AUTOMATICALLY TRACKING AND CALIBRATING ARTICULATED ROBOTS USING SLAM TECHNIQUES

Matthew Klingensmith

CMU-RI-TR-16-36

July 15, 2016

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:

Siddhartha S. Srinivasa
Michael Kaess
George Kantor
Andrew Davison

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Copyright © Matthew Klingensmith

Work in this dissertation was supported in part by Toyota USA grant No. 1011344 the U.S Office of Naval Research grant No. N000141210613 and NSF grant No. IIS-1426703

Keywords: SLAM, Calibration, Tracking, Kinematics, Manipulation, Visual Servoing, Computer Vision, 3D Mapping

For my parents, who showed me the way;
Sarah, who lit up my life;
and David, who still haunts the shadows.

Abstract

Robots still struggle with everyday manipulation tasks. An overriding problem with robotic manipulation is uncertainty in the robot's state and calibration parameters. Small amounts of uncertainty can lead to complete task failure. This thesis explores ways of tracking and calibrating noisy robot arms using computer vision, with an aim toward making them more robust. We consider three systems with increasing complexity: a noisy robot arm tracked by an external depth camera (chapter 2), a noisy arm that localizes itself using a hand-mounted depth sensor looking at an unstructured word (chapter 3), and a noisy arm that only has a single hand-mounted monocular RGB camera estimating its state while simultaneously calibrating its camera extrinsics (chapter 4). Using techniques taken from dense object tracking, fully dense SLAM and sparse general SLAM, we are able to automatically track the robot and extract its calibration parameters. We also provide analysis linking these problems together, while exploring the fundamental limitations of SLAM-based approaches for calibrating robot arms.

Acknowledgments

Writing a thesis is a lot like producing a film, so I have decided to write these acknowledgments as film credits:

Advisor ..... Siddhartha S. Srinivasa
Co-Advisor ..... Michael Kaess
Committee Member 1 ..... George Kantor
Committee Member 2 ..... Andrew Davison
Foreign Advisor 1 ..... Andrew Davison
Foreign Advisor 2 ..... Stefan Leutenegger
Emotional Support ..... Sarah Hoskins
Misc. Services ..... Sarah Hoskins
Theory Help 1 ..... Michael Koval
Theory Help 2 ..... Christopher Dellin
External Support ..... Andrew J. Bagnell
External Support ..... Nancy Pollard
External Support ..... Google Tango Team
Producer 1 ..... Leslie Ansley
Producer 2 ..... Kerry Klingensmith
Extra ..... Rebecca Klingensmith
Three Amigos ..... Yueran Yuan, Wilson Pei, Steven Blessing

This thesis work was completed in three locations: at the Carnegie Mellon Personal Robotics Lab in Pittsburgh PA, the Dyson Robotics Lab at Imperial College in London UK. and at the Rowan University Library in Glassboro NJ.

It would not have been possible without the help of my many mentors and colleagues at Carnegie Mellon, especially my advisers Sidd Srinivasa and Michael Kaess. The research spanned three projects at Carnegie Mellon: ARM-S, HERB and ADA.

My work on Project Tango under Joel Hesch with help from Ivan Dryanovski and Simon Lynen introduced me to some of the core concepts in this thesis including 3D mapping and SLAM.During my 5-month internship in London, my conversations with Andrew Davison and Stefan Leutenegger greatly improved my understanding of visual SLAM. Prof. Davison was kind enough to let me work on my thesis document while developing the SLAM/calibration algorithm in his lab.

The emotional support of my parents, wife, and friends were also critical to completing this thesis. Some of the seeds of my ideas came from Sunday BS sessions at the Chinese buffet on Forbes avenue.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

ROBOTIC manipulation requires robots to sense the world around them, and take action to manipulate the world. The robot must be able to use its sensors (cameras, lasers, depth sensors, etc.) and actuators (joints in its arms and hands) in concert to accomplish this task. This requires the robot to have a clear predictive model of how its actuators will affect the world, and how the world will affect its sensor measurments. If the robot's predictive model is even slightly incorrect, and the robot has no way of identifying or correcting error in its model, then the robot may fail to accomplish its task.

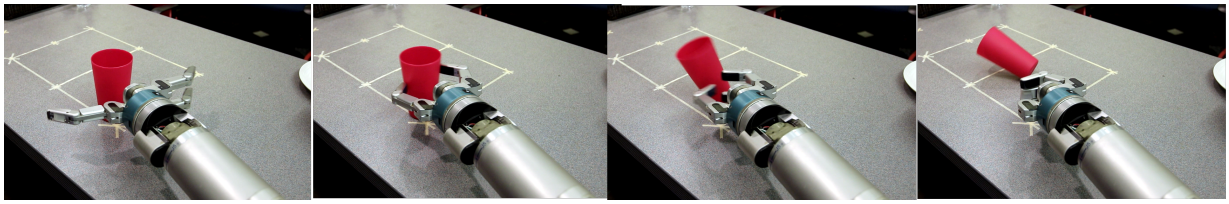


Figure 1.1: A robot attempts to grasp a cup. Somehow, the robot's hand ends up being only about a centimeter away from where it intended to be, and the cup literally slips out of the robot's grasp.

Consider, for example, the simple task of grasping a cup (Fig. 1.1). The robot must use its sensors to identify and locate the cup. It must use a predictive model of its own body and the physics of the world to determine how to grasp the cup successfully, and then it must use its actuators to carry out the action that it had predicted. A slight error in either the predicted cup location, the desired grasp location, or the robot's actuators causes the grasp to completely fail.

As humans, it seems baffling to us when a robot cannot do something as simple as grasping a cup. Robots are sometimes startingly incompetent not only because they often fail, but because they *don't even know when they have failed*.

Where do these failures come from? What errors in the robot's predictive model lead to failure? And, if the robot knows about these errors can it correct them and successfully complete its task?

1.1 Sources of Error

Since a robot manipulation system requires a tight coupling between sensors and actuators, errors in both sensing and actuation compound to produce

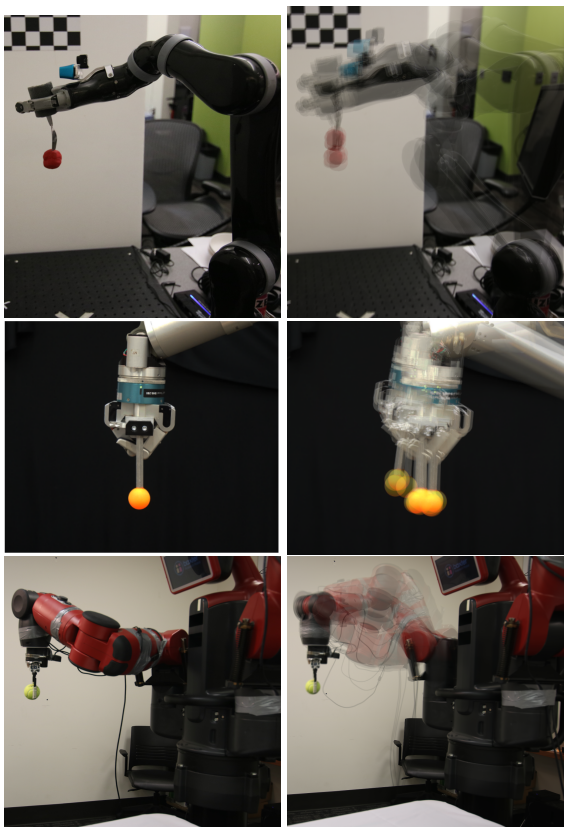


Figure 1.2: Each robot is sent to multiple inverse kinematics solutions (right) for the same end effector pose (left). If there were no error in modelling, calibration, actuation, or joint angle sensing, all of the solutions should result in exactly the same end effector pose. The Kinova Mico [28] arm (top) only puts the end effector within 5 cm of the desired pose. The Barret WAM [37] (center) is within 8 cm, and the Rethink Baxter [23] (bottom) is within 1cm.

Visual sensor error: Robot manipulation tasks require the robot to first perceive and localize objects in the scene using its sensors. All other things being equal, error in the robot's sensing will contribute to the overall error of the manipulation system. Camera systems may be noisy, and may have artifacts such as rolling shutter and blur. Depth cameras have systematic error in their measurments that gets worse over increasing depths, and suffer from missing patches of data.

Joint encoder error: another potential source of error arises from the robot's joint angle sensing being incorrect (see page 65). First, because the joint encoder might be improperly calibrated, and second because unmodeled dynamics arising from mechanisms between the motor and joint encoder (such as cables, gears, pulleys, springs, etc.) can affect the angle of the joint without the encoder being able to sense it. In particular, gear assemblies may suffer from *backlash* from the small gaps between gears, which causes the joint angle sensing to be only accurate to within a discrete resolution.

Actuation error: a further source of error arises from the robot's controllers (that is, the actual commands sent to the robot). Perfectly accurate control of robot arms and trajectory

following is not possible in principle because of unknown dynamics from external forces and the robot's internal mechanisms. Improperly tuned controllers may also cause the robot to fail to accurately follow commands.

Modelling error: finally, the robot's self-model may be incorrect. The positions and orientations of the robot's sensors with respect to its arms (called *extrinsics*) may be wrong, as well as the transformations between each of its joints. The robot may also be slightly nonrigid (that is, its links may bend or twist in response to torques). In short, the robot's assumptions about its own body may be flawed. This may lead to objects being perceived in incorrect locations, and will cause the robot's inverse and forward kinematics to be wrong.

Figure 1.2 shows the net result of these sources of uncertainty. If the robot's internal model, joint sensing, and controllers were all perfectly accurate, then it should be able to place its end effector in exactly the same pose using multiple inverse kinematics solutions – yet some robots have so much error from these sources that they are unable to place their end effectors to within a few centimeters of their target poses. This kind of error can easily result in the failure shown in figure 1.1.

1.2 Measuring and Accounting for Error

Given that the robot has uncertainty about its sensors, actuators, and internal model, how can we detect these errors and correct for them? In general, the errors can be corrected for offline in batch (*i.e. calibration*), or online during execution of the task (*i.e. tracking* and *servoing*).

In both cases, measuring the robot's error amounts to collecting data from the robot's sensors and determining if that data is consistent with the robot's state and assumptions about its own model. Correcting the error involves minimizing inconsistencies between the robot's sensors and its model. Offline calibration consists of collecting a database of sensor measurements and then optimizing the robot's state and model given this data, while online tracking and servoing involves updating the robot's state estimate and model while new sensor data is being collected.

Calibrating the robot offline has the advantage of runtime simplicity (that is, once calibrated the robot can carry out tasks without questioning its internal model or state explicitly), but has the major disadvantage of not being able to account for *new* errors that were not foreseen during calibration. During calibration, the researcher can control the robot's environment and use several "tricks" (such as visual fiducials) to improve the calibration process. Online tracking and servoing on the other hand has the advantage that the robot is constantly observing its own state and correcting for new errors that arise, at the expense of significant runtime complexity. The online system must be able to run fast enough to track the robot in real time.

This thesis will focus mainly on methods of tracking the robot online, rather than calibrating it offline. This means we will not be able to rely as much on the "tricks" often used in robot calibration (like reference objects, fiducials, or days-long optimization and training sessions), and will have to focus on runtime performance.

1.3 Robot Arm Tracking and Calibration as SLAM

One of the core ideas of this thesis is that robot calibration and tracking shares many similarities with Simultaneous Localization and Mapping (SLAM) (see page 55) problems, and that most robot calibration and tracking problems can really be reduced to SLAM problems, even if they may not look that way on the surface. This allows us to apply well-studied SLAM techniques to this domain by expressing the state space of the SLAM system as the joint angles of the robot arm, as well as accounting for additional calibration parameters.

Tracking and calibrating robot arms is challenging primarily due to their high dimensionality. To be able to manipulate the world effectively, robot arms typically have six or more degrees of freedom, each of which may have uncertainty or modelling error. Robot arms have self-occluding complex shapes that change with actuation of the joints. Hand-mounted sensors may not be able to see any of the robot arm itself, limiting their ability to infer the joint angles of the robot directly. We will address each of these challenges in the following chapters.

Chapter 2 details a system for correcting joint encoder error online using an external depth sensor. Chapter 3 extends the algorithm used in chapter 2 to a robot with a hand-mounted depth sensor by expressing the problem as one of dense geometric SLAM. Finally, chapter 4 fits the other two works into a more general graph-based SLAM framework that allows us to correct many types of error simultaneously online using a hand-mounted monocular camera.

CHAPTER 2

Dense Tracking using an External Depth Sensor

We first consider the case of using an external depth camera to track a noisy robot arm in real time. In this case, a depth sensor is mounted so that it can see all or part of the robot's arm, and the robot's joint angles are estimated from the depth image. This is the simplest of the three problems we will consider, because we generally have a good 3D model of the robot arm, and this model can be used as the basis of a tracking algorithm.

Tracking the arm in real-time also allows for a kind of 3D visual servoing, where the robot can be made to touch points in the scene while automatically correcting for its joint angle error. This rather powerful technique allows the robot to correct unforeseen errors that arise from its own dynamics, contact with the scene, and more. In addition, solving this problem elucidates the structure of the related problems we will have to solve in later sections. [1]

2.1 Problem Definition

We have a series of discrete time-steps $t_1 \dots T$ in which we receive time-synchronized measurements from the encoders $\theta_1 \dots \theta_T$ and depth images $D_1 \dots D_T$. How do we determine the true state of the robot from these data? One approach might be to track a rigid object on the robot's body (such as its end effector); but this ignores valuable data from other parts of the robot's body. Another approach is to track the entire robot arm using the depth image.In this section, we will derive a simple way of tracking the entire robot in *configuration space* (see page 63) from a series of depth images. This has several advantages: it makes full use of observations of the robot's body, it naturally constrains solutions to those which are possible given the robot's kinematics, and it provides an estimate of the joint angles which can be used for visual servoing.

The goal is to predict a maximum likelihood estimate of the robot's true configuration $\mathbf{q}_1 \dots \mathbf{q}_T$ from these measurements. In other words, we wish to find:

$$\mathbf{q}_T = \underset{\mathbf{q}}{\operatorname{argmax}} \mathbf{P}\left(\mathbf{q} | \mathbf{D}_1, \dots, \mathbf{D}_T, \theta_1, \dots \theta_T\right) \tag{2.1}$$

If we model this as a Markov process, such that the state at time T only depends on the state at time $T-1$ and measurements from time T, and assuming conditional independence between

[1] *This chapter largely reiterates earlier work that appears in [30]*

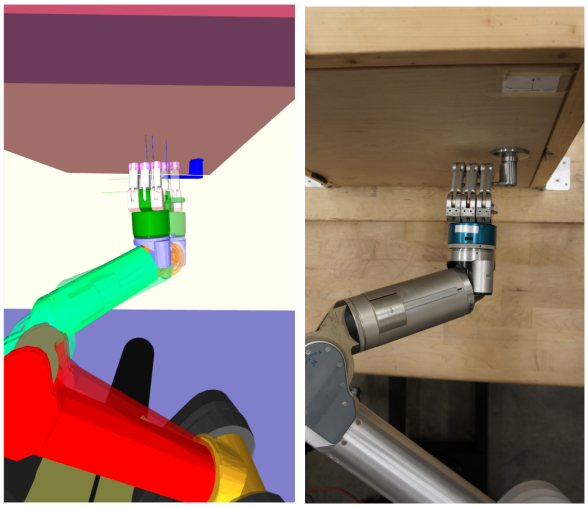


Figure 2.1: A robot arm opens a door using dense depth tracking. Left: the robot's kinematics (solid) are compared with the estimated joint angles from the tracking system (transparent). Right: the robot is shown grasping the door handle.

the depth image, configuration and joint encoders we have a maximization of the log-likelihood of two terms:

$$\mathbf{q}_T = \underset{\mathbf{q}}{\operatorname{argmax}} \, \mathrm{P}\left(\mathbf{q} | \mathbf{D}_T, \theta_T\right) \tag{2.2}$$

$$= \underset{\mathbf{q}}{\operatorname{argmax}} \, \frac{\mathrm{P}\left(\mathbf{D}_T | \mathbf{q}\right) \mathrm{P}(\mathbf{q} | \theta_T)}{\mathrm{P}\left(\mathbf{D}_T, \theta_T\right)} \tag{2.3}$$

$$= \underset{\mathbf{q}}{\operatorname{argmax}} \, \log \mathrm{P}(\mathbf{D}_T | \mathbf{q}) + \log \mathrm{P}(\mathbf{q} | \theta_T) \tag{2.4}$$

these are the depth image posterior $\mathrm{P}(\mathbf{D}_T|\mathbf{q})$ and the joint encoder posterior $\mathrm{P}(\mathbf{q}|\theta_T)$. With some simple assumptions about the structure of the depth image and the joint encoders, it is possible to write these out efficiently.

2.1.1 Depth Image Posterior

From each depth image $\mathbf{D}_T$, produce its point-cloud $\mathbf{p}_1, \ldots, \mathbf{p}_M \in \mathbf{D}_T$. We assume that given the robot's configuration $\mathbf{q}$, the points in the point cloud correspond to some true point on the robot's body corrupted by identically distributed independent random noise. Assume there is a ground-truth mapping $\beta_i(\mathbf{q})$ that gives us the point on the robot's body that corresponds to the point in the point cloud of the depth image.

Assume then that the points are distributed according to a simple isotropic normal distribution so that

$$\mathbf{p}_i \sim \beta_i(\mathbf{p}_i, \mathbf{q}) + \mathcal{N}(0, \Sigma_{\mathbf{p}}) \tag{2.5}$$

where $\Sigma_{\mathbf{p}} = \sigma_{\mathbf{p}}\mathbf{I}$ is the covariance of the point cloud noise, with $\sigma_{\mathbf{p}}$ being the noise variance. With all these assumptions in place, the probability of seeing a particular point in the point cloud given the robot's configuration is simply

$$\mathrm{P}(\mathbf{p}_i|\mathbf{q}) = \exp \frac{-\|\mathbf{p}_i - \beta_i(\mathbf{q})\|^2}{2\sigma_{\mathbf{p}}} \tag{2.6}$$

and assuming conditional independence, the depth image posterior becomes

$$\mathrm{P}(\mathbf{D}_T|\mathbf{q}) = \prod_{\mathbf{p}_i \in \mathbf{D}_T} \mathrm{P}(\mathbf{p}_i|\mathbf{q}) \tag{2.7}$$

and finally the log-likelihood becomes

$$\log \mathrm{P}(\mathbf{D}_T|\mathbf{q}) = -\frac{1}{2\sigma_{\mathbf{p}}} \sum_{\mathbf{p}_i \in \mathbf{D}_T} \|\mathbf{p} - \beta_i(\mathbf{q})\|^2 \tag{2.8}$$

Computing the posterior requires associating each point in the point cloud with a point on the robot's body. If these associations are unknown, we can use physical distance as a heuristic to match a point in the point cloud to one on the robot's body. That is:

$$\bar{\beta}_i(\mathbf{q}) = \operatorname*{argmin}_{\beta \in \mathbf{B}(\mathbf{q})} \|\beta - \mathbf{p}_i\| \tag{2.9}$$

where $\mathbf{B}(\mathbf{q})$ is the set of all points on the robot's body while the robot is in configuration $\mathbf{q}$ that can be viewed by the depth camera (and which are not self-occluded). This heuristic is similar to the one used in the Iterative Closest Point (ICP) algorithm. Because we use every pixel in the depth image, this is a *dense* tracking technique (as opposed to a technique which only uses sparse image features).

2.1.2 Joint Encoder Posterior

The relationship between the joint encoders θ and actual configuration $\mathbf{q}$ of the robot might be quite complicated; but here we use a very simple model of the joint encoder noise that is sufficient to capture it. Assume we have a fixed calibration $K_\theta(\theta)$ which converts joint encoders to robot configurations (see page 65), and assume that the true robot configuration is drawn from an isotropic Gaussian distribution around this function. That is:

$$\mathbf{q}_T \sim K_\theta(\theta_T) + \mathcal{N}(0, \Sigma_{\mathbf{q}}) \tag{2.10}$$

where $\Sigma_{\mathbf{q}} = \sigma_{\mathbf{q}}\mathbf{I}$, and $\sigma_{\mathbf{q}}$ is the isotropic variance of the noise. With these assumptions, we can write the joint encoder posterior as

$$\mathrm{P}(\mathbf{q}_T|\theta_T) = \exp -\frac{\|K_\theta(\theta_T) - \mathbf{q}\|^2}{2\sigma_{\mathbf{q}}} \tag{2.11}$$

and the log-likelihood is simply

$$\log P(\mathbf{q}_T|\theta_T) = -\frac{\|K_\theta(\theta_T) - \mathbf{q}\|^2}{2\sigma_\mathbf{q}} \tag{2.12}$$

admittedly, this model of joint encoder noise is not completely accurate, as it assumes there is time-independent random noise on the joint-encoders, when in reality there are likely to be systematic biases that depend on the configuration of the robot, time, or both. We will explore more complex posteriors in later sections.

2.2 Algorithm Implementation

We implement the algorithm using stochastic gradient ascent of the log-likelihood function. The algorithm can be summarized as follows:

1. Synchronize depth images and joint encoders at time T to produce $\mathbf{D}_T$ and θ_T.
2. Initialize the gradient descent using the mode of the joint encoder posterior $\mathbf{q}_k = K_\theta(\theta_T)$.
3. Render the robot's body in the depth image frame at time T to produce a set of body points $\mathbf{B}(\mathbf{q}_k)$
4. Match each body point to its nearest neighbor in the point cloud to calculate an estimate of $\mathrm{P}(\mathbf{D}|\mathbf{q}_k)$
5. Minimize the negative log-likelihood using stochastic gradient descent.
6. Repeat.

2.2.1 Matching to the depth image

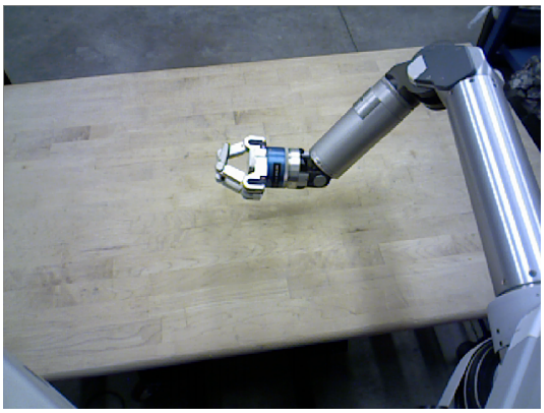
Our implementation relies on quickly rendering and matching points of the robot's body to points in the point cloud. To achieve this, we render a *synthetic point cloud* (Fig. 2.2) using a CAD model of the robot and the depth camera's intrinsics. Points are labeled by the GPU according to the link they are associated with. Each synthetic point is then matched to a real point in the point cloud using an octree. Outliers are rejected. The result is a conservative matching between the synthetic point cloud and the real one which naturally handles occlusion and segmentation.

2.2.2 Stochastic gradient descent

We compute the stochastic gradient of the log-likelihood function with respect to the robot's configuration by randomly sampling M points of the robot's body and computing:

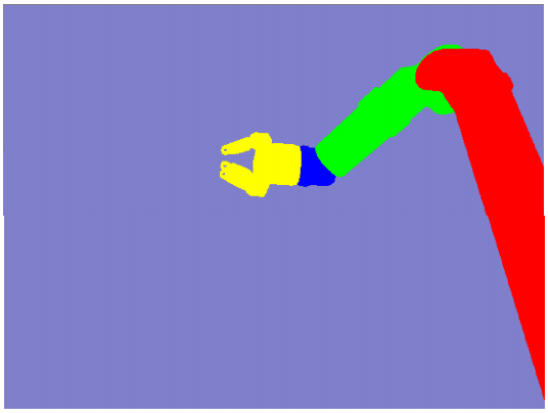
$$\frac{\partial}{\partial \mathbf{q}} \log \mathrm{P}(\mathbf{D}|\mathbf{q}) \approx \sum_{i=1}^{M} \frac{\partial}{\partial \mathbf{q}} \mathrm{P}(\mathbf{p}_i|\mathbf{q}) \tag{2.13}$$

$$= -\frac{1}{\sigma_\mathbf{p}} \sum_{i=1}^{M} \mathbf{J}_{\beta_i}^T \left(\mathbf{p} - \beta_i(\mathbf{q})\right) \tag{2.14}$$



(a) RGB input (not used)

(b) Depth input



(c) Synthetic label image

(d) Synthetic and real point clouds

Figure 2.2: Images taken during different phases of the dense tracking algorithm.

where $\mathbf{J}_{\beta_i}$ is the linear kinematic Jacobian of the robot arm evaluated for body point β_i (see page 64). The gradient of the joint encoder posterior can also be computed as

$$\frac{\partial}{\partial \mathbf{q}} \log \mathrm{P}(\mathbf{q}|\theta) = -\frac{K_\theta(\theta_T) - \mathbf{q}}{\sigma_\mathbf{q}} \tag{2.15}$$

We use the stochastic gradient, rather than the full gradient, to save on computation. The gradient descent step is then:

$$\mathbf{q}^{(k+1)} = \mathbf{q}^{(k)} - \lambda \left(\frac{\partial}{\partial \mathbf{q}} \log \mathrm{P}(\mathbf{q}|\theta) + \frac{\partial}{\partial \mathbf{q}} \log \mathrm{P}(\mathbf{D}|\mathbf{q}) \right) \tag{2.16}$$

where λ is a learning rate. We repeat until $\mathbf{q}^{(k)}$ converges, and becomes the new estimated state of the robot's joint angles.

Algorithm Step	Average time per frame (ms)
Forward Kinematics	0.1
Gradient Calculation	0.2
Data Association	1.5
Synthetic Rendering	8.0

Table 2.1: Timing data for the dense tracking algorithm. Notice that rendering the synthetic point cloud takes up the vast majority of the time. Altogether, the algorithm runs at ~ 100 Hz.

2.3 Experiments

We tested the dense tracking algorithm on the DARPA ARM-S robot, which has an Asus *Xtion Pro* depth sensor and a Barret WAM arm. Timing data for the tracker is shown in Table 2.1 . On average, the tracker runs at more than 100 Hz (the majority of time being spent creating the synthetic point cloud image and shuffling data back and forth between the GPU). This is fast enough to incorporate into a real-time control loop for visual servoing.

To verify that the algorithm was really reducing error in the robot's joint angles, we conducted an experiment where the robot was asked to touch a 2cm radius red dot repeatedly from different directions with randomly sampled inverse kinematics solutions. The red dot was detected using HSV blob segmentation in the RGB-D image, and its centroid in the point cloud was treated as the 3D target that the robot was to servo toward (Fig. 2.3). We measured how far the finger tip fell from the center of the red dot as a measure of accuracy. The experiment was performed 15 times for each case (with and without dense tracking), using the same randomly sampled starting end ending configurations for the arm.

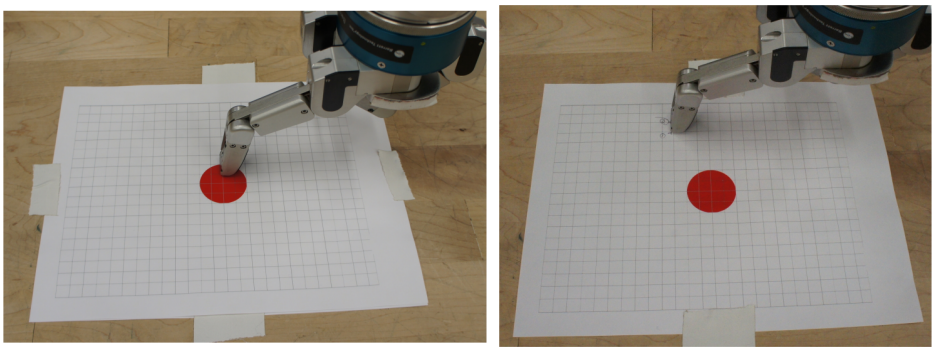
Fig. 2.3c shows the error of the fingertip when using visual servoing and without. Tracking the arm in the depth image resulted in error that was well under 2cm, while using open-loop position control resulted in errors that were in excess of 7cm.

This kind of error reduction allows us to perform more precise manipulation tasks. Figure Fig. 2.1 shows an example of such a task: the robot uses dense tracking to servo toward a door handle and open it. We performed this experiment 10 times. The robot was said to succeed whenever its fingers caged the door handle, and was said to fail whenever its fingers missed the door handle completely, collided with the door, or hit the door handle and slipped off. With visual servoing from dense arm tracking, the robot succeeded 9/10 times. With open-loop position control, the robot succeeded 0/10 times.

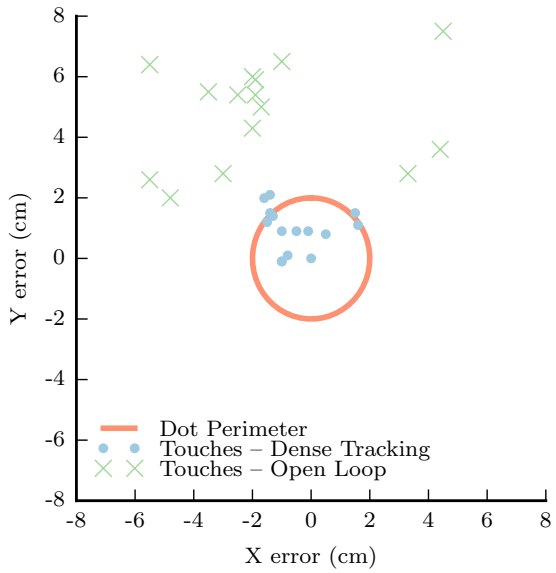
2.4 Discussion and Limitations

We have shown an example of how visual data can be used to correct joint angle uncertainty for a robot arm, and how this correction can be used to improve the robot's performance. The key idea behind this simple algorithm involves treating the problem as one of maximum likelihood estimation in the configuration space of the robot, given the sensor data.

However, this algorithm has severe limitations. First, it assumes all of the error comes from joint angle offsets along the axis of the robot's joints. It treats this uncertainty as simple isometric



(a) (b)



(c)

Figure 2.3: The visual servoing experiment. Left: the robot servos toward the red dot while tracking its joint angles with the depth image. Right: the robot servos toward the red dot without tracking its joint angles with the depth image.

Gaussian noise on the joint encoders, when in reality the noise is likely to be nonlinear and depends on factors such as gravity and torque.

The technique fails to explicitly estimate the uncertainty of the robot arm's state. Rather, it just estimates a single maximum-likelihood estimate. Because it makes the Markov assumption, the dense tracker does not make use of the entire robot trajectory to estimate the state. This makes the technique prone to local minima.

Because we use simple point-to-point data associations between the robot's body and the depth image, we run the risk of mismatching data. Because we rely on simple gradient descent to optimize the log-likelihood function, the algorithm is not very robust to outliers caused by mis-matching of data. This will occur if any un-modeled objects (such as objects held in the robots hand, or surfaces occluding the arm) are visible in the depth image near the arm.

Finally, this technique relies on a depth camera that can see all or part of the robot's arm.

What do we do if the camera is mounted on the robot's hand (and hence can't see any part of the robot's arm)? What if depth is not available? We will tackle some of these problems in the next two chapters.

CHAPTER 3

Dense SLAM using a Hand-Mounted Depth Sensor

Consider the following scenario: a robot with a hand-mounted depth sensor scans an unknown scene filled with clutter. The robot's goal is to create a geometric model of the scene that it can use for motion planning and manipulation. If the robot had perfect kinematics, and no uncertainty in its joint angle estimates, creating a geometric map of the scene would be trivial – it would just involve finding the pose of the sensor at each timestep by looking at the robot's joint encoders, and then stitching together all of the robot's sensor measurements at each pose to create a 3D model. Unfortunately, as we have seen in the previous chapters, many robots do not have perfect joint angle sensing, and some robots have a very large amount of kinematic error. With incorrect joint angle measurements, the model of the scene that the robot creates from its scan will be inaccurate.

As we have seen in chapter 2, an external depth sensor mounted on a robot's body or affixed somewhere in the scene can be used to correct joint angle error by maximizing the likelihood of a robot's configuration given its sensor data.Can the same approach that we discussed in 2 be used when the camera is mounted on the robot's hand, and can't see any part of the robot's body?

Right away, it's clear that we're dealing with a Simultaneous Localization and Mapping (SLAM) problem. The robot's pose at any given time is unknown because of kinematic uncertainty. The model of the world is likewise unknown *a. priori*. We will therefore have to estimate both the structure of the world and the robot's state simultaneously.

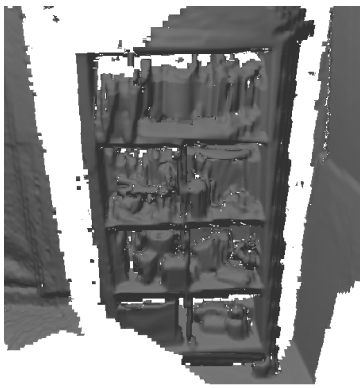
In this chapter, we will explore a dense SLAM technique which simultaneously estimates the robot's joint angles and a dense 3D geometric model of the world [1].

3.1 Problem Definition

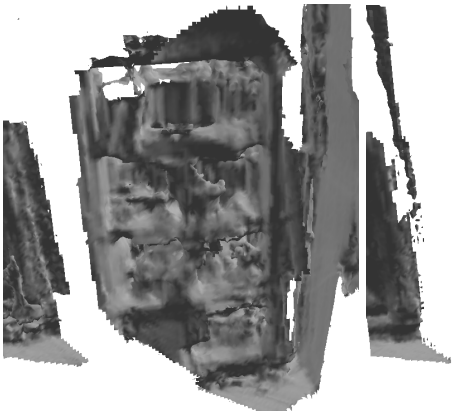
We have a series of discrete time steps $t = 1, \dots T$ with synchronized joint encoder measurements $\theta_1, \dots, \theta_T$ and depth images $\mathbf{D}_1, \dots, \mathbf{D}_T$. The task is to estimate the robot's true configuration $\mathbf{q}_1, \dots \mathbf{q}_T$ and a geometric model of the scene $\mathbf{M}$ simultaneously. Formally, we can express this as computing a maximum likelihood estimate of the robot's trajectory and geometric model

[1] This chapter is largely restating earlier work found in [32]

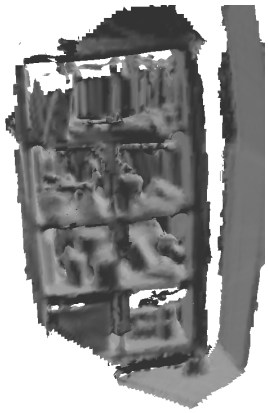
(a) Experimental setup.



(b) Kinect Fusion (baseline).



(c) Forward kinematics.



(d) ARM-SLAM

(e) Colorized ARM-SLAM.

Figure 3.1: Robot scans and reconstructions of a bookshelf at 1.5cm resolution using real depth and encoder data (see page 20). Our approach (which estimates robot configurations rather than camera poses) results in preservation of fine detail and edges that are lost when only using the robot's forward kinematics, with comparable reconstruction quality to Kinect Fusion.

$$\{\mathbf{q}_1, \ldots, \mathbf{q}_T, \mathbf{M}\}_{\text{MLE}} = \underset{\mathbf{q}_1,\ldots,\mathbf{q}_T,\mathbf{M}}{\operatorname{argmax}} \mathrm{P}\left(\mathbf{q}_1, \ldots, \mathbf{q}_T, \mathbf{M} | \mathbf{D}_1, \ldots, \mathbf{D}_T, \theta_1, \ldots, \theta_T\right) \tag{3.1}$$

this expression is extraordinarily high-dimensional, and is difficult to directly optimize it, because the dense model $\mathbf{M}$ correlates every sensor measurement together. Therefore, we solve this problem in the same way as several other dense SLAM techniques (particularly *Kinect Fusion* [24]) by breaking the problem into two parts: first, tracking the sensor with respect to a fixed dense map, and second, updating the dense map with respect to a fixed sensor pose.These two steps, performed rapidly in alteration can be made to simultaneously construct a map and track the pose of the sensor. The trade off is that both the pose and structure of the map can drift over time as small errors accumulate.

3.1.1 Mapping the Scene

Given a set of depth images $\mathbf{D}_1, \ldots, \mathbf{D}_T$ taken at known poses $C_1, \ldots, C_T$, we wish to construct a 3D geometric model of the scene $\mathbf{M}$. This is a well-studied problem in computer vision. We chose the volumetric method of Curless and Levoy [8] known as Truncated Signed Distance Field (TSDF) mapping. This is the same mapping method used by *Kinect Fusion* [24].

Using this method, the 3D model takes the form of a volumetric signed distance function:

$$\mathbf{M} = \Phi(\mathbf{x}) : \mathbb{R}^3 \rightarrow [-\tau, \tau] \tag{3.2}$$

where τ is a small radius around the surfaces of objects. The TSDF maps volumetric points in the scene $\mathbf{x}$ to their distance to the nearest surface. Inside objects, the the distance is negative, and outside the distance is positive. Exactly at the surface, the distance is zero. Therefore, the surface of the scene is implicitly represented as the level set $\Phi(\mathbf{x}) = 0$.

To generate a TSDF from a series of depth images, we take all the points $\mathbf{p}_i \in \mathbf{D}$ in the point cloud of each depth image, compute its surface normal $\mathbf{n}_i$, and use this to construct a linearization of the distance field near $\mathbf{p}_i$. For all of the points $\mathbf{x}$ within a radius τ of $\mathbf{p}_i$, we set its signed distance $\Phi(\mathbf{x})$ to the linearized value derived from the surface normal. When the linearizations of several pixels intersect, we merely average the distances. Curless [8] proved that this method minimizes the squared distance of points in the point cloud to the implicit surface generated by Φ.

In practice, this is accomplished by storing voxels V within a region of interest. Each discrete voxel contains an estimate of the signed distance, and a weight which is used to produce a running average of the TSDF as new measurements accumulate. Algorithm 1 describes how to fuse a new depth image at a known pose into the TSDF.

Using a TSDF to model the scene allows us to average out sensor measurement error as depth images accumulate. The TSDF, since it is continuous and volumetric, also has the advantage of having well-defined gradients $\nabla\Phi$, which are easy to compute using finite differencing.

3.1.2 Tracking the Arm

Assuming a fixed dense map $\mathbf{M}$, the problem of tracking the arm becomes significantly easier, since we can use the Markov assumption such that the current configuration of the robot is conditionally independent of all the previous configurations, joint encoder measurements and depth images:

$$\begin{aligned}
\mathbf{q}_{\text{MLE}} &= \underset{\mathbf{q}}{\operatorname{argmax}}\, \mathrm{P}\left(\mathbf{q} | \mathbf{D}_T, \theta_T, \mathbf{M}\right) && (3.3)\\
&= \underset{\mathbf{q}}{\operatorname{argmax}}\, \frac{\mathrm{P}\left(\mathbf{D}_T | \mathbf{q}, \mathbf{M}\right) \mathrm{P}\left(\mathbf{q} | \theta_T\right)}{\mathrm{P}(\mathbf{D}_T, \theta_T, \mathbf{M})} && (3.4)\\
&= \underset{\mathbf{q}}{\operatorname{argmax}}\, \log \mathrm{P}(\mathbf{D}_T | \mathbf{q}, \mathbf{M}) + \log \mathrm{P}(\mathbf{q} | \theta_T) && (3.5)
\end{aligned}$$

notice that Eq. 3.5 bears a striking resemblance to Eq. 2.4 , only with an additional term for the dense map $\mathbf{M}$. This is because with a known map, the problem of tracking the robot arm using a

// Given a depth image, sensor pose, previous TSDF, previous voxel weights, a weighting function, a volume of interest, and a truncation distance

Input: $\mathbf{D}, C, \Phi^{(t-1)}, W^{(t-1)}, w, V, \tau$

// Inverse of the camera transform

$C' \leftarrow C^{-1}$

// Initialize the new TSDF to the previous one

$\Phi_t \leftarrow \Phi_{t-1}$

// Initialize the weights

$W_t \leftarrow W_{t-1}$

for $\mathbf{v} \in V$ **do**

  // Depth of voxel $\mathbf{v}$

  $\mathbf{v}_c \leftarrow C'\mathbf{v}$

  // Current depth measurement for the voxel.

  $d_i \leftarrow \mathbf{D}\left[\pi(\mathbf{v}_c)\right]$

  // Dist to camera plane.

  $d_v \leftarrow \mathbf{v}_c(z)$

  // Locally linear approximation.

  $u \leftarrow d_v - d_i$

  // If dist within τ

  if $|u| < \tau$ **then**

    // Weighted average

    $\Phi_t(\mathbf{v}) \leftarrow \frac{W_t(\mathbf{v})\Phi_t(\mathbf{v}) + w(u)u}{W_t(\mathbf{v}) + w(u)}$

    $W_t(\mathbf{v}) \leftarrow W_t(\mathbf{v}) + w(u)$

  end

end

Output: Φ_t, W_t

Algorithm 1: FUSETSDF

depth image of its own body is exactly equivalent to tracking the robot arm using a depth image of a rigid body attached to its base. In this context, the dense map of the scene can be seen as *just another link* of the robot that is attached rigidly to its base.

With this framework in mind, it is possible to use the same algorithm as in chapter 2 to track the robot, replacing points on the robot's body with points in the dense map. This is essentially what we do in [32], except our representation of the dense map $\mathbf{M}$ is volumetric, rather than point-based. The joint encoder posterior here is exactly the same as in Eq. 2.4 , so we use the same method to compute it.

Depth Image Posterior

Assume that the points in the depth image are really samples of points on the surface of $\mathbf{M}$ that have been corrupted by isotropic Gaussian noise. That is, for each $\mathbf{p}_i \in \mathbf{D}$, assume that there exists a point $\mathbf{x}_i$ s.t $\Phi(\mathbf{x}_i) = 0$ that corresponds to a point on the surface of $\mathbf{M}$, and

$$\mathbf{p}_i \sim \mathbf{x}_i + \mathcal{N}(0, \Sigma_{\mathbf{p}}) \tag{3.6}$$

where $\Sigma_{\mathbf{p}} = \sigma_{\mathbf{p}}\mathbf{I}$, and $\sigma_{\mathbf{p}}$ is the variance of the depth image noise. If $\mathbf{p}_i$ is already very close to $\mathbf{x}_i$, then the value of the distance field $|\Phi(\mathbf{p}_i)| \approx \|\mathbf{p}_i - \mathbf{x}_i\|$, and therefore

$$\mathrm{P}\left(\mathbf{p}_i|\mathbf{M}\right) \approx \exp \frac{-\|\Phi(\mathbf{p}_i)\|^2}{2\sigma_{\mathbf{p}}} \tag{3.7}$$

Assuming conditional independence among all the pixels in the depth image, we arrive at the expression for the log-likelihood of the depth image posterior from equation 3.5 as

$$\log \mathrm{P}(\mathbf{D}|\mathbf{q}, \mathbf{M}) \approx \log \prod_{\mathbf{p}_i \in \mathbf{D}} \exp \frac{-\|\Phi(\mathbf{p}_i)\|^2}{2\sigma_{\mathbf{p}}} \tag{3.8}$$

$$= -\frac{1}{2\sigma_{\mathbf{p}}} \sum_{\mathbf{p}_i \in \mathbf{D}} \Phi(\mathbf{p}_i)^2 \tag{3.9}$$

note that this is only valid if the pose estimate is already very near the true pose of the sensor. Otherwise, the points from the depth image may be mis-matched to points in the dense map. Depending on the geometry of the scene, the mis-matching could be more or less severe. For instance, if the world consists of a single plane, this method would not be able to distinguish between points on the plane, and all poses that move parallel to the plane will be treated as equally probable (this is a problem that other TSDF methods suffer from as well).

3.2 Algorithm Implementation

As in chapter 2, we use stochastic gradient descent of the log-likelihood function to obtain a pose estimate for the robot arm. We interleave this pose estimation step with a TSDF mapping step following Alg. 1 . The result is a system which generates a consistent 3D map and corrects the robot's joint angles in real time.

3.2.1 TSDF Mapping

We use the TSDF mapping system from our earlier work [31]. This mapping system (called CHISEL) represents the TSDF as a spatial hash map of voxels. Each voxel stores a 16-bit signed distance, a 16-bit weight, and a 32-bit color (which is not used for tracking). The TSDF grows as more surfaces are observed, but unlike other systems, CHISEL stores only voxels that are near surfaces, rather than the entire scene. This allows very large scale maps to be stored efficiently.

3.2.2 Stochastic Gradient Descent

The gradient of the log-likelihood function expressed in Eq. 3.9 with respect to the robot's configuration $\mathbf{q}$ can be computed as a simple application of the chain rule

$$\frac{\partial}{\partial \mathbf{q}} \log \mathrm{P}(\mathbf{D}|\mathbf{q}, \mathbf{M}) = -\frac{1}{2\sigma_{\mathbf{p}}} \sum_{\mathbf{p}_i \in \mathbf{D}} \Phi(\mathbf{p}_i) \mathbf{J}_{\mathbf{p}_i}^T \nabla \Phi(\mathbf{p}_i) \tag{3.10}$$

`// Where` $\mathbf{q}_e^{(t)}$ `are the motor encoders at time` t`,` λ `is a learning rate, and` γ `is a regularization parameter.`
Input: $\mathbf{D}_t, \theta_t, \Phi^{(t-1)}, \lambda, \mathbf{q}^{(t-1)}$
$\mathbf{q}^{(k)} \leftarrow \mathbf{q}^{(t-1)}$
repeat
  `// The camera transform given the robot's joint angles.`
  $T_{\mathbf{q}} \leftarrow F_{\text{cam}}(\mathbf{q}^{(k)})$
  `// Gradient of the depth image posterior.`
  $\nabla C \leftarrow \frac{1}{\sigma_{\mathbf{p}}} \sum_{\mathbf{p} \in \mathbf{D}_t} \left[\Phi^{(t-1)}\left[T_{\mathbf{q}}\mathbf{p}\right] \mathbf{J}_{\mathbf{p}}^T \nabla \Phi^{(t-1)}\left[T_{\mathbf{q}}\mathbf{p}\right]\right]$
  `// Descend the gradient.`
  $\mathbf{q}^{(k)} \leftarrow \mathbf{q}^{(k)} - \lambda \left(\nabla C + \frac{1}{\sigma_{\mathbf{q}}}\left(\mathbf{q} - K_\theta(\theta_t)\right)\right)$
until *convergence*;
$\mathbf{q}_t \leftarrow \mathbf{q}^{(k)}$
`// Mapping step.`
$\Phi^{(t)} \leftarrow \text{FUSETSDF}\left(\Phi^{(t-1)}, \mathbf{D}_t, \mathbf{q}_t\right)$
Output: $\mathbf{q}_t, \Phi^{(t)}$

Algorithm 2: ARM-SLAM

3.3 Experiments

We conducted three types of experiments to observe the behavior of this algorithm; 2D simulation experiments, 3D simulation experiments, and a real robot experiment. [2]

[2] *Videos are available at* `http://youtu.be/QrFyaxFUs9w`

t		*Fwd. Kin.*	*Dense Fusion*	*ARM-SLAM*
500	EE Err. (pix.)	5.2 ± 5.9	3.4 ± 4.2	$\mathbf{0.8 \pm 0.7}$
	Jnt. Err (rad.)	0.08 ± 0.06	$-$	$\mathbf{0.06 \pm 0.05}$
	SDF Err (pix.)	1.4 ± 1.7	0.8 ± 0.8	$\mathbf{0.5 \pm 0.3}$
	Class Err (%)	5.7 ± 3.2	4.7 ± 2.3	$\mathbf{3.5 \pm 0.6}$
999	EE Err. (pix.)	9.2 ± 6.7	14.7 ± 17.8	$\mathbf{1.4 \pm 1.9}$
	Jnt. Err (rad.)	0.17 ± 0.07	$-$	$\mathbf{0.08 \pm 0.05}$
	SDF Err. (pix.)	6.1 ± 5.3	12.2 ± 22.2	$\mathbf{1.2 \pm 0.8}$
	Class Err. (%)	11.3 ± 6.2	9.5 ± 6.1	$\mathbf{4.4 \pm 1.1}$

Table 3.1: Results for the 2D simulation experiments (see page 19). The end effector error in pixels, joint angle error in radians, distance field error in 10^6 pixels, and occupancy classification error (the proportion of pixels mis-classified as containing an obstacle) is shown for forward kinematics, unconstrained dense fusion, and ARM-SLAM for a data-set with 500 and 999 time-steps. Our approach (ARM-SLAM) reduces all three error terms.

3.3.1 2D Simulation

In the simple 2D simulation experiment, a 3-link serial robot manipulator with a simulated 1D depth sensor scans a scene. We added zero-centered Perlin [48] noise to its joint encoder readings. That is,

$$\theta_t = \mathbf{q}_t + \beta_n \textsc{Perlin}\big(s_n \mathbf{q}_t\big) \tag{3.11}$$

where s_n, β_n are parameters which control noise frequency and magnitude, respectively. In our experiments, $s_n = 1.0, \beta_n = 0.2$. The simulated depth image is noiseless.

For the world model, we constructed a simple 2D TSDF. We compare the performance of ARM-SLAM (Alg. 2) against a simple unconstrained descent algorithm which assumes the sensor can move and rotate freely, without considering the robot kinematics (Fig. 3.2). We found that ARM-SLAM managed to both reduce end effector error and dramatically reduce model error (Table 3.1), whereas just using a 2D dense fusion technique without constraining using the robot's kinematics resulted in severe, unrecoverable drift because of the scene's self-similarity and the robot's fast motion. Note that in the real experiments, there is comparatively much less actuator noise, and a much smaller scene than in the 2D experiments.

3.3.2 3D Simulation

We developed a 3D simulation of a Kinova *Mico* robot with a hand-mounted Occipital *Structure* [44] depth sensor. In the simulation, the robot scans a simulated bookshelf. As in the 2D experiments, Perlin noise is added to the ground truth joint angles to simulate actuator uncertainty. We use the *Open Chisel* [31] chunked TSDF library for mapping. The simulated depth image is noiseless. Reconstructions were done at a resolution of 1.5 cm per voxel.

We found that ARM-SLAM was able to correct for very large actuator error (see Fig. 3.4), resulting in a final reconstruction near the ground truth (Fig. 3.3). By artificially increasing the actuator noise, we found that ARM-SLAM significantly reduced the end effector error even when the uncertainty in the camera's pose was up to 12 cm (Fig. 3.4a), we also found ARM-SLAM to

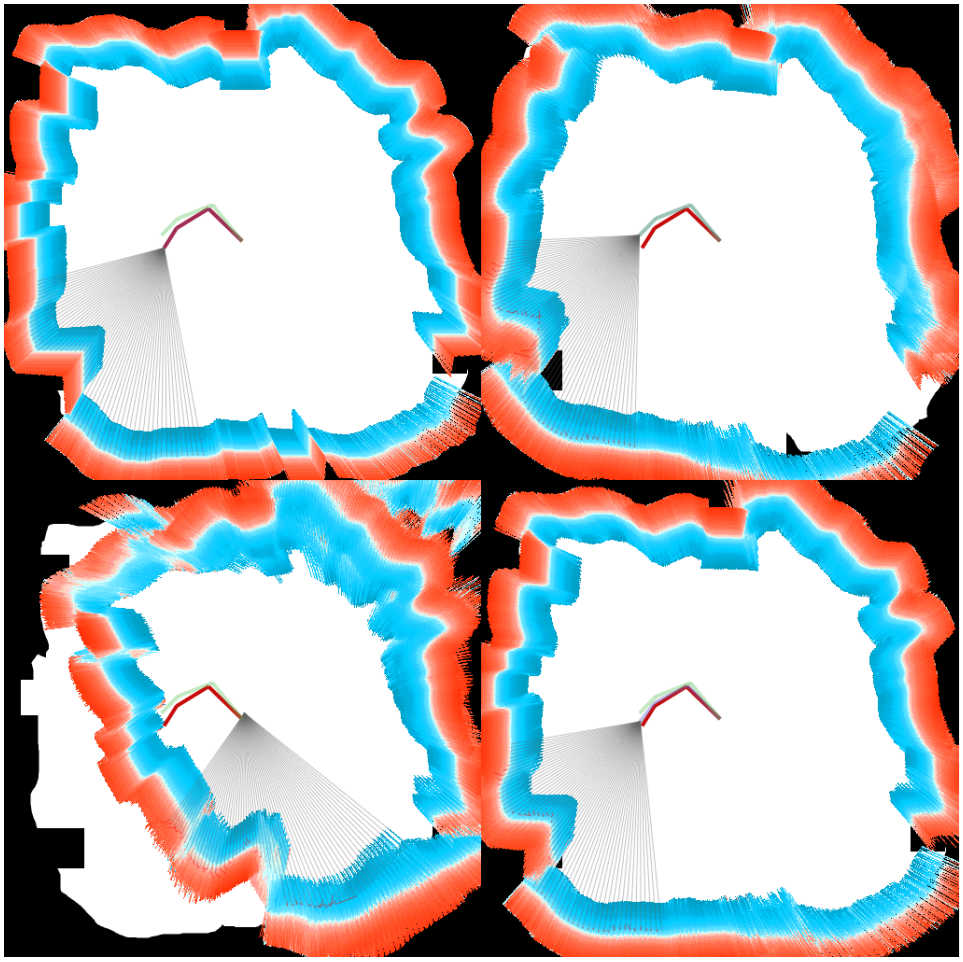
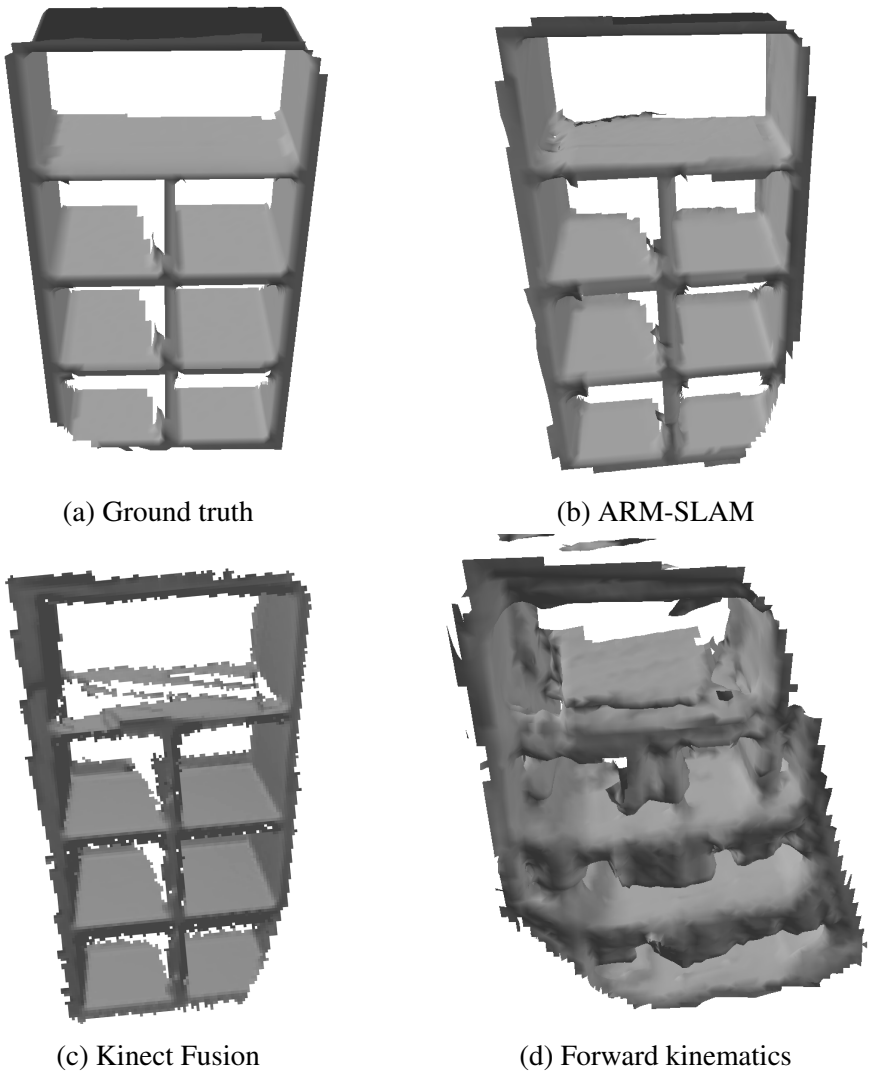


Figure 3.2: 2D simulation experiment (see page 19). The robot is shown in red. The simulated depth image is shown as gray rays. The TSDF is shown as orange or blue pixels. Top left shows the ground truth TSDF, top right is with forward kinematics only (with actuator uncertainty). Bottom left corrects actuator noise using unconstrained dense fusion. Bottom right corrects using ARM-SLAM.

be more robust to tracking failure from lost data than unconstrained Kinect Fusion (Fig. 3.4b) due to the very strong motion prior from the robot kinematics.

3.3.3 Bookshelf Scanning

Using the same framework as in the 3D simulation, we reconstructed a bookshelf with a Kinova *Mico* robot with a hand-mounted Occipital *Structure* sensor (Fig. 3.1). The robot was teleoperated using a joystick. Beforehand, the Structure sensor was extrinsically calibrated to the robot's hand using the Tsai[16] method and a fiducial, though extrinsic calibration error cannot be ruled out. The end effector deviation was measured using an *Optitrack* motion tracking system. One challenge of working with the real robot data is that the joint encoders and depth sensor are not synchronized. The joint encoder data is emitted at $\sim$ 500 Hz, whereas the camera data is



(a) Ground truth

(b) ARM-SLAM

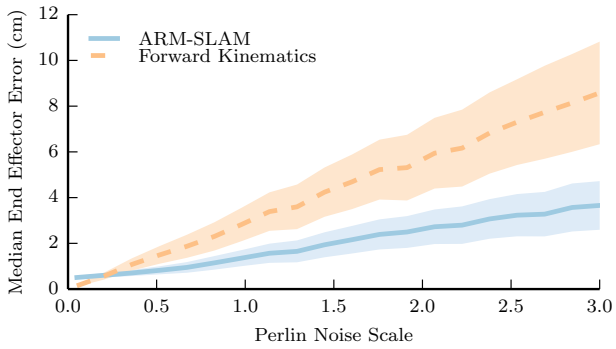
(c) Kinect Fusion

(d) Forward kinematics

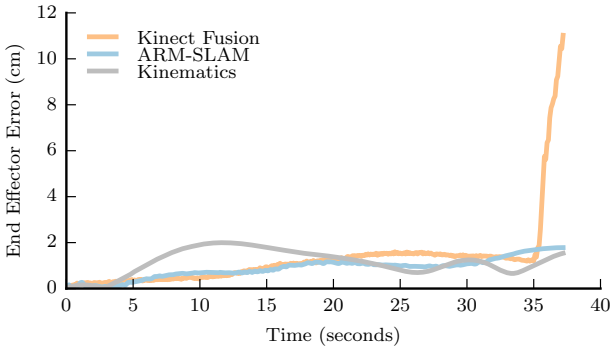
Figure 3.3: Results of the 3D simulation (see page 19) with up to 0.8 radians of added noise per joint.

produced at 30 Hz. To compensate for this, we store the robot's configuration space trajectory as a series of linearly interpolated, time-stamped waypoints. Using this, we can infer the joint encoder readings at the time when the depth image was received.

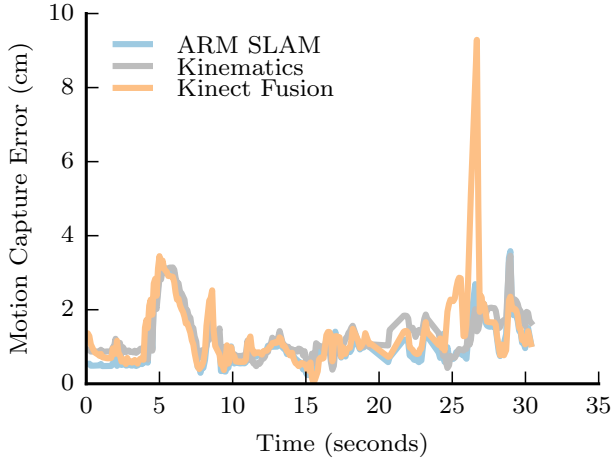
The 3D reconstructions (Fig. 3.1) show that our method is able to recover 3D structure in the scene that is lost when only the (noisy) forward kinematics are used. This is especially apparent around the edges of the bookshelf and its adjacent walls. Our reconstructions are comparable to Kinect Fusion run at the same voxel resolution (1.5 cm). We measured end-effector motion with an optical motion capture system (Fig. 3.4c) and found that Kinect fusion occasionally lost (and regained) tracking due to self-similar surfaces in the bookshelf and surrounding walls. Because of the strong motion prior from the robot's joints ARM-SLAM did not have this issue. However, our data from the motion capture system is too noisy to conclude ARM-SLAM performed any



(a) 3D simulation with added noise.



(b) 3D simulation with lost tracking.



(c) Real data validated with motion capture.

Figure 3.4: End effector error observed in the 3D simulation (see page 19) experiments. Fig. 3.4a : 100 trials with different noise seeds are run with increasing noise magnitude. Each trial lasts 60 seconds. The median deviation of the end effector from ground truth is recorded. Fig. 3.4b : in a different simulation, the robot briefly looks away from the scene and then looks back. Kinect Fusion loses tracking. Fig. 3.4c : end-effector deviation in the real dataset as measured by an optical motion capture system, Kinect Fusion briefly loses and then regains tracking.

better than forward kinematics at reporting the true pose of the end effector (ARM-SLAM had an end effector deviation of 1.2 ± 0.9cm while forward kinematics had a deviation of 1.4 ± 1.0cm). It may be that extrinsic calibration error between the sensor and rigid hand mount is dominating any error produced at the robot's joints.

3.4 Discussion and Limitations

In this chapter, we introduced a framework (called ARM-SLAM) for dense visual SLAM in a robot's configuration space. We have shown that our approach is capable of reconstructing scenes and reducing actuator uncertainty simultaneously. Unfortunately, ARM-SLAM has several limitations which must be addressed in the next few chapters.

First, since it is a pure model-based dense SLAM approach (like *Kinect Fusion*[24]), it suffers from many of the problems that plague these approaches. The system requires clear geometric structure and a large field of view to localize correctly, and since it uses no global *pose graph*, it is susceptible to drift over longer trajectories. Further, we are only able to track the configuration of the robot when a depth image is available. Also like those approaches, the underlying tracking and mapping techniques are largely based on geometric arguments, making it difficult to incorporate probabilistic models. As a consequence, we don't have a way of tracking the uncertainty in the predicted joint angles.

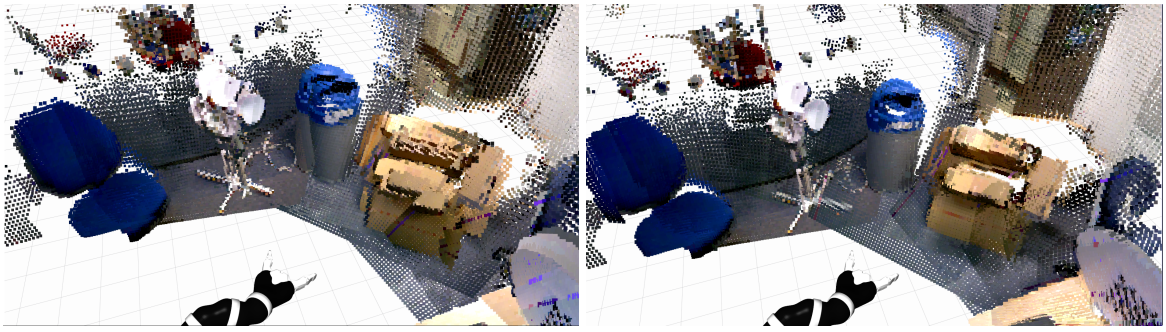
By committing to localization in the configuration space of the robot, rather than $SE(3)$, we gain the benefit of only predicting physically plausible camera poses. We are also able to express costs and priors (such as joint limit and self-collision costs) on robot configuration trivially. On the other hand, error that can't be expressed in the configuration space (such as error in the extrinsic calibration, or motion of the robot base) cannot be corrected for using our technique. Also, the more joints a robot has in comparison to $SE(3)$, the more work our technique has to do to compute Jacobian terms, and the larger the camera motion null-space is (worsening susceptibility to local minima). For instance, a 2-jointed robot pan-tilt head would be comparatively easy to localize vs. a highly redundant 50-jointed snake robot.

In spite of these limitations, our approach provides a good baseline for more complex SLAM techniques designed to address these problems that we will examine in the next chapter.

CHAPTER 4

Sparse General SLAM using a Hand-Mounted Monocular Camera

The previous chapters described techniques for tracking robot arms in real-time using both external depth cameras, and depth cameras mounted on the robot arm itself. Though fast and efficient, these techniques are both essentially simple filters on the robot's state that make a strong *Markov assumption*. That is, they assume that the current state of the robot can be derived entirely from the previous state of the robot and current sensor measurements. In the case of ARM-SLAM (the algorithm presented in chapter 3), the algorithm suffers from drift in both the robot's state and the map because of this assumption. Further, these techniques only considered the joint angles of the robot as the state of the system, ignoring error from the extrinsic calibration of the sensor, motion of the base, or inaccuracies in the robot's model. Further, they made no attempt at modeling the uncertainty of the robot state, and only predicted the maximum likelihood estimate of the state.



(a) Forward kinematics only. (b) iSAM2 with BRISK features.

Figure 4.1: Closeup of stitched point clouds obtained from pose estimates using the robot's kinematics only, and the SLAM system. The SLAM system calibrates the camera extrinsics, greatly improving the consistency of the map.

In this chapter, we will address these limitations by invoking a general graphical model that describes arm tracking and calibration. This model is able to support sensors of many kinds (even monocular sensors), and combines data from the joint encoders with data from the camera in a principled way. By making the model more general, we are able to add more degrees of freedom to the state (at some expense to performance).

4.1 Problem Definition

Consider the case of a monocular (*i.e.* 2D) camera attached to a robot arm with inaccurate joint angle sensing. Suppose that the camera is attached in such a way that it can't see any part of the robot's body. As the robot moves around and collects data, how can we:

- Predict the robot's joint angles over time?
- Create a consistent 3D map of the scene?
- Find the extrinsic calibration of the sensor relative to the end effector?
- Estimate the uncertainty in all of these unknown variables?

Formally, at a series of discrete time-steps $1, \ldots, T$, the robot receives synchronized monocular images $\mathbf{I}_1, \ldots, \mathbf{I}_T$ and joint encoder measurements of position $\theta_1, \ldots, \theta_T$ and velocity $\dot{\theta}_t, \ldots \dot{\theta}_t$. Unknown state variables of the system we wish to estimate include the robot's configuration $\mathbf{q}_1, \ldots, \mathbf{q}_T$, an unknown fixed extrinsic calibration $T_{\text{cam}} \in SE(3)$, and an unknown map $\mathbf{M}$, which we will represent as a series of discrete landmarks $\{\mathbf{l}_1, \ldots, \mathbf{l}_M\} \in \mathbb{R}^3$.

4.2 Graphical Model

The problem can be stated as a graphical model with factors representing priors and measurement posteriors connecting nodes for 3D landmarks, robot configurations, and camera extrinsics. This graphical model can be used as the basis for a sparse SLAM system (see page 56) that incrementally maximizes the likelihood of the factor graph. Fig. 4.2 shows an example of the graphical model used.

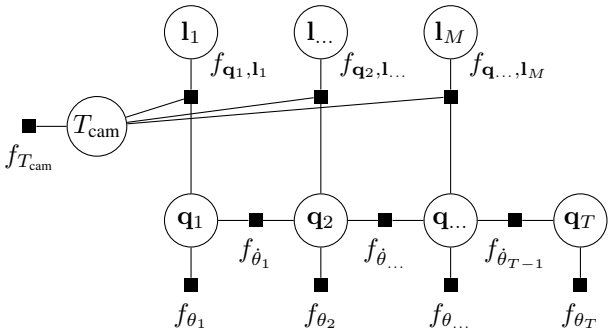


Figure 4.2: Graphical model of the SLAM system.

Each time a camera at time-step t observes landmark i, we have a factor $f_{\mathbf{q}_t,\mathbf{l}_i}$ that correlates together the camera calibration, robot configuration, and landmark position. For simplicity Fig. 4.2 shows only one landmark observation per time-step, but in fact there may be many landmarks observed. The joint encoders are connected to robot configurations by the factors f_{θ_t}. Robot configurations are themselves correlated over time by the factors $f_{\dot{\theta}_{t-1}}$. Note that due to the formulation of the factor graph, we assume all these factors to be conditionally independent of one another.

In the coming sections, we will describe how each of these nodes and factors are expressed in practice.

4.2.1 Landmark Observation Factors

The factors $f_{\mathbf{q}_t,\mathbf{l}_i}$ of the graph (Fig. 4.2) represent the posterior probability of the robot's configuration, camera extrinsics, and landmark position given observations of the landmark i at time t (Fig. 4.3).

$$f_{\mathbf{q}_t,\mathbf{l}_i} = \mathrm{P}\left(\mathbf{z}_{i,t}|\mathbf{q}_t, \mathbf{l}_i, T_{\text{cam}}\right) \tag{4.1}$$

here, we model a measurement of landmark $\mathbf{l}_i$ in camera image I_t as a 2D pixel coordinate $\mathbf{z}_{i,t}$. The 3D landmark positions $\mathbf{l}_i$ is projected onto the camera image using the projection function

$$F_\pi\left(\mathbf{q}_t, T_{\text{cam}}, \mathbf{l}_i\right) = \pi\left(\left(F(\mathbf{q})T_{\text{cam}}\right)^{-1}\mathbf{l}_i\right) \tag{4.2}$$

the measurement $\mathbf{z}_{i,t}$ can be modeled as a random variable sampled from the distribution

$$\mathbf{z}_{i,t} \sim F_\pi\left(\mathbf{q}_t, T_{\text{cam}}, \mathbf{l}_i\right) + \mathcal{N}(\mu_{\mathbf{z}}, \Sigma_{\mathbf{z}}) \tag{4.3}$$

where $\mu_{\mathbf{z}}, \Sigma_{\mathbf{z}}$ are the mean and variance of the re-projection noise, respectively. Therefore,

$$\mathrm{P}\left(\mathbf{z}_{i,t}|\mathbf{q}_t, \mathbf{l}_i, T_{\text{cam}}\right) = \mathcal{N}_{\mu_{\mathbf{z}},\Sigma_{\mathbf{z}}}\left(F_\pi\left(\mathbf{q}_t, T_{\text{cam}}, \mathbf{l}_i\right) - \mathbf{z}_{i,t}\right) \tag{4.4}$$

The Jacobian of each factor with respect to each state variable is also needed in practice to optimize the graphical model. Each of these is computed by an application of the chain rule to Eq. 4.4 . (The terms for the partial derivatives of the pixel location $\mathbf{z}$ with respect to the landmark $\mathbf{l}$ and the camera extrinsic T_{cam} are taken from GTSAM [11], and make use of the differential geometry of $SE(3)$ (section A.1.1)):

$$[x, y, z]^T = \left(F(\mathbf{q}_t)T_{\text{cam}}\right)^{-1}\mathbf{l}_i \tag{4.5}$$

$$\frac{\partial \mathbf{z}_{i,t}}{\partial \mathbf{l}_i} = \begin{bmatrix} f_x\frac{1}{z} & 0 & -f_x x\frac{1}{z^2} \\ 0 & f_y\frac{1}{z} & -f_y y\frac{1}{z^2} \end{bmatrix} \tag{4.6}$$

$$\frac{\partial}{\partial \mathbf{l}_i} f_{\mathbf{q}_t,\mathbf{l}_i} = \frac{\partial \mathbf{z}_{i,t}}{\partial \mathbf{l}_i} \tag{4.7}$$

$$\frac{\partial}{\partial \mathbf{q}_t} f_{\mathbf{q}_t,\mathbf{l}_i} = -\frac{\partial \mathbf{z}_{i,t}}{\partial \mathbf{l}_i}\mathbf{J}_{\mathbf{l}_i}(\mathbf{q}_t) \tag{4.8}$$

$$\frac{\partial}{\partial T_{\text{cam}}} f_{\mathbf{q}_t,\mathbf{l}_i} = \begin{bmatrix} xy & -1-x^2 & y & -\frac{1}{z} & 0 & \frac{x}{z} \\ 1+y^2 & -xy & -x & 0 & -\frac{1}{z} & \frac{y}{z} \end{bmatrix} \tag{4.9}$$

where F is the forward kinematics function, and $\mathbf{J}_{\mathbf{l}_i}(\mathbf{q}_t)$ is the $3 \times N$ kinematic linear Jacobian (see page 64) for the point $\mathbf{l}_i$ at robot configuration $\mathbf{q}_t$.

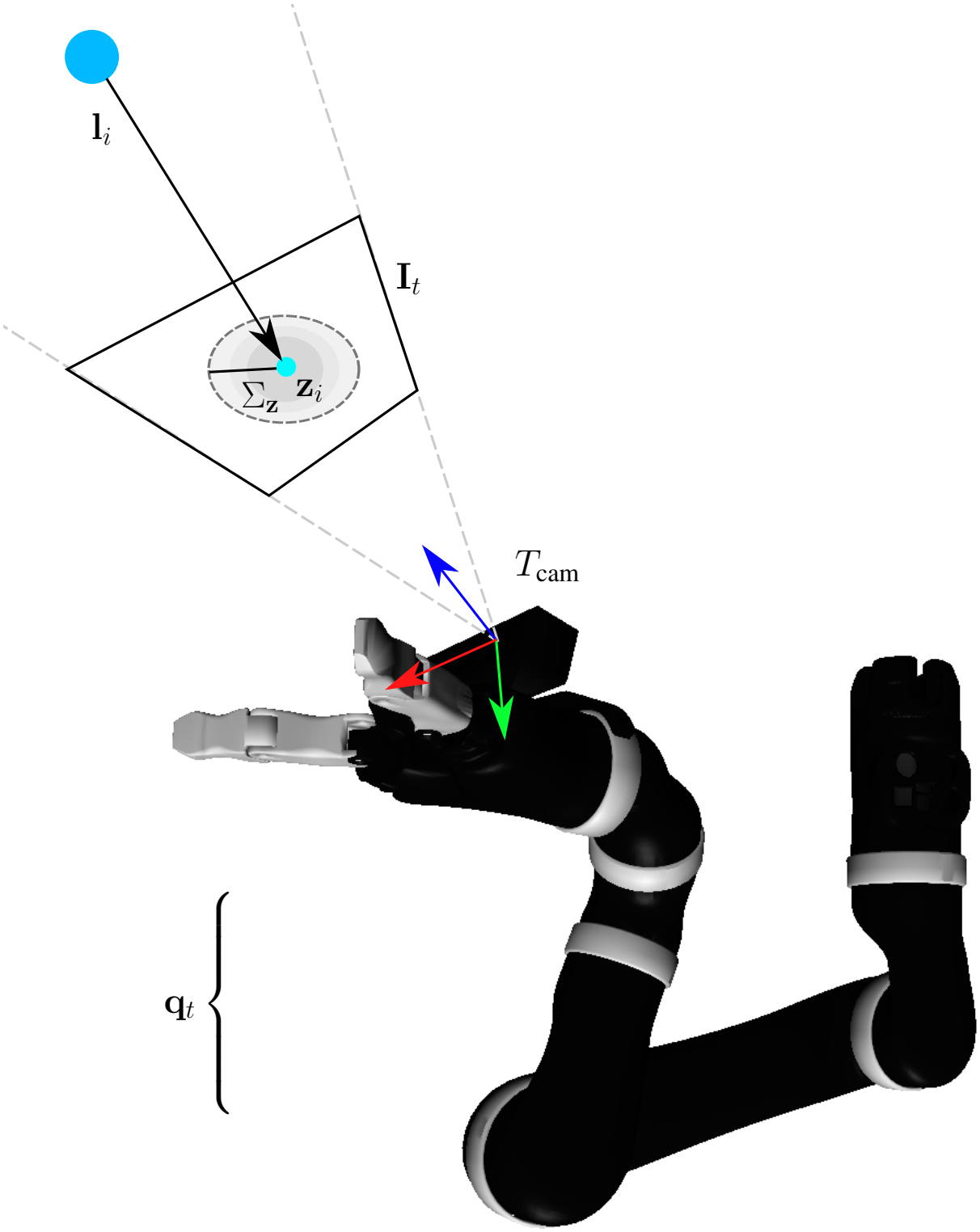


Figure 4.3: Landmark observation factor. The robot at configuration $\mathbf{q}_t$ observes landmark $\mathbf{l}_i$ in image $\mathbf{I}_t$. The noise is on the re-projection of the landmark into the image.

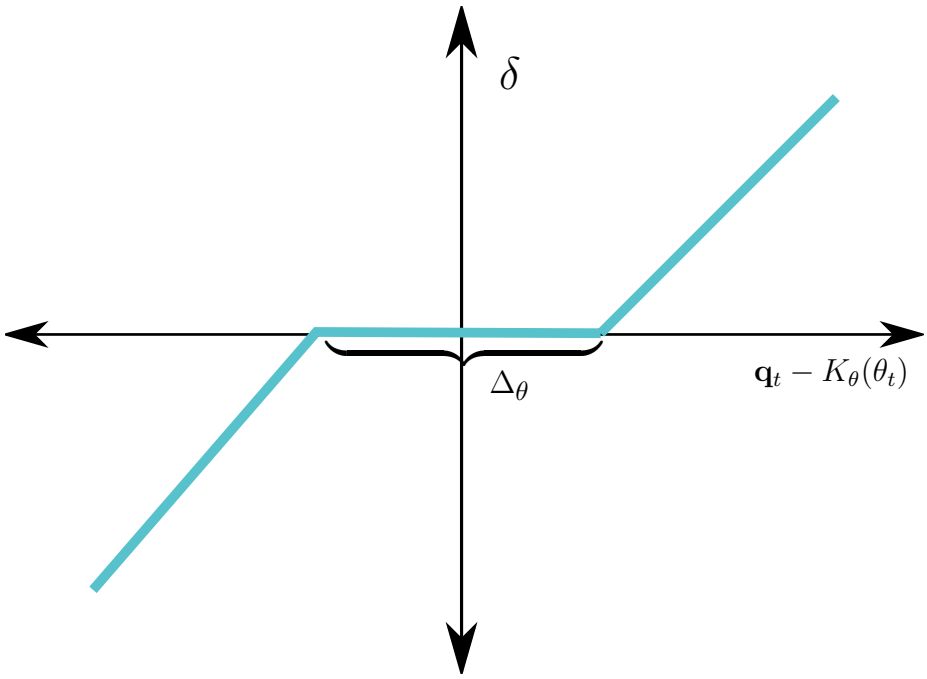


Figure 4.4: Joint encoder dead-zone function δ to account for gear backlash. Differences between configuration and joint encoders of under Δ_θ are ignored.

4.2.2 Joint Encoder Factors

The factors f_{θ_t} represent the posterior probability of the robot's configuration given the joint encoder measurements at time t. That is

$$f_{\theta_t} = \mathrm{P}\left(\theta_t | \mathbf{q}_t\right) \tag{4.10}$$

as in [32], we assume that the joint encoders are random variables drawn from Gaussian noise on the true configuration of the robot

$$\theta_t \sim K_\theta^{-1}\left(\mathbf{q}_t + \mathcal{N}(\mu_\theta, \Sigma_\theta)\right) \tag{4.11}$$

where $\mu_\theta, \Sigma_\theta$ are the mean and covariance of the joint angle noise, respectively, and K_θ is the static joint encoder calibration function (see page 65). Therefore the posterior is

$$\mathrm{P}\left(\theta_t | \mathbf{q}_t\right) = \mathcal{N}_{\mu_\theta, \Sigma_\theta}\left(\mathbf{q}_t - K_\theta(\theta_t)\right) \tag{4.12}$$

To account for gear backlash (see page 65), we add a small *dead band* Δ_θ so that configurations within Δ_θ of the joint encoders are not penalized:

$$\epsilon = \mathbf{q}_t - K_\theta(\theta_t) \tag{4.13}$$

$$\mathrm{P}\left(\theta_t | \mathbf{q}_t\right) = \mathcal{N}_{\mu_\theta, \Sigma_\theta}\left(\delta(\epsilon)\right) \tag{4.14}$$

$$\delta(\epsilon) = \max\left(0, \|\epsilon\| - \Delta_\theta\right) \operatorname{sgn}\left(\epsilon\right) \tag{4.15}$$

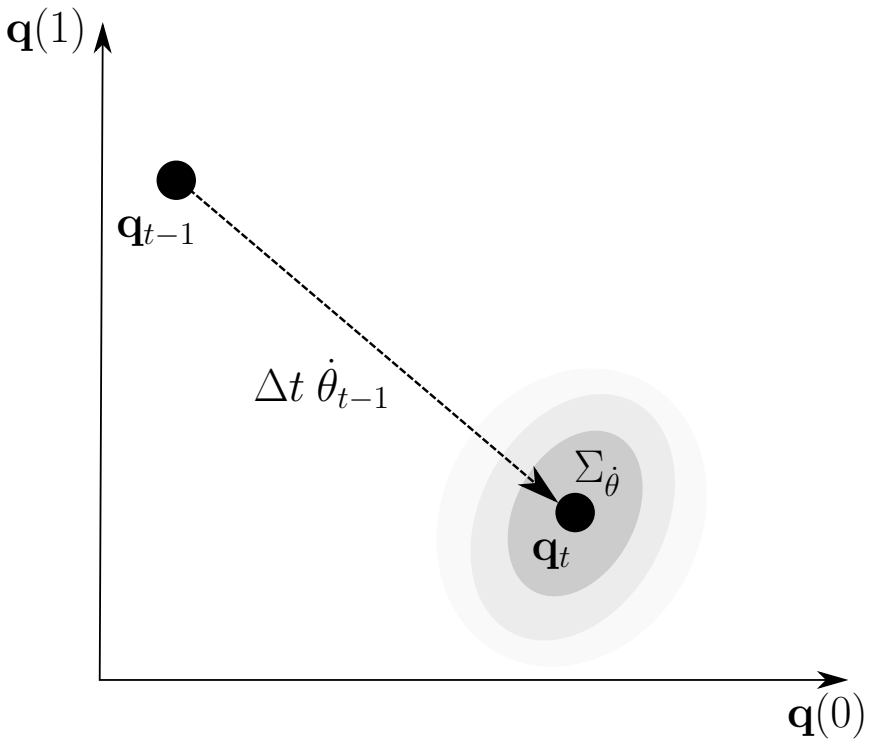


Figure 4.5: Robot dynamics factor shown for a 2-jointed robot. The joint velocity measurements at time $t-1$ are noisy, with covariance $\Sigma_{\dot{\theta}}$

where sgn is the sign function. The gear backlash term δ allows the joint angles of the robot to vary within the dead-band Δ_θ without penalty (Fig. 4.4).

Since the joint encoders directly measure the joint angles, the Jacobian of these factors are simply the $N \times N$ identity.

4.2.3 Robot Dynamics Factors

The factors $f_{\dot{\theta}_{t-1}}$ correlate the robot's joint angles given joint velocity measurements $\dot{\theta}_{t-1}$ (Fig. 4.5). That is

$$f_{\dot{\theta}_{t-1}} = \mathrm{P}(\dot{\theta}_{t-1}|\mathbf{q}_{t-1}, \mathbf{q}_t) \tag{4.16}$$

we assume that the velocity measurements are random variables drawn from the true velocity of the robot's joints. That is

$$\dot{\theta}_{t-1} \sim \frac{\mathbf{q}_t - \mathbf{q}_{t-1}}{\Delta t} + \mathcal{N}(\mu_{\dot{\theta}}, \Sigma_{\dot{\theta}}) \tag{4.17}$$

where δt is the time-step between time t and $t-1$, $\mu_{\dot{theta}}$ is the mean of the velocity noise, and $\Sigma_{\dot{\theta}}$ is its covariance. Therefore

$$\mathrm{P}(\dot{\theta}_{t-1}|\mathbf{q}_{t-1}, \mathbf{q}_t) = \mathcal{N}_{\mu_{\dot{\theta}}, \Sigma_{\dot{\theta}}} \left(\dot{\theta}_{t-1} - \frac{\mathbf{q}_t - \mathbf{q}_{t-1}}{\Delta t} \right) \tag{4.18}$$

The dynamics factors smooth our estimate of the joint angle trajectory so that it is closer to physically feasible than if only joint encoder factors are used. In part, we do this because the joint encoder posterior is not really representative of the true system dynamics that cause the joint encoders to be inaccurate. The inaccuracy stems from unmodeled systematic error that depends in part on the robot's configuration and torques it is applying. Adding a factor to constrain the motion of the joints allows us to smoothly predict the joint encoder error without having to explicitly model the physical process that causes it. Note that for some robots, a direct measurement of the velocity $\dot{\theta}$ may be unavailable. In this case, the finite differences of the joint encoders may be used. In principle, this means that the joint encoder factors f_θ and the dynamics factors $f_{\dot{\theta}}$ are not really independent.

The Jacobian of these factors is the negative $N \times N$ identity for $\mathbf{q}_1$ and the positive $N \times N$ identity for $\mathbf{q}_2$.

4.2.4 Extrinsic Prior

The factor $f_{T_{\text{cam}}}$ describes the prior distribution for the camera extrinsics. This is simply

$$f_{T_{\text{cam}}} = \mathrm{P}(T_{\text{cam}}) \tag{4.19}$$

$$= \mathcal{N}_{\mu_{T_{\text{cam}}}, \Sigma_{T_{\text{cam}}}}(T_{\text{cam}}) \tag{4.20}$$

where $\mu_{T_{\text{cam}}}, \Sigma_{T_{\text{cam}}}$ are the mean and covariance of the extrinsic prior, respectively. The Jacobian of this factor is the simple 6×6 identity.

4.3 SLAM Back-end System Implementation

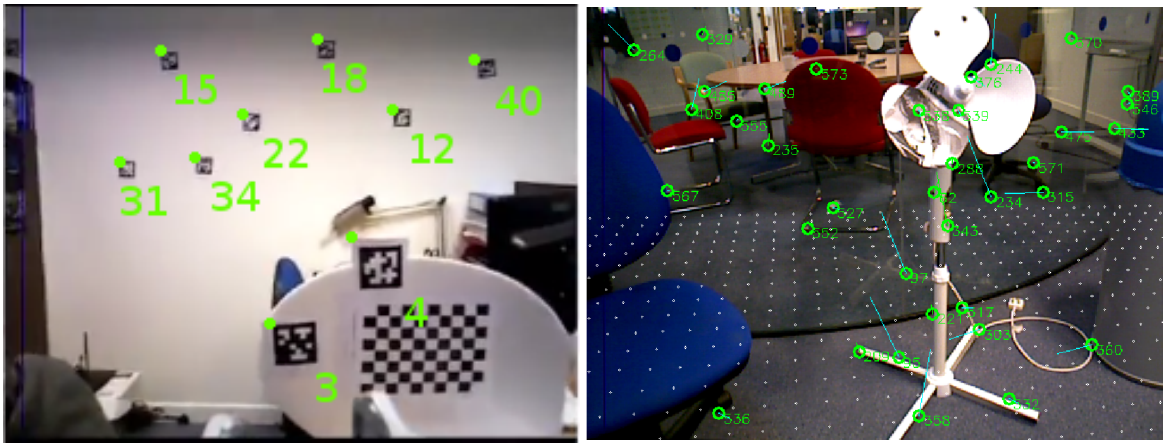
As the robot receives joint encoder measurements and observations of landmarks, we add nodes and factors to the graphical model, which is then jointly optimized to find a maximum likelihood estimate of all the unknown variables ($\mathbf{q}_1, \ldots \mathbf{q}_T$, $\mathbf{l}_1, \ldots, \mathbf{l}_M$ and T_{cam}) given all the measurements. We consider both batch and online solutions.

For the batch solution, we use Levenberg-Marquadt. For the online solution, we use the Incremental Smoothing and Mapping 2 (iSAM2) algorithm [26]. For an explanation of how these algorithms work, see see page 56. We use off-the-shelf implementations of these algorithms from the Georgia Tech Smoothing and Mapping (GTSAM) [11] library, and make use of the outlier-robust Cauchy estimator found in GTSAM (see page 68). Once factored, the graphical model can be queried for the maximum likelihood estimate of the state as well as marginal estimates of each of the parameters.

Algorithm 3 outlines the high-level online graphical SLAM algorithm. As new images are received, features are detected and matched to existing landmarks in the graph. New landmarks are triangulated based on their previous observations. Factors are added for the joint position and velocity measurements. Finally, the graph is incrementally optimized.

// Given new synchronized joint angle measurements and camera images, and a graphical model
Input: $\mathbf{I}_t, \theta_t, \dot{\theta}_t G$
// If the robot has moved more than a threshold.
if $\|\theta_t - \theta_{t-1}\| > d_\theta$ **then**
    // Add factors for joint encoders and velocities
    $G \leftarrow G + \{f_{\theta_t}, f_{\dot{\theta}_t}\}$
    // Find visible features in the graph
    $L_{\text{visible}} \leftarrow \textsc{CullFeatures}(\theta_t, G)$
    // Match visible features to the image.
    $L_{\text{new}} \leftarrow \textsc{DetectFeatures}(\mathbf{I}_t, \theta_t, L_{\text{visible}})$
    // Triangulate new features using existing observations
    $L_{\text{new}} \leftarrow \textsc{Triangulate}(L_{\text{new}}, G)$
    // Add observation factors to the graph
    for $\mathbf{l}_i \in L_{new}$ **do**
        $G \leftarrow G + \{f_{\mathbf{l}_i, \theta_t}\}$
    end
end
// Optimize the graphical model
$G \leftarrow \textsc{Optimize}(G)$
Output: G

Algorithm 3: High level graphical SLAM algorithm.



(a) April tags

(b) BRISK keypoints.

Figure 4.6: Detected April tags and their associated IDs, as well as detected BRISK keypoints with their associated IDs.

4.4 Landmark Front-end System Implementation

To acquire landmarks from the camera images, we consider two systems: one based on April [45] tags, and another based on natural image features using the Binary Robust Invariant Scalable Keypoint (BRISK)[36] features.

4.4.1 April Tag Front-end Implementation

April tags are fiducials designed for computer vision and augmented reality consisting of unique grid patterns. These tags can be detected easily in 2D images, and with a known intrinsic calibration and tag size, their poses relative to the sensor can be estimated. Using April tags, the landmark $\mathbf{l}_i$ is simply the 3D location of the top-left corner of the April tag with tag id i. The initial position estimate of $\mathbf{l}_i$ is given by the pose estimate returned by the April tags detector (which also reports the pixel locations of each of the tag's corners).

The April tags were used mainly to test the SLAM back-end system without having to worry about landmark association. Our system does not require the tags to detect and associate landmarks, but will use them when available.

4.4.2 Natural Image Feature Front-end Implementation

Acquiring and associating 3D landmarks from a series of natural images is somewhat more complicated than acquiring 3D landmarks from fiducials. First, we acquire a number of BRISK keypoints (see page 69) $k_1, \ldots, k_n$ in the image $\mathbf{I}_t$ roughly corresponding to corners and lines. Each keypoint is associated with a descriptor $d_1, \ldots, d_n$.

Then, for each keypoint, we match it to the closest potential 3D landmark visible from the estimated robot configuration $\mathbf{q}_t$. We reject the match whenever the descriptor d_i's Hamming distance to the existing landmark's last observed descriptor is too far, and accept it otherwise.

Rejected matches get added to the pool of new potential 3D landmarks, but are not added to the factor graph until the following conditions are met:

- The landmark has been matched to two or more camera images.
- There is a sufficient stereo baseline between camera observations of the landmark.
- The landmark can be successfully stereo-triangulated (see page 67) from its observations.

The stereo triangulation system uses the current estimate of the robot's configuration to estimate the motion between frames. By conservatively eliminating outliers in this way, we minimize false-positive associations between image features. Still, some false-positive associations occur.

4.5 Experiments

We attached an Asus *Xtion Pro* to the hand of a Kinova *Mico* robot, calibrated the camera intrinsics using a checkerboard fiducial, and scanned several scenes using the robot's joystick. The depth image from the Asus Xtion was not used in the SLAM system, but was only used to display the resulting reconstruction as a stitched point cloud. We use the depth image as a way of measuring the reconstruction quality of the SLAM system.

4.5.1 Results

We commanded the robot to scan an office scene using a joystick. The robot makes a loop around the scene, which consists of several objects in an office. Fig. 4.1 shows a closeup of the scene reconstructed using stitched point clouds from the hand-mounted RGB-D sensor. We hand-coded an initial guess for the camera extrinsics. Fig. 4.1a shows the resulting reconstruction when only the forward kinematics with the initial extrinsic guess is used, while Fig. 4.7b shows the resulting reconstruction using our SLAM system. Fig. 4.7 shows a similar reconstruction of a tabletop scene. Visually, the SLAM system greatly improves the quality of the map, eliminating many double surfaces and other misalignment issues. Table 4.1 shows the parameters used during this reconstruction. Notice that the uncertainty on the camera prior is much higher comparatively than the uncertainty on the joint angle measurements. Figure 4.8 shows the final extrinsic estimate from the SLAM system as compared with an image of the real robot. Visually, it appears that the extrinsic estimate is one or two centimeters further back than the real camera, but is otherwise in the correct location.

4.5.2 Performance

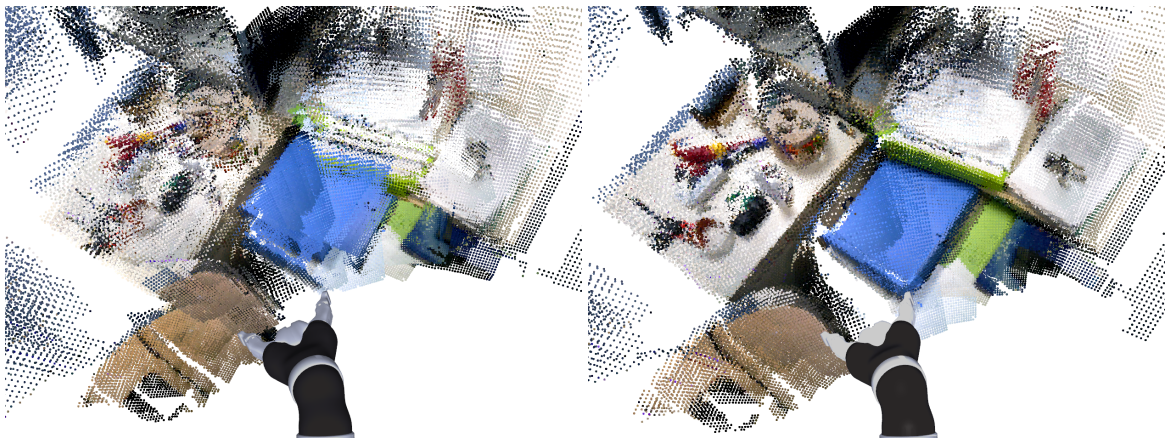
Table 4.2 shows per-iteration timing data for the SLAM system using iSAM2. Here "Graph Optimization" refers to the time spent optimizing the factor graph, "Stereo Triangulation" refers to the time spent running RANSAC and triangulating landmarks between frames, "Display" refers to the time spent drawing debug data and displaying the point cloud, "Landmark Extraction" refers to the time spent extracting and associating BRISK features to landmarks, and "Landmark pre-cull" refers to the time spent frustum culling potential landmarks for association with new keyframes.

Parameter	Value
d_θ	0.01 radians
$w_\mathcal{C}$	3.0
$\Sigma_\mathbf{z}$	$1.0\mathbf{I}_{2\times 2}$
$\mu_\mathbf{z}$	$\mathbf{0}$
Σ_θ	$0.05\mathbf{I}_{6\times 6}$
μ_θ	$\mathbf{0}$
$\Sigma_{T_{\text{cam}}}$	$0.5\mathbf{I}_{6\times 6}$
$\mu_{T_{\text{cam}}}$	$\mathbf{I}$
$\Sigma_{\dot{\theta}}$	$0.01\mathbf{I}_{6\times 6}$
$\mu_{\dot{\theta}}$	$\mathbf{0}$
Δ_θ	0.01 radians

Table 4.1: Parameters of the SLAM system used in the real data set (Fig. 4.1).

Component	Max (ms)	Mean (ms)
Graph Optimization	181	48
Stereo Triangulation	137	45
Display	121	26
Landmark Extraction	66	19
Landmark pre-cull	35	12

Table 4.2: Per-iteration timing data for the SLAM system using iSAM2.



(a) Forward kinematics only.

(b) iSAM2 with BRISK features.

Figure 4.7: Closeup of stitched point clouds obtained from pose estimates using the robot's kinematics only, and the SLAM system. The SLAM system calibrates the camera extrinsics, greatly improving the consistency of the map. In this data set the robot scanned a nearby tabletop several times from different orientations and heights.

Position f_θ	Velocity $f_{\dot{\theta}}$	$\mathbf{q}$ Err. (deg)	T_{cam} Err. (cm)	Reproj. Err. (pix)
✓	✓	**0.34**	**0.28**	**0.74**
✓	✗	0.78	0.31	1.35
✗	✓	1.66	0.76	1.52
✗	✗	4.21	9.35	38.17

Table 4.3: Effect of using joint angle position and velocity measurements as priors for camera motion. Data are for a single random trajectory for a randomly generated 6DOF robot in simulation. f_θ refers to whether the joint position encoders, or $f_{\dot{\theta}}$ the joint velocity encoders were added to the factor graph. The median joint angle error, final extrinsic translation error, and final median landmark reprojection error are shown.

Figure 4.9 shows the number of landmarks observed during the data set in Figure 3.1. The number of landmarks with two or more observations rapidly increases. At the end of the data set, there are over 1500 landmarks, mostly with 4 or more observations. Landmarks with only one observation are not included in the factor graph.

Table 4.3 shows the effect of turning on or off the joint position and velocity measurements for a simulated 6-DOF robot. As expected, without strong priors from the arm motion, the SLAM system struggles to track both the position of the end effector and the joint angles. Both velocity measurements and position measurements can be used to track the robot arm, though position measurements are more informative since they do not contribute to drift over time.

4.5.3 Simulated ground truth

We implemented a simulated system based on the SceneNet [17] camera simulator. The robot uses a real joint angle trajectory recorded from user input. The true joint angles are corrupted by

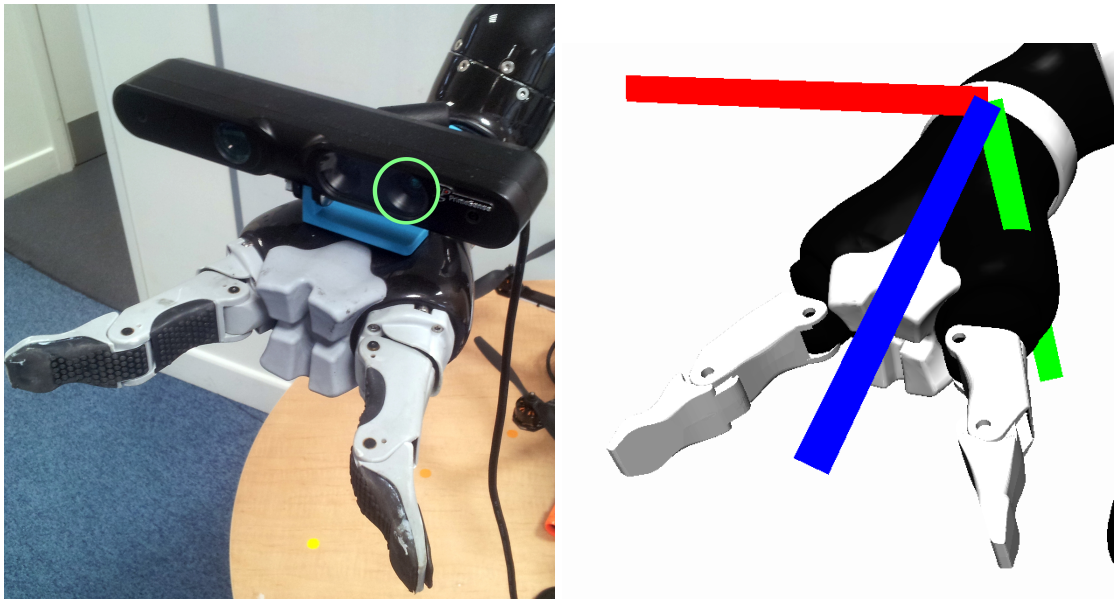


Figure 4.8: The final extrinsic estimate of the camera in the real data set (Fig. 4.1) (right) compared with a picture of the real robot (left). The lens of the RGB camera is highlighted by a green circle.

Perlin [48] noise with a magnitude of 0.1 radians per joint. The initial extrinsic of the camera is corrupted by an incorrect 30cm translation offset and 20 degrees of angular noise. The color images are noiseless and unlit. The robot scans the room making two loops, collecting 600 (320×240) images. Figure 4.10 shows simulated images used in the data set.

Figure 4.11 shows stitched point clouds from the keyframes collected over the trajectory. Using the incorrect extrinsics and noisy encoders, the bedroom has severe reconstruction errors including double or blurred surfaces. Even under these severely noisy conditions, our SLAM system produces a reconstruction nearly indistinguishable from ground truth data. Figure 4.12 shows the stitched point cloud produced by the SLAM system compared with the ground truth point cloud using the CloudCompare [14] software. Almost all points in the cloud are within a centimeter of the nearest ground truth point. There is a slight misregistration in one of the corners, where the point-to-point error is around 6 centimeters.

Figure 4.13b shows the median landmark reprojection error over the course of the experiment. As the camera extrinsics and joint angles are corrected, the landmark reprojection error decreases from ~ 5 pixels per landmark to ~ 0.5 pixels per landmark. Figure 4.13a shows the error of the final maximum likelihood estimate for the joint angle trajectory. Each joint angle individually is predicted to within 2 degrees of ground truth. Figure 4.14 shows the error of the camera extrinsic estimate over the iterations of the SLAM system. Here, x points right, y up, and z forward. The error is reduced from $\sim 10cm$ on all axes to less than $1cm$ along z and $\sim 1cm$ of error remaining along x and y. Figure 4.14c shows the error of the final maximum likelihood estimate for the camera position. The final error in position is comparatively very low, and is less than a centimeter along all axes. Note that the systematic error here is a result of the robot's trajectory.

One disappointing result was that when the noise on the joint angles was increased, the SLAM

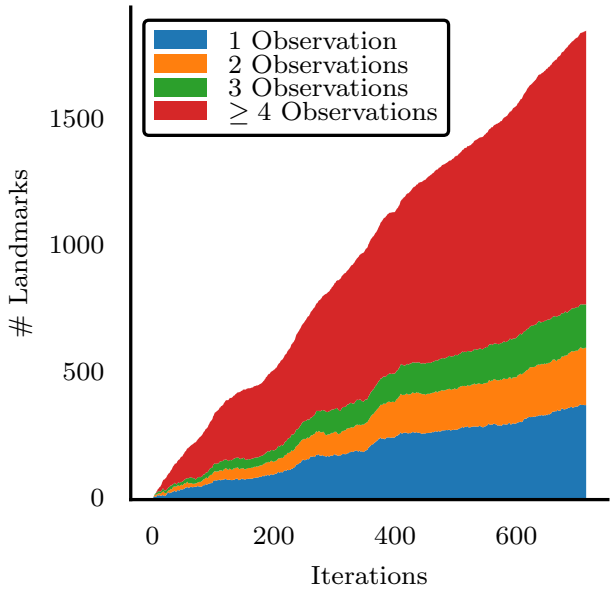


Figure 4.9: Number of landmarks by observation over time in a real data set. Landmarks with only one observation are not included in the factor graph.

system struggled with kinematic redundancies (Fig. 4.16). In particular, the Mico arm has two joints that have parallel axes of rotation. In some situations, it is difficult for the SLAM system to resolve whether both joints are moving in opposite directions, or whether neither joint is moving. Even though the pose estimate and reconstruction of the scene is accurate, the SLAM system fails to accurately predict the joint angles in these cases. We will explore why this is the case in the next section.

4.5.4 Modifying the Robot's Kinematic Structure

With another set of experiments, we wanted to explore the relationship between the structure of the robot and its camera parameters to the ease of calibration and SLAM using a simulated camera.

Figure 4.15e shows an experiment in which we varied the simulated camera's field of view between 90 and 20 degrees. The system tends to be more reliable with a larger field of view, because it is able to associate features over larger number of frames – but as the field of view increases the resolution of the image becomes an issue.

Figure 4.15a shows several randomly-generated fractal robots that were used to determine the relationship between the robot's number of degrees of freedom, and the ease of calibrating and tracking it using a visual sensor. The robots were generated by chaining together rotational joints at random orientations. The links of the robot randomly decrease in length along the chain, and have random orientation noise. For each robot, we generated a random trajectory (Fig. 4.17) which is a random walk through the robot's configuration space achieved by applying simulated random velocities to the robot's joints. This was repeated 20 times for each robot. 1000 random landmarks were sampled in a 5 meter box around the robot, and these were used for SLAM rather

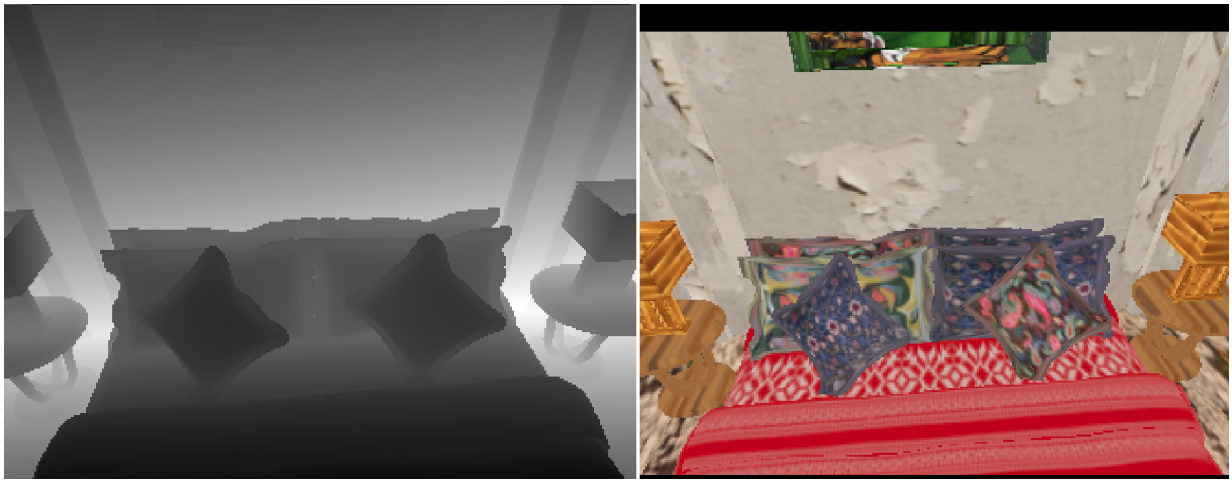


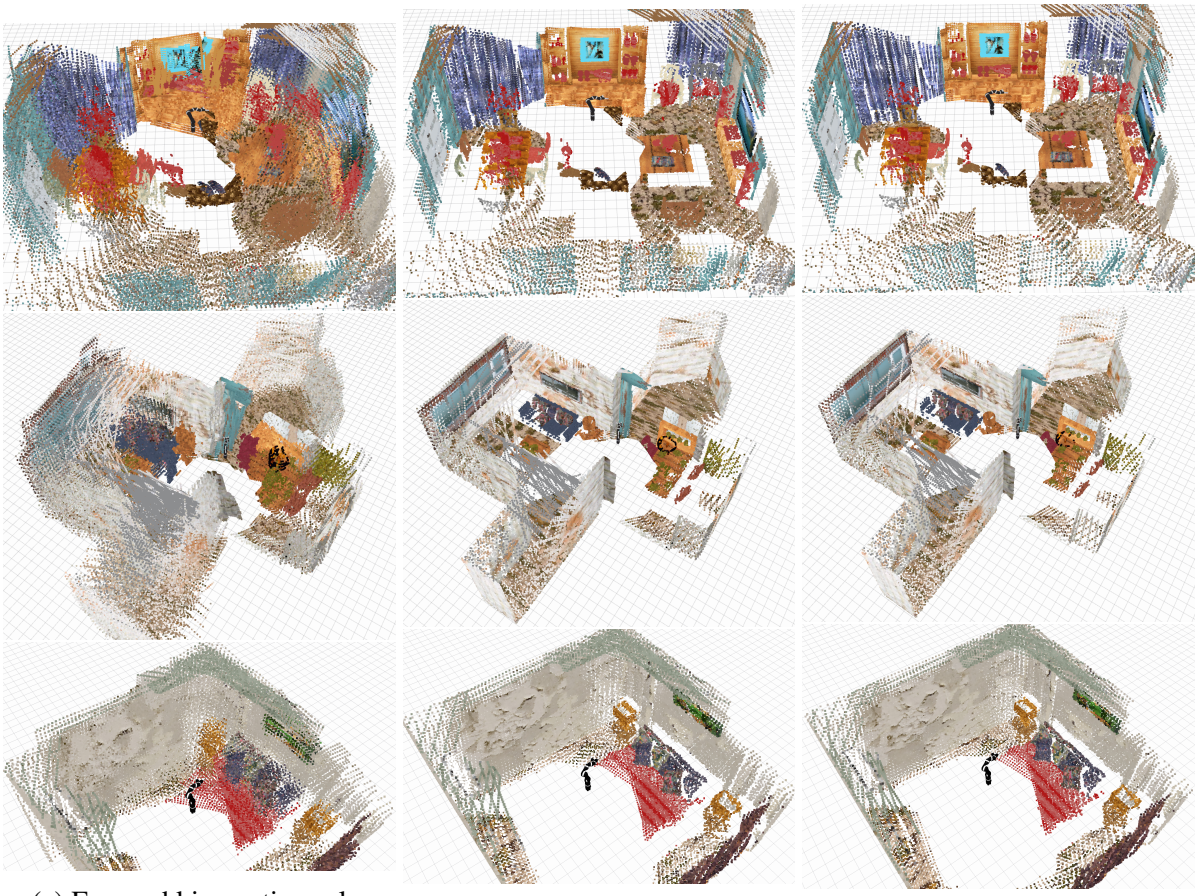
Figure 4.10: Simulated depth (left) and color (right) images.

than a simulated camera image so that the camera measurements were not biased by the scene structure. These landmarks are projected onto a simulated camera and noise is applied to their measurements. Isometric Gaussian noise was added to the joint measurements with a standard deviation of 0.05 radians. An offset of $17cm$ was applied to the camera extrinsics with rotation offsets around x, y and z of 0.1 radians.

Figure 4.15c shows how the final maximum likelihood estimate of the camera extrinsic changes with the number of degrees of freedom of the robot. With only 1, 2, or 3 degrees of freedom, the SLAM system struggles to find the correct extrinsic estimate. We believe this is because the kinds of trajectories that the robot is able to generate with so few degrees of freedom do not produce high enough baseline matches for landmarks between frames. The SLAM system performs about equally well at estimating the camera extrinsics for robots with 4 or more degrees of freedom.

Figure 4.15b shows how the median reprojection error of the landmarks changes with the number of degrees of freedom. The median reprojection error seems to modestly increase from ~ 1 pixel of error to ~ 2 pixels of error as the number of degrees of freedom increases from 1 to 19.

Figure 4.15d shows the median absolute joint angle error of the SLAM system's final maximum likelihood estimate. As the number of degrees of freedom increases, the SLAM system is less able to predict the true joint angles of the robot – in spite of the fact that the camera extrinsics are calibrated about equally well. This is because as the number of joints increases, the system becomes more and more under-constrained, requiring more measurements from the camera to accurately predict. There is also an effect caused by the decreasing size of each of the robot's links – as more links are added to the robot, the marginal additional motion at the end effector introduced by that degree of freedom is reduced, making it more difficult to predict those degrees of freedom from reprojection error alone.



(a) Forward kinematics only. (b) iSAM2 with BRISK features. (c) Ground truth

Figure 4.11: Stitched point clouds from three simulated data sets in which random noise was added to the initial camera extrinsic estimate and joint encoders.

4.5.5 Comparison with Other SLAM Systems

We tested our SLAM system against two other pure visual SLAM systems, Kinect Fusion [24] (a fully dense method using only the depth image) and ORB-SLAM [42] (a state of the art key-frame based monocular SLAM system) on a simulated dataset.

Figure 4.18 shows the camera trajectories produced by each SLAM system, and 4.19 shows the translational error in the camera position over these trajectories. All of the trajectories were transformed into the same reference frame such that the camera position at time 0 is the identity. Note that since ORB-SLAM is a purely monocular visual SLAM system, there is scale ambiguity in its trajectory estimate. We scaled the ORB-SLAM trajectory such that its total error against ground truth was minimized. The optimal scale factor was found through rough binary search to be around 4.22. The trajectories were time-synchronized and compared using linear interpolation.

Our system (labeled ISAM2) outperforms both Kinect Fusion and ORB-SLAM in this simulated dataset (Table 4.4). For the first five seconds or so, our method, Kinect Fusion, and ORB-SLAM perform about equally well, with Kinect Fusion performing the best. However,

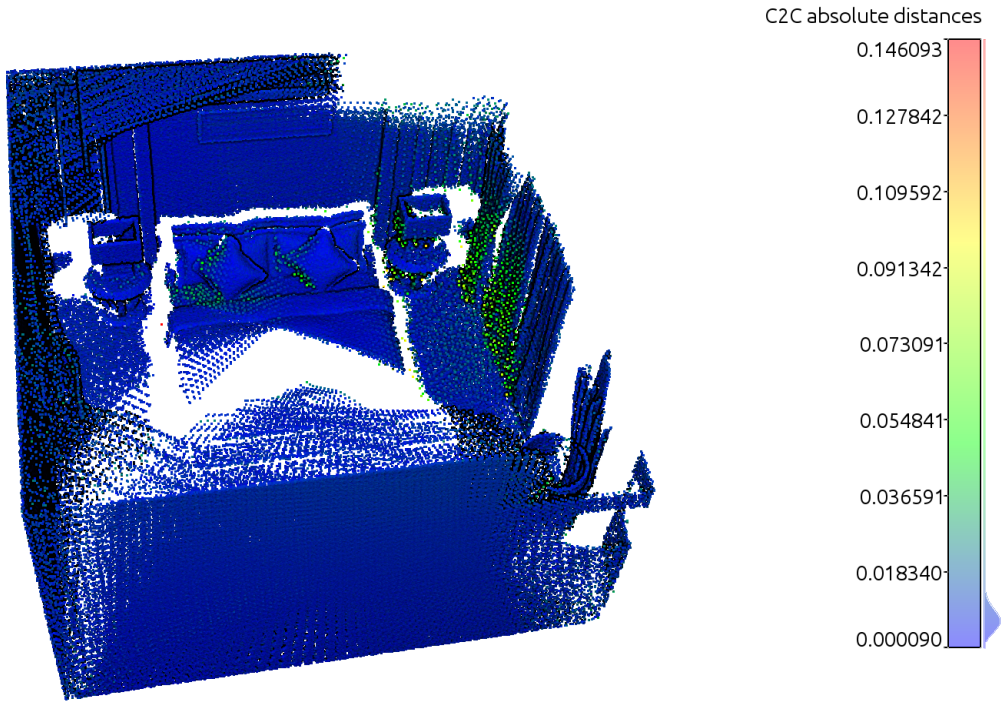


Figure 4.12: The stitched point cloud created using the SLAM system compared with the ground truth point cloud.

after around 30 seconds, both Kinect Fusion and ORB-SLAM drift away from ground truth and never recover. ORB-SLAM in particular completely loses tracking when the robot's trajectory contains a pure rotation, while Kinect Fusion fails when the camera is looking only at a flat floor. ORB-SLAM regains tracking at around 150 seconds.

By using the robot's joint angle measurements as a strong prior, our system is able to recover the motion of the camera much more robustly than the pure visual SLAM systems, and is also automatically able to recover the trajectory's scale.

Time (s)	ISAM2	Kinematics	Kinect Fusion	ORB-SLAM
10	$\mathbf{0.01 \pm 0.001}$	0.17 ± 0.003	0.01 ± 0.010	0.04 ± 0.014
30	$\mathbf{0.01 \pm 0.001}$	0.17 ± 0.004	0.08 ± 0.141	0.04 ± 0.035
150	$\mathbf{0.01 \pm 0.001}$	0.17 ± 0.006	0.61 ± 0.436	0.32 ± 0.208

Table 4.4: Comparison of SLAM systems on a simulated dataset. Camera translation error is shown in meters for the first 10, 30 and 150 seconds of the dataset. The initial simulated noise added to the robot's kinematics is also shown for reference. Our method (here called ISAM2) consistently has less translation error than Kinect Fusion and ORB-SLAM, especially as the dataset evolves.

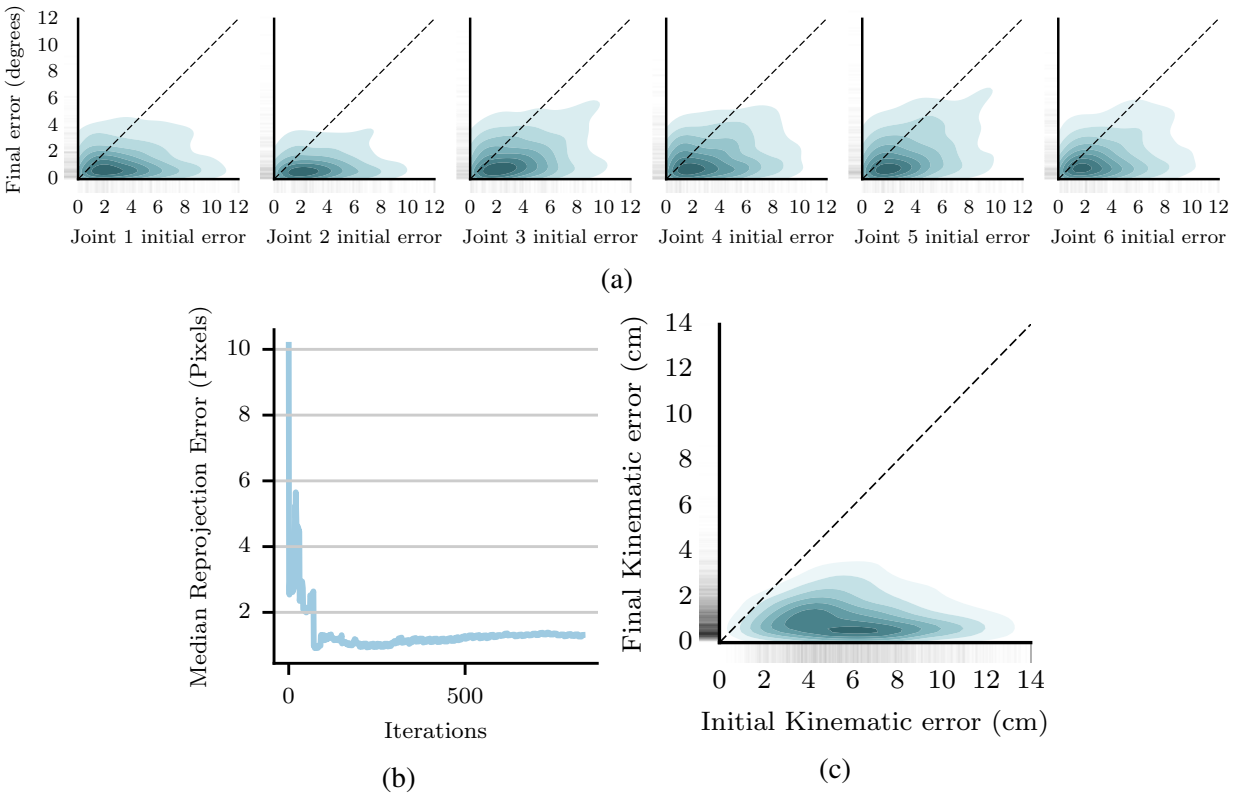
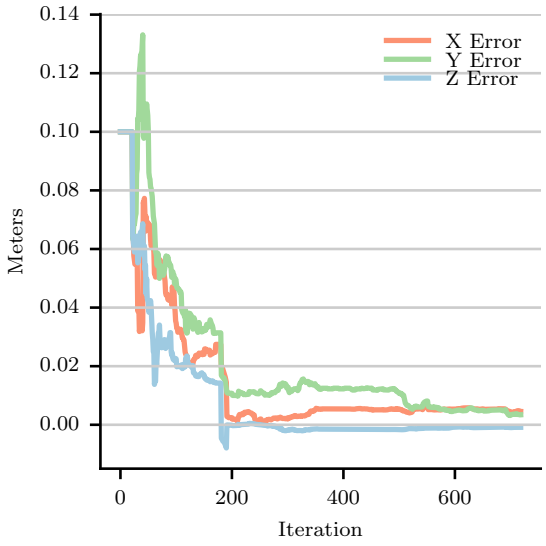
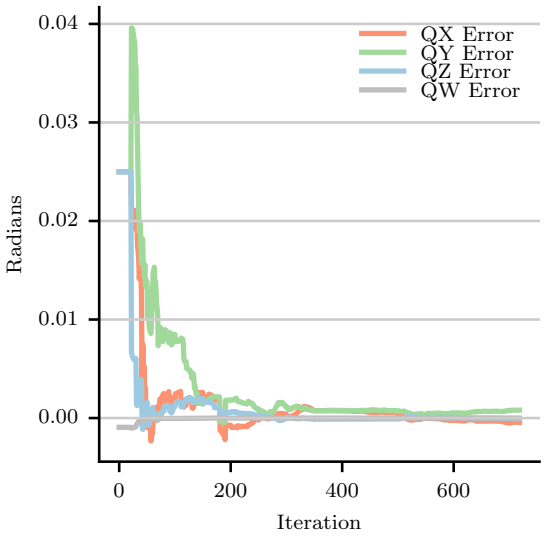


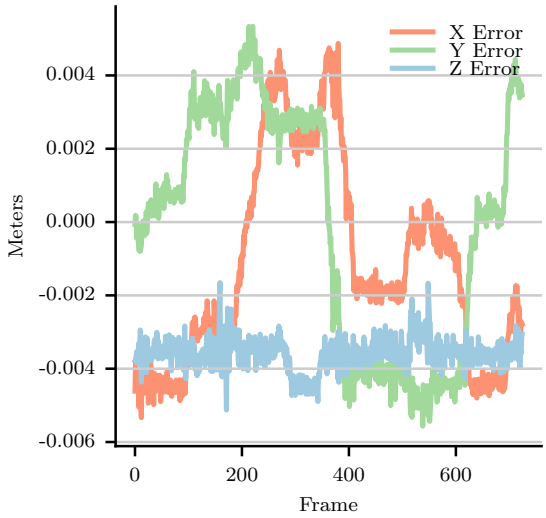
Figure 4.13: Fig. 4.13a shows absolute joint angle error in degrees in the simulated data set. The initial simulated error is drawn from a multivariate Gaussian (x axis) and is then corrected by the SLAM system (y axis). The plot shows how much error is reduced for each joint. If the SLAM system were not correcting joint angle error, all of the samples would fall on the diagonal dotted line. If the SLAM system corrected all of the joint error, then all of the samples would fall on the x axis. Fig. 4.13b shows median landmark reprojection error in the simulated data set. This is the median error over all observed landmarks per iteration of the SLAM system. Fig. 4.13c shows the translational end effector error reduction in the simulated data set as a kernel density of 2,500 samples (darker patches have more density).



(a) Extrinsic translation error.

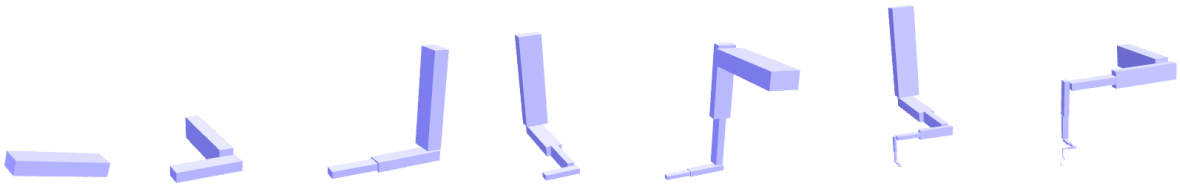


(b) Extrinsic rotation (quaternion) error.

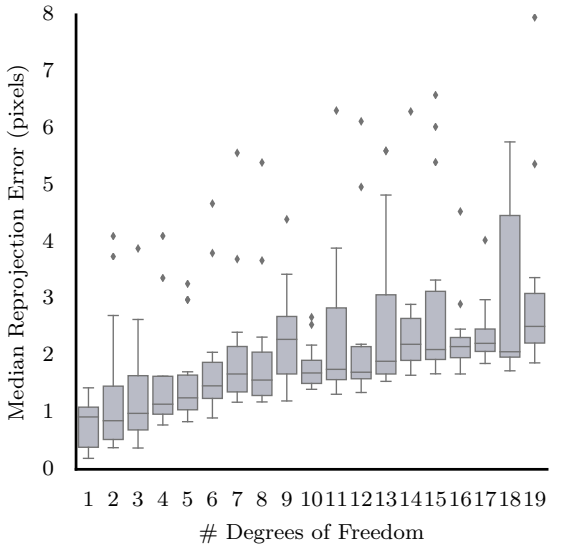


(c) End effector position error.

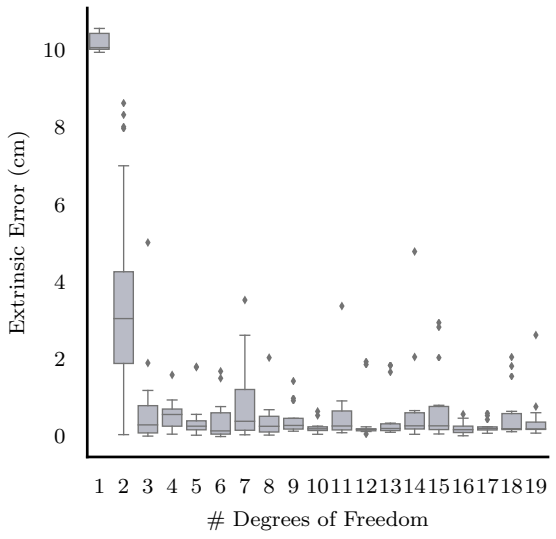
Figure 4.14: Camera extrinsic error in the simulated data set per iteration of the SLAM system. Figures 4.14a and 4.14b show the translation and rotation error of the extrinsic estimate per iteration of SLAM. Figure 4.14c shows the final end effector position error over the length of the trajectory.



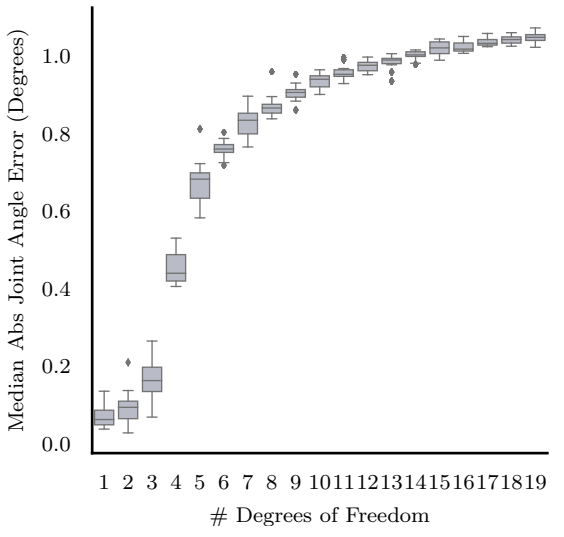
(a) Auto-generated fractal robots with 1, 2, 3, 4, 6, 10 and 20 degrees of freedom.



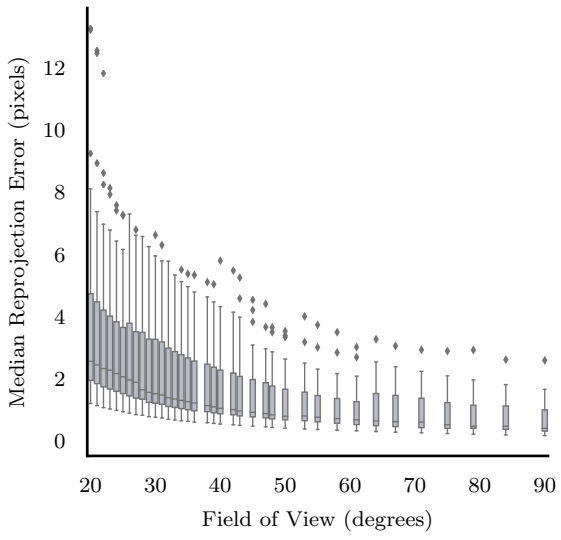
(b)



(c)



(d)



(e)

Figure 4.15: Simulated experiments exploring the effect of varying the number of degrees of freedom and camera field of view. Figures 4.15b, 4.15c, and 4.15d show the final median reprojection error, error on the camera extrinsic, and absolute error per joint angle respectively over increasing degrees of freedom. 20 experiments were performed for each number of DOFs. In figure 4.15e, the camera field of view is varied between 90 and 20 degrees, with 50 random experiments per condition on a random 6-DOF robot.

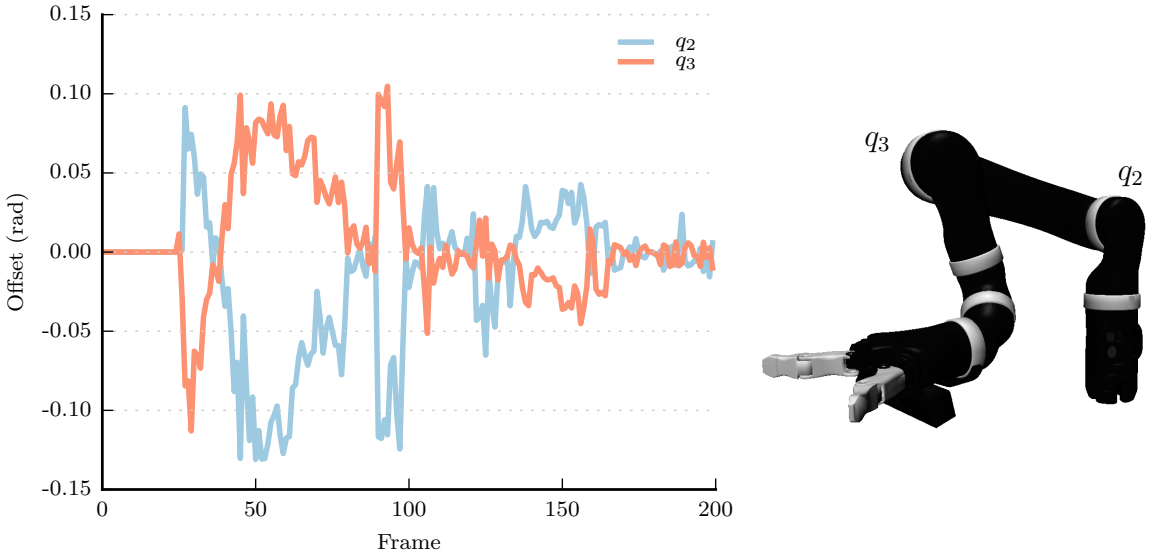


Figure 4.16: Illustration of kinematic redundancy. The true joint encoder offset is zero. q_2 and q_3 have parallel axes of rotation, so any rotation where $q_2 = -q_3$ results in a net rotation near zero. The SLAM system struggles when these redundancies are present, especially when the translational baseline between frames is small.

4.6 Discussion and Limitations

In this chapter, we set ourselves the task of calibrating and tracking a noisy robot arm using a single monocular camera looking at an unstructured world. By using Bayesian factor-graph based SLAM, we produced a system capable of simultaneously calibrating a robot's extrinsics, producing a map of 3D landmarks, and tracking the robot's joint angles with a single monocular camera.

Unfortunately, our experiments have revealed clear limitations to our approach. The key problem is that since the camera is outward facing, it can observe neither the joint angles nor the extrinsic calibration directly. Therefore, we have to infer the joint angles and extrinsic calibration from the reprojection error of the landmarks alone. As the number of degrees of freedom of the robot increase, the task of calibrating and tracking it becomes more difficult due to the fact that more data is required to prevent the SLAM problem from being under-constrained. In the extreme cases shown in figure 4.15a, where the number of degrees of freedom is much more than 6, so much data is needed to uniquely constrain the joint angles of the robot that estimating them is not possible (even when the extrinsic calibration can be estimated well).

In this work, we have referred to the problem as "kinematic redundancy" (Fig. 4.16), but in general what we have is a case of *unobservability*. In control theory, a system is said to be *observable* when its state can be inferred from its measurable output [57]. Generally, we can write a system as a function of the state which produces some output:

$$f(\mathbf{x}_t) = \mathbf{y}_t \tag{4.21}$$

If the system were fully observable, it would be possible to infer the inverse of the system such

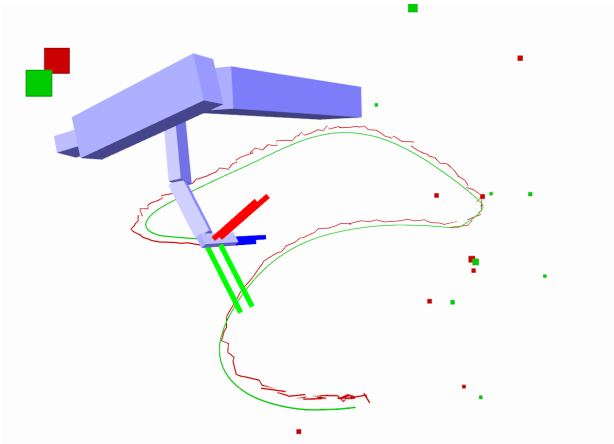


Figure 4.17: An example of a simulated and automatically generated robot and trajectory for a 6 DOF robot. The maximum likelihood estimate is shown in red. The ground truth is shown in green. Landmarks are represented as squares. The camera extrinsic is shown as an axis in the middle of the last link.

that

$$f^{-1}(\mathbf{y}_t) = \mathbf{x}_t \tag{4.22}$$

For example, if we consider a simple linear system

$$f(\mathbf{x}_t) = A\mathbf{x}_t + \mathbf{b} = \mathbf{y}_t \tag{4.23}$$

then the observability of f would depend on the inevitability of A, and the dimensions of $A, \mathbf{x}_t$ and $\mathbf{y}_t$. In other words, f is observable if and only if this linear system is fully constrained. In the under-constrained case, sometimes it is still possible to find a solution for $\mathbf{x}_t$ that satisfies all the constraints – for example by using the pseudo-inverse of A to produce a minimum-norm solution, but there is no guarantee that this solution represents the true state of the system. In general, a linear system will be fully constrained if the rank of A is greater than or equal to the size of $\mathbf{x}_t$.

So, is the problem outlined in chapter 4 observable? This depends entirely on the trajectory that the robot has taken, its number of degrees of freedom, and the observations it has made. The method we use (Bayesian inference on a factor graph) finds a maximum likelihood estimate by producing a state estimate that minimizes the reprojection error over all of the observations. In iSAM2 [26] and other systems that rely on this kind of inference, this is accomplished by linearizing the system state around the current estimate and forming the system of equations

$$\mathbf{J}^T\Sigma^{-1}\mathbf{J}\Delta\mathbf{x} = \mathbf{J}^T\Sigma^{-1}\Delta\mathbf{z} \tag{4.24}$$

where $\mathbf{J}$ is the Jacobian of the full system, $\Delta\mathbf{x}$ is a change in state (trajectory, landmark locations, etc.), $\Delta\mathbf{z}$ is the reprojection error, and Σ is the full covariance matrix of the system. The SLAM system then solves for $\Delta\mathbf{x}$ so that the measurement error is reduced toward zero. At first glance,

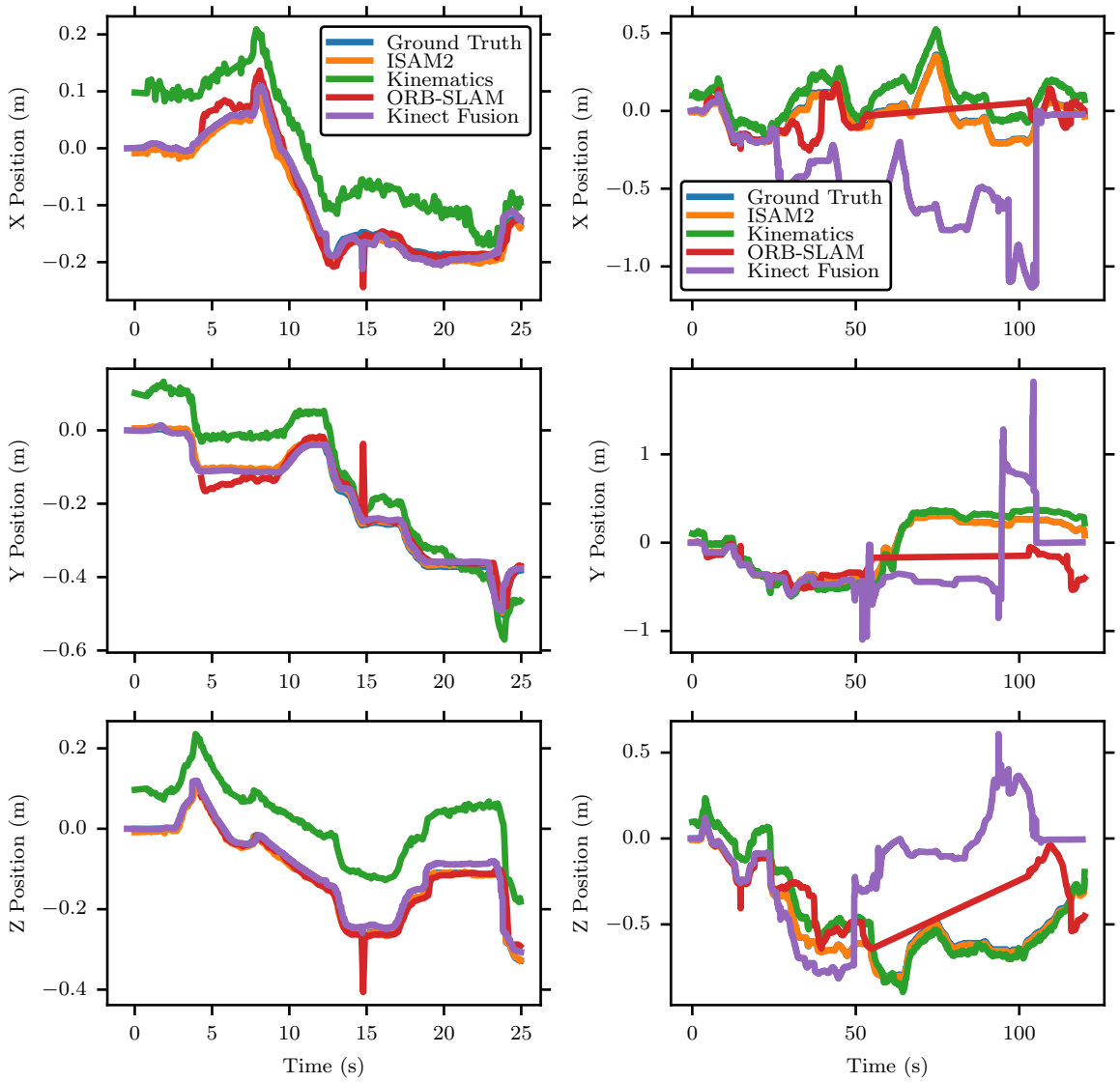


Figure 4.18: Estimated camera trajectory in a simulated dataset using different SLAM systems. The left figure shows the trajectory during the first 25 seconds, the right figure shows the same trajectory during the first 120 seconds.

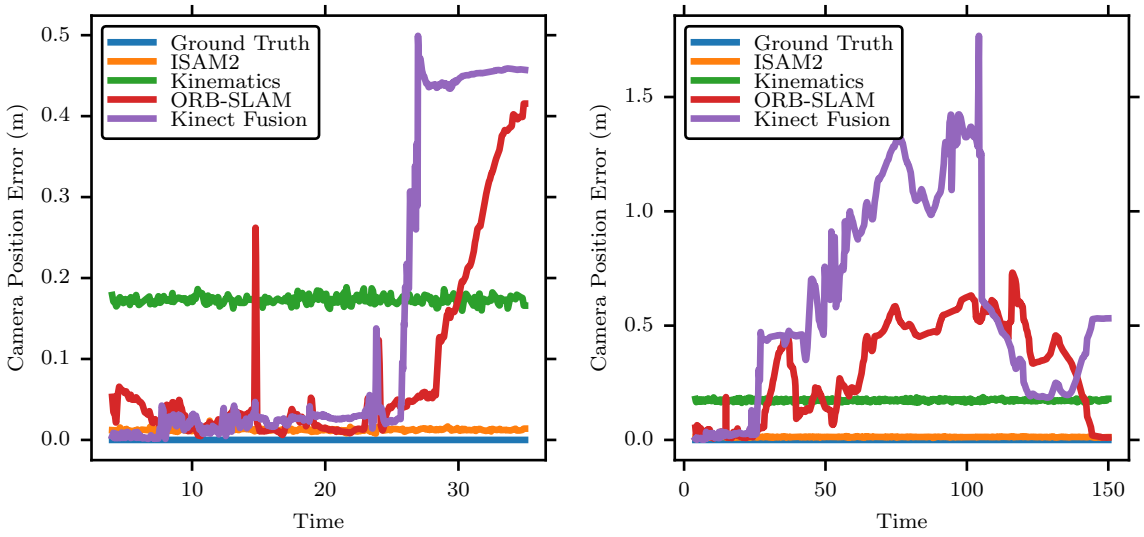


Figure 4.19: Error of the estimated camera trajectory (Fig. 4.18) vs. ground truth in a simulated dataset. The left figure shows the trajectory error for the first 35 seconds. The right figure shows the error of the same trajectory for the first 150 seconds.

this seems like a linear system much like equation 4.23; so we should be able to apply traditional observability analysis to this system of equations to find out whether our problem is observable.

When we tested the rank of $\mathbf{J}$ for 6-DOF robots, we found it to be of full rank in all but the most extreme of cases (for example, when only one landmark is observed from a very small baseline). This seems counter intuitive, because for a 6-DOF robot, there are 6 joint angles to estimate as well as the 6-DOF extrinsic calibration – twice as many parameters as can be explained by the motion of the camera. But keep in mind that we are satisfying *many* constraints on the reprojection error of *multiple* landmarks, and if enough data are collected, these constrains should be enough to estimate the state of the system.

So why does the SLAM system still sometimes fail to estimate the joint angles of the robot, if our analysis tells us that the SLAM system is observable? A recent work by Maye *et. al* [39] sheds light on this problem. According to [39],

> [...] *traditional observability analysis is helpless when it comes to noisy input data on real physical systems. It is indeed purely algebraic, performed a priori, and hence omits the numerical difficulties associated with sensory data.*

That is, even if a system is rank-deficient, it may appear to be full-rank when the data are noisy. Indeed, a Gauss-Newton like method will do even worse under these conditions, because it will fit its solution to noise in the dimensions of the system that are unobservable. By switching to a robust Cauchy estimator (see page 68), we have alleviated this problem somewhat – but the system could still be silently unobservable without us realizing it.

[39] solve this problem by instead examining the Fisher information matrix $\mathbf{J}^T\Sigma^{-1}\mathbf{J}$ in terms of its numerical rank – which counts the number of singular values of a matrix which are less than a bound ϵ depending on the scale of the noise. They apply this analysis to an online calibration problem very similar to our own, and use it to suppress updates to the system state in

directions that have been determined to be unobservable. A similar work by Zhang *et. al* [62] finds directions in which the SLAM system is degenerate and freezes them for a 6-DOF sensor platform with lasers and cameras.

Still, in our case, we have one significant advantage over the problems presented in [39] and [62] – namely, since the robot's joint encoders strongly inform its state, the system is guaranteed to be observable so long as the noise on the joint encoders is small enough. The noise in most of our experiments (including those shown in figures 4.15a and 4.16) is greatly exaggerated beyond what we would expect to see in reality to make the problem of kinematic redundancy clearer. In contrast, most SLAM problems involve an unobservable state space with a freely moving robot whose pose can drift over time.

In future work, we would like to apply the methods of [39] and [62] to our SLAM problem to not merely determine when a robot's trajectory produces unobservable motion, but to alleviate the problem of kinematic redundancy automatically. Such a system may also in principle allow a robot to decide which actions will increase the observability of its own state, and therefore would allow it to generate its own calibration trajectory autonomously.

4.7 Preliminary Work: Moving Robot Base

We have done some preliminary work on modeling the motion of the robot's base with both a 6 DOF free-floating base, and a 3 DOF holonomic planar base. Consider a noisy robot arm mounted on a moving base, with a camera mounted on the end effector. As the robot simultaneously moves its base (by wheels or some other mechanism), and joints, can we estimate the joint angles, position of the base, and the camera extrinsics simultaneously?

The key idea we can use here is that the robot's base can be considered *just another joint*, with degrees of freedom that have different measurement noise characteristics. We can use an identical factor graph (Fig. 4.2), where the first 6 or 3 degrees of freedom devoted to the 6 DOF or 3 DOF motion of the robot's base, respectively. Care must be taken to correctly model these additional degrees of freedom, since they have different topology and measurement characteristics than the robot's other joints.

4.7.1 Planar Holonomic 3-DOF Base

Consider a robot with a planar holonomic base (Fig. 4.20). The robot base has 3 degrees of freedom, $[x, y, \theta]$ which correspond to the position and orientation of the base in the plane. The base pose is in $SE(2)$, where x, y degrees of freedom have normal Euclidean linear topology, while the third degree of freedom has circular topology. The kinematic Jacobian of this base motion is thus no different than any other robot with prismatic and revolute joints.

Assume that we have no direct measurement of the base pose from encoders. The encoders factors f_θ will thus be ignored for the base. The velocity factors $f_{\dot{\theta}}$, however, will still be active, though they will likely have much more noise than the velocity factors on the other joints, depending on how the base is actuated. In addition to these factors, we need only add one more factor: a prior on the initial pose of the robot's base which grounds the trajectory to a fixed coordinate frame.

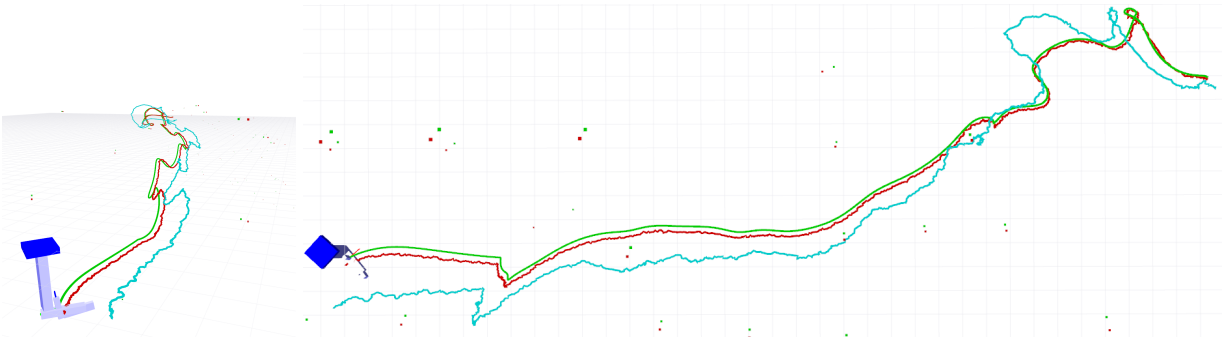


Figure 4.20: A planar holonomic 3DOF robot base (dark blue) translates and rotates in the plane. A 4-DOF robot with revolute joints (light blue) is attached to the base, and moves randomly. Landmarks (green and red dots) are observed by a simulated camera (axis on the end effector). The SLAM system reduces the error of the end effector trajectory (colors: green - ground truth, red - MLE from SLAM, cyan - noisy odometry and encoders).

The result is that absent measurements from the camera, the robot's base pose will drift over time. This is consistent with most other SLAM problems. Measurements from the SLAM system, combined with measurements from the joint encoders, should correct for this drift (which is analogous in some ways to SLAM problems using an IMU as an input).

Fig. 4.20 shows a simulated example of this kind of SLAM problem, using an auto-generated 4 DOF robot arm mounted on the base, with simulated landmarks. The velocity measurements of the base motion are simulated as the true velocity plus some zero-mean Gaussian noise. With the right parameters, it seems possible to simultaneously estimate the joint angles, camera extrinsic and base motion from the camera. However, more rigorous experiments and analysis are required to say anything more about this problem.

4.7.2 Free-floating 6-DOF Base

As in the previous section, we have a robot arm mounted to a moving base. In this case, the base is free-floating and six dimensional. The kinds of robots that this case covers include hand-held robots, flying robots, and swimming robots. We represent the base using a 4x4 homogeneous transformation matrix. Jacobians with respect to this matrix are taken in the tangent space using the exponential/log map. Like in the previous section, we assume we have some loose prior on the velocity of the base. All other parts of the system are identical to the 3DOF planar base.

Fig. 4.21 shows the SLAM system running on a simulated 4-DOF robot attached to a free-floating 6-DOF base, which is free to translate and rotate in space. Though its clear that the SLAM system manages to reduce the overall error of the robot's end effector trajectory, it doesn't do as well as in the 3-DOF case, as there is still significant drift in the base pose. Looking closely at the trajectory, we can see that in the first few timesteps the robot's base motion is incorrectly interpreted, which causes all of the subsequent timesteps and landmark locations to be wrong. We also show (bottom row Fig. 4.21) the same trajectory, but with very strong ($\Sigma = 0.01$ meter) priors on the absolute landmark locations. In this case the SLAM system is able to track the robot's position and calibrate its extrinsics because the ambiguity between the scene structure

and base motion is eliminated.

These experiments show the need for further research into the 6DOF free-floating base problem.

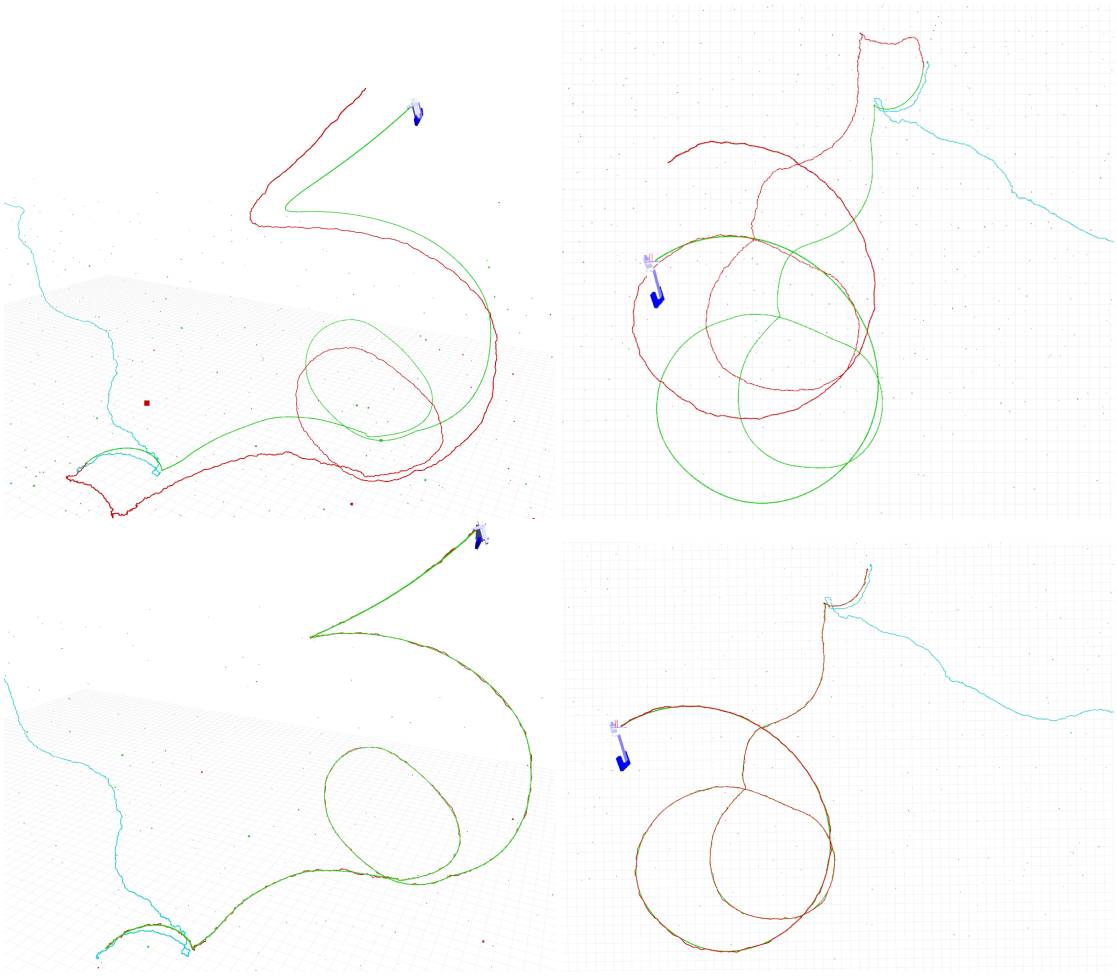


Figure 4.21: A free-floating 6-DOF (dark blue) flies freely in space. A 4-DOF robot with revolute joints (light blue) is attached to the base, and moves randomly. Landmarks (green and red dots) are observed by a simulated camera (axis on the end effector). The SLAM system reduces the error of the end effector trajectory (colors: green - ground truth, red - MLE from SLAM, cyan - noisy odometry and encoders). The top row shows the result without any prior on landmark locations. The bottom show shows the result on the same trajectory, but with strong priors on the landmark locations.

CHAPTER 5
Related Works

STATE and calibration are two of the most fundamental problems in robotics. Many of the earliest works in academic robotics have focused on estimating a robot's state or calibrating its sensors. These are generally seen as two of the main competences required for autonomous robotics. Our work falls into the category of robot tracking and calibration without fiducials or markers, using only natural image features or depth.

5.1 Hand-eye Calibration

Since at least the 1970's, there has been a wealth of research into what is called the *hand-eye calibration problem* [54]. In this problem, a camera is mounted on the end of a robot manipulator with an unknown extrinsic pose T_{cam} relative to the robot's end effector. The robot takes several measurments using the camera and attempts to compute the unknown extrinsic pose from these measurements.

In the earliest formulations of this problem, the camera measurements consist of the known 3D poses of objects relative to the camera (usually, calibration targets such as checkerboards), and the exact pose of the end effector is known with absolute certainty. When these assumptions are made, one can write down an exact global geometric solution to the problem in the form of a Sylvester equation [10] having the form:

$$AX = XB \tag{5.1}$$

where A consists of the poses of the end effector, B consists of pose measurements relative to the camera, and X is the unknown calibration T_{cam}. Once the problem is written in this form, it is usually split up into rotation and translation components which are solved seperately (since the translation component is linear, and the rotation component is not), either using a quaternion representation, or homogenous transformation matrices [54]. Since the classic formulation of the problem is not robust to noise or incorrect camera measurements, recent work [20] has reframed the problem as one of constrained global optimization that is solved iteratively.

Other works attempt to remove the requirement that camera measurements consist of known poses of objects relative to the camera. A work by Andreff *et. al* [2] solves the hand-eye calibraiton problem using structure from motion. This work is very similar to our own: it does not require fiducials for calibration, instead it does a sparse 3D reconstruction of the scene. However,

unlike our work, [2] treats the problem again as a purely geometric one relating the unknown extrinsic calibration to camera motions caused by the robot manipulator and measurements of the landmark locations. For that reason, it is not able to account for uncertainty in the joint angles, nor can it explicitly provide uncertainty estimates for all of the relevant calibration parameters. In contast, our work frames the problem as one of Bayesian state estimation, which gives us the ability to explicitly consider the uncertainty of the joint angles and other measurements.

An interesting recent work by Heller *et. al* [20] extends [2] to automatically generate the robot's trajectory that will best calibrate the camera extrinsics. We believe that this approach could also be adjusted to be used in our framework to generate high quality trajectories for calibration.

The hand-eye calibration problem is a subset of the problem we wish to solve in this work. We not only wish to find T_{cam}, but also other unknown parameters of the system (namely, the joint angles of the robot, and the motion of its base). Further, we do not want to depend on any fiducials or markers for camera measurements; instead, we wish to use the camera images of an unknown scene directly. Finally, we want to have an estimate of the *uncertainty* of each of the unknown parameters. To our knowledge, prior work on the hand-eye calibration problem has not considered the uncertainty of the extrinsic T_{cam}.

In all of our works, we get a rough initialization of the extrinsic of the camera using a classic hand-eye calibration technique from Tsai [16] and a fiducial.

5.2 Tracking Articulated Bodies

In chapter 2 we introduced an algorithm for tracking a robot arm in a dense depth image. This kind of approach is widely used in motion capture and visual servoing to track articulated bodies. In human motion capture [40], the articulated movement of a human actor's body is recorded by a computer vision system either using visual markers [46], accelerometers [50], or depth images [55].

Our work is very similar to human motion capture techniques using depth images. For example, a human tracking technique used in Microsoft's Kinect [55] first classifies human parts (head, shoulder, arm, etc.) in a depth image to produce labellings using a decision forest, and then uses these labeled parts to infer joint positions. This is analagous to our approach, which first labels pixels in the depth image based on their correspondence to the robot's links and then tracks the joint angles of the robots based on this labeling. However, in our case the labeling is far simpler because of the strong prior from the robot's joint encoders. An earlier work by Grest *et. al* [15] uses an extremely similar approach to ours to track human bodies in a depth image. By projecting a canonical human body model into the depth image, the human parts can be associated with depth pixels and then its joint angles are tracked with respect to the image. Our approach differs only in that it is incorporated into a robot control loop, and uses a strong prior from the robot's joint encoders.

Simiarly, a number of works involve tracking robot arms in depth or color images. Krainin *et. al* [34] estimate the pose of a robot manipulator as well as an object held in the hand as part of an offline modelling process using both depth and color; and incorporate this data into a Kalman filter. Hebert *et. al* [19] combine measurements from fiducials, silhouettes and depth

to track a robot arm and manipulated object with a Kalman filter. Like our work, [34] and [19] are both concerned with tracking the noisy Barrett WAM arm. Some of the most recent works in this domain (published after our work [30]) come from Schmidt *et. al* [52, 53], and concern the Dense Articulated Real Time Tracking (DART) algorithm. DART is strikingly similar to the algorithm derived in chapter 2. Like our method, it uses depth images to track noisy robot arms; but uses a much more efficient signed distance field representation of the robot's body (a representation we make use of in chapter 3) for part association, and also incorporates motion of the base and reasoning about occlusion. Lately, DART has also incorporated contact with the environment as a contraint [53].

5.3 Simultaneous Localization and Mapping (SLAM)

Simultaneous Localization and Mapping is a classic problem of robotics consisting of jointly estimating the state of the robot and the structure of the world around it. There are a wide variety of techniques for solving SLAM problems. In this work we focus on two broad categories of SLAM algorithms: "dense" techniques, which express the SLAM problem as one of computing a dense geometric map of the scene while simultaneously tracking the state relative to that map, and "sparse" techniques, which reduce the SLAM problem to one of solving a joint optimization of both the state and the map. Chapters 2 and 3 are both examples of dense techniques applied to robot arm tracking, while chapter 4 describes a sparse technique applied to the same problem.

5.3.1 Dense SLAM

The dense SLAM problem consists of estimating a detailed geometric map of the scene, as well as the state of the system over time. Methods of computing the dense geometric map and the system state vary with the kind of sensor and state space used. Early work on dense SLAM, such as DP-SLAM [13] have utilized 2D occupancy grid map representations of the dense map, and have variously used Particle Filters or the Extended Kalman Filter (EKF) for localization. These methods are sufficient for 2D maps with poses in $SE(2)$, but suffer from drift over large scales, and because they are so memory and computationally intensive, they fail to scale to 3D maps with higher dimensional state spaces.

More recent work in dense SLAM has relied on advances in GPU computing and depth sensing technology. Dense Tracking and Mapping (DTAM) [43] was one of the first fully-dense 3D SLAM systems. DTAM tracks the pose of a handheld monocular camera using direct image alignment to a keyframe for which it keeps a dense 3D map – but DTAM is limited to very small scenes and can quickly lose track of a scene if the camera moves too quickly. Kinect Fusion [24] was one of the first dense SLAM systems to use a depth camera and high-performance GPU computing to track a handheld camera in real-time. Unlike DTAM, Kinect Fusion creates a globally metric 3D map consisting of a Truncated Signed Distance Field (TSDF) [8], and it directly aligns depth images to this global map using the Iterative Closest Point [4] algorithm. While creating very detailed, photorealistic reconstructions and real-time camera tracking, Kinect Fusion suffers from drift over long trajectories and has enourmous memory requirements for dense map storage.

Much of the latest research in dense visual SLAM is focused on making up for the shortcomings of Kinect Fusion and its variants. Kintinuous [60] and Chisel [31] are systems based on Kinect Fusion which attempt to increase the scale of the 3D reconstruction by more careful management of memory. ElasticFusion [61] incorporates some ideas from sparse SLAM systems to correct misalignment and drift in the dense SLAM system by means of a dense deformation graph.

The algorithm shown in chapter 3 is a dense SLAM approach very much in the same vein as Kinect Fusion. It has a TSDF map representation, and estimates the pose of the sensor through dense geometric alignment.

5.3.2 Sparse SLAM

In contrast to dense SLAM, sparse SLAM techniques do not attempt to maintain a detailed geometric map of the scene, instead they focus on accurate estimation of the system state trajectory from a sparse set of landmarks. This allows sparse SLAM techniques to jointly optimize the positions of landmarks and the trajectory, which prevents drift and ensures consistency when the same scene is observed multiple times.

Some of the oldest SLAM techniques are sparse systems utilizing landmarks. Some of the earliest works on SLAM, including the approach from Smith and Cheeseman [56] treated the SLAM problem as a classical filtering problem over landmark positions and robot trajectory, which were estimated incrementally using the extended kalman filter. Fast-SLAM [41] treats the problem as a joint optimization of landmark locations and the robot trajectory, optimizing the trajectory using a particle filter and the landmark locations using the extended Kalman filter. Early SLAM techniques of this kind focused on 2D maps with robots in $SE(2)$. Such filtering approaches typically only try to estimate the landmark locations and position of the robot at the most recent time, rather than the full robot trajectory.

Parallel Tracking and Mapping [29] and MonoSLAM [9] are some early examples of sparse SLAM systems that use natural image features to track hand-held 3D camera motion. MonoSLAM tracks a small set of features by filtering the landmark locations and camera trajectory in an EKF simultaneously, while PTAM tracks a very large number of features using incremental bundle adjustment over a set of sparse keyframes.

Approaches relying on filtering of any kind (and in particular the EKF) for SLAM have come under scrutiny for being inconsistent with very simple counterexamples, such as stationary vehicles repeatedly measuring the locations of the same landmarks again and again, or a robot making a loop repeatedly [25]. Further, methods relying on filtering have been shown to scale poorly to large problems.

These concerns led to a number of approaches that attempt to directly solve what is called the *full SLAM problem*, that is, treating the state of the system to be the full trajectory of the robot over time rather than merely at the most recent time. These works typically treat the SLAM problem as an optimization of the Bayesian *factor graph* describing correlations between all poses of the robot and measurements, and unlike filtering approaches do not make the *Markov assumption* that the current state of the robot is conditionally independent on its previous state. The seminal work of this class of algorithms, Square Root Smoothing and Mapping [12] showed that by using sparse linear algebra over the full factor graph, the SLAM system could actually

be made more efficient than filtering approaches while also considering many more landmarks, longer trajectories, and loops. Further, this kind of approach does not require linearizing the state of the system or measurements, making it more amenable to nonlinear state spaces and measurment models. These techniques are sometimes called *smoothing* approaches since they distribute ("smooth") error over the entire trajectory.

More recent works have focused on improving the performance and scalability of the smoothing technique. Incremental Smoothing and Mapping (iSAM2) [26] is the method used as a back-end for chapter 4, and improves the efficiency of the smoothing technique in the online setting by storing a *Bayes tree*, which compresses the factor graph into conditionally independent cliques arranged in a tree. iSAM2 is able to efficiently update the maximum likelihood estimate of the trajectory and landmarks by only updating the relevant parts of the Bayes tree as new data arrives.

One state-of the art SLAM system utilizing both natural image features and a nonlinear-optimization smoothing SLAM approach comes from Leutenegger *et. al* [35] and fuses natural image data with high-frequency inertial measurements. The algorithm in chapter 4 is heavily inspired by this approach, since we use the same image features (BRISK), and also combine low-frequency measurements from a camera system with high-frequency measurements (in our case, from joint encoders rather than an inertial unit).

5.4 Calibrating with SLAM

There have been a number of works which aim to calibrate robots using SLAM. As we have written in the previous chapters, calibrating with SLAM is a desirable goal because it allows the robot to self-calibrate in real-time without relying on highly controlled environments, fiducials, or human measurements.

Of particular interest to computer vision researchers is the automatic calibration of camera intrinsic parameters (f_x, f_y, c_x, c_y etc.) from SLAM. In the SLAM framework, the camera intrinsics can easily be included as an additional factor in the joint probability density of the system. Ozog *et. al* [47] present a survey on methods of calibrating camera intrinsics using SLAM, and point out some difficulties that arise from incorrectly modelling the intrinsics. Teichmen *et. al* [59] calibrate the intrinsic parameters of a depth camera using a SLAM system. Keivian *et. al* [27] actively control a robot to best inform its calibration parameters. In chapter 4 we calibrate the intrinsics using a seperate process – but there is no reason we could not also include them as factors to be jointly optimized by the system.

There are also a number of works which attempt to estimate the extrinsic transformations between two or more cameras using SLAM. This is of particular interest for stereo reconstruction and mobile robotics. Carerra *et. al* [7] calibrate a multi-camera vehicle by co-registering the trajectories of each monocular camera as estimated by a SLAM system. Basso *et. al* [3] calibrate the extrinsic transform between a monocular and depth camera using a SLAM system. Heng *et. al* [22] calibrate the extrinsics of a multi-camera flying vehicle system using SLAM. In another work, Heng *et. al* [21] calibrate a multi-camera ground vehicle using a two-pass SLAM system. In chapter 4, we similarly calibrate the extrinsics of a camera with respect to the robot's wrist using SLAM; but to our knowledge we are the first to apply such a system to a robot with a hand-mounted camera.

A few works calibrate robot arms using SLAM-like techniques. One interesting work from Roncone *et. al* [51] calibrates the full Denavit Hartenburg parameter table of a robot using multiple self-touch constraints – an approach which is analagous to SLAM for the robot's own body. Pradeep *et. al* [49] calibrate a robot arm using a fiducial held in the robot's hand and a global bundle adjustment technique. They optimize both the camera extrinsics and joint angles of the robot. Interestingly, they speculate about a future system which might be able to omit the fiducial and calibrate using natural image features. Birbach *et. al* [5] similarly use bundle adjustment on fiducials attached to the robot's body to calibrate joint offsets and camera extrinsics. The work we present in chapter 4 is very similar to [49]; though instead of using global bundle adjustment, we use an incremental SLAM system, and we use natural image features rather than a single fiducial. Using natural image features allows us to calibrate the robot anywhere at any time, and take in much more data for calibration.

Maye *et. al* [39] provide a detailed analysis of automatic robot self calibration, using mobile robot SLAM as an example. They note that as the number of degrees of freedom of the system increases, care must be taken to create trajectories which inform the SLAM system so that the robot's state is observable. Their method alleviates this problem by only updating the maximum likelihood estimate in directions that are likely to be observable. This is a very interesting approach that is worth trying in our domain.

CHAPTER 6
Conclusion

In this work, we showed how common problems of tracking and calibrating robot arms can be expressed as SLAM problems; and presented three ways of tracking the state of a robot arm and calibrating it. First, chapter 2 showed how a robot arm could be tracked in real time using a depth image. Chapter 3 showed how the same algorithm could be inverted and combined with a dense mapping technique to simultaneously track a robot arm and map a scene in real time using a depth image. Finally, chapter 4 presented a method of simultaneously track a robot arm, calibrate its camera extrinsics, and map a scene using a monocular camera image and graphical SLAM.

We can track and calibrate even very noisy robot arms to enable them to grasp and touch small objects, or create 3D reconstructions of scenes in real-time; without fiducials or other special calibration "tricks". Our methods succeed in 2D and 3D simulations, and with real robot data. Chapter 4 in particular is highly generalizable to robots with a wide variety of sensors and noise models.

However its clear that the methods presented here have limitations that should be addressed in future works.

All of our methods assume a static scene. This limits them to simple scenarios where the robot scans a static scene from a fixed viewpoint. The dense SLAM technique shown in chapter 3 incorrectly interprets moving objects as being part of the scene's static geometry, while the sparse SLAM technique in chapter 4 rejects moving objects altogether as outliers.

Addressing this limitation would involve explicitly modeling the motion of dynamic objects in the scene, and motion of the robot's base (as in section 4.7.2). Unfortunately, this increases the degrees of freedom of the system by six per dynamic rigid object, and runs the risk of producing systems that are unobservable. Our preliminary experiments have indicated that successfully modeling the motion of the robot's base simultaneously with the motion of its arm might be feasible; but more analysis is required to implement such a system on a real robot.

Another key issue is that we have not adequately modeled the actuator noise (treating it as static Gaussian noise on the joint encoders in chapters 2 and 3, and as bandwidth-limited Gaussian noise in chapter 4). More systematic analysis of the physical structure of the robot is necessary to determine how the unmodeled dynamics affect its joint angle measurements over time. It might also be possible to regression techniques (such as Gaussian Process Regression) to learn more complex noise models for

Finally, the problem of kinematic redundancy (Fig. 4.16) needs to be addressed – if the

joint angles aren't directly observable, and have redundancies how can we prevent a SLAM or calibration system from predicting incorrect joint angles? In section 4.6, we hinted that a method from [39] that prevents the SLAM system from updating its maximum likelihood estimate along unobservable dimensions might be used to prevent this problem.

CHAPTER A

Appendix and Background

A.1 Geometry of 3D Space

The world is 3-dimensional. The dimensions (labeled x, y, and z) form an orthogonal linear basis of the world. The 3D world is sometimes referred to as the "work space", or as "task space." We represent it using the Euclidean space $\mathbb{R}^3$.

A *rigid body* $\mathbf{B}$ is a continuous set of (possibly moving) 3D points in $\mathbb{R}^3$ such that the distance between any two points $\mathbf{p}_1, \mathbf{p}_2 \in \mathbf{B}$ is always constant. The position and orientation of a rigid body in 3D space can be explained by 6 degrees of freedom (3 for translation, and 3 for rotation). The set of all such positions and orientations is called the *Special Euclidean Group* $SE(3)$. A member of this groups is referred to as a *transformation*. A transformation can be used to describe the relationship between two *reference frames* in 3D space. A reference frame is the origin of a coordinate system.

In this work we represent transformations mainly as *homogeneous transformation matrices* which have the form

$$T \in \mathbb{R}^{4\times 4} = \begin{bmatrix} & & & | \\ & \mathbf{R}_{3\times 3} & & \mathbf{t} \\ & & & | \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.1}$$

where $\mathbf{t} \in \mathbb{R}^3$ is the translation, and $R \in \mathbb{R}^{3\times 3}$ is the orthogonal rotation matrix whose columns represent the rigid bases of the reference frame implied by the transformation matrix.

The transformation matrix can be used to multiply homogeneous vectors in $\mathbb{R}^3$, transforming them into their reference frame. For example,

$$\mathbf{p}_b = T_b^a \mathbf{p}_a \tag{A.2}$$

the transformation matrix T_b^a transforms points in reference frame a to reference frame b when pre-multiplying points in frame a, where

$$\mathbf{p} = [x, y, z, 1]^T \tag{A.3}$$

is a 3D point expressed in the so-called homogeneous form.

Transformations can be concatenated by multiplying their homogeneous matrices together. They also have a well-defined inverse.

A.1.1 Differential Geometry of 3D Space

It is sometimes necessary to describe velocities or small, incremental motions in 3D space. For points and vectors, we can do this using straightforward calculus in the Euclidean space of $\mathbb{R}^3$. But what about for transformations in $SE(3)$?

Rigid transformations complicate the differential geometry of 3D space by conflating together the rotation and translation of *sets* of 3D points which must obey a pairwise distance constraint. Because of this, there is no straightforward mapping from $SE(3)$ to R^6. Rather, the transformations of $SE(3)$ exist in a *constraint manifold* that is embedded in R^6. The study of differential geometry on this constraint manifold is called *tangent space* geometry and is accomplished through the Lie Algebra. Here, we provide a brief explanation of the Lie Algebra of motions in $SE(3)$ transcribed from an excellent tutorial by Blanco [6].

Motions in $SE(3)$ can be expressed by their tangent-space forms using what are called the exponential and logarithm maps.

The Exponential Map

The exponential map takes a 6-DOF motion

$$\mathbf{v} = \begin{bmatrix} \mathbf{t} \\ \omega \end{bmatrix} \tag{A.4}$$

(where $\mathbf{t}$ is the translational motion, and ω is the rotational motion expressed as a differential rotation around x, y, and z) and converts this motion to a rotation matrix of the form

$$\exp \mathbf{v} = \begin{bmatrix} e^{[\omega]_\times} & \mathbf{V}\mathbf{t} \\ 0 & 1 \end{bmatrix} \tag{A.5}$$

and

$$\mathbf{V} = \mathbf{I}_{3\times 3} + \frac{1-\cos\theta}{\theta^2}[\omega]_\times + \frac{\theta - \sin\theta}{\theta^3}[\omega]_\times^2 \tag{A.6}$$

where $\theta = \|\omega\|$, and $[\omega]_\times$ is the skew symmetric matrix of the form:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_\times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \tag{A.7}$$

The operator $[\bullet]_\times$ also has an exponential map:

$$\exp[\omega]_\times = \mathbf{I}_{3\times 3} + \frac{\sin\theta}{\theta}[\omega]_\times + \frac{1-\cos\theta}{\theta^2}[\omega]_\times^2 \tag{A.8}$$

as well as an implicit inverse $[\bullet]_\nabla$ using the exponential map, we can convert motions in $SE(3)$ into homogeneous transformation matrices.

The Logarithm Map

The logarithm map is the inverse of the exponential map. We can take a motion expressed as a homogeneous transformation matrix and re-express it as a 6-vector in the form of equation A.4 using the inverse of each component of the exponential map:

$$\mathbf{v} = \begin{bmatrix} \mathbf{t}' \\ \omega \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{A.9}$$

$$\omega = [\ln \mathbf{R}_{3\times 3}]_\nabla \tag{A.10}$$

$$\mathbf{t}' = \mathbf{V}^{-1}\mathbf{t} \tag{A.11}$$

The symbols used here mirror those used in the exponential map.

A.2 Articulated Linkages and Robot Arms

An articulated linkage is a series of rigid bodies (called *links*) connected by joints. Each joint has one or more degrees of freedom, and two associated links A and B. A degree of freedom maps a value $q \in \mathbb{R}$ to a transformation $T_B^A(q) \in SE(3)$, which encodes the rigid transformation between links A and B. A degree of freedom may correspond to a rotation, translation, or both. A degree of freedom may have limits $q_{\text{min}}, q_{\text{max}}$, that constrain its motion.

Robot arms are a special case of the kinematic linkage. Usually, robot arms are *serial* linkages, meaning that the graph formed by links and joints is a tree, with the root fixed to the world, and allow for sensing and control of the degrees of freedom.

The set of degree of freedom values $\mathbf{q} = [q_1, \ldots, q_N]^T$ is called the *configuration* of the linkage, and is represented as a vector. The set of all possible configurations of the linkage $\mathbf{q} \in \mathbf{Q}$ is called the *configuration space* [38].

A.2.1 Forward and Inverse Kinematics

The configuration space of a linkage is related to its *workspace* $\mathbb{R}^3$ by its *forward* and *inverse* kinematics. Using forward kinematics, any point on the robot's body $\mathbf{x}$ can be transformed into the workspace by the function $F_\mathbf{x}(\mathbf{q}) : \mathbf{Q} \to \mathbb{R}^3$. Forward kinematics encode a many-to-one relationship between a configuration and the points on the robot's body, and always has a solution. The forward kinematics can be computed merely be concatenating the link transformations together $F_\mathbf{x}(\mathbf{q}) = T_N^{N-1}(q_N) \ldots T_3^2(q_2) T_2^1(q_1)$

The inverse of the kinematics function can also be defined, which for a point on the robot's body $\mathbf{x}$, returns the configuration which puts $\mathbf{x}$ at the desired location; $F_\mathbf{x}^{-1}(\mathbf{x}) : \mathbb{R}^3 \to Q$. This is in general a one-to-many function, and may have no solution.

A.2.2 The Robot Jacobian

The first-order derivatives of the kinematics functions can also be computed. The first-order derivative of the forward kinematics function $\mathbf{J}_\mathbf{x} = \frac{\partial}{\partial \mathbf{q}} F_\mathbf{x} \in \mathbb{R}^{3\times N}$ is called the *Jacobian* of the linkage. The inverse of the Jacobian $\mathbf{J}_\mathbf{x}^{-1}$ can also be defined. In general, the inverse of the Jacobian $\mathbf{J}_\mathbf{x}^{-1}$ describes the joint velocities required to move $\mathbf{x}$ at a specified linear velocity. If the robot has more or fewer degrees of freedom than 3, the Moore-Penrose pseudo-inverse of the Jacobian $\mathbf{J}_\mathbf{x}^{+}$ can be used instead. Through the *principle of virtual work*, the transpose of the Jacobian $\mathbf{J}_\mathbf{x}^{T}$ can be used to describe the instantaneous torques required to apply a linear force at $\mathbf{x}$.

In general the Jacobian is computed as the partial derivative of the forward kinematics function with respect to the robot's joints. For some robots (specifically, serial link robot manipulators with only rotational joints) the linear Jacobian can be computed easily through a series of N cross products [38]:

$$\mathbf{J}_\mathbf{x} = \begin{bmatrix} | & \dots & | & \dots \\ z_0 \times (\mathbf{x} - O_0) & \dots & z_{i-1} \times (\mathbf{x} - O_{i-1}) & \dots \\ | & \dots & | & \dots \end{bmatrix} \tag{A.12}$$

where $\mathbf{x}$ is the point of interest, z_i is the axis of rotation of the i^{th} joint, and O_i is the origin of the i^{th} joint. Each column represents the contribution of joint i to the linear velocity of $\mathbf{x}$.

A.2.3 Topology of the Configuration Space

The configuration space, $\mathbf{Q}$, is the set of all possible configurations of the robot. Depending on how the robot is structured, the configuration space may have linear, spherical, or toroidal topology. For a serial manipulator robot, the configuration space of the robot is the cross product of the configuration spaces of each joint.

If the robot is made entirely of rotary joints with positive and negative limits within $\pm\pi$, or prismatic joints with or without limits, then the topology of the configuration space is merely $\mathbb{R}^N$, where N is the dimension of the configuration space. However, if the robot has rotary joints which can "wrap around" more than once, then the joint angles are equivalent modulo 2π. That is, the joint angle $q(i) \equiv q(i) + 2k\pi \ \forall k \in \mathbb{Z}$. In this case, the topology of the configuration space is the cross product of multiple $SO(2)$ (circular) state spaces. If all of the joints have this property, then the configuration space is a hyper-torus (Fig. A.1). Still other more exotic joints have more complicated topology. A ball-and-socket joint, for instance, has $SO(3)$ topology. A robot with all ball-and-socket joints has hyper-spheroid topology.

The topology of the configuration space complicates some of the common operations (such as taking derivatives, differences, and interpolations in the configuration space) used in this paper. For clarity, we write these common operations as though the configuration space were $\mathbb{R}^N$; but in practice the topology of the configuration space has to be taken into account. Here is how some of the common operations are implemented:

Differences: the absolute difference between configurations $\mathbf{q}_1$ and $\mathbf{q}_2$ is the length of the minimum arc between them in configuration space. For instance, in $SO(3)$ it is the great circle distance between two points on the spherical configuration space. On a torus, it is the

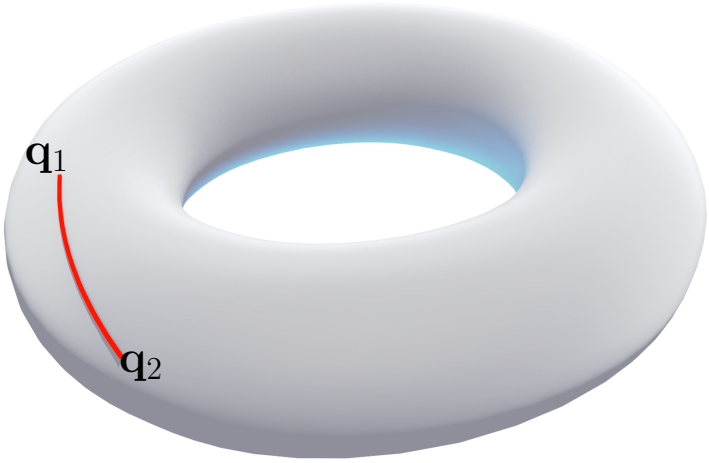


Figure A.1: A two-dimensional toroidal configuration space embedded in three dimensions. The minimum length arc between $\mathbf{q}_1$ and $\mathbf{q}_2$ is shown as a red line.

toroidal arc between the two configurations. For more complex configuration spaces, it is the sum of all the minimum arcs of the configuration spaces of each degree of freedom.

Interpolation: Interpolating between $\mathbf{q}_1$ and $\mathbf{q}_2$ consists of parameterizing the minimum arc by path length from $\mathbf{q}_1$ to $\mathbf{q}_2$. In $SO(3)$ or $SO(2)$, it is the *slerp* operation.

Derivatives: Derivatives in the configuration space are meant to be taken along the *tangent space* of the robot's configuration. Closed form tangent space transformations are available for all joint types.

At all stages of SLAM or tracking, we take care to use the proper differences, interpolations, and derivatives for the robot's configuration space.

A.2.4 Encoders

The degrees of freedom of a robot can be measured by sensors called encoders. These include for example optical, resistive, current, and mechanical sensors. In general the output of an encoder θ can be used to estimate the value of its associated degree of freedom q_θ through an *encoder calibration* function $q_\theta = K_e(\theta)$. For some encoders, the calibration function can be represented as a simple linear transformation. For other encoders, the calibration function must be more complex.

Some robots have dynamic mechanisms (such as springs, gear trains or cables) between their encoders and associated degrees of freedom. These degrees of freedom may complicate the calibration function, adding dynamic factors such as the velocity and torque of the system.

Gear Backlash

One particular dynamic effect of interest is *gear backlash*, which occurs whenever a series of gears sits between the encoder and the robot's joint. This happens because tiny gaps between the gear teeth prevent the gears from being in contact all the time. Consequently, the joint or motor

may rotate slightly without actually turning the gears. In principle, this limits the sensitivity of the joint encoder to small motions.

A.3 Data Synchronization and Interpolation

In all of the SLAM, calibration and tracking applications discussed in this work, we assume that all data from the robot (images, joint encoders, etc.) is time-synchronized. In practice, the cameras we use refresh at between 30 and 60 Hz, while the robot joint angles are sampled between 100 and 1000 Hz. Both the camera images and the joint encoders are time-stamped. To time-synchronize the camera images and joint encoders, we use linear interpolation of the joint encoders sampled by time, making use of the fact that joint encoders are sampled at a much higher rate than the camera images.

For example, suppose an image I_t is sampled at time $t = 1.5$ seconds, and we have joint encoders sampled at $100Hz$. The nearest two joint encoder measurements are at most 0.01 seconds away. Call these $\mathbf{q}_{t_{-\Delta}}$ and $\mathbf{q}_{t_{+\Delta}}$, where Δ is the joint angle sampling rate, and t falls between $t - \Delta$ and $t + \Delta$. We construct the time-synchronized joint encoder measurement $\mathbf{q}_t$ by

$$\mathbf{q}_t = \frac{t - t_{+\Delta}}{t_{+\Delta} - t_{-\Delta}} \mathbf{q}_{t_{+\Delta}} + \left(1.0 - \frac{t - t_{+\Delta}}{t_{+\Delta} - t_{-\Delta}}\right) \mathbf{q}_{t_{-\Delta}} \tag{A.13}$$

that is, the configuration is the linear interpolation of the two nearest joint encoder measurements. When interpolating, care must be taken to use the correct topological space. If the joints have limits, then simply linearly interpolating the joint angles over time works. However, if the joints are unlimited and spin freely, their angles have to be interpolated on $SO(2)$.

A.4 Geometric Computer Vision

A.4.1 Pinhole Cameras

Optics describes the relation between geometric points in the world and their corresponding 2D image projections [18]. The simplest kind of optical system is a pinhole (*i.e.*lens-less) camera. A simple pinhole camera consists of a rigid transformation $T_{\text{cam}} \in SE(3)$, and a *intrinsic matrix*

$$K = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.14}$$

where f_x, f_y are the focal lengths, and c_x, c_y are the image center. The intrinsic matrix can be used to turn pixel coordinates $\mathbf{p} \in \Omega$ into homogeneous intrinsic coordinates $\mathbf{p}'$ by the equation:

$$\mathbf{p}' = K \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} \tag{A.15}$$

the homogeneous intrinsic coordinates p' at depth z can be converted into a point in the world $\mathbf{x}$ using the projection function

$$\mathbf{x} = \pi\left(\mathbf{p}', z\right) = T_{\text{cam}} \begin{bmatrix} \mathbf{p}'_x z \\ \mathbf{p}'_y z \\ z \end{bmatrix} \tag{A.16}$$

A.4.2 Stereo Triangulation

When a 3D point $\mathbf{x}$ is observed by two cameras I_1 and I_2 with known pose, it is sometimes possible to extract the 3D point from its two 2D measurements $\mathbf{p}_1, \mathbf{p}_2$. This process (called *stereo triangulation*) relies on intersecting the two rays from the optical axes of each camera that pass through the image plane at each measurement. In practice, if there is any error in the image, or the poses of the cameras are uncertain, then the rays may not intersect. In this case, the problem can be framed as an optimization of:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \left(\|\pi_1(\mathbf{x}) - \mathbf{p}_1\| + \|\pi_2(\mathbf{x}) - \mathbf{p}_2\| \right) \tag{A.17}$$

where π_1 and π_2 are the image projections onto camera 1 and 2. There are many methods used in the geometric computer vision community for optimizing equation A.17. The method we use is one which optimizes this equation using a fast nonlinear approximation, provided by the OpenGV [33] library.

If the pose estimate is completely unknown, or has significant error, simply triangulating the points by minimizing A.17 can result in very bad estimates for $\mathbf{x}^*$. In this case, it helps to also optimize the pose estimate before triangulating points. We do this using the 5-point Stewenius [58] bundle adjustment algorithm with Random Sample Consensus (RANSAC) to first find the pose between I_1 and I_2 before triangulating points, rejecting outliers. The Stwenius algorithm solves a set of linear equations which constrain the camera's pose estimate to minimize reprojection error between the cameras. By using RANSAC as well, the pose estimate is robust to outliers.

A.4.3 RGB-D Cameras

RGB-D cameras (or *depth cameras*) are sensors which provide both color images and depth images. Projective depth cameras, such as the Microsoft *Kinect*, Asus *Xtion* or Occipital *Structure* sensor work by projecting an infrared pattern onto the scene. By measuring the deformation of the pattern as viewed by an infrared camera, an estimate for depth can be achieved with stereo triangulation. This depth estimate is limited by the stereo baseline of the infrared camera and projector, and usually falls between 0.4 and 3 meters.

The *point cloud* of a depth image is defined as all of the valid pixels in the depth image unprojected into the scene using the depth camera's intrinsics.

A.5 Factor Graphs

One key idea used in this thesis is the factor graph [1] abstraction for Bayesian estimation. For example, if some joint probability density function can be broken down into factors

$$\mathrm{P}(a, b, c) = f_1(a) f_2(a, b) f_3(a, c) f_4(b, c) \tag{A.18}$$

then we can describe the joint probability density of a, b, c as a graph with nodes representing the values of a, b and c and links representing the factors f_1, f_2, f_3 and f_4. Figure A.2 shows such a graph.

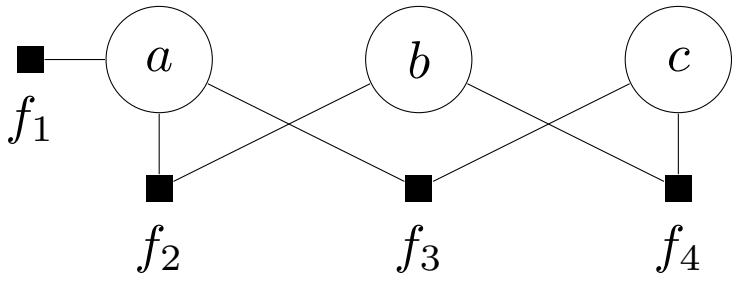


Figure A.2: An example of a factor graph for the equation A.18

The structure of the graph can be exploited to efficiently estimate the joint probability density of latent variables in the graph, conditional densities, and other properties of the system. For example, cliques in the graph represent groups of variables that are conditionally independent given the other variables, and such cliques can be estimated independently. iSAM2 [26], the algorithm we use primarily for graphical SLAM in this work, exploits the structure of the problem to convert the factor graph first into a *chordal Bayes Net* (a related graphical model containing only the variable nodes and directed edges representing joint probability distributions), which is then converted into a *Bayes tree* made up of the conditionally independent cliques of the graph. In a Bayes tree, each each node represents a conditional probability density, and is conditionally independent of its sibling nodes given its parent. As new factors are added to the graph, small modifications are made to the Bayes tree so that only a small subset of the unknown variables in the graph (*i.e* the factors and nodes corresponding to the affected cliques) have to be re-estimated. This is accomplished using message passing.

A.6 Robust m-Estimators

A common kind of optimization problem (called the least-squares problem) involves minimizing the sum of squares of a set of error terms

$$\sum_i c_i(\mathbf{x})^2 \tag{A.19}$$

where c_i are error terms and $\mathbf{x}$ is a parameter vector we wish to estimate. For example, if $\mathbf{x}$ represents the parameters of a model that is to be fitted to data, we can estimate the optimal parameters which minimize the squared residuals $c_i(\mathbf{x}) = r_i$ that encode the difference between the model and measured data.

If there is noise in the data, outliers can corrupt the result produced by least-squares. This is because as the residual r_i increases in size, its influence over the cost of the least-squares system

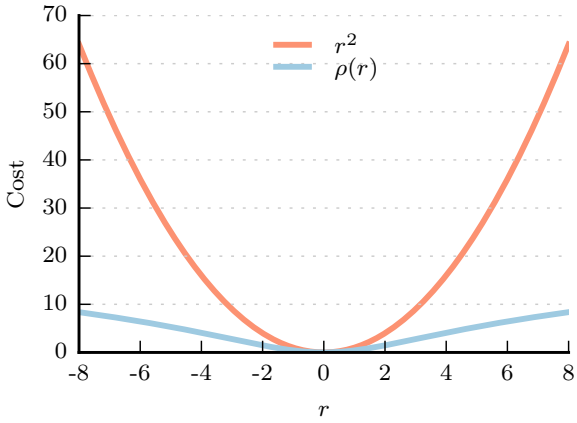


Figure A.3: Comparison of the squared cost vs. the Cauchy robust cost with $w_c = 3$

is very high in comparison to the inliers. To correct this problem, we can introduce a robust cost function [63] to replace the squared cost terms that rejects outliers.

$$\sum_i \rho(r_i) \tag{A.20}$$

where ρ is a robust cost function. The cost function (called an *m-estimator*) should be positive definite and have the property that near 0, its value should be smooth and rapidly increasing, but that far away from 0, its value doesn't grow. In this way, the outlier data doesn't greatly affect the cost.

The robust cost function we have chosen to use in this work is the Cauchy robust m-estimator. It has the form:

$$\rho(r) = \frac{{w_c}^2}{2} \log\left(1 + \left(\frac{r}{w_c}\right)^2\right) \tag{A.21}$$

where w_c is the single parameter of the cost function that affects its rate of increase. Figure A.3 shows how this function behaves with increasing residuals.

A.7 2D Image Features and Association

Sparse visual SLAM systems require a set of unique 3D *landmarks* to be observed multiple times in different camera images. However, the real world is made up of continuous and possibly moving surfaces with shadows, specular highlights, and changing appearances dependent on viewpoint and occlusion. Sparse SLAM systems therefore need a robust *front-end* for converting natural images into a set of sparse landmarks, and a way of associating points with similar visual appearances. Early SLAM systems have depended on simple features in the image such as colored blobs, lines, or corners. Others rely on man-made fiducials and tags to associate landmarks.

These days, most SLAM systems rely on *natural image features*, which convert small patches of pixel intensities into descriptors which are invariant to small changes in lighting, scale, and

orientation. These features can be extracted from natural images and then matched to existing features to produce landmarks.

A.7.1 BRISK Image Features

The algorithm in chapter 4 makes use of the Binary Robust Invariant Scalable Keypoint (BRISK)[36] features. These features aim to be consistent under changes to image rotation and scale, and are stored in a 48 to 512 bit binary image descriptor. The descriptor consists of a set of image intensity comparisons in an image patch computed for a given scale of the image. Pixel intensity comparisons are more robust to lighting changes than raw pixel intensities. Multiple scales of the image are sampled to produce features of different sizes. BRISK features are aligned at sub-pixel and sub-scale locations by interpolation, and their orientation is computed using the principal moments of the pixel intensities of the patch.

A.7.2 Landmark Association

Once natural image features are extracted from images, they must be associated with existing landmarks. In this work we use two metrics for matching landmarks to images: appearance-based and geometric. First, we attempt to project all existing landmarks onto the camera image. If the landmark cannot be projected onto the camera image (either it is out of the field of view or behind the camera), it is rejected from consideration. Landmarks are associated with their last observed BRISK feature descriptor. A brute-force comparison is made between all of the existing landmark's descriptors and the new descriptors from the image.

A new descriptor is matched to an existing landmark if the following conditions hold:

- The Hamming distance $H(d_{\text{old}}, d_{\text{new}})$ between the existing descriptor and the new descriptor is less than some threshold t_{hamming}.
- The new descriptor d_{new} is the one in the image with the lowest Hamming distance to the old descriptor d_{old}.
- The old descriptor d_{old} corresponds to the landmark with the lowest Hamming distance to the new descriptor d_{new}.
- The reprojection of the existing landmark onto the camera image is no more than t_{pixels} away from the center of the new landmark.

The Hamming distance is the number of point-wise edits required to transform one descriptor into another. These conditions are intentionally quite strict to limit the number of falsely matched landmarks. This results in several duplicated landmarks, and many rejected image keypoints in each frame.

Bibliography

[1] Hans andrea Loeliger. An introduction to factor graphs. In *IEEE Signal Processing Magazine*, pages 28–41, 2004. A.5

[2] Nicolas Andreff, Radu Horaud, and Bernard Espiau. Robot hand-eye calibration using structure-from-motion. *The International Journal of Robotics Research*, 20(3):228–248, 2001. 5.1

[3] Filippo Basso, Alberto Pretto, and Emanuele Menegatti. Unsupervised intrinsic and extrinsic calibration of a camera-depth sensor couple. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6244–6249. IEEE, 2014. 5.4

[4] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, February 1992. ISSN 0162-8828. doi: 10.1109/34.121791. URL http://dx.doi.org/10.1109/34.121791. 5.3.1

[5] O. Birbach, B. Buml, and U. Frese. Automatic and self-contained calibration of a multisensorial humanoid's upper body. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3103–3108, May 2012. doi: 10.1109/ICRA.2012.6225004. 5.4

[6] Jose-Luis Blanco. A tutorial on se (3) transformation parameterizations and on-manifold optimization. A.1.1

[7] Gerardo Carrera, Adrien Angeli, and Andrew J Davison. Slam-based automatic extrinsic calibration of a multi-camera rig. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2652–2659. IEEE, 2011. 5.4

[8] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *ACM Special Interest Group on Graphics and Interactive Techniques*, pages 303–312, 1996. 3.1.1, 3.1.1, 5.3.1

[9] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(6):1052–1067, June 2007. ISSN 0162-8828. doi: 10.1109/TPAMI.2007.1049. URL http://dx.doi.org/10.1109/TPAMI.2007.1049. 5.3.2

[10] A.S. Deif, N.P. Seif, and S.A. Hussein. Sylvester's equation: accuracy and computational stability. *Journal of Computational and Applied Mathematics*, 61(1):1 – 11, 1995. ISSN 0377-0427. doi: http://dx.doi.org/10.1016/0377-0427(94)00053-4. URL http://www.sciencedirect.com/science/article/pii/0377042794000534. 5.1

[11] Frank Dellaert. Factor graphs and gtsam: A hands-on introduction. Technical Report GT-RIM-CP&R-2012-002, GT RIM, Sept 2012. URL https://research.cc.gatech.edu/borg/sites/edu.borg/files/downloads/gtsam.pdf. 4.2.1, 4.3

[12] Frank Dellaert and Michael Kaess. Square root sam: Simultaneous localization and mapping via square root information smoothing. *International Journal of Robotics Reasearch*, 25:2006, 2006. 5.3.2

[13] Austin Eliazar and Ronald Parr. Dp-slam: Fast, robust simultaneous localization and mapping without predetermined landmarks. In *in Proc. 18th Int. Joint Conf. on Artificial Intelligence (IJCAI-03*, pages 1135–1142. Morgan Kaufmann, 2003. 5.3.1

[14] Daniel Girardeau-Montaut. Cloud compare 2.6 (gpl software). http://www.danielgm.net/cc/, 2015. 4.5.3

[15] Daniel Grest, Jan Woetzel, and Reinhard Koch. Nonlinear body pose estimation from depth images. In *Proceedings of the 27th DAGM Conference on Pattern Recognition*, PR'05, pages 285–292, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-28703-5, 978-3-540-28703-2. doi: 10.1007/11550518_36. URL http://dx.doi.org/10.1007/11550518_36. 5.2

[16] M. Gupta, S. Upadhyay, and A. K. Nagawat. Camera calibration technique using tsais algorithm. *International Journal of Enterprise Computing and Business Systems*, 2011. 3.3.3, 5.1

[17] Ankur Handa, Viorica Patraucean, Vijay Badrinarayanan, Simon Stent, and Roberto Cipolla. Scenenet: Understanding real world indoor scenes with synthetic data. *CoRR*, abs/1511.07041, 2015. URL http://arxiv.org/abs/1511.07041. 4.5.3

[18] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004. A.4.1

[19] Paul Hebert, Nicolas Hudson, Jeremy Ma, Thomas Howard, Thomas J. Fuchs, Max Bajracharya, and Joel W. Burdick. Combined shape, appearance and silhouette for simultaneous manipulator and object tracking. In *ICRA*, pages 2405–2412. IEEE, 2012. ISBN 978-1-4673-1403-9. URL http://dblp.uni-trier.de/db/conf/icra/icra2012.html#HebertHMHFBB12. 5.2

[20] Jan Heller, Didier Henrion, and Tomás Pajdla. Hand-eye and robot-world calibration by global polynomial optimization. *CoRR*, abs/1402.3261, 2014. URL http://arxiv.org/abs/1402.3261. 5.1

[21] Liang Heng, Mathias Burki, Gim Hee Lee, Paul Furgale, Roland Siegwart, and Marc Pollefeys. Infrastructure-based calibration of a multi-camera rig. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 4912–4919. IEEE, 2014. 5.4

[22] Lionel Heng, Gim Hee Lee, and Marc Pollefeys. Self-calibration and visual slam with a multi-camera system on a micro aerial vehicle. *Autonomous Robots*, 39(3):259–277, 2015. 5.4

[23] Rethink Robotics Inc. Rethink baxter robot. http://www.barrett.com/products-arm.htm, 2015. (document), 1.2

[24] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *ACM Symposium on User Interface and Software Technology*, pages 559–568, 2011. 3.1, 3.1.1, 3.4, 4.5.5, 5.3.1

[25] Simon J Julier and Jeffrey K Uhlmann. A counter example to the theory of simultaneous localization and map building. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 4, pages 4238–4243. IEEE, 2001. 5.3.2

[26] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J.J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping using the Bayes tree. *Intl. J. of Robotics Research, IJRR*, 31(2):217–236, Feb 2012. 4.3, 4.6, 5.3.2, A.5

[27] Nima Keivan and Gabe Sibley. Online SLAM with any-time self-calibration and automatic change detection. *CoRR*, abs/1411.1372, 2014. URL http://arxiv.org/abs/1411.1372. 5.4

[28] Kinova. Kinova mico arm. http://www.kinovarobotics.com, 2015. (document), 1.2

[29] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *International Symposium on Mixed and Augmented Reality*, pages 225–234, 2007. 5.3.2

[30] M. Klingensmith, T. Galluzzo, C. Dellin, M. Kazemi, J. A. Bagnell, and N. Pollard. Closed-loop Servoing using Real-time Markerless Arm Tracking. In *International Conferene on Robotics and Automation (Humanoids Workshop)*, 2013. 1, 5.2

[31] M. Klingensmith, I. Dryanovski, S. Srinivasa, and J. Xiao. Chisel: Real time large scale 3d reconstruction onboard a mobile device. In *Robotics Science and Systems*, 2015. 3.2.1, 3.3.2, 5.3.1

[32] Matthew Klingensmith, Siddhartha Srinivasa, and Michael Kaess. Articulated robot motion for simultaneous localization and mapping (arm-slam). *IEEE Robotics and Automation - Letters*, January 2016. 1, 3.1.2, 4.2.2

[33] Laurent Kneip and Paul Timothy Furgale. Opengv: A unified and generalized approach to real-time calibrated geometric vision. In *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*, pages 1–8, 2014. doi: 10.1109/ICRA.2014.6906582. URL http://dx.doi.org/10.1109/ICRA.2014.6906582. A.4.2

[34] Michael Krainin, Peter Henry, Xiaofeng Ren, and Dieter Fox. Manipulator and object tracking for in-hand 3d object modeling. *The International Journal of Robotics Research*, 30(11):1311–1327, 2011. 5.2

[35] S Leutenegger, S Lynen, M Bosse, R Siegwart, and P Furgale. Keyframe-based visual-inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34:314–334, 2014. doi: 10.1177/0278364914554813. URL http://dx.doi.org/10.1177/0278364914554813. 5.3.2

[36] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2548–2555, Washington, DC, USA, 2011. IEEE Com-

puter Society. ISBN 978-1-4577-1101-5. doi: 10.1109/ICCV.2011.6126542. URL http://dx.doi.org/10.1109/ICCV.2011.6126542. 4.4, A.7.1

[37] Barrett Technology LLC. Barrett wam robot arm. http://www.barrett.com/products-arm.htm, 2015. (document), 1.2

[38] M. Mason. *Mechanics of Robotic Manipulation*. MIT Press, Cambridge, MA, 2001. A.2, A.2.2

[39] Jérôme Maye, Hannes Sommer, Gabriel Agamennoni, Roland Siegwart, and Paul Timothy Furgale. Online self-calibration for robotic systems. *I. J. Robotic Res.*, 35(4):357–380, 2016. doi: 10.1177/0278364915596232. URL http://dx.doi.org/10.1177/0278364915596232. 4.6, 5.4, 6

[40] Thomas B. Moeslund and Erik Granum. A survey of computer vision-based human motion capture. *Comput. Vis. Image Underst.*, 81(3):231–268, March 2001. ISSN 1077-3142. doi: 10.1006/cviu.2000.0897. URL http://dx.doi.org/10.1006/cviu.2000.0897. 5.2

[41] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598. AAAI, 2002. 5.3.2

[42] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *CoRR*, abs/1502.00956, 2015. URL http://arxiv.org/abs/1502.00956. 4.5.5

[43] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. Dtam: Dense tracking and mapping in real-time. In *International Conference on Computer Vision*, pages 2320–2327, 2011. 5.3.1

[44] Occipital. Occipital structure sensor. http://structure.io/, 2015. 3.3.2

[45] Edwin Olson. Apriltag: A robust and flexible multi-purpose fiducial system. Technical report, University of Michigan APRIL Laboratory, May 2010. 4.4

[46] Optitrack. Optitrack motion capture. http://www.optitrack.com/, 2015. 5.2

[47] Paul Ozog and Ryan M Eustice. On the importance of modeling camera calibration uncertainty in visual slam. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3777–3784. IEEE, 2013. 5.4

[48] K. Perlin. Improving noise. In *ACM Special Interest Group on Graphics and Interactive Techniques*, pages 681–682, 2002. 3.3.1, 4.5.3

[49] Vijay Pradeep, Kurt Konolige, and Eric Berger. Calibrating a multi-arm multi-sensor robot: A bundle adjustment approach. In *Experimental Robotics*, pages 211–225. Springer, 2014. 5.4

[50] D. Roetenberg, H. Luinge, and P. Slycke. Xsens mvn: full 6dof human motion tracking using miniature inertial sensors. xsens motion technologies bv. Technical report, XSens, 2009. 5.2

[51] Alessandro Roncone, Matej Hoffmann, Ugo Pattacini, and Giorgio Metta. Automatic kinematic chain calibration using artificial skin: Self-touch in the icub humanoid robot. In *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*, pages 2305–2312, 2014. doi: 10.1109/ICRA.2014.6907178. URL http://dx.doi.org/10.1109/ICRA.2014.6907178. 5.4

[52] T. Schmidt, R. Newcombe, and D. Fox. DART: Dense Articulated Real-Time Tracking. In *Robotics Science and Systems*, volume 2, 2014. URL http://www.roboticsproceedings.org/RoboticsScienceandSystems10/p30.pdf. 5.2

[53] Tanner Schmidt, Katharina Hertkorn, Richard Newcombe, Zoltan Marton, Michael Suppa, and Dieter Fox. Depth-based tracking with physical constraints for robot manipulation. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 119–126. IEEE, 2015. 5.2

[54] Mili Shah, Roger D. Eastman, and Tsai Hong. An overview of robot-sensor calibration methods for evaluation of perception systems. In *Proceedings of the Workshop on Performance Metrics for Intelligent Systems*, PerMIS '12, pages 15–20, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1126-7. doi: 10.1145/2393091.2393095. URL http://doi.acm.org/10.1145/2393091.2393095. 5.1, 5.1

[55] Jamie Shotton, Toby Sharp, Alex Kipman, Andrew Fitzgibbon, Mark Finocchio, Andrew Blake, Mat Cook, and Richard Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124, 2013. 5.2

[56] R. Smith, M. Self, and P. Cheeseman. Autonomous robot vehicles. chapter Estimating Uncertain Spatial Relationships in Robotics, pages 167–193. Springer-Verlag New York, Inc., New York, NY, USA, 1990. ISBN 0-387-97240-4. URL http://dl.acm.org/citation.cfm?id=93002.93291. 5.3.2

[57] Eduardo D. Sontag. *Mathematical Control Theory: Deterministic Finite Dimensional Systems (2Nd Ed.)*. Springer-Verlag New York, Inc., New York, NY, USA, 1998. ISBN 0-387-98489-5. 4.6

[58] H. Stewénius, C. Engels, and D. Nistér. An efficient minimal solution for infinitesimal camera motion. In *Accepted for CVPR*, Zurich, Switzerland, June 2007. A.4.2

[59] Alex Teichman, Stephen Miller, and Sebastian Thrun. Unsupervised intrinsic calibration of depth sensors via slam. In *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013. doi: 10.15607/RSS.2013.IX.027. 5.4

[60] T. Whelan, M. Kaess, M.F. Fallon, H. Johannsson, J.J. Leonard, and J.B. McDonald. Kintinuous: Spatially extended KinectFusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Sydney, Australia, Jul 2012. 5.3.1

[61] T. Whelan, S. Leutenegger, R. S. Moreno, B. Glocker, and A. Davison. Elasticfusion: Dense slam without a pose graph. In *Robotics Science and Systems*, 2015. 5.3.1

[62] J. Zhang, M. Kaess, and S. Singh. On degeneracy of optimization-based state estimation problems. In *IEEE Intl. Conf. on Robotics and Automation, ICRA*, pages 809–816, Stock-

holm, Sweden, May 2016. 4.6

[63] Zhengyou Zhang. Parameter estimation techniques: A tutorial with application to conic fitting. *Image and Vision Computing*, 15(1):59–76, 1997. A.6