Simultaneous Tracking, Object Registration, and Mapping (STORM)

by

Vincent P. Kee

B.S., Massachusetts Institute of Technology (2016)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2018

Author . . . .
Department of Electrical Engineering and Computer Science
February 2, 2018

Certified by. . . . .
Gian Luca Mariottini
Perception and Localization Group Leader
The Charles Stark Draper Laboratory
Thesis Supervisor

Certified by. . . . .
Sertac Karaman
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by . . . .
Christopher Terman
Chairman, Masters of Engineering Thesis Committee

Simultaneous Tracking, Object Registration, and Mapping (STORM)

by

Vincent P. Kee

Submitted to the Department of Electrical Engineering and Computer Science
on February 2, 2018, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Electrical Engineering and Computer Science

Abstract

An autonomous system needs to be aware of its surroundings and know where it is in its environment in order to operate robustly in unknown environments. This problem is known as Simultaneous Localization and Mapping (SLAM). SLAM techniques have been successfully implemented on systems operating in the real world.

However, most SLAM approaches assume that the environment does not change during operation — the static world assumption. When this assumption is violated (e.g. an object moves), the SLAM estimate degrades. Consequently, the static world assumption prevents robots from interacting with their environments (e.g. manipulating objects) and restricts them to navigating in static environments. Additionally, most SLAM systems generate maps composed of low-level features that lack information about objects and their locations in the scene. This representation limits the map's utility, preventing it from being used for tasks beyond navigation such as object manipulation and task planning.

We present Simultaneous Tracking, Object Registration, and Mapping (STORM), a SLAM system that represents an environment as a collection of dynamic objects. STORM enables a robot to build and maintain maps of dynamic environments, use the map estimates to manipulate objects, and localize itself in the map when revisiting the environment. We demonstrate STORM's capabilities with simulation and real-world experiments and compare its performance against that of a typical SLAM approach.

Thesis Supervisor: Gian Luca Mariottini
Title: Perception and Localization Group Leader
The Charles Stark Draper Laboratory

Thesis Supervisor: Sertac Karaman
Title: Associate Professor of Aeronautics and Astronautics

Acknowledgments

I want to first thank my Draper thesis supervisor Gian Luca Mariottini for mentoring me and providing guidance throughout the project. I appreciate how he was always available to help me work out solutions to technical obstacles I would frequently encounter.

I would like to also thank Dave Johnson for providing direction and feedback on my thesis. I am also grateful for how he made sure we had all the resources we needed. A huge thank you to Jay Wong, Mitchell Hebert, and Syler Wagner for all their help with ROS and SegICP. Thank you to the rest of the Draper Mobile Manipulation team and our collaborators at MIT and Harvard for all their insight and feedback.

Many thanks to Ted Steiner and Rob Truax from the Perception and Localization Group at Draper for patiently assisting me with iSAM2 and working out the ideas of STORM.

I would also like to thank my MIT thesis advisor, Sertac Karaman, for giving me the opportunity to work on this thesis.

A big thank you to Chris Yu for giving me the opportunity to do research at Draper through the Draper Fellowship Program.

Many thanks to the great friends in the different communities I've been privileged to be a part of while at MIT: Cru, Aletheia, the Man Cave, and New House 2.

A special thank you to my parents and brother for all their support over the years.

Finally, I would like to thank God for giving me the opportunity and abilities to study and do research at MIT and Draper.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past decade, there has been a large increase in applications of robotics technologies to a wide variety of fields. Some of these applications include self-driving cars [10, 35, 53], drones [40, 14], and warehouse robots [37, 36]. These systems operate autonomously in real-world environments where people are present, and sometimes even in collaboration with people or other robots.

As a result, autonomous robots need to be aware of their surroundings and know where they are with respect to objects to robustly perform tasks such as collision-free navigation, object manipulation, and task planning (see Figure 1-1). This problem has been the focus of research for the past few decades and is known as Simultaneous Localization and Mapping (SLAM). As a robot explores an unknown environment, the robot's SLAM system uses the robot's on-board sensors to construct a model of an environment (mapping) and estimate the robot's position and orientation, or pose, in the map (localization). In the past ten years, practical SLAM techniques have been developed and successfully implemented on systems operating in the real world [17, 28, 25, 23].

However, most of these systems assume that the environment does not change as the robot explores it — the static world assumption [6]. This clearly is not the case as human environments are highly dynamic; we are constantly moving from place to place and moving objects around us. When an environment changes (e.g. objects move), most SLAM systems fail to accurately estimate their pose and maintain an

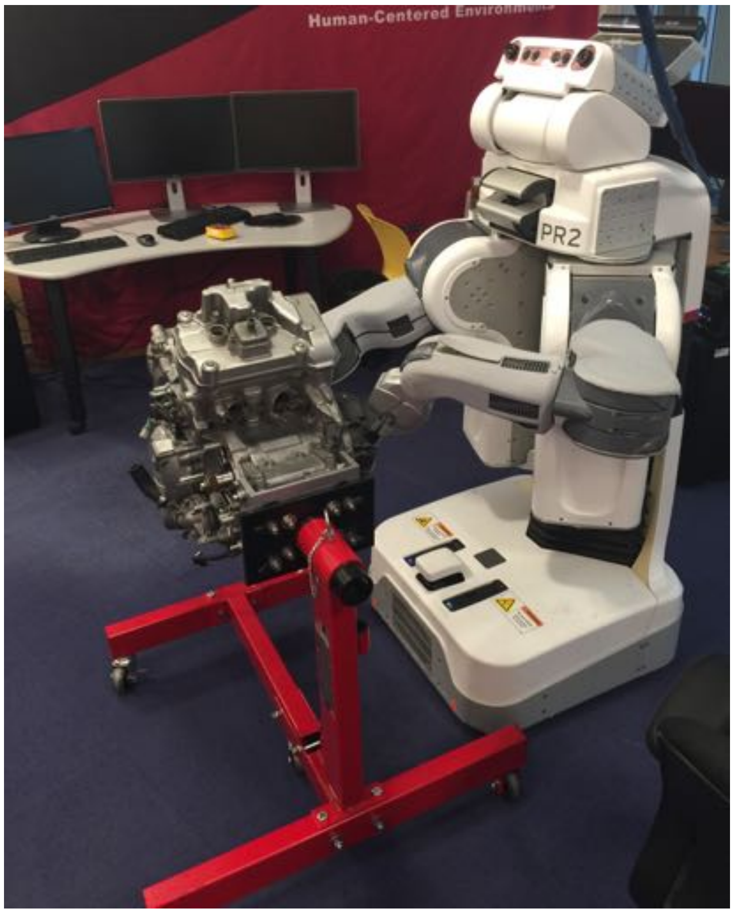


Figure 1-1: **Example of a robot in a real-world environment.** The robot needs to be aware of the identity and location of objects in its surroundings to robustly perform tasks such as collision-free navigation, object manipulation, and task planning.

accurate map of the environment. As a result, this assumption prevents robots from manipulating objects in their environment, as observing the objects after moving them will degrade the SLAM estimates. Consequently, the static world assumption limits SLAM systems to navigating in static environments, preventing true robot autonomy and functionality.

Additionally, most SLAM systems generate maps that are suitable only for navigation as the maps are composed of low-level primitives (i.e. points, surfaces, image features) [39]. These primitives do not have any associated semantic information — qualitative descriptors such as object identification [27]. These low-level representations prevent robots from being able to manipulate objects even if their SLAM systems are able to overcome the static world assumption.

This thesis is about overcoming these limitations to enable a robot to build maps of dynamic environments and use the maps to manipulate objects and localize itself.

1.1 Objective and Contributions

This thesis proposes a novel representation of the SLAM problem that models an environment as a collection of dynamic objects — Simultaneous Tracking, Object Registration, and Mapping (STORM). STORM relaxes the static world assumption, enabling a robot to operate in dynamic environments and manipulate objects. Additionally, STORM enables a robot to return to a previously mapped environment and operate as before. STORM localizes the robot in the environment and updates the map as objects move or are manipulated by the robot. We also present SegICP [56, 57], a real-time object pose estimator STORM uses to estimate 6DoF object poses.

To the best of our knowledge, STORM is the first SLAM system that builds an accurate object map of a dynamic environment and enables a robot to both manipulate objects and robustly relocalize in the future using the map.

The two main contributions of this thesis are:

1. A novel representation for modeling dynamic environments that enables robust localization and object manipulation (STORM)

2. An accurate, real-time object pose estimator (SegICP [56, 57])

1.2 Publications

The following publications resulted from research done during this MEng:

SegICP: Integrated Deep Semantic Segmentation and Pose Estimation [56]. Jay M. Wong, Vincent Kee, Tiffany Le, Syler Wagner, Gian-Luca Mariottini, Abraham Schneider, Lei Hamilton, Rahul Chipalkatty, Mitchell Hebert, David M.S. Johnson, Jimmy Wu, Bolei Zhou, and Antonio Torralba. *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2017.*

SegICP-DSR: Dense Semantic Scene Reconstruction and Registration [57]. Jay M. Wong, Syler Wagner, Connor Lawson, Vincent Kee, Mitchell Hebert, Justin Rooney, Gian-Luca Mariottini, Rebecca Russell, Abraham Schneider, Rahul Chipalkatty, and David M.S. Johnson. *arXiv preprint arXiv:1711.02216*.

1.3 Thesis Outline

This thesis is organized as follows. Chapter 2 addresses the limitations of most SLAM systems — the static world assumption and lack of semantic object-based maps — in more detail. This chapter presents existing approaches and their limitations, developing the main motivation for this work.

Chapter 3 reviews SLAM, first introducing the SLAM problem and then representing it as a probabilistic estimate. The chapter then presents the standard architecture of modern SLAM systems, their main components, and maps that they generate.

Chapter 4 presents STORM, developing the main ideas and contributions of this thesis. Chapter 5 provides analysis and results from evaluating STORM with simulation and real-world experiments, comparing STORM to a baseline approach and demonstrating its feasibility and capabilities. Chapter 6 concludes the thesis with a short discussion of directions for future work.

Chapter 2

Motivation and Related Work

In the past decade, practical SLAM techniques have been successfully implemented on a wide variety of systems operating in the real world [17, 28, 25, 23], building large-scale photorealistic maps of their environment while accurately estimating their sensor trajectory (see Figure 2-1).

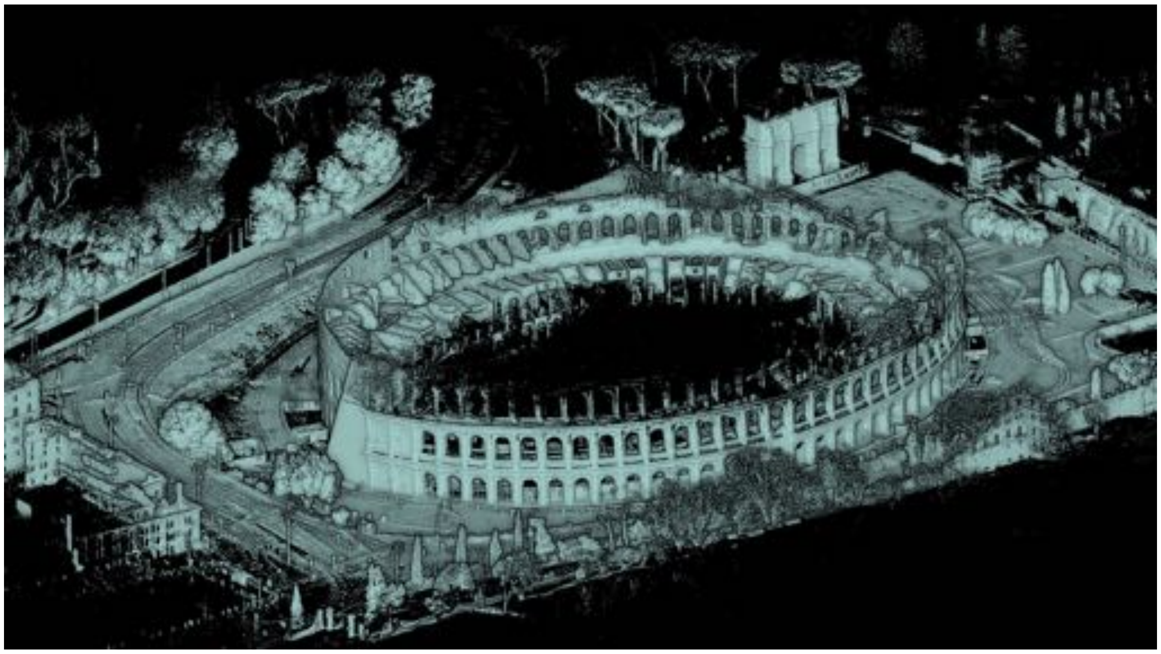


Figure 2-1: **Map of the Colosseum in Rome created in one pass** The map was created using Kaarta's Stencil [25]. Note the detail and accuracy of the model.

Despite all of these achievements, SLAM is far from solved [18, 6]. Most SLAM systems assume that their operational environment is static and their performance suffers when objects move. Additionally, most maps generated by SLAM systems are composed of low-level features without semantic information, restricting map usage beyond collision-free navigation. These two limitations are the static world assumption and the lack of semantic object-based maps. They prevent operation in a variety of real-world conditions and limit the utility of the map for tasks beyond navigation such as object manipulation.

In this chapter, we present the two major challenges and review existing work on resolving them.

2.1 Static World Assumption

As the name suggests, the static world assumption is the requirement that the environment does not change while a robot performs SLAM [6]. Most existing SLAM algorithms use a feature extractor to create landmarks from entities in the environment, and then derive the robot's location from its relative position to these landmarks. When landmarks move, the SLAM robot pose and map estimate degrade since the SLAM system handles landmark observations as if the landmarks remained in the same place.

Consider this motivating example shown in Figure 2-2: a robot performing SLAM observes a chair, creates a landmark out of the chair, and adds the landmark to its map. Suppose the chair is moved (either by the robot or an external actor) but the SLAM system assumes that the chair remained in the same pose (static world assumption). As a result, the chair's pose is not updated appropriately in the map and the robot's pose estimate is inaccurate. Consequently, the map is no longer consistent with the environment. Future observations of the chair will result in inaccurate robot localization as the SLAM system assumes the chair remains in its previous position.

As a result of the static world assumption, landmarks must remain static for the entire duration that SLAM operates in the environment. Since a feature extractor can create landmarks from almost anything in the environment, no moving people or objects can share the environment with the robot. This restriction drastically

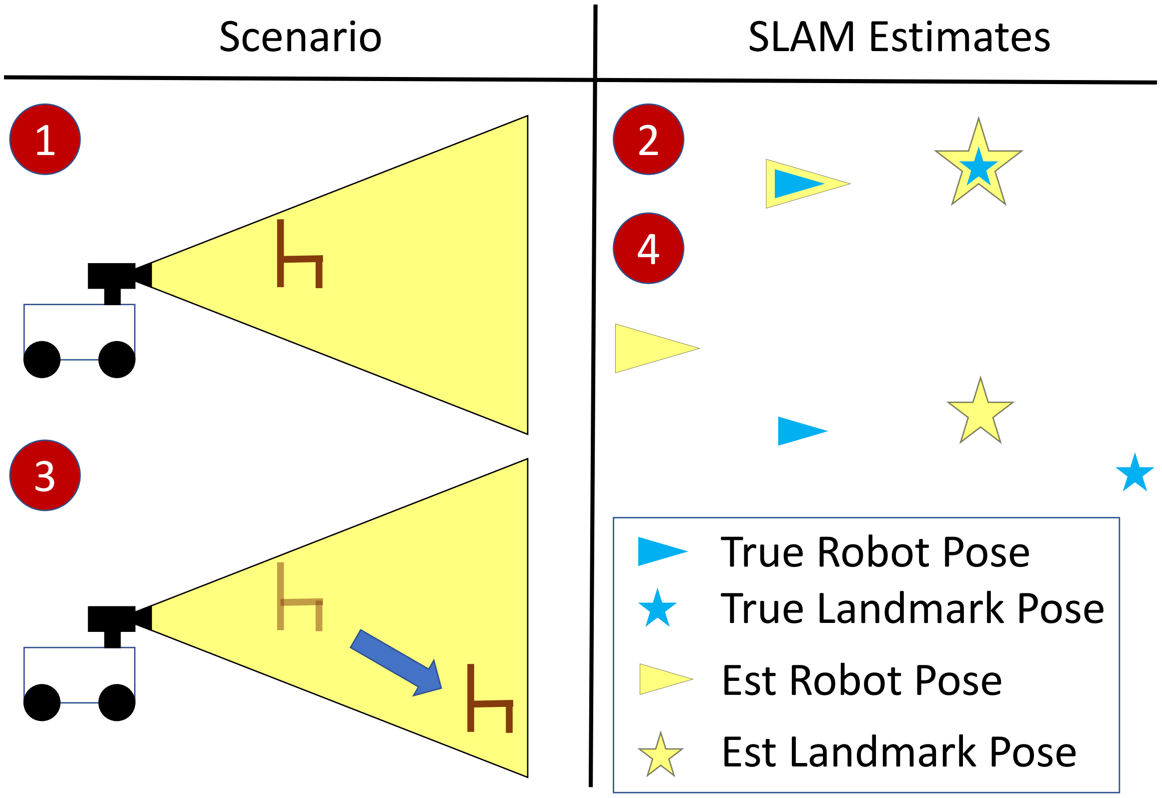


Figure 2-2: **Example of a static world assumption failure mode** 1) A robot observes a chair. 2) The SLAM system creates a landmark out of the chair and is able to estimate the chair's location quite accurately as well as the robot's position. 3) The robot stays in place but the chair moves. 4) Because of the static world assumption, the SLAM system assumes that the chair has remained in the same location. As a result, after observing the chair again, the chair's pose is not updated appropriately in the map and the robot is localized incorrectly.

limits the utility of SLAM in places such as buildings, warehouses, and urban environments where we would like to deploy autonomous robots. Additionally, as SLAM is performed in larger-scale environments over longer periods of time, the static world assumption is more likely to be violated [6]. When part of a scene changes, we would prefer to update the map rather than having to remap the entire environment.

Even when a robot is the sole actor in an environment, the static world assumption drastically limits interaction with its surroundings. Any task that could move landmarks (e.g. object manipulation) would degrade the SLAM solution. Consequently, typical SLAM systems only enable robots to navigate static environments and prevents them from interacting with their environment.

The static world assumption is clearly not suitable for robots operating in the real world. This assumption limits both the range of environments that the robot is able to operate in as well as the robot's capabilities, preventing true robot autonomy and functionality.

2.2 Lack of Semantic Object-based Maps

Most SLAM maps are composed of low-level units such as image features, geometric features, or surface representations [39] (see Figure 2-1). As described by Civera *et al.* [9], these primitives are "meaningless" as they contain no semantic information (e.g. object category). Without post-processing by additional perception systems, the robot has no notion what a collection of units represents. Consequently, the maps are only suitable for localization and navigation through the identified free space. Additionally, these maps represent environments inefficiently as many entities are used to model scenes even if the environment is simple [8, 6, 39]. For example, a SLAM map of a room with many repeated objects would typically represent each object instance with a large number of features rather than just the object identity and pose.

Semantic object-based maps offer several advantages [6]. Embedding the identity and pose of objects in the map enables robots to use the map to perform tasks at the object level such as object manipulation and task planning [31]. Physical properties (e.g. mass, moment of inertia, and stiffness) can be associated with each object, which is useful for manipulating the objects. Other properties (e.g. "do_not_move", "keep_upright", and "my_coffee_cup") can also be associated with each object, which is useful for task planning. This representation is better for human-robot interaction as humans describe environments at the level of objects and their properties, not low-level primitives [48, 6]. We want to be able to tell a robot to "bring me my cup of coffee from the kitchen counter" rather than telling the robot the grasp pose of the cup.

Representing an environment as a collection of objects rather than low-level entities is also better for modeling dynamic environments where objects may move or be manipulated by the robot [48, 15]. Rather than searching for and identifying all of the features that correspond to the moving object, the SLAM system would just update the object instance in the map. Object-based maps also enable a robot to reason about occluded parts of an environment as the robot knows the geometry of occluded portions of objects [6]. This knowledge is useful for tasks such as relocalizing in an environment where the robot observes the same objects but from a new viewpoint.

Finally, semantic object-based maps provide computational performance benefits. Object maps enable significant map compression as an object can be represented by its identity, pose, and relevant properties rather than a large collection of low-level entities [39, 38, 6]. For graph optimization-based SLAM systems, object-based maps drastically reduce computation. Each object is represented by one node whereas typical systems would represent each object using many nodes, with one node for each of the object's many features [8].

2.3 Related Work

There has been much work tackling these two limitations in SLAM independently. We briefly discuss some of the existing work.

2.3.1 SLAM in Dynamic Environments

SLAM algorithms designed for operation in dynamic environments use widely varying techniques to localize and maintain accurate maps.

Walcott-Bryant *et al.* [50, 49] compare laser scans of an environment taken at different times to detect environmental changes. Separate static and dynamic components are used to construct 2D maps of low-dynamic, indoor planar environments. Wang *et al.* [52, 51] handle moving and static objects separately and localize using only static objects to perform 2D SLAM with a ground vehicle in dynamic urban environments. Wolf *et al.* [55] maintain two occupancy grid maps to model static and dynamic parts of the environment and determine static landmarks for localization, which are tracked in a third map.

Although they all take different approaches to solve the dynamic SLAM problem, these methods ultimately try to separate the environment into static and dynamic components, with the latter two methods [52, 55] only using static landmarks or features for localization. The 2D maps these methods generate lack semantic information and cannot be used to manipulate objects in the environment.

Hähnel *et al.* [22] use expectation maximization (EM) to ignore laser measurements containing moving objects to improve localization and mapping performance in indoor and outdoor environments. Xiang *et al.* [58] use an EM-based technique to learn landmark mobilities and weight landmark measurements accordingly, giving observations of more static landmarks more weight to improve SLAM performance. However, it is not an online approach. While these EM-based approaches were able to improve SLAM results in dynamic environments, they do not build semantic maps.

Newcombe *et al.* [33] estimate a dense volumetric motion field to construct dense maps of non-rigid scenes in real-time. Like the other dynamic SLAM techniques mentioned, this estimate does not build maps with semantic information.

2.3.2 Semantic Object-based SLAM

Numerous object-based SLAM systems have been developed utilizing different techniques for combining object tracking and SLAM.

Gálvez-López *et al.* [19] incorporate bag-of-visual-words object detection into monocular SLAM, improving each system's performance. Similarly, Pillai *et al.* [34] use SLAM to improve object recognition performance. However, these approaches rely on the static world assumption.

Salas-Moreno *et al.* [39] present an 'object-oriented' SLAM approach, achieving significant map compression while generating dense surface reconstructions in real time. It does not rely on the static world assumption, but stops tracking moving objects. Sünderhauf *et al.* [46] integrate deep learning object detection techniques into SLAM to create semantic, object-oriented maps without needing a prior database of object models. However, the semantic knowledge is not used to improve SLAM.

Choudhary *et al.* [7] achieve multiple orders-of-magnitude map compression by building object-based maps with multiple robots. Choudhary *et al.* [8] combine online object discovery and modeling with SLAM to improve loop closure detection and SLAM performance. Mu *et al.* [31] model data association and object-based SLAM in a single framework improving localization and data association. However, all these approaches depend on the static world assumption and do not use the estimates to manipulate objects.

Ma *et al.* [30] present a framework to robustly detect objects and perform dense SLAM on each object in real-time. It is able to handle moving objects but the focus is on creating accurate object models rather than creating a map to manipulate objects or relocalize a robot in future visits.

2.3.3 SLAM for Manipulation

There have not been many SLAM approaches developed for object manipulation as it requires resolving the two challenges mentioned previously. We present the two approaches we are aware of. Ma *et al.* [29] extend [30] by adding in manipulation for more robust object discovery. However, it does not operate in real-time and again the focus is on discovering objects and building accurate models of them rather than using the map for relocalizing in the future. Babu *et al.* [2] couple manipulation and visual SLAM to increase success rates of a grasping task with a dynamically unstable robot. However, SLAM is only used to estimate the robot's state to improve dynamic motion plans and assemble a consistent point cloud as the robot moves. It does not estimate the object state.

2.4 Discussion

In this chapter, we presented two significant limitations of current SLAM systems that restrict real-world performance and map utility — the static world assumption and lack of semantic, object-based maps. Table 2.1 summarizes the existing work reviewed in this chapter. Most existing work has focused on only one of these challenges

Table 2.1: **Comparison of SLAM Approaches**

Work	Map dynamic scene (* ignore moving objects)	Object Map	Map for Manipulation	Real-time Operation	Relocalize with Map
Walcott-Bryant [50, 49]	✔	✘	✘	✔	✘
Wang [52]	✔	✘	✘	✔	✘
Wolf [55]	✔	✘	✘	✔	✘
Hähnel *et al.* [22]	✔	✘	✘	✔	✘
Xiang *et al.* [58]	✔	✘	✘	✘	✘
Newcombe *et al.* [33]	✔	✘	✘	✔	✘
Gálvez-López *et al.* [19]	✘	✔	✘	✔	✘
Pillai *et al.* [34]	✘	✔	✘	✔	✘
Salas-Moreno [39]	✔*	✔	✘	✔	✔
Sünderhauf *et al.* [46]	✘	✔	✘	✔	✘
Choudhary [7, 8]	✘	✔	✘	✔	✘
Mu [31]	✘	✔	✘	✔	✘
Ma [29]	✔	✔	✔	✔	✘
Babu [2]	✘	✘	✘	✔	✘
STORM	✔	✔	✔	✔	✔

with varying levels of success. Not much work has been done on using SLAM map estimates for object manipulation as it requires resolving both limitations. This is the motivation for our work.

In this thesis, we present Simultaneous Tracking, Object Registration, and Mapping (STORM), a framework for resolving both the static world assumption and lack of semantic, object-based information in SLAM maps. STORM represents its environment as a collection of dynamic objects and build maps of objects where the object pose estimates are used for both manipulation and relocalization in the future. STORM is described in detail in Chapter 4.

Chapter 3

Simultaneous Localization and Mapping (SLAM)

In this chapter, we discuss the core components of SLAM, borrowing from the excellent overviews in [42, 45, 6, 43]. We first introduce the SLAM problem and show how it can be formulated as a probabilistic estimate. We then present the standard state-of-the-art SLAM architecture and approach to solve the problem. Finally, we discuss a key assumption of most SLAM systems — the static world assumption. For a more in-depth review of SLAM, we refer readers to [47, 42, 45, 6, 43].

3.1 The SLAM Problem

The SLAM problem is as follows: as a robot explores an unknown environment, build a map of the environment and determine where the robot is in the map using the robot's noisy onboard sensors. The robot's sensors enable it to roughly estimate its own motion and observe its surroundings.

The SLAM problem can be divided into two problems that need to be solved simultaneously: localization and mapping. The localization problem is to determine where the robot is in the environment given sensor data and a map of the environment. The mapping problem is to build a map of the environment given sensor data and the robot's trajectory. SLAM is a chicken-or-egg problem as the robot needs an accurate estimate of its position and orientation, or pose, to build an accurate map but at the same time, the robot needs an accurate map to determine its pose. Note that dead reckoning — estimating the robot's pose only using on-board sensor data — will drift over time as small errors in the robot's motion estimates are integrated. The map enables the robot to correct its pose estimate when it visits part of the environment it has already mapped.

Initially, both the map and the robot's location in the environment are unknown. As a result, the robot must solve the localization and mapping problem simultaneously as it explores the environment, hence the name SLAM. Furthermore, due to sensor noise and algorithm approximations, the robot is unable perfectly estimate its motion or the surroundings, which increases the difficulty of performing SLAM. With all this inherent uncertainty in the robot's motion estimates and observations of the environment, SLAM is typically approached using probabilistic techniques [21, 45].

3.1.1 Representing SLAM as a Probabilistic Estimate

We now begin to formulate the typical SLAM problem more formally so that we can use probabilistic methods to approach the SLAM problem. In SLAM, we want to estimate the robot's trajectory and the poses of landmarks in the environment as shown in Figure 3-1. Landmarks are extracted from sensor data and can be anything from low-level image features and geometric features to a complete object. We discretize time into time-steps and denote the current time-step as t. The robot's trajectory is represented as

$$X_T = \{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T\} \tag{3.1}$$

where $\mathbf{x}_i$ represents the robot's pose in the global coordinate frame at time-step i and T is some terminal time (possibly ∞). For 2D poses, $\mathbf{x}_i = \begin{bmatrix} x_i \; y_i \; \theta_i \end{bmatrix}^\top$ where x_i and y_i denote the position and θ_i denotes the heading in the global coordinate frame. For 3D poses (as used in this thesis), $\mathbf{x}_i = \begin{bmatrix} x_i \; y_i \; z_i \; \phi_i \; \theta_i \; \psi_i \end{bmatrix}^\top$ where x_i, y_i, and z_i denote the position and ϕ_i, θ_i, and ψ_i denote the orientation in the global coordinate

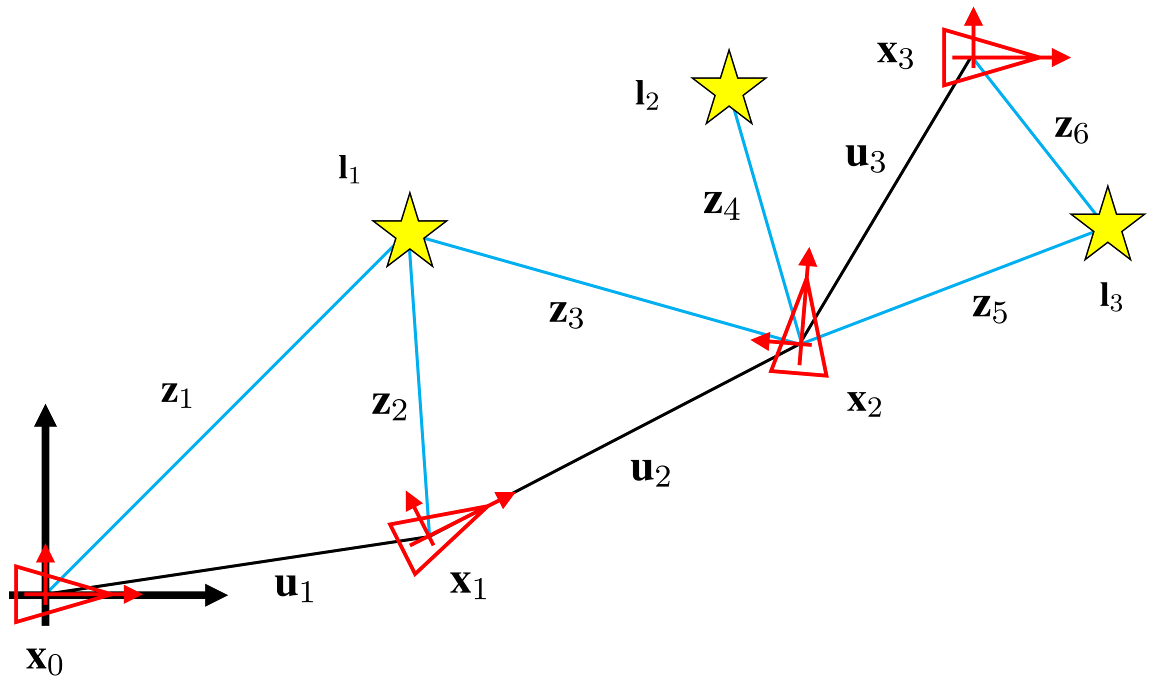


Figure 3-1: **An example of a robot performing SLAM** As a robot explores an unknown environment, it tries to estimate its trajectory and the poses of landmarks in the environment. The robot poses, $\mathbf{x}_i$, are represented by triangles and the landmark poses, $\mathbf{l}_i$, are represented by stars. Odometry measurements, $\mathbf{u}_i$, are represented by black edges and sensor measurements, $\mathbf{z}_i$, are represented by blue edges. Note that the global coordinate frame is typically set to the local coordinate frame of $\mathbf{x}_0$ unless additional information is available.

frame. The global coordinate frame is set to the local coordinate frame of the first robot pose $\mathbf{x}_0$, unless additional information is available.

The landmark poses are defined in the global coordinate frame like the robot poses. We represent the landmark poses as

$$M = \{\mathbf{l}_1, \mathbf{l}_2, \ldots, \mathbf{l}_n\} \tag{3.2}$$

where $\mathbf{l}_i$ is the pose of landmark i. Note that we estimate a landmark's pose, not its trajectory, which implicitly assumes that the landmark is static in the environment. We will discuss this important assumption more later.

To estimate X_T and M, the robot extracts measurements from its sensor data. A typical category of sensor measurements used for SLAM are odometry measurements, which give information about the robot's motion between consecutive poses. These measurements can be extracted with data from proprioceptive sensors which measure the robot's internal state (i.e. wheel rotation, heading) or exteroceptive sensors which observe the environment (i.e. track landmarks) to estimate the robot's motion. The measurements also can come from control inputs to the robot's motors; hence they are sometimes referred to as control inputs. We represent the odometry measurements as

$$U_T = \{\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_T\} \tag{3.3}$$

where $\mathbf{u}_i$ is the odometry measurement at time-step i. These odometry measurements come from the robot's motion model, which is derived from the robot's kinematic model and often nonlinear. The motion model gives a distribution of the robot's pose given an existing pose and odometry measurement:

$$\mathbf{x}_{i+1} = h(\mathbf{x}_i, \mathbf{u}_i) \oplus \boldsymbol{\omega}_u \tag{3.4}$$

where $\mathbf{x}_{i+1}$ is the pose at time $i+1$, $h(\cdot)$ is the robot's motion model, $\boldsymbol{\omega}_{\boldsymbol{u}}$ is zero-mean Gaussian noise with the information matrix (inverse of the covariance matrix) Ω_u, and $\oplus$ is the standard motion composition operator [41] that maps the measurement noise to an element of the manifold of 2D or 3D poses, SE(2) or SE(3) respectively [6]. Therefore,

$$\mathbf{x}_{i+1} \sim \mathcal{N}\Big(h(\mathbf{x}_i, \mathbf{u}_i), \Omega_u^{-1}\Big) \tag{3.5}$$

Note that if all the odometry measurements were noise-free, U_T would perfectly capture the robot's trajectory.

As the robot observes landmarks in the environment with its sensors, it is able to make sensor measurements of the environment other than the odometry readings. These measurements can come from a wide range of sensors such as cameras, LiDARs, and GPS and can range from being bearing, range, or even pose measurements of landmarks to laser rangefinder range measurements and absolute sensor pose readings. We represent these measurements as

$$Z_T = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_T, \} \tag{3.6}$$

where $\mathbf{z}_i$ is the i-th landmark measurement. These measurements come from the sensor's observation model, which is often nonlinear and a rough approximation of the actual sensor behavior [45]. The observation model predicts the expected observation given the estimated robot pose and map:

$$\mathbf{z}_i = h(\mathbf{x}_i, M) \oplus \boldsymbol{\omega}_z \tag{3.7}$$

where $h(\cdot)$ is the observation model and ω_z is zero-mean Gaussian noise with the information matrix Ω_z.

Over the past 30 years of the SLAM field's existence, there have been different probabilistic approaches to SLAM [6, 45]. For the first 20 years of the field, the popular approaches were based on Extended Kalman filters and Rao-Blackwellised particle filters and came to be known as filters. These filters are able to run in real-time and solve what is known as the online SLAM problem or filtering. The online SLAM problem is defined as follows:

$$P(\mathbf{x}_t, M | U_{t-1}, Z_t) \tag{3.8}$$

where the filter estimates a distribution over the current robot pose $\mathbf{x}_t$ and state of the map M. Filters typically run incrementally, processing each measurement individually [43].

In comparison, graph-based optimization techniques, referred to as smoothers, solve what is known as the full SLAM problem or smoothing. They represent the SLAM problem as a graph and use optimization techniques to efficiently solve the full SLAM problem. The full SLAM problem is defined as:

$$P(X_T, M | U_T, Z_T) \tag{3.9}$$

where the smoothers estimates a distribution over the entire robot's trajectory X_T and state of the map M. Note that all of the measurements Z_T over the entire history are used to solve the full SLAM problem, so smoothers run offline by nature. Additionally, smoothers typically process all the measurements at once and are referred to as batch [43]. In the past 10 years, recent developments with smoothing-based techniques have achieved state-of-the-art performance [6, 45].

Additionally, incremental smoothing approaches [26] have been introduced which enable online smoothing:

$$P(X_t, M|U_{t-1}, Z_t) \tag{3.10}$$

where the incremental smoother estimates the current trajectory, X_t, and state of the map M using the available measurements Z_t and U_{t-1}.

We refer the reader to [16, 4, 47, 43, 42, 1] for more information about filtering techniques. In the remainder of this review, we focus on approaches that estimate (3.9) and (3.10).

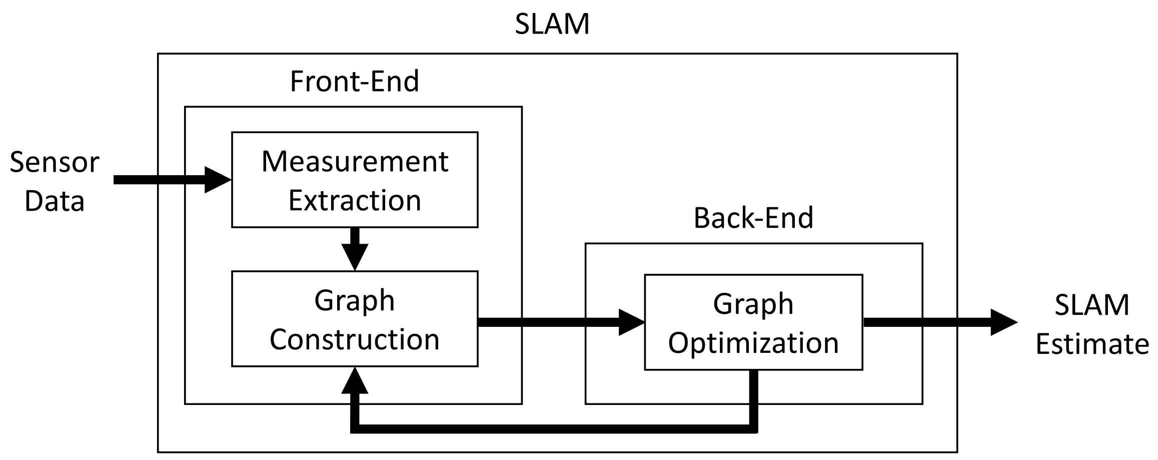


Figure 3-2: **Architecture of a Modern SLAM System** The front-end extracts measurements from raw sensor data and constructs a graph representation of the SLAM problem while the back-end optimizes the graph to solve the SLAM problem.

3.2 Architecture of a Modern SLAM System

As shown in Figure 3-2, a modern SLAM system consists of two main components: a front-end and a back-end. The front-end extracts measurements from raw sensor data and constructs a graph representation of the SLAM problem while the back-end optimizes the graph to solve the SLAM problem. Each component of STORM is discussed in greater detail below.

3.3 Front-End

The front-end plays a critical role in the performance of a SLAM system. It converts raw sensor data into a representation that the back-end can use to estimate (3.9) and (3.10). The two main tasks that the front-end performs are measurement extraction and model construction.

3.3.1 Measurement Extraction

The front-end extracts relevant measurements from sensors onboard the robot. These measurements can come from a wide variety of proprioceptive sensors (which measure internal state) and exteroceptive sensors (which acquire information about the environment). Proprioceptive sensors include inertial measurement units (IMU) and wheel and joint encoders. Exteroceptive sensors include cameras, laser scanners, and GPS receivers. As mentioned previously, there are two categories of sensor measurements: odometry measurements and landmark measurements.

Odometry measurements (U) describe the robot's motion between consecutive poses ($\mathbf{x}_i$ to $\mathbf{x}_{i+1}$). As each robot pose ($\mathbf{x}_i$) defines a local coordinate frame i, we can represent an odometry measurement as the transformation ${}^{i}\mathrm{T}_{i+1}$ expressing the pose $\mathbf{x}_{i+1}$ in the frame of $\mathbf{x}_i$.

Landmark measurements (Z) describe an observed landmark's pose ($\mathbf{l}_i$) in the frame of the robot pose ($\mathbf{x}_i$) in which it was observed. We can represent the landmark measurement as the transformation ${}^{\mathbf{x}_i}\mathrm{T}_{\mathbf{l}_i}$ expressing the pose of $\mathbf{l}_i$ in the frame $\mathbf{x}_i$. A critical part of making a landmark measurement is correctly associating each measurement with its landmark — data association [4].

Data Association

Data association happens on two time scales: short-term and long-term. The short-term data association problem is tracking features across data taken at consecutive time-steps. The long-term data association problem is loop closure detection, which is determining when the robot is revisiting part of the environment. As a robot explores an environment, its pose estimate will inevitably drift as small errors in odometry and landmark measurements accumulate over time. If a robot revisits part of the environment and is able to determine that it has done so (e.g. determining that it is observing previously observed landmarks), it can eliminate much of this accumulated error.

3.3.2 Model Construction

As the robot explores an environment and the front-end extracts measurements, the front-end uses the measurements to construct a graph of the robot's trajectory and the environment for the back-end to optimize to estimate (3.9) and (3.10).

Graph Representations of SLAM

Two common graph representations of SLAM are Dynamic Bayesian Networks (DBN) and factor graphs.

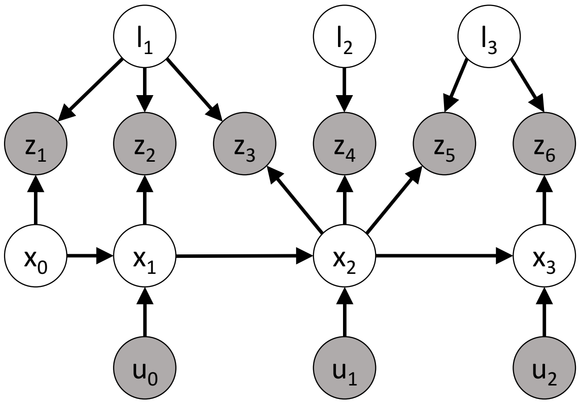


Figure 3-3: **The SLAM Problem in Figure 3-1 as a Dynamic Bayesian Network** Each node represents a random variable and each arrow represents a conditional dependence between two nodes (the child node depends on the parent node). The shaded nodes represent the observed random variables (the odometry measurements $\mathbf{u}_i$ and landmark measurements $\mathbf{z}_i$) and the white nodes represent the hidden random variables (the robot poses $\mathbf{x}_i$ and landmark poses $\mathbf{l}_i$) that SLAM is trying to estimate.

A Dynamic Bayesian Network (DBN) is a probabilistic graphical model that represents time-dependent random variables and their conditional dependencies as a directed acyclic graph (DAG) [45, 21]. As shown in Figure 3-3, each node represents a random variable and each arrow represents a conditional dependence between two nodes (the child node depends on the parent node). The connectivity of the DBN is defined by the motion (3.4) and observation (3.7) models.

Another way to represent the SLAM problem is with a different graphical model, the factor graph. A factor graph is a bipartite graph consisting of variables and factors which contain probabilistic information on variables [11]. The factors encode the conditional dependencies between variables as each edge in a factor graph is between a factor node and a variable node [45]. In SLAM, the variables represent the robot trajectory and landmark poses and the factors represent the measurements and impose probabilistic spatial constraints on the variables. The factors are defined by the motion (3.4) and observation (3.7) models. Loop closures are encoded as

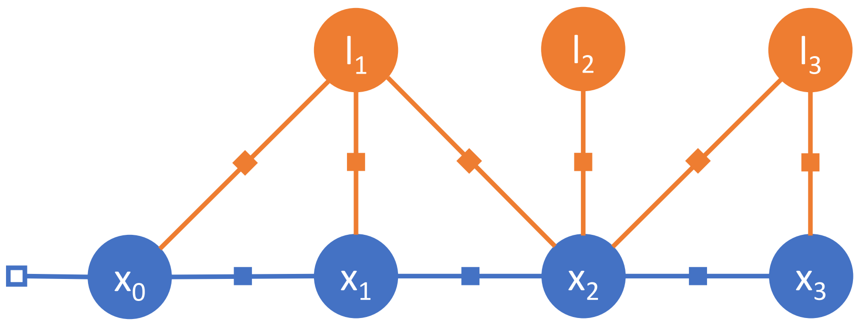


Figure 3-4: **The SLAM Problem in Figure 3-1 as a Factor Graph** Variable nodes are shown as circles and factor nodes are shown as squares. The blue circles represent robot poses ($\mathbf{x}_i$) and the orange circles represent landmark poses ($\mathbf{l}_i$). The hollow blue square represents a constraint from prior information, the solid blue squares represent odometry measurements ($\mathbf{u}_i$), and the the orange squares represent landmark measurements ($\mathbf{z}_i$).

factors between two non-consecutive robot poses and defined by the motion model as a loop closure describes the displacement between two non-consecutive robot poses [45]. Modern SLAM systems tend use factor graphs to represent the SLAM problem [6].

Once the graph is constructed, the graph is passed to the back-end to be optimized. The SLAM solution is the variable node configuration that is maximally consistent with the measurements encoded in the factors. In full SLAM, the front-end would construct a complete graph with all measurements and then pass it to the back-end. In incremental smoothing, the front-end constructs a graph as the robot explores environment and passes it to the backend.

3.4 Back-End

The back-end estimates the robot's trajectory (X_T) and landmark poses (M) by applying nonlinear least squares optimization techniques on the abstracted model of the environment created by the front-end. Since the model is a graph of constraints, the SLAM solution is the maximally likely configuration of nodes given all the measurements. This is a maximum-a-posteriori (MAP) estimate:

$$X_T, M = \underset{X_T, M}{\operatorname{argmax}} \; P(X_T, M | U_T, Z_T) \tag{3.11}$$

Before we show how SLAM can be turned into a nonlinear least squares optimization, we briefly mention an intuitive analogy for graph-based SLAM techniques from [20, 45]. We can represent the factor graph constructed by the front-end as a mass-spring model where the variables are represented by small masses and the factors are represented by springs that connect the masses and have strengths inversely proportional to the corresponding factor's covariance matrices. Thus, the resting state of the spring-mass model corresponds to the SLAM solution and the stored energy of the system corresponds to the error in SLAM solution. Note that springs incorrectly connecting masses represent wrong data associations and springs with incorrect strengths represent inaccurate certainty in the measurements and that these both would affect the resting state of the system.

Just to recap, the SLAM solution is the robot trajectory and landmark poses most consistent with the measurements. We now show how the SLAM MAP estimate can be turned into a nonlinear least squares optimization, adopting the formulation of [6, 45].

3.4.1 SLAM as a Nonlinear Least Squares Optimization

To simplify notation, we follow [6] and represent the SLAM state we are estimating as

$$\mathcal{X} = \{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \mathbf{l}_1, \mathbf{l}_2, \ldots, \mathbf{l}_n\} \tag{3.12}$$

and the set of measurements we use to estimate $\mathcal{X}$ as

$$Z = \{\mathbf{z}_k : k = 1, 2, \ldots, m\} \tag{3.13}$$

with $\mathbf{z}_k$ is defined as follows:

$$\mathbf{z}_k = h_k(\mathcal{X}_k) \oplus \boldsymbol{\omega}_k \tag{3.14}$$

where $h_k(\cdot)$ is a non-linear function that computes the measurement (from the odometry or observation model), $\mathcal{X}_k \subseteq \mathcal{X}$, and $\boldsymbol{\omega}_k$ is zero-mean Gaussian noise with the information matrix Ω_k.

Therefore,

$$\mathbf{z}_k \sim \mathcal{N}\Big(h_k\big(\mathcal{X}_k\big), \Omega_k^{-1}\Big) \tag{3.15}$$

We rewrite (3.11) using $\mathcal{X}$ and Z:

$$\mathcal{X}^* = \underset{\mathcal{X}}{\operatorname{argmax}}\ P(\mathcal{X}|Z) = \underset{\mathcal{X}}{\operatorname{argmax}}\ P(Z|\mathcal{X})P(\mathcal{X}) \tag{3.16}$$

We use Bayes' rule for the last equality and initialize the prior $P(\mathcal{X})$ to a uniform distribution unless we have prior information about $\mathcal{X}$. Note that in the case of no prior information, the MAP estimate is equivalent to the maximum likelihood estimate.

We assume that the measurements are independent (noise is uncorrelated), which allows us to refactor (3.16) as

$$\mathcal{X}^* = \underset{\mathcal{X}}{\operatorname{argmax}}\ P(Z|\mathcal{X})P(\mathcal{X}) = \underset{\mathcal{X}}{\operatorname{argmax}}\, P(\mathcal{X}) \prod_{k=1}^{m} P(\mathbf{z}_k|\mathcal{X}) \tag{3.17}$$

Since each measurement z_k only depends on a subset of $\mathcal{X}$,

$$\mathcal{X}^* = \underset{\mathcal{X}}{\operatorname{argmax}}\, P(\mathcal{X}) \prod_{k=1}^{m} P(\mathbf{z}_k|\mathcal{X}) = \underset{\mathcal{X}}{\operatorname{argmax}}\, P(\mathcal{X}) \prod_{k=1}^{m} P(\mathbf{z}_k|\mathcal{X}_k) \tag{3.18}$$

From (3.15), we know

$$P(\mathbf{z}_k|\mathcal{X}_k) = \frac{1}{\sqrt{2\pi|\Omega_k^{-1}|}} \exp\Big(-\frac{1}{2}\Big(h_k\big(\mathcal{X}_k\big) \ominus \mathbf{z}_k\Big)^\top \Omega_k \Big(h_k\big(\mathcal{X}_k\big) \ominus \mathbf{z}_k\Big)\Big) \tag{3.19}$$

where $\ominus$ is the inverse of the standard motion composition operator [41] and a "difference" on the manifold SE(3) [6].

Using the definition of squared Mahalanobis distance,

$$||a - b||_\Omega^2 = (a - b)^\top \Omega (a - b) \tag{3.20}$$

we simplify (3.19)

$$P(\mathbf{z}_k|\mathcal{X}_k) = \frac{1}{\sqrt{2\pi|\Omega_k^{-1}|}} \exp\Big(-\frac{1}{2}\Big|\Big|h_k\big(\mathcal{X}_k\big) \ominus \mathbf{z}_k\Big|\Big|_{\Omega_k}^2\Big) \propto \exp\Big(-\frac{1}{2}\Big|\Big|h_k\big(\mathcal{X}_k\big) \ominus \mathbf{z}_k\Big|\Big|_{\Omega_k}^2\Big) \tag{3.21}$$

The prior can be formulated similarly

$$P(\mathcal{X}) \propto \exp\Big(-\frac{1}{2}\Big|\Big|h_0\big(\mathcal{X}\big) \ominus \mathbf{z}_0\Big|\Big|_{\Omega_0}^2\Big) \tag{3.22}$$

which simplifies (3.18)

$$\mathcal{X}^* = \operatorname*{argmax}_{\mathcal{X}} P(\mathcal{X}) \prod_{k=1}^{m} (\mathbf{z}_k|\mathcal{X}_k) = \operatorname*{argmax}_{\mathcal{X}} \prod_{k=0}^{m} P(\mathbf{z}_k|\mathcal{X}_k) \tag{3.23}$$

Because maximizing the posterior is equivalent to minimizing the negative log posterior,

$$\begin{aligned} \mathcal{X}^* = \operatorname*{argmax}_{\mathcal{X}} \prod_{k=0}^{m} P(\mathbf{z}_k|\mathcal{X}_k) &= \operatorname*{argmin}_{\mathcal{X}} -\log\Big(\prod_{k=0}^{m} P(\mathbf{z}_k|\mathcal{X}_k)\Big) \\ &= \operatorname*{argmin}_{\mathcal{X}} \sum_{k=0}^{m} \Big|\Big|h_k\big(\mathcal{X}_k\big) \ominus \mathbf{z}_k\Big|\Big|_{\Omega_k}^2 \end{aligned} \tag{3.24}$$

(3.24) is a nonlinear least squares problem. Therefore, $\mathcal{X}^*$ can be solved with various nonlinear least squares optimization techniques and SLAM libraries, such as

GTSAM [11]. Note that this formulation assumes the noise is Gaussian. When using other noise models, this formulation will result in different cost functions [6].

This optimization is solved efficiently in large part due to the sparsity of the SLAM factor graph. Note that the degree of variable nodes is low as shown in Figure 3-4. The degree of landmark nodes stays low as landmarks are often only observed by a small subset of the robot poses. The degree of robot nodes stays low as loop closures do not happen too frequently and generally only a small subset of the landmarks are observed at each robot pose. In larger-scale environments, this sparsity increases. The reader is directed to [12, 13, 45] for more details about how SLAM solvers optimize (3.24) efficiently.

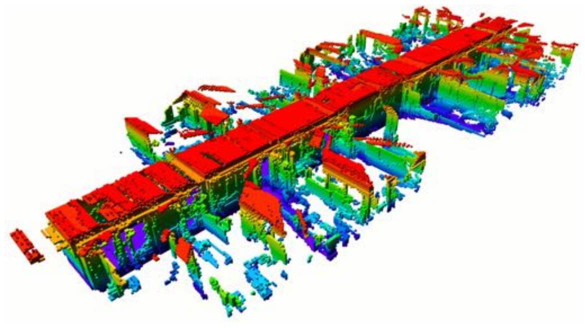
3.5 Map Types

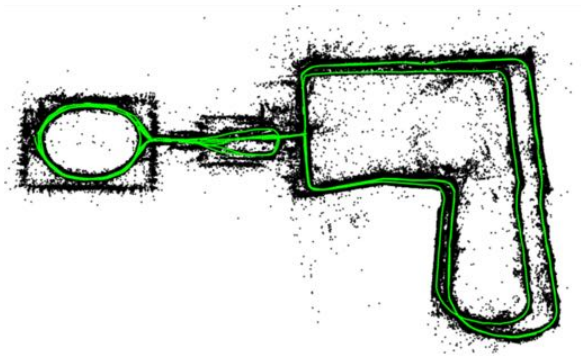
Once a SLAM system optimizes the factor graph, it has an estimate of its trajectory and the poses of landmarks in the environment. There are a variety of approaches to creating maps: some systems build a map composed of the landmarks while others only use the landmarks for localization and construct a map by projecting sensor measurements at each pose into a global coordinate frame. We briefly mention some of the different types of maps SLAM systems produce, which vary depending on the robot's sensors and map use case. The reader is directed to [48, 5, 6] for more details about the various map representations.

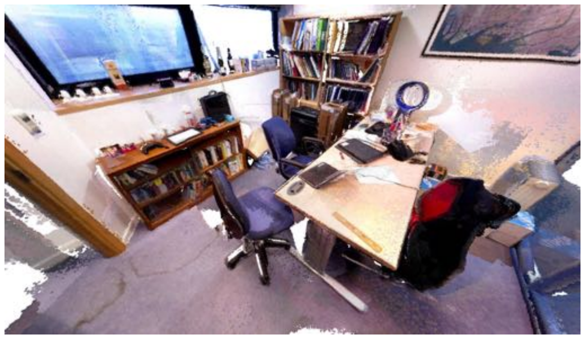
Occupancy grid maps discretize the world into a grid with each individual cell indicating the probability of a cell being occupied [45]. The top left image of Figure 3-5 depicts a 3D occupancy grid map of an office building created by a robot with a laser scanner mounted to a pan-tilt unit to generate 360° laser scans [24].

Landmark or feature-based maps are sparse representations of environments created from the landmarks identified by SLAM [45, 6]. The top right image of Figure 3-5 depicts a feature-based map of a park and college campus created by a robot with a monocular camera [32].

Dense maps are high-resolution representations of environments created from raw


(a)


(b)


(c)

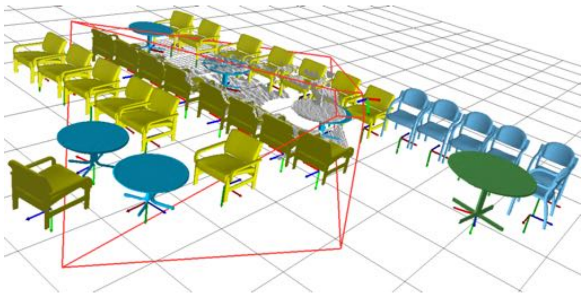

(d)

Figure 3-5: **Example SLAM Maps** Figure (a) depicts a 3D occupancy grid map of an office created with laser scans [24]. Figure (b) depicts a feature-based map of a park and college campus created with a monocular camera as well as the estimated trajectory in green [32]. Figure (c) depicts a dense map of an office created with an RGB-D camera [54]. Figure (d) depicts an object map of an office created with an RGB-D camera [39].

sensor data or surface or volume representations of the raw data [6]. The bottom left image of Figure 3-5 depicts a dense map of an office environment created using data collected with an RGB-D camera [54].

Object maps represent environments as collections of 3D objects. The bottom right image of Figure 3-5 depicts an object map of an office environment created using data collected with an RGB-D camera [39].

As mentioned in Chapter 2, these maps are typically used solely for robot navigation and localization, not for interacting with the environment.

3.6 The Static World Assumption

One key assumption most SLAM systems rely is on the static world assumption, which is the requirement that the environment does not change while performing SLAM [6]. Note how the state SLAM estimates, $\mathcal{X}$ (4.1), includes the landmark poses $\mathbf{l}_i$ with no time-dependencies. This representation of the landmark poses assumes that the landmarks do not move as the robot observes the landmarks over time. As shown in the factor graph representation of the SLAM problem (Figure 3-4), the landmarks are encoded with a single node rather than a trajectory like the moving robot. If the landmarks move, then the resulting SLAM estimates will be poor as the landmark measurements ($\mathbf{z}_i$) will be inconsistent.

As many environments are dynamic, the static world assumption significantly limits the types of environments that SLAM can function in. Additionally, as mentioned in Section 2.1, the robot cannot interact with environment because moving objects in the environment will break the static world assumption. This limitation motivates our system which we present in Chapter 4.

Chapter 4

Simultaneous Tracking, Object Registration, and Mapping (STORM)

As discussed in Chapter 2, existing SLAM algorithms are limited by the static world assumption and lack of semantic, object-based maps. To our knowledge, existing SLAM approaches only tackle one of these problems. There are no existing systems that build maps of objects where the object pose estimates are used for both manipulation and relocalization in the future.

This thesis uses some ideas of existing approaches to resolve both the static world assumption and lack of semantic information in one unified framework. Simultaneous Tracking, Object Registration, and Mapping (STORM) represents an environment as a collection of dynamic objects. STORM enables a robot to operate in dynamic environments, localizing the robot accurately even when objects are moving independently or being manipulated by the robot. STORM maintains a map of the objects parameterized with 6DoF poses, allowing a robot to manipulate objects with their pose estimates. Each object is represented by a 6DoF pose and semantic label rather than a large number of points or features. As a result, the STORM-generated map is orders of magnitude smaller than typical SLAM maps which are composed of millions of low-level entities. This representation compression enables more efficient estimation and mapping of larger environments than would be otherwise possible.

In this chapter, we present STORM. We begin by presenting an overview of

STORM and its architecture. We then describe the front-end and back-end. We end the chapter by explaining how STORM relocalizes with an existing map.

4.1 Overview of STORM

STORM represents an environment as a collection of dynamic objects as shown in Figure 4-1. In comparison to standard SLAM approaches, STORM estimates the robot's sensor trajectory and the trajectories (rather than the poses) of objects in an environment. The state STORM estimates at time t is the multivariate random variable

$$\mathcal{X} = \{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \mathbf{o}_1^1, \mathbf{o}_2^1, \ldots, \mathbf{o}_t^1, \mathbf{o}_1^2, \mathbf{o}_2^2, \ldots, \mathbf{o}_t^2, \mathbf{o}_1^m, \mathbf{o}_2^m, \ldots, \mathbf{o}_t^m\} \tag{4.1}$$

where $\mathbf{x}_i$ is the estimated 6DoF pose of the robot's sensor at time i and $\mathbf{o}_i^j$ is the estimated 6DoF pose of object j at time i. These poses are estimated in the global coordinate frame, set to the first robot sensor pose $\mathbf{x}_0$ unless additional information is given.

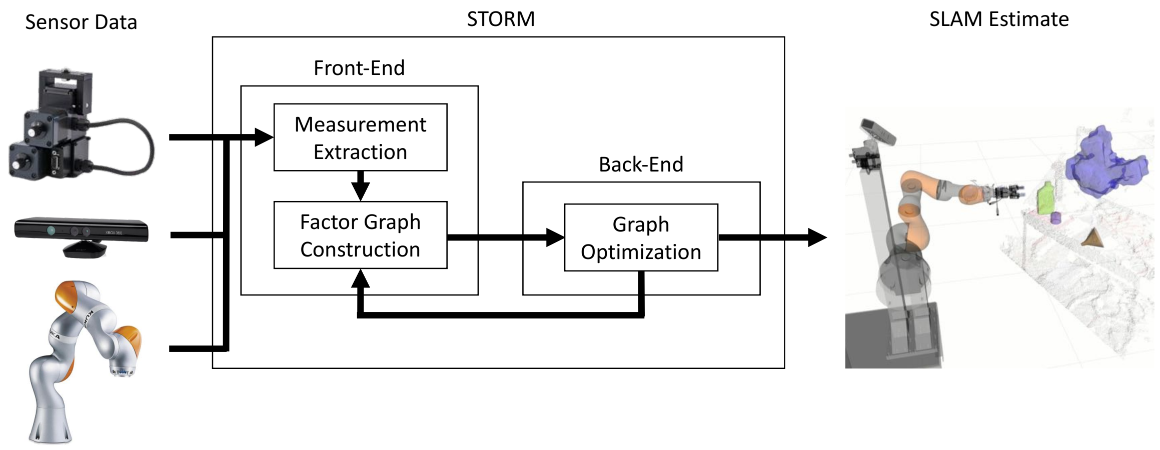
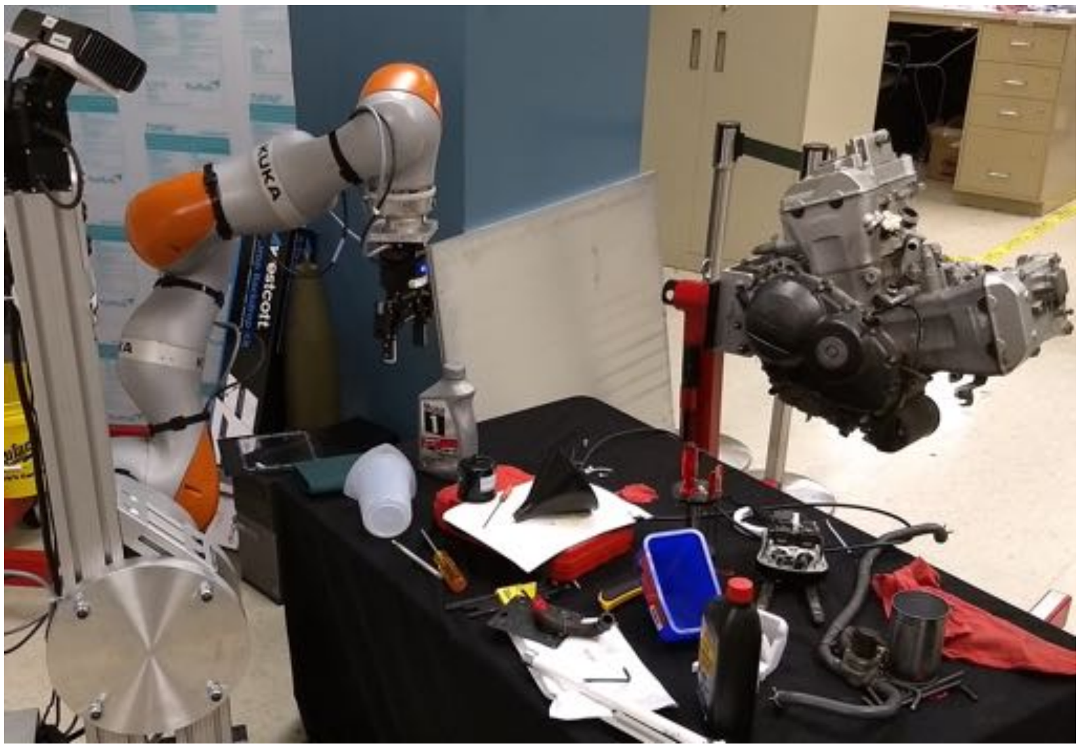
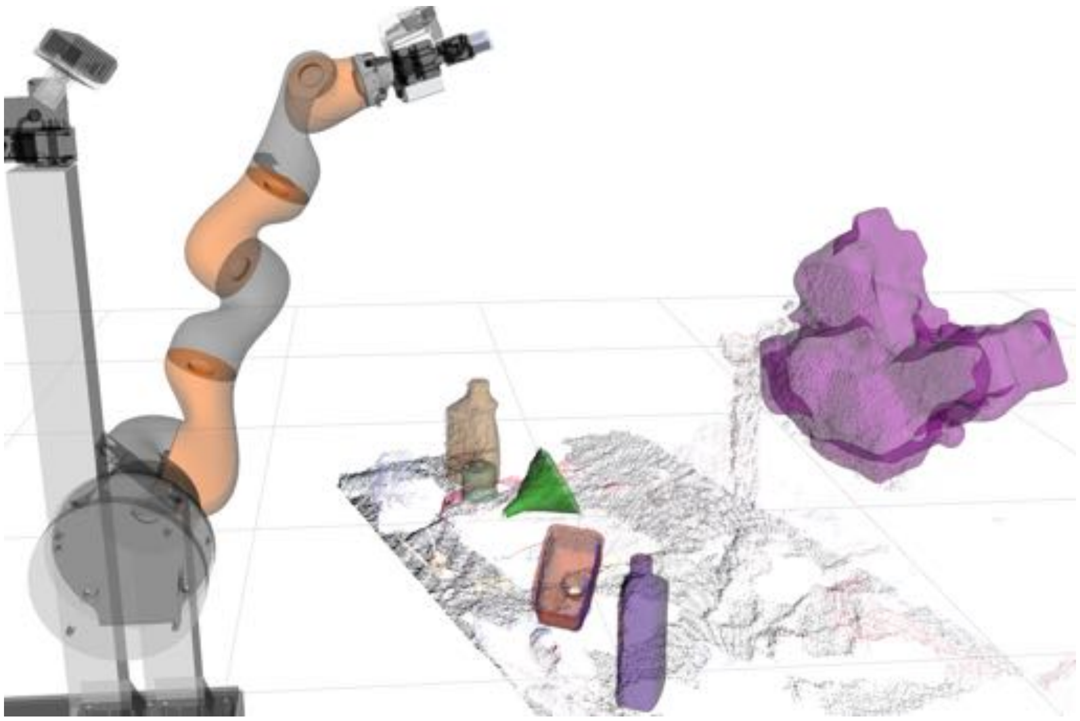


Figure 4-2: **Overview of the STORM Pipeline** The front-end of STORM extracts object measurements from sensor data to construct a factor graph. The factor graph is passed to the back-end of STORM to perform graph optimization.

The architecture of STORM is that of a standard SLAM system as described in Chapter 3. As shown in Figure 4-2, raw sensor data is passed to STORM's front-end,



(a)



(b)

Figure 4-1: **STORM represents an environment as a collection of objects** Figure (a) depicts a cluttered table of objects in front of a robot. Figure (b) depicts the raw Kinect point cloud along with STORM estimates, visualizing the robot, objects of interest, and their respective estimated poses. Notice how the object meshes closely align with the raw point cloud. Note that the images are not taken concurrently.

which extracts measurements from the data and uses them to construct a factor graph. The factor graph is passed to STORM's back-end to perform graph optimization. This modular architecture enables different components to be added or removed (e.g. different feature extractors in the front-end for different sensors). Each component of STORM is discussed in greater detail below.

4.2 Front-End

The front-end plays a critical role in the performance of a SLAM system, as described in Section 3.3 of Chapter 3. It extracts relevant measurements from proprioceptive and exteroceptive sensor data, performs data association, and constructs the factor graph that the back-end performs MAP estimation on. Without good measurements, the estimates will be poor.

Each time STORM extracts new measurements, it performs a graph update. It adds the new measurements to the factor graph, passes the factor graph to the back-end, and updates the factor graph with the optimized estimates from the back-end.

4.2.1 Measurement Extraction

STORM extracts two categories of measurements from sensor data: sensor-pose measurements (from proprioceptive sensors) and object-pose measurements (from exteroceptive sensors). When a new measurement is extracted, STORM checks for other available measurements. These new measurements along with the time elapsed since the last graph update are used to perform a graph update.

Sensor-Pose Measurements

There are two types of sensor-pose measurements that STORM handles: odometry measurements ($\mathbf{z}_i^u$) and absolute sensor pose measurements ($\mathbf{z}_i^a$). As a robot explores an environment, STORM takes odometry measurements ($\mathbf{z}_i^u$) to estimate the robot's sensor trajectory. As described in Chapter 3, odometry measurements describe the robot's sensor motion between consecutive poses ($\mathbf{x}_i$ to $\mathbf{x}_{i+1}$) and are represented as the transformation ${}^{i}\mathrm{T}_{i+1}$ expressing the sensor pose $\mathbf{x}_{i+1}$ in the frame of $\mathbf{x}_i$.

On the fixed base robot platform used for this work, the sensor is mounted on a pan-tilt unit (PTU) that gives the sensor two degrees of freedom (roll and pitch) to survey the surroundings. The PTU gives absolute angle measurements ($\mathbf{z}_i^a$) with high accuracy, enabling STORM to estimate the sensor's pose ($\mathbf{x}_i$) with high certainty. These absolute sensor pose measurements are represented as the transformation ${}^{0}\mathrm{T}_i$ expressing the pose $\mathbf{x}_i$ in the frame of the first sensor pose $\mathbf{x}_0$. Each time an odometry measurement is taken, STORM applies available object measurements from SegICP and its manipulators and performs a graph update.

Object-Pose Measurements

As a robot observes and interacts with surrounding objects, STORM takes object-pose measurements to estimate the objects' 6DoF poses. These estimates are used to build a map of the environment, manipulate the objects, and relocalize the robot in the map in future trials. On the platform used for this work, there are two ways to get object-pose measurements: SegICP (which generates SegICP measurements $\mathbf{z}_i^{sj}$) and object manipulation (which generates manipulation measurements $\mathbf{z}_i^{mj}$). Object-pose measurements are taken in the frame of the sensor and are represented as the transformation ${}^{\mathbf{x}_i}\mathrm{T}_{\mathbf{o}_i^j}$, expressing the pose of object j ($\mathbf{o}_i^j$) in the sensor frame $\mathbf{x}_i$.

SegICP SegICP [56, 57] is a perception pipeline that uses RGB-D sensor data (and proprioceptive information when available) to estimate the 6DoF poses of objects in the scene. In particular, SegICP performs deep pixel-level semantic segmentation and model-based point cloud registration without any prior object pose seeds, with an overall position accuracy of 1 cm and rotation accuracy of 2 deg at 14 Hz. Figure 4-3 shows some SegICP object pose estimates on a sample cluttered tabletop scene.

We now provide a brief overview of the SegICP pipeline. The reader is directed to [56, 57] for further details. As outlined in Figure 4-4, the RGB image from the RGB-D data is first passed through an adversarially-trained convolutional neural network

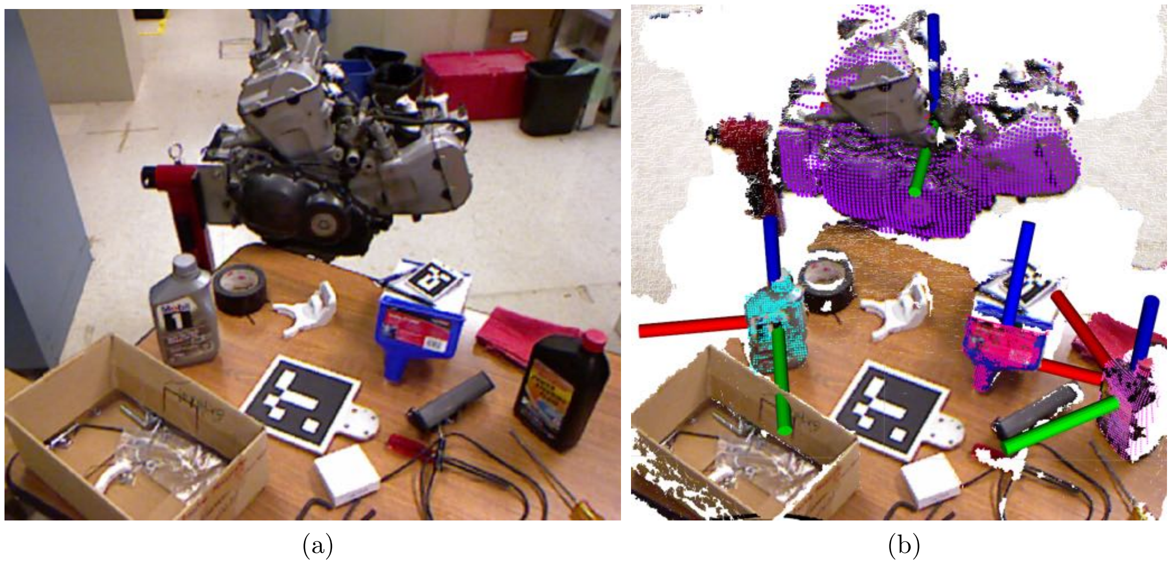


Figure 4-3: **SegICP object pose estimates of a sample cluttered tabletop scene** Figure (a) is an RGB image of the scene. Figure (b) visualizes object model point clouds at their respective estimated object poses. These are overlaid on the scene point cloud in the sensor frame $\mathbf{x}_i$. Note how the model point clouds (purple for engine, blue for oil bottle, red for blue funnel, pink for black bottle) are closely aligned with their target objects in the scene point cloud.

(CNN), SegNet [3], which outputs a segmented mask with pixel-wise semantic object labels. This segmented mask is used to extract each object's point cloud from the depth map corresponding to the original RGB image. Additionally, the labels in the mask are used to retrieve their corresponding 3D object mesh models from an object model library. These mesh models are converted into point clouds, downsampled, and registered with their corresponding object scene clouds to estimate each object's 6DoF pose. 3D-3D point cloud registration is performed for tracking by using rigid point-to-point iterative closest point (ICP).

The point cloud registration process is divided into two phases: acquisition and tracking. In acquisition, SegICP determines the visible face of the object and then initializes the tracking phase with a good initial pose and crop of the mesh model. Registration is performed using a crop of the mesh model, rather than the entire model, as the scene object point cloud only contains points visible to the RGB-D sensor. Raycasting is used to render candidate model crops from various azimuths

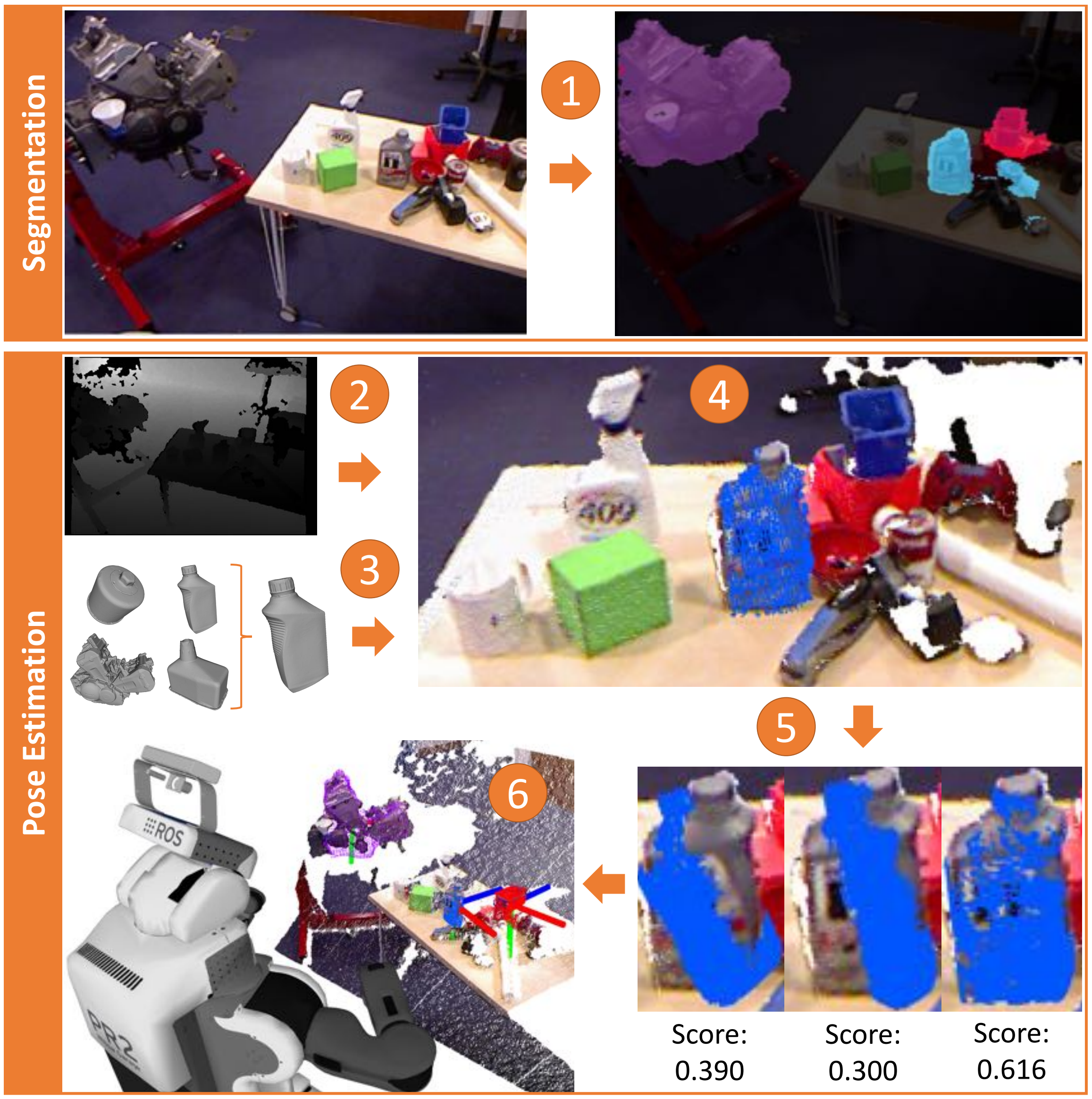


Figure 4-4: **The SegICP pipeline operating in a cluttered environment.** SegICP detects objects relevant to an automotive oil change task and estimates a 6DOF pose for each object in the scene. 1) SegICP performs semantic segmentation on an RGB image captured by a Kinect V1 mounted on a PR2 robot. The colored overlay pixels in the segmented image (top-right image) correspond to a blue funnel (red), an oil bottle (blue), and the engine (purple). These semantic labels are used to: 2) crop an object's point cloud from the depth map and 3) retrieve an object's mesh model from an object library. 4) Various crops of the mesh model are used to register against the cropped object's point cloud. 5) These hypothesis registrations are evaluated in parallel to determine the object crop with largest score. 6) The highest scoring model crop is then used to initialize a 3D-3D tracking routine (ICP).

and elevations. To remove segmentation outliers and prevent ICP from converging to incorrect local minima, each candidate crop is initialized at the median position of the object's point cloud. In parallel, each candidate crop is aligned with the scene object cloud with a few iterations of ICP. Then, each aligned candidate crop is evaluated with a model-to-scene alignment metric that determines which candidate crop aligns best with the object scene cloud. The metric finds the number of points in the candidate crop with a unique corresponding point in the object scene cloud. The highest scoring candidate crop and its estimated pose are used to initialize the tracking phase.

In tracking, additional iterations of ICP are used to further refine the object pose estimate. To make the tracking robust to imperfections on the boundary of the object's segmentation, the object's scene point cloud is further pruned by removing points outside a bounding box of the candidate crop at the latest estimated pose. The estimated pose is used as a measurement update in a Kalman filter tracking the object's 6DoF pose and twist. By fusing measured sensor motion (such as from a robot's odometry), the filter is able to handle temporary object occlusions and outlier pose estimates. The alignment metric is used to approximate the uncertainty of the current pose measurement. If the metric score is below a threshold θ, the Kalman filter propagates the objects' pose based on available odometry (and until a maximum pose uncertainty) while switching back to acquisition mode.

At the time of writing, SegICP assumes there is only one instance of each object class in the environment. Consequently, STORM has the same assumption. With this assumption, SegICP solves the data association problem by directly outputting each object's 6DoF pose in the sensor frame. Each object pose estimate is used to create a SegICP measurement ($\mathbf{z}_i^{sj}$). $\mathbf{z}_i^{sj}$ is represented as the transformation ${}^{\mathbf{x}_i}\mathrm{T}_{\mathbf{o}_i^j}$, expressing the pose of object j at time i ($\mathbf{o}_i^j$) in the sensor frame $\mathbf{x}_i$. Each time a SegICP measurement is taken (that is not initiated by another measurement), STORM applies available manipulator measurements ($\mathbf{z}_i^{mj}$) for the graph update.

Object Manipulation When manipulating an object, the robot has a low variance estimate of the object's pose ($\mathbf{o}_i^j$), assuming that the object remains grasped by the gripper. This enables STORM to have an estimate of an object's pose even if the object is not observable by SegICP. The object's pose can be retrieved by reading arm joint encoder values to recover the end-effector pose and create a manipulation measurement ($\mathbf{z}_i^{mj}$). $\mathbf{z}_i^{sj}$ is represented as the transformation ${}^{\mathbf{x}_i}\mathrm{T}_{\mathbf{o}_i^j}$, expressing the pose of object j at time i ($\mathbf{o}_i^j$) in the sensor frame $\mathbf{x}_i$. Each time an object manipulation measurement is taken (that is not initiated by another measurement), STORM applies available SegICP measurements ($\mathbf{z}_i^{sj}$) from SegICP for the graph update.

4.2.2 Constructing the Factor Graph

Like many state-of-the-art SLAM approaches, STORM uses a factor graph to model its environment and sensor trajectory. As shown in Figure 4-5, the variables nodes represent the robot sensor and object poses in SE(3) and the factor nodes encode probabilistic constraints over the nodes. These poses are all estimated in the global coordinate frame, which is set to the first robot sensor pose $\mathbf{x}_0$ unless additional information is given.

In contrast to most SLAM factor graph representations, STORM models the environment as being dynamic rather than static. Objects are assumed to be dynamic rather than static and are represented accordingly. As a result, STORM estimates each object's trajectory in the environment rather than an object's single static pose. If STORM has information about the object's motion (such as the object having onboard odometry measurements) other than by observing the object (i.e. SegICP or manipulation), this information can be used to estimate the object's pose similarly to (3.4). Otherwise, STORM models the object's motion as a random walk, where the mean of the transformation ${}^{i}\mathrm{T}_{i+1}$ expressing the object pose $\mathbf{o}_{i+1}^j$ in the frame of $\mathbf{o}_i^j$ is set to the identity transformation and its covariance is determined by the knowledge STORM has of the object's pose.

As a robot explores an environment, STORM constructs a factor graph with the measurements it extracts from its observations. Each time a graph update is performed, a new node is added to the robot sensor trajectory and every tracked object trajectory using the available information (shown in Figure 4-5).

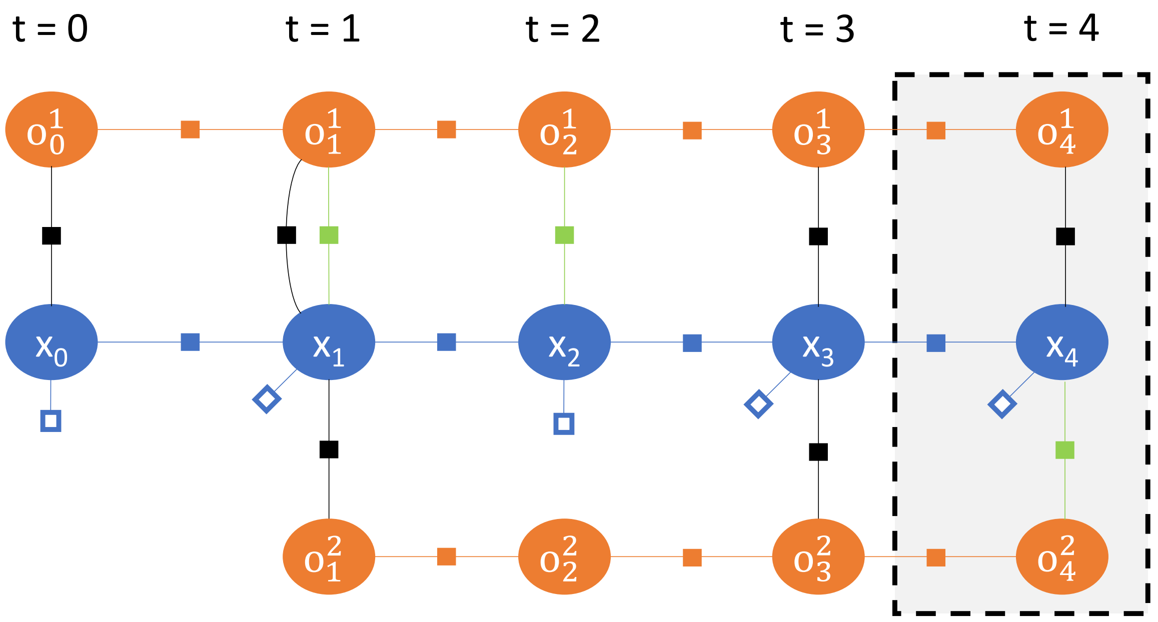


Figure 4-5: **Example STORM factor graph** Variable nodes are shown as circles and factor nodes are shown as squares. The blue circles represent robot sensor poses ($\mathbf{x}_i$) and the orange circles represent object poses ($\mathbf{o}_i^j$) in SE(3) where i is the time-step and j is the j-th object. These poses are all estimated in the global coordinate frame, which is set to the first sensor pose $\mathbf{x}_0$ unless additional information is given. The hollow blue squares represents constraints from the priors and absolute pose measurements ($\mathbf{z}_i^a$), solid blue squares represent odometry measurements ($\mathbf{z}_i^u$), the black squares represent SegICP object measurements ($\mathbf{z}_i^{sj}$), the green squares represent object manipulation measurements ($\mathbf{z}_i^{mj}$), and the orange squares represent object motion measurements ($\mathbf{z}_i^{tj}$). Note at $t = 4$, object 1 was observed with SegICP and object 2 was manipulated. These measurements, along with the odometry, absolute pose, and object motion measurements, are added to the factor graph as part of a graph update (in the shaded box).

We now discuss how the measurements are added to the factor graph.

Odometry Measurements

Odometry measurements ($\mathbf{z}_i^u$) describe the relative motion of the sensor in the environment. As the solid blue boxes in Figure 4-5 show, these measurements are used to construct factors between sensor poses at consecutive time-steps. In the cases where absolute sensor pose measurements ($\mathbf{z}_i^a$) are available, this additional information can be used to further constrain the estimate as shown by the blue hollow boxes in Figure 4-5.

At each graph update, the back-end creates a new node for the sensor trajectory ($\mathbf{x}_{i+1}$ if the last sensor pose is $\mathbf{x}_i$). If there is an odometry measurement in the update, it is used to construct the odometry factor. Otherwise, STORM assumes that the sensor has not moved and sets the mean of the odometry factor to the identity transformation. If there is an absolute sensor measurement in the update, it used to construct an absolute sensor pose measurement. Otherwise, STORM assumes that the sensor has not moved and uses the previous absolute sensor pose measurement for the new node.

Object Measurements

Object measurements describe the relative pose of an object in the sensor coordinate frame. As the solid black and green boxes in Figure 4-5 show, these measurements are used to construct factors between the object pose and the sensor pose corresponding to when the measurement was taken.

At each graph update, the back-end creates a new node for each tracked object trajectory ($\mathbf{o}^j_{i+1}$ for object j if the last object pose is $\mathbf{o}^j_i$). STORM also constructs a factor modeling each object's motion between the current object pose node and the previous object pose node (solid orange squares in Figure 4-5). As described earlier, if information about the object's motion is available (such as the object having onboard odometry estimates), this information can be used to construct the factor. Otherwise, the object's motion is modeled as a random walk, with the mean set to the identity transformation and the covariance determined by the knowledge STORM has of the object's pose. We now discuss the different types of object measurements: SegICP, Manipulation, Unobserved, Null.

1. **SegICP** Since SegICP measurements ($\mathbf{z}^{sj}_i$) are 6DoF object pose estimates in the sensor coordinate frame $\mathbf{x}_i$, they fully constrain the object pose estimate with cm and deg accuracy. If an object is observed by SegICP in consecutive frames, its random walk covariance matrix values will be relatively small as SegICP runs at 14Hz and STORM assumes the object will not have moved

more than 1 m between frames (68% of the time even in a dynamic environment). Consequently, multiple SegICP observations of an object will increase the robot's certainty of the object's pose.

2. **Manipulation** Manipulation measurements ($\mathbf{z}_i^{mj}$) also fully constrain the object pose estimate as the end effector pose is found by reading the arm joint encoder values. While an object is being manipulated, its random walk covariance matrix values are based on the accuracy of the encoders and the physical limits of the manipulator in moving the object.

3. **Unobserved** If an object is not observed in the current frame or the previous frame, then STORM does not have any information about the object's motion over the previous time frame. As a result, the random walk covariance is large and based on the estimated dynamicity of the environment.

4. **Null Observation** If an object is not observed in the current frame and STORM estimates the object should be in a sensor's field of view, the object has likely moved. Since STORM has no information about the object's current pose, the random walk covariance is set to be very large.

4.3 Back-End

As described in Chapter 3, finding the most likely sensor trajectory and map state is equivalent to finding the most likely configuration of the graph. This task is a maximum-a-posteriori (MAP) estimation problem. As the robot surveys its surroundings and constructs the factor graph, the front-end of STORM passes the factor graph to the back-end of STORM, which solves the MAP problem by incrementally smoothing the graph.

4.3.1 STORM as a Nonlinear Least Squares Optimization

Now we show how the factor graph constructed by STORM can be turned into a nonlinear least squares optimization. We adopt the formulation of [6, 45], modifying it to account for STORM's factor representation.

First, we show how the STORM problem is a MAP estimate. As mentioned previously in (4.1), the state STORM estimates at time t is the multivariate random variable

$$\mathcal{X} = \{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_t, \mathbf{o}_1^1, \mathbf{o}_2^1, \dots, \mathbf{o}_t^1, \mathbf{o}_1^2, \mathbf{o}_2^2, \dots, \mathbf{o}_t^2, \mathbf{o}_1^m, \mathbf{o}_2^m, \dots, \mathbf{o}_t^m\} \tag{4.2}$$

where $\mathbf{x}_i$ is the estimated 6DoF pose of the sensor at time i (represented by the blue nodes in Figure 4-5) and $\mathbf{o}_i^j$ is the estimated 6DoF pose of object j at time i (represented by the orange nodes in Figure 4-5).

The set of measurements $Z = \{\mathbf{z}_k : k = 1, 2, \dots, n\}$ STORM uses to estimate $\mathcal{X}$ are:

1. **SegICP measurements** of the object poses in the sensor frame (denoted by $\mathbf{z}_i^{sj}$)
2. **Manipulation measurements** of manipulated objects in the sensor frame (denoted by $\mathbf{z}_i^{mj}$)
3. **Odometry measurements** between consecutive sensor poses (denoted by $\mathbf{z}_i^{u}$)
4. **Object motion measurements** between consecutive object poses (denoted by $\mathbf{z}_i^{tj}$)
5. **Absolute sensor pose measurements** (denoted by $\mathbf{z}_i^{a}$)

Each SegICP measurement ($\mathbf{z}_i^{sj}$) is used to define a SegICP factor (represented by a black square in Figure 4-5) and described by the following measurement model:

$$\mathbf{z}_i^{sj} = h_s(\mathbf{x}_i, \mathbf{o}_i^j) \oplus \boldsymbol{\omega}_s \tag{4.3}$$

where $\mathbf{z}_i^{sj}$ is the SegICP measurement of object j at time i, $h_s(\cdot)$ computes the relative transform ${}^{\mathbf{x}_i}\mathrm{T}_{\mathbf{o}_i^j}$ expressing $\mathbf{o}_i^j$ in the frame of $\mathbf{x}_i$, $\boldsymbol{\omega}_s$ is zero-mean Gaussian noise with the information matrix Ω_s, and $\oplus$ is the standard motion composition operator [41] that maps the measurement noise to an element of the manifold of 3D poses, SE(3) [6].

Each manipulation measurement ($\mathbf{z}_i^{mj}$) is used to define an object manipulation factor (represented by a green square in Figure 4-5) and described by the following measurement model:

$$\mathbf{z}_i^{mj} = h_m(\mathbf{x}_i, \mathbf{o}_i^j) \oplus \boldsymbol{\omega}_m \tag{4.4}$$

where $\mathbf{z}_i^{mj}$ is the manipulation measurement of object j at time i, $h_m(\cdot)$ computes the relative transform ${}^{\mathbf{x}_i}\mathrm{T}_{\mathbf{o}_i^j}$ expressing $\mathbf{o}_i^j$ in the frame of $\mathbf{x}_i$, and $\boldsymbol{\omega}_m$ is zero-mean Gaussian noise with the information matrix Ω_m.

Each odometry measurement ($\mathbf{z}_i^u$) is used to define an odometry factor (represented by a blue solid square in Figure 4-5) and described by the following measurement model:

$$\mathbf{z}_i^u = h_u(\mathbf{x}_i, \mathbf{x}_{i+1}) \oplus \boldsymbol{\omega}_u \tag{4.5}$$

where $\mathbf{z}_i^u$ is the odometry measurement at time i, $h_u(\cdot)$ computes the relative transform ${}^i\mathrm{T}_{i+1}$ expressing $\mathbf{x}_{i+1}$ in the frame of $\mathbf{x}_i$, and $\boldsymbol{\omega}_u$ is zero-mean Gaussian noise with the information matrix Ω_u.

Each object motion measurement ($\mathbf{z}_i^{tj}$) is used to define an object motion factor (represented by a orange square in Figure 4-5) and described by the following measurement model:

$$\mathbf{z}_i^{tj} = h_t(\mathbf{o}_i^j, \mathbf{o}_{i+1}^j) \oplus \boldsymbol{\omega}_t \tag{4.6}$$

where $\mathbf{z}_i^{tj}$ is the object motion measurement of object j at time i, $h_t(\cdot)$ computes the relative transform ${}^i\mathrm{T}_{i+1}$ expressing $\mathbf{o}_{i+1}^j$ in the frame of $\mathbf{o}_i^j$, and $\boldsymbol{\omega}_t$ is zero-mean Gaussian noise with the information matrix Ω_t.

Each absolute sensor pose measurement ($\mathbf{z}_i^a$) is used to define an absolute sensor pose factor (represented by a hollow square in Figure 4-5) and described by the following measurement model:

$$\mathbf{z}_i^a = h_a(\mathbf{x}_i) \oplus \boldsymbol{\omega}_a \tag{4.7}$$

where $\mathbf{z}_i^a$ is the absolute sensor pose measurement at time i, $h_a(\cdot)$ computes the absolute pose of $\mathbf{x}_i$ in the frame of the first sensor pose $\mathbf{x}_0$, and $\boldsymbol{\omega}_a$ is zero-mean Gaussian noise with the information matrix Ω_a. If no absolute sensor pose measurements are available, then a prior factor ($\mathbf{z}^p$) is set on the first sensor pose node:

$$\mathbf{z}^p = \mathbf{x}_1 \oplus \boldsymbol{\omega}_p \tag{4.8}$$

where $\mathbf{z}^p$ is the prior mean of $\mathbf{x}_1$ and $\boldsymbol{\omega}_p$ is zero-mean Gaussian noise with the information matrix Ω_p. If there is no prior information on $\mathbf{x}_1$, then it is set to the identity transformation by convention [6].

To simplify notation in the rest of the STORM MAP formulation, we express each of the STORM measurements (z_k) as a function of a subset of $\mathcal{X}$ as follows:

$$\mathbf{z}_k = h_k(\mathcal{X}_k) \oplus \boldsymbol{\omega}_k \tag{4.9}$$

where $h_k(\cdot)$ is a non-linear function that computes the measurement (from the measurement model), $\mathcal{X}_k \subseteq \mathcal{X}$, and $\boldsymbol{\omega}_k$ is zero-mean Gaussian noise with the information matrix Ω_k.

Therefore,

$$\mathbf{z}_k \sim \mathcal{N}\Big(h_k\big(\mathcal{X}_k\big), \Omega_k^{-1}\Big) \tag{4.10}$$

We can formulate the STORM problem as a MAP estimate, where $\mathcal{X}^*$ is the most likely node configuration given all the measurements:

$$\mathcal{X}^* = \underset{\mathcal{X}}{\operatorname{argmax}}\ P(\mathcal{X}|Z) = \underset{\mathcal{X}}{\operatorname{argmax}}\ P(Z|\mathcal{X})P(\mathcal{X}) \tag{4.11}$$

We use Bayes' rule for the last equality and initialize the prior $P(\mathcal{X})$ to a uniform distribution unless we have prior information about $\mathcal{X}$ such as absolute sensor pose measurements as described above (in the case of no prior information, the MAP estimate is equivalent to the maximum likelihood estimate).

Note that (4.11) is in the same form as (3.11) so we can follow the derivation in

Section 3.4.1 to turn (4.11) into a nonlinear optimization problem and solve for $\mathcal{X}^*$. STORM uses iSAM2 [26] from the Georgia Tech Smoothing and Mapping (GTSAM) library [11]. iSAM2 converts the factor graph into a Bayes tree for more efficient incremental smoothing. The reader is directed to [26] for more details on iSAM2.

4.4 Relocalizing with Existing Map

When a robot revisits an environment, STORM can use the map it built previously to localize the robot. Furthermore, STORM updates the map to be consistent with the object observations it makes in the current trial. At the end of a trial, STORM stores its latest estimates of the object poses as well as the relationships (relative poses) between the objects to relocalize the robot in the map in future trials.

To relocalize with an existing map, STORM assumes that objects are not rigidly attached to each other and move independently of each other. Between robot trials, objects can move as long as they do not maintain the same relative poses with respect to each other. With this assumption, STORM can determine where it is in the map when it detects two or more objects in its field of view with the same relative poses (within a threshold).

When relocalizing, STORM constructs a factor graph just like when building a map. STORM assumes that the sensor starts at the first sensor pose of the first trial (which is the origin in the map's coordinate frame) since it has no prior information. Once STORM has determined where it is in the previous map, it updates all the absolute sensor pose measurement and prior factors to reflect where it is in the map. Additionally, STORM adds object pose estimates for objects that are unobserved in the current trial but were observed in previous trials.

Chapter 5

Experiments

In this chapter, we demonstrate that STORM constructs accurate maps of dynamic environments and can use the maps to localize and manipulate objects. We evaluate STORM with simulation and real-world experiments.

First, we describe the robot platform we use to conduct the experiments. Then, we evaluate STORM in simulation, comparing its performance against a baseline pose graph SLAM with landmarks approach described in Chapter 3. We measure the accuracy in estimating the robot's Kinect sensor pose and the object poses as well as the computation speed. Next, we demonstrate STORM's capabilities with a pair of real-world experiments. Finally, we discuss the results.

5.1 STORM Robot Platform

As shown in Figure 5-1, the robot platform used for the STORM consists of a fixed-base torso with an arm and head. The arm is a KUKA LBR iiwa14 R820 with a Robotiq 2-Finger 85 Adaptive Robot Gripper end effector and Intel RealSense SR300 RGB-D sensor mounted at the wrist. The head is a Microsoft Kinect v1 sensor mounted to a FLIR PTU-E46-17P70T pan-tilt unit (PTU). The arm has a position repeatability of ± 0.15 mm, allowing the robot to consistently grasp objects given a 6DoF pose and to accurately put RealSense observations in the Kinect frame. The PTU enables the Kinect to pan 318° and tilt 78° at 0.013° and 0.003° resolution respec-

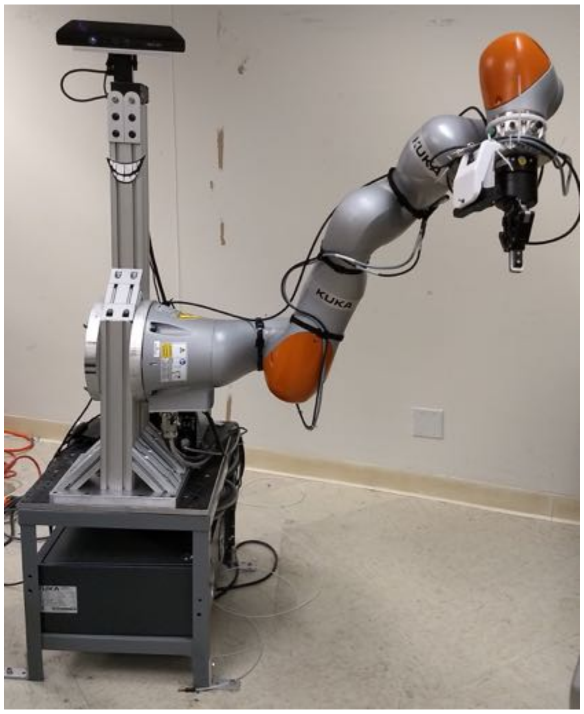


Figure 5-1: **STORM robot platform consists of a fixed-base torso with an arm and head.** The arm has a two-finger gripper and wrist-mounted RealSense RGB-D sensor. The head is a Kinect v1 mounted to a pan-tilt unit that pans 318° and tilts 78°.

tively, giving STORM accurate absolute measurements of the Kinect's orientation. The arm and PTU enable STORM to survey significantly more of the surroundings on the fixed-base platform than would be otherwise possible.

For our experiments, we set the global coordinate frame as the local coordinate frame of the Kinect at initialization (pan and tilt set to 0°).

5.2 Simulations

We evaluate STORM in simulation to quantitatively evaluate its accuracy. As shown in Figure 5-2, we visualize a Unified Robot Description Format (URDF) of the robot platform, Kinect and object pose estimates, and the simulation environment in RVIZ, the ROS 3D visualization tool.

We compare STORM to a baseline pose graph SLAM with landmarks approach as described in Chapter 3, where the objects are the landmarks. It takes the same

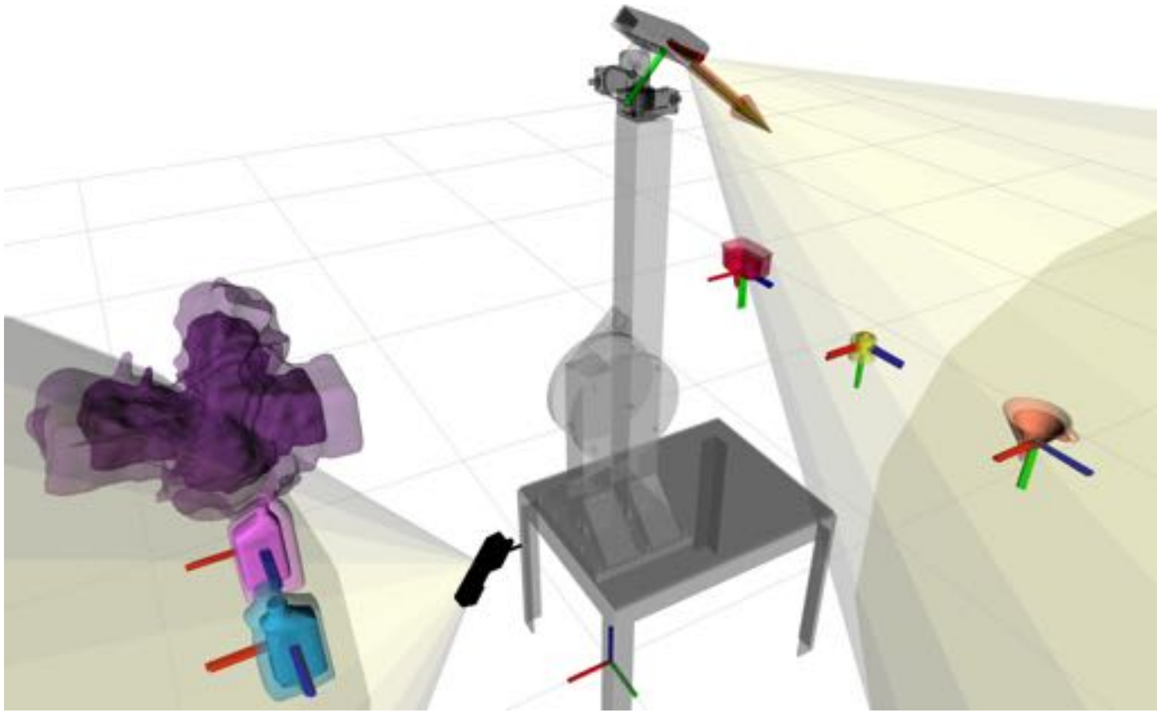


Figure 5-2: **Visualization of STORM estimates and the simulation environment** We visualize the STORM robot platform, estimated and ground truth poses of the Kinect and objects as coordinate frames, Kinect and RealSense field of views, and the simulation environment in RVIZ. We visualize the positive z-axis of the Kinect sensor ground truth pose as an opaque green arrow and the positive z-axis Kinect pose estimate as a translucent red arrow. Similarly, we render an object mesh model at the object's true pose as opaque and at 75% scale and at the object's estimated pose as translucent and at full scale. Note how the estimated and ground truth coordinate frames are closely aligned and visualizations are closely aligned.

measurements as STORM, but assumes that the objects are static. Both STORM and the baseline use the same relocalization algorithm as described in Section 4.4.

5.2.1 Simulation Robot Platform

We use the STORM robot platform as described in Section 5.1 with a few changes. To increase the space of scenarios to test, we relax the physical constraints of arm reachability and base mobility. We enable the robot to manipulate any object it has a pose estimate of. Additionally, we enable the robot to move the RealSense to any pose, greatly increasing the observable region of the environment. Finally, we enable the fixed-base platform to move, which allows for a more compelling demonstration of relocalizing with a previous map.

5.2.2 Generating Measurements in Simulation

We now discuss how we obtain measurements from the simulation environment. As discussed in Section 4.2.1, STORM handles 5 types of measurements: odometry measurements ($\mathbf{z}_i^u$), absolute sensor pose measurements ($\mathbf{z}_i^a$), object motion measurements ($\mathbf{z}_i^{tj}$), SegICP measurements ($\mathbf{z}_i^{sj}$), and manipulation measurements ($\mathbf{z}_i^{mj}$).

Odometry and Absolute Sensor Pose Measurements

When the robot moves its head, STORM takes odometry and absolute sensor pose measurements. To get the absolute sensor pose measurement, STORM looks up the transform from the robot base frame to the Kinect frame. We add measurement noise which we generate from a zero-mean Gaussian with $\sigma_{pitch} = 0.003°$, $\sigma_{yaw} = 0.013°$, $\sigma_{roll} = 0.00°$, and $\sigma_{trans} = 0.0$m for the translation pose components (from the pan-tilt unit specifications).

To generate odometry measurements, we compute the transform from the Kinect pose at the previous time-step to the Kinect pose at the current time-step. We add the same measurement noise as the absolute sensor pose measurement.

SegICP Measurements

At each time-step, STORM checks for objects in the field of view (FOV) of the Kinect or RealSense. The Kinect has a vertical FOV of 43°, horizontal FOV of 57°, and max range of 4m. The RealSense has a vertical FOV of 55°, horizontal FOV of 71.5°, and max range of 1.5m.

In the simulation environment, we do not account for occlusion. If an object's coordinate frame is in the FOV view cone of a sensor, it is considered to be observable.

STORM extracts a SegICP measurement for each object in a sensor's FOV. To get the SegICP measurement, STORM looks up the transform from the sensor frame to the object frame. We add measurement noise which we generate from a zero-mean

Gaussian with $\sigma_{trans} = 0.005$ m and $\sigma_{rot} = 1.0°$ (from [56, 57]).

Manipulation Measurements

At each time-step, STORM checks if it is manipulating an object. If it is, STORM extracts a manipulation measurement by looking up the transform from the Kinect frame to the object frame. We add measurement noise which we generate from a zero-mean Gaussian with $\sigma_{trans} = 0.015$ m and $\sigma_{rot} = 0.5°$ (modeling our uncertainty of the object's pose in the gripper).

5.2.3 Evaluation Metrics

We compare the performance of STORM with that of a standard pose graph SLAM with landmarks approach as a baseline. To quantitatively evaluate performance, we compare each object and Kinect pose estimate to its respective ground truth pose.

We use two common accuracy metrics from the SLAM literature to evaluate each estimate: the absolute trajectory (ATE) root-mean-square error (RMSE) metric [44] to measure position error and quaternion root-mean-square error (RMSE) [45] to measure orientation error. We also compare the computational performance of STORM with the baseline.

Absolute trajectory (ATE) root-mean-square error (RMSE)

The absolute trajectory (ATE) root-mean square error (RMSE) is the root-mean-square of the trajectory's position error between each estimated pose with its corresponding ground truth pose [44]. It is defined as:

$$\text{RMSE}_{\text{ATE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_i^{x,y,z} - \hat{x}_i^{x,y,z}\right)^2} \tag{5.1}$$

where $x_i^{x,y,z}$ is the i-th estimated position, $\hat{x}_i^{x,y,z}$ is the i-th ground truth position, and $x_i^{x,y,z} - \hat{x}_i^{x,y,z}$ is the Euclidean distance between the estimated and ground truth position [45].

Quaternion root-mean-square error (RMSE)

The quaternion root-mean square error (RMSE) is the root-mean-square of the trajectory's orientation error between each estimated pose with its corresponding ground truth pose [45]. It is defined as:

$$\text{RMSE}_{\text{quaternion}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\Big(\cos^{-1}(2\langle q_i, \hat{q}_i\rangle^2 - 1)\Big)^2} \tag{5.2}$$

where q_i is the quaternion representing the i-th estimated orientation, $\hat{q}_i$ is quaternion representing the i-th ground truth orientation, and $\cos^{-1}(2\langle q_i, \hat{q}_i\rangle^2 - 1)$ is the angle of rotation between the estimated and ground truth quaternion. We use the Eigen implementation of (5.2).

Computation Time

To quantitatively compare the computational performance of STORM with the baseline, we record the time to perform a graph update — that is to update and optimize the respective factor graph representations. We do not include the simulation measurement extraction processes as they are the same and take a negligible amount of time as they involve looking up a transform. The test platform is a laptop with an Intel Core i7-7820HK CPU with 4 2.9 GHz cores and 32GB of RAM.

5.2.4 Simulation Experiments

We design two experiments to demonstrate the capabilities of STORM in simulation. In these experiments, the robot observes and manipulates objects in the environment. The first experiment, which we call 'Mapping', demonstrates STORM's ability to create and maintain an accurate object map of a dynamic environment and use the object pose estimates to manipulate objects. The second experiment, which we call 'Relocalization', demonstrates STORM's ability to relocalize with the map created in the Mapping experiment, update the map, and use the estimates to manipulate objects. We assume that the robot is able to grasp an object even if its pose estimate is significantly off (by other means such as visual servoing). For reference, we mention the error of each object pose estimate when the object is grasped in Section 5.2.5.

Experiment 1: Mapping

In this experiment, the robot has no prior knowledge of the environment and builds an initial map of the dynamic environment and manipulates some objects.

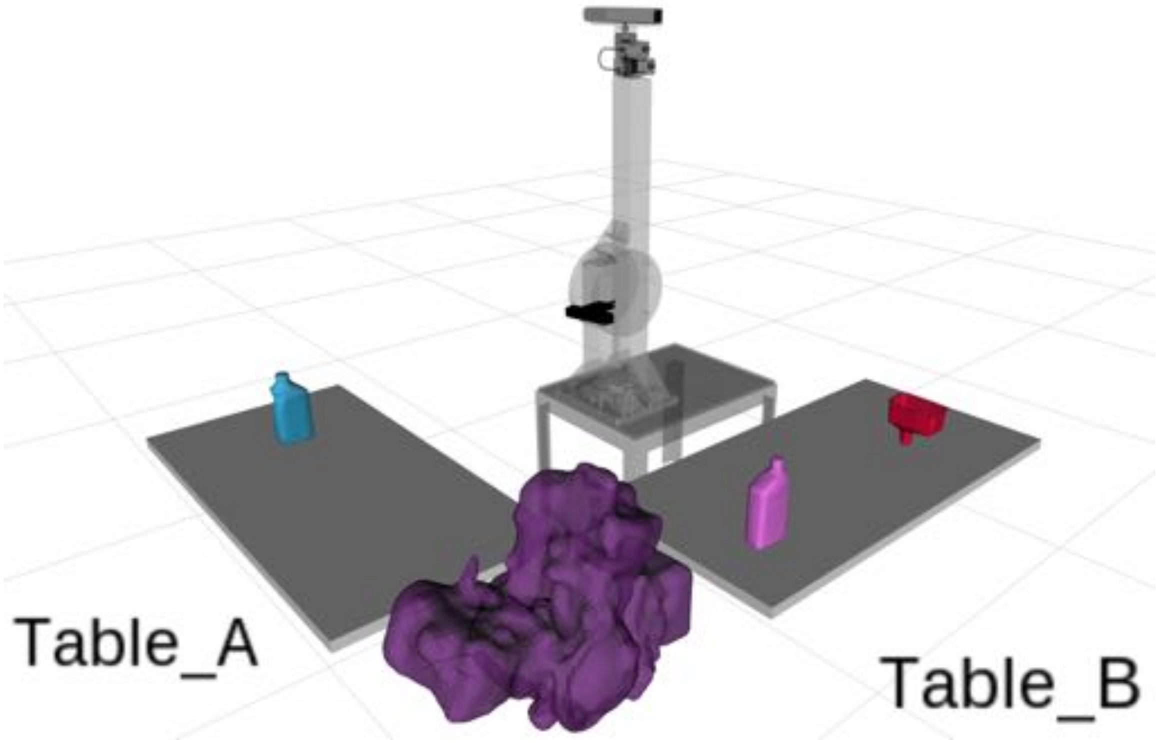


Figure 5-3: **Setup of Simulation Experiment 1: Mapping** Note that the RealSense sensor (small black object in the the middle of the image) appears to be floating and the robot arm is not visible as we relax all physical constraints in simulation for a larger space of scenarios.

At the start of the experiment, the robot platform's base frame is set at the origin of the world frame and various objects are placed in the surrounding environment. As shown in Figure 5-3, there is a table in front of the robot with an blue bottle on it (table A), a table to the left of the robot with a pink bottle and red funnel on it (table B), and a purple engine between table A and table B.

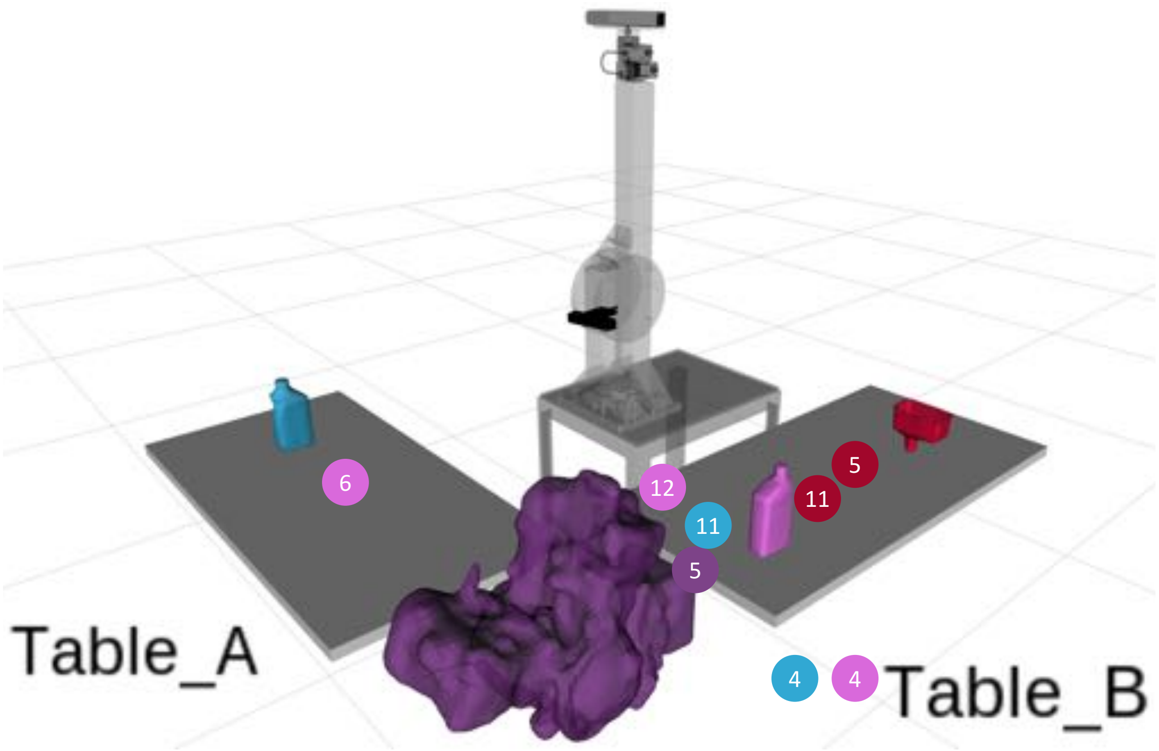


Figure 5-4: **Steps of Simulation Experiment 1: Mapping** Each colored circle depicts the approximate location of the similarly colored object at the displayed step.

As depicted in Figure 5-4, the Mapping experiment is as follows:

1. The robot begins surveying the environment and building a map, observing the blue bottle on table A with the Kinect and RealSense.
2. The robot observes the purple engine with the Kinect and RealSense.
3. The robot observes the pink bottle and red funnel on table B with the Kinect and RealSense.
4. The blue bottle and pink bottle are moved under table B, out of the field of views of the Kinect and RealSense.
5. The purple engine and red funnel are moved a few inches.
6. The pink bottle is placed on table A, close to where the blue bottle was.
7. The robot surveys the environment again now in reverse, first reobserving the red funnel on table B with the Kinect and RealSense and updating the map.

8. The robot reobserves the purple engine with the Kinect and RealSense.

9. The robot reobserves the pink bottle on table A with the Kinect and RealSense.

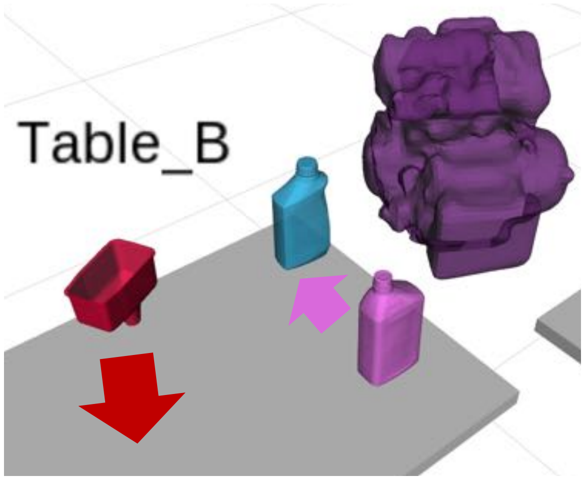
10. The robot uses the pink bottle pose estimate to grasp the bottle.

11. The red funnel is moved and the blue bottle appears back on table B.

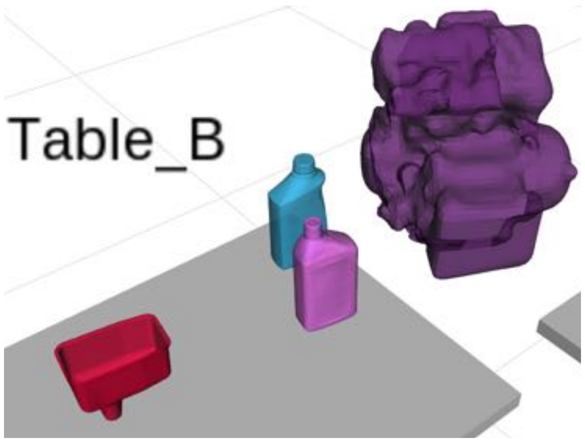
12. The robot places the pink bottle next to the blue bottle on table B

13. The robot reobserves table B.

Before Experiment 2 starts, the red funnel and pink bottle are moved independently on table B so that they do not have the same relative pose with respect to each other as shown in Figure 5-5. The blue bottle and purple engine are left untouched. Additionally, the robot platform's base frame is translated two meters in the x and y direction from the origin of the world frame and rotated 45° as shown in Figure 5-6.



(a)



(b)

Figure 5-5: **Object movement before Experiment 2** Note how the pink bottle and red bottle move as indicated on the arrows in Figure (a). The resulting environment before Experiment 2 is depicted in Figure (b).

Experiment 2: Relocalization

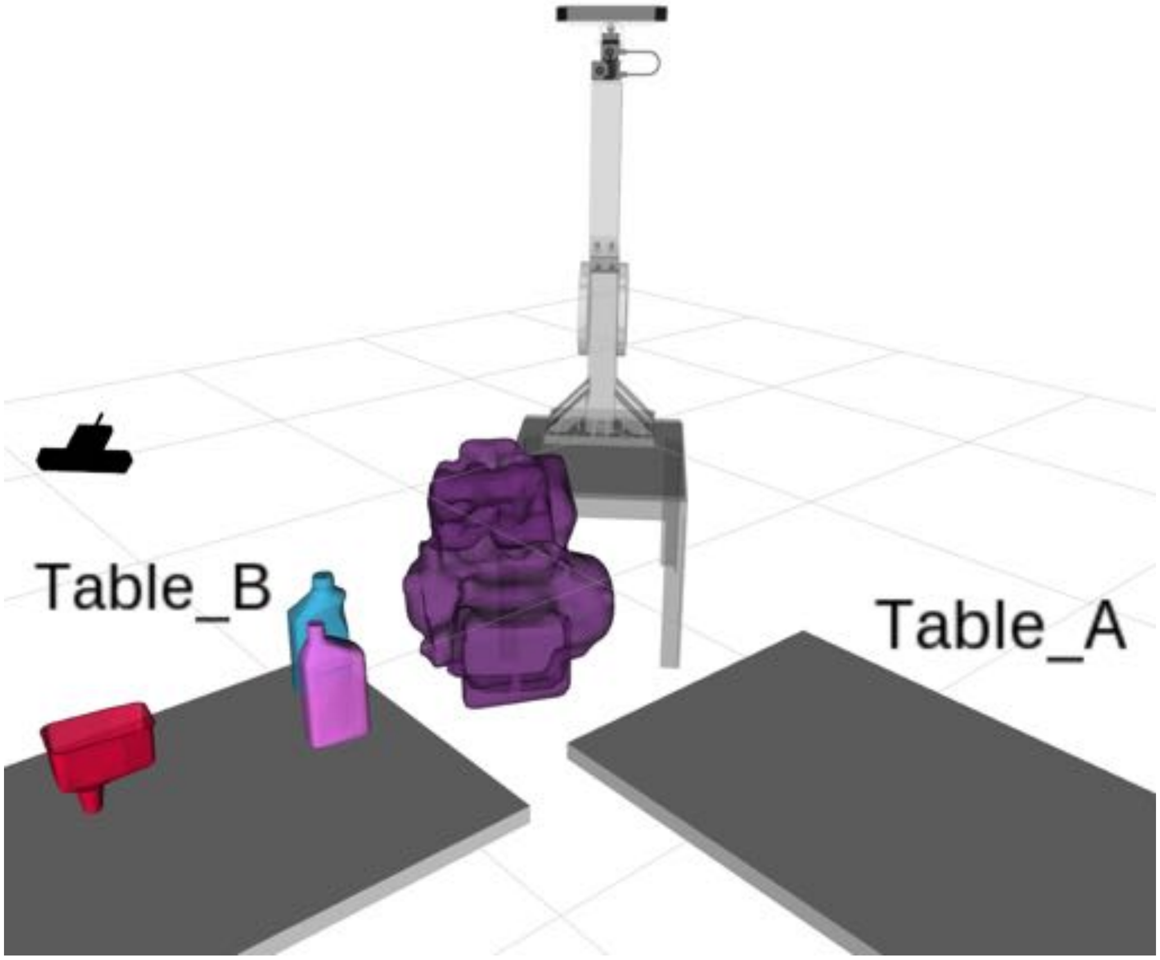


Figure 5-6: **Setup of Simulation Experiment 2: Relocalization** Note that the robot base has been translated significantly from its position in Experiment 1 and rotated 45°.

In this experiment, the robot knows that it is returning to the environment it visited in Experiment 1, but it does not know where it is in the environment. The robot attempts to relocalize itself with the map it created in Experiment 1, update the map with the observed changes in the environment, and manipulate some of the objects.

As depicted in Figure 5-7, the Relocalization experiment is as follows:

1. The robot observes the red funnel on table B with the RealSense.

2. The robot observes the pink bottle on table B with the RealSense.

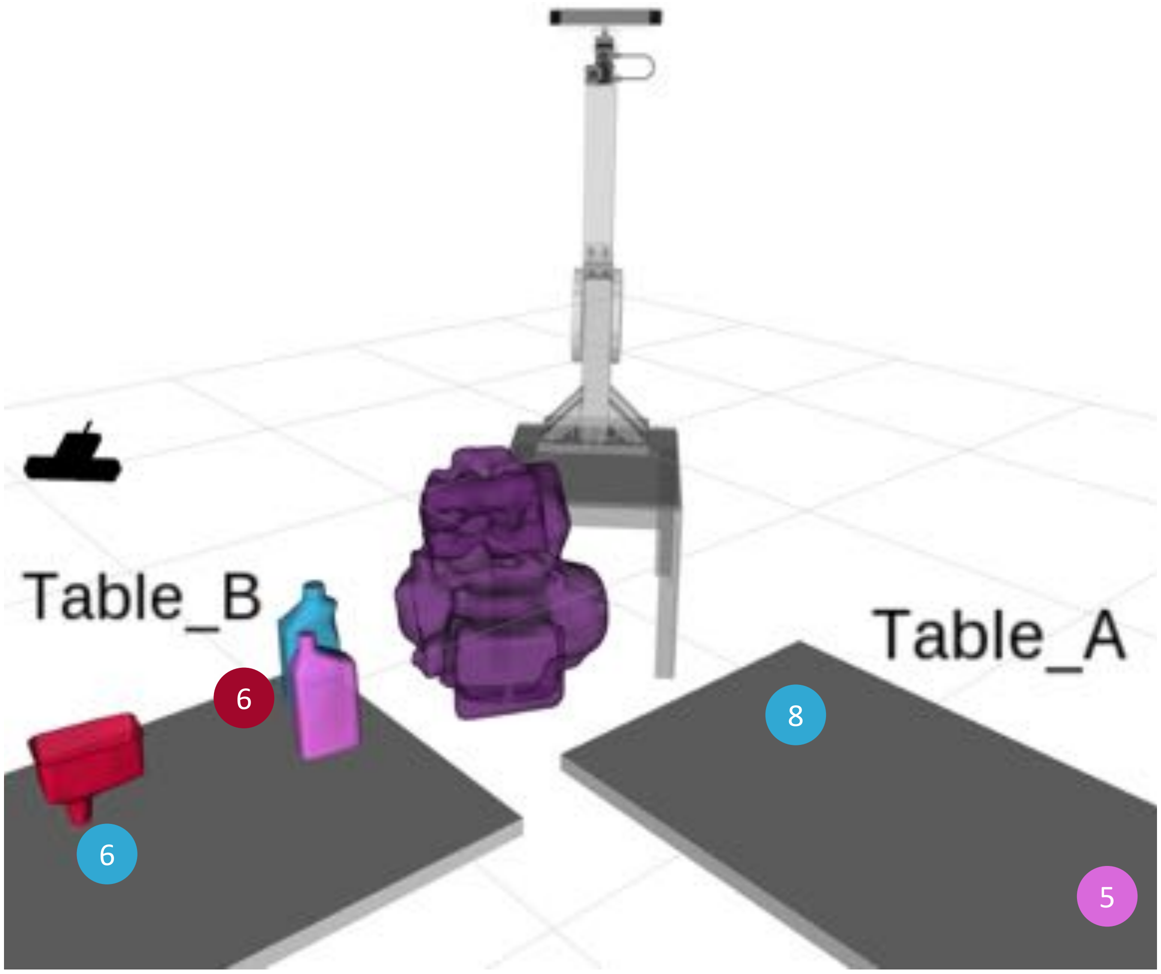


Figure 5-7: **Steps of Simulation Experiment 2: Relocalization** Each colored circle depicts the approximate location of the similarly colored object at the displayed step.

3. The robot observes the blue bottle on table B with the RealSense.

4. The robot observes the blue bottle on table B and the purple engine with the RealSense.

5. The robot then grasps the pink bottle and places it on table A.

6. The red funnel and blue bottle swap places in the meantime.

7. The robot reobserves table B with the Kinect.

8. The robot grasps the blue bottle and places it on table A.

5.2.5 Experimental Results

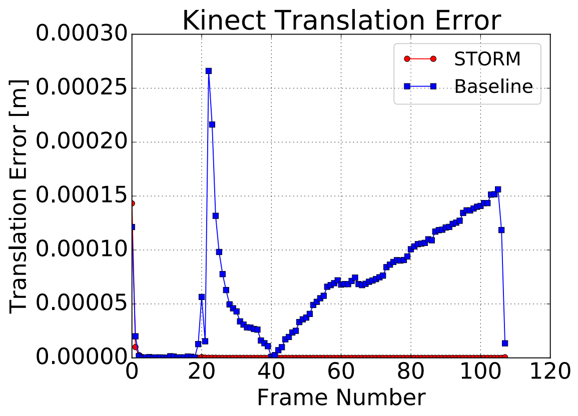
In this section, we discuss the results of the simulation experiments. We compare the performance of STORM with the performance of the baseline (pose graph SLAM with landmarks) evaluated over 24 trials by looking at their respective Kinect and object pose estimates during the experiments. We only evaluate object estimates when they are observed. In the case that an object is moved independently and is unobserved, the error would increase for both systems equally. Note that when an object moves and STORM looks at the previous object location and observes that the object is not there, STORM makes a null observation of the object. STORM now knows that the object is not there and has moved. In comparison, the baseline approach assumes the object is still there and attempts to reconcile new observations of the moved object with the previous observations, leading to greater errors in its estimates.

Experiment 1: Mapping

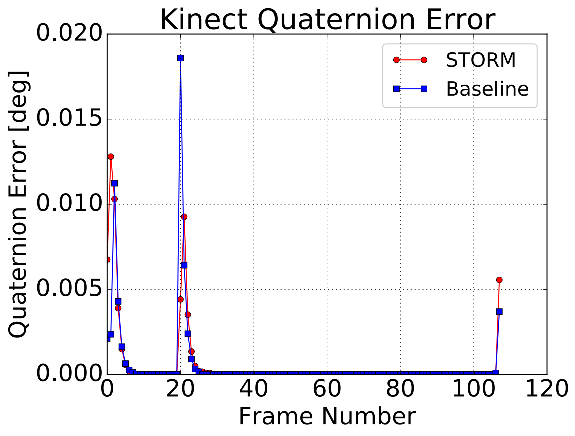
Table 5.1: **STORM and Baseline Error in Experiment 1 Over 24 Trials**

Sensors / Objects	**STORM ATE RMSE [m]** (μ, σ)	**STORM Quaternion RMSE [deg]** (μ, σ)	**Baseline ATE RMSE [m]** (μ, σ)	**Baseline Quaternion RMSE [deg]** (μ, σ)
kinect	1e-5, 1e-5	0.003, 0.0007	0.0001, 1e-5	0.003, 0.0007
Red Funnel	0.004, 0.001	1.4, 0.3	0.09, 0.001	61.0, 0.1
Pink Bottle	0.004, 0.0002	0.9, 0.03	0.3, 0.0004	22.5, 0.05
Purple Engine	0.007, 0.002	1.1, 0.4	0.07, 0.004	28.5, 0.3
Blue Bottle	0.007, 0.002	1.6, 0.6	0.3, 0.001	3.2, 0.2

Table 5.1 reports the mean and standard deviation of the ATE RMSE (5.1) and quaternion RMSE (5.2) of STORM and the baseline estimates in Experiment 1 over 24 trials. The table confirms that STORM builds more accurate maps of dynamic



(a)



(b)

Figure 5-8: **STORM vs. Baseline Kinect Pose Estimate Error in Experiment 1** The plots depict the Kinect pose estimate translation and quaternion error from STORM and the baseline during one trial of Experiment 1. Both estimate the Kinect's pose quite well. Note that the baseline estimates begin to stray around frame 20 when the moved objects are reobserved, though the error is kept low by the absolute sensor pose measurements.

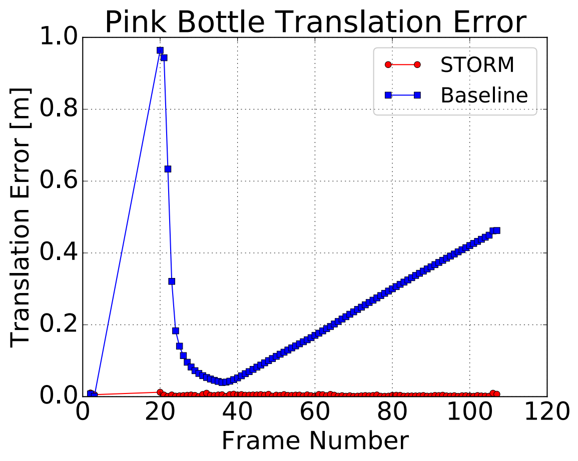
Table 5.2: **STORM and Baseline Pose Error when Grasping Pink Bottle in Experiment 1 Over 24 Trials**

STORM Translation Error [m] (μ, σ)	**STORM Quaternion Error [deg]** (μ, σ)	**Baseline Translation Error [m]** (μ, σ)	**Baseline Quaternion Error [deg]** (μ, σ)
0.005, 0.003	1.1, 0.4	0.9, 0.005	67.9, 0.3

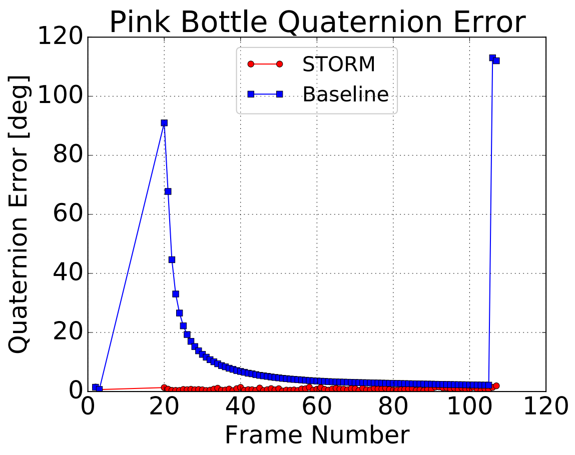
environments than the baseline approach.

Figure 5-8 compares the Kinect pose estimates of STORM and the baseline from one trial of Experiment 1. Both the STORM and baseline estimates are highly accurate due to the accurate absolute sensor pose measurements. The baseline estimates begin to drift when the moved objects are reobserved, though the error is kept low by the absolute pose measurements.

Figure 5-9 compares the pink bottle pose estimates of STORM and the baseline from one trial of Experiment 1. The STORM estimates remain accurate despite objects being manipulated and moving independently as shown in Table 5.1 and Figure 5-9. As reported in Table 5.2, the robot is able to grasp the pink bottle using the STORM estimates with high accuracy over the 24 trials of Experiment 1.



(a)



(b)

Figure 5-9: **STORM vs. Baseline Pink Bottle Pose Estimate Error in Experiment 1** The plots depict the pink bottle pose estimate translation and quaternion error from STORM and the baseline during one trial of Experiment 1. Note how the baseline translation error quickly grows once the pink bottle is moved and manipulated starting at frame 20.

The baseline does not have as accurate estimates since it assumes that the objects are all static. As a result, the baseline assumes that the pink bottle remains at the same pose over the entire experiment and smooths pink bottle measurements from when the bottle is at table A and at table B. This incorrect assumption leads to the high error in the pink bottle pose estimates as shown in Figure 5-9. The same issue is what leads to the poor estimates of the other objects with the baseline such as the blue bottle. In comparison to the STORM estimate, the robot grasps the pink bottle with a baseline pose estimate that is not accurate (shown in Table 5.2). This amount of error would not be good enough for a manipulator using just the pose estimate to grasp the pink bottle.

Plots of the other object pose estimates depict similar results and are located in Appendix A.

Experiment 2: Relocalization

Table 5.3 reports the mean and standard deviation of the ATE RMSE (5.1) and quaternion RMSE (5.2) of STORM and the baseline estimates in Experiment 2 over

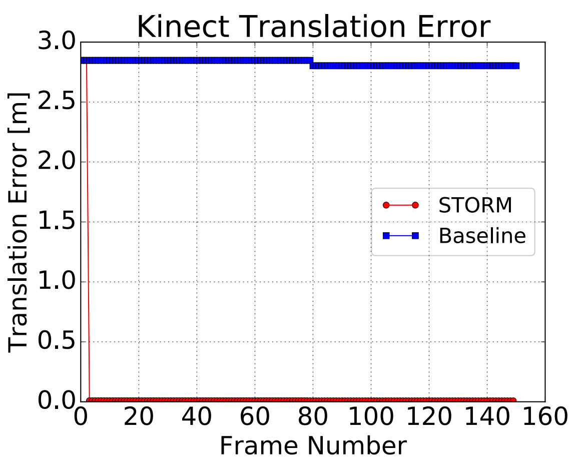
24 trials. The table confirms that STORM is able to consistently relocalize itself in the map in future trials, whereas the baseline was unable to.

Table 5.3: **STORM and Baseline Error in Experiment 2 Over 24 Trials** (values in parentheses are computed after relocalization)

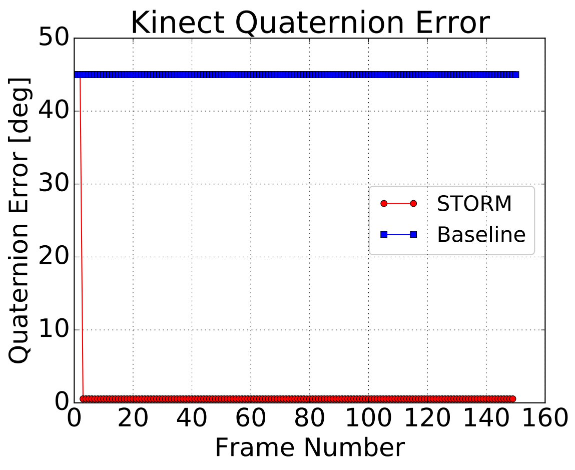
Sensors / Objects	**STORM ATE RMSE [m]** (μ, σ)	**STORM Quaternion RMSE [deg]** (μ, σ)	**Baseline ATE RMSE [m]** (μ, σ)	**Baseline Quaternion RMSE [deg]** (μ, σ)
kinect	0.5, 0.1 (0.04, 0.02)	7.3, 1.1 (1.5, 0.7)	2.8, 4e-6	45.0, 0.0003
Red Funnel	0.5, 0.1 (0.02, 0.003)	10.2, 1.5 (2.3, 0.5)	2.4, 0.001	57.5, 0.2
Pink Bottle	0.3, 0.1 (0.02, 0.01)	6.4, 1.2 (1.8, 0.7)	2.8, 0.0004	77.8, 0.1
Purple Engine	0.1, 0.1 (0.01, 0.01)	3.7, 2.1 (2.2, 1.1)	2.5, 0.002	44.9, 0.4
Blue Bottle	0.4, 0.1 (0.01, 0.004)	6.9, 2.1 (1.8, 0.6)	2.6, 0.0004	38.3, 0.1

Figure 5-10 displays the Kinect pose estimates of STORM and the baseline from one trial of Experiment 2. STORM and the baseline perform very differently. Initially, the robot does not know where it is and assumes that it is at the origin when it is actually translated 2 meters in the x and y direction and rotated 45°. Hence, the translation and quaternion error are high. However, once STORM is able to relocalize itself in the environment, the translation and quaternion error drop significantly. The baseline approach is unable to relocalize using the same technique as STORM since objects moved and were manipulated in Experiment 1 so the object pose estimates were off at the end of Experiment 1 as shown in Figure 5-9. Note that had the baseline approach localized off the first object it observed, the Kinect estimate error would be dependent on how far the object had moved since the previous trial and would not necessarily be reduced.

Figure 5-11 compares the blue bottle pose estimates of STORM and the baseline



(a)



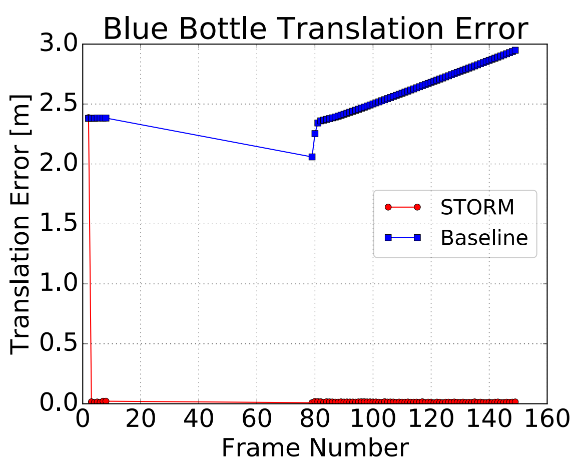
(b)

Figure 5-10: **STORM vs. Baseline Kinect Pose Estimate Error in Experiment 2** The plots depict the Kinect pose estimate translation and quaternion error from STORM and the baseline during Experiment 2. Note how both translation estimates are significantly off initially but after relocalization on frame 3, the STORM estimate error drops to levels near those on Experiment 1. The baseline approach is unable to relocalize and the translation error stays high.

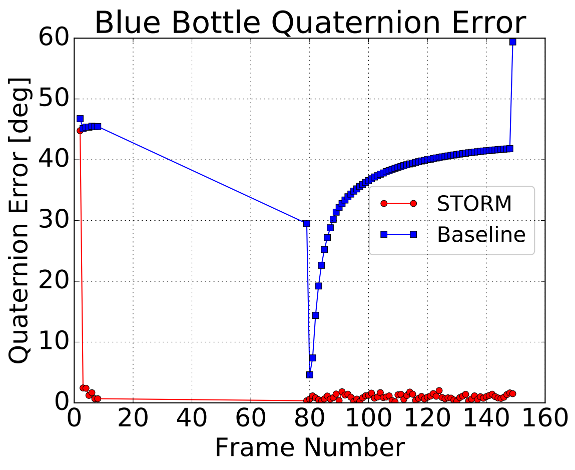
from one trial of Experiment 2. The STORM and baseline object estimate errors are due to the same causes behind the object pose errors in Experiment 1 and the Kinect pose errors in Experiment 2. Both the STORM and baseline estimates are initially off significantly, with the STORM estimates improving after relocalization. The baseline estimates do not improve as the baseline is unable to relocalize and actually worsen since the baseline assumes that the blue bottle remains static.

Table 5.4 reports the mean and standard deviation of the translation and quaternion error of the STORM and baseline object pose estimates when grasping objects in Experiment 2 over 24 trials. The table shows the large discrepancy in accuracy at the time of grasping the objects. A manipulator would not be able to grasp the pink bottle or blue bottle with the baseline estimates, whereas it could with STORM's as will be demonstrated in the real-world experiments.

Plots of the other object pose estimates depict similar results and are located in Appendix A.



(a)



(b)

Figure 5-11: **STORM vs. Baseline Blue Bottle Pose Estimate Error in Experiment 2** The plots depict the blue bottle pose estimate translation and quaternion error from STORM and the baseline during Experiment 2. Note how both translation estimates are significantly off initially but after relocalization on frame 3, the STORM estimate error drops to levels near those in Experiment 1. The baseline approach is unable to relocalize and suffers from the same issues once the objects begin to move or are manipulated so the translation error stays high.

Computational Performance

The mean time for STORM to perform a graph update was 4.3 ms with a standard deviation of 0.2 ms. The mean time for the baseline to perform a graph update was 2.4 ms with a standard deviation of 0.1 ms. Both of these graph update times are more than sufficient for real-time performance as the object pose estimator SegICP operates at 14 Hz.

Table 5.4: **STORM and Baseline Pose Error when Grasping Objects in Experiment 2 Over 24 Trials**

Objects	**STORM Translation Error [m]** (μ, σ)	**STORM Quaternion Error [deg]** (μ, σ)	**Baseline Translation Error [m]** (μ, σ)	**Baseline Quaternion Error [deg]** (μ, σ)
Pink Bottle	0.01, 0.01	2.4, 1.1	2.4, 0.002	45.0, 0.4
Blue Bottle	0.02, 0.01	2.1, 1.2	2.1, 0.002	29.4, 0.3

5.3 Real-World Experiments

We use real-world experiments to demonstrate that STORM is able to build an accurate object map of a dynamic environment and use the map to localize and manipulate objects in the real world. We use the robot platform discussed in Section 5.1. Since the platform is a fixed-base setup, the arm has physical reachability constraints, which limits the space of the manipulatable environment. Like the simulation experiments, we visualize a Unified Robot Description Format (URDF) of the robot platform, STORM estimates, and real-world sensor data in RVIZ as shown in Figure 5-12.

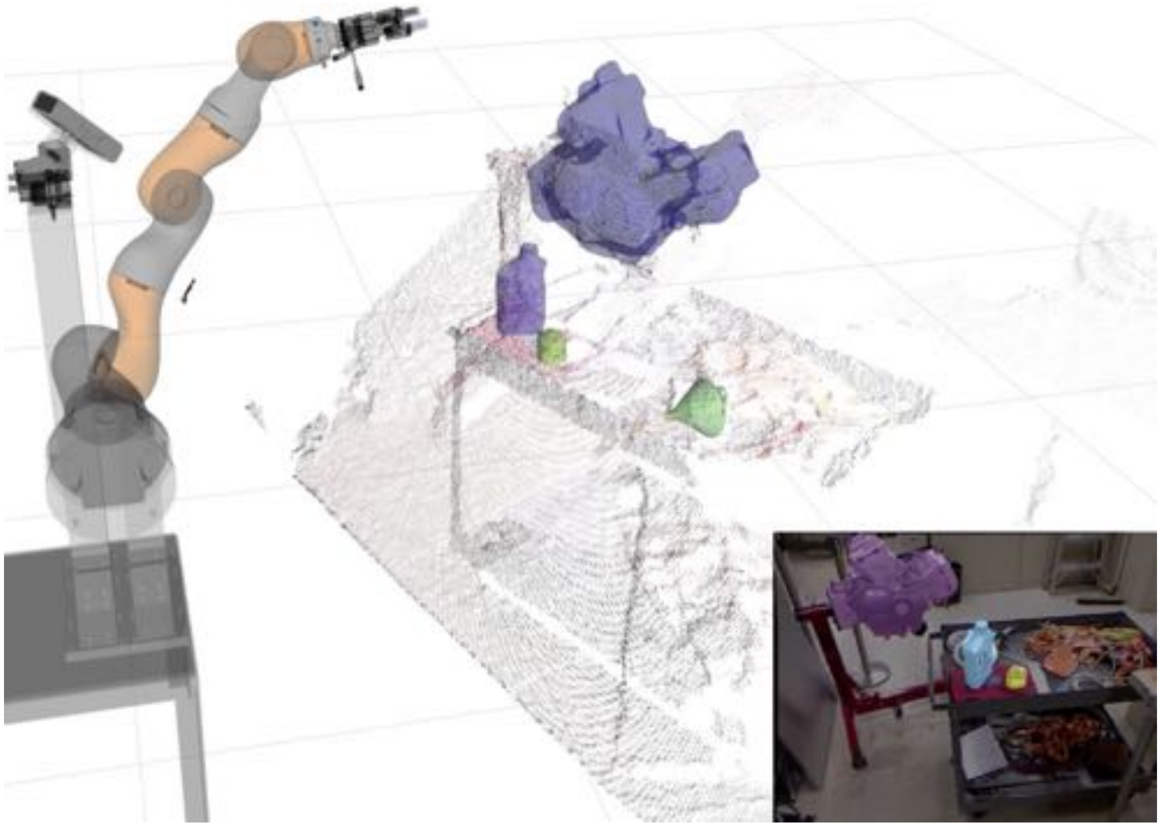


Figure 5-12: **Visualization of STORM estimates and real-world sensor data** We visualize the STORM robot platform, STORM estimates with mesh models, and real-world sensor data in RVIZ. The bottom right image is the RGB image from the Kinect overlaid with pixel-level semantic segmentation from SegNet. Note how closely the object meshes align with the objects in the raw point cloud.

5.3.1 Experiments

We design two experiments to demonstrate STORM's capabilities in a real-world environment. The first experiment, which we call 'Mapping and Relocalization', demonstrates STORM's ability to create and maintain an accurate object map of a dynamic environment and use the map to relocalize itself in the environment over many trials. The second experiment, which we call 'Manipulation', demonstrates STORM's ability to use its pose estimates of objects to grasp objects and update the map.

Experiment 1: Mapping and Relocalization

In this experiment, STORM has no prior knowledge of the environment and builds an initial map of the scene (shown in Figure 5-13).

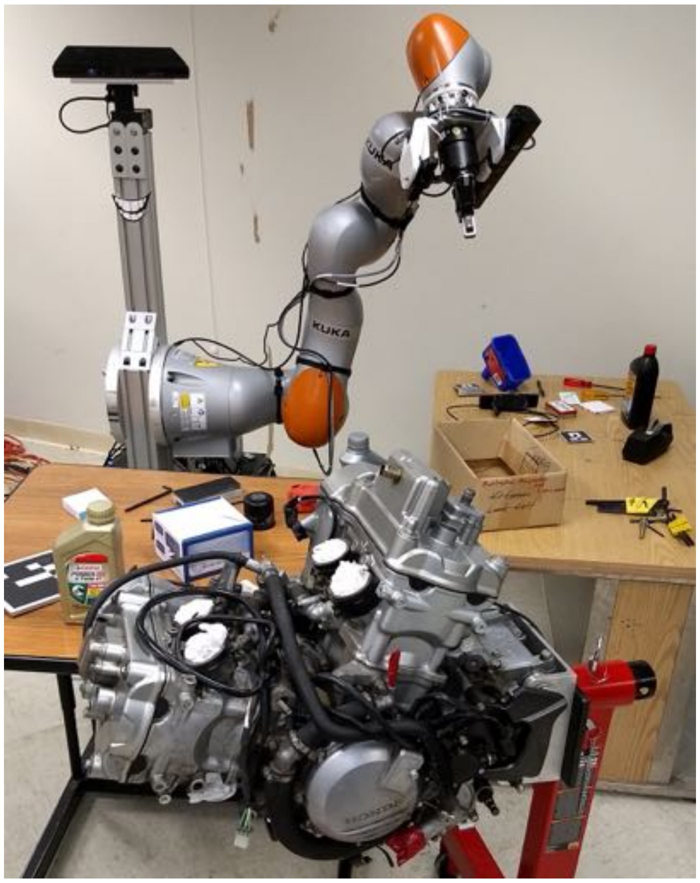


Figure 5-13: **Setup of Real-World Experiment 1: Mapping and Relocalization** Objects of interest (oil bottle, blue funnel, black power steering fluid bottle, engine) are placed on and behind two tables with clutter. Objects are moved by humans as the robot surveys the scene and relocalizes in it over 24 trials.

As people move objects in the scene, STORM maintains the map as it reobserves the objects. Over the course of 24 trials, STORM relocalizes the robot from various viewpoints even as some objects are moved between trials. Note that there are always at least two objects that are not moved so that STORM can use them to relocalize, though STORM is not informed which objects are moved or not.

Experiment 2: Manipulation

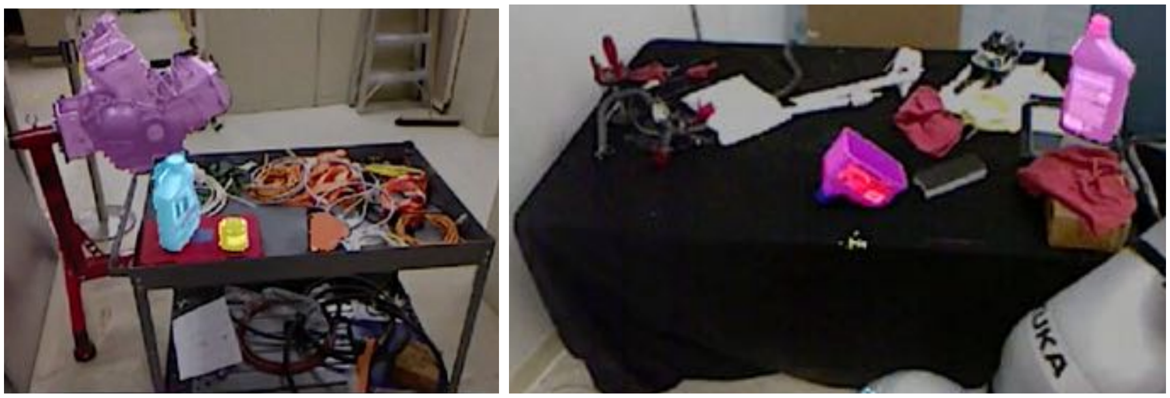


Figure 5-14: **Table Setups of Real-World Experiment 2: Manipulation** Figure (a) is of the table in front of the robot platform and Figure (b) is of the table to the left of the robot platform. The colored overlays are pixel-level semantic segmentation from SegNet. In this experiment, the robot uses STORM's object pose estimates to swap the oil bottle on the table in Figure (a) (overlaid in blue) with the power steering fluid bottle on the table in Figure (b) (overlaid in pink).

In this experiment, STORM again has no prior knowledge of the environment and builds an initial map of the scene. The setup is similar to that in Figure 5-13 but with the table setups shown in Figure 5-14.

STORM uses the pose estimates to swap the oil bottle on one table (overlaid in blue) with the power steering fluid bottle on another table (overlaid in pink). STORM then updates the map to reflect the current scene.

5.3.2 Experimental Results

In this section, we discuss the results of the real-world experiments. Note that we do not have ground truth pose information for the Kinect or the objects so we are not able to evaluate the results in the same way as the simulation experiments.

Experiment 1: Mapping and Relocalization

Figure 5-15 depicts some of the object pose estimates from the experiment.

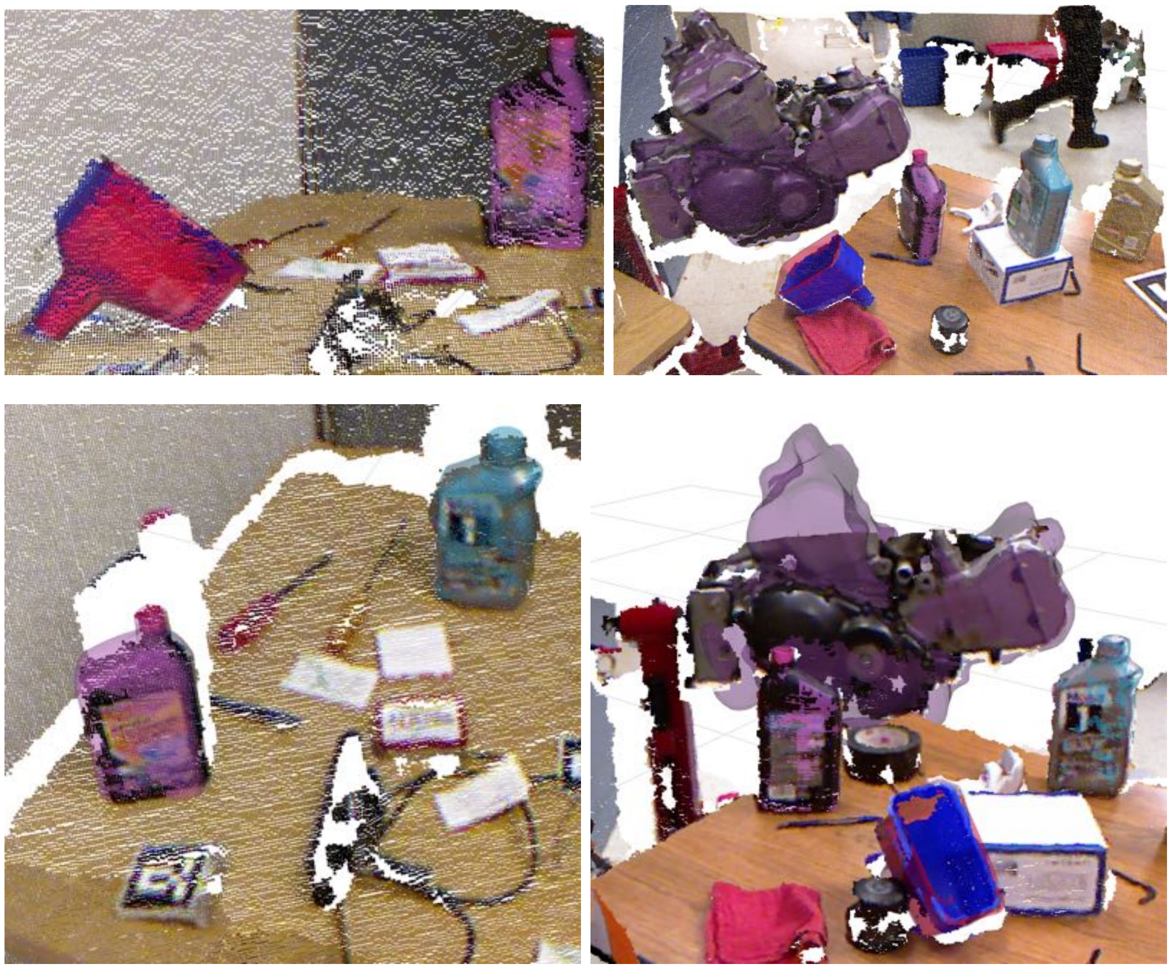


Figure 5-15: **Various Object Pose Estimates in Real-World Experiment 1** Object pose estimates from various trials in Real-World Experiment 1 are shown as object mesh models rendered at their estimated poses. Note how closely the object meshes align with the scene point cloud. The blue mesh corresponds to the gray oil bottle, the red mesh corresponds to the blue funnel, the pink mesh corresponds to the black bottle, and the purple mesh corresponds to the engine.

STORM was able to successfully build accurate maps of the dynamic environment and relocalize from the different viewpoints in the 24 trials.

Since the robot platform is fixed to the ground, we are unable to move the robot to a new location and relocalize with the Kinect. Though we do not have ground truth of the Kinect pose, we have accurate Kinect pose measurements from the encoders in the pan-tilt unit. If we are able to localize using the Kinect, we can compare the estimated Kinect pose is with the actual Kinect pose.

STORM was able to relocalize with translation error with mean of 0.026 m and standard deviation of 0.020 m and quaternion error with mean of 1.4° and standard deviation of 1.1°. STORM relocalized with lower error when relocalizing from a view similar to the view from which the map was last updated.

Experiment 2: Manipulation

STORM was able to successfully swap the oil bottle and power fluid steering bottle and update the map as shown in Figure 5-16.

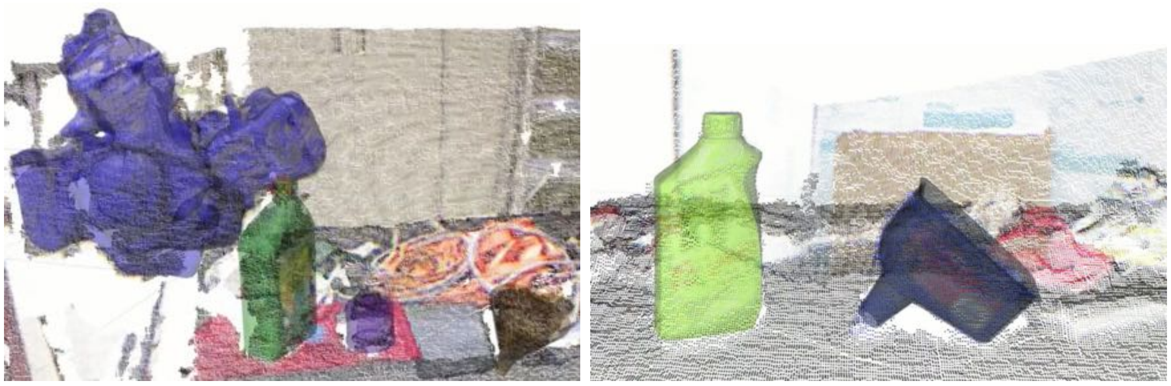

(a) (b)

Figure 5-16: **Object Pose Estimates in Real-World Experiment 2** Object pose estimates from Real-World Experiment 2 are shown as object mesh models rendered at the estimated pose. Note how closely the object meshes align with the raw point cloud.

Though we do not have ground truth poses of objects in the scene, we have some bounds on the pose tolerances for a successful grasp from the object dimensions and the gripper limits. The Robotiq 2-Finger 85 Adaptive Robot Gripper opens to 85 mm wide. Since the oil bottle is 55 mm wide and the black bottle is 60 mm wide, we have 2.5 cm of tolerance for a successful grasp if the pose estimate is perfectly aligned and less if not.

5.4 Discussion

As shown by the baseline results in Section 5.2.5, a SLAM approach that relies on the static world assumption in a dynamic environment will have inaccurate estimates of dynamic objects. Another approach that relies on the static world assumption but instead ignores moving objects such as SLAM++ [39] would also have inaccurate estimates of dynamic objects. In these cases, the estimates would be not be good enough to manipulate the objects using the pose estimates alone. Additionally, as shown in Simulation Experiment 2, these inaccurate estimates leads to an inability to relocalize using the map in future trials.

One case not demonstrated in the experiments is if an object disappears from the robot's sensors' field of view. STORM treats this case as a null observation of the object. Typical SLAM approaches assume that the object was still there such as after step 4 of Simulation Experiment 1. The robot would try and grasp the object even though it is no longer there. With STORM, the robot would know that the object is no longer there and could plan to search for the object.

With the simulation and real-world experiments, we demonstrated that STORM is able to build and maintain object-based maps of dynamic environments and use the maps to manipulate objects and relocalize in the future. In simulation, we showed both qualitatively and quantitatively that STORM is able to do these tasks better than a standard approach relying on the static world assumption. Finally, we verified STORM's performance in the real-world by demonstrating map building, relocalization, and object manipulation.

Chapter 6

Conclusions

This thesis started by presenting the SLAM problem, current accomplishments in the field, and some of the limitations of current techniques. Chapter 2 expanded on these limitations motivating our work — the static world assumption and the lack of semantic object-based maps — and reviewed research being done on these challenges. Chapter 3 provided an overview of the SLAM problem, formally defining SLAM as a probabilistic estimate. The section then reviewed the modern approach to performing SLAM, providing a foundation upon which we present our approach.

6.1 Novel Contributions

Chapter 4 presented Simultaneous Tracking, Object Registration, and Mapping (STORM), an object-based SLAM system that enables a robot to operate in dynamic environments. STORM localizes the robot accurately even when objects are moving independently or being manipulated by the robot and enables the robot to manipulate objects with the object pose estimates in the map. The key contribution of our work is the representation of an environment as a collection of dynamic (rather than static) objects. We also presented SegICP [56, 57], a real-time object pose estimator that we use in STORM's front-end. SegICP performs deep pixel-level semantic segmentation and model-based point cloud registration to robustly estimate 6DoF object poses in real-time.

Chapter 5 demonstrated that STORM constructs accurate object maps of dynamic environments and can use the map to localize and manipulate objects. We evaluated STORM in simulations and compared STORM's performance against a baseline approach. We showed that STORM was able to build an accurate map of the dynamic scene and use it to localize and manipulate objects while the baseline approach was not able to. Finally, we demonstrated STORM's capabilities in real-world experiments.

6.2 Discussion and Future Work

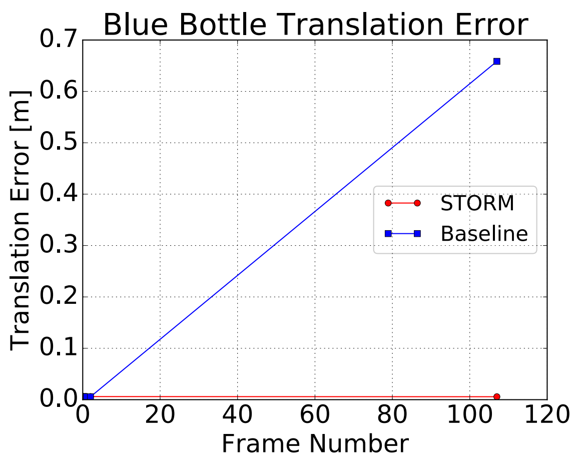
We demonstrated the benefits of representing an environment as a collection of dynamic rather than static objects. There are a number of interesting directions for future work. First, implementing STORM on a moving base would add new challenges in estimating the robot pose, necessitating detecting loop closures. Using an approach like that in [39] for loop closure detection and relocalization would likely improve STORM's performance. On the front-end, adding more object classes and handling multiple instances of a single class using an approach such as [31] would greatly increase the utility of STORM. Finally, adding additional object detectors would further constrain the factor graph and add robustness to failure modes of individual detectors.

Appendix A

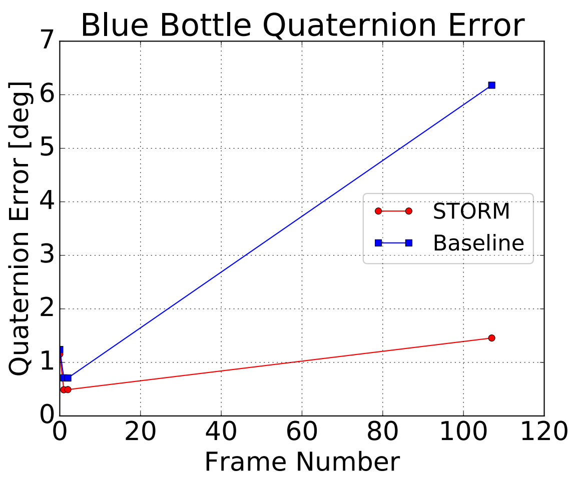
Simulation Experiment Error Plots

In this appendix, we present the remaining object pose estimate error plots from the simulation experiments that we did not present in Chapter 5.

First we show the error plots from Simulation Experiment 1: Mapping:

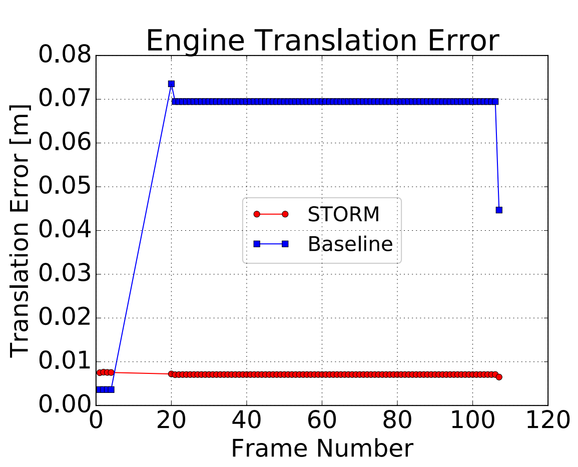


(a)

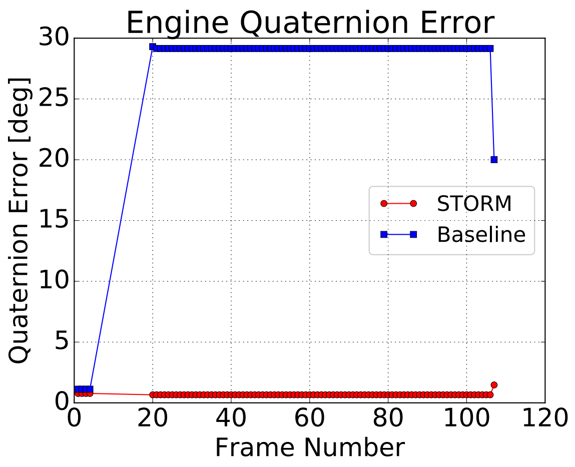


(b)

Figure A-1: **STORM vs. Baseline Blue Bottle Pose Estimate Error in Experiment 1** The plots depict the blue bottle pose estimate translation and quaternion error from STORM and the baseline during Experiment 1. Note how the baseline translation error quickly grows once the blue bottle is moved starting at frame 20.

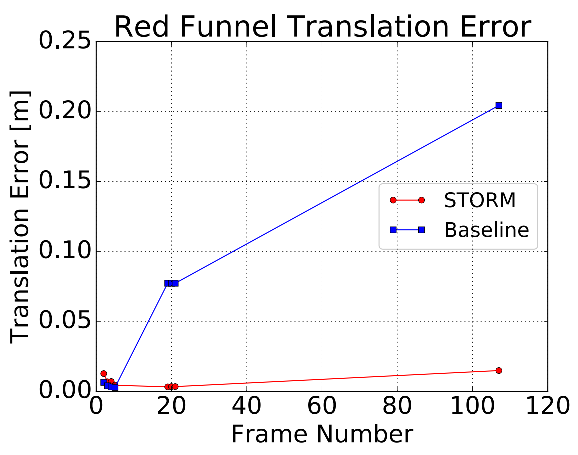


(a)

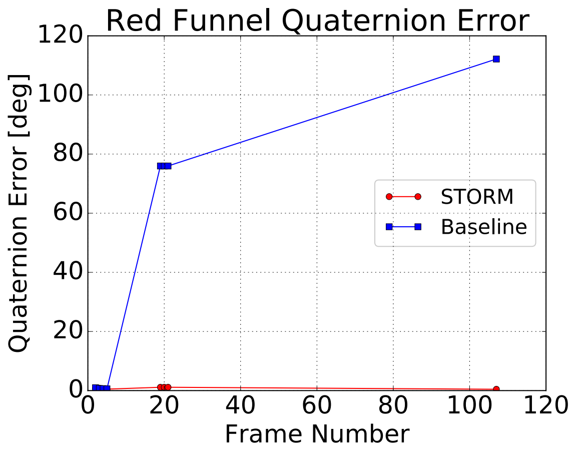


(b)

Figure A-2: **STORM vs. Baseline Engine Pose Estimate Error in Experiment 1** The plots depict the engine pose estimate translation and quaternion error from STORM and the baseline during Experiment 1. Note how the baseline translation error quickly grows once the engine is moved starting at frame 20.



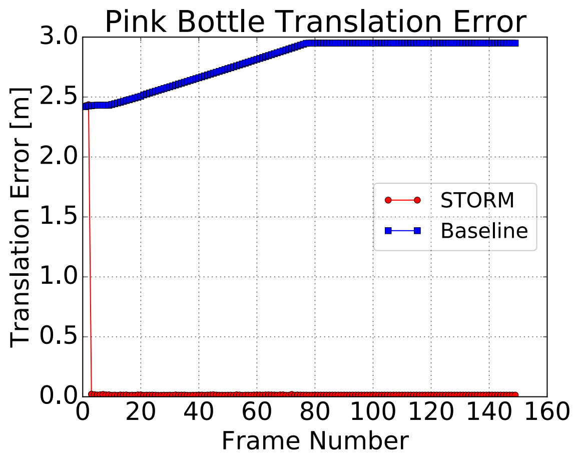
(a)



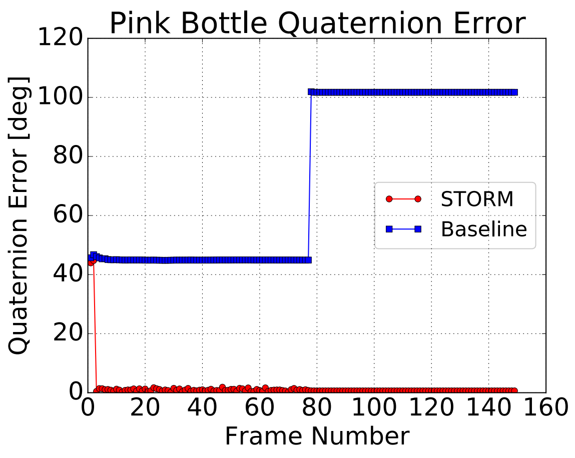
(b)

Figure A-3: **STORM vs. Baseline Red Funnel Pose Estimate Error in Experiment 1** The plots depict the red funnel pose estimate translation and quaternion error from STORM and the baseline during Experiment 1. Note how the baseline translation error quickly grows once the red funnel is moved starting at frame 20.

Next, we show the plots from Simulation Experiment 2: Relocalization:

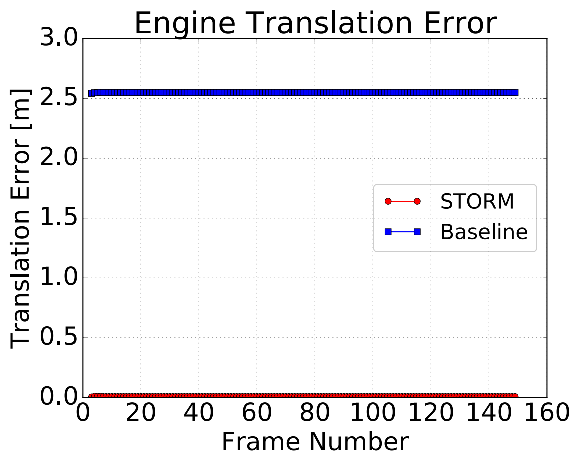


(a)

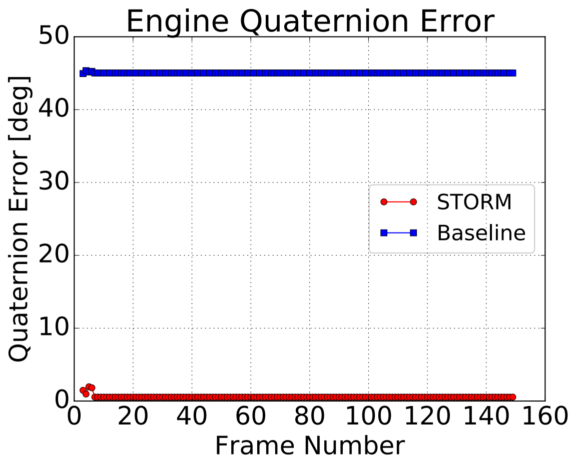


(b)

Figure A-4: **STORM vs. Baseline Pink Bottle Pose Estimate Error in Experiment 2** The plots depict the pink bottle pose estimate translation and quaternion error from STORM and the baseline during Experiment 2. Note how both translation estimates are significantly off initially but after relocalization on frame 3, the STORM estimate error drops to levels near those in Experiment 1. The baseline approach is unable to relocalize and suffers from the same issues once the objects begin to move or are manipulated so the translation error stays high.

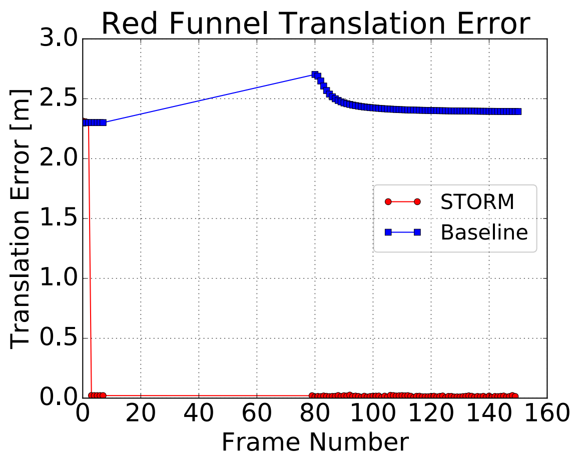


(a)

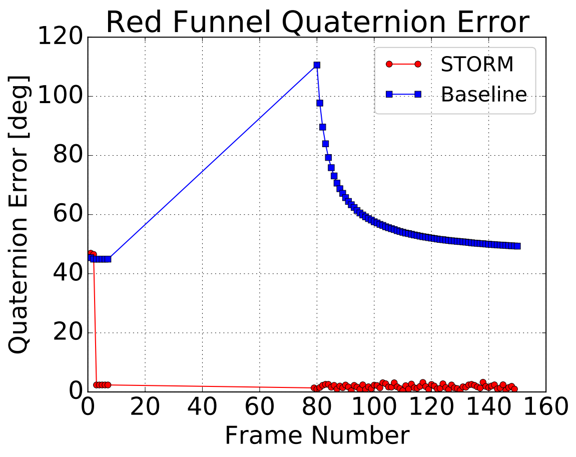


(b)

Figure A-5: **STORM vs. Baseline Engine Pose Estimate Error in Experiment 2** The plots depict the engine pose estimate translation and quaternion error from STORM and the baseline during Experiment 2. Note how both translation estimates are significantly off initially but after relocalization on frame 3, the STORM estimate error drops to levels near those in Experiment 1. The baseline approach is unable to relocalize and suffers from the same issues once the objects begin to move or are manipulated so the translation error stays high.



(a)



(b)

Figure A-6: **STORM vs. Baseline Red Funnel Pose Estimate Error in Experiment 2** The plots depict the red funnel pose estimate translation and quaternion error from STORM and the baseline during Experiment 2. Note how both translation estimates are significantly off initially but after relocalization on frame 3, the STORM estimate error drops to levels near those in Experiment 1. The baseline approach is unable to relocalize and suffers from the same issues once the objects begin to move or are manipulated so the translation error stays high.

Bibliography

[1] Josep Aulinas, Yvan R Petillot, Joaquim Salvi, and Xavier Lladó. The slam problem: a survey. *CCIA*, 184(1):363–371, 2008.

[2] Benzun Pious Wisely Babu, Christopher Bove, and Michael A Gennert. Tight coupling between manipulation and perception using slam.

[3] Vijay Badrinarayanan, Ankur Handa, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. *arXiv preprint arXiv:1505.07293*, 2015.

[4] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE Robotics & Automation Magazine*, 13(3):108–117, 2006.

[5] Wolfram Burgard, Martial Hebert, and Maren Bennewitz. World modeling. In *Springer handbook of robotics*, pages 1135–1152. Springer, 2016.

[6] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[7] S. Choudhary, L. Carlone, C. Nieto, J. Rogers, Z. Liu, H. I. Christensen, and Frank Dellaert. Multi robot object-based SLAM. In *Intl. Symp. on Experimental Robotics*, Tokyo, JP, Oct 2016. IFRR.

[8] Siddharth Choudhary, Alexander JB Trevor, Henrik I Christensen, and Frank Dellaert. Slam with object discovery, modeling and mapping. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1018–1025. IEEE, 2014.

[9] Javier Civera, Dorian Gálvez-López, Luis Riazuelo, Juan D Tardós, and JMM Montiel. Towards semantic slam using a monocular camera. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1277–1284. IEEE, 2011.

[10] Cruise. https://getcruise.com/, Jan 2018.

[11] Frank Dellaert. Factor graphs and gtsam: A hands-on introduction. 2012.

[12] Frank Dellaert and Michael Kaess. Square root sam: Simultaneous localization and mapping via square root information smoothing. *The International Journal of Robotics Research*, 25(12):1181–1203, 2006.

[13] Frank Dellaert, Michael Kaess, et al. Factor graphs for robot perception. *Foundations and Trends® in Robotics*, 6(1-2):1–139, 2017.

[14] DJI. https://www.dji.com/, Jan 2018.

[15] Renaud Dubé, Daniel Dugas, Elena Stumm, Juan Nieto, Roland Siegwart, and Cesar Cadena. Segmatch: Segment based loop-closure for 3d point clouds. *arXiv preprint arXiv:1609.07720*, 2016.

[16] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[17] Dyson. Dyson 360eye. http://www.dyson.com/vacuum-cleaners/robot/dyson-360-eye, Jan 2018.

[18] Udo Frese. Interview: Is slam solved? *KI-Künstliche Intelligenz*, 24(3):255–257, 2010.

[19] Dorian Gálvez-López, Marta Salas, Juan D Tardós, and JMM Montiel. Real-time monocular object slam. *Robotics and Autonomous Systems*, 75:435–449, 2016.

[20] Matteo Golfarelli, Dario Maio, and Stefano Rizzi. Elastic correction of dead-reckoning errors in map building. In *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, volume 2, pages 905–911. IEEE, 1998.

[21] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[22] Dirk Hahnel, Rudolph Triebel, Wolfram Burgard, and Sebastian Thrun. Map building with mobile robots in dynamic environments. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 2, pages 1557–1563. IEEE, 2003.

[23] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1271–1278. IEEE, 2016.

[24] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at `http://octomap.github.com`.

[25] Kaarta. http://www.kaarta.com/, Jan 2018.

[26] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, 31(2):216–235, 2012.

[27] Ioannis Kostavelis and Antonios Gasteratos. Semantic mapping for mobile robotics tasks: A survey. *Robotics and Autonomous Systems*, 66:86–103, 2015.

[28] Kuka. https://www.kuka.com/en-us/products/mobility/navigation-solution, Jan 2018.

[29] Lu Ma, Mahsa Ghafarianzadeh, David Coleman, Nikolaus Correll, and Gabe Sibley. Simultaneous localization, mapping, and manipulation for unsupervised object discovery. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1344–1351. IEEE, 2015.

[30] Lu Ma and Gabe Sibley. Unsupervised dense object discovery, detection, tracking and reconstruction. In *European Conference on Computer Vision*, pages 80–95. Springer, 2014.

[31] Beipeng Mu, Shih-Yuan Liu, Liam Paull, John Leonard, and Jonathan P How. Slam with objects using a nonparametric pose graph. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4602–4609. IEEE, 2016.

[32] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[33] Richard A Newcombe, Dieter Fox, and Steven M Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 343–352, 2015.

[34] Sudeep Pillai and John Leonard. Monocular slam supported object recognition. *arXiv preprint arXiv:1506.01732*, 2015.

[35] Optimus Ride. https://www.optimusride.com/, Jan 2018.

[36] Amazon Robotics. http://amazonrobotics.com/, Jan 2018.

[37] Fetch Robotics. http://fetchrobotics.com/, Jan 2018.

[38] Renato F Salas-Moreno. *Dense semantic SLAM*. PhD thesis, Imperial College London, 2014.

[39] Renato F Salas-Moreno, Richard A Newcombe, Hauke Strasdat, Paul HJ Kelly, and Andrew J Davison. Slam++: Simultaneous localisation and mapping at the level of objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1352–1359, 2013.

[40] Skydio. https://www.skydio.com/, Jan 2018.

[41] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating uncertain spatial relationships in robotics. In *Autonomous Robot Vehicles*, pages 167–193. Springer, 1990.

[42] Cyrill Stachniss. Robot mapping - ws 2012/13. http://ais.informatik.uni-freiburg.de/teaching/ws12/mapping/, Jan 2018.

[43] Cyrill Stachniss, John J Leonard, and Sebastian Thrun. Simultaneous localization and mapping. In *Springer Handbook of Robotics*, pages 1153–1176. Springer, 2016.

[44] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 573–580. IEEE, 2012.

[45] Niko Sünderhauf. *Robust optimization for simultaneous localization and mapping*. PhD thesis, Technischen Universitat Chemnitz, 2012.

[46] Niko Sünderhauf, Trung T Pham, Yasir Latif, Michael Milford, and Ian Reid. Meaningful maps – object-oriented semantic mapping. *arXiv preprint arXiv:1609.07849*, 2016.

[47] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[48] Sebastian Thrun et al. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, 1:1–35, 2002.

[49] Aisha Walcott. *Long-term robot mapping in dynamic environments*. PhD thesis, Massachusetts Institute of Technology, 2011.

[50] Aisha Walcott-Bryant, Michael Kaess, Hordur Johannsson, and John J Leonard. Dynamic pose graph slam: Long-term mapping in low dynamic environments. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1871–1878. IEEE, 2012.

[51] Chieh-Chih Wang. *SIMULTANEOUS LOCALIZATION, MAPPING AND MOVING OBJECT TRACKING*. PhD thesis, University of Sydney, 2004.

[52] Chieh-Chih Wang, Charles Thorpe, Sebastian Thrun, Martial Hebert, and Hugh Durrant-Whyte. Simultaneous localization, mapping and moving object tracking. *The International Journal of Robotics Research*, 26(9):889–916, 2007.

[53] Waymo. https://waymo.com/, Jan 2018.

[54] Thomas Whelan, Stefan Leutenegger, Renato Salas Moreno, Ben Glocker, and Andrew Davison. Elasticfusion: Dense slam without a pose graph. In *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015.

[55] Denis F Wolf and Gaurav S Sukhatme. Mobile robot simultaneous localization and mapping in dynamic environments. *Autonomous Robots*, 19(1):53–65, 2005.

[56] Jay M Wong, Vincent Kee, Tiffany Le, Syler Wagner, Gian-Luca Mariottini, Abraham Schneider, Lei Hamilton, Rahul Chipalkatty, Mitchell Hebert, David Johnson, et al. Segicp: Integrated deep semantic segmentation and pose estimation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[57] Jay M Wong, Syler Wagner, Connor Lawson, Vincent Kee, Mitchell Hebert, Justin Rooney, Gian-Luca Mariottini, Rebecca Russell, Abraham Schneider, Rahul Chipalkatty, et al. Segicp-dsr: Dense semantic scene reconstruction and registration. *arXiv preprint arXiv:1711.02216*, 2017.

[58] Lingzhu Xiang, Zhile Ren, Mengrui Ni, and Odest Chadwicke Jenkins. Robust graph slam in dynamic environments with moving landmarks. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 2543–2549. IEEE, 2015.