For example, perhaps a developer wishes to implement an AR application in which an end-user can scan their environment, place virtual labels on some objects in the environment, and trigger the application to automatically translate the labels to a different language. This application could be useful for language-learning; however, its usability quickly collapses if a user moves some of the labeled objects in the room and the application fails to re-localize the objects' corresponding labels accordingly. A SLAM system that can register and localize moving objects can provide a layer of functionality that enables an application to overcome this limitation.

Though it's clear that this layer of functionality is necessary in order to implement many real-world AR applications, implementing a SLAM system that can provide this functionality is challenging. To solve this problem, one must propose a process for registering when certain map points belong to moving objects and one must also propose a process for continually re-localizing those points. In many works that address these problems, the object registration problem is solved by developing the system to search for specific *potentially*-moving objects based on object appearance, but this approach is not generalized for any object in the scene that might move. Additionally, though a solution to the object localization problem may be straightforward in theory, poor feature tracking and poor localization constraints can cause dynamic object localization to be troublesome in practice.

The primary focus of this dissertation is the proposal and exploration of a novel approach for registering and localizing unknown moving objects in the context of markerless, monoc-

ular SLAM. That is, the approach is a solution to the monocular (single-camera) SLAM problem which can detect the existence of moving objects in the scene with no prior knowledge of the objects' structure, appearance, or existence. This approach is far more general and less constrained than many existing related works, which often pre-train their systems to recognize specific types of objects that have a high likelihood of moving. The generalized nature of the approach presented in this dissertation is designed with the intention of enhancing the basic functionality of SLAM, as AR applications of the future will likely need to leverage more than just device localization data from the lower-level systems they are built on top of in order to achieve their desired level of functionality.

1.1 Contributions

This dissertation presents and details the following contributions to the field of computer vision (and more specifically, visual SLAM):

- A novel deep-learning model for determining the suitability of two image frames for use in structure-from-motion-based point cloud reconstruction, specifically in the context of visual SLAM map initialization (detailed in Chapter 4).
- A novel, geometric approach for registering when an unknown object in a scene has begun to move during the SLAM process, as well as a corresponding approach to re-localize the reconstruction of the object as it subsequently moves through the scene (detailed in Chapter 5).
- LUMO-SLAM: a proof-of-concept markerless, monocular SLAM system, designed and developed from the ground up, which implements the aforementioned approach for registering and localizing unknown moving objects in scene, in realtime, with no prior knowledge of the objects' structure, appearance, or existence required (detailed in Chapter 6). Additionally, source code for LUMO-SLAM is made openly available under the MIT license at: https://github.com/batroutman/LUMO-SLAM.

1.2 Assumptions and Constraints

Regarding the proposed unknown moving object registration and localization approach, the following assumptions are made to clearly constrain the problem domain:

- The camera's calibration parameters are given.
- The moving objects to-be-tracked are rigid and non-deformable.
- These moving objects are feature-rich and yield many easily-trackable points.
- The SLAM system that implements the approach can observe the moving objects from multiple angles *before* the objects begin to move.
- The initial movement of the objects is observed by the implementing SLAM system.

1.3 Peer-Reviewed Publications

1.3.1 Published Works

The contributions discussed in this dissertation cover research that has also been previously published in [1], [2], [3], and [4]:

[1] B. Troutman and M. Tuceryan, "Towards fast and automatic map initialization for monocular SLAM systems," in *Proceedings of the 2nd International Conference on Robotics, Computer Vision and Intelligent Systems - ROBOVIS*, INSTICC, SciTePress, 2021, pp. 22–30, ISBN: 978-989-758-537-1. DOI: 10.5220/0010640600003061.

[2] B. Troutman and M. Tuceryan, "Rapid structure from motion frame selection for markerless monocular SLAM," in *Robotics, Computer Vision and Intelligent Systems*, P. Galambos, E. Kayacan, and K. Madani, Eds., Cham: Springer International Publishing, 2022, pp. 172–189, ISBN: 978-3-031-19650-8. DOI: 10.1007/978-3-031-19650-8_9.

[3] B. Troutman and M. Tuceryan, "Registration and localization of unknown moving objects in monocular SLAM," in *2022 IEEE 2nd International Conference on Intelligent Reality (ICIR)*, 2022, pp. 43–48. DOI: 10.1109/ICIR55739.2022.00025.

[4] B. Troutman and M. Tuceryan, "Towards dynamic realtime object labeling in augmented reality," in *2022 IEEE 2nd International Conference on Intelligent Reality (ICIR)*, 2022, pp. 49–53. DOI: 10.1109/ICIR55739.2022.00026.

1.3.2 Other Works

This dissertation also presents new system details and results that have yet to be published. Accordingly, an article covering these details and results is being developed for publication to act as a canonical reference for the LUMO-SLAM system.

1.4 Dissertation Organization

This dissertation is organized into eight chapters. This first chapter has motivated the dynamic SLAM problem and provided a brief specification of the proposed contributions while also indicating the assumptions and constraints of the primary problem addressed in this research (unknown moving object registration and localization). Chapter 2 provides an overview of many of the existing works in both the *visual* SLAM literature and the *dynamic* SLAM literature, including brief explorations of the notable techniques used in these existing systems. This is followed by Chapter 3, which provides an explanation of the theoretical concepts necessary to implement modern, keyframe-based visual SLAM systems.

After these preliminary chapters, Chapter 4 covers the first contribution proposed in this dissertation: a deep-learning approach for determining initialization suitability in markerless, monocular SLAM. It is worth noting, however, that this specific contribution veers slightly from the focus of the rest of this body of work, as this dissertation is primarily focused on the registration and localization of unknown moving objects. The contribution discussed in Chapter 4 ultimately ties back into the rest of the work when it is contextualized in the implementation of LUMO-SLAM, discussed in Chapter 6.

Once the contribution regarding initialization suitability is covered, Chapter 5 explains the contribution of unknown moving object registration and localization in a theoretical context. Then, Chapter 6 provides a detailed system overview of the proof-of-concept SLAM system that implements the approaches from Chapter 4 and Chapter 5. This system, LUMO-SLAM, acts as the principal contribution of this body of work. Chapter 7 then provides

extensive experimental results and analysis of LUMO-SLAM's performance with regards to standard SLAM accuracy, moving object localization accuracy, and moving object registration accuracy. Finally, Chapter 8 concludes this dissertation and discusses notable areas for future work.

2. LITERATURE REVIEW

2.1 Visual SLAM

Many solutions to the visual SLAM problem have emerged over the past several decades. There exist a number of strong SLAM implementations that leverage stereo (binocular) cameras [5]–[7], as well as implementations that leverage RGB-D (depth) cameras [8]–[11]. However, both of these sensor configurations fundamentally come with limitations. For instance, stereo camera configurations often require a significant baseline in order to leverage the stereo nature of the cameras; this large baseline requirement may disqualify stereo configurations from applications in which the physical size of the system must remain small. Additionally, the technology that facilitates RGB-D cameras typically struggles in outdoor settings that contain high amounts of natural light. Thus, a solution to the *monocular* (single-camera) SLAM problem can be viewed as more desirable than solutions to the stereo and RGB-D variants of the problem, as monocular systems have potential to be more versatile. However, the monocular SLAM problem can also be more difficult to solve, as these systems must solve the SLAM problem with less data than the aforementioned approaches.

Early solutions to monocular SLAM primarily relied on filtering approaches, in which probabilistic models are used in conjunction with each incoming frame measurement to estimate the pose of the camera and location of the landmarks, or "map points", in realtime [12]–[15]. However, these classical filtering approaches have largely been replaced in the visual SLAM literature by various forms of pose graph optimization. These approaches are typically known as "keyframe-based" approaches, as savepoints of camera poses ("keyframes") are periodically recorded in the map along with their feature observations to repeatedly refine the map point and keyframe estimates in the map. Among the most notable keyframe-based monocular SLAM systems are PTAM [16], [17] and ORB-SLAM [18].

Parallel Tracking and Mapping, or "PTAM", is a seminal monocular SLAM system developed by Klein and Murray in 2007 [16]. The architectural novelty of PTAM is that it organizes the process of map generation and refinement into a separate thread from the camera localization process. It also employs bundle adjustment to achieve extremely accurate localization, which is significant for AR as slight errors in camera localization can become

noticeable when virtual objects are overlayed onto the scene. The approach demonstrated in PTAM is also particularly interesting due to Klein and Murray's successful mobile adaptation of the system, in which PTAM was able to run in realtime on an Apple iPhone 3G, circa 2009 [17]. The accuracy and realtime capability of PTAM (in addition to its openly-available source code) has made it a foundational system in the visual SLAM literature.

A more recent visual SLAM system that has proven to be foundational in the literature is ORB-SLAM, developed by Mur-Artal et al. in 2015 [18]. ORB-SLAM is an openly-available monocular SLAM system (with later iterations adding stereo and RGB-D support [19]) that has grown increasingly-popular over the past several years. In addition to incredibly-accurate realtime camera localization, ORB-SLAM also implements a number of useful features. Among these features include a novel (and reliable) automatic map initialization algorithm, place recognition, and loop closing. These features make ORB-SLAM incredibly versatile; ORB-SLAM's versatility, accuracy, and openly-available source code are responsible for cultivating the substantial researcher interest around the system.

Though there are a number of other notable monocular SLAM systems in the literature (see [20], [21], [22], [23], and [24]), PTAM and ORB-SLAM are perhaps the most notable keyframe-based approaches. LUMO-SLAM draws inspiration from PTAM and ORB-SLAM for the implementation of its basic SLAM pipeline. Specifically, the separation of map refinement and localization threads, the usage of ORB-SLAM's robust map initialization criteria, the extensive use of bundle adjustment for map refinement, and the utilization of visual bag-of-words vectors for loop detection are all features that follow in the footsteps of PTAM and ORB-SLAM. However, despite drawing inspiration from these systems, LUMO-SLAM's pipeline is substantially different than either of these foundational systems, as LUMO-SLAM is adapted to also address the *dynamic* SLAM problem.

2.2 Dynamic SLAM

Currently, the basic visual SLAM problem is mostly a problem of engineering and system optimization, as it seems there is little new theoretical ground to cover in this area while there is also great utility in making SLAM implementations both more robust in real-world

scenarios and also less computationally expensive. The pursuit of higher robustness has helped motivate a derivative of the SLAM problem: the *dynamic* SLAM problem.

The domain of dynamic SLAM is quite broad, as it largely includes any SLAM solution that is designed to address scenes with non-static components. This basic subset of SLAM is important, as work from Fuentes-Pacheco et al. [25] indicates that typical SLAM approaches tend to struggle in environments that contain many moving objects. The current attempts at solving the dynamic SLAM problem fall into two categories: (1) systems that address moving objects to *add robustness* to a standard SLAM implementation, and (2) systems that address moving objects in order to *extend* the standard SLAM solution to provide additional functionality related to moving objects (typically in the form of object localization data) [26].

Dynamic SLAM implementations that address moving objects solely to increase the robustness of the system's localization accuracy accomplish this by implementing some form of *motion segmentation* (the process of separating features associated with static objects from features associated with moving objects), and then using the results of the motion segmentation to ignore non-static features. This task has frequently been implemented with semantic segmentation networks which classify each pixel into a number of categories corresponding to pre-trained object types. For example, DynaSLAM [27] makes use of Mask R-CNN [28] to identify pixels in the image that are associated with potentially-moving objects and then uses this segmentation information for static map reconstruction and background inpainting. Similarly, DS-SLAM [29] and SOF-SLAM [30] make use of SegNet [31] instead of Mask R-CNN for this motion segmentation process while the work of Brasch et al. [32] employs ICNet [33]. PSPNet-SLAM [34] and the system proposed by Han and Xi [35] segment moving objects with a pyramid scene parsing network (PSPNet) [36] instead of the aforementioned semantic segmentation models. In addition to semantic segmentation networks, object detection networks have also been used to perform motion segmentation. Object detection networks are similar to semantic segmentation networks; except, object detection networks typically identify bounding boxes for instances of a class, rather than classifying each pixel. Examples of object detectors being used for motion segmentation in SLAM include the work

of Ai et al. [37] which uses YOLOv4 [38], Dynamic-SLAM [39] which uses SSD [40], and PLD-SLAM [41] which uses MobileNets [42].

The aforementioned approaches only make use of motion segmentation to increase the robustness of the system localization estimates. There are other approaches, however, which continue to utilize semantic segmentation networks and object detection networks for motion segmentation, but also provide some degree of moving object tracking. Some examples include ClusterVO [43] which uses YOLO9000 [44] for motion segmentation, DynaSLAM II [45] which builds off of its predecessor (DynaSLAM, mentioned above), DOT [46] which uses Detectron2 [47], VDO-SLAM [48] which uses Mask R-CNN [28], and EM-Fusion [49] which also uses Mask R-CNN. Each of these approaches, however, utilize RGB-D or stereo camera configurations. Thus, the object localization is somewhat straightforward as depth information is available.[1]

A couple of monocular approaches that still utilize supervised models for motion segmentation include CubeSLAM [51] (leveraging YOLO9000 [44]) and the work of Nair et al. [52] (leveraging Mask R-CNN). These approaches can estimate the depth/scale of the moving object by assuming that it is close to the ground plane. This enables these systems to contextualize the points of the moving object against static points with known locations in the map. However, the ground plane assumption limits these implementations by making them only capable of tracking objects that are on the ground plane.

Systems that perform motion segmentation with semantic segmentation networks or object detection networks are very common in the domain of dynamic SLAM; however, the usage of trained models comes with a steep limitation. For any of these systems, they can only detect moving objects that their respective models have been trained to identify. This poses a practical challenge, as the models cannot be trained to identify every possible object in practice. Additionally, these models identify specific objects, but do not analyze their movement. This can cause inefficiency in the system, as static objects that *might* move (such as a still chair) may be detected as a potentially-moving object and its features will be ignored, even if the object is not moving and could otherwise be used for camera localization.

[1]↑Though VDO-SLAM [48] supports monocular camera configurations, it requires depth information to be predicted with MonoDepth2 [50].

There are a couple of notable systems that manage to provide dynamic SLAM solutions without requiring *a priori* knowledge of the moving objects via supervised models. Specifically, DymSLAM [53] performs motion segmentation by computing permutation preferences [54] for the quantized residuals of feature matches and then uses these metrics to cluster features by the motion models they belong to. Additionally, Judd et al. [55] presented MVO, which is a multimotion visual odometry approach that clusters "tracklets" based on their reprojection errors. Though both of these approaches are more generalized than systems that implement motion segmentation with supervised models, these specific implementations also require stereo/RGB-D configurations.

The work of Migliore et al. [56] and that of Hsiao and Wang [57] both propose monocular SLAM systems that identify and estimate moving object trajectories. Both of these systems are based on filtering approaches and attempt to use various filtering techniques to localize individual map points. This limits the accuracy of the moving object localization significantly. Though modern SLAM literature has mostly moved from filtering approaches to keyframe-based implementations, these works are still notable because they are among the earliest solutions to dynamic SLAM while also providing moving object localization (in addition to constraining their systems to the monocular configuration).

For further exploration of dynamic SLAM literature, Saputra et al. [58] provide a comprehensive survey of the growing body of literature surrounding visual SLAM (and also structure from motion) in dynamic environments. Additionally, Xu et al. [26] offer a similar overview of the current efforts to solve the dynamic SLAM problem, with particular focus on feature choice.

The existing literature on dynamic SLAM often presents systems that solve the motion segmentation problem with supervised models and provide moving object localization data by leveraging depth information. There are very few works, however, that aim to provide monocular dynamic SLAM solutions that do not depend on prior information about the nature of the moving objects. Though the principal contribution of this dissertation, LUMO-SLAM, is not a "silver bullet" solution to the dynamic SLAM problem, as it leaves great room for improvements in robustness, it fills this void in the literature by implementing a more-generalized approach to unknown moving object registration and localization in a

modern, markerless, monocular context with no required prior knowledge of object structure, appearance, or existence.

3. THEORETICAL BACKGROUND

Visual SLAM is a complex process that requires the implementation of several algorithms to work together in order to provide realtime camera localization and scene mapping. The algorithms, as well as the underlying mathematical framework supporting them, are somewhat specialized; thus this chapter provides brief summaries of the flagship topics that are necessary to understand in order to develop visual SLAM systems (in particular, LUMO-SLAM, which is discussed extensively in Chapter 6).

First, this chapter covers the SE(3) Lie group and how it is used to model points and transformations in 3D space. Then, the SE(3) Lie group is used to present the pinhole camera model, which models the perspective-based projections of 3D points into images. After these theoretical pillars are covered, the topic of image features is explored, as the observation of image features provides the first direct link between the theoretical models and the real world. This is followed by a section that covers a number of increasingly-popular approaches for performing structure from motion with image features, which is a common approach used to initialize the map in markerless, monocular systems. The exploration of structure from motion approaches is followed by a section on point triangulation, which is necessary for map growth. After these topics, the Perspective-n-Point problem and its subsequent solution (EPnP) is covered, as EPnP is the backbone of camera localization in many systems, including LUMO-SLAM. Finally, this chapter is concluded with a brief explanation of bundle adjustment, which is an optimization process used to refine visual SLAM maps.

3.1 Geometric Transformations in 3D with the SE(3) Lie Group

The goal of visual SLAM is to determine a set of 3D points that are observed in the environment while also deducing the transformation (or "pose") of the camera as it moves through the environment in realtime. SE(3) is an important topic in visual SLAM, and in computer vision in general, as it provides a model for representing transformations and their effect on 3D points in a scene.

A point, (x, y, z), can be transformed by representing the point as a homogeneous column vector, $\mathbf{X} = [\ x\ y\ z\ 1\ ]^\top$. With homogeneous vectors, vectors that only differ in scale are considered equivalent. The point can then be transformed (rotated and translated in space) by premultiplying $\mathbf{X}$ with a 4×4 SE(3) transformation matrix, where

$$SE(3) = \left\{ \mathbf{T} \;\middle|\; \mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}, \mathbf{R} \in SO(3), \mathbf{t} \in \mathbb{R}^3 \right\} \tag{3.1}$$

and

$$SO(3) = \left\{ \mathbf{R} \;\middle|\; \mathbf{R} \in \mathbb{R}^{3\times 3}, \mathbf{R}^\top \mathbf{R} = \mathbf{R}\mathbf{R}^\top = \mathbf{I}, |\mathbf{R}| = 1 \right\} \tag{3.2}$$

.

Specifically, $\mathbf{R}$ is a 3×3 matrix in the SO(3) Lie group which represents the rotation to be applied to the current coordinate frame and $\mathbf{t}$ is a 3×1 matrix representing the desired spatial translation of the current coordinate frame (with respect to the given rotation, $\mathbf{R}$).

Thus, if $\mathbf{T} \in SE(3)$ with rotation $\mathbf{R}$ and translation $\mathbf{t}$, $\mathbf{X}$ can be transformed by $\mathbf{R}$ and $\mathbf{t}$ with the product $\mathbf{TX}$. This results in a new column vector, $[\ x^*\ y^*\ z^*\ 1\ ]^\top$, where the newly-transformed point coordinates are given by (x^*, y^*, z^*).

Transformations in SE(3) can also be composed of multiple sub-transformations, such that

$$\mathbf{X}^* = \mathbf{T}\mathbf{X} \qquad s.t. \quad \mathbf{T} = \mathbf{T}_n \dots \mathbf{T}_1 \mathbf{T}_0 \tag{3.3}$$

where $\mathbf{X}^*$ is the transformed point and $\mathbf{T}_\mathrm{i}$ is the ith transformation applied to the point (note that the order of transformations starts with the rightmost transformations and ends with the leftmost transformations). The chaining of multiple transformations is useful for decomposing the movement of specific objects into object movement and overall scene/camera movement. This is explored further in Chapter 5. For a comprehensive review of the SE(3) Lie group, see [59].

3.2 The Pinhole Camera Model

The framework demonstrated in Section 3.1 can be extended to additionally model the projection of 3D points into a standard image using the pinhole camera model. The pinhole camera model relates a point in the 3D scene with its perspective-projected image coordinates with the following equation:

$$\mathbf{x} = \mathbf{K}\mathbf{X} \tag{3.4}$$

.

In this model, the 3D point is represented with the column vector, $\mathbf{X} = [\ x\ y\ z\ ]^\top$, $\mathbf{K}$ is the intrinsic camera parameter matrix (sometimes referred to as the "calibration matrix"), and the projection of the 3D point in the camera is derived from the resulting homogeneous column vector $\mathbf{x} = [\ u\ v\ w\ ]^\top$, where the image coordinates are given by $(u/w, v/w)$.[1]

Specifically, the intrinsic camera parameter matrix, $\mathbf{K}$, takes the form

$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

where f_x and f_y are the theoretical x- and y-scaled focal lengths for the camera[2], c_x and c_y are the image pixel coordinates of the principal point of the image (which typically coincides with the center of the image), and s is a special-case skew parameter, which almost always takes a value of 0.

Though Equation 3.4 models the projection of a 3D point into an image with a given focal length and principal point, it assumes that the camera is oriented at the origin with no rotation. To account for the camera taking on a different viewing direction and position, Equation 3.4 is expanded with

[1]↑Note, the pinhole camera model assumes the image uses coordinate axes such that x increases to the right and y increases downward, with $(0, 0)$ representing the top-left corner of the image.

[2]↑Theoretically, the focal length need only be represented by a single parameter, f. However, in the case of real-world CCD cameras (which often use sensors with pixels that are slightly non-square), two different focal lengths are needed for each direction in order to accurately model the projection.

$$\mathbf{x} = \mathbf{KEX} \qquad \text{where} \qquad \mathbf{E} = [\ \mathbf{R}\ \mathbf{t}\ ] \tag{3.6}$$

where $\mathbf{X}$ now takes the form of a homogeneous column vector, $[\ x\ y\ z\ 1\ ]^\top$, and $\mathbf{R}$ and $\mathbf{t}$ take the form of a rotation matrix and translation vector, as described in Section 3.1. In this form, $\mathbf{E}$ indicates the transformation that must be applied to all scene points in order to simulate the scenario of a camera that is *not* positioned at the origin while exhibiting no rotation. Thus, $\mathbf{E}$ actually indicates the *opposite* of the camera localization parameters; however, this distinction is seldom relevant in solving the SLAM problem. Thus, $\mathbf{E}$ is viewed as the representation of the camera's localization parameters and is often referred to as the "extrinsic camera parameters" or the "camera pose". This model is useful, as continuous estimation of the extrinsic camera parameters is one of the primary goals of the SLAM process.

3.3 Salient Feature Extraction, Description, and Matching

As many of the theoretical methods used in the visual SLAM process rely on 2D point projections as input, image feature extraction acts as the foundation that makes these theoretical methods applicable to the real world. Feature collection involves identifying salient, easily-trackable patches (or "features") of an image and providing an identification process for these features such that each feature is distinct from other features in the image, but similar to the corresponding feature in a different image that views the same scene. A pair of corresponding features are both, presumably, projections of the same 3D scene point. This assumption provides a basis for the methods used for triangulation and camera localization in the visual SLAM process.

Strong feature locations in an image can be deduced in many ways, but one of the fastest standard approaches is the FAST feature detection algorithm [60]. Assuming a greyscale image, the FAST algorithm classifies an image patch as a salient feature by comparing the intensity of the patch's center pixel with the intensities of the pixels that make up a small, 7×7 circle around the center pixel. In the 7×7 patch configuration, the patch is classified as a salient feature if 12 contiguous pixels along the circle are either all brighter or all

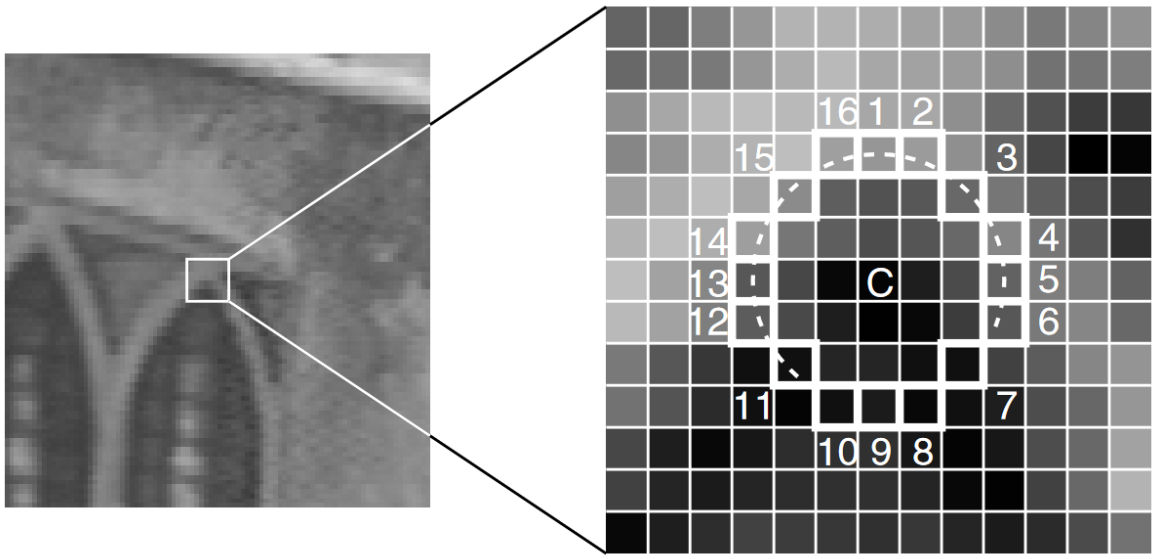


Figure 3.1. Example of the FAST algorithm's analysis of an image patch using circle of diameter 7, first presented in the original FAST paper, [60]. Note that the patch is classified as a salient feature because the 12 pixels that intersect with the dotted line contain higher intensity values than the center pixel, C.

darker than the center pixel (given some additional threshold to account for noise). Many implementations also use an adaptation of this approach by using a 9×9 image patch while requiring 16 contiguous pixels in order to determine salience. Figure 3.1 (forwarded directly from [60]) illustrates how the FAST algorithm is applied to an arbitrary 7×7 image patch.

The FAST feature detector is able to scan through an image and provide a useful collection of feature locations quickly, but each feature also needs to be prescribed an identifier, or "descriptor", so it can be recognized and properly associated with the corresponding feature in other images that view the same scene. Many specifications exist for feature description, including foundational descriptors such as SIFT [61] or SURF [62]. However, these standard approaches are known for relatively slow matching speed; thus, other descriptor specifications have emerged specifically to rectify this shortcoming. One of the most significant alternatives to SIFT and SURF is BRIEF [63].

BRIEF speeds up the matching process by specifying the feature descriptor as a binary vector. This is useful because even brute-force matching of binary descriptors can be performed quickly by simply computing the Hamming distance between pairs of descriptors, which can be done rapidly on most hardware as this is equivalent to getting the sum of the XOR values between bit pairs. The individual features of the binary vector correspond to preset pixel-pair comparisons within the image patch. For each feature's preset pixel pair, (p_1, p_2), if $I(p_1) < I(p_2)$ (where $I(p)$ is the image intensity at pixel p), then the feature value is set to 1; otherwise, it is set to 0. The preset pixel pairings for each feature are arbitrarily determined by sampling from an isotropic Gaussian distribution, further specified in [63].

Though BRIEF provides a strong framework for fast-matching descriptors, its matching recall struggles when there is rotation disparity between a pair of corresponding features. ORB [64], on the other hand, is a binary feature descriptor that adds rotation invariance to the BRIEF framework. It achieves this by computing an angle for each FAST feature with the intensity centroid [65] of the feature's image patch. Specifically, the angle, θ, of a feature is computed with

$$\theta = \text{arctan2}(m_{01}, m_{10}) \tag{3.7}$$

where m_{pq} are the moments of the patch, defined by

$$m_{pq} = \sum_{x,y} x^p y^q I(x, y) \tag{3.8}$$

.

The preset pixel pairs are then stored in lookup tables, where each table acts as an atlas for the pixel pairings under a specific rotation. Specifically, ORB precomputes a lookup table for every 12 degrees. In addition to rotation invariance, ORB also provides a more robust set of pixel pairings than initially proposed by BRIEF; this was achieved by identifying the 256 uncorrelated pairings with the highest variance from a large training set. For more details, see [64].

ORB, acting as a FAST detector with a rotated BRIEF descriptor, yields strong matching performance while also maintaining low computation cost. Consequently, it is used heavily

in systems such as ORB-SLAM [18] as well as LUMO-SLAM, which is discussed in detail in Chapter 6.

3.4 Map Initialization with Structure from Motion

In a monocular SLAM system, the map can be continuously grown by triangulating 3D locations for feature matches across pairs of previously-observed image frames. However, before additional points can be triangulated into the map, the camera poses associated with each prospective point must be either known or estimated by the system. Yet, estimation of the camera poses is typically carried out with a PnP algorithm [66], which requires a set of 3D-to-2D correspondences. With these approaches typically used for map growth, the generation of new points requires the estimation of camera poses while the estimation of camera poses requires the estimation of point locations. This presents a challenge for markerless systems, in which no reference points are given to provide a set of starting points for the map to build onto. This challenge motivates the process of map initialization, which aims to estimate an initial set of camera poses and 3D points to represent the map without relying on known marker information.

Map initialization for monocular SLAM systems is particularly challenging, as a single frame does not contain sufficient information to infer the 3D localization data for its image features. Thus, it has become common for monocular SLAM systems to perform map initialization by silently observing many frames as the user begins to move the camera, and then identifying a pair of these frames to use for feature matching and structure from motion (SfM). SfM is a process which uses a set of 2D-to-2D image feature correspondences from a pair of images in order to deduce the difference between their camera poses. Once the difference between the camera poses is estimated, the first camera can be oriented at the origin and the second camera can be localized with the estimate provided from the SfM process. From this point, the camera poses can be used in conjunction with their 2D-to-2D feature correspondences to triangulate each feature's respective 3D point location using the triangulation approach described in 3.5.

Popular SfM algorithms tend to compute either the essential matrix, the fundamental matrix, or the homography matrix in order to then solve for the SE(3) transformation between the viewing cameras. These matrices and their relationships to the SfM process are described in the following sections.

3.4.1 The Essential Matrix

The essential matrix [67] is a real, 3×3, non-zero matrix that relates a pair of camera poses with

$$\boldsymbol{E} = [\mathbf{t}]_{\times}\mathbf{R} \tag{3.9}$$

where $\boldsymbol{E}$ is the essential matrix and $\mathbf{R}$ and $\mathbf{t}$ are the rotation and translation components of the pose difference between the cameras.[3] For further specification, $[\mathbf{t}]_{\times}$, is the skew-symmetric matrix representation that would be used to perform a vector cross product with $\mathbf{t}$. This takes the form

$$[\mathbf{t}]_{\times} = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \tag{3.10}$$

where t_x, t_y, and t_z are the components of $\mathbf{t}$. In addition to this formulation, the essential matrix is also acts as a representation of epipolar geometry, which is further discussed in Section 3.4.2.

The essential matrix is particularly useful in the SfM process because, if known, the essential matrix can be decomposed into the components of the camera pose difference, $\mathbf{R}$ and $\mathbf{t}$, at an arbitrary baseline scale. For the purpose of map initialization, the deduction of $\mathbf{R}$ and $\mathbf{t}$ can be viewed as the ultimate goal of the SfM process. The decomposition of $\boldsymbol{E}$ into $\mathbf{R}$ and $\mathbf{t}$ is performed by using the singular value decomposition (SVD) of $\boldsymbol{E}$ in conjunction with the following standard theorem [68]:

[3]↑Note that the essential matrix, $\boldsymbol{E}$, is distinct from the extrinsic camera parameter matrix, $\mathbf{E}$.

Theorem 3.4.1. *Let the singular value decomposition of* $\boldsymbol{E}$ *be* $\mathbf{U}\text{diag}(1,1,0)\mathbf{V}^\top$. *The four possible factorizations of* $\boldsymbol{E} = [\mathbf{t}]_\times\mathbf{R}$ *are*

$$[\mathbf{t}]_\times = \mathbf{U}\mathbf{Z}\mathbf{U}^\top or\ \mathbf{U}\mathbf{Z}^\top\mathbf{U}^\top, \quad \mathbf{R} = \mathbf{U}\mathbf{W}\mathbf{V}^\top or\ \mathbf{U}\mathbf{W}^\top\mathbf{V}^\top \tag{3.11}$$

where

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{Z} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.12}$$

Theorem 3.4.1 results in two values for $\mathbf{t}$ and two values for $\mathbf{R}$. The four possible $(\mathbf{R}, \mathbf{t})$ pairings each provide a hypothesis for the true pose difference between the cameras. The correct pose can then be selected by evaluating the cheirality constraint[4] [69] for each pose hypothesis.

To select the correct pose difference among the four hypotheses, each pose is paired with the default pose, $[\ \mathbf{I}\ \mathbf{0}\ ]$, which represents a translation at the origin with no rotation. The default pose assumes the role of the *primary* camera (associated with the first image) and the hypothesis pose assumes the role of the *secondary* camera (associated with the second image). Given this pairing of camera poses, a single, accurate 2D-to-2D feature correspondence can then be triangulated with the approach described in Section 3.5. The resulting 3D point can be used to evaluate the cheirality constraint, as the triangulated point will only appear in front of *both* cameras when the correct pose is used.

The technical details of the cheirality check are fairly straightforward. Given the triangulated point, $\mathbf{X} = [\ x\ y\ z\ w\ ]^\top$, the point is in front of the primary camera (with pose $[\ \mathbf{I}\ \mathbf{0}\ ]$) if $zw > 0$ and the point is also in front of the secondary camera (with pose hypothesis $[\ \mathbf{R}\ \mathbf{t}\ ]$) if $z'w > 0$, where z' is the third component of $[\ \mathbf{R}\ \mathbf{t}\ ]\mathbf{X}$. Once the cheirality check exposes the correct pose hypothesis, the first camera can be associated with the default pose while the second camera can be associated with the correct pose hypothesis. At this point,

[4] "Cheirality" is the constraint that dictates that points viewed by a camera must be located *in front* of the camera in the 3D space.

the goal of the SfM process has been achieved, as the pose difference between the cameras, [$\mathbf{R}$ $\mathbf{t}$], has been successfully deduced.

Though this process illuminates the utility of the essential matrix and how it can be used to deduce the pose difference between a pair of cameras, it does not address how the essential matrix is obtained without prior knowledge of this pose difference. However, the essential matrix for a pair of images can, in fact, be estimated without prior knowledge of the pose difference by leveraging a set of 2D-to-2D feature correspondences from the two images. This can be accomplished with one of a number of algorithms, the most popular of which may be Nistér's five-point algorithm [70].

The five-point algorithm is a relatively sophisticated algorithm which uses at least five 2D-to-2D feature correspondences from the image pair to estimate the essential matrix. It does this by generating an $n \times 9$ matrix (where n is the number of correspondences), which is composed of the components of the correspondences. The algorithm then requires deducing four vectors which span the right nullspace of this matrix. Nistér asserts that the components of the essential matrix are a linear combination of these vectors, and thus the remainder of the algorithm requires solving for the coefficients of this linear combination. For brevity, full details of the algorithm, including details on efficient implementation of its steps, are left to Nistér's paper [70].

Though the five-point algorithm is robust, it tends to be more computationally expensive than other approaches. Another (and generally much faster) way to estimate the essential matrix is to first estimate the *fundamental* matrix with the eight-point algorithm and then convert the fundamental matrix to the essential matrix by using the theoretical relationship between the two matrices. This process is described in Section 3.4.2.

3.4.2 The Fundamental Matrix

The fundamental matrix is an analog of the essential matrix, related by the equation

$$\boldsymbol{E} = \mathbf{K}^{\top}\mathbf{F}\mathbf{K} \tag{3.13}$$

where $\boldsymbol{E}$ is the essential matrix, $\mathbf{F}$ is the fundamental matrix, and $\mathbf{K}$ is the intrinsic camera calibration matrix.[5] The fundamental matrix also expresses an important constraint derived from epipolar geometry. Epipolar geometry refers to the relationship between 3D points and the constraints on their projections in a given pair of stereo images. The fundamental matrix encodes this epipolar constraint with the following equation:

$$\mathbf{x'}^{\top}\mathbf{F}\mathbf{x} = 0 \tag{3.14}$$

where $\mathbf{F}$ is the fundamental matrix and $\mathbf{x}$ and $\mathbf{x'}$ are homogeneous column vectors of the image keypoint locations for a single 3D point in the first image and second image, respectively.

The epipolar constraint from Equation 3.14 can be used to estimate the fundamental matrix with a set of eight or more 2D-to-2D image correspondences from a given pair of stereo images. This approach is aptly named the "eight-point algorithm."

The eight-point algorithm is a simple alternative to the five-point algorithm, as it functions by reformulating the constraint from Equation 3.14 into the following flattened form:

$$\begin{bmatrix} xx' & yx' & x' & xy' & yy' & y' & x & y & 1 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_8 \end{bmatrix} = 0 \tag{3.15}$$

where $f_0 \dots f_8$ are the components of the fundamental matrix (in row-major order), and (x, y) and (x', y') represent the image keypoint locations of a single 3D point in the first and second images, respectively. The leftmost matrix can be expanded vertically with the keypoint locations of seven or more additional points to yield an $n \times 9$ matrix, where n is the number of points used. It then follows that the components of the fundamental matrix coincide with the right nullspace of this matrix, which is also the singular vector corresponding with the smallest singular value of this matrix (i.e., the last column of $\mathbf{V}$ if the SVD of this matrix is $\mathbf{UDV}^{\top}$), as the fundamental matrix is defined up to an arbitrary

[5] In this context, it is assumed that both of the cameras use the same intrinsic camera calibration matrix, $\mathbf{K}$.

scale. Deeper exploration of both the fundamental matrix and the eight-point algorithm is provided in [68].

Upon estimation of the fundamental matrix, the known intrinsic camera calibration matrix, $\mathbf{K}$, can be used with Equation 3.13 to deduce the essential matrix and recover the camera pose difference with the process described in Section 3.4.1. It is also worth noting, for implementation purposes, that a set of scene points that all lie on one of a few degenerate ruled quadrics (namely, a plane), the eight-point algorithm is ill-suited to accurately estimate the fundamental matrix. Thus, in some SLAM systems [18], the estimation of the fundamental matrix is paired with the estimation of a homography, which suffers from complementary degenerate cases. This is further explored in the next section.

3.4.3 The Homography Matrix

Another matrix that is useful for the SfM process is the homography matrix. Where the essential and fundamental matrices are representations of epipolar geometry, the homography matrix is instead a homogeneous 3×3 matrix that represents a planar, projective transformation. That is, it can describe how points on some plane are projected onto another plane with

$$\mathbf{x}' = \mathbf{H}\mathbf{x} \tag{3.16}$$

where $\mathbf{H}$ is the homography matrix, $\mathbf{x}$ is a homogeneous column vector representing the 2D plane coordinates for a point on some plane, and $\mathbf{x}'$ is a homogeneous column vector representing the 2D plane coordinates of the same point as it is projected onto another plane. With this relation, $\mathbf{x}$ and $\mathbf{x}'$ can also be seen as image keypoint locations for a 2D-to-2D correspondence, much like in Equation 3.14.

As homographies only define *planar*, projective transformations, homographies can only be used to relate sets of 2D-to-2D correspondences that map to *coplanar* points in 3D. This is complementary to the degenerate case of the eight-point algorithm, which requires the 3D points to be *non-coplanar*. Thus, SLAM systems may employ both fundamental matrix

estimation as well as homography estimation during map initialization to provide a higher degree of robustness.

Like the other SfM approaches, the homography matrix can be estimated from a small set of 2D-to-2D feature correspondences from a pair of images by using the four-point algorithm. The four-point algorithm is a low-cost, straightforward algorithm that requires at least four feature correspondences to deduce the homography matrix. This is achieved by using the constraint in Equation 3.16 to deduce that $\mathbf{x}'$ and $\mathbf{Hx}$ are parallel, and thus

$$\mathbf{x}' \times \mathbf{Hx} = \mathbf{0} \tag{3.17}$$

where $\times$ is the cross product of the two vectors. When computed with the matrix multiplication formulation of the cross product, $[\mathbf{x}']_\times$, the components of $\mathbf{H}$ can be factored out in the form of

$$\begin{bmatrix} 0 & 0 & 0 & -x & -y & -1 & y'x & y'y & y' \\ x & y & 1 & 0 & 0 & 0 & -x'x & -x'y & -x' \\ -y'x & -y'y & -y' & x'x & x'y & x' & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_8 \end{bmatrix} = \mathbf{0} \tag{3.18}$$

where $h_0, h_1, \ldots h_8$ are the components of $\mathbf{H}$ in row-major order, and (x, y) and (x', y') are the image coordinates for a correspondence in the first image and second image, respectively. For implementation purposes, the third row of the leftmost matrix is typically omitted, as it is not linearly independent. Thus, the formulation becomes

$$\begin{bmatrix} 0 & 0 & 0 & -x & -y & -1 & y'x & y'y & y' \\ x & y & 1 & 0 & 0 & 0 & -x'x & -x'y & -x' \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_8 \end{bmatrix} = \mathbf{0} \tag{3.19}$$

.

The leftmost matrix in Equation 3.19 can then be expanded vertically with at least three additional 2D-to-2D correspondences, resulting in an $n \times 9$ matrix, where n is twice the number of correspondences used. Since $\mathbf{H}$ is only determined up to an arbitrary non-zero

scale factor, the right nullspace of this $n \times 9$ matrix coincides with the unit singular vector corresponding to its smallest singular value. This vector, as described in Section 3.4.2, is the last column of $\mathbf{V}$, given the SVD of the matrix is $\mathbf{UDV}^\top$. Thus, the components of this vector provide the solution for the components of the homography matrix, $\mathbf{H}$.

There are a number of popular approaches that recover the parameters of the camera pose difference, $\mathbf{R}$ and $\mathbf{t}$, from the estimated homography matrix, including [71], [72], and [73]. The approach proposed in [73] is common, as it is implemented in the popular OpenCV computer vision library [74]. The approach ultimately manages to estimate two hypotheses for both $\mathbf{R}$ and $\mathbf{t}$, in the form of $\mathbf{R}_a, \mathbf{R}_b, \mathbf{t}_a$, and $\mathbf{t}_b$. The final pose hypotheses are then grouped as $[\ \mathbf{R}_a\ \ \mathbf{t}_a\ ]$, $[\ \mathbf{R}_b\ \ \mathbf{t}_b\ ]$, $[\ \mathbf{R}_a\ \ -\mathbf{t}_a\ ]$, and $[\ \mathbf{R}_b\ \ -\mathbf{t}_b\ ]$. Much like in the case of essential matrix decomposition, the correct pose hypothesis is selected by evaluating the cheirality constraint, as discussed in Section 3.4.1. As exact implementation details of this decomposition process are extensive, the remaining technical details of the approach are left in [73].

3.5 Point Triangulation

Point triangulation is the process of estimating the location of a 3D point by using two or more images that contain the point's projections. It is an important component of the SLAM process, as it is used to continually grow the system's map and can even be used to facilitate the initial generation of the map. To triangulate a point into 3D space, the intrinsic camera parameters, $\mathbf{K}$, are assumed to be known as well as the poses of two cameras viewing the point, $[\ \mathbf{R}\ \mathbf{t}\ ]$ and $[\ \mathbf{R}'\ \mathbf{t}'\ ]$, and the point's projections in each image, $\mathbf{x}$ and $\mathbf{x}'$.

To formulate the process for point triangulation, recall the standard pinhole camera model,

$$\mathbf{x} = \mathbf{KEX} \tag{3.20}$$

where $\mathbf{X}$ is a homogeneous column vector of some point in 3D space, $\mathbf{E}$ is the pose of the camera ($[\ \mathbf{R}\ \mathbf{t}\ ]$), $\mathbf{K}$ is the intrinsic camera parameter matrix, and $\mathbf{x}$ is a homogeneous

column vector representing the projection of $\mathbf{X}$ into the image. For simplicity, $\mathbf{K}$ and $\mathbf{E}$ are combined into a single matrix, $\mathbf{P}$, yielding

$$\mathbf{x} = \mathbf{PX} \tag{3.21}$$

.

Since homogeneous matrices are used, Equation 3.21 indicates that $\mathbf{x}$ and $\mathbf{PX}$ are parallel, and thus $\mathbf{x} \times \mathbf{PX} = 0$. This constraint can be also be written as $[\mathbf{x}]_\times \mathbf{PX}$, where

$$[\mathbf{x}]_\times = \begin{bmatrix} 0 & -1 & y \\ 1 & 0 & -x \\ -y & x & 0 \end{bmatrix} \tag{3.22}$$

where (x, y) is the observed image keypoint location for $\mathbf{X}$ through the camera associated with $\mathbf{P}$. Since the goal of triangulation is to solve for $\mathbf{X}$, $[\mathbf{x}]_\times \mathbf{PX}$ can be viewed as an $\mathbf{AX} = \mathbf{0}$ problem, where

$$\mathbf{A} = [\mathbf{x}]_\times \mathbf{P} = \begin{bmatrix} x\mathbf{p}_2^\top - \mathbf{p}_0^\top \\ y\mathbf{p}_2^\top - \mathbf{p}_1^\top \\ x\mathbf{p}_1^\top - y\mathbf{p}_0^\top \end{bmatrix} \tag{3.23}$$

and $\mathbf{p}_\mathrm{i}^\top$ is the ith row of $\mathbf{P}$ such that $\mathbf{P} = [\ \mathbf{p}_0^\top\ \mathbf{p}_1^\top\ \mathbf{p}_2^\top\ ]^\top$. In practice, the third row of $\mathbf{A}$ is typically ignored as it is not linearly independent.

To triangulate the point $\mathbf{X}$, this constraint can be formulated for two separate cameras, $\mathbf{P}$ and $\mathbf{P}'$, that each view the point from a different position. The resulting value for $\mathbf{A}$ becomes

$$\mathbf{A} = \begin{bmatrix} x\mathbf{p}_2^\top - \mathbf{p}_0^\top \\ y\mathbf{p}_2^\top - \mathbf{p}_1^\top \\ x'\mathbf{p}'_2{}^\top - \mathbf{p}'_0{}^\top \\ y'\mathbf{p}'_2{}^\top - \mathbf{p}'_1{}^\top \end{bmatrix} \tag{3.24}$$

where (x, y) are the image coordinates resulting from camera matrix $\mathbf{P}$ (the first camera) and (x', y') are the image coordinates resulting from camera matrix $\mathbf{P}'$ (the second camera). The right nullspace of $\mathbf{A}$ coincides with the homogeneous solution for $\mathbf{X}$. Thus, as discussed in Section 3.4.1 and Section 3.4.3, this nullspace can be deduced directly as it coincides with the last column of $\mathbf{V}$, given that the SVD of $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$. However, the last column of $\mathbf{V}$ is equivalent to $[\ xw\ yw\ zw\ w\ ]^\top$, where (x, y, z) is the 3D location of $\mathbf{X}$. Consequently, this column vector must be normalized by its w component to recover the true values for the 3D point location, (x, y, z).

3.6 Camera Localization with the Efficient Perspective-n-Point Algorithm

One of the primary goals of visual SLAM is to estimate the pose of the camera on every frame. This localization task is often framed as a Perspective-n-Point (PnP) problem, in which a set of 3D points and their observed 2D projections in a single image are provided and the goal is to use these 2D-to-3D correspondences to estimate the camera pose. There are many proposed solutions to this problem, the earliest of which include [75], [76], [77], [78], and [79]; however, a particularly efficient solution to the PnP problem is the Efficient PnP (EPnP) algorithm proposed by Lepetit, et al. [80].

The EPnP algorithm is an efficient, non-iterative solution to the PnP problem that has $O(n)$ complexity with respect to the number of correspondences. It has also become a standard approach, as it is implemented in the OpenCV computer vision library. To briefly summarize the algorithm, the novelty of the approach is that it represents each given 3D point as a weighted sum of four control points, such that

$$\mathbf{p}_{\mathrm{i}}^{w} = \sum_{\mathrm{j}=1}^{4} \alpha_{\mathrm{ij}} \mathbf{c}_{\mathrm{j}}^{w}, \qquad \sum_{\mathrm{j}=1}^{4} \alpha_{\mathrm{ij}} = 1, \qquad \mathrm{i} = 1, \ldots, n \tag{3.25}$$

where $\mathbf{p}_{\mathrm{i}}^{w}$ is the i^{th} 3D reference point provided in the world-coordinate frame, $\mathbf{c}_{\mathrm{j}}^{w}$ is the j^{th} control point in the world-coordinate frame, and α_{ij} make up the homogeneous barycentric coordinates for point $\mathbf{p}_{\mathrm{i}}$. The control points are selected such that the centroid of the reference points acts as one of the control points and three vectors which form a basis along the principal directions of the set of reference points act as the remaining control points.

With known control points, all values for α_{ij} can be estimated trivially using the known values of $\mathbf{p}_i^w$.

This formulation is useful, as the same property is upheld in the camera-coordinate frame:

$$\mathbf{p}_i^c = \sum_{j=1}^{4} \alpha_{ij} \mathbf{c}_j^c \tag{3.26}$$

.

In this equation, $\mathbf{p}^c$ and $\mathbf{c}^c$ represent a reference point and a control point, respectively, in the *camera* coordinate space. The next step of the approach is to then solve for the control points in the camera-coordinate frame, as the camera pose can be solved in many ways if a set of points is known in both the world- and camera-coordinate frames.

Substituting Equation 3.26 into the standard pinhole model presented in Section 3.2 yields the following two constraints:

$$\sum_{j=1}^{4} \alpha_{ij} f_x x_j^c + \alpha_{ij}(c_x - u_i) z_j^c = 0 \tag{3.27}$$

and

$$\sum_{j=1}^{4} \alpha_{ij} f_y y_j^c + \alpha_{ij}(c_y - v_i) z_j^c = 0 \tag{3.28}$$

where control point j in the camera-coordinate frame takes the form (x_j^c, y_j^c, z_j^c) and, in this context, $\mathbf{x}$ is assumed to take the form $[\ u\ v\ 1\ ]^\top$, where the scalar, w, is already factored out. Thus, (u_i, v_i) represents the observed image projection coordinates for point i.

These constraints can then be used to to formulate a $2n \times 12$ matrix, $\mathbf{M}$, such that $\mathbf{Mx} = \mathbf{0}$, where $\mathbf{x} = [\ \mathbf{c}_1^{c\top}\ \mathbf{c}_2^{c\top}\ \mathbf{c}_3^{c\top}\ \mathbf{c}_4^{c\top}\ ]$. Though there are many ways to estimate $\mathbf{M}$, Lepetit, et al. propose that an efficient approach is to represent the vector, $\mathbf{x}$, as a weighted sum of the columns of the right-singular vectors of $\mathbf{M}$ that correspond to the null singular values of $\mathbf{M}$. For more extensive implementation details, see [80]. At this point, the control points are known in both the world-coordinate frame and the camera-coordinate frame; thus, computing the transformation between them (which coincides with the camera pose) can be computed both directly and quickly with one of a number of standard approaches.

3.6.1 Outlier Robustness with Random Sample Consensus

Random Sample Consensus (RANSAC) [81] is a generalized framework used for performing robust model fitting. Specifically, it enables model fitting algorithms to identify gross outliers in their input data and exclude them from the model estimation. The simplicity of RANSAC makes it easily applicable to a wide variety of algorithms, including EPnP.

When applied to EPnP, RANSAC requires the random selection of some subset of the 2D-to-3D correspondences. This subset is then fed as input to the EPnP algorithm and the camera pose is estimated from these correspondences. Next, the resulting camera pose is used to compute the reprojection error for each correspondence. Correspondences which are outliers will trend towards *high* reprojection errors, while correspondences that are inliers will trend towards *low* reprojection errors. Consequently, an error threshold is set ahead of time to discriminate between presumed inliers and outliers. Correspondences with sufficiently high reprojection errors are omitted from further calculation (and denoted as outliers), and the remaining low-error correspondences are fed as input to the EPnP algorithm again. This process iterates until either a set number of iterations takes place or until all input points are viewed as inliers for the resulting camera pose.

3.7 Bundle Adjustment

A set of 2D-to-3D correspondences from a single camera frame is often sufficient to estimate a usefully-accurate pose for the camera and a single 2D-to-2D feature correspondence between two cameras (with known poses) may be sufficient to triangulate the correspondence's 3D point location; however, the accuracy of these estimates is often limited, as the estimates are developed with somewhat narrow constraints and are thus particularly affected by noise in the data. This limited accuracy becomes troublesome as new points are triangulated with increasingly-inaccurate camera poses and new camera poses are, in turn, estimated with increasingly-inaccurate point locations. To improve these estimates, a subset of the estimated camera poses and all triangulated points are stored in the map and refined with a process known as "bundle adjustment."

Bundle adjustment, in short, is a non-linear least-squares minimization of the reprojection errors of a set of feature observations; this optimizes the camera poses and 3D point locations associated with these measurements. The objective function for this problem takes the form

$$\min_{\mathbf{C}_i, \mathbf{X}_j \forall (i,j)} \sum_{i=1}^{m} \sum_{j=1}^{n} \left\| \mathbf{x}_{ij} - f(\mathbf{C}_i, \mathbf{X}_j) \right\|^2 \tag{3.29}$$

where $\mathbf{C}_i$ and $\mathbf{X}_j$ are vectors that denote the parameters of camera pose i and 3D point j, respectively, given m camera poses and n triangulated points, $\mathbf{x}_{ij}$ is the observed image keypoint coordinates of point j in camera i, and $f(\mathbf{C}_i, \mathbf{X}_j)$ is the pinhole projection function that predicts the keypoint location of point j in camera i.

Since the pinhole camera model uses homogeneous coordinates, the formulation of the reprojection error is non-linear. Thus, the Levenberg-Marquardt algorithm [82] is commonly used to perform this minimization by computing the adjustment vector, δ, for all values being optimized, $\{\mathbf{C}, \mathbf{X}\}$, with the equation

$$(\mathbf{J}^\top \mathbf{J} + \lambda \mathbf{I})\delta = \mathbf{J}^\top (\mathbf{x} - f(\mathbf{C}, \mathbf{X})) \tag{3.30}$$

where $\mathbf{J}$ is the Jacobian matrix containing the gradients of pinhole projection function, f, with respect to the camera poses and point locations, $\{\mathbf{C}, \mathbf{X}\}$, and λ is an adjustable damping factor used to aid in optimization. The Jacobian's columns represent the optimization parameters (camera poses and point locations) while its rows represent the the image keypoints observed. This typically results in a large, but sparse, Jacobian; consequently, this sparsity is typically exploited to accelerate the bundle adjustment process. For more implementation details, see [83].

When performed on an entire map containing thousands of triangulated points and hundreds of camera poses, bundle adjustment is far from a realtime process. However, bundle adjustment is often applied to subsets of a SLAM system's map in order to achieve a useful degree of refinement in realtime or near-realtime. This variant of bundle adjustment is referred to as "*windowed* bundle adjustment" and it is leveraged extensively in LUMO-SLAM, which is further discussed in Chapter 6.

4. INITIALIZATION SUITABILITY IN MARKERLESS MONOCULAR SLAM

In order for a visual SLAM system to continually localize a camera in its environment, the system must have access to a virtual mapping of the corresponding environment to use as a reference. In markerless SLAM systems, this mapping does not exist prior to the system's runtime. Instead, the system's environment map (or at least a subset of it) must be generated programmatically at the beginning of the system's execution. This programmatic map generation process is also known as *map initialization*, as SLAM systems still continue to build and refine their maps after this initial subset of the map is generated.

Map initialization is a relatively easy problem to solve in the context of RGB-D and stereo SLAM systems, as these modalities of visual SLAM are provided with enough data to initialize the map in a single frame. RGB-D SLAM systems utilize both color imagery and depth maps, which can be used to back-project image features into the 3D space directly. Stereo SLAM systems make use of two independent cameras, each of which view the same scene at slightly different positions. In this case, each iteration of the SLAM loop has access to two frames which can be analyzed for feature matches; this enables the system to initialize the map in a single loop iteration with one of the SfM approaches shown in Section 3.4.

Monocular SLAM systems, however, pose an additional challenge to the map initialization problem in that they do not have access to depth data or synchronous stereoscopic imagery. To solve this problem, keyframe-based monocular SLAM systems will often make use of SfM approaches (much like stereo SLAM systems) by recording multiple image frames over time and isolating a pair with a sufficiently large baseline. Assuming that sufficiently few scene objects have moved during this initial camera movement, this image pair can then be used to facilitate map initialization with SfM techniques, as seen in stereo SLAM systems.

Though this map initialization approach can successfully generate the initial subset of the map in monocular systems, it also presents a unique challenge to overcome if the system is to be functional in a real-world setting. The challenge posed by this approach is the problem of *frame selection*.

Frame selection is the problem of selecting two image frames that are well-conditioned for map initialization via SfM techniques. For example, a pair of frames are *ill*-conditioned for map initialization if there is an insufficiently-small ratio between the baseline of the cameras and the average Euclidean distance between the observed points and cameras. Though the average parallax of the triangulated map points (with respect to the cameras) can be used as an analog for this metric, it is impossible to compute this value without first generating the map. Additionally, for reliable accuracy, some SfM approaches require the scene points to be mostly *coplanar* (Section 3.4.3) and other SfM approaches require the scene points to be mostly *non-coplanar* (Section 3.4.2).

The fact that the evaluation of these conditions is interdependent with map existence is troublesome for two reasons: (1) if the image pairs are ill-conditioned, the resulting map may be too unreliable to facilitate an accurate evaluation of its own reliability, and (2) attempts at map initialization are often computationally expensive and are best kept at a minimum if applied to a real-world SLAM system. Thus, the goal of evaluating the initialization suitability of a pair of image frames is to determine if the image pair is likely to result in a reliable mapping *before* attempting to generate the map.

4.1 Existing Approaches

Out of the growing body of visual SLAM literature, relatively few works address the frame selection problem. This is, in part, due to the vast utility and ease afforded by constraining the SLAM problem to make use of known environment data or fiducial markers. Though systems that make use of known environment data often go as far as to leverage this data for camera localization [84]–[88], even largely-markerless systems like MonoSLAM [12] still utilize fiducials to aid in solving the map initialization problem, which enables the system to avoid the frame selection problem altogether.

However, though markers help to simplify the SLAM problem as a whole (and can even be used to enhance monocular SLAM capabilities by providing scale data [89]), their use constrains the applicability of any SLAM system that requires them. Marker-based systems require environment preparation on behalf of the user and are unable to perform pose es-

timation when the markers fall out of view; these traits are meaningfully restrictive when considering real-world applications of SLAM. Thus, there has been increased interest in the development of SLAM systems that forego the dependency on markers by instead tracking natural features and initially reconstructing their 3D locations with SfM techniques [16]–[18], [90], [91]. The frame selection problem is relevant to these types of SLAM implementations, as they are monocular configurations with SfM-based map initialization.

To address the frame selection problem, Parallel Tracking and Mapping (PTAM) [16], [17] and a later work by Sun, et al. [90] require the end-user to manually indicate which frames are to be used for map initialization during runtime. Though this approach is straightforward and effective, it is not ideal for real-world SLAM applications as user intervention may be inconvenient or may even require a high degree of savvy from the end-user.

ORB-SLAM [18], on the other hand, implements an automated approach to frame selection. It accomplishes this by first recording a reference frame during runtime and then attempting initialization on each subsequent frame, using the reference frame and current frame as the frame pair. After attempting initialization, the system uses a novel suitability criteria to determine if the resulting map quality is sufficient to move on from the initialization module. Specifically, ORB-SLAM checks that the resulting map contains a large number of triangulated points that uphold the cheirality constraint [69], have low reprojection error, and have high parallax with the two localized camera frames. These metrics provide a robust assessment of the map quality; however, this brute-force approach is computationally expensive, as the entire map initialization process is performed on every frame. This makes it ill-suited for applications that need to be implemented on resource-limited platforms like mobile phones or low-cost embedded systems.

Outside of user-assisted and brute-force frame selection techniques, there are relatively few works that attempt to provide a fast, automatic approach for frame selection. One of the earliest relevant works that proposes a solution to this problem is a contribution by Tomono [92]. In this work, Tomono presents a frame selection approach in which the feature matches in the image pair can be used to predict the degeneracy of the fundamental matrix before its computation. This approach, however, only applies to the estimation of the fundamental matrix, making it unsuitable for systems that use essential matrix estimation or

homography estimation to perform map initialization. Additionally, the degeneracy metric provided in this approach does not distinguish between degeneracies caused by poor baseline and degeneracies caused by high coplanarity of the scene points. This distinction would be particularly useful for systems that utilize multiple SfM approaches for high applicability. Lastly, this approach presents a threshold parameter to determine the suitability of the frame selection; however, the appropriate value for this parameter changes as the number of correspondences changes, which could make the implementation of this approach on a real-world application of SLAM challenging.

A simpler, yet still effective, approach to frame selection is used in VINS-Mono [91]. VINS-Mono implements a suitability criteria in which a frame pair is used for map initialization if the frames share at least 30 correspondences that have endpoint disparities over 20 pixels. A later SLAM system [93] implements a similar approach, except it evaluates the median and standard deviation of the endpoint disparities to not only determine the suitability of the frame pair, but to also determine the specific SfM approach that the frame pair is most suited for. Though these approaches are very practical, they fail under pure rotational cases and their accuracy leaves much room for improvement, as shown in [1].

4.2 A Deep Learning Solution

Though the approaches used in [93] and [91] are straightforward and effective, deep learning can be used to achieve significantly improved accuracy over these methods while maintaining low computation cost. This can be accomplished by extracting correspondences from the candidate frame pair and then deducing summary data from this set of correspondences. This summary data can then be used as input for a small classifier to determine if the set of correspondences is likely to yield a successful mapping. If the model indicates that a successful mapping is likely, then an SfM technique can be used to initialize the map. Otherwise, the system can skip to the next frame, collect correspondences between the reference frame and the new frame, and start the process over.

Since different SfM algorithms have different degeneracy conditions, a specific classifier can be developed for each approach. This way, if a SLAM system needs to perform map

initialization and is currently looking at a planar scene, the classifier for the fundamental matrix approach (see Section 3.4.2) may indicate a negative result while the classifier for the homography approach (see Section 3.4.3) may indicate a positive result. If a SLAM system incorporates multiple SfM approaches for map initialization, this combination of model outputs can be used to inform the system as to which approach is viable for the current frame pair.

4.2.1 Model Configurations

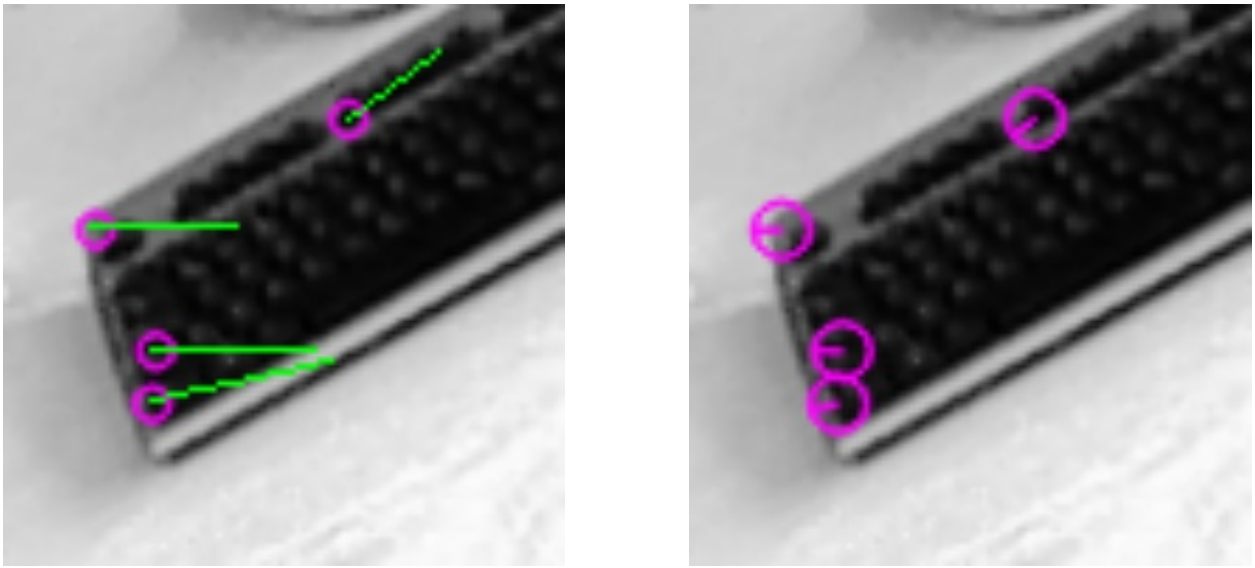
To demonstrate this concept, five classifiers are trained for each of the three SfM approaches described in Section 3.4, totaling 15 classifiers. Each classifier receives the same 23 correspondence summary features as input. Given that a single correspondence consists of a point in the first image, (x, y), and a point in the second image, (x', y'), the model input features include: the number of correspondences extracted for the image pair, the mean of the correspondence endpoint disparities (the distance between (x, y) and (x', y')), the standard deviation of the correspondence endpoint disparities, the minimum and maximum values for each component of the correspondences (x, y, x' and y'), the range of each component of the correspondences, and a normalized eight-vector which acts as an analog for the distribution of correspondence directions. Specifically, the direction, θ, of a correspondence is defined by

$$\theta = \arctan\left(\frac{y' - y}{x' - x}\right) \tag{4.1}$$

.

After the direction is computed for a correspondence, the direction is used to increment one of the eight features in this vector. Once all directions have been computed, this vector is normalized before being appended to the model inputs. Figure 4.1 shows a concrete example of the computation of the direction vector.

For each SfM approach, five classifiers are developed in an attempt to maximize model accuracy. The first classifier is a basic logistic regression model that simply runs the 23 input features through a sigmoid activation, interpreting 1 as the positive class and 0 as the negative class. These classifiers were first presented in [1] and showed improved precision



(a) (b)

Figure 4.1. (a) An example of four tracked features (circled) with their motion correspondences indicated by green lines. The uncircled endpoints of the motion correspondences indicate the initial positions of the features (from a previous frame) and the circled endpoints indicate the features' positions in the current frame. (b) Visualization of the computed direction for each correspondence. Note that each feature's direction coincides with the angle of its correspondence shown in (a). These directions are discretized about the eight cardinal directions (N, NE, E, SE, S, SW, W, NW) for the construction of the model's directional input vector. In this example, since the topmost correspondence is mostly southwest-facing while the other correspondences are mostly west facing, the direction vector would be (0, 0, 0, 0, 0, 1, 3, 0), or (0, 0, 0, 0, 0, 0.32, 0.95, 0) after normalization.

over the approach described in [93]. The remaining four models are deeper neural networks that run the 23 input features through two dense hidden layers (with ReLU activations) and result in two-class softmax output layers. The only difference between each neural network is the size of the hidden layers, which are 8×8, 16×16, 32×32, and 64×64.

4.2.2 Training and Labeling Criteria

To train each model, sample image pairs are extracted from sequences of the TUM RGB-D dataset [94]. Each sequence makes up a video of a different scene as the camera undergoes different movements in the environment.

Model data is generated from these sequences by segmenting each sequence into smaller batches of frames and extracting feature correspondences between pairs of frames in each batch. To compensate for the dramatic differences in camera movements between different sequences, each sequence is segmented with a different batch size. Specifically, the batch sizes used include 10 frames, 30 frames, 45 frames, and 60 frames. Within each batch, image pairs are extracted by simply pairing the first frame in the batch with every other frame in the batch. This helps ensure that each batch provides image pairs that demonstrate *little* camera movement as well as image pairs that demonstrate *substantial* camera movement.

As it is useful to prepare training data that is similar to that which the models will see in a real-world scenario, correspondences are extracted from each image pair by detecting and matching ORB [64] features, which are extracted homogeneously throughout the image.

The classification label for each image pair is determined by reconstructing the scene with each model's associated SfM approach and then evaluating the quality of the map in a fashion similar to that seen in ORB-SLAM [18]. This means that each image pair will have three labels associated with it: a label for the fundamental matrix classifiers, a label for the essential matrix classifiers, and a label for the homography matrix classifiers. Re-evaluating the labels for each approach is necessary because the SfM approaches may generate maps of substantially different quality given a single set of correspondence data.

After an SfM approach is used to reconstruct a map of the correspondences, map quality is determined by three characteristics of the triangulated points: parallax, reprojection error,

and cheirality. Specifically, the reconstruction is classified in the negative class if it contains less than 50 triangulated points with parallax greater than 2° or if it has less than m points that both uphold the cheirality constraint and also have less than a 4 pixel reprojection error, where m is the maximum between the values 50 and $0.9 \times n$, and n is the number of inliers deduced from the corresponding RANSAC scheme used in the SfM matrix estimation process. If neither of these failure conditions are met, then the map quality is considered sufficient and the image pair is given a label associated with the positive class for the corresponding classifier.

After labels are generated for each sample, the samples are duplicated for each SfM approach and then balanced to include the same number of positive and negative samples to improve training. This is achieved by duplicating samples from the deficient class. The resulting data is shuffled and 20% is split into a validation group to evaluate the model performance. Additionally, 2,498 samples of validation data are generated from a separate sequence from the TUM RGB-D dataset (a sequence that is never used in training), to further evaluate model performance. In total, classifiers evaluating suitability for essential matrix estimation are trained on 23,537 samples, classifiers evaluating suitability for fundamental matrix estimation are trained on 31,619 samples, and classifiers evaluating suitability for homography matrix estimation are trained on 23,372 samples. Their validation set sizes are 5,885 samples, 7,905 samples, and 5,844 samples, respectively.

4.2.3 Model Accuracy

The cross-validation results for each model are shown in Tables 4.1, 4.2, and 4.3, and validation results against the unseen sequence are shown in Tables 4.4, 4.5, and 4.6.

The results from cross-validation show a relatively high F1 score for the best model in each initialization approach, with the 16×16 neural network reporting an F1 of 0.7853 for predicting suitability of essential-matrix-based initialization, the 32×32 neural network reporting an F1 of 0.8476 for predicting suitability of fundamental-matrix-based initialization, and the 64×64 neural network reporting an F1 of 0.7948 for predicting suitability of homography-matrix-based initialization. When moving to the results measured with the

Table 4.1. Cross validation performance for models trained for essential-matrix-based initialization using 80%-20% data split for training and testing, respectively.

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.6384	0.6556	0.6109	0.6325
Neural Network (8×8)	0.7288	0.7974	0.5997	0.6846
Neural Network (16×16)	0.7845	0.7684	**0.8030**	**0.7853**
Neural Network (32×32)	**0.7976**	**0.8334**	0.7344	0.7808
Neural Network (64×64)	0.7691	0.7806	0.7365	0.7579

Table 4.2. Cross validation performance for models trained for fundamental-matrix-based initialization using 80%-20% data split for training and testing, respectively.

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.6114	0.6115	0.6223	0.6169
Neural Network (8×8)	0.7462	0.7847	0.6749	0.7257
Neural Network (16×16)	0.7598	0.8593	0.6182	0.7190
Neural Network (32×32)	**0.8521**	**0.8691**	**0.8273**	**0.8476**
Neural Network (64×64)	0.7820	0.8587	0.6723	0.7542

Table 4.3. Cross validation performance for models trained for homography-matrix-based initialization using 80%-20% data split for training and testing, respectively.

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.6458	0.6379	0.6437	0.6407
Neural Network (8×8)	0.7435	0.8136	0.6438	0.7188
Neural Network (16×16)	0.7827	0.8462	0.7006	0.7665
Neural Network (32×32)	0.7707	0.8851	0.6317	0.7373
Neural Network (64×64)	**0.8126**	**0.8983**	**0.7127**	**0.7948**

Table 4.4. Validation performance for models trained for essential-matrix-based initialization, using test data generated from an unseen sequence.

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.5969	0.7089	0.6001	0.6500
Neural Network (8×8)	**0.7870**	**0.7525**	0.6468	**0.6957**
Neural Network (16×16)	0.7226	0.6084	**0.7372**	0.6667
Neural Network (32×32)	0.7030	0.5943	0.6638	0.6271
Neural Network (64×64)	0.6869	0.5663	0.7181	0.6332

Table 4.5. Validation performance for models trained for fundamental-matrix-based initialization, using test data generated from an unseen sequence.

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.4696	0.1897	0.5598	0.2834
Neural Network (8×8)	0.5965	**0.8724**	0.5897	0.7037
Neural Network (16×16)	0.5733	0.8713	0.5571	0.6797
Neural Network (32×32)	**0.7354**	0.8312	**0.8463**	**0.8387**
Neural Network (64×64)	0.6137	0.8716	0.6153	0.7213

Table 4.6. Validation performance for models trained for homography-matrix-based initialization, using test data generated from an unseen sequence.

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.5857	0.6265	**0.6324**	0.6294
Neural Network (8×8)	**0.7118**	0.7256	0.5632	**0.6341**
Neural Network (16×16)	0.6749	0.6667	0.5343	0.5932
Neural Network (32×32)	0.6962	**0.7257**	0.5063	0.5965
Neural Network (64×64)	0.6277	0.5808	0.5776	0.5792

unseen sequence, the scores are slightly lower than the scores of the cross-validation set, as the models neither have exposure to this sequence nor to sequences of the same scene during training. Regardless, the F1 scores of the deep neural network models show an improvement over the baseline logistic regression models.

In addition to the accuracy metrics provided, each model runs inference for a single sample in 4 milliseconds on a machine sporting a Ryzen 7 5800X CPU and GTX 1080 GPU using DeepLearning4j. Coupled with the accuracy metrics discussed above, the results indicate that these models can be usefully integrated into real-world SLAM systems to accelerate the suitability evaluation process during map initialization. Consequently, the 32×32 neural network model for predicting fundamental-matrix-based initialization suitability is used to aid in the map initialization of LUMO-SLAM, which is discussed in detail in Chapter 6.

5. MOVING OBJECT REGISTRATION AND LOCALIZATION

As AR and VR are developed to become more pervasive technologies, the need for a higher degree of functionality from underlying tracking solutions becomes apparent. For example, imagine a developer wants to create an AR application that enables an end-user to annotate arbitrary objects in their environment with labels that specify the names of the objects. This sort of application could then programmatically translate the labels to a target language for the end-user, enabling them to engage in an immersive language-learning process. If this application is supported by a normal SLAM system, then problems arise when an agent begins moving labeled objects in the environment. Modern SLAM systems do not account for deformations in the mapped scene, so object labels would remain static in the environment even after scene objects have been moved. This would likely disrupt the user's experience and potentially cause the environment to become polluted with inaccurate labels. However, if the underlying SLAM system dynamically registered and relocalized moving objects in the environment, then it could enable the application to relocalize any labels that are intended to be associated with moving objects. This is just one basic example of how robust AR systems of the future will be made possible with the advent of dynamic moving object registration and localization in SLAM.

It is also worth noting that AR applications of the future will take particular advantage of a moving object registration and localization approach that does not require the system to be trained on specific objects ahead of time. Though there are many existing dynamic SLAM implementations that offer some degree of moving object registration and localization, most of them rely on this kind of training, which greatly limits the applicability of this functionality. The approach discussed in the following sections, however, is a general approach for solving the moving object registration and localization problem, and its integration into the SLAM process will enable the system to register and localize arbitrary moving objects in the scene without any prior knowledge of the objects' structure, appearance, or existence.

5.1 Modeling Moving Objects in SE(3)

Before exploring how moving objects can be detected and relocalized, it is important to first cover how an object's position and orientation can be represented in the pinhole camera model. To review from Sections 3.1 and 3.2, the pinhole camera model is a mathematical abstraction that projects 3D points into a 2D image plane with the following equation:

$$\mathbf{x} = \mathbf{KEX} \tag{5.1}$$

Expanded, this equation becomes:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{5.2}$$

.

In this equation, the point (x, y, z) is transformed by the camera's extrinsic parameters, $\mathbf{E}$, and then perspective-projected into the image plane represented by the camera's intrinsic parameters, $\mathbf{K}$, to the image point, $(u/w, v/w)$. This model can also take on a slightly modified representation in which the extrinsic parameters are represented as a true SE(3) matrix:

$$\begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{5.3}$$

.

This representation is particularly useful, as multiple transformations can now be modeled in the projection of a point. Specifically, the transformation of a point (before it is projected into the image) can be viewed as a composition of transformations in which the point is first transformed by a model transformation, $\mathbf{M}$, and then transformed by the cam-

era's extrinsic parameters, **E**, before finally being projected into the image with the camera's intrinsic parameter matrix, **K**. This is demonstrated in the following adaptation of the pinhole camera model equation:

$$\mathbf{x} = \mathbf{KEMX} \tag{5.4}$$

.

Now, the motion of moving objects can be modeled with a different model transformation matrix, **M**, for each moving object. If a point is part of the static map (and thus does not have an associated model transformation), then its model matrix, **M**, can simply take the form of the identity matrix, **I**.

Given this formulation, it becomes clear that the goal of moving object localization is to solve for the value of the model transformation matrix, **M**, for each moving object, given the object points represented with varying values of **X**, their projected locations in the image, $(u/w, v/w)$, and the camera's parameters, **K** and **E**.

5.2 Localizing Moving Objects with EP*n*P

As the EPnP algorithm shown in Section 3.6 can be used to deduce the extrinsic camera parameter matrix, **E**, it can also be used to deduce the model transformation matrix, **M**, for a set of points if the extrinsic camera parameters are already known (or have already been estimated). This is accomplished by viewing the result from the EPnP algorithm as a single transformation, **T**, that composes all transformations from a point, **X**, to the point's projected location in the image, **x**, such that

$$\mathbf{x} = \mathbf{KTX} \tag{5.5}$$

.

If all points, **X**, are object points of the same potentially-moving object (as opposed to points associated with the static map), then the constraint from Equation 5.4 also accurately

models the motion of the object points for some value of $\mathbf{M}$. Given this additional constraint, it follows that

$$\mathbf{KTX} = \mathbf{KEMX} \tag{5.6}$$

and

$$\mathbf{T} = \mathbf{EM} \tag{5.7}$$

where $\mathbf{T}$ is the resulting transformation estimated from the EPnP algorithm, $\mathbf{E}$ is the extrinsic camera parameter matrix, and $\mathbf{M}$ is the model transformation matrix for the object points of the potentially-moving object. If the EPnP algorithm is first used on a batch of static map points, then the resulting transformation will represent the extrinsic camera parameters, $\mathbf{E}$, as the model transformation for the static map is defined as the identity matrix, $\mathbf{I}$. Given the extrinsic camera parameters from this calculation, the EPnP algorithm can then be used on a batch of potentially-moving object points to deduce the composite transformation, $\mathbf{T}$, from which the model transformation for the object, $\mathbf{M}$, can be extracted with

$$\mathbf{E}^{-1}\mathbf{T} = \mathbf{M} \tag{5.8}$$

.

5.3 Registering Moving Objects with RANSAC and EPnP

The above approach is sufficient for localizing a set of map points belonging to a non-static object; however, the approach does not provide a technique for differentiating between static map points and moving object map points. Given purely-monocular image data, the problem of automatically distinguishing between static and non-static map points is non-trivial. Despite the challenges posed by this problem, however, a practical solution exists by taking advantage of the reprojection error utilized by the RANSAC algorithm, discussed in Section 3.6.1.

When coupled with the EPnP algorithm, the RANSAC algorithm identifies outlier points by evaluating the error between the observed feature points and their corresponding projections in the image, using the 3D object points and the candidate transformation model that is being estimated. Normally, object points with high reprojection error are interpreted as either noisy points that have not been triangulated properly, or as points that were incorrectly associated with their corresponding image feature points. However, high reprojection error may also be the result of an object moving in the scene.

If a majority of the map points that are matched in a frame are static in the scene, then the RANSAC/EPnP algorithm will deduce the extrinsic camera parameters while classifying any moving object points as outliers, as their motion is not sufficiently explained by the camera movement alone. This resulting camera transformation matrix can then be used to determine the map points with high reprojection error, which includes map points associated with objects that are not currently in their initially-triangulated position. The RANSAC/EPnP algorithm can then be used on these high-error points in an attempt to identify any additional consistent motion. If a geometric transformation is estimated among these points with at least six inliers (as the localization problem has six degrees of freedom), then this result can be interpreted as a composite motion between the camera and a moving object. From this point, the inlier map points from this estimate can be removed from the static map and subsequently grouped together into a data structure unifying them under a single model transformation, $\mathbf{M}$, which can be deduced with the previously-computed camera extrinsics, $\mathbf{E}$, and the resulting transformation from applying RANSAC/EPnP on the outlier points, $\mathbf{T}$, using Equation 5.8.

This process can be repeated on the remaining set of outlier points by removing map points that are found to be associated with moving objects until fewer than six points remain. This means that, theoretically, any number of moving objects can be identified in a single frame, given that at least six points are accurately registered on each object.

It is worth noting, however, that this approach has a few practical limitations. Firstly, in practice, this approach requires that moving objects be feature-rich to ensure a higher likelihood of registering at least six points on the object. Secondly, this approach expects the moving object points to have been triangulated while the object was static, *before* it begins

moving. This constraint is necessary in a monocular system, as there is insufficient data to jointly localize and triangulate an object while it moves without facing troublesome scale ambiguity. Lastly, some motions may result in some of the object points yielding relatively low reprojection error (imagine an object that is rotated about one of its feature points; the feature point at the origin of this rotation is otherwise static in the context of the rest of the map). This means that, under some motions, there may be a small number of object points that are not properly registered as moving object points. It is the burden of the system developer to work around this largely-innocuous limitation; however, a simple real-world solution is implemented and discussed in Chapter 6.

6. LUMO-SLAM

LUMO-SLAM is a proof-of-concept, keyframe-based SLAM system that performs markerless, monocular simultaneous localization and mapping while also registering and localizing unknown moving objects in the scene using the approach described in Chapter 5. This system is a significant contribution, as it is the first markerless, monocular SLAM system to perform moving object registration and localization with no prior knowledge of the objects' structure, appearance, or existence.

A system diagram of LUMO-SLAM is shown in Figure 6.1. The following sections explain the components of the system, including each of its processes, in detail. After exploring the system details in this chapter, the next chapter covers LUMO-SLAM's quantitative results and their analysis.

Additionally, the source code for LUMO-SLAM is made publicly available at https://github.com/batroutman/LUMO-SLAM.

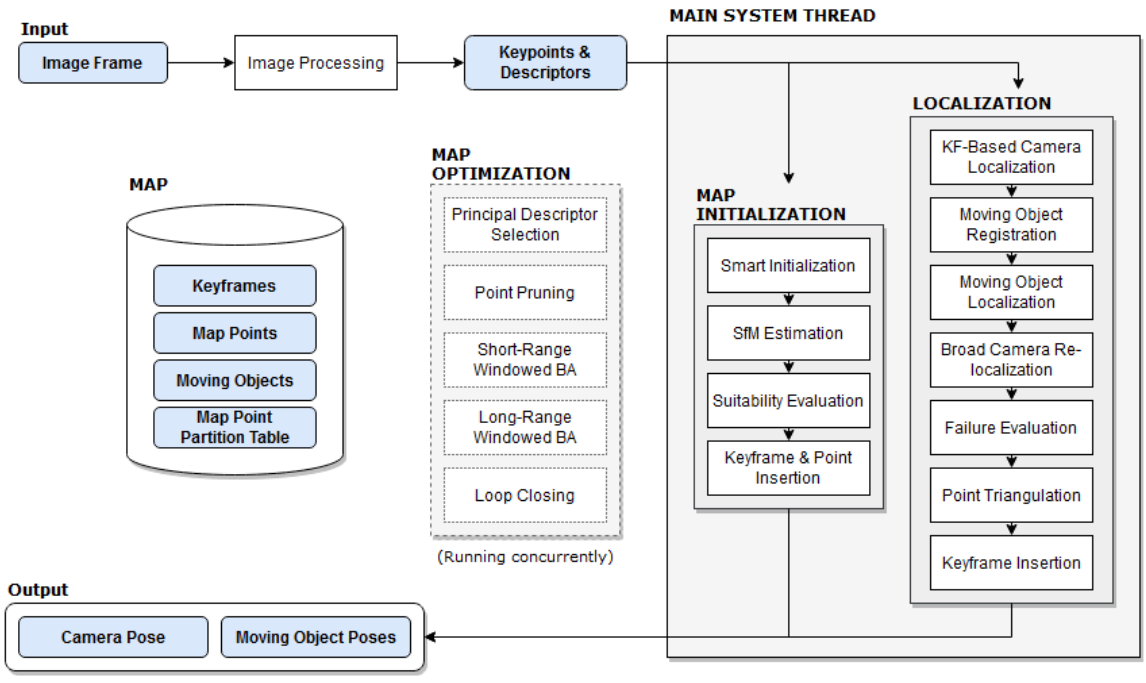


Figure 6.1. LUMO-SLAM system diagram.

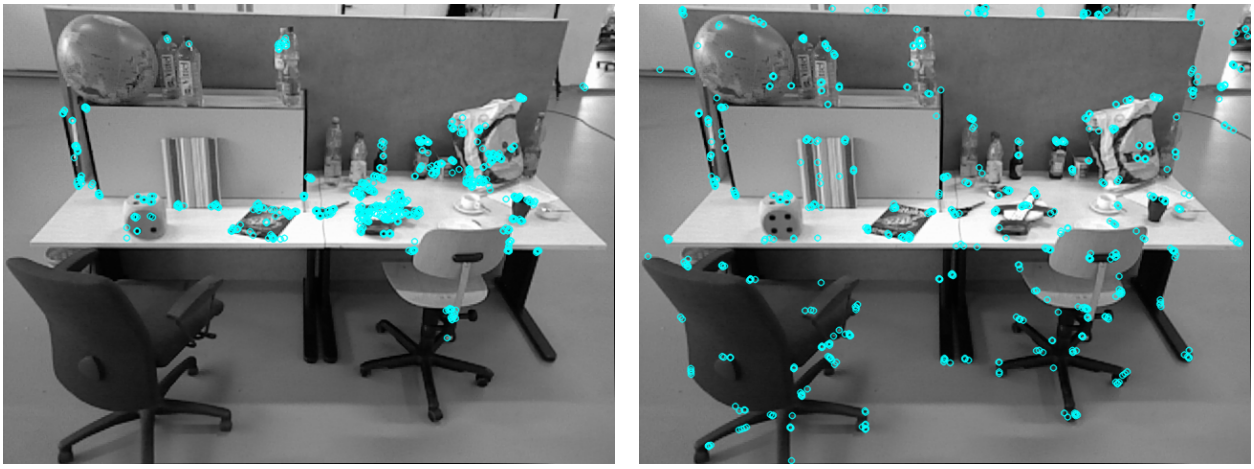
6.1 Image Processing

Every iteration of LUMO-SLAM's main loop begins with the same image processing approach. As a new frame is pulled by the system, it is converted to greyscale (if not already in monochrome format) and then it is used to construct an image pyramid for feature extraction. Specifically, the image pyramid consists of 8 copies of the frame, each downscaled from the previous copy by a factor of 1.2. This allows the system to extract salient features of many sizes without changing the feature extraction algorithm. Once the image pyramid is constructed, ORB features (as discussed in Section 3.3) are extracted evenly throughout the image.

Even-spread feature extraction is useful for well-conditioning the collection of features for map initialization and also for providing better constraints for camera localization. In order to extract features evenly, FAST features with a score[1] of at least 20 are extracted throughout the entire image. 20 is a weak feature extraction threshold, but it enables the system to gather keypoints in many areas of the image. This large collection of feature points is then filtered down by organizing the features into bins, grouped by the regions of the image that they were extracted from. The system then evaluates each bin and prunes features until there are between $n_{\mathbf{min}}$ and $n_{\mathbf{max}}$ features in each cell[2]. To accomplish this, the features of each bin are classified as either "strong" or "weak" by assessing the response scores of each feature. A feature is classified as "strong" if its FAST response is at or above 60; otherwise, the feature is classified as "weak". For each bin, if the number of strong features meets or exceeds $n_{\mathbf{max}}$, then just the strongest $n_{\mathbf{max}}$ features are retained in that bin. Otherwise, if the number of strong features still meets or exceeds $n_{\mathbf{min}}$, then all of the strong features are retained in the bin while all weak features are pruned. Finally, if neither of these conditions are met (indicating that the bin is populated with few strong features and, potentially, many weak features), then the strongest $n_{\mathbf{min}}$ features are retained in the bin, given that there are already at least $n_{\mathbf{min}}$ features in the bin. After extraneous features are pruned, a global limit

[1]↑A feature's FAST score coincides with the sum of the absolute differences between the feature's central point and each of the contiguous high-contrast points within the feature's evaluation circle.

[2]↑Though the values for $n_{\mathbf{min}}$ and $n_{\mathbf{max}}$ are configurable, good values that work well on most datasets are 4 and 8, respectively.



(a) Default ORB extraction in OpenCV (b) LUMO-SLAM feature extraction

Figure 6.2. ORB feature distribution comparison between the API provided by OpenCV (a) and the approach implemented in LUMO-SLAM (b). Each example extracts 700 features; however, the features are excessively concentrated towards the center of the image when using OpenCV's API while the features are spread-out in LUMO-SLAM's implementation. The consistent extraction of features in many parts of the image provides more-informative constraints for localization and mapping.

on the total number of features in the image is imposed to prevent excessive computational overhead in subsequent modules. This configurable global limit is typically set between 700 and 900 features to ensure realtime performance.

Figure 6.2 illustrates the difference in feature distribution between the default feature extraction approach used in OpenCV and the feature extraction approach used in LUMO-SLAM. Basic feature extraction places all emphasis on the strength of the features, causing the extracted features to overpopulate some regions of the image and underpopulate others. However, when extracting features with the approach described above, usable features are extracted from more regions of the image.

Once the final FAST features have been extracted from the image, their intensity centroid angles are calculated and are subsequently used to compute an ORB descriptor for each feature. The ORB features' keypoint and descriptor values are passed on to the remaining

modules of the system, and the original image is unused for the remainder of the loop iteration.

6.2 Map Initialization

As LUMO-SLAM is a markerless, monocular SLAM system, map initialization must be performed by the system before the normal camera localization process can be performed. Map initialization is the process of establishing an initial set of triangulated map points to track while also recording an initial set of keyframes to aid in map optimization. With no fiducial markers in the environment, the problem of map initialization is challenging, especially for a monocular system.

To solve the problem of map initialization, LUMO-SLAM records features in a reference frame (typically, the first frame pulled by the system), and matches them to features in the current frame. From there, these 2D-to-2D image correspondences are evaluated for their initialization suitability, and then used in an SfM estimate to construct the initial elements of the map.

Not every frame will result in a set of correspondences that are suitable for an SfM estimate, so the map initialization module may require many frames to establish an initial map. Because of this, the reference frame is reset (set to the current frame) every 300 frames to maintain sufficient feature tracking.

Once map initialization completes successfully, the map initialization module is retired and remains unused for the remainder of the system runtime. The following sections describe, in detail, the steps associated with map initialization for every SLAM loop iteration.

6.2.1 Smart Initialization

To accelerate the map initialization process, the initialization suitability evaluation approach described in Chapter 4 is used to quickly predict the likelihood of successful initialization, given the current set of correspondences. For ease of implementation, LUMO-SLAM only utilizes a fundamental matrix estimation approach, and thus only makes use of the 32×32 fundamental-matrix-based model from Chapter 4. However, the system could easily

be adapted to make use of homography estimation as well, and consequently could make use of a homography suitability model in addition to its current fundamental matrix suitability model.

As described in Chapter 4, the system computes a small set of summary features for the set of correspondences it has collected between the reference frame and the current frame. It then feeds these features through a forward pass of the trained 32×32 fundamental matrix model to predict the likelihood that this set of correspondences will result in a sufficient reconstruction, given that they would be used to estimate the fundamental matrix.

If the model indicates a high likelihood of initialization success, then the correspondences are forwarded on to the next stage of map initialization. Otherwise, map initialization is interrupted and the system skips to the next frame to attempt map initialization again, repeating this process until the map is initialized.

6.2.2 Structure from Motion Estimation

If the classifier from the previous process indicates that the set of correspondences are suitable for map initialization, then they are used as input for the eight-point algorithm to estimate the fundamental matrix between the pair of views. For ease of implementation, this is the only SfM approach used in this process; however, additional SfM techniques could be added in the future to provide more robustness. It is also worth noting that a consequence of only using a fundamental matrix estimate for map initialization is that the system can only reliably initialize the map when viewing a scene that contains points that are *not* mostly coplanar in 3D space. The addition of a homography estimation could complement the current approach by enabling the system to also initialize the map with coplanar points, but this addition is currently left for future work, as it is not a necessary component of the SLAM process.

As described in Section 3.4.2, the eight-point algorithm solves for the fundamental matrix, $\mathbf{F}$, by using the constraint,

$$\mathbf{x}'\mathbf{F}\mathbf{x} = 0 \tag{6.1}$$

for at least eight 2D-to-2D image correspondences, where $\mathbf{x}$ and $\mathbf{x}'$ are homogeneous column vectors representing the image coordinates of a projected point in the primary image and the secondary image, respectively.

After solving for the fundamental matrix, the camera poses for the primary and secondary cameras are deduced by first using the known camera intrinsic parameters, $\mathbf{K}$, to convert the fundamental matrix into the essential matrix, $\boldsymbol{E}$, with Equation 6.2.

$$\boldsymbol{E} = \mathbf{K}^\top \mathbf{F} \mathbf{K} \tag{6.2}$$

The essential matrix is then decomposed into four camera pose hypotheses using its singular value decomposition (SVD), as shown in Section 3.4.1. Specifically, the reference frame is always assumed to be positioned with the default pose (represented with a 4×4 identity matrix in SE(3)), and the current frame's pose is hypothesized with each of the four factorizations of $\boldsymbol{E}$.

To select the correct pose hypothesis for the current frame, all of the correspondences are triangulated with the assumed reference frame pose and the respective hypothesis pose. For each triangulated point, the cheirality is evaluated against both the reference frame and the current frame. Ultimately, the hypothesized pose with the most triangulated points that uphold the cheirality constraint for both the reference frame and the current frame is selected as the correct pose for the current frame.

6.2.3 Suitability Evaluation

Before committing to the usage of the pose selected from the process described above, the map's quality is evaluated directly by temporarily constructing the map with the selected pose and assessing a number of criteria regarding the reconstruction. Notably, this highly useful suitability evaluation process is largely inspired by the approach used in ORB-SLAM [18].

To evaluate the quality of the map, the correspondences are triangulated using the default pose for the reference frame while the current frame's pose is assumed to coincide with the pose selected from the SfM estimation process. After triangulating all correspondences, three

factors are observed for each point: the point's cheirality about both of the cameras, the point's reprojection error in each camera, and the parallax of the two vectors that are derived by connecting the point to each of the cameras.

For this evaluation, a point is considered high-quality if it upholds the cheirality constraint in both cameras and has a reprojection error of less than 4 pixels in each camera. The map quality is then deemed sufficient if two criteria are met: (1) the number of high-quality points meets or exceeds $\mathbf{max}(50, 0.9 * n)$, where n is the number of inliers deduced from the eight-point algorithm estimate, and (2) the number of points resulting in parallax values greater than 2° meets or exceeds 50.

If these criteria indicate that the map quality is sufficient, then the system commits to the pose selection and its corresponding map reconstruction. Otherwise, the temporary map is thrown out, the processes is halted, and map initialization makes another attempt on the next frame.

6.2.4 Initial Keyframe and Point Insertion

Upon passing the suitability evaluation, the estimated pose and corresponding triangulated points are used to populate the map with its initial data. Specifically, a keyframe is generated for both the reference frame and the current frame. The keyframes primarily consist of a camera pose, the images' keypoint locations, the keypoints' corresponding ORB descriptors, and references to any triangulated points that correspond to the keyframes' keypoints. The first keyframe in the map corresponds to the reference frame and its pose is set to the default pose (which constitutes zero rotation and a translation vector at the origin). The second keyframe in the map corresponds to the current frame and its pose is set to coincide with the pose estimated from the eight-point algorithm. In addition to the first two keyframes, the triangulated map points for the correspondences between the two frames are also inserted into the map. With the insertion of triangulated map points, each keyframe is also given a list of references to the map points it observes. This aids in map optimization and allows the system to quickly extract relevant map points during camera localization.

After inserting the initial keyframes and triangulated map points, the map is briefly refined with 100 iterations of full bundle adjustment. As discussed in Section 3.7, bundle adjustment is the non-linear minimization of the reprojection error of the map points' projections into their observing keyframes. This process increases the accuracy of the map elements and can also be completed quickly (specifically, sub-second) when the map contains the limited amount of data that it has at this stage of initialization. Upon completion of bundle adjustment, points with very high reprojection error are pruned, and then the map is refined with 100 more iterations of bundle adjustment.

Once these two rounds of bundle adjustment are complete, the map is considered initialized, as it contains a robust set of triangulated map points and corresponding keyframes to use for localization. From this point forward, the system defaults to the localization module and the map initialization module is not revisited for the remainder of the system's runtime.

6.3 Localization

Once the map is initialized, the localization module can use the map data to perform normal camera localization on each incoming frame. In addition to performing basic camera localization, the localization module is also responsible for registering map points as moving object points, localizing any moving objects it has registered, and growing the map by recording new keyframes and triangulating new map points. The localization module is composed of a set of seven sequential processes. In order, these processes include keyframe-based camera localization, moving object registration, moving object localization, broad camera re-localization, failure evaluation, point triangulation, and keyframe insertion. Each of these processes are detailed in the following sections.

6.3.1 Keyframe-Based Camera Localization

The first process executed in the localization module is keyframe-base camera localization. The primary goal of this process is to get a quick initial estimate of the camera's current pose without relying on guided feature matching. In order to accomplish this, the system keeps track of the keyframe that was inserted into the map most recently (referred to as the "current

keyframe"). Since the current keyframe is frequently updated as features move, ORB features can be reliably brute-force matched between the current frame and the current keyframe without needing to predict the features' locations in the current image. The utilization of unguided feature matching is particularly useful in facilitating moving object registration, which relies on the system's ability to register features that are moving in unexpected ways.

Given that the feature matching is unguided, a hamming distance of 40 is used as the threshold to determine good matches between two binary ORB descriptors. After these 2D-to-2D correspondences are developed, substantial mismatches are pruned by removing any correspondences that have endpoint disparities greater than 25% of the image width. With these finalized matches, the current keyframe's descriptors are updated with the matching descriptors from the current frame to improve matching capability in future frames.

To perform camera localization, the correspondences are evaluated individually to identify those that are associated with triangulated map points, as some correspondences will be associated with map points that have not existed in the system long enough to deduce their 3D location. Once the applicable correspondences have been converted into 3D-to-2D correspondences (consisting of the triangulated map points and their respective feature points in the current frame), these correspondences are used as input for the EPnP algorithm in a RANSAC scheme, followed by an iterative optimization that minimizes the reprojection error of the input points. The result of this solver is the camera's six degree-of-freedom (6DoF) pose and a mapping of the inlier and outlier correspondences. The inlier/outlier discrimination made by the RANSAC scheme is not only useful for enabling the EPnP algorithm to estimate a pose that is robust to outliers, but this explicit identification of the outliers is also useful for registering unknown moving objects in the scene, which is completed in the next stage of the localization module.

6.3.2 Moving Object Registration

The second stage of the localization module is moving object registration. This process uses the approach described in Section 5.3 to identify triangulated map points that are attached to objects that are moving in the scene. As described in Section 5, this task

is achieved by exploiting the assumption that map points on moving objects are likely to have high reprojection error when the object is moving. Since the outlier classification from the keyframe-based camera localization is based on reprojection error, the outlier 3D-to-2D correspondences deduced from the previous process are useful for evaluating object movement.

To register if any matched map points are being influenced by the presence of an unknown object moving in the scene, the outlier correspondences from keyframe-based camera localization are fed back into the EPnP/RANSAC solver. If a valid pose is estimated with at least 6 inliers, the inlier correspondences are evaluated with a small set of practical conditions to confirm their status as moving object points. Specifically, for these inliers to qualify as moving object points, the spread of their feature points in the current frame has to be greater than 10% of the image width and less than 30% of the image width. This ensures that the object is at a suitable distance from the camera to facilitate a reliable localization estimate.

If the above conditions are met, then the inlier map points and their neighboring map points are removed from the static map and are all consolidated into a unifying data structure, indicating that the associated map points are part of a moving object. Neighboring map points can be identified quickly because triangulated map points are organized into a partition table, which is best described as a hash table in which the map points are indexed based on their discretized positions in the map.

In addition to the collection of associated map points, the model pose for the object is also saved to the moving object data structure. The model pose is computed using Equation 5.8, as the camera pose is already known from the previous process and the composite transformation, $\mathbf{T}$, is the result of the EPnP/RANSAC solver on the outlier correspondences.

This moving object registration process is repeated on the remaining outliers until fewer than six inliers remain. The consequence of this decision is that the system can register multiple moving objects in a single frame (assuming the objects all begin moving on the same frame and with different motions). To maintain realtime performance, the system also implements a configurable limit to the number of iterations that the registration process performs; though, it is worth noting that this is seldom needed in practice.

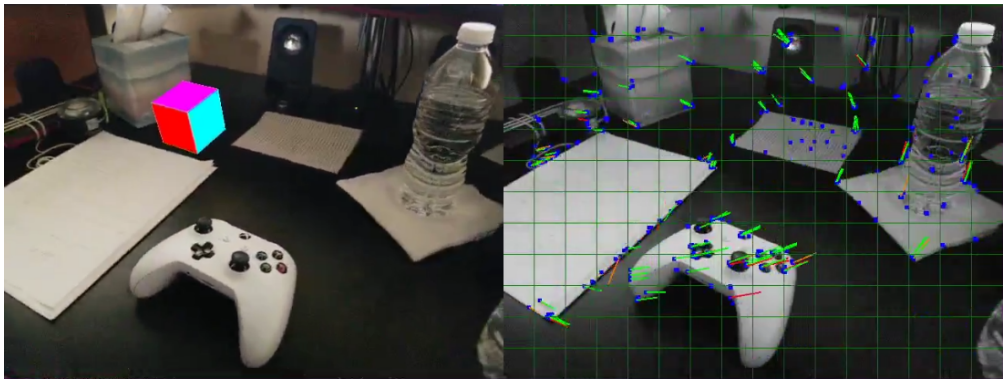
Figure 6.3 illustrates the moving object registration process on a sequence involving a user physically moving a game controller. While the controller is static, its feature points are mapped out. Then, the user moves the controller and the outlier feature matches are fit to a consistent motion model, causing the system to register the corresponding map points as moving object points.

6.3.3 Moving Object Localization

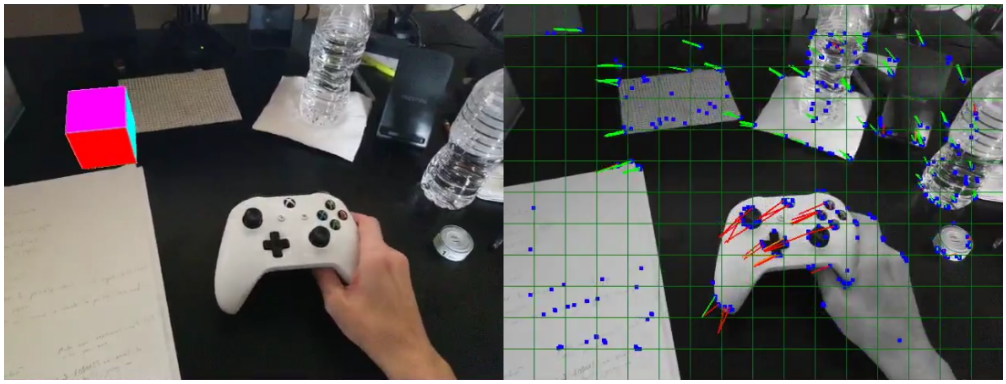
The third stage of the localization module performs moving object localization. In this process, each moving object that has been registered in the system during runtime is re-localized if it is found in the current frame, regardless of whether or not it was registered on the same frame.

To perform object localization for a given object in the system, the principal descriptors (described in Section 6.4.1) are extracted from the object's associated map points. These descriptors are used to perform unguided feature matching on the current frame, which establishes correspondences between the moving object points and their projections in the image.

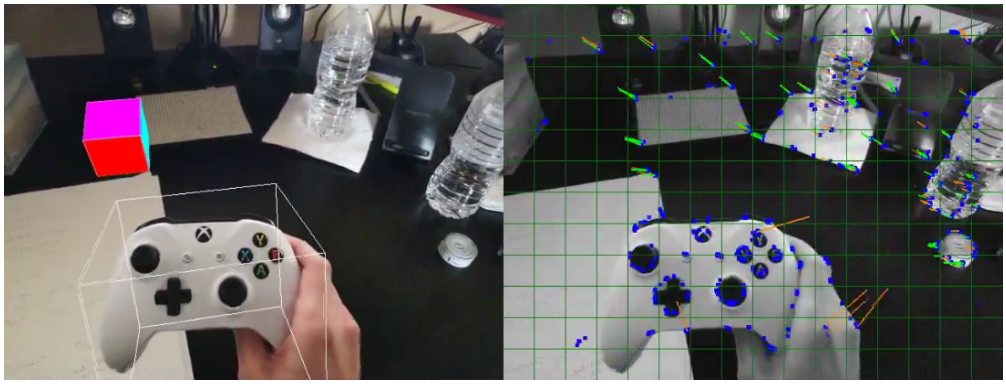
If there are at least six matches, these 3D-to-2D correspondences are fed into the EPnP solver in a RANSAC scheme to estimate a composite transformation for these points. As in the previous process, the composite transformation estimated by the EPnP solver is used in conjunction with the previously estimated camera pose to deduce a new model pose for the object. This is accomplished with Equation 5.8. If this model pose places a majority of the object points in front of the camera and the median reprojection error of these transformed points is sufficiently small (i.e., under approximately 4 pixels), then the model pose is trusted to be accurate and the associated moving object data structure is updated to reflect this new model pose. The updated model poses on the system's registered objects can be used by higher level applications that need access to moving object localization data, like the object labeling application demonstrated in Figure 6.4.



(a) Unknown object before movement

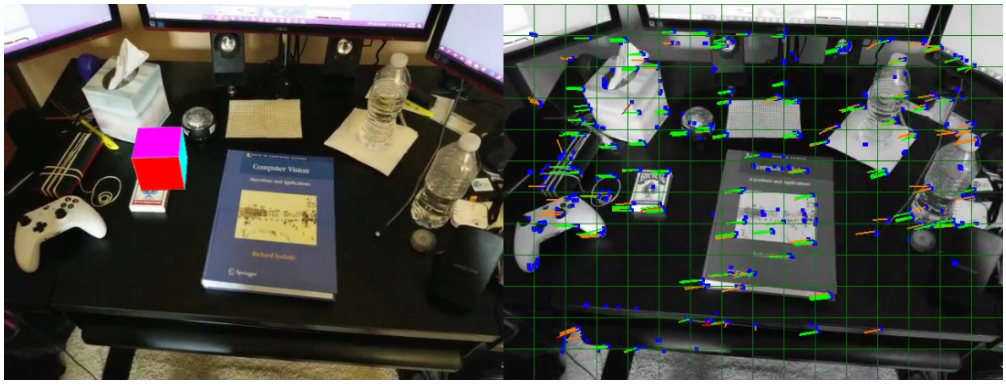


(b) Unknown object begins moving (outlier features detected)

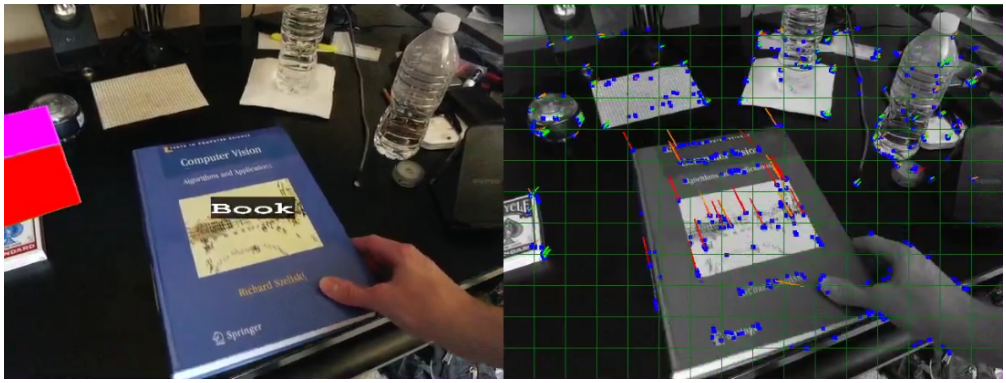


(c) Unknown object during movement, registered as a moving object (indicated by the bounding box around the object)

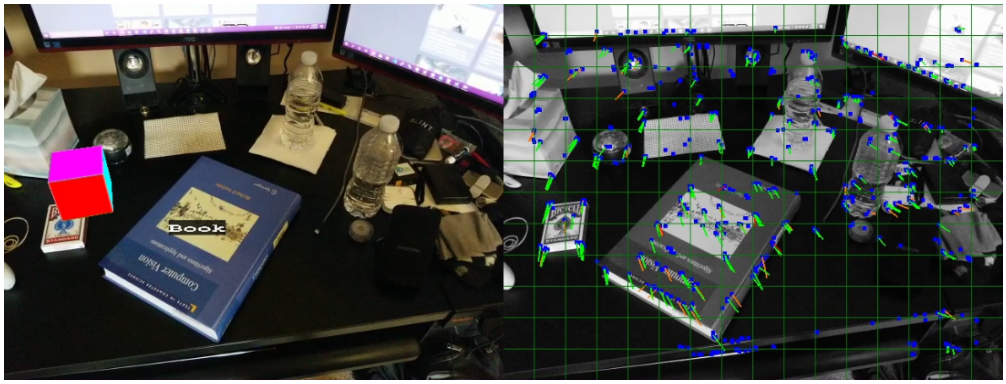
Figure 6.3. LUMO-SLAM running on a sequence in which a game controller is physically moved by the user. In this sequence, features of a game controller on a desk are mapped out (a) and then registered into a new moving object structure when the controller is picked up by the user (b) and (c).



(a) Initial static scene, including AR view (left) and LUMO-SLAM's correspondence tracking (right)



(b) Initial movement of the book: outlier features are detected, leading to the registration of a moving object and a user-defined label being attached to the object.



(c) Proper label localization when the camera moves away from the book

Figure 6.4. An object labeling application supported by LUMO-SLAM's camera localization and unknown object localization capabilities. In this application, the user pre-defines a label ("Book") for an object they want to annotate (in this case, a textbook). Then, after mapping out the environment (a), the user moves the object and the user's label is visually attached to the object using LUMO-SLAM's object localization data (b), displaying the label in the augmented view (left). Even as the user views the object from a different distance or angle, the localization data of the object is used to keep the label visually consistent with the textbook (c).

6.3.4 Broad Camera Re-Localization

After moving object localization, the localization module performs a process called broad camera re-localization. There are many different types of camera movements that would cause tracking loss if the system only localized the camera with keyframe-based localization. For example, the camera may pan in on a mapped portion of the environment and then rotate towards a different mapped portion of the environment. In this scenario, keyframe-based localization may fail as the current keyframe likely would not be connected to the map points that are currently in-view. So, the primary goal of broad camera re-localization is to rectify this limitation by matching features from the current frame to map points that have been triangulated in the broader map, rather than confining feature matching to the points viewed by a single keyframe.

To fulfill this task, the triangulated map points used in keyframe-based localization are gathered along with their neighboring map points. As mentioned in Section 6.3.2, neighboring map points are extracted quickly with the partition table, which indexes triangulated map points by their discretized position in the map. Using this larger pool of triangulated map points, the estimated camera pose is used to compute the projected locations of each point. Any points whose projections fall outside of the viewable image space are pruned to maximize the efficiency of the following steps of the process.

After pruning points that are predicted to fall off-screen, the principal descriptors of each map point are extracted. Using the information associated with a given principal descriptor, the system computes the normal vector of the descriptor's viewing direction as well as the projected viewing direction of the corresponding map point in the current frame. Since descriptor values are subject to change under significant viewpoint changes, the map point is then pruned if there is at least a 60° difference between the descriptor normal and the current viewing direction of the map point.

Once the map points are pruned based on the consistency of their viewing directions, they are pruned once more using a form of non-maximum suppression in which map points with higher tracking frequency are prioritized over those with low tracking frequency. This step is useful for both reducing the computation cost of feature matching and avoiding

low-quality points that are likely to eventually be pruned by the concurrent point pruning process, described in Section 6.4.2.

Guided ORB matching is performed on these finalized map points by constraining the feature search of each map point to the area near its predicted feature location. With guided matching, a much weaker matching threshold of 100 is used, as mismatches are less likely due to the feature localization constraint. To confirm a high-quality match, a Lowe ratio test [61] with a threshold of 0.8 is used to filter out matches in which the quality of the best and second-best matches are too similar. Specifically, this ratio, r, is calculated with

$$r = \frac{d_0}{d_1} \tag{6.3}$$

where d_0 is the hamming distance between the principal descriptor and its closest localized match, and d_1 is the hamming distance between the principal descriptor and the second-closest localized match. If the resulting ratio exceeds 0.8, the match is thrown out as to avoid uncertain matches.

After completing guided ORB matching, these remaining matches are used to update the recorded tracking frequency of the searched map points. Specifically, for each of the map points that were searched for, it is noted that the map point either *matched* to a feature in the current frame (as expected), or *failed* to be matched to a feature in the current frame, despite being expected to appear. This information is later used in the point pruning process to remove low-quality map points.

Finally, the 3D-to-2D correspondences provided by guided ORB matching are fed back into an EPnP solver in a RANSAC scheme to estimate the 6DoF camera pose. Once this camera pose is refined with an iterative minimization of the reprojection error, it replaces the old camera pose estimate.

6.3.5 Failure Evaluation

The previous four processes constitute the entirety of LUMO-SLAM's localization procedures. After these localization processes have concluded, the system evaluates a number of criteria to determine if the final camera localization estimate is sufficient, or if the sys-

tem failed to provide a reasonable estimate. If this failure evaluation determines that the localization estimate is insufficient, then the pose estimate is not forwarded to higher level applications and the localization module skips the remaining processes, starting over on the next frame.

To evaluate the sufficiency of the camera's localization estimate, there are a number of criteria that are assessed. The first criteria involves comparing the localization estimate with a prediction of the camera pose. To do this, the system keeps track of a constant velocity motion model for the camera pose on each frame. This simple model is used to predict the camera's pose based on its pose in the last tracked frame with the added camera velocity (scaled by the number of frames that have passed since the velocity was last computed). If the localization estimate is not within 10 units[3] of this predicted pose, then the process halts and concludes that camera localization has failed for this frame.

If the localization estimate is sufficiently similar to the motion model's estimate, then the inlier rate from keyframe-based camera localization and total number of tracked map points are used to evaluate the quality of the localization estimate. Specifically, if the ratio of inliers to total correspondences falls below 0.6, then the system deems the localization estimate to be insufficiently poor. Additionally, if the number of tracked map points (triangulated map points that were registered in the current frame) is less than 10, the system also deems the localization estimate to be poor, and skips to the next frame.

Given that each of the above conditions pass, the final evaluation of this process is a cheirality check. Put simply, if less than 50% of the triangulated map points in the image are projected to be in front of the image (based on the camera localization estimate), then the system qualifies the estimate as poor, and subsequently halts the process and skips to the next frame.

6.3.6 Point Triangulation

If all quality checks from the previous process pass, then the localization module moves on to a point triangulation process that grows the map by calculating the position of un-

[3]↑A "unit" in LUMO-SLAM's map equates to the baseline between the first two keyframes inserted as a result of map initialization.

triangulated map points. Keyframe-based camera localization often matches ORB features in the current frame to ORB features that have not yet been triangulated from the current keyframe; these correspondences are triangulated in this process, barring that they are ill-conditioned.

Before a correspondence is triangulated, the epipolar constraint (described in Section 3.4.2) is used to check if the point is a mismatch, making it unsuitable for triangulation. To do this, the prospective triangulating camera poses are selected. For point triangulation, the current frame's pose is used as the primary camera and the secondary camera is selected from the collection of keyframes associated with the map point. Specifically, the keyframe farthest from the current frame is used as the secondary camera in order to maximize the baseline between the cameras. Once the camera poses have been selected, the similarity transformation between the cameras, $\mathbf{T}_{ji}$, is computed with

$$\mathbf{T}_{ji} = \mathbf{T}_j \mathbf{T}_i^{-1} \tag{6.4}$$

where $\mathbf{T}_i$ is the primary camera and $\mathbf{T}_j$ is the secondary camera. This similarity transformation is used to compute the essential matrix, $\boldsymbol{E}$, directly with Equation 3.9. The essential matrix is then converted to the fundamental matrix, $\mathbf{F}$, with the following adaptation of Equation 3.13:

$$\mathbf{F} = (\mathbf{K}^{-1})^\top \boldsymbol{E} \mathbf{K}^{-1} \tag{6.5}$$

.

Note that this formulation uses $\mathbf{K}^{-1}$, the inverse of the known intrinsic camera parameter matrix. With the fundamental matrix, the epipolar constraint from Equation 3.14 is evaluated with the projected image points. If the product $\mathbf{x}'^\top \mathbf{F} \mathbf{x}$ exceeds a value of 1, then the correspondence is considered a mismatch and is subsequently rejected from the rest of the triangulation process.

If the correspondence sufficiently upholds the epipolar constraint, then a 3D point is triangulated for the map point using the approach described in Section 3.5, using the primary

and secondary poses specified above, along with their respective image feature locations for the map point.

After triangulating the map point, the reliability of this estimate is evaluated with a number of criteria similar to that used in map initialization. A triangulated point is considered reliable if its parallax with the triangulating cameras exceeds 1°, its reprojection error is less than 4 pixels in each of the triangulating cameras, and the point upholds the cheirality constraint with both of the triangulating cameras. If a triangulated point is considered reliable, it is linked to its corresponding map point and added to the map.

Once the process has triangulated each point, it refines the estimates by performing a small-windowed bundle adjustment on the map. This small window consists of the current frame and the current keyframe to keep the computation time within realtime constraints.

6.3.7 Keyframe Insertion

The last process of the localization module is keyframe insertion. This process is not only performed in order to maintain consistent camera localization as the camera moves, but it is also used to grow the map and facilitate map optimization with bundle adjustment.

By the time the localization module reaches this step, the quality of the estimated camera localization is assumed to be high. With that assumption, the current frame may be used to register a new keyframe in the map. A new keyframe consists of the keypoints and descriptors from the current frame, references matching those keypoints and descriptors to existing map points, new untriangulated map point objects for the unmatched keypoints, and the estimated camera pose for the current frame. To prevent overloading the system with excessive data and slowing down the frametime, the current frame is conditionally used to generate a new keyframe if there are fewer than 70 feature matches with the current keyframe or if the median endpoint disparity of these matches exceeds 3.5% of the frame width. These criteria indicate that the viewpoint is changing significantly and the system's ability to deduce feature matches is waning. Inserting a new keyframe into the map effectively revives the system's ability to make strong feature matches.

After the keyframe insertion process is complete (regardless of whether or not a keyframe is inserted), the localization model submits its relevant data, such as the estimated camera pose and moving object poses, to the output buffer for higher level applications to make use of.

6.4 Map Optimization

When triangulating new points and inserting new keyframes, the localization module prioritizes maintaining its ability to consistently localize the camera throughout the runtime of the system. The consequence of this is that points and keyframes are often inserted as soon as possible, and the accuracy of their placement may be lacking as the map continues to grow.

To rectify this shortcoming, the map optimization module provides a number of independent, concurrent threads, each of which address a different map refinement process. All of these processes are bootstrapped upon the completion of map initialization and they do not terminate until the containing application shuts down with the rest of the system.

Each of these processes (which, unlike each process from the previous modules, runs concurrently to the other processes) is described in detail throughout the following sections.

6.4.1 Principal Descriptor Selection

As a map point is connected to additional keyframes, its collection of associated descriptors grows. Realtime matching is not feasible in the broad camera re-localization process if the current frame's features must be compared to every descriptor associated with a given map point. The volume of descriptors even grows so rapidly that attempting to filter through the descriptors with other criteria (such as viewing direction) is not practical under realtime constraints either. To rectify this, a single descriptor is appointed among the complete collection of a given map point's descriptors to act as the point's principal descriptor. This descriptor is then used for the map point when performing ORB matching in the broad camera re-localization process.

To select a map point's principal descriptor, this process runs a loop that frequently updates the principal descriptors for any map points that have recently become associated with new keyframes (and thus, new descriptors as well). However, since map points may need a principal descriptor before this process has had a chance to select one for them, the default principal descriptor for all new map points is simply configured to be the first descriptor that was associated with the map point. This remains as the principal descriptor for the map point until this process selects a new descriptor for the point.

To select the principal descriptor for a specific map point, the system compares each of the map point's associated descriptors to each of the other associated descriptors. Specifically, the hamming distance is computed for every combination of descriptor pairings. For each descriptor, the values of its hamming distances with every other descriptor are then sorted in ascending order and then the median of these values is selected to quantify the descriptor's likeness to the others. The descriptor whose median hamming distance is the lowest is then appointed as the principal descriptor for the map point. This approach is largely inspired by ORB-SLAM [18], and an example of this process can be seen demonstrated in Figure 6.5.

Once a principal descriptor is selected for a map point, the map point begins to use this descriptor during the broad camera re-localization process. The principal descriptor for a given map point will only be re-evaluated (and potentially updated) if the map point becomes associated with new keyframes in future frames.

6.4.2 Point Pruning

As LUMO-SLAM is generous in its triangulation of new map points, it is clear that point pruning is necessary to enforce high map quality. Without pruning, the map will become oversaturated with triangulated points that largely go unused. This hinders the system's ability to quickly identify points that are useful for localization. Furthermore, poorly-triangulated map points, if erroneously used for localization, can cause the system to fail outright. The concurrent point pruning process runs alongside the rest of the system to rectify these issues by evaluating the quality of triangulated points and pruning those that may be a liability to the system.

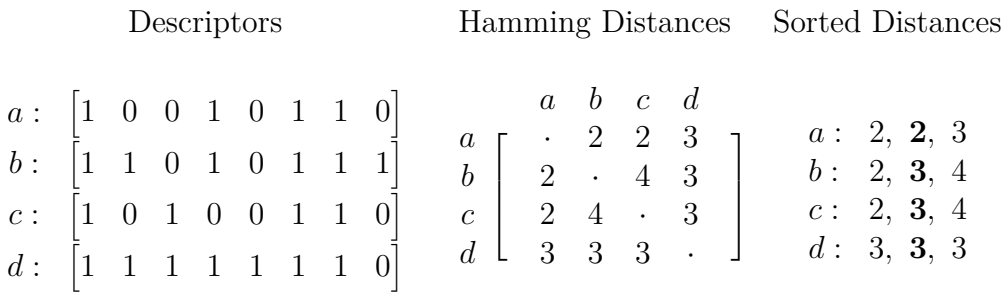


Figure 6.5. An example of the principal descriptor selection process for a map point that holds four descriptors, a, b, c, and d. Note that, in practice, the ORB descriptors are 256 bits long; however, the descriptors in this example are only 8 bits for demonstration. The hamming distances are recorded for each pair of descriptors and, in this demonstration, are organized in a matrix. For each descriptor, the values of its corresponding row in the matrix are extracted and sorted. From these sorted distances, the median is used to compare each descriptor. Descriptor a is selected as the principal descriptor in this example, as it has the lowest median distance to the other descriptors (a value of 2).

This process performs pruning in a manner similar to that seen in ORB-SLAM [18]. Specifically, the process continually evaluates a queue of newly-triangulated map points, which is repeatedly updated as new points are triangulated. When the system evaluates a point from the queue, it first checks that the point has been given sufficient opportunity to strongly integrate into the map. Thus, if fewer than five keyframes have been inserted since a given point was triangulated, then the system skips the point on this iteration and leaves it in the queue. Otherwise, it evaluates the corresponding map point for potential pruning.

There are two criteria used to determine if a map point needs to be removed from the map. Firstly, the map point will be removed if it is observed by fewer than 3 keyframes. If this is the case, it is assumed that the point is weak, as it is unlikely to be viewed in future. Secondly, the map point will be removed if it is tracked less than 25% of the time that it has been expected to be tracked during broad camera re-localization. It follows that, if the system is repeatedly projecting a point into the image and cannot find it, then the point's use is limited.

Barring these two conditions, a map point will be deemed high quality and is consequently removed from the queue, signifying its permanent membership of the map.

6.4.3 Short-Range Windowed Bundle Adjustment

An important process for maintaining high reconstruction quality for recently-triangulated map points is short-range windowed bundle adjustment. As explained in Section 3.7, bundle adjustment is a non-linear minimization of the reprojection error of a collection of 3D points and the cameras that have imaged them. This optimization problem is solved with a Levenberg-Marquardt solver, which optimizes the map's triangulated map points and keyframes by using their corresponding keypoint locations as constraints for this minimization problem.

With an increasingly large pool of points and keyframes populating the map, bundle adjustment becomes infeasible under realtime constraints if performed on the entire map. However, bundle adjustment can be repeatedly performed on a subset of the map and still rapidly provide meaningful refinement.

To perform short-range windowed bundle adjustment, the system repeatedly collects the 10 most-recent keyframes and all of their shared triangulated map points. Reprojection errors for each of the point-to-keyframe associations are computed and points with extremely high reprojection errors (more than 50 pixels) are retriangulated if possible (in case the high error is the result of a previous, bad bundle adjustment), or pruned from the map otherwise. This pruning is important, as just a few substantial outliers can cause the entire bundle adjustment process to fail.

After this pruning step, the remaining points and keyframes are optimized with 10 iterations of bundle adjustment. With the optimization confined to this small subset of the map, refinement can be performed approximately realtime. So, though a currently-visible subset of the map is being modified, the modification occurs too quickly to cause visible disruption in the eventual image augmentation (assuming that LUMO-SLAM is being used for AR).

6.4.4 Long-Range Windowed Bundle Adjustment

As short-range windowed bundle adjustment is responsible for refining the quality of recently-triangulated points quickly, long-range windowed bundle adjustment is responsible for refining a substantially larger view of the map. This is particularly useful for sequences with loops, as they are more likely to reap the benefits of this optimization upon revisiting past sections of the map.

In contrast to the bundle adjustment process described in the previous section, long-range windowed bundle adjustment isolates the first 60 keyframes of the map for refinement. Since the optimization process will be significantly slower with this larger set of data, this process also maintains a 15-keyframe distance from the current (most-recently inserted) keyframe to prevent map updates from being noticeably visible in image augmentation. If the system determines that the last keyframe in the current window is too close to the current keyframe (in terms of the number of keyframes separating the two), then the process simply stalls, waiting for more keyframes to be inserted to maintain the appropriate following distance.

If the keyframe window upholds the required following distance, then bundle adjustment is performed on these 60 keyframes and all of their shared map points for 20 iterations.

Additionally, this process also fixes the location of the first and last keyframes in the window as to prevent fragmentation between optimized and un-optimized sections of the map over time.

After updating the optimized map points and keyframe poses, this process shifts the keyframe window down 30 keyframes (half of the window size) and repeats the process for the next 60 keyframes. With these large windows, long-range windowed bundle adjustment requires several seconds to optimize a batch of data; however, its contribution to the improvement of the map and localization accuracy is indubitably invaluable.

6.4.5 Loop Closing

As small inaccuracies accumulate in the system's map point and keyframe estimates, the camera localization estimates can suffer from noticeable "drift" if the camera revisits a previously-mapped area. The standard approach to solving this problem is to detect when drift may have occurred and subsequently correct it. This process is known as loop closure.

To implement loop closure, this concurrent process compares every new keyframe to all previous keyframes. As the map may grow to a substantial size, it is necessary to perform this loop detection process with an approach that is faster than unguided feature matching to ensure that the process does not fall behind the rate at which new keyframes are inserted. LUMO-SLAM instead compares the likeness of keyframes by comparing a hierarchical bag-of-words (BoW) vector for each keyframe, which summarizes the descriptors associated with its respective keyframe. This approach is based off of the work presented in [95], in which binary descriptors are collected from a large set of training images and then recursively clustered with k-medians clustering to develop a tree structure to efficiently index new descriptors. LUMO-SLAM borrows the BoW descriptor vocabulary developed in [96], in which the vocabulary is built in the same fashion described in [95], but with ORB descriptors from the Bovisa 2008-09-01 dataset [97] instead of regular BRIEF descriptors.

When loop detection is performed on a given keyframe, the system computes the Euclidean distance between the keyframe's BoW vector and the BoW vectors of all previous keyframes. Given these BoW distances, unguided ORB matching is performed between the

given keyframe and each of the 10 most-similar keyframes to gather stronger criteria for determining loop existence. Going through the most-similar keyframes first, if any of these top 10 keyframes result in 50 or more ORB feature matches, then the system considers there to be a loop between the given keyframe and the matching keyframe. Subsequently, the system begins the loop closure process between these keyframes.

The first step of LUMO-SLAM's loop closure process is to merge the matching map points between the "dangling keyframe" (the given keyframe that was evaluated for loop existence) and the "ground truth keyframe" (the older keyframe that the dangling keyframe matched to). To do this, the matching map points of the dangling keyframe are updated to point to the original map points, which are referenced by the ground truth keyframe.

Once map point merging is complete, the sequence of keyframes separating the ground truth keyframe from the current keyframe as well as all of their associated triangulated map points are used to develop constraints that will later be used for optimizing the map after the pose of the dangling keyframe is corrected. Specifically, there are two primary pieces of data that are computed for these constraints. The first piece of data is the similarity transformation between pairs of keyframes that are adjacent to each other in the aforementioned keyframe sequence. The similarity transformation, $\mathbf{T}_{\mathrm{ji}}$, between a keyframe at position i and an adjacent keyframe at position $\mathrm{j} = \mathrm{i} + 1$, can be modeled in SE(3) as

$$\mathbf{T}_{\mathrm{ji}} = \mathbf{T}_{\mathrm{j}}\mathbf{T}_{\mathrm{i}}^{-1} \tag{6.6}$$

where $\mathbf{T}_{\mathrm{i}}$ and $\mathbf{T}_{\mathrm{j}}$ are the poses of the keyframes at positions i and j, respectively. However, rather than using the SE(3) formulation for this calculation, this optimization solution uses LUMO-SLAM's internal representation of poses, which consists of a versor (unit quaternion), $\mathbf{q}$, to represent the pose's rotation and a translation vector, $\mathbf{t}$, to represent the pose's translation, which is applied *before* the rotation. The computation of the *rotation* of the similarity transformation, $\mathbf{q}_{\mathrm{ji}}$, is then computed with

$$\mathbf{q}_{\mathrm{ji}} = \mathbf{q}_{\mathrm{j}} \circ \mathbf{q}_{\mathrm{i}}^{-1} \tag{6.7}$$

where $\mathbf{q}_\mathrm{j}$ is the rotation component of the keyframe at position j, $\mathbf{q}_\mathrm{i} = (q_w, q_x, q_y, q_z)$ is the rotation component of the keyframe at position i, $\mathbf{q}_\mathrm{i}^{-1} = (-q_w, q_x, q_y, q_z)$ is the quaternion inverse of $\mathbf{q}_\mathrm{i}$, and $\circ$ denotes a composition operation between the quaternions, implemented with the Hamilton product.

The translation component of the similarity transformation, $\mathbf{t}_\mathrm{ji}$, can then be easily computed with

$$\mathbf{t}_\mathrm{ji} = (t_{x\mathrm{j}} - t_{x\mathrm{i}},\ t_{y_\mathrm{j}} - t_{y_\mathrm{i}},\ t_{z\mathrm{j}} - t_{z\mathrm{i}}) \tag{6.8}$$

where t_{wk} indicates the w^{th} translation component of the k^{th} keyframe pose. Thus, the loop closing process records similarity transformations in the form of $(\mathbf{q}_\mathrm{ji}, \mathbf{t}_\mathrm{ji})$ for each pair of adjacent keyframes in between the ground truth keyframe and current keyframe. These values are ultimately used in the eventual optimization step.

The second piece of data that must be recorded is the set of point transformations between triangulated map points and their associated keyframes. To specify, each triangulated point in the keyframe sequence, $\mathbf{X} = [\ x\ y\ z\ 1\ ]^\top$, is paired with the pose of a keyframe that observes it, $\mathbf{T}$. The point's position with respect to the keyframe, $\mathbf{X}'$, is computed with

$$\mathbf{X}' = \mathbf{T}\mathbf{X} \tag{6.9}$$

.

Though this example is demonstrated with the SE(3) Lie group, the system instead opts to use the versor/vector formulation, like in the computation of similarity transformations. A point's transformed position, (x', y', z'), with respect to a given keyframe pose, $(\mathbf{q}, \mathbf{t})$, can be computed with the following versor/vector formulation:

$$(0,\ x',\ y',\ z') = \mathbf{q} \circ (0,\ x + t_x,\ y + t_y,\ z + t_z) \circ \mathbf{q}^{-1} \tag{6.10}$$

Note, in this formulation, (x, y, z) represents the point's world-space position while (x', y', z') represents the point's position with respect to the keyframe pose, $(\mathbf{q}, \mathbf{t})$.

Though every possible pairing between triangulated map points and their observing keyframes' poses could be computed to maximally constrain the optimization, this may not be practical if the size of the keyframe sequence is excessively large. So, to save on computation cost, point transformations are computed for at most 10 observing keyframes for any given point.

After these values are computed, the map correction process begins. The first step of this process is to accurately re-localize the dangling keyframe by feeding the correspondences associated with its newly-merged map points into a Levenberg-Marquardt solver. The system also uses the pose of the ground truth keyframe (rather than that of the dangling keyframe) to initialize this optimization reliably, as loop detection indicates that the ground truth keyframe's pose is close to the true pose for the dangling keyframe. Once the dangling keyframe's pose is corrected, it is locked for the remainder of the process.

At this point, the drift in the keyframe position has simply been shifted back by a single keyframe. To morph the remainder of the keyframe sequence (and its associated map points) into the correct state, a two-step optimization process occurs using the values computed previously.

First, the keyframe poses are corrected by minimizing the error in the similarity transformations between corresponding pairs of adjacent keyframes. This takes the form

$$\min_{\mathbf{q}_\mathrm{i},\mathbf{q}_\mathrm{j}\forall(\mathrm{i},\mathrm{j})} \sum_{(\mathrm{i},\mathrm{j})} \left\| \mathbf{q}_\mathrm{j} \circ \mathbf{q}_\mathrm{i}^{-1} - \mathbf{q}'_\mathrm{ji} \right\|^2 \tag{6.11}$$

where $\mathbf{q}'_\mathrm{ji}$ is the previously-computed rotation component of the similarity transformation from keyframe i to keyframe j, and $\mathbf{q}_\mathrm{j} \circ \mathbf{q}_\mathrm{i}^{-1}$ is the value of the rotation component using the current keyframe poses. Additionally, the objective function for the translation components takes the form

$$\min_{\mathbf{t}_\mathrm{i},\mathbf{t}_\mathrm{j}\forall(\mathrm{i},\mathrm{j})} \sum_{(\mathrm{i},\mathrm{j})} \left\| \mathbf{t}_\mathrm{j} - \mathbf{t}_\mathrm{i} - \mathbf{t}'_\mathrm{ji} \right\|^2 \tag{6.12}$$

where $\mathbf{t}'_{\mathrm{ji}}$ is the previously-computed translation component of the similarity transformation from keyframe i to keyframe j, and $\mathbf{t}_{\mathrm{j}} - \mathbf{t}_{\mathrm{i}}$ is the value of the translation component using the current keyframe poses.

These objective functions are solved with a gradient descent approach that automatically tunes the associated hyperparameter, α, by evaluating the translation error each on step. With a typical loop size of 270 keyframes, this optimization requires approximately 30 to 40 milliseconds when performed on a high-end desktop computer.

After the keyframe poses are corrected, their associated map points are corrected by minimizing the error in the transformed point positions between map points and their observing keyframes. The objective function for this problem takes the form

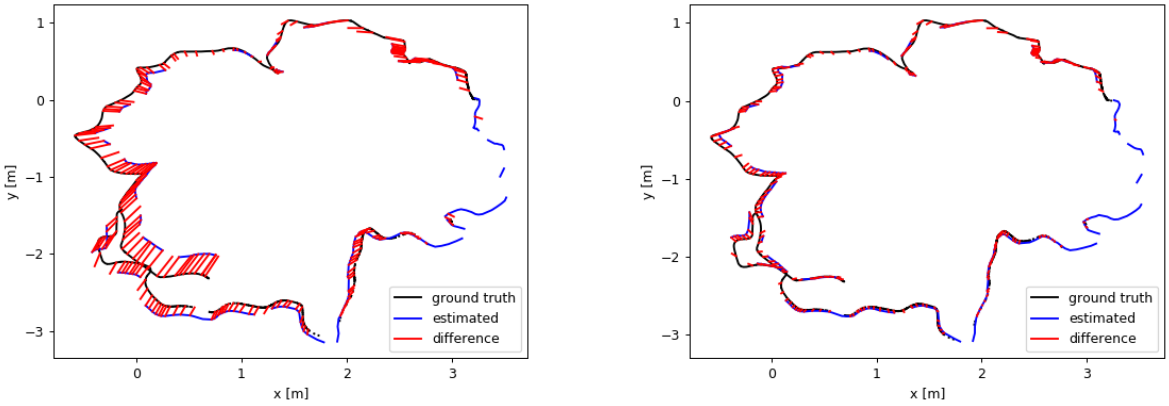
$$\min_{x_{\mathrm{i}},y_{\mathrm{i}},z_{\mathrm{i}}\forall \mathrm{i}} \sum_{(\mathrm{i},\mathrm{j})} \left\| \mathbf{q}_{\mathrm{j}} \circ (0,\ x_{\mathrm{i}} + t_{x\mathrm{j}},\ y_{\mathrm{i}} + t_{y_{\mathrm{j}}},\ z_{\mathrm{i}} + t_{z\mathrm{j}}) \circ \mathbf{q}_{\mathrm{j}}^{-1} - \mathbf{p}'_{\mathrm{ij}} \right\|^2 \tag{6.13}$$

where $\mathbf{p}'_{\mathrm{ij}}$ is the previous transformed point position for point i in keyframe j and $\mathbf{q}_{\mathrm{j}} \circ (0,\ x_{\mathrm{i}} + t_{x\mathrm{j}},\ y_{\mathrm{i}} + t_{y_{\mathrm{j}}},\ z_{\mathrm{i}} + t_{z\mathrm{j}}) \circ \mathbf{q}_{\mathrm{j}}^{-1}$ is the value of the transformed point position using the current values for point i and keyframe j.

This optimization problem is also solved with gradient descent and uses the same α value computed from the keyframe optimization process. A typical indoor environment may generate upwards of 50,000 point transformations to use for this optimization when loop closure is performed. In practice, this takes approximately 80 to 90 milliseconds under the same conditions as described above.

Once these optimizations have morphed the map into a state that is more consistent with the observed matches between the ground truth keyframe and current keyframe, the loop has been closed. The use of these optimization approaches allow the map to retain the refinement gained from any previous bundle adjustment processes that have been performed; this is in contrast to simply retriangulating map points after the drift is corrected.

To conclude the loop closure process, 10 iterations of fixed-window bundle adjustment are run on the last 10 keyframes in the keyframe sequence along with the ground truth keyframe and all triangulated map points shared between them. After the loop closing process finishes,



(a) Without loop closure

(b) With loop closure

Figure 6.6. Top-down trajectory-comparison graphs for two runs of the freiburg2_desk sequence from the TUM RGB-D dataset [94], one *without* loop closure enabled (a) and one *with* loop closure enabled (b). For each chart, the ground truth trajectory is denoted in black, the estimated trajectory is denoted in blue, and the error between them is denoted in red. The sequence begins and ends on the left side of each chart. Note that the errors are pronounced at the end of the sequence when the system does not perform loop closure, while the system is substantially more accurate when making use of loop closure.

a brief 30-keyframe countdown is implemented to block loop closure from being performed excessively as the camera continues to explore the previously-visited area.

In total, this loop closure process typically takes between 300 and 400 milliseconds on a high-end desktop computer. This efficiency is necessary for permitting realtime map use for the other processes in the system. The substantial impact of this loop closing approach is demonstrated in Figure 6.6.

7. EXPERIMENTAL RESULTS AND ANALYSIS OF LUMO-SLAM

To evaluate the overall performance of the LUMO-SLAM system described in the previous chapter, experimental results are compiled for several sequences to illustrate the system's capabilities and limitations. The following experiments are organized into three categories: (1) experiments that evaluate the camera localization accuracy of LUMO-SLAM, (2) experiments that quantify the accuracy of the moving object localization estimates that LUMO-SLAM provides, and (3) experiments that illustrate LUMO-SLAM's capacity to accurately register unknown moving objects.

To evaluate standard SLAM accuracy, the absolute trajectory error of the system is evaluated for several standard SLAM sequences and compared to a number of state-of-the-art SLAM systems. In order to evaluate the accuracy of LUMO-SLAM's moving object localization estimates, custom sequences with ground truth moving object data are used to compare LUMO-SLAM's estimates against real-world measurements. Finally, LUMO-SLAM's moving object registration capabilities are assessed with several custom sequences that contain moving objects under varying conditions.

7.1 Standard SLAM Accuracy

The most fundamental assessment for any visual SLAM system is the evaluation of the system's camera localization accuracy, as estimating the camera pose is the primary function of SLAM systems. To evaluate the accuracy of LUMO-SLAM's camera localization estimates, the system is run on several sequences from the popular TUM RGB-D [94] dataset[1] and its estimates are compared with the ground truth measurements provided by the dataset.

The metric used to quantify the system's accuracy is absolute trajectory error (ATE). ATE is a popular and straightforward metric for evaluating SLAM accuracy. The ATE between two sequences of corresponding camera positions is computed by minimizing the root mean square error (RMSE) between the positions of each corresponding pair of cameras

[1]↑Though the dataset from [94] provides RGB-D data, the depth data is not used in the evaluation of LUMO-SLAM's accuracy, as LUMO-SLAM is a monocular SLAM system.

along their respective trajectories. This optimization is typically performed with Horn's method [98] or Arun's method [99], which both fit one trajectory to the other with seven degrees of freedom (7DoF). When this optimization is used to fit an estimated trajectory to a ground truth trajectory, the resulting RMSE is observed as the ATE between them.

The ATE is computed for LUMO-SLAM's keyframe trajectory estimates using the tool provided by [94], which utilizes Horn's method to fit the estimated trajectory to the ground truth. Though this approach isn't fully robust against outliers [100], the results computed do not indicate that the use of ATE or Horn's method is troublesome for the estimates provided by LUMO-SLAM. These results are displayed in Table 7.1.

To contextualize the results, the data presented in [18] is forwarded to Table 7.1 so that LUMO-SLAM can be compared to other state-of-the-art visual SLAM systems. These systems include ORB-SLAM [18], PTAM [16], LSD-SLAM [20], and RGB-D SLAM [101], [102]. Additionally, for each system, every sequence is processed five times and the results provided in the table denote the median ATE for each set of runs.

Given the results shown in Table 7.1, it is clear that, though LUMO-SLAM's camera localization accuracy falls short of the results achieved by ORB-SLAM and PTAM, it is still usefully accurate, as its results rival those of LSD-SLAM. This observation is interesting, as bundle adjustment is typically credited with causing increased accuracy in SLAM systems which utilize it; however, LUMO-SLAM does not achieve the same accuracy as the other systems that also make extensive use of bundle adjustment.

To hypothesize the cause of this reduced accuracy, it is likely that LUMO-SLAM's camera localization results are hindered by the system's separation of feature extraction from feature matching. Both ORB-SLAM and PTAM perform true guided searches for map points when performing camera localization. That is, these systems *start* their localization processes by deducing predicted feature locations in the image, and then begin evaluating *all* of the pixels in each feature's predicted image patch to deduce the best match. LUMO-SLAM, on the other hand, performs feature extraction first and largely ignores any other data from the image after the features have been blindly extracted. These features are then used in both unguided and guided matching; however, this causes a likely-meaningful difference between LUMO-SLAM and these predecessors. When LUMO-SLAM performs guided searching for

Table 7.1. Median absolute trajectory errors comparing LUMO-SLAM, ORB-SLAM, PTAM, LSD-SLAM, and RGB-D SLAM with the TUM RGB-D dataset. Results for ORB-SLAM, PTAM, LSD-SLAM, and RGB-SLAM are forwarded from the work presented in [18]. Results are aligned to the provided ground truth with 7DoF before computing error. Note, "X" indicates tracking loss that causes significant portions of the sequence to go unprocessed while "-" indicates missing results, as RGB-D SLAM results are provided from the benchmark website for only a subset of the sequences evaluated here.

	Absolute Keyframe Trajectory RMSE Error (cm)				
Sequence	**LUMO-SLAM**	**ORB-SLAM**	**PTAM**	**LSD-SLAM**	**RGB-D SLAM**
fr1_xyz	3.50	0.90	1.15	9.00	1.34
fr2_xyz	4.85	0.30	0.20	2.15	1.42
fr1_floor	7.11	2.99	X	38.07	3.51
fr1_desk	5.98	1.69	X	10.65	2.52
fr2_360_kidnap	40.72	3.81	2.63	X	100.5
fr2_desk	23.24	0.88	X	4.57	3.94
fr3_long_office	24.43	3.45	X	38.53	-
fr3_nstr_tex_far	20.88	(ambiguity detected)	4.92/34.74	18.31	-
fr3_nstr_tex_near	41.79	1.39	2.74	7.54	-
fr3_str_tex_far	3.29	0.77	0.93	7.95	-
fr3_str_tex_near	6.42	1.58	1.04	X	-
fr2_desk_person	24.01	0.63	X	31.73	2.00
fr3_sit_xyz	8.42	0.79	0.83	7.73	-
fr3_sit_halfsph	10.47	1.34	X	5.87	-
fr3_walk_xyz	12.48	1.24	X	12.44	-
fr3_walk_halfsph	30.69	1.74	X	X	-

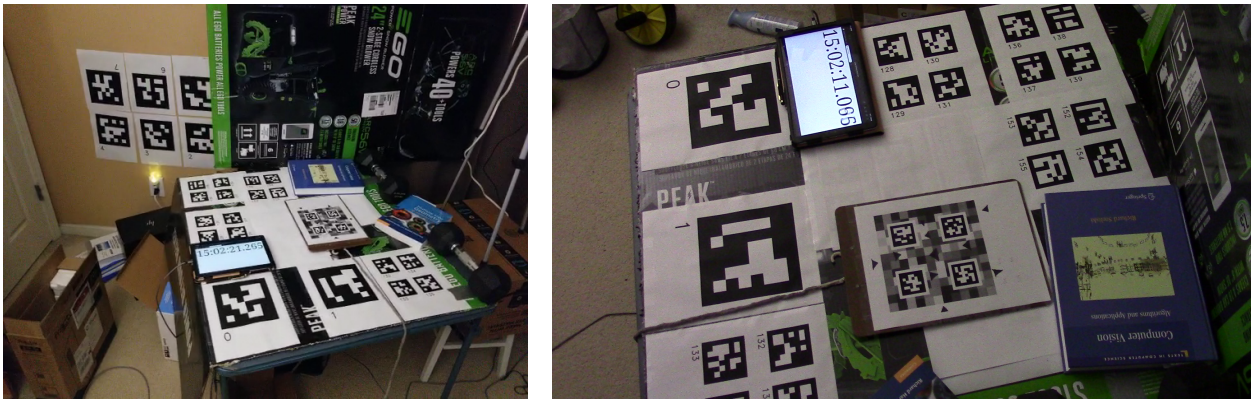
image features, it ignores a majority of the search patch and only evaluates a handful of features for each map point. It is likely that this is introducing noise into the feature matches, as the few features provided by the image processing stage may not always include the optimal feature location for each relevant map point. This presents a new problem for exploration, as a SLAM system which provides unknown moving object registration is ill-suited for the guided feature matching approaches of ORB-SLAM and PTAM, since the projected image locations of moving object features are likely to be unpredictable.

Despite this limitation in LUMO-SLAM's camera localization accuracy, the overall quality of its estimates here is still usefully high. These results indicate that the algorithm implemented by LUMO-SLAM is sufficient for providing realtime camera localization and the comparison with other state-of-the-art systems also informs areas for future work and improvements.

7.2 Moving Object Localization Accuracy

In addition to evaluating standard SLAM metrics, such as the ATE of the camera localization estimates, it is also important to quantify the accuracy of LUMO-SLAM's localization estimates for moving objects. This task is challenging, as there are few standard, public datasets which include ground truth values for moving objects. Additionally, as LUMO-SLAM is a proof-of-concept system and its current implementation still requires certain conditions to be met in order to successfully register and localize moving objects in the real world (which is explored in the next section), the few public datasets that provide this ground truth information are often ill-suited for use with LUMO-SLAM.

To rectify this, a custom scene that makes extensive use of ArUco fiducial markers [103] is built. This scene contains several static fiducial markers of varying sizes scattered across a table as well as the wall behind it. It also includes a small set of markers taped to a clipboard, which acts as the moving object in the following experiments. Given the known real-world distance between a pair of the static markers, a custom-built offline marker-based mapping application analyzes a video of the scene to accurately map the position of the markers at real-world scale. This mapping can then be used to accurately localize the object as it is



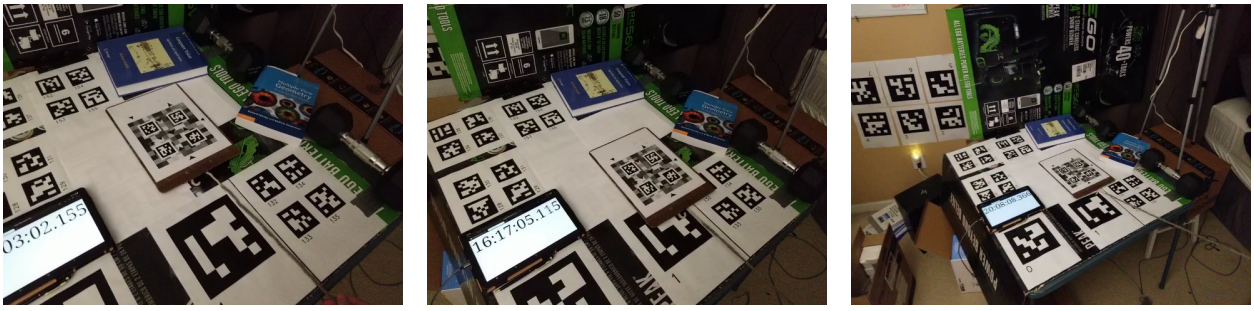
(a) Prepared environment (b) Static camera view

Figure 7.1. Prepared, marker-based environment used for collecting ground truth localization data for a moving object while the LUMO-SLAM system observes the object movement. (a) shows a wide view of the environment, as seen by LUMO-SLAM, while (b) shows the narrow, stationary view captured by the static camera. The view in (b) is used for determining ground truth localization data for the moving clipboard by leveraging the pre-mapped marker locations. This provides localization data at real-world scale that can be compared against LUMO-SLAM's estimates.

moved during any given experiment. To accomplish this, an additional camera is mounted in a static position, observing many static markers in the scene as well as maintaining a constant view of the object. This static camera is eventually used to determine the trajectory of the object at real-world scale. The different views of the static camera and SLAM camera are illustrated in Figure 7.1.

For each experiment, the static camera records the object while the SLAM camera moves freely throughout the environment. During the experiment, the clipboard is moved with a string and is consequently registered and localized by LUMO-SLAM. Once the run has concluded, the static camera's footage is then fed into a custom-built offline localization application which uses the known marker mapping to calculate the object's trajectory with respect to the markers at real-world scale.

This procedure is used to collect three different sequences of the environment containing object movement. When LUMO-SLAM detects a moving object, it is clear that the primary factor that may affect its *object localization accuracy* is the distance that the object is from



(a) Close view (b) Midrange view (c) Far view

Figure 7.2. Example views of each of the three experiments for evaluating moving object localization accuracy. (a) shows the sequence in which the camera maintained a short distance from the moving object (clipboard), (b) shows the sequence in which the camera maintained a moderate distance from the moving object, and (c) shows the sequence in which the camera maintained a large distance from the moving object.

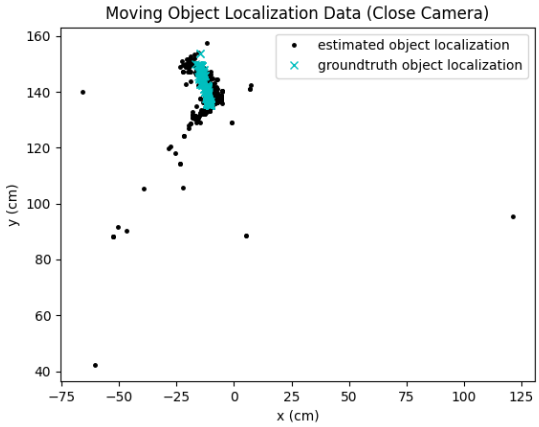
the camera. Thus, each of the three sequences observe the object being moved in the scene at different object-camera distances. The distances at which the camera observes the object during movement are demonstrated in Figure 7.2, which includes a view of the sequence which has a short object-camera distance, a view of the sequence which has a midrange object-camera distance, and a view of the sequence which has a large object-camera distance.

After recording each of these sequences and computing the ground truth object trajectory with the static camera's footage, LUMO-SLAM's estimated object trajectory is fit to the ground truth object trajectory with 7DoF. The resulting object ATEs are computed for each sequence and displayed in Table 7.2. Visualizations of the object trajectories (both ground truth and estimated) are also provided in Figure 7.3.

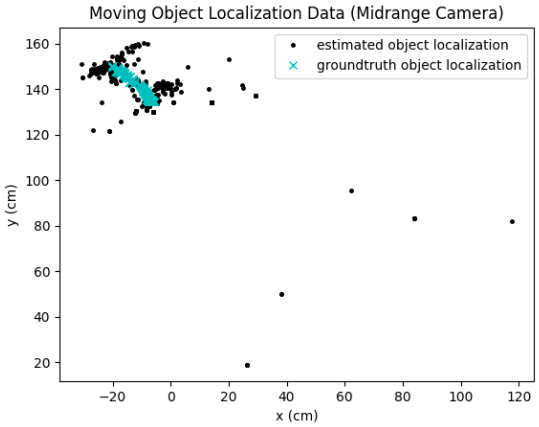
The results predictably indicate that the system tends to provide more accurate object localization estimates when the object is closer to the camera, as the median object ATE is 5.78 cm during the close-camera sequence, 7.86 cm during the midrange-camera sequence, and 30.95 cm during the far-camera sequence. These specific results also indicate a constraint on this approach's applicability: accurate object localization may only be reasonably expected when the object is less than approximately one meter from the camera. This informs potential use cases for this approach. For instance, this approach may be suitable for

Table 7.2. Median absolute trajectory errors (ATEs) for moving objects when fit and evaluated against ground truth data. The errors are organized by the objects' approximate distance from the camera during the sequence. Since the SLAM camera is free-hand and the object movement is not precisely controlled during each take, the object-camera distance is slightly different on each frame of any given sequence; thus, approximate object-camera distances for the whole sequence are provided in parentheses. Note, when the camera is farther away from the object, localization accuracy dwindles. However, localization accuracy is very strong under 100 cm.

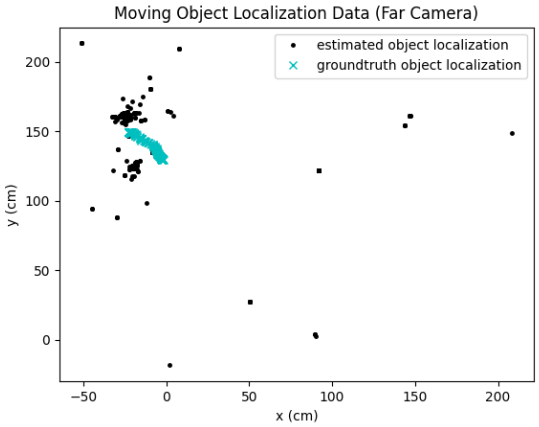
Object-Camera Distance	**Median Object ATE**
Close (∼70 cm)	5.78 cm
Midrange (∼100 cm)	7.86 cm
Far (∼150 cm)	30.95 cm



(a) Close Camera



(b) Midrange Camera



(c) Far Camera

Figure 7.3. Visualizations of the ground truth object movement (shown in blue) and the estimated object movement (shown in black). (a) shows the localization results during the sequence in which the camera is approximately 70cm away from the object, (b) shows the localization results during the sequence in which the camera is approximately 100cm away from the object, and (c) shows the localization results during the sequence in which the camera is approximately 150cm away from the object. Note that, though all cases contain outliers, as the distance from the camera increases, the localization estimates become less stable.

an application that involves a user moving objects on their desk, but not so much for an application that has to estimate the position of neighboring cars on the road that are several meters away.

It is also important to note that object-camera distance is the only factor controlled in these experiments because other reasonable factors (such as lighting conditions, object speed, camera movement, etc.) tend to have a significant impact on object *registration* capability rather than localization accuracy. These factors are instead, consequently, explored in Section 7.3.

7.3 Moving Object Registration Accuracy

Though there are few factors that may have an impact on the accuracy of the moving object localization estimates computed by LUMO-SLAM, there are a significant number of factors that may impact moving object *registration*. For instance, low lighting conditions may make feature matching more challenging, thus hindering the system's ability to register moving objects. Additionally, excessively-fast object motion may cause blurring that makes the matching of moving object features unstable.

To illuminate LUMO-SLAM's capacity for unknown moving object registration, 70 sequences are created, each of which involves moving objects (specifically, one or two textbooks) that satisfy one of seven different, likely-relevant, conditions. Each of the sequences can thus be categorized by the condition it satisfies. These categories include:

- **Fast Motion**: These sequences involve very fast initial movement from the moving object. This condition is relevant, as fast, real-world motion may cause motion blur that affects LUMO-SLAM's ability to accurately match the object's features.
- **Slow Motion**: In contrast to the sequences in the "fast motion" category, these sequences involve very slow initial object movement. This condition is relevant, as slow object movement leaves time for keyframe insertion, which may continually register minor outliers as new map points, causing object registration to fail.

- **High Light**: These sequences act somewhat as a baseline, in which a single moving object is translated through the environment with sufficient lighting. The system is expected to perform best on these sequences.
- **Low Light**: These sequences involve the object moving through the environment as in other cases, except the lighting of the environment is very low. This condition is relevant as low lighting may cause reduced contrast in the image, and consequently may negatively impact feature extraction and matching. This may make moving object registration more challenging.
- **Moving Camera**: These sequences involve camera movement that takes place during the object's initial motion. This contrasts other tests, in which the camera is mostly still during object registration. This condition is relevant, as camera motion may cause repeated keyframe insertion, which may result in duplicate map point generation for moving object points which could cause object registration to fail.
- **Multiple Objects**: These sequences include two moving objects, one of which is moved (and ideally registered) before the other. Though there is no clear reason why this type of experiment would fail, testing these sequences can help validate the fact that this approach can be applied to multiple concurrent moving objects.
- **Rotation-Only**: Most sequences involve object registration via object *translation*; however, these sequences register the object by *rotating* it. This condition is relevant, as rotational movements result in lower reprojection errors *towards* the center of rotation and higher reprojection errors *away* from the center of rotation. This may cause moving objects to fail registration, as points close to the center of rotation may not be registered as outliers.

Each category contains 10 different sequences, resulting in a total of 70 sequences ran through LUMO-SLAM to evaluate its capacity for accurately registering moving objects under these conditions. The results of these runs are summarized in Table 7.3.

As shown in Table 7.3, LUMO-SLAM broadly has high capacity to accurately register moving objects across these conditions. However, it still demonstrates notable weaknesses.

Table 7.3. Successful object registrations performed by LUMO-SLAM in sequences demonstrating varying conditions. The values in the right-hand column indicate the number of times LUMO-SLAM successfully registered the moving object in the respective sequence out of ten runs. The table also includes the results for the sequence containing multiple objects, in which a successful registration is counted when *both* objects were registered and when *at least one* object was registered (denoted in parentheses).

Sequence Type	**Successful Registrations**
fast motion	0 / 10
slow motion	10 / 10
high light	10 / 10
low light	8 / 10
moving camera	6 / 10
multiple objects	7 / 10 (9 / 10)
rotation-only	10 / 10

For instance, LUMO-SLAM failed to register the moving object on any of the sequences in which the object is moved very quickly. This is likely the result of motion blur on the object obfuscating its features. Additionally, the system appears to have reduced capacity to register objects when the camera is moving, registering the object in only 6 out of the 10 related sequences. This is likely caused by continuous keyframe insertion, which in turn likely causes the moving object features to suffer from duplicate map point generation, causing the reprojection errors to be continually truncated. Surprisingly, despite the slightly reduced number of successful registrations (8 out of 10), the system appears to maintain fairly strong registration capability in low light. It is also somewhat surprising that the system demonstrated such strong performance on the rotation-only sequences, as it managed to successfully register the object on all 10 sequences. This is likely partially afforded by the fact that the book's rotation was always parallel to the table it rested on, which caused substantial feature movement across the object. In addition to the rotation-only sequences, LUMO-SLAM also successfully registered the object in all of the high light and slow motion sequences. Finally, the system interestingly demonstrated slightly-reduced success in the multi-object sequences. Specifically, LUMO-SLAM successfully registered the first object on 9 out of 10

of the sequences, but only registered both objects in 7 out of the 10 sequences. Given that the first object was more-consistently registered than the second (and given that the first object is the same book used in the other sequences), it is likely that the second object is simply less suited for moving object registration, as its features may be lower quality.

These results help illustrate the capabilities and limitations of LUMO-SLAM's moving object registration implementation. Ultimately, though the system doesn't have substantial robustness yet, the results indicate that LUMO-SLAM is usefully functional under the right conditions. As in the other experiments, these results also greatly inform areas for future work and improvements.

8. CONCLUSION AND FUTURE WORK

In conclusion, the dynamic SLAM problem poses a challenging, yet necessary, enhancement to the functionality of SLAM. In addition to a novel deep learning approach for determining map initialization suitability, this dissertation has proposed, implemented, and validated a general, novel approach for registering and localizing unknown moving objects in the scene with no required prior knowledge of the objects' structure, appearance, or existence.

The approach itself proposes accomplishing this goal by mapping out feature points of the objects *before* they begin moving and then analyzing the reprojection error of these points *after* they begin moving to solve the motion segmentation problem. Additionally, feature points can be appropriately clustered (based on the moving objects they are associated with) by using the EPnP algorithm in a RANSAC scheme. This ultimately provides a mapped reconstruction of the moving object which can be re-localized each frame in realtime with the same EPnP process. This approach was also validated by implementing it in the custom-built, monocular SLAM system, LUMO-SLAM. This dissertation has also shown, through quantitative results of LUMO-SLAM's general SLAM performance in comparison to state-of-the-art SLAM systems, that LUMO-SLAM maintains localization accuracy comparable to existing SLAM implementations despite being designed to also register and localize moving objects. Though there are certain practical limitations that prevent LUMO-SLAM from being a dynamic SLAM solution for *all* use cases, the results achieved for its registration and localization accuracy indicate that this approach is highly-appropriate for *some* use cases, such as those in which a user needs to track the movement of a dynamic moving object but can also manually mediate the registration of the object.

The results also illuminate some of the limitations of the current implementation. For example, ORB features appear to be surprisingly brittle for unguided feature matching, as their matching strength diminishes greatly as the camera baseline increases, even in a reasonable range. Additionally, the system seems to struggle registering the features on an already-registered moving object if the object is not being viewed from its principal viewing angle. This greatly limits the functionality of the system and is likely the result of insufficient feature choice. The system also experiences lower camera localization accuracy than its

keyframe-based counterparts, ORB-SLAM and PTAM. It is likely that this limitation is a result of unrefined feature matching.

Given these limitations, a few appropriate areas for future work are noted below:

- As one of the major challenges in the development of the system revolved around successful, unguided feature matching, the development of features stronger than ORB would likely be a worthy direction for future research.

- To improve localization accuracy, the initial unguided feature matches could be refined by searching for the optimally-matching image patch, localized *around* the initial matching image patch.

- To improve the recall of already-registered moving objects, multiple descriptors for moving object points could be searched for in the current image to yield better results. The runtime capability of this, however, warrants further investigation.

- To enable the system to initialize in planar scenes as well as non-planar scenes, homography estimation could be added to the initialization process, similar to the approach used in ORB-SLAM.

- Place recognition (i.e., the ability to quickly determine the best subset of the map to match an image frame against) is perhaps one of the most important problems in the practical SLAM domain as it is used to detect loops and recover from tracking loss. The place recognition approach implemented in LUMO-SLAM is highly dependent on trained models and yields limited useful results. The development of a stronger (and perhaps, even "tunable") place recognition approach would enable better results in loop detection and even pave the way for a simplification in the system's overall architecture.

- On the note of place recognition, recovery from severe tracking loss could be implemented using the current place recognition approach in order to provide a high degree of real-world robustness to the system.

Overall, this body of work provides an interesting and useful step in the direction of developing an enhanced SLAM solution. This solution, among many other efforts in the field of dynamic SLAM, gives reason to be both excited and optimistic about the future capabilities of SLAM and their corresponding AR applications.

REFERENCES

[1] B. Troutman and M. Tuceryan, "Towards fast and automatic map initialization for monocular SLAM systems," in *Proceedings of the 2nd International Conference on Robotics, Computer Vision and Intelligent Systems - ROBOVIS*, INSTICC, SciTePress, 2021, pp. 22–30, ISBN: 978-989-758-537-1. DOI: 10.5220/0010640600003061.

[2] B. Troutman and M. Tuceryan, "Rapid structure from motion frame selection for markerless monocular SLAM," in *Robotics, Computer Vision and Intelligent Systems*, P. Galambos, E. Kayacan, and K. Madani, Eds., Cham: Springer International Publishing, 2022, pp. 172–189, ISBN: 978-3-031-19650-8. DOI: 10.1007/978-3-031-19650-8_9.

[3] B. Troutman and M. Tuceryan, "Registration and localization of unknown moving objects in monocular SLAM," in *2022 IEEE 2nd International Conference on Intelligent Reality (ICIR)*, 2022, pp. 43–48. DOI: 10.1109/ICIR55739.2022.00025.

[4] B. Troutman and M. Tuceryan, "Towards dynamic realtime object labeling in augmented reality," in *2022 IEEE 2nd International Conference on Intelligent Reality (ICIR)*, 2022, pp. 49–53. DOI: 10.1109/ICIR55739.2022.00026.

[5] L. M. Paz, P. Piniés, J. D. Tardós, and J. Neira, "Large-scale 6-DOF SLAM with stereo-in-hand," *IEEE transactions on robotics*, vol. 24, no. 5, pp. 946–957, 2008.

[6] T. Pire, T. Fischer, J. Civera, P. De Cristóforis, and J. J. Berlles, "Stereo parallel tracking and mapping for robot localization," in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, IEEE, 2015, pp. 1373–1378.

[7] C. Mei, G. Sibley, M. Cummins, P. Newman, and I. Reid, "RSLAM: A system for large-scale mapping in constant-time using stereo," *International journal of computer vision*, vol. 94, pp. 198–214, 2011.

[8] R. A. Newcombe, S. Izadi, O. Hilliges, *et al.*, "Kinectfusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE international symposium on mixed and augmented reality*, IEEE, 2011, pp. 127–136.

[9] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, and J. McDonald, "Real-time large-scale dense RGB-D SLAM with volumetric fusion," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 598–626, 2015.

[10] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-D mapping with an RGB-D camera," *IEEE transactions on robotics*, vol. 30, no. 1, pp. 177–187, 2013.

[11] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "Elasticfusion: Real-time dense SLAM and light source estimation," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1697–1716, 2016.

[12] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-time single camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 1052–1067, 2007.

[13] A. Chiuso, P. Favaro, H. Jin, and S. Soatto, "Structure from motion causally integrated over time," *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 4, pp. 523–535, 2002.

[14] E. Eade and T. Drummond, "Scalable monocular SLAM," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, IEEE, vol. 1, 2006, pp. 469–476.

[15] J. Civera, A. J. Davison, and J. M. Montiel, "Inverse depth parametrization for monocular SLAM," *IEEE transactions on robotics*, vol. 24, no. 5, pp. 932–945, 2008.

[16] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2007, pp. 225–234. DOI: 10.1109/ISMAR.2007.4538852.

[17] G. Klein and D. Murray, "Parallel tracking and mapping on a camera phone," in *2009 8th IEEE International Symposium on Mixed and Augmented Reality*, 2009, pp. 83–86. DOI: 10.1109/ISMAR.2009.5336495.

[18] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: A versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015. DOI: 10.1109/TRO.2015.2463671.

[19] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE transactions on robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[20] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *Computer Vision - ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., Cham: Springer International Publishing, 2014, pp. 834–849, ISBN: 978-3-319-10605-2. DOI: 10.1007/978-3-319-10605-2_54.

[21] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2017.

[22] H. Strasdat, J. Montiel, and A. J. Davison, "Scale drift-aware large scale monocular SLAM," *Robotics: Science and Systems VI*, vol. 2, no. 3, p. 7, 2010.

[23] H. Lim, J. Lim, and H. J. Kim, "Real-time 6-DOF monocular visual slam in a large-scale environment," in *2014 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2014, pp. 1532–1539.

[24] K. Pirker, M. Rüther, and H. Bischof, "CD SLAM-continuous localization and mapping in a dynamic world," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2011, pp. 3990–3997.

[25] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, "Visual simultaneous localization and mapping: A survey," *Artificial intelligence review*, vol. 43, pp. 55–81, 2015.

[26] Z. Xu, Z. Rong, and Y. Wu, "A survey: Which features are required for dynamic visual simultaneous localization and mapping?" *Visual Computing for Industry, Biomedicine, and Art*, vol. 4, no. 1, pp. 1–16, 2021.

[27] B. Bescos, J. M. Fácil, J. Civera, and J. Neira, "DynaSLAM: Tracking, mapping, and inpainting in dynamic scenes," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4076–4083, 2018.

[28] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[29] C. Yu, Z. Liu, X.-J. Liu, *et al.*, "DS-SLAM: A semantic visual SLAM towards dynamic environments," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 1168–1174.

[30] L. Cui and C. Ma, "SOF-SLAM: A semantic visual slam for dynamic environments," *IEEE access*, vol. 7, pp. 166 528–166 539, 2019.

[31] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[32] N. Brasch, A. Bozic, J. Lallemand, and F. Tombari, "Semantic monocular SLAM for highly dynamic environments," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 393–400.

[33] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, "ICNet for real-time semantic segmentation on high-resolution images," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 405–420.

[34] X. Long, W. Zhang, and B. Zhao, "PSPNet-SLAM: A semantic SLAM detect dynamic object by pyramid scene parsing network," *IEEE Access*, vol. 8, pp. 214 685–214 695, 2020.

[35] S. Han and Z. Xi, "Dynamic scene semantics SLAM based on semantic segmentation," *IEEE Access*, vol. 8, pp. 43 563–43 570, 2020.

[36] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2881–2890.

[37] Y.-b. Ai, T. Rui, X.-q. Yang, *et al.*, "Visual SLAM in dynamic environments based on object detection," *Defence Technology*, vol. 17, no. 5, pp. 1712–1721, 2021.

[38] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[39] L. Xiao, J. Wang, X. Qiu, Z. Rong, and X. Zou, "Dynamic-SLAM: Semantic monocular visual localization and mapping based on deep learning in dynamic environment," *Robotics and Autonomous Systems*, vol. 117, pp. 1–16, 2019.

[40] W. Liu, D. Anguelov, D. Erhan, *et al.*, "SSD: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, Springer, 2016, pp. 21–37.

[41] C. Zhang, T. Huang, R. Zhang, and X. Yi, "PLD-SLAM: A new RGB-D SLAM method with point and line features for indoor dynamic scene," *ISPRS International Journal of Geo-Information*, vol. 10, no. 3, p. 163, 2021.

[42] A. G. Howard, M. Zhu, B. Chen, *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[43] J. Huang, S. Yang, T.-J. Mu, and S.-M. Hu, "ClusterVO: Clustering moving instances and estimating visual odometry for self and surroundings," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2168–2177.

[44] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[45] B. Bescos, C. Campos, J. D. Tardós, and J. Neira, "DynaSLAM II: Tightly-coupled multi-object tracking and SLAM," *IEEE robotics and automation letters*, vol. 6, no. 3, pp. 5191–5198, 2021.

[46] I. Ballester, A. Fontán, J. Civera, K. H. Strobl, and R. Triebel, "DOT: Dynamic object tracking for visual SLAM," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 11 705–11 711.

[47] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, *Detectron2*, https://github.com/facebookresearch/detectron2, 2019.

[48] J. Zhang, M. Henein, R. Mahony, and V. Ila, "VDO-SLAM: A visual dynamic object-aware SLAM system," *arXiv preprint arXiv:2005.11052*, 2020.

[49] M. Strecke and J. Stuckler, "EM-Fusion: Dynamic object-level SLAM with probabilistic data association," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 5865–5874.

[50] C. Godard, O. Mac Aodha, M. Firman, and G. J. Brostow, "Digging into self-supervised monocular depth estimation," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 3828–3838.

[51] S. Yang and S. Scherer, "CubeSLAM: Monocular 3-D object SLAM," *IEEE Transactions on Robotics*, vol. 35, no. 4, pp. 925–938, 2019.

[52] G. B. Nair, S. Daga, R. Sajnani, *et al.*, "Multi-object monocular SLAM for dynamic environments," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2020, pp. 651–657.

[53] C. Wang, B. Luo, Y. Zhang, *et al.*, "DymSLAM: 4D dynamic scene reconstruction based on geometrical motion segmentation," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 550–557, 2020.

[54] Y. Zhang, B. Luo, and L. Zhang, "Permutation preference based alternate sampling and clustering for motion segmentation," *IEEE Signal Processing Letters*, vol. 25, no. 3, pp. 432–436, 2017.

[55] K. M. Judd, J. D. Gammell, and P. Newman, "Multimotion visual odometry (MVO): Simultaneous estimation of camera and third-party motions," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 3949–3956.

[56] D. Migliore, R. Rigamonti, D. Marzorati, M. Matteucci, D. G. Sorrenti, *et al.*, "Use a single camera for simultaneous localization and mapping with mobile object tracking in dynamic environments," in *ICRA Workshop on Safe navigation in open and dynamic environments: Application to autonomous vehicles*, 2009, pp. 12–17.

[57] C.-H. Hsiao and C.-C. Wang, "Achieving undelayed initialization in monocular SLAM with generalized objects using velocity estimate-based classification," in *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011, pp. 4060–4066.

[58] M. R. U. Saputra, A. Markham, and N. Trigoni, "Visual SLAM and structure from motion in dynamic environments: A survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 2, pp. 1–36, 2018.

[59] V. S. Varadarajan, *Lie groups, Lie algebras, and their representations*. Springer Science & Business Media, 2013, vol. 102.

[60] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking," in *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, IEEE, vol. 2, 2005, pp. 1508–1515.

[61] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004, ISSN: 1573-1405. DOI: 10.1023/B:VISI.0000029664.99615.94.

[62] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," *Lecture notes in computer science*, vol. 3951, pp. 404–417, 2006.

[63] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary robust independent elementary features," in *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11*, Springer, 2010, pp. 778–792.

[64] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[65] P. L. Rosin, "Measuring corner properties," *Computer Vision and Image Understanding*, vol. 73, no. 2, pp. 291–307, 1999.

[66] E. Marchand, H. Uchiyama, and F. Spindler, "Pose estimation for augmented reality: A hands-on survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 12, pp. 2633–2651, 2016. DOI: 10.1109/TVCG.2015.2513408.

[67] R. Y. Tsai and T. S. Huang, "Uniqueness and estimation of three-dimensional motion parameters of rigid objects with curved surfaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 1, pp. 13–27, 1984. DOI: 10.1109/TPAMI.1984.4767471.

[68] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Second. Cambridge University Press, ISBN: 0521540518, 2004.

[69] R. Hartley, "Cheirality," *International Journal of Computer Vision*, vol. 26, pp. 41–61, 1998. DOI: 10.1023/A:1007984508483.

[70] D. Nistér, "An efficient solution to the five-point relative pose problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 756–770, 2004. DOI: 10.1109/TPAMI.2004.17.

[71] O. D. Faugeras and F. Lustman, "Motion and structure from motion in a piecewise planar environment," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 2, no. 03, pp. 485–508, 1988.

[72] Z. Zhang and A. R. Hanson, "3D reconstruction based on homography mapping," *Proc. ARPA96*, pp. 1007–1012, 1996.

[73] E. Malis and M. Vargas, "Deeper understanding of the homography decomposition for vision-based control," Ph.D. dissertation, INRIA, 2007.

[74] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[75] M. Dhome, M. Richetin, J.-T. Lapreste, and G. Rives, "Determination of the attitude of 3D objects from a single perspective view," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 12, pp. 1265–1278, 1989. DOI: 10.1109/34.41365.

[76] R. Horaud, B. Conio, O. Leboulleux, and B. Lacolle, "An analytic solution for the perspective 4-point problem," *Computer Vision, Graphics, and Image Processing*, vol. 47, no. 1, pp. 33–44, 1989.

[77] R. M. Haralick, C.-n. Lee, K. Ottenburg, and M. Nölle, "Analysis and solutions of the three point perspective pose estimation problem.," in *CVPR*, vol. 91, 1991, pp. 592–598.

[78] L. Quan and Z. Lan, "Linear n-point camera pose determination," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 21, no. 8, pp. 774–780, 1999.

[79] B. Triggs, "Camera pose and calibration from 4 or 5 known 3D points," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, IEEE, vol. 1, 1999, pp. 278–284.

[80] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An accurate $O(n)$ solution to the PnP problem," *International journal of computer vision*, vol. 81, pp. 155–166, 2009.

[81] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[82] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[83] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustmenta modern synthesis," in *Vision Algorithms: Theory and Practice: International Workshop on Vision Algorithms Corfu, Greece, September 21–22, 1999 Proceedings*, Springer, 2000, pp. 298–372.

[84] T. Korah, J. Wither, Y.-T. Tsai, and R. Azuma, "Mobile augmented reality at the hollywood walk of fame," in *2011 IEEE Virtual Reality Conference*, 2011, pp. 183–186. DOI: 10.1109/VR.2011.5759460.

[85] M. Maidi, M. Preda, and V. H. Le, "Markerless tracking for mobile augmented reality," in *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, 2011, pp. 301–306. DOI: 10.1109/ICSIPA.2011.6144077.

[86] A. Ufkes and M. Fiala, "A markerless augmented reality system for mobile devices," in *2013 International Conference on Computer and Robot Vision*, 2013, pp. 226–233. DOI: 10.1109/CRV.2013.51.

[87] T. Kobayashi, H. Kato, and H. Yanagihara, "Novel keypoint registration for fast and robust pose detection on mobile phones," in *2013 2nd IAPR Asian Conference on Pattern Recognition*, 2013, pp. 266–271. DOI: 10.1109/ACPR.2013.67.

[88] C. Arth, C. Pirchheim, J. Ventura, D. Schmalstieg, and V. Lepetit, "Instant outdoor localization and SLAM initialization from 2.5D maps," *IEEE Transactions on Visualization and Computer Graphics*, vol. 21, no. 11, pp. 1309–1318, 2015. DOI: 10.1109/TVCG.2015.2459772.

[89] Z. Xiao, X. Wang, J. Wang, and Z. Wu, "Monocular ORB SLAM based on initialization by marker pose estimation," in *2017 IEEE International Conference on Information and Automation (ICIA)*, 2017, pp. 678–682. DOI: 10.1109/ICInfA.2017.8078992.

[90] L. Sun, J. Du, and W. Qin, "Research on combination positioning based on natural features and gyroscopes for ar on mobile phones," in *2015 International Conference on Virtual Reality and Visualization (ICVRV)*, 2015, pp. 301–307. DOI: 10.1109/ICVRV.2015.55.

[91] T. Qin, P. Li, and S. Shen, "VINS-Mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018. DOI: 10.1109/TRO.2018.2853729.

[92] M. Tomono, "3-D localization and mapping using a single camera based on structure-from-motion with automatic baseline selection," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2005, pp. 3342–3347. DOI: 10.1109/ROBOT.2005.1570626.

[93] M. M. Butt, H. Zhang, X. Qiu, and B. Ge, "Monocular SLAM initialization using epipolar and homography model," in *2020 5th International Conference on Control and Robotics Engineering (ICCRE)*, 2020, pp. 177–182. DOI: 10.1109/ICCRE49379.2020.9096497.

[94] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[95] D. Galvez-López and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012. DOI: 10.1109/TRO.2012.2197158.

[96] R. Mur-Artal and J. D. Tardós, "Fast relocalisation and loop closing in keyframe-based SLAM," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 846–853. DOI: 10.1109/ICRA.2014.6906953.

[97] A. Bonarini, W. Burgard, G. Fontana, M. Matteucci, D. Sorrenti, and J. Tardos, "RAWSEEDS: Robotics advancement through web-publishing of sensorial and elaborated extensive data sets," in *International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2006, p. 93.

[98] B. K. P. Horn, "Closed-form solution of absolute orientation using unit quaternions," *J. Opt. Soc. Am. A*, vol. 4, no. 4, pp. 629–642, Apr. 1987. DOI: 10.1364/JOSAA.4.000629. [Online]. Available: https://opg.optica.org/josaa/abstract.cfm?URI=josaa-4-4-629.

[99] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-D point sets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 5, pp. 698–700, 1987. DOI: 10.1109/TPAMI.1987.4767965.

[100] S. H. Lee and J. Civera, *What's wrong with the absolute trajectory error?* 2022. DOI: 10.48550/ARXIV.2212.05376. [Online]. Available: https://arxiv.org/abs/2212.05376.

[101] N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard, "Real-time 3D visual slam with a hand-held RGB-D camera," in *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum*, Vasteras, Sweden, Apr. 2011.

[102] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, "An evaluation of the RGB-D slam system," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 1691–1696. DOI: 10.1109/ICRA.2012.6225199.

[103] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2014.01.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320314000235.

VITA

Blake Troutman earned his Bachelor of Science (B.Sc.) degree in computer science with a minor in mathematics from Purdue University, Indianapolis in 2018 and earned his Master of Science (M.Sc.) degree in computer science from Purdue University, Indianapolis in 2019. He began his Ph.D. studies at Purdue University, Indianapolis in 2020, studying computer vision and visual simultaneous localization and mapping for augmented reality. He has published multiple papers in peer-reviewed conferences sponsored by the Institute of Electrical and Electronic Engineers (IEEE) and the Institute for Systems and Technologies of Information, Control and Communication (INSTICC), he is the winner of the Best Student Paper Award from the 2nd International Conference on Robotics, Computer Vision and Intelligent Systems, and he is pending graduation for his Ph.D. in May, 2023.

In addition to his education and research experience, he has been working as a professional software developer for five years and has been working as a teaching assistant for numerous graduate and undergraduate courses for six years. His responsibilities as a teaching assistant have covered a broad range of obligations, such as grading, holding office hours, giving lectures, and developing quizzes. The courses he has tended to in his teaching assistantships have covered many topics, including:

- Introductory programming in Python
- Introductory programming in C/C++ and Java
- Client-side web development with HTML, CSS, and JavaScript
- Server-side web development with PHP
- Relational database design and implementation
- Software engineering
- Graduate-level object-oriented design in Java
- Graduate-level distributed computing

Blake Troutman's primary research interest is computer vision, particularly focusing on visual simultaneous localization and mapping for augmented reality. Additionally, he also has recreational research interest in deep learning.

PUBLICATIONS

[1] B. Troutman and M. Tuceryan, "Towards fast and automatic map initialization for monocular SLAM systems," in *Proceedings of the 2nd International Conference on Robotics, Computer Vision and Intelligent Systems - ROBOVIS*, INSTICC, SciTePress, 2021, pp. 22–30, ISBN: 978-989-758-537-1. DOI: 10.5220/0010640600003061.

[2] B. Troutman and M. Tuceryan, "Rapid structure from motion frame selection for markerless monocular SLAM," in *Robotics, Computer Vision and Intelligent Systems*, P. Galambos, E. Kayacan, and K. Madani, Eds., Cham: Springer International Publishing, 2022, pp. 172–189, ISBN: 978-3-031-19650-8. DOI: 10.1007/978-3-031-19650-8_9.

[3] B. Troutman and M. Tuceryan, "Registration and localization of unknown moving objects in monocular SLAM," in *2022 IEEE 2nd International Conference on Intelligent Reality (ICIR)*, 2022, pp. 43–48. DOI: 10.1109/ICIR55739.2022.00025.

[4] B. Troutman and M. Tuceryan, "Towards dynamic realtime object labeling in augmented reality," in *2022 IEEE 2nd International Conference on Intelligent Reality (ICIR)*, 2022, pp. 49–53. DOI: 10.1109/ICIR55739.2022.00026.