Robotics: Science and Systems 2025
Los Angeles, CA, USA, June 21–25, 2025

Kinodynamic Trajectory Following with STELA: Simultaneous Trajectory Estimation & Local Adaptation

Edgar Granados, Sumanth Tangirala, Kostas E. Bekris
Dept. of Computer Science, Rutgers University, NJ, USA
{eg585, st1122, kb572}@cs.rutgers.edu

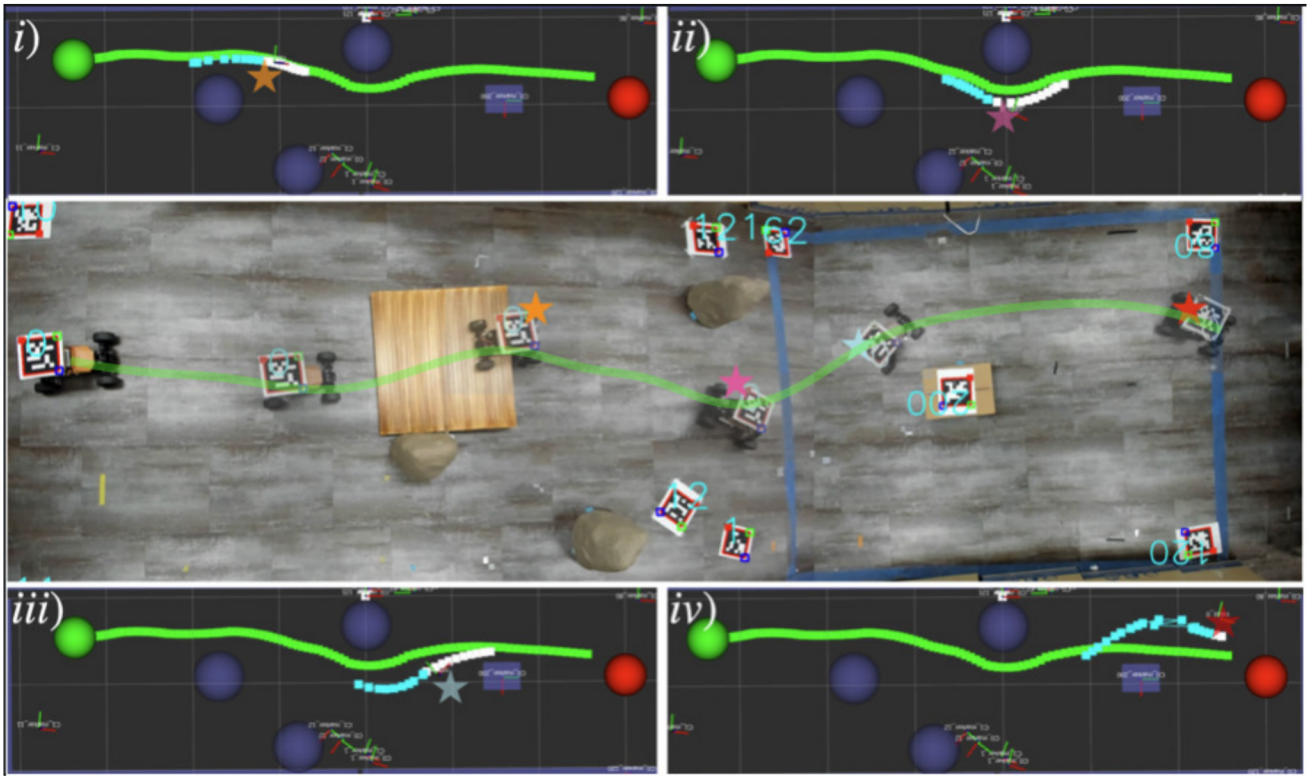


Fig. 1: Middle: STELA execution on a *real* MuSHR robot. The middle image is a composite from 2 top-down cameras used for localization, covering a 7.6mx2.3m workspace. The robot follows a trajectory computed by a planner with knowledge of the obstacles (rocks and boxes) but no knowledge of the ramp, affecting execution. Top and Bottom: i) STELA estimation and plan when the robot is on the unknown ramp; ii) the robot recovers from the ramp and avoids an obstacle; iii) STELA adapts the plan to follow the planned trajectory while avoiding another obstacle; iv) the robot reaches the end of the plan without collisions. Rviz visualization includes obstacles, planned trajectory (green), forward horizon (white), and history (cyan). Stars indicate corresponding states between the visualization and the real robot.

Abstract*—State estimation and control are often addressed separately, leading to unsafe execution due to sensing noise, execution errors, and discrepancies between the planning model and reality. Simultaneous control and trajectory estimation using probabilistic graphical models has been proposed as a unified solution to these challenges. Previous work, however, relies heavily on appropriate Gaussian priors and is limited to holonomic robots with linear time-varying models. The current research extends graphical optimization methods to vehicles with arbitrary dynamical models via Simultaneous Trajectory Estimation and Local Adaptation (STELA). The overall approach initializes feasible trajectories using a kinodynamic, sampling-based motion planner. Then, it simultaneously: (i) estimates the past trajectory based on noisy observations, and (ii) adapts the controls to be executed to minimize deviations from the planned, feasible trajectory, while avoiding collisions. The proposed factor graph representation of trajectories in STELA can be applied for any dynamical system given access to first or second-order state update equations, and introduces the duration of execution between two states in the trajectory discretization as an optimization variable. These features provide both generalization and flexibility in trajectory following. In addition to targeting computational efficiency, the proposed strategy performs incremental updates of the factor graph using the iSAM algorithm and introduces a time-window mechanism. This mechanism allows the factor graph to be dynamically updated to operate over a limited history and forward horizon of the planned trajectory. This enables online updates of controls at a minimum of 10Hz. Experiments demonstrate that STELA achieves at least comparable performance to previous frameworks on idealized vehicles with linear dynamics. STELA also directly applies to and successfully solves trajectory following problems for more complex dynamical models. Beyond generalization, simulations assess STELA's robustness under varying levels of sensing and execution noise, while ablation studies highlight the importance of different components of STELA. Real-world experiments validate STELA's practical applicability on a low-cost MuSHR robot, which exhibits high noise and non-trivial dynamics.
Website: https://go.rutgers.edu/46618xjt

I. INTRODUCTION

This paper focuses on achieving reliable simultaneous trajectory estimation and following for kinodynamic systems in static, partially modeled environments, under sensing and actuation uncertainty as well as reality gaps for the planning robot model. It proposes Simultaneous Trajectory Estimation and Local Adaptation (STELA), a graphical optimization framework that builds on top of prior methods for simultaneous state estimation and control based on factor graph representations [7, 22]. It extends: (i) the efficacy of such approaches by increasing the success rate of returning feasible, collision-free trajectories, even under significant noise, (ii) their applicability to more general dynamical systems, and (iii) achieves improved computational efficiency.

Motivation: Reliable mobile robot navigation, especially for low-cost platforms, such as the MuSHR robot used in this work and shown in Fig. 1, can be challenging due to observation noise, high actuation errors, and a significant reality gap of the underlying models. For robots with significant dynamics, the planning models are often analytical dynamic expressions, which allow for fast computation but tend to be rough approximations. This results in significant deviations from the planned trajectory during execution. While system identification [16, 3, 44] can reduce the model gap, it does not fully address it. This is often because the environment is also partially observable. For instance, a floor map indicating obstacles may be available, but not all aspects of the environment are modeled, such as friction coefficients and traversable features, such as ramps and speed obstructions.

An approach to deal with the model gap is to use feedback controllers for trajectory following, given the latest state estimate [12, 33]. However, observation and actuation noise can lead to errors in state estimation, where the focus is often filtering, i.e., estimating the latest robot pose incrementally. State estimation noise can compound to result in deviations in trajectory following. In addition, most trajectory following controllers ignore obstacles and are executed independently from the state estimation process, so significant trajectory deviations also lead to collisions.

An alternative strategy is replanning online, e.g., with sampling-based motion planners (SBMP) [21, 27, 23] or trajectory optimization planners [36, 41, 20, 42]. SBMPs can eventually provide high-quality, feasible solutions, but typically it is not possible to achieve high-frequency replanning rates for dynamical systems (e.g., at or above 10Hz) to deal with the model gap without a trajectory follower. Optimization-based approaches can sometimes provide solutions fast, but due to their local nature, they may get stuck in local minima if not properly initialized and can be parameter sensitive.

Factor Graph Optimization: A promising direction for addressing the above issues, which this paper builds on, is probabilistic graphical models based on factor graphs and corresponding optimization methods [7]. They can simultaneously solve trajectory estimation and control or planning challenges as a unified problem [22, 29]. These solutions perform smoothing instead of filtering, i.e., they estimate the entire most likely trajectory the robot has followed given all the available observations. Smoothing often leads to improved estimates relative to filtering. Smoothing solutions were traditionally slower to compute, but the progress with factor graph optimization tools has allowed such problems to be solved online. Furthermore, these methods are able to adapt the robot controls simultaneously to achieve collision avoidance by taking obstacles into account during the control optimization process, or alternatively, come up with new planning solutions.

Nevertheless, most of the existing solutions in this space: (i) rely heavily on Gaussian priors regarding the underlying probabilistic processes, which may not reflect the true uncertainty of the system, and (ii) are limited to holonomic robots given linear-time varying models, reducing their applicability on non-holonomic vehicles with significant dynamics. Furthermore, it is also desirable to operate such solutions at high frequencies to maximize robustness to disturbances.

Proposed Method and Contribution: The proposed STELA framework first calls an asymptotically optimal SBMP for kinodynamic systems [23, 27] in order to acquire a feasible, collision-free trajectory given the available planning model. A key aspect of STELA is that it aligns the output of the SBMP with the consecutive trajectory optimization via a common graphical representation. Factor Graphs (FGs) are a natural interface for this purpose. In particular, the underlying motion graph produced by an SBMP is transformed by the proposed approach into a FG. Then, STELA uses the extracted FG to simultaneously perform: (a) smoothing of the past trajectory given the latest observations, and (b) dynamically adapting the controls to be executed so as to minimize deviations from the SBMP trajectory, while avoiding collisions.

The proposed FG representation in STELA, illustrated in Fig. 5, is general in nature and can be applied to any dynamical system given access to first or second-order state update equations. It does not make any assumptions in terms of Gaussian priors for the underlying processes, and it does not require linear-time varying models, which limit applicability to idealistic holonomic vehicles. Instead, it only uses the solution achieved by the SBMP as a prior and employs non-linear factors representing the system's dynamics. The proposed FG also includes the duration of execution between two states in the trajectory discretization as a variable to be optimized. This allows the optimizer to *stretch* or *contract* edges depending on the estimated state of the system. The combination of these features provides *generalization*, in terms of the range of the dynamical systems that can be modeled, as well as *high success rate* in finding a collision-free path even when the robot has deviated from the planned solution.

For increased *computational efficiency*, the proposed strategy allows for the use of incremental updates of the FG by: (a) using the iSAM2 optimizer, and also (b) introducing a sliding window mechanism. In particular, given the adopted FG representation and seeking the Maximum a posteriori (MAP) solution via incremental inference, the iSAM2 optimizer incorporates high-frequency observations, adapting the underlying

graph in a computationally efficient manner, obtaining the most likely estimate of the robot's past trajectory. The sliding window mechanism allows the factor graph to be dynamically updated at high frequency by operating over a limited past history and forward horizon of the planned trajectory. The combination of these features enables online control updates to be generated at a minimum of 10Hz and on average at 20-30Hz, depending on the setup.

Experimentation: The framework is tested first in simulation for different environments and different levels of observation and actuation noise both for an idealized holonomic model used in prior work as well a second-order dynamical system not addressable by prior FG work corresponding to a MuSHR robot. The simulations demonstrate that STELA achieves at least comparable, and often superior, performance to previous frameworks on the idealized vehicle with linear dynamics. More critically, STELA also achieves a high success rate for the second-order dynamical system and good robustness given different noise levels. Ablation studies highlight the importance of different components of the approach, such as the importance of the SBMP initialization, the introduction of control duration as an optimization variable, and the incorporation of the sliding window approach. Real-world experiments validate STELA's practical applicability on a low-cost MuSHR robot that exhibits high noise and non-trivial dynamics.

II. RELATED WORK

Motion planning consists of finding a plan for a robot to move in an environment from a starting state to a desired goal region without collisions. Multiple classifications of motion planning algorithms have been proposed [11, 32, 2, 30]. This paper primarily focuses on Sampling-Based Motion Planners (SBMP) and trajectory optimization.

SBMPs build graphical representations of the underlying robot's state space and can provide guarantees, namely probabilistic completeness (PC) and asymptotic optimality (AO), for kinematic [21, 9] and kinodynamic [34, 27, 23] systems. Additionally, SBMPs can discover different solutions (i.e., among different homotopic classes) and easily handle task-specific constraints (i.e., collisions, limits), but can be negatively impacted computationally as the cost of SBMP subroutines increases, e.g., forward propagation, collision checking, and nearest neighbor discovery [26], which is an issue when planning for dynamical systems.

Motion planning can also be viewed as a numerical optimization problem. Example optimization-based motion planners include Covariant Hamiltonian optimization (CHOMP)[36], stochastic trajectory optimization (STOMP) [20], and Sequential Convex Optimization (TrajOpt) [38]. Starting from an initial, potentially infeasible, *guess*, an optimizer attempts to minimize an objective function subject to constraints, such as system dynamics, collisions, reaching the goal, and possibly other aspects, e.g., energy minimization. Optimization methods can quickly converge to a feasible, high-quality solution if the initial guess is in the vicinity of one. Non-trivial environments and systems, however, can challenge convergence and may require careful definition of parameters, such as *obstacle potentials* [36]. Frequently, the initial guess also imposes a discretization of the solution trajectory, limiting the set of possible solutions. Continuous trajectory representations via Gaussian Processes [30] can minimize the discretization problem for linear time-varying (LTV) systems that are stabilized using stochastic differential equations (SDEs), i.e., LTV-SDE systems.

There are also integrative frameworks in the planning literature. For kinematic systems, a precomputed graph of convex sets can be used by a mixed-integer optimizer to find a valid path [28]. An approach for dynamical systems involves the pre-computation of motion primitives, which are *stitched* together by a planner [13, 32]. An interleaving approach uses a graph to generate suggestions used by an optimizer [31]. Simultaneous localization and planning (SLAP) [1] models the challenge as a POMDP, which is approximated offline by a roadmap in belief space that is updated online via observations.

In the area of **state estimation**, there has been a transition from tools that explicitly estimate the probability distribution regarding the robot's state to optimization techniques, such as those that employ Factor Graphs (FGs) [7]. A FG is a probabilistic graphical model that can be used to represent the joint probability mass function of the variables that comprise the system [24]. FGs aim to exploit the underlying, known structure of the problem by using sparse linear algebra techniques and find solutions via optimization algorithms, such as Gauss-Newton. Non-linear problems require frequent re-linearization, which is a costly operation to perform online over the entire graph. The incremental smoothing and mapping (iSAM2) [18] algorithm uses the Bayes tree [17] to keep updates local, avoiding a full re-linearization of the graph. FGs are well suited for addressing a variety of problem formulations, such as simultaneous localization and mapping (SLAM) [5, 8, 15] and simultaneous trajectory estimation and mapping (STEAM) [4]. Multiple initial guesses are possible via a network of trajectories so as to initialize a FG-based planning solution for holonomic systems [14].

The two closest methods to this work are also employing FGs and **integrate state estimation and planning/control**. They correspond to Simultaneous Trajectory Estimation And Planning (STEAP) [29] and Simultaneous Control And Trajectory Estimation (SCATE) [22]. SCATE builds on top of STEAP and also deals with dynamic obstacles. Specifically, STEAP employs Gaussian processes as dynamics, while SCATE uses a linear time-invariant (LTI) approximation of the dynamics. SCATE adds the LTI dynamics and dynamic obstacle factors to the FG. The output of STEAP is a *plan* but does not compute controls directly and relies on an external controller. SCATE outputs a control to be applied to the robot. To operate at high frequencies, SCATE relies on a low-level controller and state estimator. Neither of the two techniques is directly applicable to systems with non-linear dynamics. Prior work also uses naive initialization of these methods, such as a straight line connection of the start and the goal, which can lead to local minima due to the local nature of numerical optimization.

III. PROBLEM SETUP

Consider a robot with state space $\mathbb{X}$ and control space $\mathbb{U}$ tasked with navigating a workspace $\mathbb{W}$ from initial state x_0 to a goal region X_G. Obstacles divide $\mathbb{X}$ into collision-free $\mathbb{X}_{\text{f}}$ and obstacle $\mathbb{X}_{\text{o}}$ subsets. The **true dynamics** $\dot{x}_t = f(x_t, u_t)$ ($x_t \in \mathbb{X}$, $u_t \in \mathbb{U}$) govern the robot's motions but are not perfectly known and may exhibit non-holonomic constraints.

In terms of its dynamics, the robot has access only to an **approximate dynamics model** via a parameterized function $\dot{x}_t = \hat{f}_\rho(x_t, u_t)$, which is defined by a set of physical parameters ρ. The approximate dynamics model $\hat{f}_\rho$ is a simplification of the true dynamics f and does not necessarily have the same expression as f, i.e., there may not be a choice of physical parameters ρ, which will allow $\hat{f}_\rho$ to be perfectly identified with f. Example physical parameters for a car-like robot are friction coefficients, steering, and throttle gains.

In addition, the approximate model $\hat{f}_\rho$ employed models the obstacle-free workspace as a flat, planar surface with uniform friction. In reality, however, the true workspace can also exhibit: (i) different friction from the one the robot assumes, which can vary over the workspace, and (ii) an uneven surface that can include traversable obstructions, such as a ramp.

A **plan** p_T is a sequence of T piece-wise constant controls $\{u_0, \dots, u_{T-1})\}$, where each control u_i is executed for a timestep Δt_i. When a plan p_T is executed at a state x_i, it produces a **trajectory** $\tau_f(x_t, p_T)$, i.e., a sequence of states $\{x_t, \cdots, x_{t+T}\}$ according to a dynamics model f. Due to the gap between the true dynamics f and the planning model $\hat{f}_\rho$, the executed robot trajectory $\tau_f(x_i, p_T)$ does not match the planned trajectory $\tau_{\hat{f}_\rho}(x_i, p_T)$ for the same plan p_T.

The robot has access to noisy sensing that provides discrete **measurements** $z(t)$, which partially inform about the robot's state $x(t)$, such as sensing the robot's pose from external sensors. A state estimation process uses measurements $z(0:t)$ to compute **estimated states** $\bar{x}(0:t)$. This work assumes perfect knowledge of obstacles' poses during execution, such as walls or obstacles that the robot should not collide with.

A **controller** $\pi_{\hat{f}}(\bar{x}_i, \tau_{\hat{f}_\rho})$ is employed to track the planned trajectory $\tau_{\hat{f}_\rho}$ given the estimated states $\bar{x}_i$ and returns controls $u \in \mathbb{U}$. With some abuse of notation[1], denote as $\tau_f(x_i, \pi_{\hat{f}})$ the trajectory executed by the robot when the controller $\pi_{\hat{f}}$ is executed starting at state x_i.

Problem Definition: Given a: (i) start state $x_0 \in \mathbb{X}_{\text{f}}$, (ii) goal region $X_G \subset \mathbb{X}_{\text{f}}$, (iii) approximate dynamics model $\hat{f}_\rho(x, u)$ given identified parameters ρ, (iv) desired trajectory $\tau_{\hat{f}_\rho}(x_0, p_T)$ that brings the robot in X_G, and (v) online measurements $z(t)$, the objective is to simultaneously compute estimated robot states $\bar{x}(t)$ and execute a controller $\pi_{\hat{f}}(\bar{x}_0, \tau_{\hat{f}_\rho})$ so that the executed trajectory $\tau_f(x_0, \pi_{\hat{f}})$ is collision free and brings the robot inside X_G.

Secondary objectives include minimizing the error between estimated states $\bar{x}(t)$ and true states $x(t)$, minimizing the error between the planned and the executed trajectory, as well as minimizing the cost of the executed trajectory. In this work, the cost corresponds to the trajectory duration.

[1]Trajectories above were defined for open-loop plans p_T. Here the definition is adapted to receive as input the controls arising from the closed-loop controller $\pi_{\hat{f}}$.

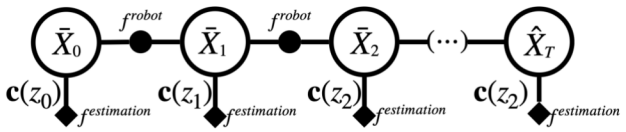


Fig. 2: A typical trajectory estimation FG at time T uses state observations $z^x(0:T)$ and the robot model $\dot{x}_t = \hat{f}_\rho(x_t, u_t)$ to generate state estimates $\bar{X}(0:T)$. The unary factors impose a cost between observations and estimated states. The binary factors correspond to the robot's dynamics.

Additional notation: The goal region is defined by a single configuration q_G in the robot's configuration space $\mathbb{Q}$ so that: $X_G = \{x \in \mathbb{X}_f \mid d(\mathbb{M}(x), q_G) < \epsilon\}$, or equivalently, $X_G = \mathcal{B}(q_G, \epsilon)$, where ϵ is a goal radius in $\mathbb{Q}$ according to distance function d. The function $\mathbb{M}(x)$ maps states to configurations. Estimated *past/current* states are represented by $\bar{x}(t)$ and future (estimated) states are represented as $\hat{x}(t)$.

The following are relevant Lie group concepts and operations; refer to [39] for a more in-depth explanation. Consider $q_i \in \mathcal{M}$ where $\mathcal{M}$ is a Lie group of dimension m and $\dot{q}_i \in \mathbb{R}^m (\cong T_{q_i}\mathcal{M})$ is a constant velocity (an element in the tangent space of $\mathcal{M}$ at q). The function $\texttt{Exp} : \mathbb{R}^m \to \mathcal{M}$ maps vector elements to the manifold with its inverse being $\texttt{Log} : \mathcal{M} \to \mathbb{R}^m$. The $\texttt{Between} : \mathcal{M} \to \mathcal{M}$ operation is defined as $\texttt{Between}(q_a, q_b) = q_a^{-1} \circ q_b$ and computes the element that would *move* q_a to q_b. Forward integration in a Lie group is defined as $q_{i+1} = q_i \circ \texttt{Exp}(\dot{q}_i \Delta t_i)$.

IV. FOUNDATIONS: INFERENCE VIA FACTOR GRAPHS

Factor graphs (FGs) have been highly adopted for sensor fusion, state estimation, and localization problems. Given the relationship between estimation and control, recent work also explores their use in control and planning [4, 29, 30].

In probabilistic inference, the objective is to find the set of values θ given events e. The posterior density of θ is computed via Bayes rule: $p(\theta|e) \propto p(\theta)p(e|\theta)$, where $p(\theta)$ is the prior on θ and $p(e|\theta)$ is the likelihood function. Given a prior and a likelihood, the optimal solution is found by the *maximum a posteriori* (MAP) operation:

$$\theta^* = \arg\max_\theta p(\theta|e) = \arg\max_\theta p(\theta) l(\theta; e),$$

where the likelihood function $l(\theta; e) = p(e|\theta)$ specifies the probability of events e given θ. The general likelihood function for non-linear FGs is defined as: $l(\theta; e) \propto \exp(\frac{1}{2}||h(\theta, e)||^2_\Sigma)$, where h is a *measurement function* with covariance Σ.

Formally, a FG is a bipartite graph $FG = (\Theta, \mathcal{F}, \mathcal{E})$ with two types of nodes: factors $f_i \in \mathcal{F}$ and variables $\theta_i \in \Theta$. FG edges are always between factor nodes and variable nodes. Given an FG, its posterior distribution is:

$$p(\theta|e) \propto \prod_{n=0}^{N} f_n(\theta_n).$$

The MAP estimate can be reduced to a non-linear least squares problem and solved with standard solvers. Most robotic problems require the solver to operate at high frequency and incor-

porate new data on demand. Standard solvers are not enough in robotics, as they do not take advantage of the sparsity or the *incremental* nature of robotic problems. To alleviate this, iSAM [19] exploits the problem's structure imposed via the FG. iSAM uses a Bayes tree to avoid re-linearization of variables unaffected by a new measurement. Finally, a FG implies some level of discretization, which is also present in other estimation and trajectory optimization approaches. The following subsections review FG representation from the literature for estimation and planning/control.

A. Trajectory Estimation

Fig. 2 presents a typical FG for past trajectory estimation. It computes $p(\theta_{TE}|e) \propto f^{TE} = f^{traj} f^{estimation}$ where $f^{traj} = \prod_{i=0}^{t} f_i^{robot}(\theta_i)$ is the trajectory derived from the robot's model and $f^{estimation} = \prod_{i=0}^{t} f_i^{estimation}(\theta_i)$ are factors that incorporate the measurements. Typically, the underlying discretization is given by the frequency of the measurements. Discretization can be problematic for complex dynamical systems. To avoid discretization of the dynamics, methods often employ a Gaussian Process (GP) to model the system, achieving continuous-time trajectories but limiting applicability to holonomic robots with linear expressions.

Benefits of Trajectory Estimation vs Filtering While a controller only requires the latest state, methods such as Kalman/particle filters, which provide incremental estimates of the latest state, solve the Bayesian *filtering* problem. The latest robot state estimate, however, of an optimal solver for the *most likely trajectory* problem, which STELA focuses on, can be different (and more accurate/less uncertain) than that of an optimal solver for *filtering*. Solving for the *most likely trajectory* is typically computationally more expensive than solving *filtering*. Given the least square approximations, factor graph frameworks allow solving such problems online. STELA takes advantage of this to provide high-frequency MLE updates of the robot's past trajectory. In this way, it provides better estimates of the robot's latest state, and future controls are simultaneously co-optimized based on these improved state predictions.

B. Trajectory Optimization as a Motion Planner

Motion planning can be seen as an optimization problem where the cost of the trajectory $\mathbf{cost}(\tau)$ produced by the plan p_T is minimized subject to i) start state condition: $\tau(0) = x_0$, ii) goal condition: $\tau(T) \in X_G$, iii) dynamics condition: $\dot{x}_t = \hat{f}_\rho(\hat{x}_t, u_t)$ and iv) collision-free condition $\tau_{\hat{f}_\rho} \in \mathbb{X}_{\mathrm{f}}$. Additionally, problem-specific constraints can be added, i.e., smoothness on the controls, energy minimization, etc.

FGs have also been used for motion planning of holonomic systems. In the factor graph setting, trajectories can be obtained via: $p(\theta_{Pl}|e) \propto f^{robot} f^{obstacle} f^{prior}$ [30], where f^{robot} are the dynamical system factors, $f^{obstacle}$ the obstacle avoidance factors and f^{prior} the (fixed) start and goal conditions. Fig. 3 provides an example.

For a general dynamical system, each factor f^{robot} corresponds to solving a *steering function*. Dynamics linearization has been proposed as an alternative to solving a steering function for certain systems [43, 10]. In practice, finding a feasible solution depends both on the quality of the prior and the complexity of the environment. In previous factor graph approaches, these limitations are alleviated by using a holonomic robot modeled via a Gaussian Process and relying on random re-initializations.

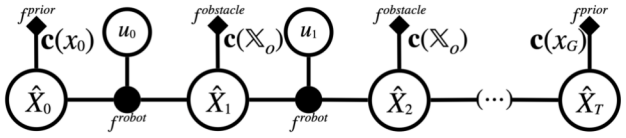


Fig. 3: An FG for robot planning employs the robot's model $\dot{x}_t = \hat{f}_\rho(\hat{x}_t, u_t)$ on a dynamics factor to compute a trajectory of T states, starting in $\hat{x}_0$ and ending in the goal region $\bar{X}_T \in X_G$. Beyond the ternary dynamics factor, there are costs imposed for the optimization by unary factors for obstacle avoidance ($\mathbf{c}(\mathbb{X}_o)$) over the intermediate state variables ($\hat{X}_1 : \hat{X}_{T-1}$), a state prior for the initial state ($\mathbf{c}(x_0)$) and a goal region prior for the final state ($\mathbf{c}(X_G)$).

FGs can suffer from local minima, making any solution heavily dependent on the *initial guess*. For the motion planning problem, a good initialization is not always trivial: a dynamically feasible solution may be in collision while a collision-free one may not be feasible.

V. Simultaneous Trajectory Estimation and Local Adaptation (STELA)

The STELA framework uses numerical optimization for simultaneously solving trajectory estimation and control. It introduces a general FG representation tailored for kinodynamic trajectory following, given feasible desired plans generated by an SBMP. Fig. 4 presents the overarching system and the processes it involves. Offline, a system identification process builds a dynamics model $\hat{f}_\rho(x_t, u_t)$ that bridges the gap with the target robot (see the Experiments Section for this process, which is based on FG tools). Given the model, a kinodynamic SBMP is called to solve a motion planning query given the available environment map. The resulting feasible plan p_T from the planner is forwarded to the STELA module, which also consumes from a perception system online observations $z(t)$ regarding the robot's poses. STELA internally generates improved estimates $\bar{x}(0:t)$ of past robot states and forwards controls $u(t)$ to the robot that minimize deviation from the planned trajectory, move the robot to the desired goal region X_G, and avoid collisions.

A. Initialization of desired trajectory via SBMP

Given the identified robot model $\hat{f}_\rho(x_t, u_t)$, an environment map that identifies obstacle regions $\mathbb{X}_{\mathrm{o}}$ and a motion planning query specifying x_0 and X_G, the approach calls an asymptotically optimal kinodynamic Sampling-Based Motion Planner (SBMP) [23, 27] tasked to generate a feasible, collision-free trajectory $\tau_{\hat{f}_\rho}(x_0, p_T)$ that solves the query on the provided map for the given model.

The output of the planner is treated as the desired trajectory. It is represented via a discretized graphical representation $\mathcal{G} =$

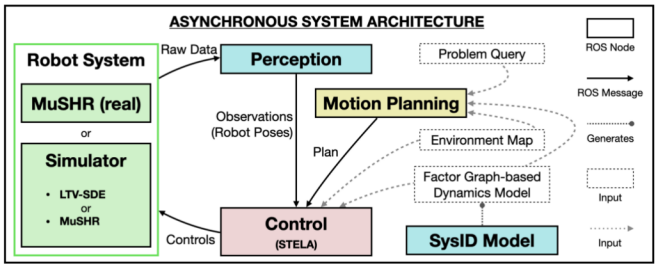


Fig. 4: Asynchronous system architecture: Offline, a system identification process generates a FG-based dynamics model of the robot system. A motion planner receives the dynamics model, the environment, and a motion planning query as input to generate a desired solution plan that addresses the query for the given map and model. Upon completion, the desired plan, the model, and the environment are sent to STELA. Online, raw data –i.e., images from external cameras– are processed by a perception process to provide robot state observations to the control module. These observations are used by STELA to estimate the executed trajectory and generate controls to be forwarded to the robot at a high frequency. The closed-loop framework enables the system to adapt to noise dynamically, execution errors, and the gap between the planning model and the real system. In the accompanying experiments, the robot system is either a real MuSHR robot or a simulated system, where both an idealized LTV-SDE robot and an analytical dynamics model of a MuSHR robot are considered.

(N, E) where a node $n_i \in N$ represents a reachable, collision-free state $x_i = [q_i, \dot{q}_i]^T$ and an edge $e_{ij} = \{n_i, n_j\} \in E$ contains a control, duration pair $(u, \Delta t)$ that drives the robot from state n_i to state n_j according to $\hat{f}_\rho$. An upper threshold for the duration Δt of edges $e_{ij} \in E$ of the desired trajectory is applied (0.5sec for the LTV-SDE system and 0.1sec for MuSHR in the accompanying experiments). If a control is applied for longer than Δt in the solution SBMP trajectory, then it is broken into multiple smaller edges in the graphical representation of the desired trajectory so that none exceed the duration threshold. In this way, the number of discrete states used to initialize STELA is not fixed, and it is adaptive to the output solution of the SBMP.

B. The STELA Factor Graph

STELA builds on top of the incremental smoothing and mapping (iSAM) framework [19] and is executed at a high frequency to consume robot measurements $z(t)$ that arrive asynchronously. STELA is initialized by converting each edge of the desired trajectory into a *dynamics factor graph* as in Fig. 5. The proposed FG includes six different factor types. Each factor is defined as $f^j(\cdot) \propto \exp(\frac{1}{2}||h^j(\cdot)||^2_\Sigma)$ for a factor-specific error function $h^j(\cdot)$. All the FG variables $q_i, \dot{q}_i, u_i$ and Δt_i are initialized according to the desired trajectory.

The **integration factor** operates a configuration q_i, the velocity $\dot{q}_i$, the duration Δt_i, and the predicted next configuration $q^{pred}_{i+1} = q_i \circ \texttt{Exp}(\dot{q}_i \Delta t_i)$. The error function is then defined as $h^{integration}(q_{i+1}, q_i, \dot{q}_i, \Delta t_i) = \texttt{Log}(\texttt{Between}(q^{pred}_{i+1}, q_{i+1}))$.

The **dynamics factor** explains the evolution of the velocity given the control input. The control input u_i is used to obtain an acceleration in the local frame via a system-specific function $\ddot{q} = f(u_i)$. The error function uses the predicted velocity term $\dot{q}^{pred}_{i+1} = \dot{q}_i + \ddot{q}\Delta t$ to compute the error function: $h^{dynamics}(\dot{q}_{i+1}, \dot{q}_i, u_i, \Delta t_i) = \dot{q}^{pred}_{i+1} - \dot{q}_{i+1}$. As $\dot{q} \in \mathbb{R}^m$, an Euler integration scheme is sufficient.

The **observation factor** incorporates observations to estimate the executed trajectory. This work considers observations of the configuration q^i_z that are generated asynchronously as the robot moves. Observations have a known (but noisy) timestamp from which the elapsed time from q_i to the observation is Δt^i_z. The predicted observation is then $q^{pred}_z = q_i \circ \texttt{Exp}(\dot{q}_i \Delta t_z)$ and the error is $h^{observation}(q_i, \dot{q}_i) = \texttt{Log}(\texttt{Between}(q^{pred}_z, q_z))$. Observation factors using measurements of velocities or higher-order magnitudes can be integrated in a similar scheme.

The **prior factor** is a unary factor that penalizes deviations of some factor graph variable v from a given constant value v^{prior}. The error is defined as: $f^{prior}(v) = \texttt{Log}(\texttt{Between}(v, v^{prior}))$. Prior factors are added to $q_i, \dot{q}_i$, and Δt_i variables given the desired trajectory. They are not applied, however, to control u_i variables to provide the ability to STELA to adapt the future trajectory given the latest state estimates. Control variables are only initialized to the value corresponding to the desired trajectory.

The **obstacle factor** introduces a notion of safety by *pushing* states away from obstacles. While the SBMP trajectory initialization is collision-free, a robot may move dangerously close to obstacles due to the model gap and noise. Th factor, following the definition in [30], uses a distance function $d(\cdot)$ to obstacles (from the environment map) and an ϵ threshold:

$$h^{obstacle}(q_i) = \begin{cases} -d(q_i) + \epsilon, & d(q_i) \leq \epsilon; \\ 0, & d(q_i) > \epsilon. \end{cases}$$

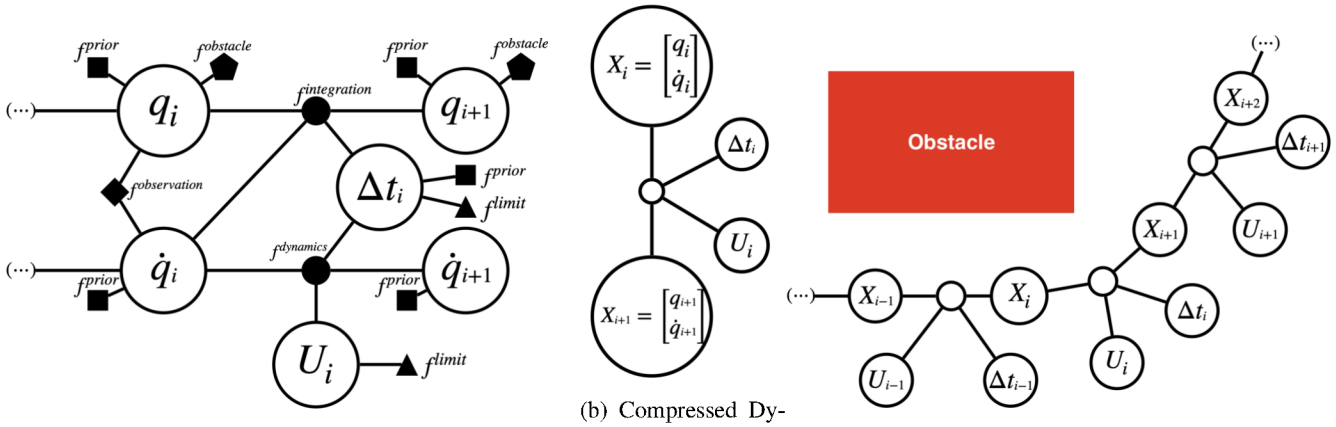
Two obstacle factors are considered: one using a precomputed SDF and another using distance computations from a collision checker, i.e., the PQP library [25]. The SDF factor allows for precomputation of the environment and only considers the distance to the closest obstacle. The PQP factor can be called online and is defined per obstacle, allowing multiple factors to be *active* at the cost of increased computation. Given a reasonable threshold distance ϵ, even in a cluttered environment, most obstacle factors will remain *inactive*. As the error is zero in these cases, re-linearization is unnecessary.

The **limits factor** penalizes values that exceed a predefined value v^{lim}. The error for an upper limit of a variable v is:

$$h^{limit}(v) = \begin{cases} v - v^{lim}, & v \geq v^{lim}; \\ 0, & \texttt{otherwise}. \end{cases}$$

Lower limits can be computed similarly. Multi-value limits are defined element-wise. Limit factors are applied to control variables u_i to guarantee a feasible solution and the duration variables Δt_i to keep them within a reasonable range.

Fig. 5a shows the dynamics factor graph constructed from two SBMP-nodes and one SBMP-edge. Each SBMP-edge introduces one integration factor and one dynamics factor. Obstacle factors are added per configuration. Limit factors constraint $\Delta t_i > 0$ and controls as the SBMP solution must



(a) Dynamics Factor Graph (b) Compressed Dynamics Factor Graph (c) STELA Trajectory

Fig. 5: (Left) The dynamics factor graph corresponding to each edge of the desired trajectory with all associated factors. (Middle) For visualization purposes, the dynamics factor graph is also presented in a *compressed* form, which is symbolized by a hollow factor. (Right) A collision-free trajectory, consisting of a dynamic factor graph sequence, is shown.

respect physical limits. Five prior factors are included, one per configuration q, one per velocity $\dot{q}$, and one for the duration Δt_i to penalize deviations from the desired trajectory. Finally, observation factors are added as new observations arrive.

C. Inference over a Sliding Window

A FG associated with the entire desired trajectory returned by the SBMP planner has at least $2|N| + 2|E|$ FG-variables and $6|N|+5|E|$ FG-factors. At runtime, the number of factors can further and quickly increase due to high-frequency observations. A sliding window approach (Fig. 6) is proposed to alleviate the resulting computational cost of the optimization. The window is divided into a forward horizon using n^{fwd} future nodes of the planned trajectory relative to the current state and a limited past history of n^{hist} past variables.

For the past history, STELA follows similar strategies for trajectory estimation as in the literature [29] (i.e., related to Section IV-A and Figure 2), so that $p(\theta_{TE}|e) \propto f^{TE} = \prod_{j=curr-n^{hist}}^{curr} f_j^{integration} f_j^{dynamics} f_j^{observation}$, where $curr$ is the current state. In contrast to previous approaches (Fig. 2), STELA only estimates the past n^{hist} states, and the forward propagation process is modeled by the combination of the general integration and dynamics factors of the proposed FG. The prior, limit, and obstacle factors are not used for the trajectory estimation component of the optimization.

For the forward horizon, STELA does not perform planning or control from scratch given a naive initialization as in Section IV-B and Fig. 3. Instead, STELA locally adapts the SBMP plan. Local adaptation refers to adapting the controls to minimize the error with the desired trajectory, respect the dynamics, and avoid obstacles. Thus, the forward horizon part of the window performs a local adaptation given: $p(\theta_{LA}|e) \propto \prod_{j=curr}^{curr+n^{fwd}} f_j^{integration} f_j^{dynamics} f_j^{obstacle} f_j^{prior} f_j^{limits}$. The observation factors are not used for the local adaptation component of the optimization.

STELA simultaneously performs trajectory estimation and local adaptation through the inference of:

$$p(\theta_{STELA}|e) \propto f^{TE} f^{LA} \tag{1}$$

D. The STELA Algorithm

Algorithm 1 STELA

Require: $\mathcal{T}_{sbmp}$

FG $\leftarrow$ sbmp_to_fg($\mathcal{T}_{sbmp}$)
curr $\leftarrow 0$
while $\bar{x}(t) \notin X_G$ **do**
  $j \leftarrow max\{curr - n^{hist}, 0\}$ ▷ History
  $k \leftarrow min\{curr + n^{fwd}, length(\text{FG})\}$ ▷ Horizon
  for $i \in \{j, \dots, k\}$ **do** ▷ *Lookahead*
    $\hat{x}_i \leftarrow$ FG.estimate(i)
    $S_i \leftarrow$ FG.covariance(i)
  end for
  FG += $f^{observation}(q_{curr}, \dot{q}_{curr}, z_{new})$ ▷ Add z_{new}
  $\hat{dt}_{curr} \leftarrow$FG.estimate($\Delta t_{curr}$)
  $\hat{u}_{curr} \leftarrow$FG.estimate($u_{curr}$) ▷ Adapt control
  send_control($\hat{u}_{curr}$)
  if (clock() - prev) $> \hat{dt}_{curr}$ **then**
    prev = clock()
    FG-= $\{(q_j, \dot{q}_j, u_j, \Delta t_j\}$ ▷ Remove history
    ▷ Add to Forward Horizon
    FG += { $f^{integration}(q_{k+1}, q_k, \dot{q}_{k+1}, \Delta t_k)$, $f^{dynamics}(\dot{q}_{k+1}, \dot{q}_k, u_{k+1}, \Delta t_k)$, $f^{prior}(q_{k+1}), f^{prior}(\dot{q}_{k+1}), f^{prior}(\Delta t_k)$, $f^{limits}(u_{k+1}), f^{limits}(\Delta t_k), f^{obstacle}(q_{k+1})$ }
    curr++ ▷ Move to next state
  end if
end while

At the beginning of the execution, the initial n^{fwd} nodes of the desired trajectory $\mathcal{T}_{sbmp}$ are converted into a factor graph FG. At each iteration, while the goal has not been reached, STELA performs the following operations: lookahead, trajectory estimation, addition of observations, and local adaptation.

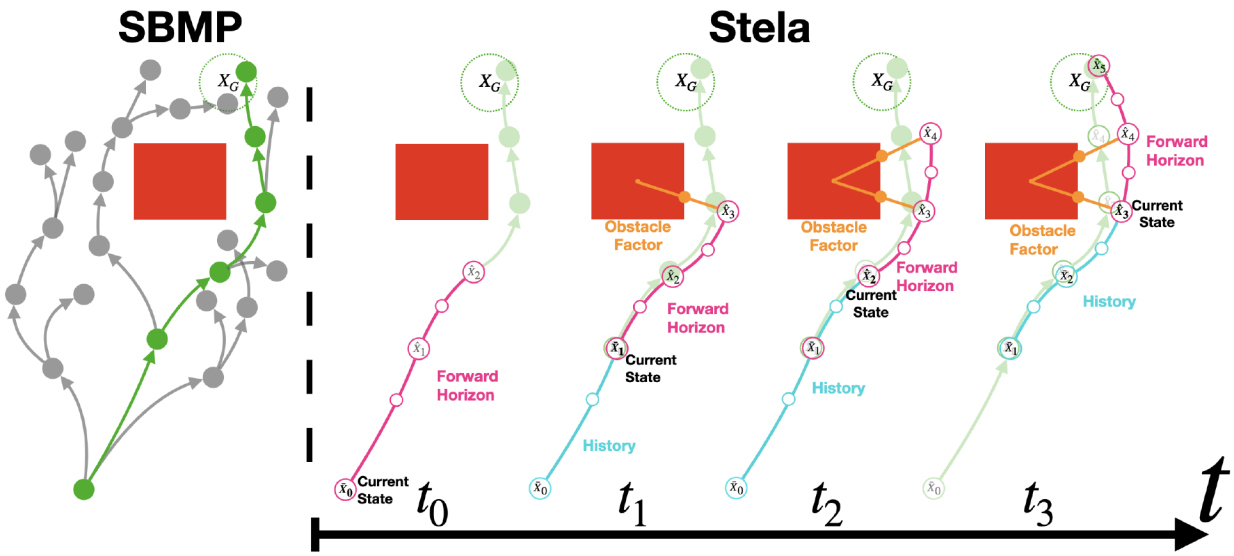


Fig. 6: STELA with a sliding window in action: Offline, the SBMP generates a graphical representation from which a feasible, collision-free, goal-reaching trajectory is obtained. At initialization, the factor graph converts forward-horizon nodes (image: $n^{fwd} = n^{hist} = 2$) into a dynamics factor graph and starts execution. Running at a given frequency, at each iteration, STELA updates the factor graph, incorporating new observations and moving the window. The window update at time t_i is determined by comparing the variable Δt_i to the elapsed time since the last window update. Window updates incorporate the next node from the planned trajectory and (if necessary) delete history nodes.

STELA seeks to exploit the incremental nature of the iSAM2 algorithm by locally updating and querying the factor graph as necessary. The lookahead step obtains the estimated states of the forward horizon alongside their associated error. New observations are added to the corresponding state, which may change the estimation of q_i,q_{i+1},u_i or Δt_i; obtaining the current estimate for u_i and Δt_i. When the estimated duration of the current edge Δt_i elapses, STELA *moves* to the next node by a) deleting the history variables and associated factors to keep n^{hist} forward variables, b) adding the next node from the planned trajectory and c) changing the current node.

E. STELA *vs Previous Factor Graph-based Approaches*

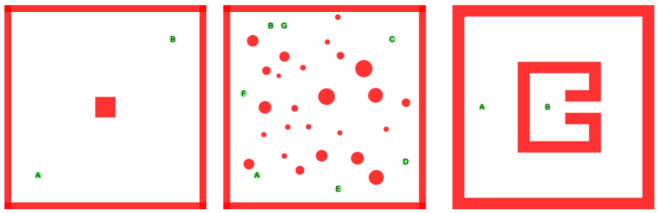
STELA can be seen as generalization of both STEAP [29] and SCATE [22] but has also unique features.

A. The formulations of STEAP and SCATE restrict their applicability to holonomic dynamical systems. In the case of STEAP, the dynamics are modeled via a Gaussian Process. STELA's formulation allows the use of general dynamical systems expressed via first or second-order analytical expressions.

B. The prior approaches collapse q and q_t into a single factor, but STELA introduces separate factors, which results in a better exploitation of the sparsity characteristics of FGs.

C. The sliding window mechanism keeps the number of factors in the FG relatively constant (up to the number of observation factors). This allows for predictable, high-frequency updates and better performance over long trajectories and complex environments.

D. STELA utilizes Lie operations for the integration factor, enhancing the approximation of dynamics and accelerating performance. This also renders STELA broadly applicable to a wide range of robotic systems.



(a) Simple Obstacle (b) Forest (c) Bug Trap

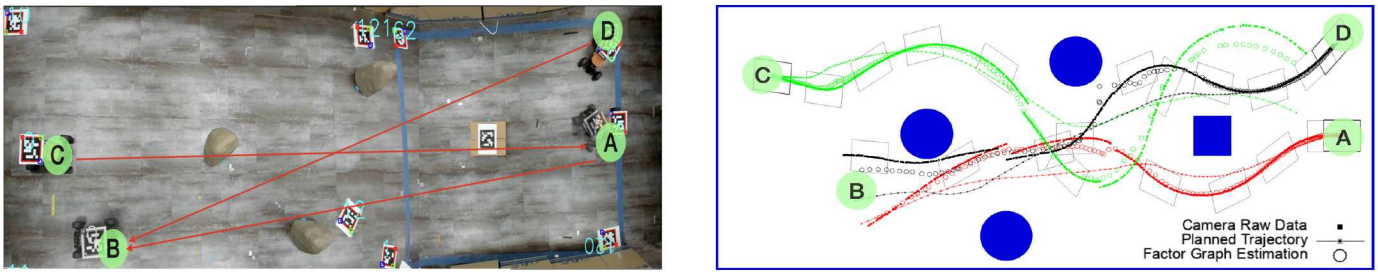
Fig. 7: Simulated environments used for experiments with the LTV-SDE and MuSHR models. Letters indicate candidate starts and goals. The Simple Obstacle environment is a basic setup. Forest evaluates performance among many obstacles with some narrow passages. The Bug Trap is challenging due to the long, narrow passage.

E. Planning via an optimization approach requires continuously solving a Boundary Value Problem (BVP) (or access to a steering function). This can result in solutions with local minima or the inability to converge in systems with non-linear dynamics. A core insight behind STELA is that this challenge can be simplified if a feasible initiation from an SBMP is used, where the SBMP has explored the state space for high-quality, feasible, and collision-free solutions.

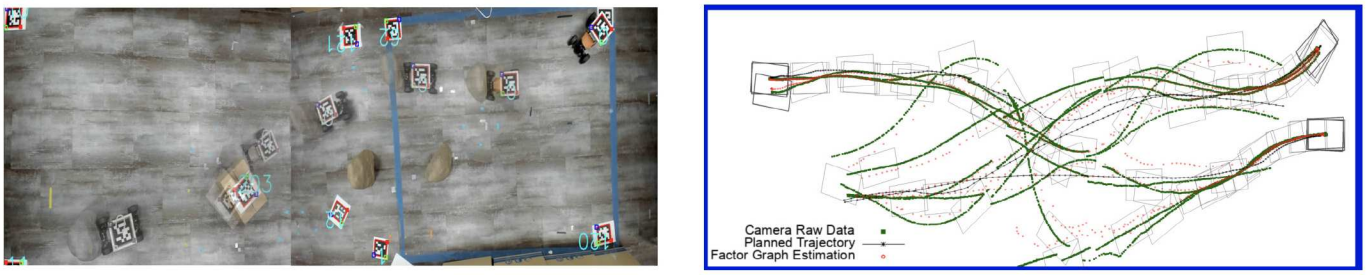
VI. EXPERIMENTS

A. *Experimental setup*

STELA is tested in simulation in the environments shown in Fig. 7 and with a real MuSHR [40] robot in the environments shown in Fig. 1 and 8. The system is tested against four levels of state space noise in simulation: $\dot{x}_t = f(x_t, u_t) + N(0, I \cdot \sigma_i^x)$, and four levels of observation noise $z_t = h(x_t) + N(0, I \cdot \sigma_i^z)$, where I is the identity matrix of appropriate dimension. Simulated systems are implemented as factor graphs using the GTSAM [6] library. Initial feasible trajectories are obtained using the AO-RRT kinodynamic planner [23] from the ML4KP

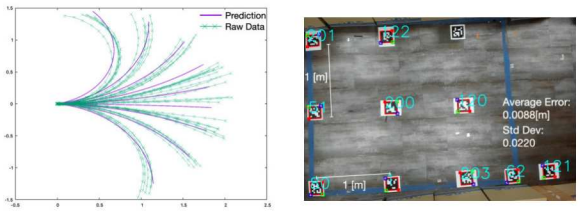


Multiple obstacles



Movable boxes

Fig. 8: Experiments on a real MuSHR. (Top) The robot navigates between (A-B), (C-A), and (D-B), avoiding obstacles. Initial poses in color, and final poses in gray. (Bottom) The robot follows a desired trajectory planned without obstacles. During execution, the environment has movable obstacles. Similar to the Ramp experiment of Figure 1, this setup tests STELA under partial observability.



(a) MuSHR (real) SysId (b) Observation Noise

Fig. 9: (Left) System Identification results for the real MuSHR. The model closely predicts the behavior of the robot along constant control trajectories, but a gap is still present. (Right) Observations z_t are camera estimates of the robot's pose with the highest level of **observation noise** (σ_3^z) chosen to match the real-world setup.

library. Both STELA and the comparison point SCATE are implemented via GTSAM with Threading Building Blocks [35], allowing parallelization for specific functions. Both algorithms are executed in a server with 72 cores, but each experiment is limited to 8 cores (the number of cores found in most computers) for fair comparison.

All experiments are performed in a ROS-based system, where STELA and the comparing approaches are implemented as standalone nodes. All communication within nodes and the environment (either simulated or real) is performed through ROS messages, introducing additional unmodeled noise as time delays or lost messages. The update functions in STELA and SCATE are implemented as ROS timers with a given frequency, which may not be respected due to extra computation time for the algorithm or from external sources.

The **robot systems** considered for experiments are:

LTV-SDE adopted from [45] (eq. 50), $q, \dot{q} \in \mathbb{R}^2$. The controls $u \in [-0.2, 0.2] \times [-0.2, 0.2]$ represent acceleration.

MuSHR modeled as a second order system, where $q \in SE(2)$ and $\dot{q} \in \mathbb{R}^3$; the control $u \in \mathbb{R}^2$ corresponds to acceleration and steering angle.

For the real MuSHR, a **system identification** process to close the model gap is first performed. The parameters ρ to be identified are the acceleration gain and the angular velocity gain. In particular, using observed trajectories τ of the system under a known plan $p(T)$, the parameters ρ are estimated so that the predictions given by the model best match the observations. An additional step was needed for the steering angle, as it has a non-linear bias. A polynomial fits the *effective* steering angle vs the input. The system identification is solved via least squares optimization on a factor graph using the integration and dynamics factor. The controls correspond to the executed plan and are imposed via a prior factor. The steering angle in the trajectories is defined using a five-degree polynomial, which is fit from the same data. Figure 9a visualizes the raw data for the real platform and the prediction for constant control trajectories of fixed duration after system identification. Asynchronous observations $\hat{x}_{i+\epsilon}$ of the robot's state are used for the sys. id process where $\epsilon \in [0, 1]$ and assuming a noise distribution $\mathcal{N}(0, \sigma)$, between states x_i and x_{i+1}. The initial guess is obtained by forward propagating $\hat{f}_{\rho_0}(x_0, u)$ for the duration of the plan.

The simulated MuSHR uses the same identified model as the real system with added noise in $\dot{q}$. Observations z_t are camera estimates of the robot's pose. The highest level of **observation noise** σ_3^z matches the real-world setup, where ArUco markers [37] are positioned 1m apart on a grid (Fig. 9b). The mean error and the standard deviation of these pose estimation were measured given ground-truth.

Comparison Points: Table V shows the number of problems (start-goal queries) per scene selected to test STELA against the alternatives. Multiple repetitions per query are performed.

LTV-SDE - Simple Obstacle																
	Open Loop				SCATE-Naïve				SCATE-SBMP				STELA			
	σ_0^x	σ_1^x	σ_2^x	σ_3^x	σ_0^x	σ_1^x	σ_2^x	σ_3^x	σ_0^x	σ_1^x	σ_2^x	σ_3^x	σ_0^x	σ_1^x	σ_2^x	σ_3^x
σ_0^z	1.0	0.24	0.0	0.0	1.0	1.0	1.0	0.75	0.88	0.88	0.88	0.88	1.0	1.0	1.0	1.0
σ_1^z					1.0	1.0	0.75	0.50	1.0	1.0	1.0	0.88	1.0	1.0	1.0	0.95
σ_2^z					1.0	1.0	0.75	0.75	1.0	1.0	0.88	0.75	1.0	1.0	1.0	1.0
σ_3^z					0.75	1.0	1.0	1.0	1.0	1.0	1.0	0.88	1.0	1.0	0.95	0.95

TABLE I: Success rates for the LVT-SDE on the Simple Obstacle scene for the different approaches. Columns correspond to different techniques and different levels of actuation noise. Rows correspond to different levels of observation noise.

LTV-SDE - Forest																				
	Open Loop				SBMP Replanning				SCATE-Naïve				SCATE-SBMP				STELA			
	σ_0^x	σ_1^x	σ_2^x	σ_3^x	σ_0^x	σ_1^x	σ_2^x	σ_3^x	σ_0^x	σ_1^x	σ_2^x	σ_3^x	σ_0^x	σ_1^x	σ_2^x	σ_3^x	σ_0^x	σ_1^x	σ_2^x	σ_3^x
σ_0^z	1.0	0.0	0.0	0.0	0.9	0.25	0.1	0.1	0.40	0.40	0.50	0.35	1.0	0.95	0.90	0.35	1.0	1.0	1.0	0.96
σ_1^z					0.95	0.15	0.1	0.1	0.40	0.40	0.35	0.40	1.0	0.95	0.85	0.35	1.0	1.0	1.0	0.96
σ_2^z					0.9	0.25	0.1	0.1	0.35	0.30	0.40	0.40	1.0	1.00	0.90	0.20	1.0	1.0	1.0	0.98
σ_3^z					0.8	0.15	0.1	0.1	0.30	0.25	0.30	0.25	0.9	0.75	0.65	0.25	1.0	1.0	1.0	0.92

TABLE II: Success rates for the LVT-SDE on Forest for the different approaches.

LTV-SDE - Bug Trap												
	Open Loop				SCATE-SBMP				STELA			
	σ_0^x	σ_1^x	σ_2^x	σ_3^x	σ_0^x	σ_1^x	σ_2^x	σ_3^x	σ_0^x	σ_1^x	σ_2^x	σ_3^x
σ_0^z	1.0	0.0	0.0	0.0	1.0	0.88	0.88	0	1.0	1.0	1.0	0.72
σ_1^z					1.0	1.0	0.88	0.13	1.0	1.0	1.0	0.80
σ_2^z					1.0	0.88	0.62	0	1.0	1.0	0.96	0.84
σ_3^z					1.0	0.75	0.75	0.13	1.0	0.96	0.92	0.56

TABLE III: Success rates for the LVT-SDE on the Bug Trap. SCATE-Naïve fails to initialize with a feasible plan and, therefore, is never successful.

MuSHR (sim) - Simple Obstacle								
	Open Loop				STELA			
	σ_0^x	σ_1^x	σ_2^x	σ_3^x	σ_0^x	σ_1^x	σ_2^x	σ_3^x
σ_0^z	1.0	0.1	0.0	0.0	1.0	1.0	0.80	0.72
σ_1^z					1.0	1.0	0.80	0.72
σ_2^z					1.0	1.0	0.80	0.80
σ_3^z					0.96	1.0	0.80	0.80

TABLE IV: Success rates for the sim. MuSHR on Simple Obstacle.

The baseline comparison point is open-loop execution of the desired trajectory. SCATE was also chosen as a comparison point. SCATE was originally implemented in MATLAB, and for consistency and performance purposes, it was re-implemented in C++, similar to STELA, reutilizing similar components. Two variations of SCATE are considered. The first follows the original SCATE by initializing the factor graph with a naïve straight-line plan from start to goal. The second variant is initialized with the same desired plan from the SBMP as the proposed STELA approach.

	Trajs	Reps
Simple Obst.	1	5
Forest	10	10
Bug Trap	2	5

TABLE V: Number of trajectories and repetitions for each scene.

Since SCATE requires a constant dt between the states of the factor graph, the variable control durations of this tree had to be normalized by performing a weighted average operation over the edges and the states. Both implementations of SCATE use the same obstacle factor as STELA and also include a factor to ensure that the controls are within the system's limits, which was observed to assist their performance. The SCATE algorithm is only applicable to the LTV-SDE system but not to MuSHR.

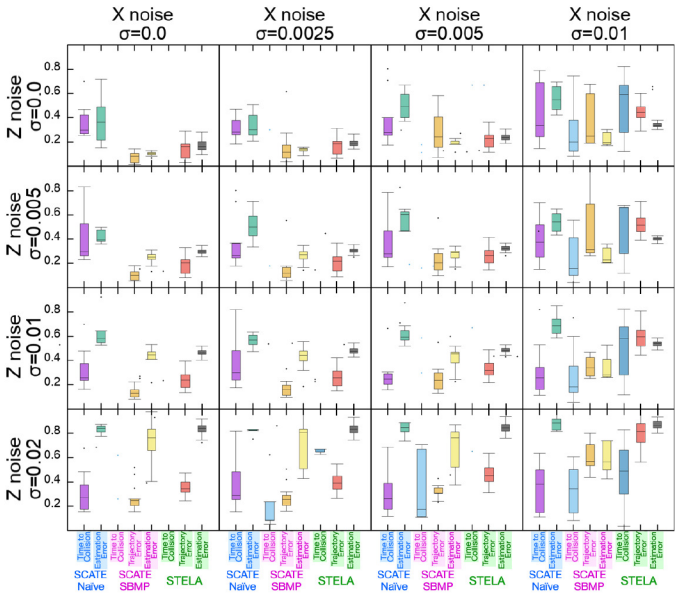
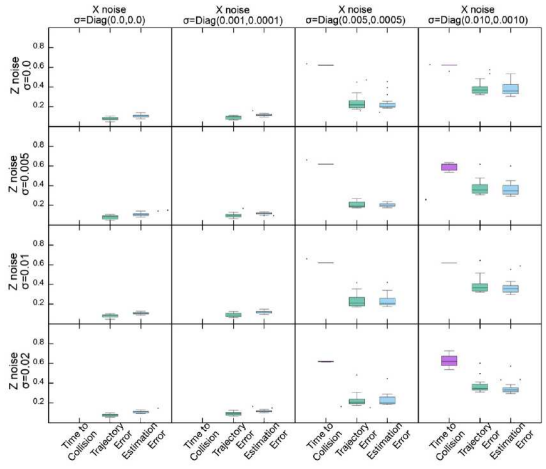
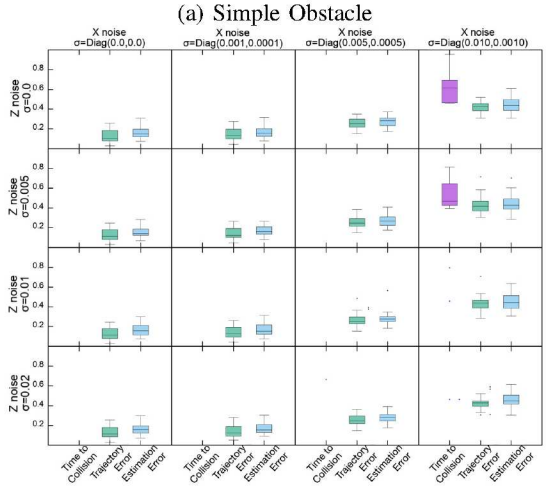


Fig. 10: LTV-SDE Forest - Comparison of the Time to Collision, Normalized Trajectory Error, and Estimation Error between SCATE-Naïve, SCATE-SBMP, and STELA. Trajectory Error is skipped since SCATE-Naïve does not have an initial trajectory to track.

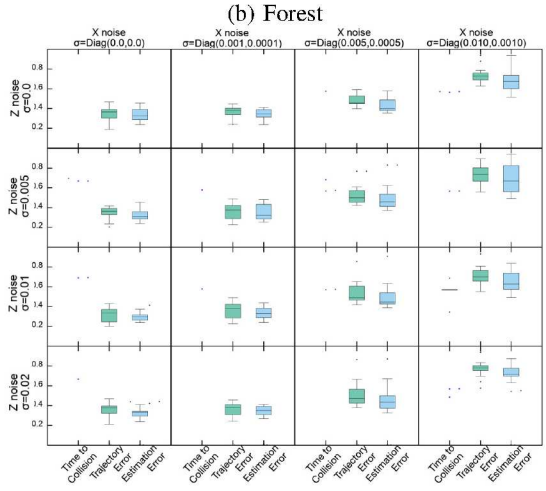
Metrics: Four metrics are considered during the evaluation. *Success:* Ratio of experiments where the robot reached the goal without collisions; the most critical metric for safety. *Cost:* Normalized duration of executed trajectories; normalized given the minimum duration solution per problem. *Estimation Error:* Error between ground truth and estimated trajectory; ground truth is not available for the real MuSHR. *Computation:* Time to perform an update for each algorithm; this metric reports only the time taken by the optimizer.



(a) Simple Obstacle



(b) Forest



(c) Bug Trap

Fig. 11: STELA results for MuSHR (sim). Three normalized metrics reported. *Time to collision* is the rate of a trajectory traversed before a collision (no data if the success rate is 100%). *Trajectory error* (lower is better) is the L2-norm between the planned and executed trajectory, normalized over the highest error. *Estimation error* (lower is better) is the L2-norm between the estimated trajectory and the ground truth, normalized over the highest error. STELA exhibits a very natural and slow degradation in performance as noise increases.

MuSHR (sim) - Forest								
	Open Loop				STELA			
	σ_0^x	σ_1^x	σ_2^x	σ_3^x	σ_0^x	σ_1^x	σ_2^x	σ_3^x
σ_0^z	1.0	0.0	0.0	0.0	1.0	1.0	1.00	0.88
σ_1^z					1.0	1.0	0.96	0.92
σ_2^z					1.0	1.0	0.98	0.94
σ_3^z					0.98	1.0	1.00	0.96

TABLE VI: Success rates for the sim. MuSHR on Forest.

MuSHR (sim) - Bug Trap								
	Open Loop				STELA			
	σ_0^x	σ_1^x	σ_2^x	σ_3^x	σ_0^x	σ_1^x	σ_2^x	σ_3^x
σ_0^z	1.0	0.0	0.0	0.0	1.00	1.00	0.95	0.85
σ_1^z					0.85	0.90	0.80	0.90
σ_2^z					0.90	0.95	0.80	0.70
σ_3^z					0.95	1.00	0.90	0.75

TABLE VII: Success rates for the sim. MuSHR on Bug Trap.

B. Simulation Results

Tables I,II, and III show the success rate of each algorithm per environment for the LTV-SDE system in simulation. Fig. 10 presents the Time to Collision, Normalized Trajectory Error, and Estimation Error for the LTV-SDE Forest scenario. Experiments with zero success rates are not included in the plots. Similarly, tables IV,VI, and VII show success rates for the simulated MuSHR (sim) alongside Figs. 11. The computation time per iteration for the forest environment is shown in Fig. 13 for both STELA and SCATE.

OPEN-LOOP showcases the effects of noise on the system's dynamics, resulting in collisions as soon as the minimum actuation noise level is introduced. No experiments were performed for non-zero observation noise since no estimation was performed in the case of the OPEN-LOOP baseline.

SBMP-REPLANNING is an online replanning strategy, which uses an informed, sampling-based, kinodynamic tree planner to generate new controls every second given the latest

Fig. 12: Aggregate results of the Normalized Trajectory Error and Estimation Error for MuSHR (real) across scenes.

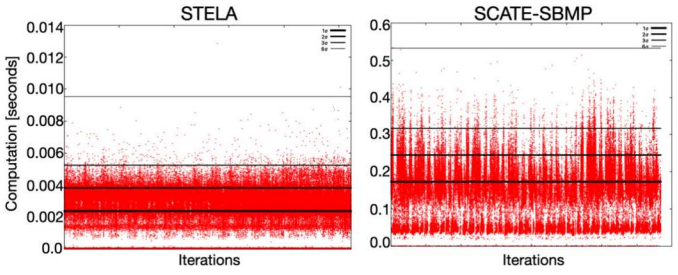


Fig. 13: The computation time per call of STELA and SCATE-SBMP shows a significant advantage in using the incremental computation and sliding window approach of STELA.

observation z_t and the same dynamics model as STELA. This strategy is only tested in the forest environment for the LTV-SDE system (Table II). This strategy degrades fast with higher dynamics noise.

SCATE-Naïve works well in the Simple Obstacle environment for the LTV-SDE system. It faces significant challenges, however, in the Forest environment and fails to find any feasible paths in the Bug Trap environment, with no results reported. Initializing SCATE with the feasible trajectory from the SBMP allows for nearly 100% success for the lowest actuation noise levels across all three environments for the LTV-SDE system. As actuation noise increases, however, the performance of SCATE degrades. In setups with high actuation noise, both versions of SCATE perform significantly worse than STELA. As a reminder, the SCATE comparison point is not directly applicable to the MuSHR system.

STELA gets a 100% success rate on the lowest noise levels while maintaining a high success rate on the most challenging levels across all environments. The increase in cost on the higher noise levels can be explained by the effect of the time variables in the factor graph optimization, which *stretches* the edges as needed so as to avoid collisions and still solve problems. The STELA exhibits very similar behavior for the second-order, non-holonomic MuSHR car as with the idealized, holonomic LTV-SDE system, which highlights its ability to work for different models of robotic systems.

C. Real Experiments

Real experiments are performed with a MuSHR [40] robot in the scenes of Figures 1 and 8. The "multiple obstacles" environment is similar to the setups from simulated experiments, where collisions with obstacles are considered failures. The objective of the "movable boxes" and "ramp" environments is to test the ability to adapt to unmodeled environmental features. The second environment considers a set of *movable* boxes that are not present during planning, and the robot can collide online without considering a failure. The third environment uses a ramp (not present during planning) that the robot needs to traverse to reach the goal. A total of 21 experiments were performed, nine on multiple obstacles (one collision), nine on movable obstacles, and three on the ramp environment (no collisions). Fig. 8 shows qualitative results, while Fig. 12 reports the aggregated deviation from the planned trajectory (trajectory error) and the estimation error.

D. STELA *Ablation*

An ablation study is performed for STELA given the simulated MuSHR system on the Forest environment. The ablation evaluation of the effect of the sliding window size, the use of the duration ΔT as a factor variable, the impact of the obstacle factor, as well as the impact of initializing with the SBMP trajectory versus a naïve initialization. The size of the sliding window is divided into the future horizon and the past history. The reduction of either of these values from the default value of 10 in STELA results in performance degradation. This showcases the effects of performing trajectory estimation, i.e., smoothing, instead of filtering, as well as having a longer horizon than just computing the next control to be executed. STELA without time as a variable also performs worse, which shows the benefit of letting the optimizer adapt the duration of the edges so as to hit the desired states under the effects of noise. STELA without the obstacle factor is more susceptible to collisions, even though it still uses prior factors that push the solution towards the desired, collision-free trajectory. The low-cost trajectories returned from the SBMP are likely, however, to be in close proximity to obstacles, which makes following them susceptible to collisions without an obstacle factor. Two versions of the obstacle factor were tested: an SDF-based one, used by the default approach, and an obstacle factor for multiple obstacles within the distance threshold as computed online, given PQP distance calls. The multi-obstacle, PQP variant slightly underperforms the SDF one. Finally, a naïve initialization (straight-line to the goal) instead of the SBMP trajectory initialization results in significant performance degradation. Frequently, no feasible solution is achieved given the initialization of the approach, and then the algorithm fails. The approach proceeds to work if a solution is found around the initialization.

VII. Discussion

This paper presents STELA, a novel approach that seamlessly integrates the output of kinodynamic sampling-based motion planning with an integrated approach for trajectory estimation and following through factor graph optimization. STELA's effectiveness is highlighted by its ability to dynamically adapt plans based on real-time sensor data, resulting in improved accuracy in trajectory following and robustness against unmodeled environmental variations and noise. The experimental evaluations indicate that STELA enhances the practical applicability of model-based planning and control methods. It also significantly outperforms alternatives in simulated evaluations as noise increases, while achieving desirable high-frequency control update rates.

VIII. Limitations

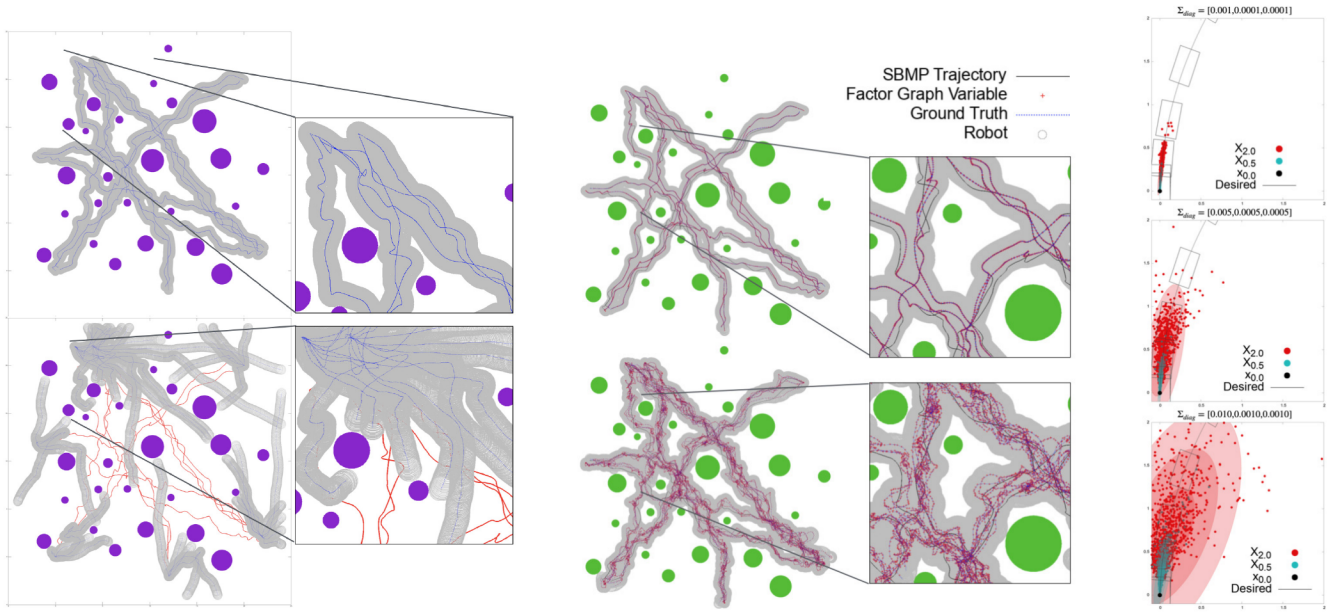
While STELA offers a significant increase in performance and computational efficiency, it shares some limitations of existing factor graph approaches. In the presence of extreme noise and deviation from the desired trajectory, STELA will fail to converge. To test the effects of significant noise, an additional experiment for the MuSHR(sim) robot evaluates STELA under higher levels of dynamics noise, as shown in Table IX, where $\sigma_i^x \in [0.011, 0.015, 0.020]$ (along the columns). The extreme noise level σ_6^x results mostly in failures, where 24% of failures arise from *Indeterminant Linear System Exception*, i.e., the accumulation of numerical errors, which does not occur for the lower noise levels; where

	Mushr - Forest		
	σ_4^x	σ_5^x	σ_6^x
σ_0^z	0.60	0.37	0.05
σ_1^z	0.66	0.35	0.03
σ_2^z	0.61	0.36	0.07
σ_3^z	0.67	0.34	0.07

TABLE IX: STELA's success rates for the MuSHR(sim) robot for high dynamics noise.

	STELA	STELA with variable window Sizes			Time not as Variable	Obstacle Factor		Naive Initialization
		10 Future - 0 Past Nodes	1 Future - 10 Past Nodes	1 Future - 0 Past Nodes		None	PQP	
(σ_1^x, σ_1^z)	1.00	0.95	0.78	0.80	0.95	0.80	1.0	0.38
(σ_2^x, σ_2^z)	1.00	0.95	0.80	0.72	0.97	0.90	1.0	0.37
(σ_3^x, σ_3^z)	1.00	0.73	0.45	0.41	0.77	0.80	0.98	0.36
(σ_4^x, σ_4^z)	0.96	0.42	0.07	0.14	0.62	0.64	0.84	0.33

TABLE VIII: Ablation study of STELA. Comparisons are shown for (left to right): different sliding window sizes, no time factor variable, variations in the obstacle factors, and naive initialization (discretized straight line path from start to goal) instead of the SBMP initialization.



(a) LTV: Open-loop (b) LTV: STELA (c) MuSHR (sim):

Fig. 14: The effects of state-space noise in collision on the Forest environment for the Open-loop baseline (left) and the proposed STELA (middle). The top images correspond to the no-noise setup, while the bottom images correspond to the highest observation and dynamics noise (σ_3^x and σ_3^z). Each image shows 100 trajectories (10 SBMP initializations with 10 repetitions each). STELA can still return collision-free solutions under significant noise, while open-loop execution of the desired trajectories results in collisions. (Left) For a desired trajectory (black line) of MuSHR (Sim), confidence ellipses of 1σ, 2σ, and 3σ of forward propagating for 0.5s and 2s under the three levels (from top to bottom) of dynamics noise for 1K runs.

the only failure mode corresponds to potential collisions. While there is degradation of performance as the noise levels increase, the earlier experiments have established that STELA outperforms alternatives and succeeds under reasonable noise levels as well as for the real-world setup.

Higher noise levels motivate the use of online replanning of the desired trajectory in parallel to executing STELA. Such replanning can be especially useful if there are solution paths along different homotopic classes. It may also be beneficial in the context of dynamic obstacles, which is a setup that was not tested in the current evaluation. STELA's formulation, however, should naturally extend to such setups.

IX. Acknowledgements

Work by the authors in this paper was partially supported by NSF NRT-FW-HTF award 2021628. Any opinions, findings and conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. Kostas Bekris holds concurrent appointments as a Professor of Rutgers University and as an Amazon Scholar. This paper describes work performed at Rutgers University and is not associated with Amazon.

References

[1] Agha-mohammadi, A.a., Agarwal, S., Kim, S.K., Chakravorty, S., Amato, N.M.: Slap: Simultaneous localization and planning under uncertainty via dynamic replanning in belief space. IEEE Transactions on Robotics 34(5), 1195–1214 (2018)

[2] Atreya, P., Biswas, J.: State supervised steering function for sampling-based kinodynamic planning. arXiv preprint arXiv:2206.07227 (2022)

[3] Bähnemann, R., Burri, M., Galceran, E., Siegwart, R., Nieto, J.: Sampling-based motion planning for active multirotor system identification. In: 2017 IEEE International Conference on Robotics and Automation (ICRA). pp. 3931–3938. IEEE (2017)

[4] Barfoot, T.D., Tong, C.H., Särkkä, S.: Batch continuous-time trajectory estimation as exactly sparse gaussian process regression. In: Robotics: Science and Systems. vol. 10, pp. 1–10. Citeseer (2014)

[5] Cunningham, A., Paluri, M., Dellaert, F.: Ddf-sam: Fully distributed slam using constrained factor graphs. In: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 3025–3030. IEEE (2010)

[6] Dellaert, F., Contributors, G.: borglab/gtsam (May 2022), https://github.com/borglab/gtsam)

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

[8] Forster, C., Carlone, L., Dellaert, F., Scaramuzza, D.: Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. In: Robotics: Science and Systems XI (2015)

[9] Gammell, J.D., Srinivasa, S.S., Barfoot, T.D.: Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In: 2015 IEEE international conference on robotics and automation (ICRA). pp. 3067–3074. IEEE (2015)

[10] Goretkin, G., Perez, A., Platt, R., Konidaris, G.: Optimal sampling-based planning for linear-quadratic kinodynamic systems. In: 2013 IEEE International Conference on Robotics and Automation. pp. 2429–2436. IEEE (2013)

[11] Hauser, K.: Motion and Path Planning, pp. 1–11. Springer Berlin Heidelberg, Berlin, Heidelberg (2020), https://doi.org/10.1007/978-3-642-41610-1_165-1

[12] Hoffmann, G.M., Tomlin, C.J., Montemerlo, M., Thrun, S.: Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing. In: 2007 American control conference. pp. 2296–2301. IEEE (2007)

[13] Hönig, W., Ortiz-Haro, J., Toussaint, M.: db-a*: Discontinuity-bounded search for kinodynamic mobile robot motion planning. In: 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 13540–13547. IEEE (2022)

[14] Huang, E., Mukadam, M., Liu, Z., Boots, B.: Motion planning with graph-based trajectories and gaussian process inference. In: 2017 IEEE International Conference on Robotics and Automation (ICRA). pp. 5591–5598. IEEE (2017)

[15] Jiang, F., Marmon, A., De Courten, I., Rasi, M., Dellaert, F.: Probabilistic tracking with deep factors. arXiv preprint arXiv:2112.01609 (2021)

[16] Johansson, R., Robertsson, A., Nilsson, K., Verhaegen, M.: State-space system identification of robot manipulator dynamics. Mechatronics 10(3), 403–418 (2000)

[17] Kaess, M., Ila, V., Roberts, R., Dellaert, F.: The bayes tree: An algorithmic foundation for probabilistic robot mapping. In: Algorithmic Foundations of Robotics IX: Selected Contributions of the Ninth International Workshop on the Algorithmic Foundations of Robotics. pp. 157–173. Springer (2011)

[18] Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J.J., Dellaert, F.: isam2: Incremental smoothing and mapping using the bayes tree. The International Journal of Robotics Research 31(2), 216–235 (2012)

[19] Kaess, M., Ranganathan, A., Dellaert, F.: isam: Incremental smoothing and mapping. IEEE Transactions on Robotics 24(6), 1365–1378 (2008)

[20] Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., Schaal, S.: Stomp: Stochastic trajectory optimization for motion planning. In: 2011 IEEE international conference on robotics and automation. pp. 4569–4574. IEEE (2011)

[21] Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. The international journal of robotics research 30(7), 846–894 (2011)

[22] King-Smith, M., Tsiotras, P., Dellaert, F.: Simultaneous control and trajectory estimation for collision avoidance of autonomous robotic spacecraft systems. In: 2022 International Conference on Robotics and Automation (ICRA). pp. 257–264. IEEE (2022)

[23] Kleinbort, M., Granados, E., Solovey, K., Bonalli, R., Bekris, K.E., Halperin, D.: Refined analysis of asymptotically-optimal kinodynamic planning in the state-cost space. In: 2020 IEEE International Conference on Robotics and Automation (ICRA). pp. 6344–6350. IEEE (2020)

[24] Kschischang, F., Frey, B., Loeliger, H.A.: Factor graphs and the sum-product algorithm. IEEE Transactions on Information Theory 47(2), 498–519 (2001)

[25] Larsen, E., Gottschalk, S., Lin, M.C., Manocha, D.: Fast proximity queries with swept sphere volumes. Tech. rep., Technical Report TR99-018, Department of Computer Science, University of North Carolina (1999)

[26] LaValle, S.M.: Planning algorithms. Cambridge university press (2006)

[27] Littlefield, Z., Bekris, K.E.: Efficient and asymptotically optimal kinodynamic motion planning via dominance-informed regions. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 1–9. IEEE (2018)

[28] Marcucci, T., Petersen, M., von Wrangel, D., Tedrake, R.: Motion planning around obstacles with convex optimization. Science robotics 8(84), eadf7843 (2023)

[29] Mukadam, M., Dong, J., Dellaert, F., Boots, B.: Steap: simultaneous trajectory estimation and planning. Autonomous Robots 43, 415–434 (2019)

[30] Mukadam, M., Dong, J., Yan, X., Dellaert, F., Boots, B.: Continuous-time gaussian process motion planning via probabilistic inference. The International Journal of Robotics Research 37(11), 1319–1340 (2018)

[31] Natarajan, R., Choset, H., Likhachev, M.: Interleaving graph search and trajectory optimization for aggressive quadrotor flight. IEEE Robotics and Automation Letters 6(3), 5357–5364 (2021)

[32] Ortiz-Haro, J., Hönig, W., Hartmann, V.N., Toussaint, M., Righetti, L.: idb-rrt: Sampling-based kinodynamic motion planning with motion primitives and trajectory optimization. arXiv preprint arXiv:2403.10745 (2024)

[33] Paden, B., Čáp, M., Yong, S.Z., Yershov, D., Frazzoli,

E.: A survey of motion planning and control techniques for self-driving urban vehicles. IEEE Transactions on intelligent vehicles 1(1), 33–55 (2016)
[34] Paden, B., Frazzoli, E.: A generalized label correcting method for optimal kinodynamic motion planning. In: Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics. pp. 512–527. Springer (2020)
[35] Pheatt, C.: Intel® threading building blocks. Journal of Computing Sciences in Colleges 23(4), 298–298 (2008)
[36] Ratliff, N., Zucker, M., Bagnell, J.A., Srinivasa, S.: Chomp: Gradient optimization techniques for efficient motion planning. In: 2009 IEEE international conference on robotics and automation. pp. 489–494. IEEE (2009)
[37] Romero-Ramirez, F.J., Muñoz-Salinas, R., Medina-Carnicer, R.: Speeded up detection of squared fiducial markers. Image and vision Computing 76, 38–47 (2018)
[38] Schulman, J., Ho, J., Lee, A.X., Awwal, I., Bradlow, H., Abbeel, P.: Finding locally optimal, collision-free trajectories with sequential convex optimization. In: RSS. vol. 9, pp. 1–10. Berlin, Germany (2013)
[39] Sola, J., Deray, J., Atchuthan, D.: A micro lie theory for state estimation in robotics. arXiv preprint arXiv:1812.01537 (2018)
[40] Srinivasa, S.S., Lancaster, P., Michalove, J., Schmittle, M., Summers, C., Rockett, M., Scalise, R., Smith, J.R., Choudhury, S., Mavrogiannis, C., et al.: Mushr: A low-cost, open-source robotic racecar for education and research. arXiv preprint arXiv:1908.08031 (2019)
[41] Toussaint, M.: Robot trajectory optimization using approximate inference. In: Proceedings of the 26th annual international conference on machine learning. pp. 1049–1056 (2009)
[42] Toussaint, M.: Newton methods for k-order markov constrained motion problems. arXiv:1407.0414 (2014)
[43] Webb, D.J., Van Den Berg, J.: Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics. In: 2013 IEEE international conference on robotics and automation. pp. 5054–5061. IEEE (2013)
[44] Wu, T., Movellan, J.: Semi-parametric gaussian process for robot system identification. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 725–731 (2012)
[45] Zheng, D., Ridderhof, J., Zhang, Z., Tsiotras, P., Aghamohammadi, A.A.: Cs-brm: A probabilistic roadmap for consistent belief space planning with reachability guarantees. IEEE Transactions on Robotics (2024)