





Data-Driven Batch Localization and SLAM Using Koopman Linearization

Zi Cong Guo , *Student Member, IEEE*, Frederike Dümbgen , *Member, IEEE*,
James R. Forbes , *Member, IEEE*, Timothy D. Barfoot , *Fellow, IEEE*

Abstract—We present a framework for model-free batch localization and SLAM. We use lifting functions to map a control-affine system into a high-dimensional space, where both the process model and the measurement model are rendered bilinear. During training, we solve a least-squares problem using groundtruth data to compute the high-dimensional model matrices associated with the lifted system purely from data. At inference time, we solve for the unknown robot trajectory and landmarks through an optimization problem, where constraints are introduced to keep the solution on the manifold of the lifting functions. The problem is efficiently solved using a sequential quadratic program (SQP), where the complexity of an SQP iteration scales linearly with the number of timesteps. Our algorithms, called Reduced Constrained Koopman Linearization Localization (RCKL-Loc) and Reduced Constrained Koopman Linearization SLAM (RCKL-SLAM), are validated experimentally in simulation and on two datasets: one with an indoor mobile robot equipped with a laser rangefinder that measures range to cylindrical landmarks, and one on a golf cart equipped with RFID range sensors. We compare RCKL-Loc and RCKL-SLAM with classic model-based nonlinear batch estimation. While RCKL-Loc and RCKL-SLAM have similar performance compared to their model-based counterparts, they outperform the model-based approaches when the prior model is imperfect, showing the potential benefit of the proposed data-driven technique.

I. INTRODUCTION

STATE estimation, and simultaneous localization and mapping (SLAM) in particular, are of prime importance for many robotics systems. Most state-estimation methods rely on complex modelling and/or problem-specific techniques, and the procedure often involves linearization. Moreover, when process or measurement models are inaccurate, or even unavailable, model-based state estimation methods can struggle to produce accurate and consistent navigation results. For many applications [1], [2], [3], the procedure for modelling and/or solving the estimation problem is much more involved than the data-gathering process.

In this work, we propose the Reduced Constrained Koopman Linearization (RCKL) framework for data-driven localization and SLAM. The method

- learns a high-dimensional (bi)linear system model from data without requiring prior models,

Manuscript received February 13, 2024; accepted July 17, 2024. This paper was recommended for publication by Editor Javier Civera upon evaluation of the reviewers' comments. This work was funded in part by the Swiss National Science Foundation, Postdoc Mobility under Grant 206954, and in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

Zi Cong Guo, Frederike Dümbgen, and Timothy D. Barfoot are with the University of Toronto Robotics Institute, Toronto, Ontario, Canada (email: zc.guo@mail.utoronto.ca; frederike.dumbgen@utoronto.ca; tim.barfoot@utoronto.ca).

James R. Forbes is with the Department of Mechanical Engineering, McGill University, Montreal, Quebec, Canada (email: james.richard.forbes@mcgill.ca).

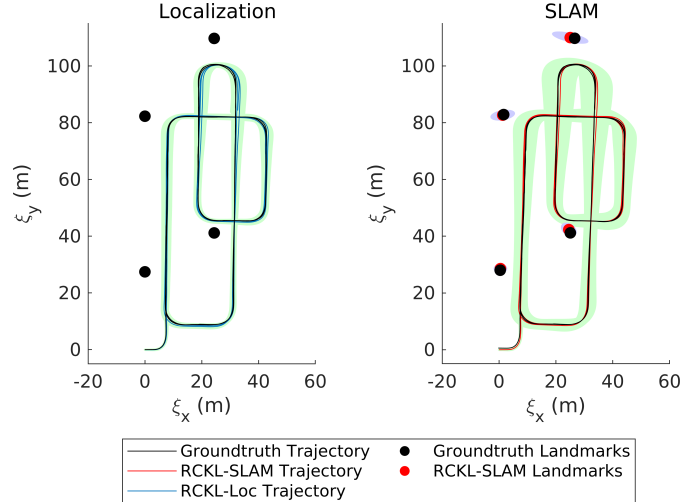


Fig. 1: Visualization of the trajectory output of RCKL-Loc (left), and the trajectory and landmark output of RCKL-SLAM (right), on Experiment 2 described in Section X-D, showing the estimators' mean states and mean landmark positions compared to the groundtruth. The green regions and the grey regions are, respectively, the 3σ covariances of the trajectory and of the landmarks. The estimators' trajectories and landmarks are close to the groundtruth and are within the estimated 3σ bounds, despite the algorithm having no prior knowledge of the system models.

- is applicable to systems with control-affine dynamics,
- solves for the unknown states and landmark positions with an inference cost that scales linearly with the number of timesteps, and
- has a training cost that scales linearly with the amount of data and an inference cost that is independent of the amount of training data.

RCKL is an extension of the Koopman State Estimator (KoopSE) [4], a framework for data-driven localization where the landmarks are at the same positions at test time as during training. RCKL extends KoopSE by allowing for localization with landmarks at different positions at test time, and allowing for SLAM when the landmarks are unknown at test time. Specifically, the main novelties of RCKL over KoopSE are (i) a formulation that allows for the joint estimation of poses and landmarks, (ii) the introduction of constrained optimization over the manifolds defined by the lifting functions, and (iii) a formulation of a Sequential Quadratic Program (SQP) [5] to efficiently perform this optimization. Given a robot with unknown nonlinear dynamics and measurements, we can first learn its high-dimensional model in a factory setting with groundtruth, and then deploy the robot in the field and use RCKL-SLAM to map the new landmarks. Once the landmarks are known, we can then potentially use RCKL-Loc to perform batch (or recursive) localization against the map.

This paper is structured as follows. We review related work in Section II, summarize themes in our notation in Section III, and discuss system lifting with Koopman linearization in Sections IV and V. We outline the model-learning procedure in Section VI. We then build up the design of RCKL through Sections VII-IX, where we discuss the procedure and limitations of Unconstrained Koopman Linearization (UKL) in Section VII, Constrained Koopman Linearization (CKL) in Section VIII, and finally RCKL in Section IX. We present experimental results in Section X and conclude in Section XI.

II. RELATED WORK

Two common concepts for data-driven algorithms are kernel embeddings [6] and the Koopman operator [7], both used to transform a system into a simpler form in a high-dimensional space. Kernel methods lift the system by embedding probability distributions in high-dimensional spaces [8], and there has been some work on kernelized state estimation and SLAM for limited model types using Gaussian processes [9], [10] and kernel embeddings [11]. However, pure kernel methods scale poorly, typically cubically with the amount of training data. Koopman-based methods, on the other hand, work by lifting the variables directly, and both the model-learning process and the inference process typically scale well with data [12]. There is some work on Koopman-based state estimation, validated for small problems in simulation [13], [14]. However, validations on real-world datasets are limited, and efficient learning and inference techniques have yet to be developed for large-scale state-estimation problems such as SLAM. Rather, most of the attention on Koopman in robotics has been on filtering [15] or model-predictive control for nonlinear systems [16], [17], [18], which are applications with short time horizons. In general, there is limited work on using the Koopman operator to lift and solve large batch optimization problems, including SLAM.

There is a large body of work on system identification with kernels [19], [20] and Koopman techniques [21], [22], [23]. These methods solve linear-like optimization problems a lifted space and typically have better convergence properties than classic methods. However, there is little work on actually employing the identified lifted models for state estimation. Converting them back into the original state space and using classic nonlinear estimation methods would still be challenging, especially if the models are noisy or high dimensional [24]. Instead, we seek a data-driven method that both identifies the models and employs them for estimation efficiently in the lifted space.

One method for data-driven state estimation is KoopSE [4], which lifts a control-affine system such that standard batch linear-Gaussian state estimation methods can be used. KoopSE is validated on a mobile robot dataset. However, KoopSE incorporates all of the measurements as part of one large model for the whole environment. Thus, when carrying out localization, the landmarks are required to be at the same positions at test time as during training. With no explicit specification for the landmarks, this formulation also cannot be used for SLAM. This paper aims to begin filling this gap by formulating data-driven bilinear localization and SLAM for control-affine systems.

III. NOTATION

We denote matrices with capital boldface letters, \mathbf{A} , vectors with lowercase boldface letters, \mathbf{a} , and scalars with normal-faced letters, a . We use $\text{diag}(\mathbf{A}_1, \dots, \mathbf{A}_N)$ to denote the block-diagonal matrix with the blocks being $\mathbf{A}_1, \dots, \mathbf{A}_N$. We denote quantities in the original, unlifted space with Greek letters, ξ , and quantities in the lifted space with Roman letters, \mathbf{x} . We use the same letter of the two alphabets to denote connected quantities whenever possible, e.g., ξ for the original state and \mathbf{x} for the lifted state. For the lifted quantities, we denote batch terms with italics, \mathbf{x} , and non-batch terms without italics.

IV. KOOPMAN LIFTING OF PROCESS MODELS

In this section, we briefly introduce the concept of Koopman lifting [7] of discrete process models. For a comprehensive review of the Koopman operator, see [12], [17].

The Koopman framework's main idea is to lift a nonlinear autonomous system into a high-dimensional space where the system becomes linear. Suppose a noiseless autonomous system (i.e., no inputs) is governed by $\xi_k = \mathbf{f}(\xi_{k-1})$, where ξ_k is the state at timestep k and $\mathbf{f}(\xi)$ is the process model. Then, there exists an embedding, \mathbf{p} , and a linear operator (the Koopman operator), \mathcal{K} , such that $\mathbf{p}(\xi_k) = \mathcal{K}\mathbf{p}(\xi_{k-1})$. We call $\mathbf{x}_k = \mathbf{p}(\xi_k)$ the embedded state. The lifted system dynamics become $\mathbf{x}_k = \mathcal{K}\mathbf{x}_{k-1}$, which is linear. Although a finite-dimensional \mathcal{K} and \mathbf{p} may be hard (or impossible; see [25]) to find, this transformation always exists. As an example from [25], suppose that a 2-dimensional system is governed by the nonlinear dynamics

$$\begin{bmatrix} \xi_{1,k} \\ \xi_{2,k} \end{bmatrix} = \mathbf{f} \left(\begin{bmatrix} \xi_{1,k-1} \\ \xi_{2,k-1} \end{bmatrix} \right) = \begin{bmatrix} \lambda \xi_{1,k-1} \\ \mu \xi_{2,k-1} + (\lambda^2 - \mu) \xi_{1,k-1}^2 \end{bmatrix}, \quad (1)$$

for some parameters λ and μ . In this case, the nonlinear system can be transformed into a linear system with the finite-dimensional lifting function $\mathbf{p}(\xi_k) = [\xi_{1,k} \ \xi_{2,k} \ \xi_{1,k}^2]^T = [x_{1,k} \ x_{2,k} \ x_{3,k}]^T$, yielding

$$\begin{bmatrix} x_{1,k} \\ x_{2,k} \\ x_{3,k} \end{bmatrix} = \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \mu & \lambda^2 - \mu \\ 0 & 0 & \lambda^2 \end{bmatrix} \begin{bmatrix} x_{1,k-1} \\ x_{2,k-1} \\ x_{3,k-1} \end{bmatrix}. \quad (2)$$

In cases where an exact finite-dimensional Koopman operator cannot be found, using a truncated approximation can still be reasonable [25]. This concept can be generalized to non-autonomous systems [26]. However, there is limited work on identifying and using the Koopman operator for estimation in the presence of noise, inputs, and measurements. In the next section, we generalize the Koopman lifting framework to our systems of interest, which include noisy process and measurement models.

V. LIFTING THE FULL SYSTEM

We focus on systems with process and measurement models of the form

$$\xi_k = \mathbf{f}(\xi_{k-1}, \nu_k, \omega_k) \quad (3a)$$

$$= \mathbf{f}_0(\boldsymbol{\xi}_{k-1}) + \sum_{i=1}^{N_\nu} \mathbf{f}_i(\boldsymbol{\xi}_{k-1}) \nu_{k,i} + \boldsymbol{\omega}_k, \quad (3b)$$

$$\boldsymbol{\gamma}_{k,j} = \mathbf{g}(\boldsymbol{\xi}_k, \boldsymbol{\psi}_j, \boldsymbol{\eta}_{k,j}) \quad (3c)$$

$$= \sum_{i=1}^{N_{g,\xi}} \sum_{m=1}^{N_{\psi,m}} \mathbf{g}_{\xi,i}(\boldsymbol{\xi}_k) \pi_{\psi,m}(\boldsymbol{\psi}_j) + \boldsymbol{\eta}_{k,j}, \quad (3d)$$

where $\boldsymbol{\xi}_k \in \mathbb{R}^{N_\xi}$ is the robot state, $\boldsymbol{\nu}_k \in \mathbb{R}^{N_\nu}$ the control input, $\boldsymbol{\omega}_k \in \mathbb{R}^{N_\omega}$ the process noise, all at timestep k . $\boldsymbol{\psi}_j \in \mathbb{R}^{N_\psi}$ is the position of landmark j , $\boldsymbol{\gamma}_{k,j} \in \mathbb{R}^{N_\gamma}$ the measurement of $\boldsymbol{\psi}_j$ at timestep k , and $\boldsymbol{\eta}_{k,j} \in \mathbb{R}^{N_\eta}$ the measurement noise for $\boldsymbol{\gamma}_{k,j}$. \mathbf{f}_i are the various components of the process model, and $\mathbf{g}_{\xi,i}, \pi_{\psi,m}$ are the various components of the measurement model. We have chosen to focus on systems with a control-affine process model (3b) since the model can be written exactly as a lifted bilinear model when the system is noiseless [27]. In the same spirit, we have chosen a measurement model of the form (3d), where it is nonlinear in the state and the landmark position but not both, such that the model can be exactly written as a lifted bilinear model. Many common robot process and measurement models can be written in this form.

The problem of localization is to estimate the states, $\boldsymbol{\xi}_k$, given a series of inputs, $\boldsymbol{\nu}_k$, landmark positions, $\boldsymbol{\psi}_j$, and measurements, $\boldsymbol{\gamma}_{k,j}$. The problem of SLAM is to estimate the states while moving the landmark positions to the list of unknowns: estimate $\boldsymbol{\xi}_k$ and $\boldsymbol{\psi}_j$ given only $\boldsymbol{\nu}_k$ and $\boldsymbol{\gamma}_{k,j}$. Both problems are difficult when the models, \mathbf{f}_i , $\mathbf{g}_{\xi,i}$, $\pi_{\psi,m}$, are nonlinear, and especially so if the models are inaccurate or unknown. Rather than solving these problems directly, we first embed each of the states, inputs, landmarks, and measurements into high-dimensional spaces,

$$\mathbf{x}_k = \mathbf{p}_\xi(\boldsymbol{\xi}_k), \quad \mathbf{u}_k = \mathbf{p}_\nu(\boldsymbol{\nu}_k), \quad (4a)$$

$$\boldsymbol{\ell}_j = \mathbf{p}_\psi(\boldsymbol{\psi}_j), \quad \mathbf{y}_{k,j} = \mathbf{p}_\gamma(\boldsymbol{\gamma}_{k,j}), \quad (4b)$$

where

$$\mathbf{p}_\xi : \mathbb{R}^{N_\xi} \rightarrow \mathcal{X}, \quad \mathbf{p}_\nu : \mathbb{R}^{N_\nu} \rightarrow \mathcal{U}, \quad (5a)$$

$$\mathbf{p}_\psi : \mathbb{R}^{N_\psi} \rightarrow \mathcal{L}, \quad \mathbf{p}_\gamma : \mathbb{R}^{N_\gamma} \rightarrow \mathcal{Y}, \quad (5b)$$

are the embeddings associated with the manifolds $\mathcal{X}, \mathcal{U}, \mathcal{L}$, and \mathcal{Y} , respectively. These embeddings are user-defined and can be customized for specific systems. We discuss some choices for these embeddings later in the Experiments (Section X). For the rest of the paper, we assume that the manifolds are within finite-dimensional vector spaces:

$$\mathcal{X} \subseteq \mathbb{R}^{N_x}, \quad \mathcal{U} \subseteq \mathbb{R}^{N_u}, \quad \mathcal{L} \subseteq \mathbb{R}^{N_\ell}, \quad \mathcal{Y} \subseteq \mathbb{R}^{N_y}, \quad (6)$$

and let $N = \max(N_x, N_u, N_\ell, N_y)$.

For the process model, we follow the same lifting procedure as [4], [27], where the control-affine model becomes a bilinear-Gaussian model in the lifted space. That is, (3b) becomes

$$\mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_k + \mathbf{H}(\mathbf{u}_k \otimes \mathbf{x}_{k-1}) + \mathbf{w}_k, \quad (7)$$

where \otimes represents the tensor product, equivalent to the Kronecker product for a finite-dimensional \mathcal{X} . The reasoning is that a deterministic control-affine model (i.e., $\boldsymbol{\omega}_k = \mathbf{0}$) involves

products of nonlinear functions of the robot state and the control input individually, but not nonlinear functions of both quantities. Thus, it can be written exactly as a lifted bilinear model [27]. When there is system noise, [4] suggest that a stochastic control-affine model can be modelled similarly with an additive Gaussian noise. In the same spirit, we now lift the measurement model in (3d) so that it also becomes bilinear in the deterministic case,

$$\boldsymbol{\gamma}_{k,j} = \mathbf{g}(\boldsymbol{\xi}_k, \boldsymbol{\psi}_j, \mathbf{0}) \Rightarrow \mathbf{y}_{k,j} = \mathbf{C}(\boldsymbol{\ell}_j \otimes \mathbf{x}_k), \quad (8)$$

and assume that the noise from a stochastic measurement model can also be modelled as additive-Gaussian noise. This results in the full lifted system,

$$\mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_k + \mathbf{H}(\mathbf{u}_k \otimes \mathbf{x}_{k-1}) + \mathbf{w}_k, \quad (9a)$$

$$\mathbf{y}_{k,j} = \mathbf{C}(\boldsymbol{\ell}_j \otimes \mathbf{x}_k) + \mathbf{n}_{k,j}, \quad (9b)$$

where $\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ and $\mathbf{n}_{k,j} \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ are the process and measurement noises, respectively, and

$$\mathbf{w}_k \in \mathcal{X}, \quad \mathbf{n}_{k,j} \in \mathcal{Y}, \quad (10a)$$

$$\mathbf{A} : \mathcal{X} \rightarrow \mathcal{X}, \quad \mathbf{B} : \mathcal{U} \rightarrow \mathcal{X}, \quad (10b)$$

$$\mathbf{H} : \mathcal{U} \otimes \mathcal{X} \rightarrow \mathcal{X}, \quad \mathbf{C} : \mathcal{L} \otimes \mathcal{X} \rightarrow \mathcal{Y}, \quad (10c)$$

$$\mathbf{Q} \in \mathcal{X} \times \mathcal{X}, \quad \mathbf{R} \in \mathcal{Y} \times \mathcal{Y}, \quad (10d)$$

where \mathbf{Q} and \mathbf{R} are positive definite. Note that for a closer parallel to the process model, the measurement model would contain three terms, $\mathbf{y}_{k,j} = \mathbf{C}_1\mathbf{x}_k + \mathbf{C}_2\boldsymbol{\ell}_j + \mathbf{C}_3(\boldsymbol{\ell}_j \otimes \mathbf{x}_k) + \mathbf{n}_{k,j}$, and the subsequent derivations would still follow through. However, the extra terms had little effects during the experimental evaluation, and therefore the one-term version is presented for simplicity.

As we will see in Section VII, it is significantly easier to work with the lifted bilinear system in (9) than with the original nonlinear system in (3). However, we first need to learn the lifted models from data.

VI. SYSTEM IDENTIFICATION

A. Lifted Matrix Form of Dataset

Our objective is to learn the lifted system matrices $\mathbf{A}, \mathbf{B}, \mathbf{H}, \mathbf{C}, \mathbf{Q}, \mathbf{R}$ in (9) from data. We assume a dataset of the control-affine system in (3), including the ground-truth state transitions with their associated control inputs for P states: $\{\hat{\boldsymbol{\xi}}_i, \boldsymbol{\xi}_i, \boldsymbol{\nu}_i\}_{i=1}^P$. Here, $\hat{\boldsymbol{\xi}}_i$ transitions to $\boldsymbol{\xi}_i$ under input $\boldsymbol{\nu}_i$. If the dataset consists of a single trajectory of $P+1$ states and i represents the timestep, then we would set $\hat{\boldsymbol{\xi}}_i = \boldsymbol{\xi}_{i-1}$. We also assume a dataset of the associated landmark measurements: $\{\{\boldsymbol{\gamma}_{i,j}, \boldsymbol{\psi}_{i,j}\}_{j=1}^{\beta_i}\}_{i=1}^P$. Here, β_i is the number of landmarks seen at timestep i , $\boldsymbol{\psi}_{i,j}$ is the position of the j th landmark at timestep i , and $\boldsymbol{\gamma}_{i,j}$ is the measurement received from landmark $\boldsymbol{\psi}_{i,j}$ at timestep i . This format allows for data from one or multiple training trajectories to be used at once, and also allows us to combine datasets with different landmark positions. We write the data neatly in block-matrix form:

$$\boldsymbol{\Xi} = [\boldsymbol{\xi}_1 \quad \cdots \quad \boldsymbol{\xi}_P], \quad \hat{\boldsymbol{\Xi}} = [\hat{\boldsymbol{\xi}}_1 \quad \cdots \quad \hat{\boldsymbol{\xi}}_P], \quad (11a)$$

$$\boldsymbol{\Upsilon} = [\boldsymbol{\nu}_1 \quad \cdots \quad \boldsymbol{\nu}_P], \quad (11b)$$

$$\boldsymbol{\Gamma} = [\boldsymbol{\gamma}_{1,1} \quad \cdots \quad \boldsymbol{\gamma}_{1,\beta_1} \mid \cdots \mid \boldsymbol{\gamma}_{P,1} \quad \cdots \quad \boldsymbol{\gamma}_{P,\beta_P}], \quad (11c)$$

$$\Psi = [\psi_{1,1} \ \cdots \ \psi_{1,\beta_1} \mid \cdots \mid \psi_{P,1} \ \cdots \ \psi_{P,\beta_P}]. \quad (11d)$$

The data lifts to $\{\hat{\mathbf{x}}_i, \mathbf{x}_i, \mathbf{u}_i\}_{i=1}^P$ and $\{\{y_{i,j}, \ell_{i,j}\}_{j=1}^{\beta_i}\}_{i=1}^P$ in the embedded space such that

$$\mathbf{x}_i = \mathbf{A}\hat{\mathbf{x}}_i + \mathbf{B}\mathbf{u}_i + \mathbf{H}(\mathbf{u}_i \otimes \hat{\mathbf{x}}_i) + \mathbf{w}_i, \quad (12a)$$

$$\mathbf{y}_{i,j} = \mathbf{C}(\ell_j \otimes \mathbf{x}_i) + \mathbf{n}_{i,j}, \quad (12b)$$

for some unknown noise, $\mathbf{w}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$, $\mathbf{n}_{i,j} \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$. We rewrite the lifted versions of the data and the noises in block-matrix form:

$$\mathbf{X} = [\mathbf{x}_1 \ \cdots \ \mathbf{x}_P], \quad \hat{\mathbf{X}} = [\hat{\mathbf{x}}_1 \ \cdots \ \hat{\mathbf{x}}_P], \quad (13a)$$

$$\mathbf{U} = [\mathbf{u}_1 \ \cdots \ \mathbf{u}_P], \quad \mathbf{W} = [\mathbf{w}_1 \ \cdots \ \mathbf{w}_P], \quad (13b)$$

$$\mathbf{Y} = [\mathbf{y}_{1,1} \ \cdots \ \mathbf{y}_{1,\beta_1} \mid \cdots \mid \mathbf{y}_{P,1} \ \cdots \ \mathbf{y}_{P,\beta_P}], \quad (13c)$$

$$\mathbf{L} = [\ell_{1,1} \ \cdots \ \ell_{1,\beta_1} \mid \cdots \mid \ell_{P,1} \ \cdots \ \ell_{P,\beta_P}], \quad (13d)$$

$$\mathbf{N} = [\mathbf{n}_{1,1} \ \cdots \ \mathbf{n}_{1,\beta_1} \mid \cdots \mid \mathbf{n}_{P,1} \ \cdots \ \mathbf{n}_{P,\beta_P}], \quad (13e)$$

$$\check{\mathbf{X}} = \underbrace{[\mathbf{x}_1 \ \cdots \ \mathbf{x}_1]}_{\beta_1} \mid \cdots \mid \underbrace{[\mathbf{x}_P \ \cdots \ \mathbf{x}_P]}_{\beta_P}, \quad (13f)$$

where we defined $\check{\mathbf{X}}$ as the states duplicated based on the number of landmarks seen at each timestep. We also define $S = \sum_{i=1}^P \beta_i$ as the total number of measurements. Then, \mathbf{X} , $\hat{\mathbf{X}}$, \mathbf{U} , \mathbf{W} are all P columns wide, while \mathbf{Y} , \mathbf{L} , \mathbf{N} , $\check{\mathbf{X}}$ are all S columns wide. With these definitions, the lifted matrix form of the system for (12) is

$$\mathbf{X} = \mathbf{A}\hat{\mathbf{X}} + \mathbf{B}\mathbf{U} + \mathbf{H}(\mathbf{U} \odot \hat{\mathbf{X}}) + \mathbf{W}, \quad (14a)$$

$$\mathbf{Y} = \mathbf{C}(\mathbf{L} \odot \check{\mathbf{X}}) + \mathbf{N}, \quad (14b)$$

where \odot denotes the Khatri-Rao (column-wise) tensor product.

B. Loss Function

The model-learning problem is posed as

$$\{\mathbf{A}^*, \mathbf{B}^*, \mathbf{H}^*, \mathbf{C}^*, \mathbf{Q}^*, \mathbf{R}^*\} = \underset{\{\mathbf{A}, \mathbf{B}, \mathbf{H}, \mathbf{C}, \mathbf{Q}, \mathbf{R}\}}{\operatorname{argmin}} V(\mathbf{A}, \mathbf{B}, \mathbf{H}, \mathbf{C}, \mathbf{Q}, \mathbf{R}), \quad (15)$$

where the loss function, $V = V_1 + V_2$, is the sum of

$$V_1 = \frac{1}{2} \left\| \mathbf{X} - \mathbf{A}\hat{\mathbf{X}} - \mathbf{B}\mathbf{U} - \mathbf{H}(\mathbf{U} \odot \hat{\mathbf{X}}) \right\|_{\mathbf{Q}^{-1}}^2 + \frac{1}{2} \left\| \mathbf{Y} - \mathbf{C}(\mathbf{L} \odot \check{\mathbf{X}}) \right\|_{\mathbf{R}^{-1}}^2 - \frac{1}{2} P \ln |\mathbf{Q}^{-1}| - \frac{1}{2} S \ln |\mathbf{R}^{-1}|, \quad (16a)$$

$$V_2 = \frac{1}{2} P \tau_A \|\mathbf{A}\|_{\mathbf{Q}^{-1}}^2 + \frac{1}{2} P \tau_B \|\mathbf{B}\|_{\mathbf{Q}^{-1}}^2 + \frac{1}{2} P \tau_H \|\mathbf{H}\|_{\mathbf{Q}^{-1}}^2 + \frac{1}{2} S \tau_C \|\mathbf{C}\|_{\mathbf{R}^{-1}}^2 + \frac{1}{2} P \tau_Q \operatorname{tr}(\mathbf{Q}^{-1}) + \frac{1}{2} S \tau_R \operatorname{tr}(\mathbf{R}^{-1}), \quad (16b)$$

where $|\mathbf{X}|$ represents the determinant of \mathbf{X} , and the norm is a weighted Frobenius matrix norm: $\|\mathbf{X}\|_{\mathbf{W}} = \sqrt{\operatorname{tr}(\mathbf{X}^T \mathbf{W} \mathbf{X})}$. Similar to [4], V_1 represents the negative log-likelihood of the Bayesian posterior, and V_2 contains prior terms to penalize the description lengths of \mathbf{A} , \mathbf{B} , \mathbf{H} , and \mathbf{C} and inverse-Wishart (IW) priors for the covariances, \mathbf{Q} and \mathbf{R} . The regularizing hyperparameters, $\tau_A, \tau_B, \tau_H, \tau_C, \tau_Q, \tau_R$, can be tuned to maximize performance if desired.

We find the critical points by setting the derivatives of V

with respect to the model parameters $(\frac{\partial V}{\partial \mathbf{A}}, \frac{\partial V}{\partial \mathbf{B}}, \frac{\partial V}{\partial \mathbf{H}}, \frac{\partial V}{\partial \mathbf{C}}, \frac{\partial V}{\partial \mathbf{Q}^{-1}}$, and $\frac{\partial V}{\partial \mathbf{R}^{-1}}$) to zero. We define

$$\mathbf{V} = \mathbf{U} \odot \hat{\mathbf{X}}, \quad \mathbf{J} = \mathbf{X} - \mathbf{A}\hat{\mathbf{X}} - \mathbf{B}\mathbf{U} - \mathbf{H}\mathbf{V}, \quad (17a)$$

$$\mathbf{Z} = \mathbf{L} \odot \check{\mathbf{X}}, \quad \mathbf{T} = \mathbf{Y} - \mathbf{C}\mathbf{Z}. \quad (17b)$$

This yields the following expressions that can be solved for \mathbf{A} , \mathbf{B} , \mathbf{C} , and \mathbf{H} :

$$\begin{bmatrix} \hat{\mathbf{X}}\hat{\mathbf{X}}^T + P\tau_A \mathbf{1} & \hat{\mathbf{X}}\mathbf{U}^T & \hat{\mathbf{X}}\mathbf{V}^T \\ \mathbf{U}\hat{\mathbf{X}}^T & \mathbf{U}\mathbf{U}^T + P\tau_B \mathbf{1} & \mathbf{U}\mathbf{V}^T \\ \mathbf{V}\hat{\mathbf{X}}^T & \mathbf{V}\mathbf{U}^T & \mathbf{V}\mathbf{V}^T + P\tau_H \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{A}^T \\ \mathbf{B}^T \\ \mathbf{H}^T \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{X}}\mathbf{X}^T \\ \mathbf{U}\mathbf{X}^T \\ \mathbf{V}\mathbf{X}^T \end{bmatrix}, \quad (18a)$$

$$\mathbf{C} = (\mathbf{Y}\mathbf{Z}^T)(\mathbf{Z}\mathbf{Z}^T + S\tau_C \mathbf{1})^{-1}, \quad (18b)$$

after which we obtain

$$\mathbf{Q} = \frac{1}{P} \mathbf{J}\mathbf{J}^T + \tau_A \mathbf{A}\mathbf{A}^T + \tau_B \mathbf{B}\mathbf{B}^T + \tau_H \mathbf{H}\mathbf{H}^T + \tau_Q \mathbf{1}, \quad (18c)$$

$$\mathbf{R} = \frac{1}{S} \mathbf{T}\mathbf{T}^T + \tau_C \mathbf{C}\mathbf{C}^T + \tau_R \mathbf{1}, \quad (18d)$$

where $\mathbf{1}$ represents the identity operator of the appropriate domains. The time complexity of solving for \mathbf{A} , \mathbf{B} , \mathbf{H} , \mathbf{Q} and \mathbf{R} is $\mathcal{O}(N^3(P+S))$, linear in the amount of training data.

C. Data Augmentation

If training data is scarce, system invariances can be exploited if they are known. The general system in (9) allows the model behaviour to vary arbitrarily with the states and with the landmark locations, but this freedom is not necessary for many systems. A robot may drive similarly within a room, and a rangefinding sensor may measure distances based only on the relative location of landmarks with respect to the robot. Although the injection of known invariances into (9) is an interesting avenue for future work, for now we can use data augmentation [28] to improve model accuracy. We make copies of the gathered data, perturb them based on any known invariances, and add them into the dataset matrices in (11). See the dataset experiments in Section X for an example.

VII. UNCONSTRAINED KOOPMAN LINEARIZATION (UKL)

We now outline the procedure to set up and solve a batch estimation problem with UKL. This approach is in the same spirit as in [4], which solves batch state estimation with fixed landmarks through unconstrained optimization. UKL generalizes [4] in that the formulation includes new landmark positions at test time, allowing for general localization (UKL-Loc) when the test landmarks are known but are different than during training time, and also allowing for SLAM (UKL-SLAM) when the test landmarks are unknown. We then discuss the limitations of using the minimum-cost solution of UKL, and how it leads to the introduction of manifold constraints for CKL, presented in the next section. Although the constraints turn out to be crucial for performance, we start by outlining UKL as it serves as a basis for CKL.

A. UKL Batch Estimation Problem Setup

Having learned the system matrices from training data in Section VI, we now move to an environment with a new

set of landmarks, $\{\psi_j\}_{j=1}^V$, where we wish to solve for a sequence of states, $\{\xi_k\}_{k=0}^K$, given a series of inputs, $\{\nu_k\}_{k=1}^K$, and measurements, $\{\{\gamma_{k,j}\}_{j=1}^V\}_{k=0}^K$. If the landmarks are all known, the problem is simply localization. If any landmarks are unknown, the problem is SLAM and we also wish to estimate the unknown landmarks. We do this in the lifted space, where the quantities become $\{\ell_j\}_{j=1}^V$, $\{\mathbf{x}_k\}_{k=0}^K$, $\{\mathbf{u}_k\}_{k=1}^K$, and $\{\{\mathbf{y}_{k,j}\}_{j=1}^V\}_{k=0}^K$, respectively. Since the inputs are completely determined at test time, we can simplify the bilinear process model of (9a) into a linear model. Observe that $\mathbf{u}_k \otimes \mathbf{x}_{k-1} = (\mathbf{u}_k \otimes \mathbf{1})\mathbf{x}_{k-1}$, where here $\mathbf{1} : \mathcal{X} \rightarrow \mathcal{X}$. Using the same trick as in [4], we use the known input at test time, \mathbf{u}_k , to define a new time-varying system matrix, \mathbf{A}_{k-1} , and a new input, \mathbf{v}_k , as

$$\mathbf{A}_{k-1} = \mathbf{A} + \mathbf{H}(\mathbf{u}_k \otimes \mathbf{1}), \quad \mathbf{v}_k = \mathbf{B}\mathbf{u}_k. \quad (19)$$

With this, we have a system in which the process model is linear-Gaussian:

$$\mathbf{x}_k = \mathbf{A}_{k-1}\mathbf{x}_{k-1} + \mathbf{v}_k + \mathbf{w}_k, \quad k = 1, \dots, K, \quad (20a)$$

$$\mathbf{y}_{k,j} = \mathbf{C}(\ell_j \otimes \mathbf{x}_k) + \mathbf{n}_{k,j}, \quad k = 0, \dots, K, \quad j = 1, \dots, V, \quad (20b)$$

where $\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ and $\mathbf{n}_{k,j} \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$. As the input is always given at test time, this form applies to both localization and SLAM.

We will now describe the process for solving UKL-SLAM, then briefly outline UKL-Loc as a special case of UKL-SLAM. A more efficient method for solving UKL-Loc using the RTS smoother [24] is described in Appendix A.

B. Solving UKL-SLAM

If some or all landmarks are unknown, we can solve for both the poses and the unknown landmarks by formulating and solving a batch linear SLAM problem. We first describe the case where all landmark positions are unknown and where the robot receives a measurement for all landmarks at each timestep, and later describe how to modify the method to allow for variations in the problem. As is often done, we assume that \mathbf{x}_0 is known in order to render a unique solution to the SLAM problem. We define

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_K \end{bmatrix}, \quad \ell = \begin{bmatrix} \ell_1 \\ \vdots \\ \ell_V \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_K \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_K \end{bmatrix}, \quad \mathbf{y}_k = \begin{bmatrix} \mathbf{y}_{k,1} \\ \vdots \\ \mathbf{y}_{k,V} \end{bmatrix}. \quad (21)$$

The pose errors and measurement errors are, respectively,

$$\mathbf{e}_{\mathbf{x},k}(\mathbf{x}_k) = (\mathbf{A}_{k-1}\mathbf{x}_{k-1} + \mathbf{v}_k) - \mathbf{x}_k, \quad (22a)$$

$$\mathbf{e}_{\mathbf{y},k,j}(\mathbf{x}_k, \ell_j) = \mathbf{C}(\ell_j \otimes \mathbf{x}_k) - \mathbf{y}_{k,j}. \quad (22b)$$

We can formulate the cost function as

$$J(\mathbf{x}, \ell) = \frac{1}{2} \sum_{k=1}^K \mathbf{e}_{\mathbf{x},k}(\mathbf{x}_k)^T \mathbf{Q}^{-1} \mathbf{e}_{\mathbf{x},k}(\mathbf{x}_k) + \frac{1}{2} \sum_{k=1}^K \sum_{j=1}^V \mathbf{e}_{\mathbf{y},k,j}(\mathbf{x}_k, \ell_j)^T \mathbf{R}^{-1} \mathbf{e}_{\mathbf{y},k,j}(\mathbf{x}_k, \ell_j), \quad (23)$$

or, in block-matrix form,

$$J(\mathbf{q}) = \frac{1}{2} \mathbf{e}(\mathbf{q})^T \mathbf{W}^{-1} \mathbf{e}(\mathbf{q}), \quad (24)$$

where

$$\mathbf{q} = \begin{bmatrix} \mathbf{x} \\ \ell \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} \mathbf{Q} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix}, \quad (25a)$$

$$\mathbf{Q} = \text{diag}(\mathbf{Q}, \dots, \mathbf{Q}), \quad \mathbf{R} = \text{diag}(\mathbf{R}, \dots, \mathbf{R}), \quad (25b)$$

$$\mathbf{e} = \begin{bmatrix} \mathbf{e}_{\mathbf{x}} \\ \mathbf{e}_{\mathbf{y}} \end{bmatrix}, \quad \mathbf{e}_{\mathbf{x}} = \begin{bmatrix} \mathbf{e}_{\mathbf{x},1} \\ \vdots \\ \mathbf{e}_{\mathbf{x},K} \end{bmatrix}, \quad \mathbf{e}_{\mathbf{y}} = \begin{bmatrix} \mathbf{e}_{\mathbf{y},1} \\ \vdots \\ \mathbf{e}_{\mathbf{y},K} \end{bmatrix}, \quad \mathbf{e}_{\mathbf{y},k} = \begin{bmatrix} \mathbf{e}_{\mathbf{y},k,1} \\ \vdots \\ \mathbf{e}_{\mathbf{y},k,V} \end{bmatrix}. \quad (25c)$$

Note that the sparsity pattern in \mathbf{W} implicitly represents a co-visibility graph, since there are only entries corresponding to process priors among adjacent poses, or measurements from poses to visible landmarks. The UKL-SLAM optimization objective is

$$\min_{\mathbf{q}} J(\mathbf{q}). \quad (26)$$

Note that although $J(\mathbf{q})$ is quadratic with respect to \mathbf{e} , it is generally non-quadratic with respect to \mathbf{q} , and thus (26) cannot be solved in one shot. However, we can still use classic Gauss-Newton optimization to efficiently solve (26) iteratively by exploiting sparsity structures. See Appendix B and Appendix C for details on solving (26) with classic Gauss-Newton. The optimal update, $\delta\mathbf{q}$, can be computed in complexity $\mathcal{O}(N^3(V^3 + V^2K))$ when we have more timesteps than landmarks, or $\mathcal{O}(N^3(K^3 + K^2V))$ when we have more landmarks than timesteps. We then update the operating point with

$$\mathbf{x}_{\text{op}} \leftarrow \mathbf{x}_{\text{op}} + \alpha\delta\mathbf{x}, \quad \ell_{\text{op}} \leftarrow \ell_{\text{op}} + \alpha\delta\ell, \quad (27)$$

for a step size α . An appropriate step size can be found using a backtracking line search [5]. Iterating until convergence yields (\mathbf{x}^*, ℓ^*) , or \mathbf{q}^* in batch form.

The distribution of the batch system's solution is $\mathbf{q} \sim \mathcal{N}(\hat{\mathbf{q}}, \hat{\mathbf{P}}_{\mathbf{q}})$, where $\hat{\mathbf{q}} = \begin{bmatrix} \hat{\mathbf{x}} \\ \hat{\ell} \end{bmatrix}$ and $\hat{\mathbf{P}}_{\mathbf{q}} = \begin{bmatrix} \hat{\mathbf{P}}_{\mathbf{x}} & \hat{\mathbf{P}}_{\mathbf{x}\ell} \\ \hat{\mathbf{P}}_{\mathbf{x}\ell}^T & \hat{\mathbf{P}}_{\ell} \end{bmatrix}$. We have the batch mean from the Gauss-Newton solution: $\hat{\mathbf{q}} = \mathbf{q}^*$. We can compute the individual distributions for \mathbf{x}_k and ℓ_j as follows. The means, $\hat{\mathbf{x}}_k$ and $\hat{\ell}_j$, are simply the corresponding blocks from $\hat{\mathbf{x}}$ and $\hat{\ell}$. As a feature of classic Gauss-Newton SLAM, we can also compute the covariances, $\hat{\mathbf{P}}_{\mathbf{x},k}$ and $\hat{\mathbf{P}}_{\ell,j}$, without raising the computation complexity of SLAM. See Appendix C for details on computing covariances. The final output is $\mathbf{x}_k \sim \mathcal{N}(\hat{\mathbf{x}}_k, \hat{\mathbf{P}}_{\mathbf{x},k})$, $\ell_j \sim \mathcal{N}(\hat{\ell}_j, \hat{\mathbf{P}}_{\ell,j})$.

C. Recovering Estimates from Lifted Space

After solving for the lifted means and covariances of the states and landmarks, we now convert them back into the original space. We define the block structures of the means and covariances of states and landmarks in the original space for future reference:

$$\zeta = \begin{bmatrix} \xi \\ \psi \end{bmatrix}, \quad \zeta \sim \mathcal{N}(\hat{\zeta}, \hat{\Sigma}_{\zeta}), \quad (28a)$$

$$\xi \sim \mathcal{N}(\hat{\xi}, \hat{\Sigma}_{\xi}), \quad \psi \sim \mathcal{N}(\hat{\psi}, \hat{\Sigma}_{\psi}), \quad (28b)$$

$$\xi = \begin{bmatrix} \xi_1 \\ \vdots \\ \xi_K \end{bmatrix}, \quad \psi = \begin{bmatrix} \psi_1 \\ \vdots \\ \psi_V \end{bmatrix}, \quad (28c)$$

$$\xi_k \sim \mathcal{N}(\hat{\xi}_k, \hat{\Sigma}_{\xi,k}), \quad \psi_j \sim \mathcal{N}(\hat{\psi}_j, \hat{\Sigma}_{\psi,j}). \quad (28d)$$

In general, recovering ξ_k from \mathbf{x}_k involves a retraction step [29]. However, we will restrict the lifting functions to include the original variables as the top parts of the lifted states later in Section VIII. The recovery procedure for the states then becomes simply picking off the top part of $\hat{\mathbf{x}}_k$ to get $\hat{\xi}_k$, and picking off the top-left block of $\hat{\mathbf{P}}_{\mathbf{x},k}$ to get $\hat{\Sigma}_{\mathbf{x},k}$. The procedure for the landmarks is analogous. See [4] for a recovery procedure for general lifting functions.

D. Other UKL Estimation Problems

We can leverage the flexibility of classic Gauss-Newton SLAM to handle variations to UKL-SLAM, including localization and mapping as special cases. It is possible that $\mathbf{y}_{k,j}$ is missing, meaning the robot does not receive a measurement from ℓ_j at timestep k . Or, it is possible that a pose, \mathbf{x}_k , or a landmark position, ℓ_j , is known. In both cases, we can modify (57) to incorporate missing or known variables without affecting complexity. See Appendix B for more details. If all of the landmarks are known, SLAM reduces to localization (i.e., UKL-Loc). If all of the robot poses are known, SLAM reduces to mapping. In both of these cases, the cost in (24) becomes quadratic in the remaining unknowns, and using the sparse Cholesky solver (see Appendix C) yields the minimum-cost solution after just one iteration.

E. Limitations of UKL-Loc and UKL-SLAM

At first, solving localization and SLAM through unconstrained optimization seems tempting, especially for UKL-Loc whose solution can be found in one shot. However, observe that there are no conditions that enforce the solution to be on the manifold defined by the lifting functions. In the case of UKL-Loc, we automatically enforce that $\mathbf{u}_k \in \mathcal{U}$, $\mathbf{y}_{k,j} \in \mathcal{Y}$, $\ell_j \in \mathcal{L}$, as these quantities are given and are directly lifted at test time. However, there is no guarantee that the minimum-cost solution of (24) satisfies $\mathbf{x}_k^* \in \mathcal{X}$. Similarly, for UKL-SLAM, there is no guarantee that $\mathbf{x}_k^* \in \mathcal{X}$ and $\ell_j^* \in \mathcal{L}$. In both cases, the minimum-cost solution could be very far from the lifting-function manifold. This can be problematic because the trained models are likely poor for governing the behaviour of off-manifold states and landmarks. The states and landmarks within the training data are always on the manifold since they are being lifted from data (11) to their lifted versions in (13). If the minimum-cost solution for (24) happens to be far away from the manifold, the estimates may not be reflective of the training data.

For localization, our experiments suggest that the minimum-cost solution of UKL-Loc tends to be close enough to the manifold when there are frequent measurements. In the simulations in Section X, we see that the UKL-Loc trajectory outputs are nearly as good as those of the model-based localizers. We hypothesize that the measurements are ‘pulling’ the estimates back onto the manifold, although the exact pulling mechanism

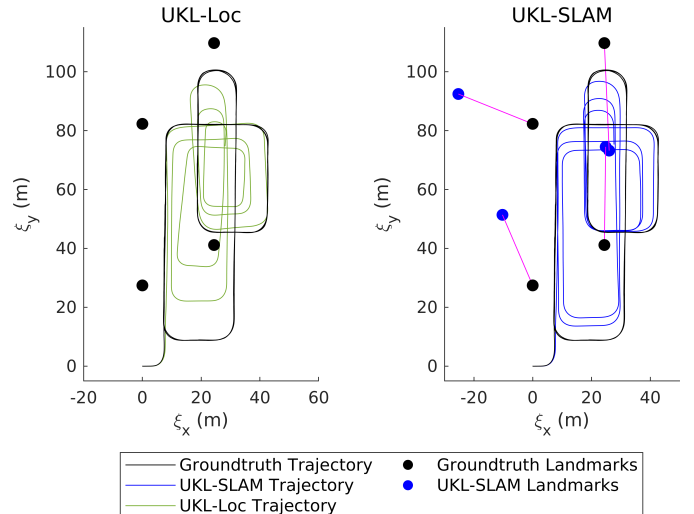


Fig. 2: Visualization of the trajectory output of UKL-Loc (left), and the trajectory and landmark output of UKL-SLAM (right), on Experiment 2 described in Section X-D, showing the estimators’ mean states and mean landmark positions compared to the groundtruth. The pink lines show the landmark correspondences between the groundtruth and the output of UKL-SLAM. We observe that the trajectory estimate of UKL-Loc is poor, which we typically see when the measurements are sporadic. For UKL-SLAM, both the trajectory and the landmark estimates are poor, which we typically see regardless of the regularity of measurements.

is still unclear. When the measurements are sporadic, however, there are gaps in the measurement stream during which the system can drift far away from the manifold from purely dead reckoning. Thus, although unconstrained Koopman localization can work in scenarios where the measurements are regular, such as in [4], it cannot work with sporadic measurements.

On the other hand, the minimum-cost solutions of UKL-SLAM tend to yield very poor trajectory and landmark estimates in general as a result of deviating too much from the manifold. This trend appears to be present irregardless of the type or dimension of the lifting functions used in (4). See Fig. 2 for a visualization of the outputs of UKL-Loc and of UKL-SLAM on a dataset with sporadic measurements.

The manifold deviation is related to a known challenge of finding Koopman-invariant subspaces for Koopman process models [12]. For a noiseless system, a Koopman-invariant subspace is such that any on-manifold state stays on the manifold after passing through the lifted process model. For our case, this means that given lifting functions \mathbf{p}_ξ and \mathbf{p}_μ , the original process model, $\xi_k = \mathbf{f}(\xi_{k-1}, \mu_k)$, can be written exactly as a lifted model, $\mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_k + \mathbf{H}(\mathbf{u}_k \otimes \mathbf{x}_{k-1})$, such that

$$\mathbf{x}_k \in \mathcal{X} \quad \forall \mathbf{x}_{k-1} \in \mathcal{X}, \quad \forall \mathbf{u}_k \in \mathcal{U}. \quad (29)$$

Finding Koopman-invariant subspaces for real-world control problems is challenging and an active area of research [30], [31], [32]. In the literature, the Koopman operator framework is commonly used for problems with short time horizons such as model-predictive control [18], where the manifold deviations are small and can be removed by a retraction step. For example, we could drop the solution back into the original space and then re-lift it at the beginning

of a new time window [21], [29]. However, the solution quality deteriorates with longer time horizons [33]. In our case, ensuring invariance is further complicated with the addition of the measurement model and the addition of lifted landmarks as unknowns. For batch optimization over very long time horizons such as in SLAM, we find that reprojections cannot sufficiently improve the solution quality after converging to an off-manifold solution.

There is some work on converting nonconvex state-estimation problems to high-dimensional convex problems with only linear constraints [13], [14]. However, the conversions assume noiseless systems and ideal, possibly infinite-dimensional Koopman operators, both of which cannot be achieved in practice. As a result, these conversions have yet to be applied on real-world datasets.

In the next section, we solve the same estimation problem for the system (9), with the addition of constraints to enforce the solution to be on the manifold of the lifting functions.

VIII. CONSTRAINED KOOPMAN LINEARIZATION (CKL)

In this section, we present the setup and the solution process for CKL. We present the setup of the optimization problem, the conversion to an SQP, and finally the algorithm for solving the SQP in linear time. We first focus on CKL-SLAM, where we will reuse much of the development for UKL-SLAM. We then show the necessary modifications for other CKL estimation problems including dead reckoning, mapping, and localization (CKL-Loc), followed by a note on reasonable SQP initializations for these problems.

A. CKL-SLAM Problem Setup

For CKL-SLAM, the optimization objective is identical to (26), with the added constraint that the solution must lie on the manifold of the lifting functions. The objective is

$$\begin{aligned} \min_{\mathbf{q}} \quad & J(\mathbf{q}) \\ \text{s.t.} \quad & \mathbf{x}_k \in \mathcal{X}, \quad k = 1, \dots, K, \\ & \ell_j \in \mathcal{L}, \quad j = 1, \dots, V, \end{aligned} \quad (30)$$

where $J(\mathbf{q})$ is defined in (24). In order to simplify the form of the constraints, we enforce that the estimated quantities' lifting functions, $\mathbf{p}_\xi(\cdot)$ and $\mathbf{p}_\ell(\cdot)$, consist of the original quantities stacked on top of the actual lifted features:

$$\mathbf{x}_k = \mathbf{p}_\xi(\xi_k) = \begin{bmatrix} \xi_k \\ \tilde{\mathbf{p}}_\xi(\xi_k) \end{bmatrix} = \begin{bmatrix} \xi_k \\ \tilde{\mathbf{x}}_k \end{bmatrix}, \quad (31a)$$

$$\ell_j = \mathbf{p}_\psi(\psi_j) = \begin{bmatrix} \psi_j \\ \tilde{\mathbf{p}}_\psi(\psi_j) \end{bmatrix} = \begin{bmatrix} \psi_j \\ \tilde{\ell}_j \end{bmatrix}, \quad (31b)$$

where

$$\tilde{\mathbf{p}}_\xi : \mathbb{R}^{N_\xi} \rightarrow \mathbb{R}^{N_x - N_\xi}, \quad \tilde{\mathbf{p}}_\psi : \mathbb{R}^{N_\psi} \rightarrow \mathbb{R}^{N_\ell - N_\psi}. \quad (32)$$

We can write the manifold constraints as equality constraints:

$$\mathbf{x}_k \in \mathcal{X} \Rightarrow \mathbf{h}_x(\mathbf{x}_k) = \tilde{\mathbf{x}}_k - \tilde{\mathbf{p}}_\xi(\xi_k) = \mathbf{0}, \quad (33a)$$

$$\ell_j \in \mathcal{L} \Rightarrow \mathbf{h}_\ell(\ell_j) = \tilde{\ell}_j - \tilde{\mathbf{p}}_\psi(\psi_j) = \mathbf{0}. \quad (33b)$$

The optimization objective in (30) becomes

$$\begin{aligned} \min_{\mathbf{q}} \quad & J(\mathbf{q}) \\ \text{s.t.} \quad & \mathbf{h}(\mathbf{q}) = \mathbf{0}, \end{aligned} \quad (34)$$

where

$$\mathbf{h}(\mathbf{q}) = \begin{bmatrix} \mathbf{h}_x(\mathbf{x}) \\ \mathbf{h}_\ell(\ell) \end{bmatrix}, \quad \mathbf{h}_x(\mathbf{x}) = \begin{bmatrix} \mathbf{h}_x(\mathbf{x}_1) \\ \vdots \\ \mathbf{h}_x(\mathbf{x}_K) \end{bmatrix}, \quad \mathbf{h}_\ell(\ell) = \begin{bmatrix} \mathbf{h}_\ell(\ell_1) \\ \vdots \\ \mathbf{h}_\ell(\ell_V) \end{bmatrix}. \quad (35)$$

This objective is identical to the UKL-SLAM objective in (26), with the added manifold constraints. Note that $\mathbf{h}(\mathbf{q})$ is generally nonlinear, but it has an exploitable structure: each of the blocks in $\mathbf{h}_x(\mathbf{x})$ can be nonlinear but affects only one robot state, and each of the blocks in $\mathbf{h}_\ell(\ell)$ affects only one landmark. This blockwise structure will be especially important for solving the optimization problem efficiently.

B. Solving CKL-SLAM with a Sequential Quadratic Program

We formulate an SQP [5] to solve (34). The Lagrangian is

$$L(\mathbf{q}, \boldsymbol{\lambda}) = J(\mathbf{q}) - \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{q}), \quad (36)$$

where $\boldsymbol{\lambda}$ is the Lagrange multiplier. Given the Lagrangian, we can solve for the SQP's optimal update, $(\delta \mathbf{q}, \delta \boldsymbol{\lambda})$. The solution process involves similar matrix structures as the unconstrained problem in (26). See Appendix D for more details on the SQP formulation for CKL-SLAM. We then update the operating point of the primal variable and the multiplier as

$$\mathbf{q}_{\text{op}} \leftarrow \mathbf{q}_{\text{op}} + \alpha \delta \mathbf{q}, \quad \boldsymbol{\lambda}_{\text{op}} \leftarrow \boldsymbol{\lambda}_{\text{op}} + \alpha \delta \boldsymbol{\lambda}, \quad (37)$$

with an appropriate step size α . With (34) containing the same $J(\mathbf{q})$ as (26) and a blockwise structure on $\mathbf{h}(\mathbf{q})$, we can extend the sparsity exploitation process of UKL-SLAM to CKL-SLAM. See Appendix E for details. With this, the optimal updates can be solved in the same time complexity as the unconstrained problem: $\mathcal{O}(N^3(V^3 + V^2K))$ when we have more timesteps than landmarks, or $\mathcal{O}(N^3(K^3 + K^2V))$ when we have more landmarks than timesteps.

To extract the covariances of the estimates, $\hat{\Sigma}_{\xi,k}$ and $\hat{\Sigma}_{\psi,j}$, we need to take into account the effect of the added constraints. We show in Appendix F that at convergence, the batch covariance of the original variable satisfies

$$\hat{\Sigma}_\zeta^{-1} = \mathbf{S}_\parallel^T \mathbf{F} \mathbf{S}_\parallel, \quad (38)$$

where \mathbf{F} is the Gauss-Newton approximation of the Hessian of the Lagrangian [34, §2], [35, §3.2], and $\mathbf{S}_\parallel = \text{null}(\mathbf{S})$ is a matrix constructed by a basis of the nullspace of \mathbf{S} . We can then use a similar procedure as for UKL-SLAM to extract the required covariances from $\hat{\Sigma}_\zeta^{-1}$. This can be done without raising the computational complexity of SLAM.

C. Other CKL Estimation Problems and SQP Initializations

CKL-SLAM can be easily modified for other estimation problems including dead reckoning, localization (CKL-Loc), and mapping. We would make similar adjustments to the cost as described in Section VII-D for UKL-SLAM, and also analogous adjustments to the constraints. For all of these

problems, the matrix sparsity structures are preserved, and we can still use the nullspace method to efficiently solve the SQP.

In the presence of nonlinear constraints, the problems of dead reckoning, mapping, localization, and SLAM are all nonlinear optimization problems, and good initial points are often required to avoid convergence to local minima. For dead reckoning, mapping, and localization, we acquire an initial point by dropping the constraints (i.e., the UKL solution). For SLAM, we initialize the robot trajectory by dead reckoning, then initialize the landmarks by mapping from the dead-reckoned trajectory.

IX. REDUCED CONSTRAINED KOOPMAN LINEARIZATION (RCKL)

With the manifold constraints introduced, CKL-SLAM yields much better results than UKL-SLAM. However, as we will explain, CKL is sensitive to poorly fit process models of the lifted features. As a result, CKL's outputs are often still less accurate than those of the model-based methods under experimental evaluation. We now introduce RCKL, a framework that further improves upon CKL by removing the portion of the learned process model that tends to be poorly fit. We first present the framework, and then discuss our hypothesis for its improvement over CKL.

A. Reducing the Koopman Process Model

We analyze the Koopman process model in (9a) with the form of the lifting function, $\mathbf{p}_\xi(\cdot)$, enforced in (31a). We break down \mathbf{A} , \mathbf{B} , and \mathbf{H} into two components:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_\xi \\ \tilde{\mathbf{A}} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{B}_\xi \\ \tilde{\mathbf{B}} \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} \mathbf{H}_\xi \\ \tilde{\mathbf{H}} \end{bmatrix}, \quad (39)$$

where

$$\mathbf{A}_\xi \in \mathbb{R}^{N_\xi \times N_x}, \quad \tilde{\mathbf{A}} \in \mathbb{R}^{(N_x - N_\xi) \times N_x}, \quad (40a)$$

$$\mathbf{B}_\xi \in \mathbb{R}^{N_\xi \times N_u}, \quad \tilde{\mathbf{B}} \in \mathbb{R}^{(N_x - N_\xi) \times N_u}, \quad (40b)$$

$$\mathbf{H}_\xi \in \mathbb{R}^{N_\xi \times N_x N_u}, \quad \tilde{\mathbf{H}} \in \mathbb{R}^{(N_x - N_\xi) \times N_x N_u}. \quad (40c)$$

Then, the process model (9a), along with the form of $\mathbf{p}_\xi(\xi_k)$ in (31a), yields

$$\begin{cases} \xi_k = \mathbf{A}_\xi \mathbf{x}_{k-1} + \mathbf{B}_\xi \mathbf{u}_k + \mathbf{H}_\xi (\mathbf{u}_k \otimes \mathbf{x}_{k-1}) + \mathbf{w}_{\xi,k}, & (41a) \\ \tilde{\mathbf{x}}_k = \tilde{\mathbf{A}} \mathbf{x}_{k-1} + \tilde{\mathbf{B}} \mathbf{u}_k + \tilde{\mathbf{H}} (\mathbf{u}_k \otimes \mathbf{x}_{k-1}) + \tilde{\mathbf{w}}, & (41b) \end{cases}$$

where $\mathbf{w}_k = \begin{bmatrix} \mathbf{w}_{\xi,k} \\ \tilde{\mathbf{w}}_k \end{bmatrix} \sim \mathcal{N} \left(\mathbf{0}, \begin{bmatrix} \mathbf{Q}_\xi & \mathbf{Q}_{\xi\tilde{x}} \\ \mathbf{Q}_{\xi\tilde{x}}^T & \mathbf{Q}_{\tilde{x}} \end{bmatrix} \right)$.

If there are no constraints in place, both models are necessary to fully determine \mathbf{x}_k . With the manifold constraints introduced, however, (41a) alone is sufficient to determine \mathbf{x}_k , since the constraint $\mathbf{h}_x(\mathbf{x}_k) = \mathbf{0}$ in (33a) enforces that $\tilde{\mathbf{x}}_k = \tilde{\mathbf{p}}_\xi(\xi_k)$. This means that when (41b) is a poor model, we can use (41a) to only determine the original state variables, and let the lifting-function constraints determine the lifted features.

To clarify, the lifted features are still used in the process model of (41a) as part of the previous state, \mathbf{x}_{k-1} , but just not determined by the model for the current state. Note that reducing the model does not break the theoretical justifications previously mentioned in Section V, including the fact that the

conversion of a control-affine model to a lifted bilinear model is exact in the noiseless case.

To train the reduced process model, we train for $\mathbf{A}_\xi, \mathbf{B}_\xi, \mathbf{H}_\xi, \mathbf{Q}_\xi$. The procedure is similar to the one described in Section VI, except that the transitioned state is not lifted in the process model. That is, the model in (12) becomes

$$\xi_i = \mathbf{A}_\xi \hat{\mathbf{x}}_i + \mathbf{B}_\xi \mathbf{u}_i + \mathbf{H}_\xi (\mathbf{u}_i \otimes \hat{\mathbf{x}}_i) + \mathbf{w}_i, \quad (42a)$$

$$\mathbf{y}_{k,j} = \mathbf{C}(\ell_j \otimes \mathbf{x}_k) + \mathbf{n}_{i,j}, \quad (42b)$$

where \mathbf{x}_i is replaced with ξ_i in the process model. After making similar modifications to the loss function in (16), the solutions for $\mathbf{A}_\xi, \mathbf{B}_\xi, \mathbf{H}_\xi, \mathbf{Q}_\xi$ can be found with

$$\mathbf{V} = \mathbf{U} \odot \hat{\mathbf{X}}, \quad \mathbf{J} = \Xi - \mathbf{A}_\xi \hat{\mathbf{X}} - \mathbf{B}_\xi \mathbf{U} - \mathbf{H}_\xi \mathbf{V}, \quad (43a)$$

$$\begin{bmatrix} \hat{\mathbf{X}} \hat{\mathbf{X}}^T + P_{\tau_A} \mathbf{1} & \hat{\mathbf{X}} \mathbf{U}^T & \hat{\mathbf{X}} \mathbf{V}^T \\ \mathbf{U} \hat{\mathbf{X}}^T & \mathbf{U} \mathbf{U}^T + P_{\tau_B} \mathbf{1} & \mathbf{U} \mathbf{V}^T \\ \mathbf{V} \hat{\mathbf{X}}^T & \mathbf{V} \mathbf{U}^T & \mathbf{V} \mathbf{V}^T + P_{\tau_H} \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{A}_\xi^T \\ \mathbf{B}_\xi^T \\ \mathbf{H}_\xi^T \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{X}} \Xi^T \\ \mathbf{U} \Xi^T \\ \mathbf{V} \Xi^T \end{bmatrix}, \quad (43b)$$

$$\mathbf{Q}_\xi = \frac{1}{P} \mathbf{J} \mathbf{J}^T + \tau_A \mathbf{A}_\xi \mathbf{A}_\xi^T + \tau_B \mathbf{B}_\xi \mathbf{B}_\xi^T + \tau_H \mathbf{H}_\xi \mathbf{H}_\xi^T + \tau_Q \mathbf{1}. \quad (43c)$$

At test time, we modify the time-varying quantities defined in (19),

$$\mathbf{A}_{\xi,k-1} = \mathbf{A}_\xi + \mathbf{H}_\xi (\mathbf{u}_k \otimes \mathbf{1}), \quad \mathbf{v}_k = \mathbf{B}_\xi \mathbf{u}_k, \quad (44)$$

and modify error functions in (22) to be

$$\mathbf{e}_{\mathbf{v},k}(\mathbf{x}_k) = (\mathbf{A}_{\xi,k-1} \mathbf{x}_{k-1} + \mathbf{v}_k) - \xi_k, \quad (45a)$$

$$\mathbf{e}_{\mathbf{y},k,j}(\mathbf{x}_k, \ell_j) = \mathbf{C}(\ell_j \otimes \mathbf{x}_k) - \mathbf{y}_{k,j}, \quad (45b)$$

where the pose error is modified to weigh only the ξ_k component of \mathbf{x}_k , while the measurement error is still using the full set of features in \mathbf{x}_k . The cost function is modified from (23) to be

$$\begin{aligned} J(\mathbf{x}, \ell) = & \frac{1}{2} \sum_{k=1}^K \mathbf{e}_{\mathbf{v},k}(\mathbf{x}_k)^T \mathbf{Q}_\xi^{-1} \mathbf{e}_{\mathbf{v},k}(\mathbf{x}_k) \\ & + \frac{1}{2} \sum_{k=1}^K \sum_{j=1}^V \mathbf{e}_{\mathbf{y},k,j}(\mathbf{x}_k, \ell_j)^T \mathbf{R}^{-1} \mathbf{e}_{\mathbf{y},k,j}(\mathbf{x}_k, \ell_j), \end{aligned} \quad (46)$$

which can be written in the familiar block-matrix form of $J(\mathbf{q}) = \frac{1}{2} \mathbf{e}(\mathbf{q})^T \mathbf{W}^{-1} \mathbf{e}(\mathbf{q})$ in (24). This means that the RCKL-SLAM objective is the same form as (35), the CKL-SLAM objective. To solve for \mathbf{q} , we can follow the same procedure as Section VIII-B. The complexity of the SQP iterations are the same, possibly with a lower coefficient owing to the smaller process model.

B. Theoretical Justification of RCKL

For the experimental scenarios presented Section X, RCKL's process model always performs better than CKL's model. See Fig. 3 for a visual example of noiseless dead reckoning with UKL, CKL, and RCKL, where RCKL's output is much closer to the groundtruth trajectory. We now discuss our hypothesis for this improvement in performance.

The main difference between RCKL and CKL is that RCKL avoids fitting a model for the evolution of the lifted features. A poor process model of the features would negatively affect

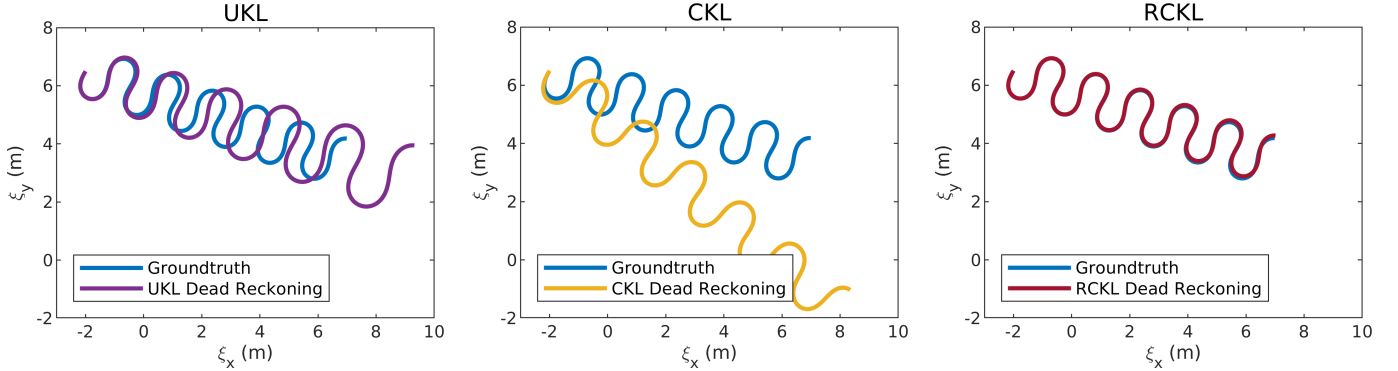


Fig. 3: *UKL vs. CKL vs. RCKL in noiseless dead reckoning.* We train the three estimators on noiseless data corresponding to Simulation 1 described in Section X-A1, then compare their dead-reckoning outputs on a noiseless test trajectory. UKL’s output is on top of groundtruth at first but eventually drifts off as the states deviate from the feature manifold. CKL’s output remains on the feature manifold, but solution is worsened by the poor process models on the features. Meanwhile, RCKL stays on the manifold and its output corresponds to the groundtruth almost exactly.

the whole system through the constraints. In our experiments, we see that when a model contains all of the lifting functions required to fit the system, adding extraneous lifting functions to CKL degrades its performance. Even when $\tilde{\mathbf{x}}_{k-1}$ is essential for determining ξ_k in (41a), its own evolution to $\tilde{\mathbf{x}}_k$ in (41b) may be poor, which would affect ξ_k through the constraint $\mathbf{h}_x(\mathbf{x}_k) = \tilde{\mathbf{x}}_k - \tilde{\mathbf{p}}_\xi(\xi_k) = \mathbf{0}$.

This concept is related to how Koopman invariance (29) is hard to satisfy especially for the process models of the lifted features. See [12, p. 263] for an example of where the Koopman representation of even a simple system becomes intractable when using polynomial features. The higher-order polynomials added require even higher-order polynomials to fit their own process model, ad infinitum.

After selecting the features from a large pool of possible lifting functions, we find that the linear process models on the features tend to be poorly fit. RCKL, in contrast, fits a linear model on only the original variables and relies on the nonlinear manifold constraints for the evolution of the features. Assuming that we have enough training data to avoid overfitting, adding more lifting functions in RCKL would only add more expressiveness for fitting the model.

Note that RCKL’s reduced model follows the same spirit as the original sparse identification of nonlinear dynamics (SINDy) algorithm [36]. For a nonlinear system, $\xi_k = \mathbf{f}(\xi_{k-1})$, SINDy optimizes for a sparse \mathbf{A} and a low-dimensional $\mathbf{p}_\xi(\cdot)$ such that the model can be written as $\xi_k = \mathbf{A}\mathbf{p}_\xi(\xi_{k-1})$. That is, SINDy identifies the nonlinear functions in $\mathbf{p}_\xi(\cdot)$ necessary to reconstruct the original system. This form is similar to our reduced process model in (42a) with \mathbf{p}_ξ as the lifting function. However, SINDy’s nonlinear model is not originally compatible with Koopman methods (without constraints). Instead, many Koopman methods use a modified version of SINDy to reconstruct $\mathbf{p}_\xi(\xi_k) = \mathbf{A}\mathbf{p}_\xi(\xi_{k-1})$ [25], which is essentially finding the full process model. With the manifold constraints introduced, the reduced model is once again possible for Koopman systems.

One potential drawback of RCKL is that for some systems, the process model of some features could be simpler to construct than that of the original variables. In this case, a

hybrid method could be used. As this is likely very problem-specific, we leave the investigation of this idea for future work.

X. EXPERIMENTS AND RESULTS

We evaluate the performance of the Koopman estimators for localization and SLAM in two simulation scenarios and on two real-world datasets. We investigate the performances of UKL, CKL, and RCKL in simulation, and focus on RCKL on the datasets, owing to its superior performance in simulation. We compare our results with those of a classic model-based nonlinear batch estimator optimized using Gauss-Newton [24, §8/9]. We term this estimator “MB” for model-based. To demonstrate the advantages of our data-driven approach, we also compare with a classic estimator where the model parameters are imperfect, and we term this estimator “MBI” for model-based imperfect. We will show that our Koopman estimators can outperform MBI by learning the model parameters. Both the Koopman estimators and the model-based estimators follow the initialization procedure outlined in Section VIII-C: initializing with dead reckoning for localization, and initializing with mapping from dead reckoning for SLAM.

To evaluate accuracy, we compute the root-mean-squared-error (RMSE) of an estimator’s mean output. To evaluate consistency, we compute the normalized trajectory-level Mahalanobis distance of an estimator’s mean and covariance outputs. This Mahalanobis distance measures the consistency of the entire batch solution and is computed with $\sqrt{(\delta\zeta)^T \Sigma^{-1} (\delta\zeta) / N}$, where $\delta\zeta$ is the error of the estimator’s mean outputs compared to the groundtruth, Σ^{-1} is the inverse covariance output, and N is the degrees of freedom of the system. An accurate estimator has an RMSE close to 0, and a consistent estimator has a Mahalanobis distance close to 1. For evaluating SLAM, we first perform an alignment step where the groundtruth is aligned to the output trajectory through a rigid transformation before computing the RMSEs and the Mahalanobis distances. This is because SLAM is used to derive a relative map between the trajectory and the landmarks, and the global SLAM error within the frame of the first pose is usually of lesser interest.

In the experiments below, the main lifting functions used are squared-exponential Random Fourier Features (SERFFs) [37],

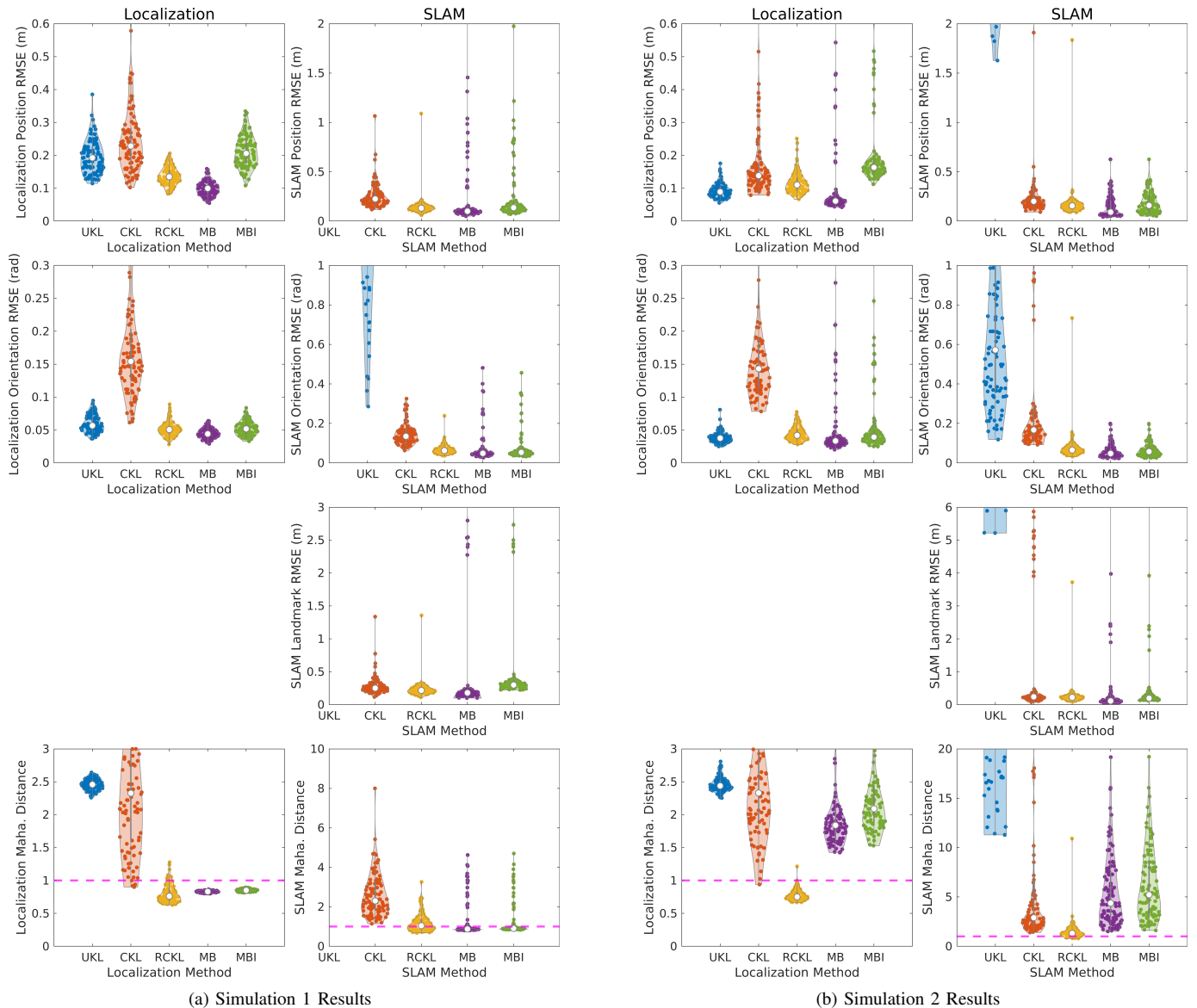


Fig. 4: Localization and SLAM results for Simulation 1 (left) and Simulation 2 (right). RMSEs and Mahalanobis distances for localization (left plots) and SLAM (right plots) for UKL, CKL, RCKL, MB (model-based with correct μ), and MBI (model-based with imperfect μ). In both simulations, RCKL has slightly higher RMSEs than MB but encounters local minima less frequently, and it has lower RMSEs than MBI. UKL-SLAM RMSEs are off the charts, and CKL-SLAM RMSEs are more reasonable than UKL-SLAM but still higher than RCKL-SLAM, demonstrating the necessity of the constraints and the reduced process model in RCKL. RCKL is also more consistent than MB and MBI.

which take in any number of vector-space inputs and generate a user-specified number of sinusoidal functions. SERFFs are common lifting functions for Koopman methods used for their experimentally determined high performance [4], [38], and for their connection to the squared-exponential kernel [4], [39]. The formulas for generating the SERFFs can be found in [37]. Note that the generation of SERFFs involve picking hyperparameter values that can be tuned based on data if necessary. Below, we use the notation $\mathbf{r}_x(\mathbf{x})$ to denote the SERFFs for input \mathbf{x} .

A. Simulation Setup

For both simulation scenarios, the setup consists of a robot driving in a 2D plane, receiving measurements from landmarks scattered around the plane. We evaluate localization and SLAM on 100 test instances, each with a 1000-timestep trajectory and

10 landmarks. The measurements are scaled by a constant factor of $\mu = 1.05$, which our Koopman estimators will learn from data. We compare the performance of the Koopman estimators against MB, whose measurement model uses the correct value of $\mu = 1.05$, and MBI, whose measurement model uses $\mu = 1$. To clarify, MB and MBI are using the same raw data, but they are doing estimation with slightly different prior measurement models, thus affecting their solutions. See [24, §8/9] for how the prior measurement models are used for computing costs and Jacobians in classic Gauss-Newton estimation.

1) Simulation 1: Unicycle Model, Range Measurements

The inputs for the unicycle model [40] are the translational and rotation speeds of the robot, and the measurements are distances. For timestep k and landmark j , the robot state, input,

landmark position, and measurement are, respectively,

$$\xi_k = \begin{bmatrix} \xi_{x,k} \\ \xi_{y,k} \\ \xi_{\theta,k} \end{bmatrix}, \quad \nu_k = \begin{bmatrix} u_k \\ \omega_k \end{bmatrix}, \quad \psi_j = \begin{bmatrix} \psi_{x,j} \\ \psi_{y,j} \end{bmatrix}, \quad \gamma_{k,j} = \mu r_{k,j}, \quad (47)$$

where $(\xi_{x,k}, \xi_{y,k})$ is the robot's global position, $\xi_{\theta,k}$ is the robot's global orientation, u_k is the robot's linear speed, ω_k is its angular speed, $(\psi_{x,j}, \psi_{y,j})$ is the landmark's global position, $r_{k,j}$ is the range measurement from the robot to the j th landmark, and μ is the constant scaling factor. Contrary to the assumption made in (31), the state is not within a vector space since it includes orientation as a circular quantity. As such, we convert it to an augmented state, ξ'_k , with the substitutions $\xi_{\cos \theta, k} = \cos \xi_{\theta, k}$ and $\xi_{\sin \theta, k} = \sin \xi_{\theta, k}$. For CKL and RCKL, we introduce an additional state constraint, $h_\xi(\xi'_k) = \xi_{\cos \theta, k}^2 + \xi_{\sin \theta, k}^2 - 1 = 0$. See Appendix G for more details on handling the orientation. We used SERFFs for the state, the landmark position, the measurements, and input:

$$\xi'_k = \begin{bmatrix} \xi_{x,k} \\ \xi_{y,k} \\ \xi_{\cos \theta, k} \\ \xi_{\sin \theta, k} \end{bmatrix}, \quad \mathbf{p}_{\xi'}(\xi'_k) = \begin{bmatrix} \xi'_k \\ \mathbf{r}_{x,y}(\xi_{x,k}, \xi_{y,k}) \\ \mathbf{r}_{\cos \theta, \sin \theta}(\xi_{\cos \theta, k}, \xi_{\sin \theta, k}) \\ \mathbf{r}_{x, \cos \theta}(\xi_{x,k}, \xi_{\cos \theta, k}) \\ \mathbf{r}_{x, \sin \theta}(\xi_{x,k}, \xi_{\sin \theta, k}) \\ \mathbf{r}_{y, \cos \theta}(\xi_{y,k}, \xi_{\cos \theta, k}) \\ \mathbf{r}_{y, \sin \theta}(\xi_{y,k}, \xi_{\sin \theta, k}) \end{bmatrix}, \quad (48a)$$

$$\mathbf{p}_\psi(\psi_j) = \begin{bmatrix} \psi_j \\ \mathbf{r}_\psi(\psi_j) \end{bmatrix}, \quad \mathbf{p}_\gamma(\gamma_{k,j}) = \begin{bmatrix} \gamma_{k,j} \\ \mathbf{r}_\gamma(\gamma_{k,j}) \end{bmatrix}, \quad \mathbf{p}_\nu(\nu_k) = \begin{bmatrix} \nu_k \\ \mathbf{r}_\nu(\nu_k) \end{bmatrix}, \quad (48b)$$

where for the state, SERFFs are constructed on all possible pairs of variables in ξ'_k rather than constructing $\mathbf{r}_{\xi'}(\xi_{x,k}, \xi_{y,k}, \xi_{\cos \theta, k}, \xi_{\sin \theta, k})$. We find this form to be sufficient for our environment, without the burden of a vastly high-dimensional lifted state and requiring much more training data to avoid overfitting.

2) Simulation 2: Bicycle Model, Range-Squared Measurements

The setup is the same as for Simulation 1, except that the robot has a bicycle process model [41] and the measurement is the squared range (to show the generalizability of our data-driven methods):

$$\nu_k = \begin{bmatrix} u_k \\ \phi_k \end{bmatrix}, \quad \gamma_{k,j} = \mu r_{k,j}^2, \quad (49)$$

where u_k is still the linear speed but ϕ_k is the steering angle of the bicycle's front axle. The lifting functions used are in the same form as in (48) except for the input, for which we account for the circular input of ϕ_k by using

$$\mathbf{p}_\nu(\nu_k) = \begin{bmatrix} u_k \\ \cos \phi_k \\ \sin \phi_k \\ \mathbf{r}_{u, \cos \phi, \sin \phi}(u_k, \cos \phi_k, \sin \phi_k) \end{bmatrix}. \quad (50)$$



Fig. 5: Setup for Experiment 1. A wheeled robot drives around 17 cylindrical landmarks (2 are not visible in this photo) in an indoor environment. It logs wheel odometry and measures the range to its surrounding landmarks using a laser rangefinder. The landmarks for testing are highlighted in red.

B. Simulation Results Discussion

The results for Simulation 1 and Simulation 2 are shown in Fig. 4. We see that MB tends to have the lowest errors for localization and SLAM, whereas RCKL has similar or slightly higher errors. This outcome is expected since the models and the noise distributions are perfectly known for MB, whereas the Koopman estimators have learned the model through noisy data. However, RCKL has similar or better accuracy compared to MBI, demonstrating the advantages of the data-driven approach. In addition, while the model-based estimators, MB and MBI, encounter local minima in both localization and SLAM, RCKL appears to encounter local minima much less frequently.¹ This suggests that another potential benefit of reformulating nonlinear estimation problems in a lifted space is improved convergence properties.²

For the other Koopman estimators, we see that UKL-Loc is viable in our environment where a measurement is received from every landmark at every timestep. However, UKL-SLAM is not at all viable, and only becomes viable when the constraints are added to yield CKL-SLAM. CKL still has higher errors than the model-based estimators, MB and MBI. It only become comparable to MB for RCKL, that is, after the reduced process model is employed.

C. Experiment 1: Indoor Navigation with Laser Rangefinder

The setup consists of a wheeled robot driving around in a 2D indoor environment, with 17 tubes scattered throughout the space to act as landmarks (see Fig. 5). The robot has an

¹We have verified the optimality of our solutions by running our optimization problems two times: the first time is after following our standard initialization procedure described in Section VIII-C, and the second time is after initializing at groundtruth. Through this procedure, we can verify that a local minimum has occurred when the first solution (standard initialization) has a higher cost than the second solution (groundtruth initialization). Global optimality is harder to verify, but we assume that the groundtruth initialization converges near the global minimum for our considered noise levels. Then, a global minimum has likely occurred when the first and second solution has the same cost.

²We chose not to compare to globally optimal estimation methods because they tend to be prohibitively expensive for even moderately sized problems [42]. If desired, one could attempt to add a global optimality certificate [43] to either MB or RCKL.

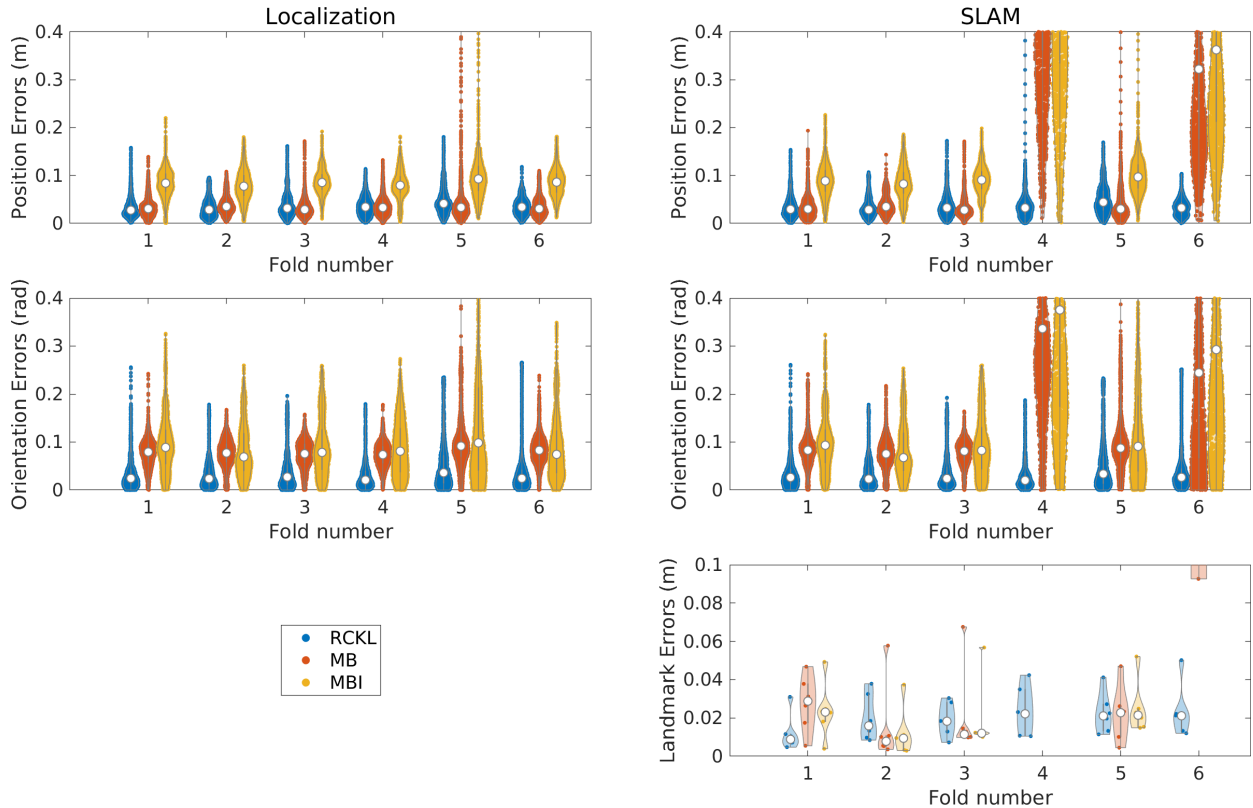


Fig. 6: *Localization and SLAM errors for Experiment 1*: position and orientation errors of the mean trajectory and the mean landmark outputs of RCKL, MB, and MBI (whose robot-to-rangefinder distance is offset by 10 cm), on the 6 dataset folds. For localization, RCKL has similar position errors and generally lower orientation errors compared to MB. Against MBI, RCKL has lower position and orientation errors, demonstrating the advantage of the data-driven approach. These trends are also present for SLAM on folds 1, 2, 3, and 5. On folds 4 and 6, the model-based SLAM errors are much higher as a result of convergence to local minima (see Fig. 8 for a visualization).

odometer to measure its translational and rotational speed, and uses a laser rangefinder to measure its range to the cylindrical landmarks. The groundtruth positions of the robot are recorded using a Vicon motion capture system. All data are logged at 10 Hz. There are approximately 4 visible landmarks at any given time. The robot state, input, landmark position, and measurement are in the same form as in (47), where we adopt the common practice of using interoceptive measurements as inputs in the process model [44]. We use the lifting functions given in (48).

To empirically evaluate RCKL, we split the entire 20-minute dataset into training data and testing data. We use $5/6$ of the trajectories for training, where measurements derived from only 11 of the 17 landmarks are used to train the measurement model. We test on the remaining $1/6$ trajectories using measurements derived from only the 6 remaining landmarks. We assume that there are no factors such as slippage that would cause the robot to drive differently in one area of the room compared to another, and that the measurements are dependent only on the landmark positions relative to the robot poses. Thus, we perform data augmentation by adding translational and rotational transformations of the original training data to the training set.

We do 6-fold cross-validation by repeating the training/testing procedure for different sections of the dataset’s trajectories, and compare the results of RCKL to MB. To demonstrate the advantages of the model-free framework, we also compare

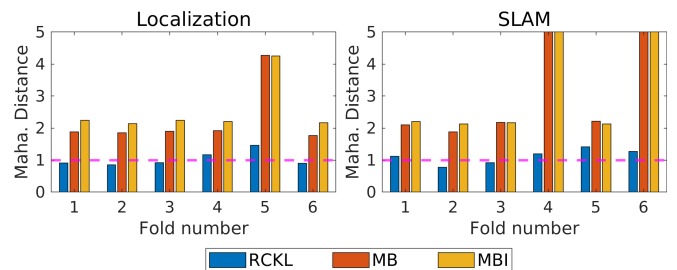


Fig. 7: *Mahalanobis distances of localization and SLAM for Experiment 1*: Mahalanobis distances of the outputs of RCKL, MB, and MBI (whose robot-to-rangefinder distance is offset by 10 cm), on the 6 folds of the dataset. For localization, RCKL’s Mahalanobis distances are close to 1, signifying that RCKL-Loc is consistent. MB’s Mahalanobis distances are slightly higher than 1, signifying that its estimates are slightly overconfident, and MBI is similarly to marginally more overconfident than MB. For localization, MB and MBI are more overconfident in fold 5 than in the other folds as a result of converging to local minima. These trends are also present for SLAM on folds 1, 2, 3, and 5. On folds 4 and 6, the estimates of the model-based estimators, MB and MBI, are way overconfident as a result of convergence to local minima.

these results with MBI, whose robot-to-rangefinder distance in the measurement model is offset by 10 cm with respect to the generated measurements. Again, MB and MBI are using the same raw measurements but different prior models. The errors for RCKL, MB, and MBI are shown in Fig. 6, and their respective Mahalanobis distances are shown in Fig. 7.

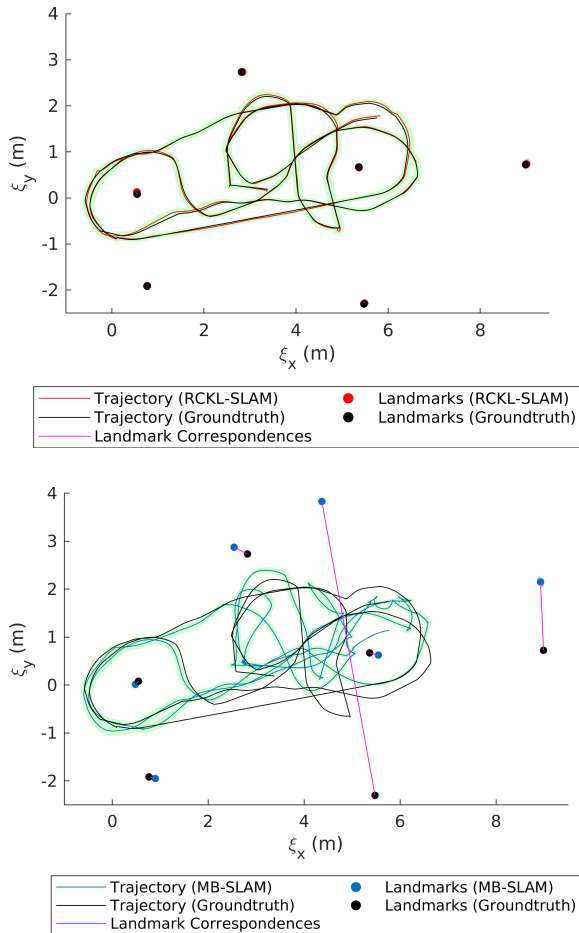


Fig. 8: Visualization of SLAM output of Experiment 1 fold 6 for RCKL-SLAM (top) and MB-SLAM (bottom), showing the estimators’ mean states and mean landmark positions compared to the groundtruth. The green regions and the grey regions are, respectively, the 3σ covariances of the trajectory and of the landmarks, though the 3σ bounds of the landmarks are barely visible. The pink lines show the landmark correspondences between the groundtruth and the estimators’ outputs. RCKL-SLAM’s trajectories and landmarks are close to the groundtruth and are generally within the estimated 3σ bounds. On the other hand, MB-SLAM has converged to a local minimum with one landmark on the opposite side of the trajectories, and its estimate is far from being within 3σ bounds of the groundtruth.

In Fig. 6, we see that for both localization and SLAM, RCKL has similar error levels as MB but outperforms MBI. The Mahalanobis distances in Fig. 7 show that RCKL is fairly consistent. Meanwhile, the model-based estimators are overconfident, and even more overconfident with the model offset. These results suggest that RCKL has learned a more accurate and more consistent model than MB and especially MBI. While RCKL appears to have not encountered any local minima in localization or SLAM, the model-based estimators have encountered local minima in 1 of the 6 folds for localization, and 2 of the 6 folds for SLAM (see Fig. 8). This suggests that RCKL has better convergence properties.

D. Experiment 2: Golf Cart with RFID Measurements

For this experiment, we train on Dataset A3 and test on 3000 steps of Dataset A1 of [45]. The setup for these two

TABLE I: Ablation results of Experiment 2 for localization (top) and SLAM (bottom). The first row corresponds to RCKL, the second row corresponds to MB, the third row corresponds to RCKL with MB’s measurement model, and the fourth row corresponds to RCKL with MB’s process model. The fifth row is MBI, the model-based estimator whose measurement model has imperfect scaling factors. For both localization and SLAM, RCKL has similar or lower RMSEs than MB, and a more consistent Mahalanobis distance. RCKL is especially better than MBI, demonstrating the advantage of the data-driven method.

Localization Method	Position RMSE (m)	Orientation RMSE (rad)	Maha. distance
RCKL Process, RCKL Measurement	0.592	0.0246	0.834
MB Process, MB Measurement	0.768	0.0339	6.167
RCKL Process, MB Measurement	0.790	0.0359	1.092
MB Process, RCKL Measurement	0.582	0.0253	6.161
MBI (imperfect model)	8.187	0.1249	6.422

SLAM Method	Position RMSE (m)	Orientation RMSE (rad)	Landmark RMSE (m)	Maha. distance
RCKL Process, RCKL Measurement	0.552	0.0293	1.073	0.865
MB Process, MB Measurement	0.495	0.0280	1.233	6.154
RCKL Process, MB Measurement	0.553	0.0289	1.298	0.971
MB Process, RCKL Measurement	1.781	0.0699	2.307	6.956
MBI (imperfect model)	1.428	0.0638	7.773	6.240

datasets consists of a cart driving on a golf course. The robot is equipped with a transponder with four radio-frequency (RF) antennae mounted on the four corners of the robot. As it moves, it measures ranges to RF tags placed on top of ten traffic cones that are scattered around the course. The range measurements are sporadic, occurring only at certain points on the trajectory. The measurements for each transponder contain a rather large scaling factor, corresponding to μ in (47), of around 1.2. The groundtruth positions of the robot and of the cones are found with GPS. The control input consists of linear and angular velocities found with a fibre-optic gyro and wheel encoders. With just dead reckoning, the test trajectory drifts in orientation owing to repeated turns in the same direction [45]. This drift will be corrected by localization or SLAM.

Similar to Experiment 1, the robot state, input, landmark position, and measurement are in the same form as in (47), and the same form of lifting functions are used as in (48). The landmark locations in A1 and A3 are the same. As such, we train on A3 with measurements from only 5 landmarks, and test on A1 with measurements from the 4 remaining landmarks, ignoring the one landmark with no measurements. We treat each RFID tag as an independent measurement model, so there are three sets of $\{\mathbf{C}, \mathbf{R}\}$ to train, ignoring the one tag with only two measurements. We do a similar data augmentation procedure as Experiment 1, adding translational and rotational transformations on the original training data into the training set. We compare against MB, whose model contains the correct scaling factors, and against MBI, whose model uses a scaling factor of 1. The error plots for RCKL and

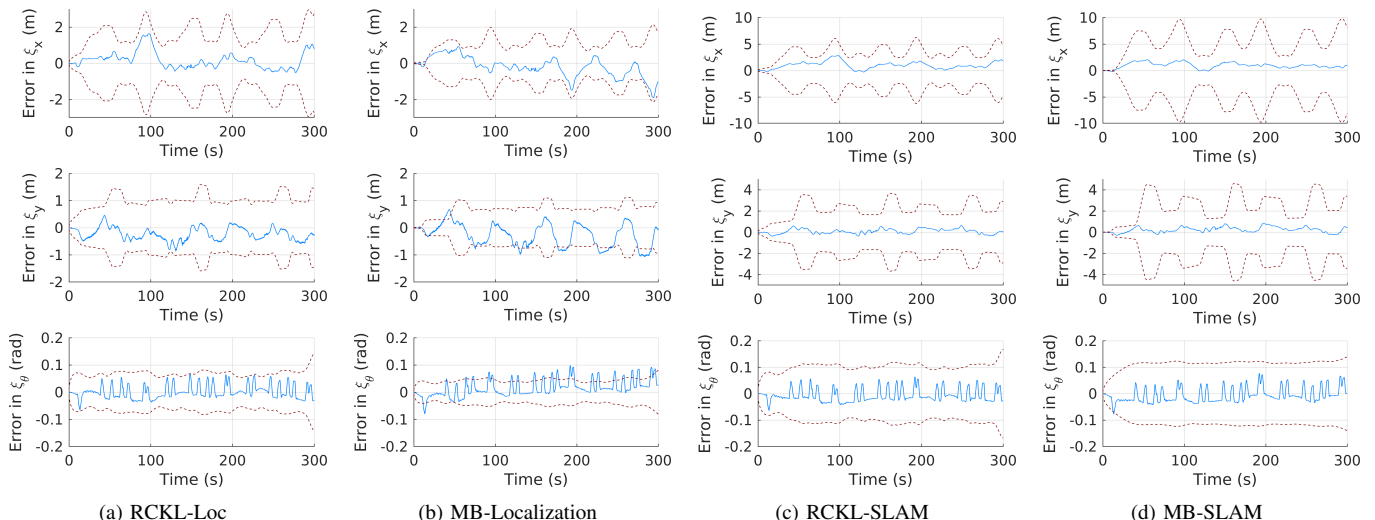


Fig. 9: Error plots for Experiment 2, comparing RCKL-Loc with MB-Localization (left two plots) and comparing RCKL-SLAM with MB-SLAM (right two plots). The blue lines represent the errors of the estimated trajectories, and the red envelopes represent the estimated 3σ covariance bounds. For both localization and SLAM, RCKL’s errors are similar to or smaller than those of MB. RCKL’s errors are bounded by the 3σ bounds, while MB’s errors sometimes exceeds the 3σ bounds especially in localization. The shapes of the errors and the covariance envelopes of RCKL are similar to those of MB, suggesting that RCKL’s outputs reflect the model reasonably well.

the model-based estimators are shown in Fig. 9. A visualization of the outputs of RCKL-Loc and RCKL-SLAM can be seen in Fig. 1. As an ablation study, we also perform estimation with parts of RCKL’s learned models swapped out with MB’s prior models. The RMSEs and Mahalanobis distances of the above estimators are shown in Table I.

Fig. 9 qualitatively validates RCKL, since the errors and covariances of RCKL appear similar to those of MB. Quantitatively, we see from Table I that RCKL has similar or better accuracy and consistency than MB in both localization and SLAM. This suggests that RCKL’s learned models are better than MB’s prior models. The ablation results in Table I also show that swapping to RCKL’s measurement model reduces more error than swapping to RCKL’s process model. This suggests that RCKL’s measurement model is especially better than MB’s measurement model, while the two process models are of similar quality. Finally, RCKL is vastly better than MBI, demonstrating the advantage of the data-driven framework.

E. Computation Time

We briefly discuss the computation time of RCKL compared with that of the model-based estimators. It is difficult to compare the convergence speed of RCKL vs. MB based on cost tolerances since the two algorithms have vastly different cost functions. Thus, we instead view convergence as being sufficiently close to the groundtruth. In this sense, RCKL takes 1-3 times longer to converge than MB and MBI, depending on the environment setup. As an example, we compare the computation time of CPU-based implementations of RCKL and MB on an Intel Core i7-9750H Processor. We show in Fig. 10 the estimator runtime for the folds where both estimators converged near the groundtruth in Experiment 1. Each fold contains 200 seconds of data, consisting of 2000 timesteps (and inputs) and about 8000 range measurements. As seen in Fig. 10, both estimators converge near the groundtruth

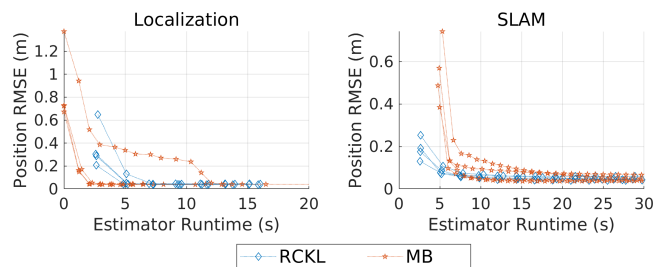


Fig. 10: Position RMSE vs. estimator runtime for Experiment 1 for folds 1, 2, 3, 4, and 6 for localization, and folds 1, 2, 3, and 5 for SLAM, ignoring the folds where MB converges to a local minimum. For both RCKL and MB, markers are placed at every iteration, and the first markers are placed at the end of their initialization procedures. On all of these 200-second trajectories, both estimators converged to near the groundtruth within 30 seconds.

in less than 30 seconds. RCKL has the same Big-O inference complexity as the model-based estimators for both localization and SLAM, scaling linearly with the number of timesteps. The model-based estimators work with smaller matrices and thus have faster iterations than RCKL. However, they are not substantially faster overall because they usually require more iterations to converge near the groundtruth.

F. Hyperparameter Tuning

We discuss our procedure and recommendations for tuning the hyperparameters involved in RCKL. The main hyperparameters unique to this algorithm are the regularizers in (16b), namely $\tau_A, \tau_B, \tau_H, \tau_C, \tau_Q, \tau_R$. We used a different set of hyperparameter values for each of the experimental scenarios presented (Simulation 1, Simulation 2, Experiment 1, Experiment 2). We hand-tuned these values by manually adjusting them until a desired performance is reached on a validation dataset. Hyperparameters were never trained nor tuned on the test dataset. To achieve maximum performance, we recommend further refining the hyperparameters through a

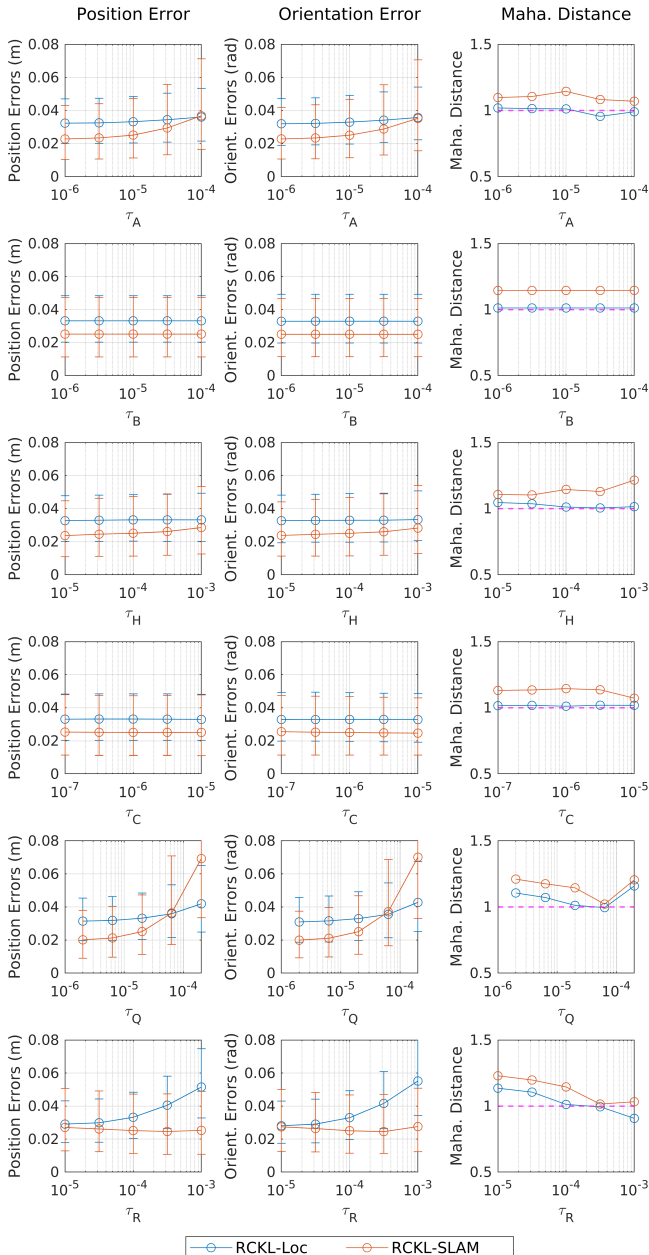


Fig. 11: *Hyperparameter sensitivity test*: Position errors, orientation errors, and Mahalanobis distances of RCKL-Loc and RCKL-SLAM results on Experiment 1 (all 6 folds) vs. different hyperparameter values ($\tau_A, \tau_B, \tau_H, \tau_C, \tau_Q, \tau_R$). For all plots, the middle marker is placed at the hand-tuned hyperparameter value used to generate the results in Section X-C. In each row, a hyperparameter is varied an order of magnitude above and below its hand-tuned value, while all other hyperparameters are set to their hand-tuned values. In the first two columns, we show the 25th, 50th, and 75th percentiles of the errors by representing them with the error bars' lower bounds, the circle markers, and the error bars' upper bounds. In the third column, the markers represent the mean Mahalanobis distances.

grid search or other hyperparameter optimization methods [46]. In our experiments, we saw that RCKL's performance is sensitive to only a few hyperparameter values. To show this, we performed a sensitivity analysis on the τ 's used in Experiment 1. Fig. 11 shows the accuracy and consistency of RCKL-Loc and RCKL-SLAM as we individually vary each of the τ 's within an order of magnitude above and below our hand-tuned

values. We see that RCKL's accuracy and consistency does not change significantly across the tested ranges of τ_A, τ_B, τ_H , and τ_C . The covariance regularizers, τ_Q and τ_R , have larger effects on performance. Notably, the estimates are generally less consistent if τ_Q and τ_R are set either too low or too high, and generally less accurate if τ_Q and τ_R are set too high.

There are other settings that can be classified as hyperparameters, such as the lifting functions used and parameters within the SQP (e.g., μ in the line search acceptance criterion; see (64) in Appendix D). We recommend using standard methods found in literature to choose these parameters. For example, we chose SERFFs as our lifting functions because they are universal [37], but more methodical Koopman identification methods [47], [48] can be used as well.

XI. CONCLUSION

The results highlight the advantages of RCKL's data-driven approach to localization and SLAM. When the classic model-based estimator that relies on Gauss-Newton has access to perfect models in simulation, it has similar or slightly better accuracy than RCKL. However, RCKL has similar or slightly better accuracy than the model-based estimator with an imperfect model. On real-world datasets, RCKL is generally more accurate and more consistent than the classic model-based estimator, and even more so when comparing with classic estimators with imperfect models. While the performance of the model-based estimators depends on the validity of their prior models, RCKL simply applies the high-dimensional model learned through data. In addition, RCKL appears to have the unanticipated benefit of being less prone to local minima.

Many avenues deserve interest for future work. It is worth investigating the origin of RCKL's improvement in convergence properties as a result of lifting estimation into a high-dimensional space. Another direction is to investigate updating the solution incrementally for real-time localization or SLAM, possibly by applying factor graph solvers [49], [50] or smoothing methods [51] in the lifted space. As well, we can broaden the model types that can be converted into lifted forms, making a further step towards data-driven state estimation for general robotics systems. Moreover, many of the techniques developed in this paper could potentially be applied for other applications, such as data-driven optimal control of nonlinear systems.

REFERENCES

- [1] W. Zhao, J. Panerati, and A. P. Schoellig, "Learning-based bias correction for time difference of arrival ultra-wideband localization of resource-constrained mobile robots," *IEEE RA-L*, vol. 6, no. 2, pp. 3639–3646, 2021.
- [2] Y. Tao, L. Wu, J. Sidén, and G. Wang, "Monte Carlo-based indoor RFID positioning with dual-antenna joint rectification," *Electronics*, vol. 10, no. 13, 2021.
- [3] C. Pezzato, R. Ferrari, and C. H. Corbato, "A novel adaptive controller for robot manipulators based on active inference," *IEEE RA-L*, vol. 5, no. 2, pp. 2973–2980, 2020.
- [4] Z. C. Guo, V. Korotkine, J. R. Forbes, and T. D. Barfoot, "Koopman linearization for data-driven batch state estimation of control-affine systems," *IEEE RA-L*, vol. 7, no. 2, pp. 866–873, 2022.
- [5] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer, 2006.
- [6] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *Ann. Stat.*, vol. 36, no. 3, p. 1171–1220, Jun 2008.

- [7] B. O. Koopman, "Hamiltonian systems and transformation in Hilbert space," *PNAS*, vol. 17, no. 5, pp. 315–318, 1931.
- [8] L. Song, J. Huang, A. Smola, and K. Fukumizu, "Hilbert space embeddings of conditional distributions with applications to dynamical systems," in *Proceedings of the 26th ICML*, ser. ICML '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 961–968.
- [9] J. Ko and D. Fox, "GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models," in *IEEE/RSJ IROS*, 2008, pp. 3471–3476.
- [10] B. Ferris, D. Fox, and N. Lawrence, "Wifi-SLAM using Gaussian process latent variable models," vol. 7, 01 2007, pp. 2480–2485.
- [11] Y. Nishiyama, A. Afsharnejad, S. Naruse, B. Boots, and L. Song, "The nonparametric kernel Bayes smoother," in *AISTATS*, 2016.
- [12] S. L. Brunton and J. N. Kutz, *Data-Driven Science and Engineering*. Cambridge, U.K.: Cambridge Univ. Press, 2019.
- [13] A. Surana, M. O. Williams, M. Morari, and A. Banaszuk, "Koopman operator framework for constrained state estimation," in *IEEE 56th CDC*, 2017, pp. 94–101.
- [14] A. Surana, *Koopman Framework for Nonlinear Estimation*. Cham: Springer International Publishing, 2020, pp. 59–79.
- [15] M. Netto and L. Mili, "A robust data-driven Koopman Kalman filter for power systems dynamic state estimation," *IEEE Trans. Power Syst.*, vol. 33, no. 6, pp. 7228–7237, 2018.
- [16] I. Abraham and T. Murphey, "Active learning of dynamics for data-driven control using Koopman operators," *IEEE Trans. Robot.*, vol. 35, pp. 1071–1083, 2019.
- [17] A. Mauroy, I. Mezić, and Y. Suzuki, *The Koopman Operator in Systems and Control*. New York, NY, USA: Springer Publishing, 2020.
- [18] M. Korda and I. Mezić, "Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control," *Automatica*, vol. 93, 11 2016.
- [19] G. Pillonetto, F. Dinuzzo, T. Chen, G. De Nicolao, and L. Ljung, "Kernel methods in system identification, machine learning and function estimation: A survey," *Automatica*, vol. 50, no. 3, pp. 657–682, 2014.
- [20] C.-A. Cheng, H.-P. Huang, H.-K. Hsu, W.-Z. Lai, and C.-C. Cheng, "Learning the inverse dynamics of robotic manipulators in structured reproducing kernel Hilbert space," *IEEE Trans. Cybern.*, vol. 46, no. 7, pp. 1691–1703, 2016.
- [21] A. Mauroy and J. Goncalves, "Linear identification of nonlinear systems: A lifting technique based on the Koopman operator," in *IEEE 55th CDC*, 2016, pp. 6500–6505.
- [22] —, "Koopman-based lifting techniques for nonlinear systems identification," *IEEE Trans. Automat. Contr.*, vol. 65, no. 6, pp. 2550–2565, 2020.
- [23] D. Bruder, C. D. Remy, and R. Vasudevan, "Nonlinear system identification of soft robot dynamics using Koopman operator theory," in *IEEE ICRA*, 2019, pp. 6244–6250.
- [24] T. D. Barfoot, *State Estimation for Robotics*. Cambridge, U.K.: Cambridge Univ. Press, 2017.
- [25] S. L. Brunton, B. W. Brunton, J. L. Proctor, and J. N. Kutz, "Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control," *PLOS ONE*, vol. 11, no. 2, pp. 1–19, 02 2016.
- [26] J. L. Proctor, S. L. Brunton, and J. N. Kutz, "Generalizing koopman theory to allow for inputs and control," *SIAM Journal on Applied Dynamical Systems*, vol. 17, no. 1, pp. 909–930, 2018.
- [27] D. Bruder, X. Fu, and R. Vasudevan, "Advantages of bilinear Koopman realizations for the modeling and control of systems with unknown dynamics," *IEEE RA-L*, vol. 6, no. 3, pp. 4369–4376, 2021.
- [28] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
- [29] P. van Goor, R. Mahony, M. Schaller, and K. Worthmann, "Reprojection methods for Koopman-based modelling and prediction," 07 2023.
- [30] N. Takeishi, Y. Kawahara, and T. Yairi, "Learning Koopman invariant subspaces for dynamic mode decomposition," *Proc. of NIPS*, vol. 30, p. 1130–1140, 12 2017.
- [31] M. Haseli and J. Cortés, "Learning Koopman eigenfunctions and invariant subspaces from data: Symmetric subspace decomposition," *IEEE Trans. Autom. Control*, vol. 67, no. 7, pp. 3442–3457, 2022.
- [32] G. Mamakoukas, I. Abraham, and T. D. Murphey, "Learning stable models for prediction and control," *IEEE T-RO*, vol. 39, no. 3, pp. 2255–2275, 2023.
- [33] E. Jenson, M. Bando, K. Sato, and D. Scheeres, "Robust nonlinear optimal control using Koopman operator theory," *AAS/AIAA Astrodynamics Specialist Conference*, 08 2022, paper AAS 22-647.
- [34] H. G. Bock, E. Kostina, and O. Kostyukova, "Covariance matrices for parameter estimates of constrained parameter estimation problems," *SIAM J. Matrix Anal. Appl.*, vol. 29, no. 2, p. 626–642, mar 2007.
- [35] F. Messerer, K. Baumgärtner, and M. Diehl, "Survey of sequential convex programming and generalized Gauss-Newton methods," *ESAIM: Proceedings and Surveys*, vol. 71, pp. 64–88, 2021.
- [36] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Sparse identification of nonlinear dynamics with control (sindyc)," *10th IFAC Symposium NOLCOS*, vol. 49, no. 18, pp. 710–715, 2016.
- [37] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Proc. of NeurIPS*, vol. 20, 2008, pp. 1177–1184.
- [38] A. M. DeGennaro and N. M. Urban, "Scalable extended dynamic mode decomposition using random kernel approximation," *SIAM Journal on Scientific Computing*, vol. 41, no. 3, pp. A1482–A1499, 2019.
- [39] S. Bak, S. Bogomolov, B. Hency, N. Kochdumper, E. Lew, and K. Potomkin, "Reachability of Koopman linearized systems using random Fourier feature observables and polynomial zonotope refinement," in *Computer Aided Verification*, S. Shoham and Y. Vizel, Eds. Cham: Springer International Publishing, 2022, pp. 490–510.
- [40] D. P. Tsakiris, K. Kapellos, C. Samson, P. Rives, and J.-J. Borrelly, "Experiments in real-time vision-based point stabilization of a nonholonomic mobile manipulator," in *Experimental Robotics V*, A. Casals and A. T. de Almeida, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 570–581.
- [41] P. Polack, F. Althé, B. d'Andréa Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" in *IEEE ITSS IV*, 2017, pp. 812–818.
- [42] C. Holmes, F. Dümbgen, and T. D. Barfoot, "On semidefinite relaxations for matrix-weighted state-estimation problems in robotics," *arXiv:2308.07275*, 2023.
- [43] F. Dümbgen, C. Holmes, and T. D. Barfoot, "Safe and smooth: Certified continuous-time range-only localization," *IEEE RA-L*, vol. 8, no. 2, pp. 1117–1124, 2023.
- [44] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA, USA: The MIT Press, 2005.
- [45] J. Djughash, B. Hamner, and S. Roth, "Navigating with ranging radios: Five data sets with ground truth," *J. Field Robotics*, vol. 26, pp. 689–695, 09 2009.
- [46] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A. Boulesteix, D. Deng, and M. Lindauer, "Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges," *WIRES DATA MIN KNOWL*, vol. 13, 01 2023.
- [47] S. Dahdah and J. R. Forbes, "decargroup/pykoop." 2022. [Online]. Available: <https://github.com/decargroup/pykoop>
- [48] B. M. de Silva, K. Champion, M. Quade, J.-C. Loiseau, J. N. Kutz, and S. L. Brunton, "PySINDY: A Python package for the sparse identification of nonlinear dynamical systems from data," *J. Open Source Softw.*, vol. 5, no. 49, p. 2104, 2020. [Online]. Available: <https://doi.org/10.21105/joss.021104>
- [49] M. Qadri, P. Sodhi, J. G. Mangelson, F. Dellaert, and M. Kaess, "Incopt: Incremental constrained optimization using the Bayes tree," in *2022 IEEE/RSJ IROS*, 2022, pp. 6381–6388.
- [50] B. Bazzana, T. Guadagnino, and G. Grisetti, "Handling constrained optimization in factor graphs for autonomous navigation," *IEEE RA-L*, vol. 8, no. 1, pp. 432–439, 2023.
- [51] P. Sodhi, S. Choudhury, J. G. Mangelson, and M. Kaess, "Ics: Incremental constrained smoothing for state estimation," in *2020 IEEE ICRA*, 2020, pp. 279–285.
- [52] Z. Wang and G. Dissanayake, "Observability analysis of slam using fisher information matrix," in *IEEE ICARCV*, 2008, pp. 1242–1247.
- [53] K. Takahashi, J. Fagan, and M. Chen, "A sparse bus impedance matrix and its application to short circuit study," *Proceedings of the PICA Conference*, 1973.
- [54] T. Barfoot, J. Forbes, and D. Yoon, "Exactly sparse Gaussian variational inference with application to derivative-free batch nonlinear state estimation," *IJRR*, vol. 39, pp. 1473–1502, 07 2020.
- [55] S. Boyd and L. Vandenberghe, *Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares*. Cambridge, U.K.: Cambridge Univ. Press, 2018.

APPENDIX

A. Solving UKL-Loc with the RTS smoother

Here, we outline the process of applying the RTS smoother to the system in (20) to solve UKL-Loc. This may be a more

efficient solution process than the sparse Cholesky solver. Note that the two methods are algebraically equivalent [24, §3].

With the landmarks given, we convert the measurement model to be linear-Gaussian using the same trick as in the process model. Observe that

$$\ell_j \otimes \mathbf{x}_k = (\ell_j \otimes \mathbf{1})\mathbf{x}_k = (\mathbf{1} \otimes \mathbf{x}_k)\ell_j. \quad (51)$$

With this, we can rewrite the measurement equation at each timestep k by stacking the measurements, \mathbf{y}_k , and defining matrices, \mathbf{C}' and \mathbf{R}' as

$$\mathbf{y}_k = \begin{bmatrix} \mathbf{y}_{k,1} \\ \vdots \\ \mathbf{y}_{k,V} \end{bmatrix}, \quad \mathbf{C}' = \begin{bmatrix} \mathbf{C}(\ell_1 \otimes \mathbf{1}) \\ \vdots \\ \mathbf{C}(\ell_V \otimes \mathbf{1}) \end{bmatrix}, \quad \mathbf{R}' = \text{diag}(\mathbf{R}, \dots, \mathbf{R}), \quad (52)$$

where \mathbf{y}_k is filled with known values of the landmarks and \mathbf{R}' is \mathbf{R} stacked diagonally V times. The system in (20) can then be written as

$$\mathbf{x}_k = \mathbf{A}_{k-1}\mathbf{x}_{k-1} + \mathbf{v}_k + \mathbf{w}_k, \quad k = 1, \dots, K, \quad (53a)$$

$$\mathbf{y}_k = \mathbf{C}'\mathbf{x}_k + \mathbf{n}_k, \quad k = 0, \dots, K, \quad (53b)$$

where $\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$, $\mathbf{n}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}')$. If any measurement is missing, we can make \mathbf{C}' and \mathbf{R}' be time-varying and omit the corresponding entries when constructing \mathbf{y}_k , \mathbf{C}'_k , and \mathbf{R}'_k . In any case, we have converted the bilinear time-invariant system in (9) into a linear time-varying (LTV) system. We can then apply the standard RTS smoother [24] to (53) to solve for the states in $\mathcal{O}(N^3KV)$ time. The output is $\mathbf{x}_k \sim \mathcal{N}(\hat{\mathbf{x}}_k, \hat{\mathbf{P}}_{\mathbf{x},k})$, where $\hat{\mathbf{x}}_k$ is the state mean and $\hat{\mathbf{P}}_{\mathbf{x},k}$ is the covariance at timestep k , for all $k = 0, \dots, K$. In contrast to [4], by using a measurement model that is individually applicable to each landmark, we have generalized localization to environments with new landmark positions.

B. Gauss-Newton Setup for UKL-SLAM

We use Gauss-Newton optimization to solve (26) for \mathbf{x} and ℓ . Given an operating point, \mathbf{q}_{op} , the Gauss-Newton approximation of (24) yields

$$J(\mathbf{q}) = J(\mathbf{q}_{\text{op}} + \delta\mathbf{q}) \approx J(\mathbf{q}_{\text{op}}) - \mathbf{g}^T \delta\mathbf{q} + \frac{1}{2} \delta\mathbf{q}^T \mathbf{F} \delta\mathbf{q}, \quad (54)$$

where

$$\mathbf{F} = \mathbf{H}^T \mathbf{W}^{-1} \mathbf{H}, \quad \mathbf{g} = \mathbf{H}^T \mathbf{W}^{-1} \mathbf{e}(\mathbf{q}_{\text{op}}), \quad (55a)$$

$$\mathbf{H} = \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{G}_x & \mathbf{G}_\ell \end{bmatrix}, \quad \mathbf{A}^{-1} = \begin{bmatrix} -\mathbf{A}_0 & \mathbf{1} & & \\ & \ddots & \ddots & \\ & & & \ddots \\ & & & & -\mathbf{A}_{K-1} & \mathbf{1} \end{bmatrix}, \quad (55b)$$

$$\mathbf{G}_x = \text{diag}(\mathbf{G}_{\mathbf{x},1}, \dots, \mathbf{G}_{\mathbf{x},K}), \quad \mathbf{G}_\ell^T = [\mathbf{G}_{\ell,1}^T \quad \dots \quad \mathbf{G}_{\ell,K}^T], \quad (55c)$$

$$\mathbf{G}_{\mathbf{x},k}^T = [\mathbf{G}_{\mathbf{x},k,1}^T \quad \dots \quad \mathbf{G}_{\mathbf{x},k,V}^T], \quad \mathbf{G}_{\ell,k} = \text{diag}(\mathbf{G}_{\ell,k,1}, \dots, \mathbf{G}_{\ell,k,V}), \quad (55d)$$

where, using (51), the measurement Jacobians are

$$\mathbf{G}_{\mathbf{x},k,j} = \left. \frac{\partial \mathbf{y}_{k,j}}{\partial \mathbf{x}} \right|_{\mathbf{x}_{\text{op},k}, \ell_{\text{op},j}} = \mathbf{C}(\ell_{\text{op},j} \otimes \mathbf{1}), \quad (56a)$$

$$\mathbf{G}_{\ell,k,j} = \left. \frac{\partial \mathbf{y}_{k,j}}{\partial \ell} \right|_{\mathbf{x}_{\text{op},k}, \ell_{\text{op},j}} = \mathbf{C}(\mathbf{1} \otimes \mathbf{x}_{\text{op},k}). \quad (56b)$$

Note that \mathbf{F} is at least positive semidefinite when \mathbf{Q} and \mathbf{R} are positive definite, and \mathbf{F} is positive definite when the lifted system is observable [24]. Observability of the lifted system depends on the test data and the Jacobians in \mathbf{H} [52]. We found that in simulation and experimental testing, when UKL has a good initialization, \mathbf{F} is positive definite and successful convergence of the Gauss-Newton algorithm is realized. See Section VIII-C for an initialization procedure. This empirically suggests that when the original system is observable, the lifted system is also observable once given good priors in (16b), reasonable lifting functions, and sufficient training data. An interesting avenue for future work is explicitly enforcing observability when learning \mathbf{A} , \mathbf{B} , \mathbf{H} , and \mathbf{C} .

The optimal update, $\delta\mathbf{q}$, satisfies

$$\mathbf{F} \delta\mathbf{q} = \mathbf{g}, \quad (57)$$

where we can solve for $\delta\mathbf{q}$ by following the standard approach for classic batch SLAM [24]. As we will see in Appendix C, the complexity of solving (57) is $\mathcal{O}(N^3(V^3 + V^2K))$ when we have more timesteps than landmarks, or $\mathcal{O}(N^3(K^3 + K^2V))$ when we have more landmarks than timesteps.

As mentioned in Section VII-D, we can easily accommodate for missing or known variables. If $\mathbf{y}_{k,j}$ is missing, we simply remove $\mathbf{e}_{\mathbf{y},k,j}$ from the cost and delete the corresponding blocks in \mathbf{R}^{-1} , $\mathbf{G}_{\mathbf{x},k}$, and $\mathbf{G}_{\ell,k}$. If \mathbf{x}_k or ℓ_j is already known, we modify (57) to incorporate the known variable by setting \mathbf{x}_k or ℓ_j to its value, removing it from the list of variables in \mathbf{q} , and moving its corresponding equation in $\mathbf{F} \delta\mathbf{q}$ to the right-hand side as part of \mathbf{g} .

The CKL-SLAM problem (34) contains the same objective function as UKL-SLAM (26). As we will see in Appendices D and E, CKL-SLAM's solution process uses the same Gauss-Newton approximation as described above. The case is similar for RCKL-SLAM, where owing to its slightly-modified cost function (46), the only required changes are to replace $\mathbf{Q}^{-1} = \begin{bmatrix} \mathbf{Q}_\xi^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$ in (25b) and $\mathbf{A}_{k-1} = \begin{bmatrix} \mathbf{A}_{\xi,k-1} \\ \mathbf{0} \end{bmatrix}$ in (55b).

C. Solving UKL-SLAM in Linear Time

Although the linear system in (57) is very large, it can be efficiently solved with Cholesky factorization and by exploiting sparsity patterns. \mathbf{F} has the structure

$$\mathbf{F} = \begin{bmatrix} \mathbf{A}^{-T} \mathbf{Q}^{-1} \mathbf{A}^{-1} + \mathbf{G}_x^T \mathbf{R}^{-1} \mathbf{G}_x & \mathbf{G}_x^T \mathbf{R}^{-1} \mathbf{G}_\ell \\ \mathbf{G}_\ell^T \mathbf{R}^{-1} \mathbf{G}_x & \mathbf{G}_\ell^T \mathbf{R}^{-1} \mathbf{G}_\ell \end{bmatrix} = \begin{bmatrix} \mathbf{F}_x & \mathbf{F}_{x\ell} \\ \mathbf{F}_{x\ell}^T & \mathbf{F}_\ell \end{bmatrix}, \quad (58)$$

which exhibits the usual SLAM arrowhead pattern where \mathbf{F}_x is block-tridiagonal and \mathbf{F}_ℓ is block-diagonal [24]. As discussed in Section VII-B, \mathbf{F} is usually positive definite. Then, when we have more robot poses than landmarks, we can use the lower-Cholesky decomposition on \mathbf{F} ,

$$\mathbf{F} = \underbrace{\begin{bmatrix} \mathbf{L}_x & \mathbf{0} \\ \mathbf{L}_{x\ell} & \mathbf{L}_\ell \end{bmatrix}}_L \underbrace{\begin{bmatrix} \mathbf{L}_x^T & \mathbf{L}_{x\ell}^T \\ \mathbf{0} & \mathbf{L}_\ell^T \end{bmatrix}}_{L^T} = \begin{bmatrix} \mathbf{L}_x \mathbf{L}_x^T & \mathbf{L}_x \mathbf{L}_\ell^T \\ \mathbf{L}_{x\ell} \mathbf{L}_x^T & \mathbf{L}_{x\ell} \mathbf{L}_\ell^T + \mathbf{L}_\ell \mathbf{L}_\ell^T \end{bmatrix}, \quad (59)$$

where we can use the fact that F_{11} is block-tridiagonal to efficiently compute the nonzero blocks in L_{xx} (see [24, §3]), and then compute $L_{x\ell}$ and $L_{\ell\ell}$. For the right-hand side of (57), \mathbf{g} has the structure

$$\mathbf{g} = \mathbf{H}^T \mathbf{W}^{-1} \mathbf{e}_{\text{op}} = \begin{bmatrix} \mathbf{A}^{-T} \mathbf{Q}^{-1} \mathbf{e}_{v,\text{op}} + \mathbf{G}_x^T \mathbf{R}^{-1} \mathbf{e}_{y,\text{op}} \\ \mathbf{G}_\ell^T \mathbf{R}^{-1} \mathbf{e}_{y,\text{op}} \end{bmatrix} = \begin{bmatrix} \mathbf{g}_x \\ \mathbf{g}_\ell \end{bmatrix}, \quad (60)$$

for which we can efficiently compute using the appropriate blocks. The solution process for (57) becomes first solving $\mathbf{L}\mathbf{p} = \mathbf{g}$ for a placeholder variable \mathbf{p} using forward substitution, then solving $\mathbf{L}^T \hat{\mathbf{q}} = \mathbf{p}$ for $\hat{\mathbf{q}}$ using backward substitution. Thus, (57) can be solved without ever constructing the large sparse system, rather just working with the required blocks. The exact solution for $\hat{\mathbf{q}}$ can be computed in $\mathcal{O}(N^3(V^3 + V^2K))$ time. On the other hand, if we have more landmarks than poses, we can use the upper-Cholesky decomposition with a similar procedure, solving in $\mathcal{O}(N^3(K^3 + K^2V))$ time. See [24, §9] for an example.

After iterating until convergence, we can also efficiently compute the covariances of the system. To find $\hat{\mathbf{P}}_{\mathbf{x},k}$ and $\hat{\mathbf{P}}_{\ell,j}$, we compute the diagonal blocks of \mathbf{F}^{-1} . Since $\hat{\mathbf{P}}_{\mathbf{q}} = \mathbf{F} = \mathbf{L}\mathbf{L}^T$ [24], we can use [53] to compute only the blocks of \mathbf{F}^{-1} corresponding to the nonzero blocks of \mathbf{L} . This can be done without raising the computational complexity of SLAM given the sparsity pattern of \mathbf{L} present in SLAM problems [54]. The final output is $\mathbf{x}_k \sim \mathcal{N}(\hat{\mathbf{x}}_k, \hat{\mathbf{P}}_{\mathbf{x},k})$, $\ell_j \sim \mathcal{N}(\hat{\ell}_j, \hat{\mathbf{P}}_{\ell,j})$.

D. Formulating the SQP for (R)CKL-SLAM

We formulate an SQP [5] to solve (34). The Lagrangian is $L(\mathbf{q}, \boldsymbol{\lambda}) = J(\mathbf{q}) - \boldsymbol{\lambda}^T \mathbf{h}(\mathbf{q})$, where $\boldsymbol{\lambda}^T = [\boldsymbol{\lambda}_x^T \quad \boldsymbol{\lambda}_\ell^T]$, $\boldsymbol{\lambda}_x^T = [\boldsymbol{\lambda}_{x,1}^T \quad \cdots \quad \boldsymbol{\lambda}_{x,K}^T]$, $\boldsymbol{\lambda}_\ell^T = [\boldsymbol{\lambda}_{\ell,1}^T \quad \cdots \quad \boldsymbol{\lambda}_{\ell,V}^T]$ are the Lagrange multipliers. Using the same matrix structures as (55), we can write the Jacobian of the Lagrangian at an operating point, \mathbf{q}_{op} and $\boldsymbol{\lambda}_{\text{op}}$, as $\left. \frac{\partial L}{\partial \mathbf{q}^T} \right|_{\mathbf{q}_{\text{op}}} = -\mathbf{g} - \mathbf{S}^T \boldsymbol{\lambda}_{\text{op}}$, where \mathbf{g} is defined in (55), and \mathbf{S} is the Jacobian of the constraints, defined as

$$\mathbf{S} = \left. \frac{\partial \mathbf{h}}{\partial \mathbf{q}} \right|_{\mathbf{q}_{\text{op}}} = \begin{bmatrix} \mathbf{S}_x & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_\ell \end{bmatrix}, \quad (61a)$$

$$\mathbf{S}_x = \text{diag}(\mathbf{S}_{x,1}, \dots, \mathbf{S}_{x,K}), \quad \mathbf{S}_\ell = \text{diag}(\mathbf{S}_{\ell,1}, \dots, \mathbf{S}_{\ell,V}), \quad (61b)$$

$$\mathbf{S}_{x,k} = \left. \frac{\partial \mathbf{h}_x(\mathbf{x})}{\partial \mathbf{x}} \right|_{\mathbf{x}_{\text{op},k}}, \quad \mathbf{S}_{\ell,j} = \left. \frac{\partial \mathbf{h}_\ell(\ell)}{\partial \ell} \right|_{\ell_{\text{op},j}}. \quad (61c)$$

Note that \mathbf{S} is block-diagonal owing to the blockwise structure of the constraints.

We use a generalized Gauss-Newton approximation of the Hessian of the Lagrangian [34, §2], [35, §3.2], which simply corresponds to the Gauss-Newton approximation of the objective function:

$$\left. \frac{\partial^2 L}{\partial \mathbf{q} \partial \mathbf{q}^T} \right|_{\mathbf{q}_{\text{op}}} \approx \left(\left. \frac{\partial J}{\partial \mathbf{q}^T} \right|_{\mathbf{q}_{\text{op}}} \right) \left(\left. \frac{\partial J}{\partial \mathbf{q}} \right|_{\mathbf{q}_{\text{op}}} \right) = \mathbf{F}, \quad (62)$$

where \mathbf{F} is defined in (55). In contrast to the full Hessian of the Lagrangian, which may be indefinite, the Gauss-Newton approximation is guaranteed to be at least positive semidefinite. The condition for a valid descent direction from the SQP is

that the reduced Hessian, or the Hessian projected onto the tangent space of the constraints, is positive definite [5, p. 452]. With our Gauss-Newton approximation, the reduced Hessian is at least positive semidefinite. It is also empirically full rank in our experiments, thus meeting the SQP's requirement. See Appendix E for more discussion on this condition.

We now follow the procedure in [5] to set up the SQP system of equations at each iteration as

$$\begin{bmatrix} \mathbf{F} & \mathbf{S}^T \\ \mathbf{S} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \delta \mathbf{q} \\ -\delta \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{g} + \mathbf{S}^T \boldsymbol{\lambda}_{\text{op}} \\ -\mathbf{h} \end{bmatrix}, \quad (63)$$

then update the operating point of the primal variable and the multiplier as $\mathbf{q}_{\text{op}} \leftarrow \mathbf{q}_{\text{op}} + \alpha \delta \mathbf{q}$ and $\boldsymbol{\lambda}_{\text{op}} \leftarrow \boldsymbol{\lambda}_{\text{op}} + \alpha \delta \boldsymbol{\lambda}$, with an appropriate step size α . We find α using a backtracking line search with an acceptance criterion [5],

$$\phi_1(\mathbf{q}_{\text{op}} + \alpha \delta \mathbf{q}, \mu) \leq \phi_1(\mathbf{q}_{\text{op}}, \mu) + \eta \alpha \left. \frac{\partial J}{\partial \mathbf{q}^T} \right|_{\mathbf{q}_{\text{op}}} \delta \mathbf{q}, \quad (64)$$

where $\eta \in (0, 1)$, $\mu > 0$, and $\phi_1(\mathbf{q}, \mu) = J(\mathbf{q}) + \mu \|\mathbf{h}(\mathbf{q})\|_1$ is the \mathcal{L}_1 merit function.

We can now solve for $(\delta \mathbf{q}, \delta \boldsymbol{\lambda})$ and iterate to convergence, yielding $(\mathbf{q}^*, \boldsymbol{\lambda}^*)$. We then recover the mean estimates in the original space, $\hat{\zeta}$, by picking off the top blocks of the lifted estimates in \mathbf{q}^* . Although the linear system in (63) is quite large, as we will see in Appendix E, we can solve it efficiently by exploiting the SLAM arrowhead structure in \mathbf{F} and the block-diagonal structure in \mathbf{S} .

E. Solving (R)CKL-SLAM in Linear Time

In this section, we efficiently solve the large linear system in (63) for $\delta \mathbf{q}$ and $\delta \boldsymbol{\lambda}$, and also describe the requirements on the SQP Hessian for valid descent directions. We define $\mathbf{t} = \delta \mathbf{q}$ and $\mathbf{v} = \delta \boldsymbol{\lambda}$ for brevity.

We solve the SQP with the nullspace method [5]. This method is recommended for systems where the degrees of freedom of the free variables is small. For our system, this number simply corresponds to the degrees of freedom of the original system in (3), since the rest of the variables are lifted from, and thus constrained by, the original variables. As we will see, (63) can be solved in the same time complexity as the unconstrained problem: $\mathcal{O}(N^3(V^3 + V^2K))$ when we have more timesteps than landmarks, or $\mathcal{O}(N^3(K^3 + K^2V))$ when we have more landmarks than timesteps.

We break down \mathbf{t} and \mathbf{v} as $\mathbf{t} = \begin{bmatrix} \mathbf{t}_x \\ \mathbf{t}_\ell \end{bmatrix}$, $\mathbf{v} = \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_\ell \end{bmatrix}$, $\mathbf{t}_x^T = [\mathbf{t}_{x,1}^T \quad \cdots \quad \mathbf{t}_{x,K}^T]$, $\mathbf{t}_\ell^T = [\mathbf{t}_{\ell,1}^T \quad \cdots \quad \mathbf{t}_{\ell,V}^T]$, $\mathbf{v}_x^T = [\mathbf{v}_{x,1}^T \quad \cdots \quad \mathbf{v}_{x,K}^T]$, $\mathbf{v}_\ell^T = [\mathbf{v}_{\ell,1}^T \quad \cdots \quad \mathbf{v}_{\ell,V}^T]$.

Let $\mathbf{S}_{\parallel} = \text{null}(\mathbf{S})$ be a matrix constructed by a basis of the nullspace of \mathbf{S} , where \mathbf{S} is defined in (61). Then, $\text{span}(\mathbf{S}_{\parallel})$ represents the tangent space of the linearized constraints. Let \mathbf{S}_{\perp} be a matrix that completes the basis for \mathbf{S}_{\parallel} , meaning that the square matrix $[\mathbf{S}_{\parallel} \quad \mathbf{S}_{\perp}]$ is full rank³. Since \mathbf{S} is block-

³Note that one choice for \mathbf{S}_{\perp} that contains the desired block-diagonal sparsity structure is $\mathbf{S}_{\perp} = \mathbf{S}^T$. However, other choices of a block-diagonal \mathbf{S}_{\perp} are just as efficient, and fixing $\mathbf{S}_{\perp} = \mathbf{S}^T$ in particular does not simplify any future computations. Rather, this restriction could limit a user from freely constructing \mathbf{S}_{\perp} to resolve any numerical issues that may arise. Thus, in this section, we kept \mathbf{S}_{\perp} as a separate entity from \mathbf{S}^T , in the same spirit as how the nullspace method is presented in [5].

diagonal, we can choose a block-diagonal construction of $\mathbf{S}_{||}$ and \mathbf{S}_{\perp} as

$$\mathbf{S}_{||} = \begin{bmatrix} \mathbf{S}_{||,x} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_{||,\ell} \end{bmatrix}, \quad \mathbf{S}_{\perp} = \begin{bmatrix} \mathbf{S}_{\perp,x} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_{\perp,\ell} \end{bmatrix}, \quad (65a)$$

$$\mathbf{S}_{||,x} = \text{diag}(\mathbf{S}_{||,x,1}, \dots, \mathbf{S}_{||,x,K}), \quad \mathbf{S}_{||,\ell} = \text{diag}(\mathbf{S}_{||,\ell,1}, \dots, \mathbf{S}_{||,\ell,V}), \quad (65b)$$

$$\mathbf{S}_{\perp,x} = \text{diag}(\mathbf{S}_{\perp,x,1}, \dots, \mathbf{S}_{\perp,x,K}), \quad \mathbf{S}_{\perp,\ell} = \text{diag}(\mathbf{S}_{\perp,\ell,1}, \dots, \mathbf{S}_{\perp,\ell,V}), \quad (65c)$$

$$\mathbf{S}_{||,x,k} = \text{null}(\mathbf{S}_{x,k}), \quad \mathbf{S}_{||,\ell,j} = \text{null}(\mathbf{S}_{\ell,j}), \quad (65d)$$

$$\mathbf{S}_{\perp,x,k} = \text{null}(\mathbf{S}_{||,x,k}^T), \quad \mathbf{S}_{\perp,\ell,j} = \text{null}(\mathbf{S}_{||,\ell,j}^T). \quad (65e)$$

We decompose \mathbf{t} into two components, \mathbf{t}_{\perp} and $\mathbf{t}_{||}$:

$$\mathbf{t} = \mathbf{t}_{\perp} + \mathbf{t}_{||}, \quad \mathbf{t}_{\perp} = \mathbf{S}_{\perp} \mathbf{c}_{\perp}, \quad \mathbf{t}_{||} = \mathbf{S}_{||} \mathbf{c}_{||}. \quad (66)$$

Here, \mathbf{t}_{\perp} represents the update direction orthogonal to the constraints, and $\mathbf{t}_{||}$ represents the update direction tangent to the constraints. \mathbf{c}_{\perp} and $\mathbf{c}_{||}$ are, respectively, the coordinates of \mathbf{t}_{\perp} and $\mathbf{t}_{||}$ in the basis of \mathbf{S}_{\perp} and $\mathbf{S}_{||}$. We write \mathbf{c}_{\perp} and $\mathbf{c}_{||}$ as $\mathbf{c}_{\perp} = \begin{bmatrix} \mathbf{c}_{\perp,x} \\ \mathbf{c}_{\perp,\ell} \end{bmatrix}$, $\mathbf{c}_{||} = \begin{bmatrix} \mathbf{c}_{||,x} \\ \mathbf{c}_{||,\ell} \end{bmatrix}$, $\mathbf{c}_{\perp,x}^T = [\mathbf{c}_{\perp,x,1}^T \ \dots \ \mathbf{c}_{\perp,x,K}^T]$, $\mathbf{c}_{\perp,\ell}^T = [\mathbf{c}_{\perp,\ell,1}^T \ \dots \ \mathbf{c}_{\perp,\ell,V}^T]$, $\mathbf{c}_{||,x}^T = [\mathbf{c}_{||,x,1}^T \ \dots \ \mathbf{c}_{||,x,K}^T]^T$, $\mathbf{c}_{||,\ell}^T = [\mathbf{c}_{||,\ell,1}^T \ \dots \ \mathbf{c}_{||,\ell,V}^T]$.

To make the coordinates $\mathbf{c}_{||,x,k}$ and $\mathbf{c}_{||,\ell,j}$ more interpretable, we can enforce that the choice of $\mathbf{S}_{||,x,k}$ and $\mathbf{S}_{||,\ell,j}$ has the identity matrix of the appropriate size on top:

$$\mathbf{S}_{||,x,k} = \begin{bmatrix} \mathbf{1}_{N\xi} \\ * \end{bmatrix}, \quad \mathbf{S}_{||,\ell,j} = \begin{bmatrix} \mathbf{1}_{N\psi} \\ * \end{bmatrix}, \quad (67)$$

where the $*$ block consists of any entries such that (65d) is satisfied. Then, each element in $\mathbf{c}_{||,x,k}$ and $\mathbf{c}_{||,\ell,j}$ represents, respectively, the update in each dimension of ξ_k and ψ_j tangent to the constraints. This structure will also simplify the extraction of covariances later.

To solve for \mathbf{t} , we first solve for \mathbf{t}_{\perp} , then solve for $\mathbf{t}_{||}$. From the second equation of (63), we have $\mathbf{S}\mathbf{t} = -\mathbf{h}$,

$$\Rightarrow \mathbf{S}(\mathbf{S}_{\perp} \mathbf{c}_{\perp} + \mathbf{S}_{||} \mathbf{c}_{||}) = -\mathbf{h}, \quad (68)$$

$$\Rightarrow (\mathbf{S}\mathbf{S}_{\perp}) \mathbf{c}_{\perp} = -\mathbf{h}.$$

We assume that \mathbf{S} has full row rank, or else the SQP is not feasible. Note that $\mathbf{S}\mathbf{S}_{\perp}$ is nonsingular. To see why, notice that $[\mathbf{S}_{||} | \mathbf{S}_{\perp}]$ has full rank, so the product $\mathbf{S}[\mathbf{S}_{||} | \mathbf{S}_{\perp}] = [\mathbf{0} | \mathbf{S}\mathbf{S}_{\perp}]$ has full row rank, and thus $\mathbf{S}\mathbf{S}_{\perp}$ is a square nonsingular matrix. Therefore, we can uniquely solve for \mathbf{c}_{\perp} from (68). With our choice of \mathbf{S}_{\perp} in (65), $\mathbf{S}\mathbf{S}_{\perp}$ also becomes block-diagonal with the same structure as in (65). We can thus solve for \mathbf{c}_{\perp} in $\mathcal{O}(N^3(K+V))$ time with

$$(\mathbf{S}_{x,k} \mathbf{S}_{\perp,x,k}) \mathbf{c}_{\perp,x,k} = -\mathbf{h}_{x,k}, \quad k = 1, \dots, K, \quad (69a)$$

$$(\mathbf{S}_{\ell,j} \mathbf{S}_{\perp,\ell,j}) \mathbf{c}_{\perp,\ell,j} = -\mathbf{h}_{\ell,j}, \quad j = 1, \dots, V. \quad (69b)$$

To find $\mathbf{t}_{||}$, we premultiply the first equation of (63) by $\mathbf{S}_{||}^T$ to eliminate any $\mathbf{S}_{||}^T \mathbf{S}^T$ terms:

$$\mathbf{S}_{||}^T (\mathbf{F}\mathbf{t} - \mathbf{S}^T \mathbf{v}) = \mathbf{S}_{||}^T (\mathbf{g} + \mathbf{S}^T \boldsymbol{\lambda}), \quad (70a)$$

$$\Rightarrow \mathbf{S}_{||}^T \mathbf{F}(\mathbf{S}_{\perp} \mathbf{c}_{\perp} + \mathbf{S}_{||} \mathbf{c}_{||}) = \mathbf{S}_{||}^T \mathbf{g}, \quad (70b)$$

$$\Rightarrow (\mathbf{S}_{||}^T \mathbf{F} \mathbf{S}_{||}) \mathbf{c}_{||} = \mathbf{S}_{||}^T \mathbf{g} - \mathbf{S}_{||}^T \mathbf{F} \mathbf{S}_{\perp} \mathbf{c}_{\perp}, \quad (70c)$$

$$\Rightarrow \mathbf{F}_{||} \mathbf{c}_{||} = \mathbf{g}_{||}, \quad (70d)$$

where $\mathbf{F}_{||} = \mathbf{S}_{||}^T \mathbf{F} \mathbf{S}_{||}$ and $\mathbf{g}_{||} = \mathbf{S}_{||}^T \mathbf{g} - \mathbf{S}_{||}^T \mathbf{F} \mathbf{S}_{\perp} \mathbf{d}_{\perp}$. Here, $\mathbf{F}_{||}$ represents the reduced Hessian [5], with its size just being the degrees of freedom of the original unlifted system.

The SQP solution is guaranteed to be a direction of descent if $\mathbf{F}_{||}$ is positive definite [5, p. 452]. Since \mathbf{F} is at least positive semidefinite, $\mathbf{F}_{||}$ is at least positive semidefinite. We then only require that $\mathbf{F}_{||}$ is full rank. This can be interpreted as observability of the lifted constrained system, where the solution of the states and landmarks is unique given the process prior, measurements, and constraints. Unlike in UKL, the constraints contribute to the observability of the system. That is, \mathbf{F} may not be full rank and is only positive semidefinite (i.e., the unconstrained system is unobservable), but $\mathbf{F}_{||}$ is positive definite (i.e., the constrained system is observable). This happens when the unobservable space in \mathbf{F} is projected out by $\mathbf{S}_{||}$. We see this commonly in RCKL-SLAM, where the evolution of the lifted features is ‘moved’ from the process model to the nonlinear constraints, and the system is only observable with the constraints in place.

Similar to the case for UKL, the observability of (R)CKL depends on the priors, lifting functions, training data, and the observability of the original system. In our experiments, the rank requirement always holds empirically when the SQP is initialized according to Section VIII-C.

Using \mathbf{F} 's breakdown in (58), $\mathbf{F}_{||}$ can be written as

$$\mathbf{F}_{||} = \begin{bmatrix} \mathbf{S}_{||,x}^T \mathbf{F}_x \mathbf{S}_{||,x} & \mathbf{S}_{||,x}^T \mathbf{F}_{x\ell} \mathbf{S}_{||,\ell} \\ \mathbf{S}_{||,\ell}^T \mathbf{F}_{\ell x} \mathbf{S}_{||,x} & \mathbf{S}_{||,\ell}^T \mathbf{F}_{\ell} \mathbf{S}_{||,\ell} \end{bmatrix} = \begin{bmatrix} \mathbf{F}_{||,x} & \mathbf{F}_{||,x\ell} \\ \mathbf{F}_{||,\ell x} & \mathbf{F}_{||,\ell} \end{bmatrix}, \quad (71)$$

where, owing to the block-diagonal structure of $\mathbf{S}_{||}$, the SLAM arrowhead sparsity structure is preserved. This is to say, $\mathbf{F}_{||,x\ell}$ is block-tridiagonal and $\mathbf{F}_{||,\ell\ell}$ is block-diagonal. The blocks in $\mathbf{F}_{||}$ can be computed in parallel in $\mathcal{O}(N^3KV)$ time. The computation of $\mathbf{g}_{||}$ is similarly efficient, since $\mathbf{S}_{||}^T$ is block-diagonal and $\mathbf{S}_{||}^T \mathbf{F} \mathbf{S}_{\perp}$ also preserves the sparsity pattern of \mathbf{F} . Then, we can use a Cholesky decomposition on $\mathbf{F}_{||}$, similar to the one done on \mathbf{F} in (59), to efficiently solve for $\mathbf{c}_{||}$ in (70d). Again, the complexity is $\mathcal{O}(N^3(V^3 + V^2K))$ when the lower Cholesky decomposition is used when there are more timesteps than landmarks, but the upper Cholesky decomposition can be used for the other case. Finally, we use \mathbf{c}_{\perp} and $\mathbf{c}_{||}$ to form \mathbf{t}_{\perp} and $\mathbf{t}_{||}$, and then form the full update variable, \mathbf{t} , with (66), all in linear time.

To solve for the Lagrange multiplier update, \mathbf{v} , we premultiply the first equation of (63) by \mathbf{S}_{\perp}^T :

$$\mathbf{S}_{\perp}^T (\mathbf{F}\mathbf{t} - \mathbf{S}^T \mathbf{v}) = \mathbf{S}_{\perp}^T (\mathbf{g} + \mathbf{S}^T \boldsymbol{\lambda}), \quad (72a)$$

$$\Rightarrow (\mathbf{S}\mathbf{S}_{\perp})^T \mathbf{v} = \mathbf{S}_{\perp}^T (\mathbf{F}\mathbf{t} - \mathbf{g} - \mathbf{S}^T \boldsymbol{\lambda}), \quad (72b)$$

where we can solve for \mathbf{v} in linear time since the square matrix $\mathbf{S}\mathbf{S}_{\perp}$ is full rank and block-diagonal. With this, we have efficiently solved for \mathbf{t} and \mathbf{v} in time $\mathcal{O}(N^3(V^3 + V^2K))$.

F. Proof of (R)CKL-SLAM Covariance Extraction

Below, we will show that when $\mathbf{S}_{||,x,k}$ and $\mathbf{S}_{||,\ell,j}$ have the structures shown in (67), then the batch covariance of the original variables satisfies

$$\hat{\Sigma}_\zeta = \mathbf{F}_{||}^{-1} = (\mathbf{S}_{||}^T \mathbf{F} \mathbf{S}_{||})^{-1}, \quad (73)$$

where $\mathbf{F}_{||}$ is constructed at the solution $(\mathbf{q}^*, \boldsymbol{\lambda}^*)$. Since $\mathbf{F}_{||}$ has the SLAM arrowhead sparsity structure, we use a similar procedure as for UKL-SLAM to extract the required covariances by finding the corresponding nonzero blocks of $\mathbf{F}_{||}$.

We now show (73). After convergence, we construct the left-hand side of the SQP in (63) once more, yielding $\mathbf{M} = \begin{bmatrix} \mathbf{F} & \mathbf{S}^T \\ \mathbf{S} & \mathbf{0} \end{bmatrix}$. \mathbf{M} is termed the Karush–Kuhn–Tucker (KKT) matrix [55]. Before inverting \mathbf{M} , note that \mathbf{F} may not be invertible. As discussed in Appendix E, this occurs when only the constrained system is observable, which is most commonly seen in RCKL-SLAM. However, as long as $\mathbf{F}_{||}$ is invertible and \mathbf{S} has full row rank, then \mathbf{M} is invertible. This is because the SQP solution, $\delta \mathbf{q}$ and $\delta \boldsymbol{\lambda}$ in (63), is unique, as seen in Appendix E by construction. We now write the inverse of the KKT matrix as $\mathbf{M}^{-1} = \begin{bmatrix} \mathbf{P}_F & \mathbf{P}_S^T \\ \mathbf{P}_S & \mathbf{P}_Z \end{bmatrix}$, where \mathbf{P}_F , \mathbf{P}_S , and \mathbf{P}_Z has the same size as \mathbf{F} , \mathbf{S} , and the $\mathbf{0}$ in \mathbf{M} , respectively. Then, it is known that $\hat{\mathbf{P}}_q$, the covariance of the primal variable, satisfies $\hat{\mathbf{P}}_q = \mathbf{P}_F$ [34].

We cannot use the Schur complement on \mathbf{M} to solve for \mathbf{P}_F since \mathbf{F} is not necessarily invertible. However, computing \mathbf{P}_F is actually not necessary. When the constraints are satisfied at the minimum-cost solution, any uncertainty as a result of noisy information would only be within the constraint manifold's tangent space [34]. That is, we can write $\mathbf{q}^* = \mathbf{S}_{||} \mathbf{c}^*$, where \mathbf{c}^* is the coordinate of \mathbf{q}^* in the basis of $\mathbf{S}_{||}$. The solution distribution in the basis of $\mathbf{S}_{||}$ satisfies $\mathbf{c} \sim \mathcal{N}(\hat{\mathbf{c}}, \hat{\mathbf{P}}_c)$, where $\hat{\mathbf{c}} = \mathbf{c}^*$ and $\hat{\mathbf{P}}_c = \mathbf{S}_{||} \hat{\mathbf{P}}_q \mathbf{S}_{||}^T$, affirming that the covariance of \mathbf{q} is within the space of $\mathbf{S}_{||}$. Then, observe that $\hat{\Sigma}_\zeta = \hat{\mathbf{P}}_c$, where $\hat{\Sigma}_\zeta$ is the covariance of the original state variables and is exactly the covariance we need. This is because owing to the structure of the bases in (67), the dimensions in \mathbf{c} exactly represent the dimensions of the original variables, so the covariance ζ is equivalent to the covariance of \mathbf{c} .

We write out the expression $\mathbf{M} \mathbf{M}^{-1} = \mathbf{1}$, yielding

$$\begin{bmatrix} \mathbf{F} & \mathbf{S}^T \\ \mathbf{S} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{P}_F & \mathbf{P}_S^T \\ \mathbf{P}_S & \mathbf{P}_Z \end{bmatrix} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}. \quad (74)$$

We premultiply first expression of (74) by $\mathbf{S}_{||}^T$:

$$\mathbf{F} \mathbf{P}_F + \mathbf{S}^T \mathbf{P}_S = \mathbf{1}, \quad (75a)$$

$$\Rightarrow \mathbf{S}_{||}^T \mathbf{F} \mathbf{S}_{||} \hat{\mathbf{P}}_c \mathbf{S}_{||}^T + \mathbf{S}_{||}^T \mathbf{S}^T \mathbf{P}_S = \mathbf{S}_{||}^T, \quad (75b)$$

$$\Rightarrow (\mathbf{S}_{||}^T \mathbf{F} \mathbf{S}_{||}) \hat{\mathbf{P}}_c \mathbf{S}_{||}^T = \mathbf{S}_{||}^T, \quad (75c)$$

$$\Rightarrow \hat{\mathbf{P}}_c = (\mathbf{S}_{||}^T \mathbf{F} \mathbf{S}_{||})^{-1}, \quad (75d)$$

where the last derivation uses the fact that $\mathbf{S}_{||}$ has full column rank. Thus, $\hat{\Sigma}_\zeta = \hat{\mathbf{P}}_c = (\mathbf{S}_{||}^T \mathbf{F} \mathbf{S}_{||})^{-1}$.

G. (R)CKL Estimation with Orientation

Suppose the state of a 2D robot is $\boldsymbol{\xi}_k = [\xi_{x,k} \ \xi_{y,k} \ \xi_{\theta,k}]^T$, where $(\xi_{x,k}, \xi_{y,k})$ is its position and $\xi_{\theta,k}$ is its orientation. Since $\xi_{\theta,k}$ is circular, simply lifting with $\mathbf{x}_k = \mathbf{p}_\xi(\boldsymbol{\xi}_k)$ in (31a) would not work: $\xi_{\theta,k}$ would be part of the lifted state but we cannot enforce that $\xi_{\theta,k} = \xi_{\theta,k} + 2m\pi$ for $m \in \mathbb{Z}$. Instead, we do a variable transformation on $\boldsymbol{\xi}_k$ as $\boldsymbol{\xi}'_k = [\xi_{x,k} \ \xi_{y,k} \ \xi_{\cos \theta,k} \ \xi_{\sin \theta,k}]^T$, where the conversion from $\boldsymbol{\xi}_k$ to $\boldsymbol{\xi}'_k$ is $\xi_{\cos \theta,k} = \cos \xi_{\theta,k}$, $\xi_{\sin \theta,k} = \sin \xi_{\theta,k}$. We introduce the constraint $h_\xi(\boldsymbol{\xi}'_k) = \xi_{\cos \theta,k}^2 + \xi_{\sin \theta,k}^2 - 1 = 0$. We then select a lifting function $\tilde{\mathbf{p}}_{\boldsymbol{\xi}'_k}(\boldsymbol{\xi}'_k)$, and then write the lifted state as $\mathbf{x}_k = \begin{bmatrix} \boldsymbol{\xi}'_k \\ \tilde{\mathbf{p}}_{\boldsymbol{\xi}'_k}(\boldsymbol{\xi}'_k) \end{bmatrix} = \begin{bmatrix} \boldsymbol{\xi}'_k \\ \tilde{\mathbf{x}}_k \end{bmatrix}$, where the manifold constraint is now the lifting function constraint in addition to the orientation constraint on $\boldsymbol{\xi}'_k$:

$$\mathbf{x}_k \in \mathcal{X} \Rightarrow \mathbf{h}_\mathbf{x}(\mathbf{x}_k) = \begin{bmatrix} \tilde{\mathbf{x}}_k - \tilde{\mathbf{p}}_{\boldsymbol{\xi}'_k}(\boldsymbol{\xi}'_k) \\ \xi_{\cos \theta,k}^2 + \xi_{\sin \theta,k}^2 - 1 \end{bmatrix} = \mathbf{0}. \quad (76)$$

Note that the additional constraint still acts on each state individually at each timestep, meaning that $\mathbf{h}_\mathbf{x}(\mathbf{x})$ is still block-diagonal. Thus, the rest of the procedure for (R)CKL estimation follows as usual. Afterwards, we get $\hat{\xi}_{\theta,k} = \text{atan2}(\hat{\xi}_{\sin \theta,k}, \hat{\xi}_{\cos \theta,k})$.

With the additional constraint, the covariance extraction needs to be slightly modified. $\hat{\Sigma}_{\boldsymbol{\xi},k}$, the covariance of $\boldsymbol{\xi}_k$, is not the same as $\hat{\mathbf{P}}_{\mathbf{c},k}$, the covariance of \mathbf{x}_k in the coordinates of the constraint tangent space defined by $\mathbf{S}_{||,x,k}$. $\boldsymbol{\xi}'_k$ has 4 dimensions but only 3 degree of freedom. As such, we enforce a different structure on $\mathbf{S}_{||,x,k}$ than in (67):

$$\mathbf{S}_{||,x,k} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & S_{||,\cos \theta,k} \\ 0 & 0 & S_{||,\sin \theta,k} \\ * & * & * \\ \vdots & \vdots & \vdots \end{bmatrix}, \quad (77)$$

where the * entries are any entries so that $\mathbf{S}_{||,x,k} = \text{null}(\mathbf{S}_{\mathbf{x},k})$ is satisfied. Then, given $\hat{\mathbf{P}}_{\mathbf{c},k}$ in the form

$$\hat{\mathbf{P}}_{\mathbf{c},k} = \begin{bmatrix} \hat{\sigma}_{\mathbf{c},x,k}^2 & \hat{\sigma}_{\mathbf{c},xy,k}^2 & \hat{\sigma}_{\mathbf{c},x\theta,k}^2 \\ \hat{\sigma}_{\mathbf{c},xy,k}^2 & \hat{\sigma}_{\mathbf{c},y,k}^2 & \hat{\sigma}_{\mathbf{c},y\theta,k}^2 \\ \hat{\sigma}_{\mathbf{c},x\theta,k}^2 & \hat{\sigma}_{\mathbf{c},y\theta,k}^2 & \hat{\sigma}_{\mathbf{c},\theta,k}^2 \end{bmatrix}, \quad (78)$$

the position covariance is still the upper-left 2×2 block, and the variance of $\xi_{\cos \theta,k}$ and $\xi_{\sin \theta,k}$ can be found with

$$\hat{\Sigma}_{\boldsymbol{\xi},\cos \theta,k} = S_{||,\cos \theta,k}^2 \hat{\sigma}_{\mathbf{c},\theta,k}^2, \quad \hat{\Sigma}_{\boldsymbol{\xi},\sin \theta,k} = S_{||,\sin \theta,k}^2 \hat{\sigma}_{\mathbf{c},\theta,k}^2, \quad (79)$$

either of which can be converted to the orientation variance, $\hat{\Sigma}_{\boldsymbol{\xi},k,\theta}$, through a linear transformation of a Gaussian at the mean, $\hat{\xi}_{\theta,k}$. We see in our experiments that when $\hat{\Sigma}_{\boldsymbol{\xi},\cos \theta,k}$ and $\hat{\Sigma}_{\boldsymbol{\xi},\sin \theta,k}$ are small, using either variance yield the same value for $\hat{\Sigma}_{\boldsymbol{\xi},\theta,k}$.