InCOpt: Incremental Constrained Optimization using the Bayes Tree

Mohamad Qadri[1], Paloma Sodhi[1], Joshua G. Mangelson[3], Frank Dellaert[2], and Michael Kaess[1]

Abstract— **In this work, we investigate the problem of incrementally solving constrained non-linear optimization problems formulated as factor graphs. Prior incremental solvers were either restricted to the unconstrained case or required periodic batch relinearizations of the objective and constraints which are expensive and detract from the online nature of the algorithm. We present InCOpt, an Augmented Lagrangian-based incremental constrained optimizer that views matrix operations as message passing over the Bayes tree. We first show how the linear system, resulting from linearizing the constrained objective, can be represented as a Bayes tree. We then propose an algorithm that views forward and back substitutions, which naturally arise from solving the Lagrangian, as upward and downward passes on the tree. Using this formulation, InCOpt can exploit properties such as fluid/online relinearization leading to increased accuracy without a sacrifice in runtime. We evaluate our solver on different applications (navigation and manipulation) and provide an extensive evaluation against existing constrained and unconstrained solvers.**

I. Introduction

We address the problem of solving for hard constraints within incremental factor graph optimizers. State estimation and Simultaneous Localization and Mapping (SLAM) are typically formulated as factor graph inference problems where variable nodes represent states and factor nodes encode priors or observation likelihoods as soft constraints. In many scenarios, both priors and observations are better modeled as *hard constraints*. For instance, contact observations when a robot finger touches an object are better modeled as a constraint. How do we efficiently perform inference in the presence of such hard constraints?

Prior work has formulated this problem as incremental constrained smoothing (ICS) [1]. The key idea behind ICS is to solve for a constrained optimization objective *incrementally*, i.e. as new observations arrive at each time step, without having to re-solve from scratch. It does so by leveraging primal-dual methods like the Augmented Lagrangian [2] to split the constrained objective into two alternating primal and dual steps, each of which can be solved efficiently by leveraging sparse, incremental matrix factorization. However, a key limitation of ICS is that it assumes a fixed linearization point for earlier states in the matrix factorization. This makes it unsuitable for nonlinear problems that require frequent relinearizations. These limitations are addressed by iSAM2 [3] that exploits a connection between graphical models and sparse linear algebra. It replaces the sparse matrix factorization with a Bayes tree structure. Incremental edits can be made to the Bayes tree directly allowing fluid/online relinearization. However, iSAM2 does not handle hard constraints and only works with unconstrained objectives.

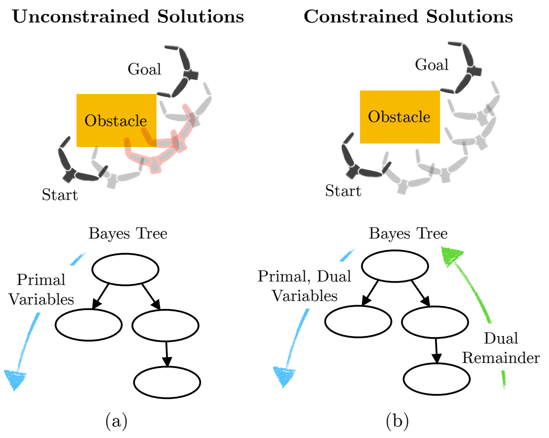


Fig. 1: Unconstrained and constrained factor graph solutions for planning a manipulator to move around obstacles. **(a)** Unconstrained objectives as solved by iSAM2 invoke a single downward pass on the Bayes tree to compute primal solutions. **(b)** Constrained objectives as solved by InCOpt invoke multiple alternating upward and downward passes on the Bayes tree to solve for both primal and dual variables until convergence.

We propose **Inc**remental **C**onstrained **Opt**imization (InCOpt), that leverages the Bayes tree data structure to handle hard constraints with online relinearizations. Our key insight is that the matrix updates in the primal-dual steps can be translated to message passing operations on the Bayes tree. Specifically, each iteration performs alternating upward and downward passes on the Bayes tree, which are equivalent to a forward and backward matrix substitution respectively. Unlike ICS, we do not need to maintain a full matrix, but instead, maintain sub-matrices within each node of the Bayes tree. This is key to enabling online relinearization and more efficient updates.

Our main contributions are:

1) A primal-dual constrained optimization framework formulated as message passing on a Bayes tree.
2) A novel algorithm to efficiently solve the constrained objective with online relinearization without having to recompute solutions from scratch at every time step.
3) Open-source code along with empirical evaluation on different tasks such as navigation and manipulation[4].

II. Related Work

Factor graphs for SLAM: Simultaneous Localization and Mapping (SLAM) has been a central focus for roboticists and others for several decades [4]. Initial solutions used an Extended Kalman Filter which proved to be unscalable due to the curse of dimensionality [5], [6]. [7] and [8] capitalized on the inherent sparsity of the SLAM problem to handle large numbers of variables. However, these methods still required

[1]M. Qadri, P. Sodhi, and M. Kaess are with The Robotics Institute, Carnegie Mellon University, USA. {mqadri, psodhi, kaess}@cs.cmu.edu

[2]F. Dellaert is with The School of Interactive Computing, Georgia Institute of Technology, USA. {frank.dellaert}@cc.gatech.edu

[3]J. Mangelson is with The Electrical and Computer Engineering Department at Brigham Young University, USA. {joshua_mangelson}@byu.edu

This work was partially supported by the Facebook FRAIM program.

[4]Code is available at https://github.com/rpl-cmu/incopt

factorizing large matrices (i.e. Jacobians) at each time step and were dependent on good initialization points.

Incremental unconstrained solutions: Incremental Smoothing and Mapping (iSAM) [9] viewed the problem as probabilistic inference over factor graphs, proposed updating existing jacobians with new measurements, and performing single batch updates only at predefined optimizer steps which allowed for real-time performance. However, the accuracy of the solution was highly dependent on the quality of the linearizations at any time step. iSAM2 [3] moved away from the matrix view to represent the linearized system as a Bayes tree [10]. Now on-time relinearization and partial state updates can be done leading to increased accuracy without a sacrifice of runtime. However, iSAM2 is an unconstrained solver not suited for handling hard constraints that may arise in different robotics applications (especially with factor graphs gaining traction in areas such as planning [11], [12] [13] and control [14], [15], [16]).

Batch constrained solutions: [16] used Sequential Quadratic Programming (SQP) to solve a factor graph containing both soft and hard equality constraints and [17] proposed solving an equality-constrained factor graph using a hybrid elimination procedure. [18] introduced an active-set method to solve for both equality and inequality constraints within a factor graph framework. [19] proposed splitting large-scale non-linear inference problems (focusing on SLAM) into subgraphs with equality constraints between separator variables allowing for distributed optimization using ADMM. [20] viewed incremental SLAM as a constraint/cycle selection problem with loop closure cycles represented as constraints. However, [16]–[20] either solve batch constrained optimization problems or are not easily amenable to *incremental* constrained optimization on factor graphs where previous matrix factorizations are reused. ICS [1] extended iSAM to handle hard constraints. However, it only allowed for batch relinearizations at specific intervals and hence, suffers a performance hit when frequent linearizations of the objective and constraints are required. In this paper, we propose InCOpt which tackles these limitations.

III. Problem Formulation and Notation

Consider the following nonlinear least-squares objective subject to a set of nonlinear hard equality $g(\cdot)$ and inequality $h(\cdot)$ constraints:

$$\begin{aligned}\hat{x} =& \underset{x}{\operatorname{argmin}}\ \frac{1}{2}\sum_{i=1}^{I} ||f_i(x_i) - z_i||^2_{\Sigma_i} \\ \text{s.t.,}\quad & g_q(x_q) = 0 \quad , \quad q = 1, \ldots Q \\ & h_p(x_p) \leq 0 \quad , \quad p = 1, \ldots P\end{aligned} \tag{1}$$

where each f_i, g_q, h_p is a non-linear function of a subset of variables x_i, x_q, x_p of x and $||e||^2_\Sigma \triangleq e^T\Sigma^{-1}e$ is the squared Mahalanobis distance with covariance Σ . We linearize measurement functions $f_i(\cdot)$, $g_q(\cdot)$, $h_p(\cdot)$ about a linearization point x^0 to get the subproblem:

$$\begin{aligned}\Delta^* =& \underset{\Delta}{\operatorname{argmin}}\ \frac{1}{2}\sum_{i=1}^{I} ||A_i\Delta_i - b_i||^2_2 \\ \text{s.t.,}\quad & G_q\Delta_q + g_q(x^0) = 0 \quad , \quad q = 1, \ldots Q \\ & H_p\Delta_p + h_p(x^0) \leq 0 \quad , \quad p = 1, \ldots P\end{aligned} \tag{2}$$

where $A_i = \Sigma_i^{-\frac{1}{2}}F_i$, $b_i = \Sigma_i^{-\frac{1}{2}}(z_i - f_i(x_i^0))$, with F_i being the measurement function Jacobian, Δ the state update vector, G_q and H_p the Jacobians of the equality and inequality constraints, and $g(x^0)$, $h(x^0)$ the constraints evaluated at the linearization point x^0 (see [1] for more details).
We define the set of all non-linear factors:
$\mathcal{F} = f \cup m$ where:
$f = \{f_i\}$ (all soft constraints)
$m = \{g_q \ \cup \ h_p\}$ (all hard constraints).

and define the set of all linearized factors as:

$\partial\mathcal{F} = \partial f \cup \partial m$ where:
$\partial f = \{(A_i, b_i)\}$
$\partial m = \{(M_k, m_k(x^0))\} = \{(G_q, g_q(x^0))\} \cup \{(H_p, h_p(x^0))\}$

We also denote as M_k^j the partial derivative of the hard constraint m_k with respect to variable j evaluated at x^0.

We now write the Augmented Lagrangian [2] for the constrained objective in Eq. 2:

$$\begin{aligned}\mathcal{L}(\Delta, v, u) =& \frac{1}{2}\sum_i ||A_i\Delta_i - b_i||^2_2 + \sum_q v_q^T\left(G_q\Delta_q + g_q(x_q^0)\right) \\ &+ \sum_q \frac{\rho_q}{2}||G_q\,\Delta_q \ + \ g_q(x_q^0)||^2_2 + \sum_p u_p^T\left(H_p\Delta_p + h_p(x_p^0)\right) \\ &+ \sum_p \frac{\rho_p}{2}||\max(H_p\Delta_p + h_p(x_p^0), 0)||^2_2\end{aligned} \tag{3}$$

where ρ_p and ρ_q are the penalty terms, v_q u_p the dual variables associated with each equality, inequality constraint respectively, v, u their concatenation, and the $\max$ operator applied element-wise. We follow the derivation from ICS [1] to get the iterative primal-dual updates in matrix form,

$$R^Ty = G^Tv + H^Tu \tag{4a}$$
$$R\Delta = d - y \tag{4b}$$
$$v^+ = v + \rho(G\Delta + g(x^0)) \tag{4c}$$
$$u^+ = \max\left(0, u + \rho(H\Delta + h(x^0))\right) \tag{4d}$$

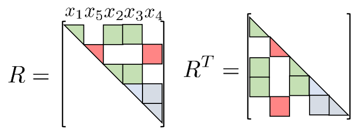
where $[R \mid d]$ results from the QR factorization of the system $[A \mid b]$ with A, b constructed by collecting all A_i, b_i into a single large system. Similarly, $[G \mid g]$, $[H \mid h]$ are constructed by collecting all G_q, g_q and H_p, h_p. In ICS, Eqs. 4a–4d are solved as full matrix updates (see example of the fully constructed matrices in Fig. 2). However, this can cause ICS to accumulate errors in its estimates since it assumes a fixed linearization point when constructing these matrices.

IV. Approach

We propose InCOpt which leverages the Bayes tree data structure as introduced in iSAM2 [3] to handle hard constraints with online relinearizations. Our key insight is that the matrix updates in the primal-dual steps in Eq. 4 can be translated to message passing operations on the Bayes tree.

A. Augmented Gaussian Factor Graph

In iSAM2, linearizing a non-linear factor graph encoding an *unconstrained* objective (i.e no hard constraints) at some value x^0 results in a Gaussian Factor Graph (GFG) representing the linear objective $L(\Delta) = \frac{1}{2}||A\Delta - b||^2_2$. Each factor in the GFG encodes a linearized constraint $[A_i \mid b_i]$. For the



$$d = \begin{bmatrix} d_1 \\ d_5 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} \quad \Delta = \begin{bmatrix} \Delta_1 \\ \Delta_5 \\ \Delta_2 \\ \Delta_3 \\ \Delta_4 \end{bmatrix}$$

$$G^T = \begin{bmatrix} 0 & 0 & 0 \\ \frac{\partial g_2}{\partial x_5} & 0 & 0 \\ 0 & 0 & \frac{\partial h_1}{\partial x_2} \\ 0 & \frac{\partial g_1}{\partial x_3} & 0 \\ \frac{\partial g_2}{\partial x_4} & \frac{\partial g_1}{\partial x_4} & 0 \end{bmatrix} \quad G = \begin{bmatrix} 0 & \frac{\partial g_2}{\partial x_5} & 0 & 0 & \frac{\partial g_2}{\partial x_4} \\ 0 & 0 & 0 & \frac{\partial g_1}{\partial x_3} & \frac{\partial g_1}{\partial x_4} \\ 0 & 0 & \frac{\partial h_1}{\partial x_2} & 0 & 0 \end{bmatrix}$$

Fig. 2: The matrices typically constructed to perform the iterative update in Eq. 4 (corresponding to the example in Fig. 3). We will show how InCOpt performs these calculations by operating directly on the Bayes tree.

Augmented Lagrangian in Eq. 3 we now additionally need to include quadratic augmentation terms $\sum_q \frac{\rho_q}{2}||G_q\Delta_q + g_q(x^0)||_2^2$ and $\sum_p \frac{\rho_p}{2}||\max(H_p\Delta_p + h_p(x^0), 0)||_2^2$ in the GFG. At time step n, an inequality constraint is inactivated (by setting $H_p = 0$ and $h_p(x^0) = 0$) if it is satisfied (i.e. $h_p(x^0) \leq 0$) and activated otherwise. Hence, the inequality augmentation term is equal to:

$$\begin{cases} 0 & h_p(x^0) \leq 0 \\ \frac{\rho_p}{2}||H_p\Delta_p + h_p(x^0)||_2^2 & h_p(x^0) > 0 \end{cases} \tag{5}$$

Each linearized equality and inequality constraint can then be encoded as a factor in the GFG with $A_q = \sqrt{\frac{\rho_q}{2}}G_q$, $b_q = -\sqrt{\frac{\rho_q}{2}}g_q(x^0)$ and $A_p = \sqrt{\frac{\rho_p}{2}}H_p$, $b_p = -\sqrt{\frac{\rho_p}{2}}h_p(x^0)$.

We hence end up with an Augmented Gaussian Factor Graph (A-GFG) encoding a linearized objective $\frac{1}{2}||\mathcal{A}\Delta - \mathcal{B}||_2^2$ where $\mathcal{B} = \{b_i, b_q, b_p\}$ includes all error terms and $\mathcal{A} = \{A_i, A_q, A_p\}$ includes all quadratic costs as well as augmentation terms associated with equality/inequality constraints. We can now re-write the Lagrangian in Eq. 3 as an A-GFG along with linear dual terms:

$$\mathcal{L}(\Delta, v, u) = \frac{1}{2}||\mathcal{A}\Delta - \mathcal{B}||_2^2 + v^T\left(G\Delta + g(x^0)\right) + u^T\left(H\Delta + h(x^0)\right) \tag{6}$$

Fig. 3a shows an example of a constrained objective encoded as a non-linear factor graph and Fig. 3c shows its associated A-GFG.

B. Bayes Tree Construction

Now that we have an A-GFG representation of the problem, how do we construct the Bayes tree? We start with a brief summary of how iSAM2 converts the GFG into a

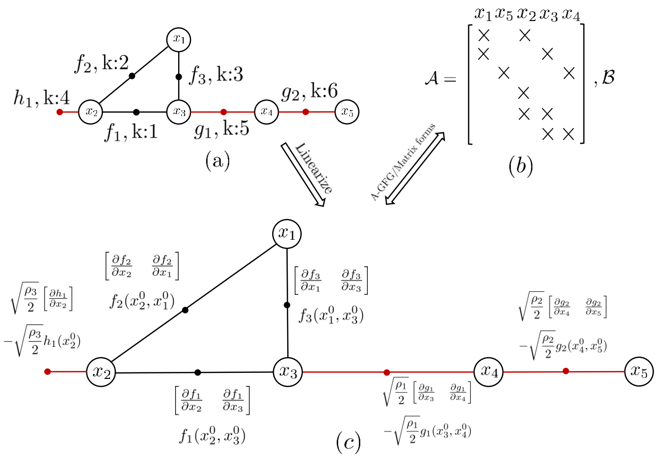


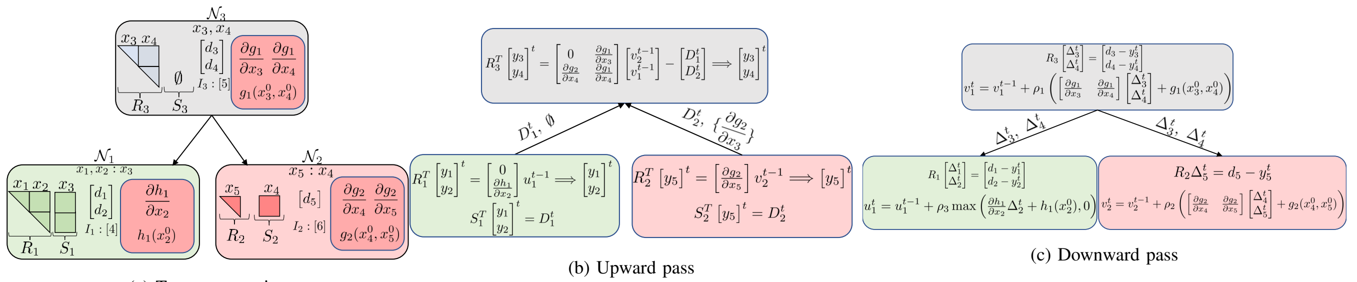
Fig. 3: **(a)** An example of a constrained non-linear factor graph. Each factor is assigned a global index k **(c)** The A-GFG. Each factor encodes the linearization of a non-linear factor. **(b)** shows the equivalent linear system represented by the A-GFG where the elimination order is: x_1, x_5, x_2, x_3, x_4

Bayes tree. The GFG is first converted into a Bayes net using variable elimination. The resulting Bayes net has a special property of being *chordal* that is exploited to find a tree structure over its cliques. This results in the *Bayes Tree* which allows for efficient inference not possible with the original factor graph structure. The Bayes tree $\mathcal{T}$ is composed of nodes $\mathcal{N}$ each representing a clique $\mathcal{C}$ in the underlying Bayes net. We use the subscript c to refer to node-specific elements throughout this paper. Each node contains a conditional density $P(F_c|S_c)$ over a set of frontal variables F_c given a set of separator variables S_c. Under Gaussian assumptions, this conditional density $P(F_c|S_c)$ is effectively a factorized matrix $[R_c \mid d_c]$ stored in the node.

In InCOpt, the process of converting an A-GFG to a Bayes tree is the same. However, in addition to the factorized matrix $[R_c \mid d_c]$, each node also stores the latest linearization ∂m_c of all *constrained factors* m_c *used to form the clique* $\mathcal{C}$ in addition to a list of indices, I_c, globally indexing these factors. These indices are used to retrieve factor-specific information such as the associated Lagrange multiplier u or v and penalty term ρ (example in Fig. 4a).

C. Bayes Tree Updates and Fluid Relinearization

Now that we have a Bayes tree, we would like to be able to update variables in the tree online as their linearization points change. The Bayes tree $\mathcal{T}$ can be effectively seen as a replacement for the full sparse matrix $[R \mid d]$ from Eq.



(a) Tree construction (b) Upward pass (c) Downward pass

Fig. 4: **(a)** The Bayes tree computed from the A-GFG in Fig. 3. We store the latest linearization of all hard constraints ∂m_c as well as the list of factor indices I_c in the corresponding node. **(b)** Upward pass at inner iteration t. Each node computes its local vector y_c and passes the residual D and all partials with respect to separator variables to the parent. **(c)** Downward pass at inner iteration t. Each node updates its Δ_c as well as the Lagrange multiplier of all constraints whose index is in I_c using $\Delta_c \cup \Delta_{sep}$. We assign the following dual terms to each constraint: $h_1 \Leftrightarrow u_1, g_1 \Leftrightarrow v_1, g_2 \Leftrightarrow v_2$

4. With the matrix form, there was this limitation that we could not relinearize select variables on the fly. However, unlike columns of a matrix, nodes in the tree are sufficiently decoupled to relinearize subsets of variables as needed. With *Fluid/Online Relinearization*, a variable is marked for relinearization if the deviation of its current estimate from the linearization point is greater than a threshold β. At update time n, a set of variables marked for linearization V_r is first computed. Then a list of *orphan* nodes $\mathcal{N}_o$ is generated where the orphans and their descendants, together, form the subtree $\mathcal{T}_o^{n-1}$ that remains unchanged. The complementary subtree $\mathcal{T}_r^{n-1}$, consists of all nodes $\mathcal{N}_r$ that contain one or more variables in V_r. These nodes are updated to obtain the subtree $\mathcal{T}_r^n$. The original Bayes tree is finally updated as $\mathcal{T}^n = \mathcal{T}_r^n \cup \mathcal{T}_o^{n-1}$. For more details about the update step, we refer the reader to Alg. 6 in [3].

In InCOpt, we use the same relinearization machinery as iSAM2. However, we additionally update the linearization of all hard constraints stored inside nodes $\mathcal{N}_r$. We do this by viewing the Bayes tree at update time n, as a set of nodes $\{\mathcal{N}_c^{n-1}\}$ satisfying:

$$\mathcal{T}^{n-1} = \{\mathcal{N}_c^{n-1} \mid \mathcal{N}_c^{n-1} \in \mathcal{T}_o^{n-1} \mid\mid \mathcal{N}_c^{n-1} \in \mathcal{T}_r^{n-1}\} \quad (7)$$

Now, $\forall\, \mathcal{N}_c^{n-1} \in \mathcal{T}_r^{n-1}$, the associated non-linear factors are relinearized and the QR factorization ($[R_c \mid d_c]$), as well as ∂m_c and I_c, are recomputed. Hence, each node will always contain the latest linearization of the constraints at any time step as determined by the mechanism of fluid relinearization.

D. Bayes Tree Solution Overview

While in iSAM2, updating Δ at time step n involves a single pass from the root of the Bayes tree to its leaves (equivalent to a single back-substitution), InCOpt handles constraints by performing alternating upward and downward passes (equivalent to forward and back-substitutions) to update Δ and the dual variables until a termination condition is satisfied. We next describe one upward and downward passes corresponding to a single inner iteration step t of the algorithm.

E. Tree Solution: Upward Pass

To solve for y in Eq. 4a (referred to as the dual remainder), InCOpt performs an upward pass on the Bayes tree. First, we distribute the blocks of y across the Bayes tree where each node $\mathcal{N}_c$ only stores the block required to compute its clique-Δ_c during the downward pass. Now, forward substitution can be viewed as equivalent to a post order traversal over the Bayes tree (i.e. process all children of a node before processing the node itself) with specific operations performed by each node and messages passed from children to parents.

In essence, each node $\mathcal{N}_c$ updates its local vector y_c as:

$$R_c^T y_c^t = G_c^T v^{t-1} + H_c^T u^{t-1} - e_c^t \quad (8)$$

where R_c is the node clique conditional, e_c^t is a residual vector computed from the product of the children's separator matrices with their local y_{child}^t. G_c^T and H_c^T are computed using both local information stored inside the node (∂m_c) as well as cached gradients passed to the node by its immediate children. v^{t-1}, u^{t-1} are the corresponding dual terms constructed via queries using the per-factor global index stored in I_c. As explained in section IV-C, only a subset of the nodes $\mathcal{N}_r \in \mathcal{T}_r$ are relinearized at any optimizer step. Hence, the upward pass can be significantly sped up by ending the post order traversal at the orphan nodes $\mathcal{N}_o$. See Algorithm 2 for more details and Fig. 4b for an example.

F. Tree Solution: Downward Pass

In InCOpt, the Δ update (Eq. 4b) and dual ascent steps (Eq. 4c and Eq. 4d) are computed by traversing the Bayes tree from root to leaves. Each node $\mathcal{N}_c$ first updates its Δ_c by solving:

$$R_c \Delta_c^t = d_c - y_c^t - S_c \Delta_{sep}^t \quad (9)$$

where Δ_{sep}^t is the state update vector corresponding to each separator variable of $\mathcal{N}_c$. Compared to iSAM2 where each node $\mathcal{N}_c$ updates its local Δ_c by solving:

$$R_c \Delta_c^t = d_c - S_c \Delta_{sep}^t \quad (10)$$

Eq. 9 depends on y_c^t, computed in the upward pass.

In InCOpt, once $\mathcal{N}_c$ updates its Δ_c^t, it then computes the new constraint violation CV_k^t for each constraint $\{m_{c_k} \mid m_{c_k} \in m_c \text{ and } \mathit{Index}(m_{c_k}) = k \in I_c\}$ using the updated Δ_c^t and Δ_{sep}^t, followed by the dual ascent step:

$$\begin{aligned} u_k^t &= \max(0, u_k^{t-1} + \rho_k^{t-1} CV_k^t);\ \textit{if } m_{c_k} \textit{ is an inequality} \\ v_k^t &= v_k^{t-1} + \rho_k^{t-1} CV_k^t;\ \textit{if } m_{c_k} \textit{ is an equality} \end{aligned} \quad (11)$$

Finally, a new penalty term ρ_k^t is computed using the following adaptation scheme: $\rho_k^t = \rho_k^{t-1} \times \frac{1}{r'}$, $r' > 1$ if an *inequality constraint* is satisfied at time t, $\rho_k^t = \rho_k^{t-1} \times r'', r'' > 1$ if the constraint violation decreases by less than a minimum change factor for both equalities and inequalities, and $\rho_k^t = \rho_k^{t-1}$ otherwise. See Algorithm 3 for more details and Fig. IV-F for an example.

G. One step of InCOpt

Algorithm 1 One step of InCOpt

```
Initialization: $\mathcal{T} = \varnothing, \mathcal{F} = \varnothing, \Theta = \varnothing$
Known at step $n-1$: Bayes tree $\mathcal{T}^{n-1}$, non linear factors $\mathcal{F}$, linearization
point $\Theta^{n-1}$, $\Delta$, Lagrange Multipliers $u^{n-1}, v^{n-1}$, penalty terms $\rho^{n-1}$, nIter,
$\epsilon_{primal}$, MaxDeltaThreshold $T$
Input at step $n$: New non linear factors $\mathcal{F}' = m' \cup f'$, New variables $\Theta'$
 1) Add new factors $\mathcal{F} := \mathcal{F} \cup \mathcal{F}'$
 2) Initialize any new variables $\Theta'$ and add $\Theta := \Theta \cup \Theta'$
 3) $\forall$ Factors $F_i \in m'$, initialize $u_i$ or $v_i$ and $\rho_i$
 4) Fluid Relinearization as described in subsection IV-C
 5) Redo top of Bayes tree + update the per-node linearizations of constrained factors
    $\partial m_c$ and the global index list $I_c$ as described in subsections IV-B and IV-C
 6) Solve for $\Delta$:
       While $t <$ maxNumInnerIter
          upwardPass($\mathcal{T}$)
          primalResidual = downwardPass($\mathcal{T}$)
          $\Delta_j^t = \min(\Delta_j^t, T)\ \forall\ j \in \text{Dim}(\Delta^t)$
          if $\left(||\Delta||_\infty^t \geq T \mid\mid primalResidual < \epsilon_{primal}\right)$ break; else $t = t + 1$
 7) Current estimate given by $\Theta \bigoplus \Delta$
```

The InCOpt algorithm is summarized in Algorithm 1. At time step n, a new set of factors $\mathcal{F}$ containing soft (and possibly hard constraint) are added to the non-linear least square problem in Eq. 1. Variables are relinearized using fluid relinearization and the Bayes tree is updated. The new Δ is then computed using the proposed iterative process (an upward pass followed by a downward pass). We cap the values of Δ by a maximum threshold T to ensure that the linearized constraints ∂m remain a good approximation of the non-linear factors m until they are next relinearized. This inner loop is terminated if 1) the primal residual is less than a predefined ϵ (i.e at convergence) or 2) if any value in Δ is greater than T. Finally, the updated solution to Eq. 1 is given by $\Theta \bigoplus \Delta$ with $\bigoplus$ being the retraction operator.

Algorithm 2 Tree Solution - Upward pass at inner iteration t

```
**PostOrderTreeTraversal**(Bayes tree $\mathcal{T}$)
**In:** Bayes tree node $\mathcal{N}_c$, Orphan nodes $\mathcal{N}_o$
*chlds* = $\mathcal{N}_c$*.children*
$e_c^t = \bar{\mathbf{0}}$
For each $\partial m_{c_k} \in \partial m_c$ and each $(M_k^j, m_k(x^0)) \in \partial m_{c_k}$
  $FrontMap[f] = \{(M_k^j, m_k(x^0), k) \mid f \in Front(\mathcal{N}_c), k \in I_c, j = f\}$
  $SepMap[s] = \{(M_k^j, m_k(x^0), k) \mid s \in Sep(\mathcal{N}_c), k \in I_c, j = s\}$
if $\mathcal{N}_c \notin Leaf(\mathcal{T})$
  $FrontMap = FrontMap \cup \{\forall chlds.SepMap[f]$ if $f \in Front(\mathcal{N}_c)\}$
  $SepMap = SepMap \cup \{\forall chlds.SepMap[s]$ if $s \notin Front(\mathcal{N}_c)\}$
  $e_c^t = collectETerm(\mathcal{N}_c)$
$G_c^T, H_c^T, v^{t-1}, u^{t-1} = constructJacobian(FrontMap)$
Solve for $y_c^t$: $R_c^T y_c^t = G_c^T v^{t-1} + H_c^T u^{t-1} - e_c^t$
$\forall s \in Sep(\mathcal{N}_c)$
  $\mathcal{N}_c.D[s] = S_c^T[s] y_c^t + \sum_{chlds} chlds.D[s]$
If $\mathcal{N}_c \in \mathcal{N}_o$ *break;*
```

Note: *Front*($\mathcal{N}_c$) and *Sep*($\mathcal{N}_c$) *return the frontal and separator variables of node* $\mathcal{N}_c$ *respectively, Leaf*($\mathcal{T}$) *are the leaves of the Bayes tree.* $\mathcal{N}_c$*.children return the children of* $\mathcal{N}_c$*. Index* k *in* m_{c_k} *is the* ***global*** *index of the constraint factor* m_k*.* M_k^j *is the partial derivative of* m_k *with respect to variable j evaluated at* x^0*.* $D[s]$ *holds the accumulated residual terms corresponding to variable* s *(term* D *in Fig. 4b) and will be accessed by* $\mathcal{N}_c'$*s parent.* $\bar{\mathbf{0}}$ *is a zero vector*

```
**procedure** COLLECTETERM
**In**: $\mathcal{N}_c$. **Init**: $e_c^t = \bar{\mathbf{0}}$
For each $f \in Frontal(\mathcal{N}_c)$ and each $chld \in \mathcal{N}_c.children$
  $e_c^t.block(f) \mathrel{+}= chld.D[s]$ if $s = f \ \forall s \in chld.D$
return $e_c^t$
**end procedure**
```

Note: e_c^t*.block*(f) *returns a vector view corresponding to frontal variable* f *as determined by the clique node* $\mathcal{N}_c$ *variable ordering.*

```
**procedure** CONSTRUCTJACOBIAN
**In**: *FrontalMap*. **Init**: $G_c^T, H_c^T = \mathbf{0}$, $v^{t-1}, u^{t-1} = \bar{\mathbf{0}}$
For each $f \in FrontalMap$ and each $(M_k^f, k) \in FrontalMap[f]$
  if $m_k$ is an equality constraint
    Set $G_c^T.block(f,k) = M_k^f$ and $v^{t-1}.block(k) = v_k^{t-1}$
  else if $m_k$ is an inequality constraint
    Set $H_c^T.block(f,k) = M_k^f$ and $u^{t-1}.block(k) = u_k^{t-1}$
return $G_c^T, H_c^T, v^{t-1}, u^{t-1}$
**end procedure**
```

Note: u^{t-1}*.block*(k) and v^{t-1}*.block*(k) *return the vector view corresponding to the factor with index k.* G_c^T*.block*(f, k) *and* H_c^T*.block*(f, k) *return a view of the matrices with start row corresponding to frontal variable* f *(as determined by the clique node frontal variable ordering) and start column* k*.* $\mathbf{0}$ *is a zero matrix*

V. Results and Evaluation

We evaluate InCOpt against constrained and unconstrained solver baselines on three different applications. InCOpt is

Algorithm 3 Tree Solution - Downward Pass at inner iteration t

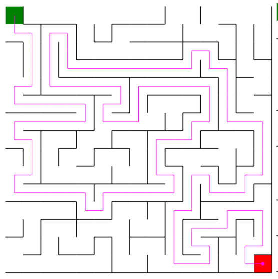
```
**InOrderTreeTraversal**(Bayes tree $\mathcal{T}$)
**In** : $\mathcal{N}_c \in \mathcal{T}, \Delta_{sep}^t$
Solve:
  $R_c \Delta_c^t = d_c - y_c^t - S_c \Delta_{sep}^t$
$\Delta_u^t = \Delta_c^t \cup \Delta_{sep}^t$
For each $\partial m_{c_k} \in \partial m_c$
  Sum $= \bar{\mathbf{0}}$
  For each $(M_k^j, m_k(x^0)) \in \partial m_{c_k}$
    Sum $\mathrel{+}= M_k^j \Delta_u[j]$
  $CV_k^t =$ Sum $+ m_k(x^0)$
  $\rho_k^{t-1} = getPenalty(m_{c_k})$
  If $m_{c_k}$ is an inequality constraint
    $u_k^{t-1} = getDual(m_{c_k})$
    $u_k^t = \max(0,\ u_k^{t-1} + \rho_k^{t-1} CV_k)$
     if $CV_k^t < 0$
       $\rho_k^t = \frac{1}{r'} \rho_k^{t-1}$, return;
  else if $m_{c_k}$ is an equality constraint
    $v_k^{t-1} = getDual(m_{c_k})$
    $v_k^t = v_k^{t-1} + \rho_k^{t-1} CV_k$
  if $|CV_k^t| < \frac{1}{r}|CV_k^{t-1}|$ set $\rho_k^t = \rho_k^{t-1} \times r''$ else $\rho_k^t = \rho_k^{t-1}$
```

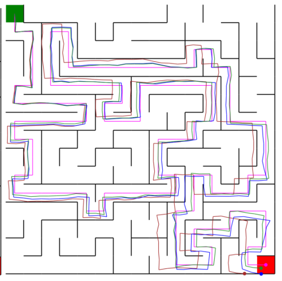
implemented with the GTSAM [21] C++ library.

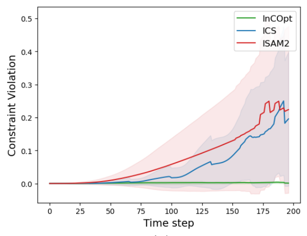
A. 2D Navigation

We evaluate InCOpt against ICS (constrained) and iSAM2 (unconstrained) on a simulated 2D navigation setting where a point robot navigates from a start to an end position (Fig. 5a). ICS is configured to perform a batch relinearization every 100 optimizer steps with the maximum number of inner iterations *maxNumInnerIter* set to 100. Random mazes are generated then solved using a backtracking algorithm to obtain the ground-truth path (GT). We assume a constant scale in which the width of the traversable region in pixels equals 1 meter. Noisy odometry and prior pose measurements are computed by adding a random noise $\sim \mathcal{N}(0, \text{diag}([0.05, 0.05]))$ to the GT. Fig. 5f shows the factor graph used by ICS and InCOpt: f_i are noisy odometry measurements and $h_<^i(x_i), h_>^i(x_i)$ are two inequality constraints that enforce a lower and upper bound on the estimate at x_i ($L_i < x_i < U_i$ i.e. box constraints).

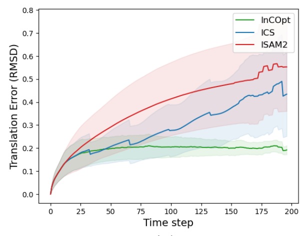
With iSAM2, the hard constraints $h_<^i(x_i), h_>^i(x_i)$ are replaced with soft constraints $f_<^i(x_i), f_>^i(x_i)$ taking the form of a hinge loss ($= 0$ if the constraint is satisfied and > 0 otherwise). In practice, these constraints could reflect a prior


(a)

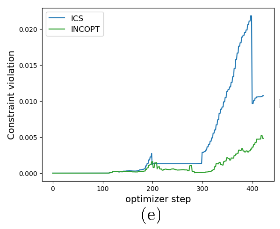

(b)



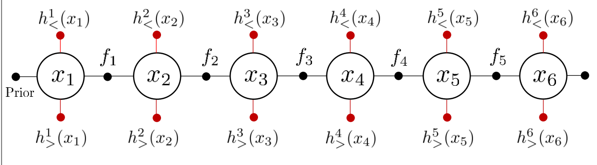
(c)



(d)



(e)



(f)

Fig. 5: **(a)** Example of a generated random maze for a point robot navigating from the green to the red cells **(b)** The path estimated from noisy measurements by the different algorithms. GT: Magenta, Red: iSAM2, Blue: ICS, Green: InCOpt. **(c)** and **(d)** show the std and mean constraint violation and std and mean tracking error (in m) at each time step. Stats were averaged over 100 randomly generated mazes. We see that InCOpt better satisfies the constraints both qualitatively and quantitatively leading to a lower RMSD error **(e)** Constraint satisfaction per optimizer step for a single maze solve. InCOpt constraint violation remain low over all optimizer steps while ICS's constraint violation increases then drops once a periodic batch relinearization occurs ($t = 400$). **(f)** Factor graphs used by ICS and InCOpt. With iSAM2, the hard box constraints $h_<^i$ and $h_>^i$ are replaced by soft constraints $f_<^i$ and $f_>^i$.

knowledge of the map and are a way to inject additional information into the estimation framework: the estimated trajectory of a robot should not run through obstacles.

TABLE I: Averaged metrics over 100 randomly generated mazes. iSAM2 (unconstrained, online relinearization), ICS (constrained, batch relinearization), IncOpt (constrained, online relinearization)

	iSAM2	ICS	InCOpt
Avg RMSD in x (m)	0.254 ±0.118	0.215 ±0.082	**0.147** ±0.029
Avg RMSD in y (m)	0.287 ±0.146	0.209 ±0.080	**0.149** ±0.033
Avg total runtime (s)	0.048	0.062	0.079
Avg num linearizations	69	437	474

We show that incorporating hard constraints leads to better trajectory estimates compared to using soft constraints. In table I, we report the averaged smoothed root mean square deviation (RMSD) between the GT and the estimated trajectory of each algorithm where:

$$\text{RMSD (smoothing)} = \frac{1}{T}\sum_{t=1}^{T}\text{RMSD}(\text{Traj}[t], \text{GT}[t])$$

with Traj$[t]$ and GT$[t]$ being respectively, the estimated and ground-truth trajectories at time t. Fig. 5c shows the average constraint violation and Fig. 5d shows the average RMSD of each algorithm at each time step. All statistics are averaged over 100 randomly generated mazes. From table I, we note that incorporating inequality hard constraints in both ICS and InCOpt leads to better trajectory estimates compared to soft constraints as specified with iSAM2. In addition, InCOpt produces better estimates compared to ICS. This is due to InCOpt leveraging fluid relinearization to relinearize each variable immediately once the deviation of its current estimate from the linearization point is above a threshold.

On the other hand, ICS performs a single batch relinearization only every 100 optimizer step which may cause the linearized objective to become a poor approximation of the original non-linear problem. This is illustrated in Fig. 5e which shows the constraint violation of each algorithm for a single maze solve. ICS's constraint violation continues to increase until a batch relinearization occurs (at $t = 400$) at which point the constraint violation decreases. For InCOpt, the constraint violation remains small across all time steps demonstrating the benefits of performing just-in-time relinearization. Finally, we also report in table I the average total runtime of all algorithms. iSAM2 has the fastest average runtime as it is an unconstrained solver. ICS runtime is slightly lower than InCOpt at the expense of lower accuracy.

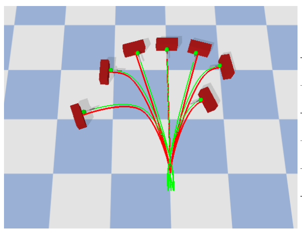
B. *2D Planar Pushing*

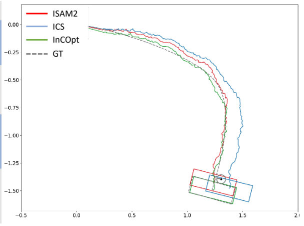
Our second application of interest is 2D planar pushing: a probe is in permanent contact with an object and our objective is to estimate the trajectory of the object using noisy contact measurements. Fig. 6a shows sample ground truth trajectories generated using the PyBullet simulator and Fig. 6g shows the factor graphs used by ICS, InCOpt, and iSAM2 to solve the estimation problem. Noisy odometry measurements f_i are computed by adding a random noise $\sim \mathcal{N}(0, \text{diag}([7.5e\text{-}3, 7.5e\text{-}3, 5e\text{-}3]))$ to the GT. We add a unary *Contact Factor* at each pose (see [22]) which minimizes the distance between the probe and the object to ensure that they are in constant contact at the boundary.

In ICS and InCOpt, the contact factor is a hard equality

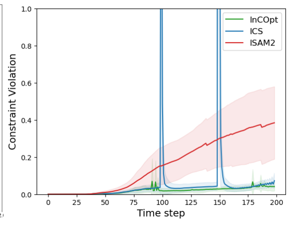
TABLE II: Metrics for 7 generated planar pushing trajectories. InCOpt is able to achieve the lowest RMSD over most trajectories while having a comparable runtime. Runtime is correlated with number of online relinearizations. InCOpt performs more relinearizatons compared to ICS but achieves higher accuracy.

	Traj 1			Traj 2			Traj 3			Traj 4			Traj 5			Traj 6			Traj 7		
	iSAM2	ICS	*InCOpt*	iSAM2	ICS	*InCOpt*	iSAM2	ICS	*InCOpt*	iSAM2	ICS	*InCOpt*	iSAM2	ICS	*InCOpt*	iSAM2	ICS	*InCOpt*	iSAM2	ICS	*InCOpt*
RMSD in x (m)	0.073	0.083	0.026	0.053	0.035	0.014	0.097	0.018	0.012	0.061	0.034	0.016	0.063	0.011	0.010	0.040	0.019	0.010	0.039	0.017	0.018
RMSD in y (m)	0.069	0.133	0.042	0.121	0.061	0.030	0.027	0.027	0.026	0.030	0.031	0.029	0.0364	0.0349	0.0348	0.079	0.035	0.021	0.067	0.172	0.033
RMSD in θ (rad)	0.029	0.045	0.012	0.087	0.065	0.050	0.025	0.023	0.024	0.0183	0.0182	0.0180	0.027	0.024	0.026	0.062	0.047	0.041	0.0230	0.0505	0.0231
Constraint violation	0.222	0.097	0.090	0.416	0.060	0.030	0.411	0.050	0.028	0.397	0.156	0.044	0.890	0.024	0.021	0.263	0.054	0.023	0.176	0.068	0.053
Runtime (s)	0.253	0.268	0.293	0.244	0.260	0.431	0.251	0.266	0.302	0.252	0.265	0.265	0.240	0.260	0.290	0.245	0.261	0.433	0.252	0.267	0.272
Num linearization	263	303	353	266	303	862	242	303	380	262	303	263	248	303	365	259	303	819	266	303	265

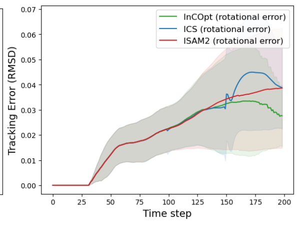

(a) Sample ground truth trajectories generated using the PyBullet Simulator



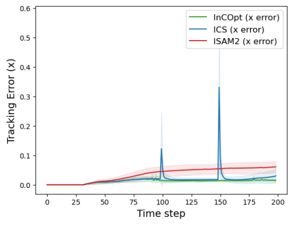
(b) Path estimates



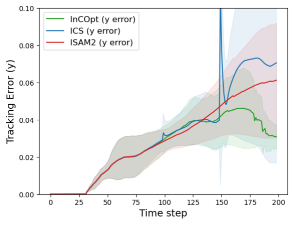
(c) Constraint violation



(d) Tracking rotational error

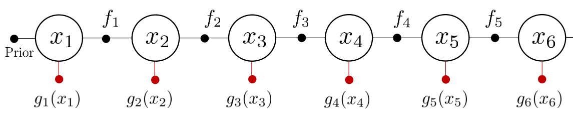


(e) Tracking translational error (x)



(f) Tracking translational error (y)

value of the metric at each time step over 7 trajectories



(g) Factor graph for planar pushing

Fig. 6: **(a)** Example of sample trajectories generated using PyBullet. **(b)** The path estimated from noisy measurements by the different algorithms. GT: gray, Red: iSAM2, Blue: ICS, Green: InCOpt. **(c)** shows the std and mean constraint violation. **(d)** shows the rotational error (in rad). **(e)**, **(f)** show the translational errors in x,y respectively (in m). All plots show the averaged value of each metric at each time step over 7 trajectories. **(g)** Factor graphs used by ICS and InCOpt. With iSAM2, the hard contact constraints g_i are replaced by soft constraints f_i

constraint: the probe has to be in contact with the object at all times. In iSAM2, it is encoded as a soft constraint. Table II, reports the RMSD of the GT and estimate of each algorithm for seven sampled trajectories. Figs. 6c,d,e, and f show the averaged per-step constraint violation, rotational and translational (x, y) errors over the seven trajectories respectively. We see that InCOpt's constraint violation remains small over all time steps compared to ICS and iSAM2 leading to more accurate estimates and a lower RMSD. We note that the largest accuracy gains with the constrained factors occurs in the position estimates and less so in orientation. This makes sense since enforcing equality constraints at the contact point still retains some ambiguity in orientation estimates about a single point of contact.

C. 3D Manipulation Planning

InCOpt's solutions are invariant to the noise models specified for hard constraints. GPMP2 [13] is a motion planner that frames the planning problem as probabilistic inference over factor graphs. However, it requires manual tuning of various noise terms/cost weights which constitutes one of its bottlenecks. InCOpt alleviates this tuning process and ensures satisfaction of safety constraints independently of the specified noise model weights. We start with a summary of the original GPMP2 algorithm and then present a new formulation that involves hard constraints. GPMP2 defines support states x_t which are N dimensional vectors containing the target angle at time t, θ_t^j, for each joint j (N being the number of Degrees of Freedom (DOF) of the arm). The *Gaussian Process* factors f_i^{gp} encourage smoothness between consecutive support states. P_{sc} and P_{es} are priors on the first and last support state to ensure that the arm starts and ends at the target configurations. The factors f_i^{Ob} and f_i^{ObIn} are the *regular* and *interpolated* obstacle costs that try to maximize the arm's distance to the environment's obstacles. Specifically, the robot arm is represented by a set of spheres $S_j,\ j \in [1 \dots J]$ as shown in Fig. 7a. The obstacle cost f_i^{Ob} (and similarly for f_i^{ObIn}) is then defined as:

$$f_i^{Ob} = \sum_{j=1}^{J} \mathbf{c}[d(s_j)] \text{ s.t. } \mathbf{c}(z) = \begin{cases} -d(z) + \epsilon & \text{if } d(z) \leq \epsilon \\ 0 & \text{if } d(z) > 0 \end{cases}$$

where $d(s_j)$ is the distance from a sphere s_j to the nearest obstacle as encoded by a signed distance field and ϵ is a safety distance. We define the following metrics used to quantify the quality of an optimized trajectory:

$$\text{smoothness} = \frac{1}{\frac{1}{\text{DOF}} \sum_{t=1}^{S} \sum_{j=1}^{\text{DOF}} ||Wrap(\theta_t^j - \theta_{t+1}^j)||_2^2} \quad (12)$$

$$\text{obstacle cost} = \frac{1}{2} \sum_{n=1}^{N_{ObIn}} ||f_n^{ObIn}||_2^2 + \frac{1}{2} \sum_{n=1}^{N_{Ob}} ||f_n^{Ob}||_2^2 \quad (13)$$

where DOF $= 7$ for the simulated Barrett WAM, S is the number of support states, N_{Ob} and N_{ObIn} are respectively, the number of regular and interpolated obstacle factors, and the *Wrap* function wraps angles to $[-\pi, \pi]$. In GPMP2, the smoothness and obstacle costs need to be balanced out by manually tuning their associated noise parameters σ_{gp} and σ_{ob}. Figs. 7b and 7c illustrate this trade-off: when using GPMP2 with the unconstrained solvers Levenberg-Marquardt (LM), Dogleg, or iSAM2, the optimized trajectory has a low obstacle cost for small σ_{ob} but unsmooth transitions between support states. For a large σ_{ob}, the trajectory is smooth at the expense of a high obstacle collision cost.

With GPMP2+InCOpt, we replace f_i^{ObIn} and f_i^{Ob} with the constrained factor h_i^{ObIn} and h_i^{Ob}: these are hard inequality constraints that force the obstacle cost of the resulting trajectory to go to zero (Fig. 7d shows the updated GPMP2 factor graph). From Figs. 7b and 7c, we see that GPMP2+InCOpt generates a trajectory that minimizes the obstacle cost and also remains smooth $\forall\ \sigma_{ob}$: InCOpt first pushes the trajectory to a low obstacle cost region. Once the obstacle cost inequality constraints are satisfied, we can view the GPMP2 factor graph as only containing the Gaussian Process factors f_i^{gp} and priors P_{sc}, P_{ec} which InCOpt, then, attempts to satisfy.

D. Analysis on Runtime and Number of relinearizations

We show that InCOpt performs fewer relinearizations and has better runtime compared to ICS while achieving similar accuracy. We use 2D navigation (inequality constraints) and planar pushing (equality constraints) for this analysis. Figs. 8a and 8b show the number of linearized variables and time per optimizer step for planar pushing and Figs. 8c and 8d show the same quantities for 2D navigation. These plots are averaged over 10 different runs of the same maze/planar pushing trajectory with different random seeds used for noise generation. For ICS, we see periodic peaks corresponding to the specified batch relinearize step. For InCOpt, the number of relinearized variables is strictly determined by the number of variables whose Δ is above the relinearization threshold

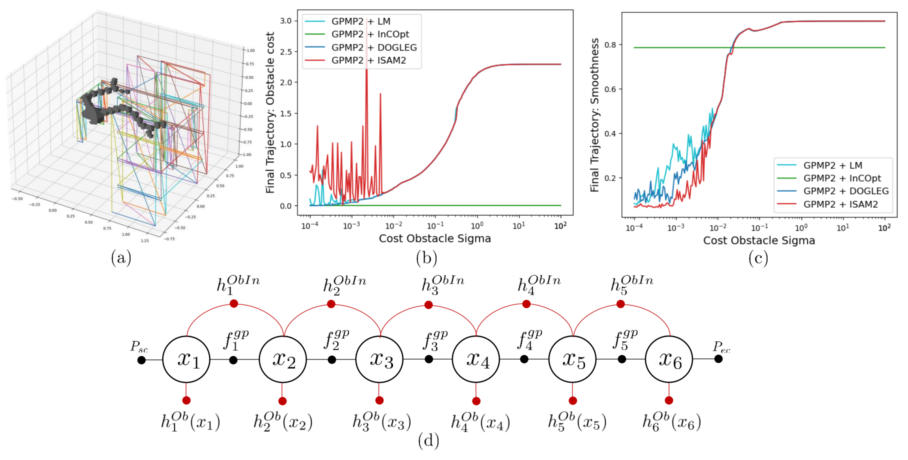


Fig. 7 **(a)** Example of a 3D planning scenario. A simulated Barrett WAM need to plan a collision-free trajectory from points A to B. **(b)** Obstacle cost of the planned trajectories **(c)** smoothness of the planned trajectories. Both plots generated with $\sigma_{gp} = 1$ and 200 different values of σ_{ob}. **(d)** Constrained factor graph for the 3D manipulation planning problem (GPMP2) as used by InCOpt. The original GPMP2 uses unconstrained factors f_i^{Ob} and f_i^{ObIn} instead of h_i^{Ob} and h_i^{ObIn} .

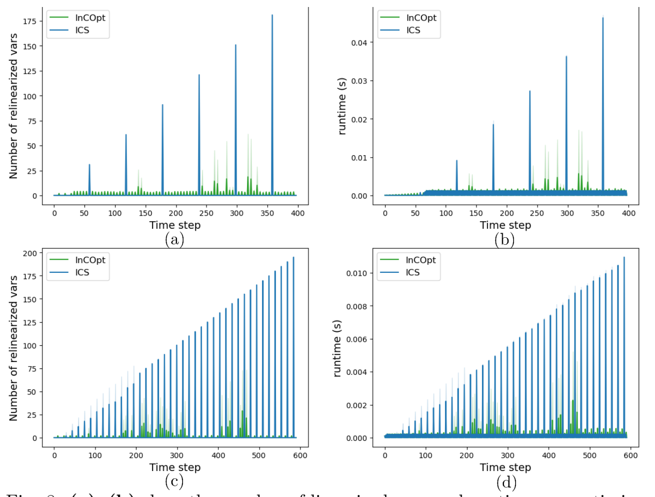


Fig. 8: **(a)**, **(b)** show the number of linearized vars and runtime per optimizer step for planar pushing. **(c)** and **(d)** show the same metrics for 2D navigation. InCOpt performs less relinearizations and has a faster runtime compared to ICS to achieve the same accuracy levels.

at any time step (i.e. on-time relinearizations). For both algorithms, spikes in runtime occur at the time steps where relinearizations are performed. As seen in Table III, InCOpt performs less relinearizations compared to ICS leading to a lower overall runtime for the same accuracy levels.

TABLE III: InCOpt vs ICS: number of relinearization/runtime to achieve the same accuracy levels.

Planar Pushing	Avg Num relinearization	Average runtime (s)	Translation error (m)	Rotational error (rad)
ICS	636	0.347	0.048	0.032
InCOpt	356	0.294	0.048	0.033
2D Navigation	Avg Num relinearization	Average runtime (s)	Translation error (m)	
ICS	2603	0.184	0.202	
InCOpt	393	0.076	0.209	

VI. Conclusion and Future Work

We proposed InCOpt, a primal-dual constrained optimization solver formulated as message passing on the Bayes tree. We evaluated InCOpt on online estimation problems such as 2D navigation and planar pushing and demonstrated how leveraging hard constraints can lead to increased accuracy without a significant rise in runtime. We also tested our solver on 3D manipulation planning examples and showed how enforcing hard constraints on GPMP2's obstacle cost terms ensures satisfaction of safety constraints independently of the cost weights.

We list a few limitations of our current approach that we plan to address in future work. First, our current formulation of the Augmented Gaussian Factor Graph may result in an under-determined system if an inequality constraint is satisfied and inactivated. Currently, our solution is to specify additional factors as regularizers. However, a more general approach would be to automatically detect such cases and regularize the system when required. Second, a fundamental assumption when solving a non-linear constrained objective (Eq. 1) by solving successive linear subproblems (Eq. 2) is that the linearized objective remains a good approximation of the non-linear problem throughout the inner loop. This is currently ensured by manually selecting the max Δ threshold T. Correctly setting T was especially important for high-dimensional problems, such as 3D arm planning, where Δ encodes changes in arm joint angles. Another implication is that the optimized trajectory may lie at a local minimum where the inequality constraints are satisfied (obstacle cost is minimized) but the arm is not exactly starting and ending at the pre-specified configurations. We selected different thresholds T according to the sensitivity of the obstacle cost to a change in each joint angle. Hence, another direction for future work is to do away with manually setting the threshold T by devising a trust-region augmented Lagrangian method.

References

[1] P. Sodhi, S. Choudhury, J. G. Mangelson, and M. Kaess, "ICS: Incremental constrained smoothing for state estimation," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2020.

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[3] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the Bayes tree," *Intl. J. of Robotics Research*, vol. 31, no. 2, pp. 216–235, Feb. 2012.

[4] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Trans. Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[5] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[6] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[7] R. M. Eustice, H. Singh, and J. J. Leonard, "Exactly sparse delayed-state filters for view-based slam," *IEEE Trans. Robotics*, vol. 22, no. 6, pp. 1100–1114, 2006.

[8] F. Dellaert and M. Kaess, "Square root SAM: Simultaneous localization and mapping via square root information smoothing," *Intl. J. of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, 2006.

[9] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Incremental smoothing and mapping," *IEEE Trans. Robotics*, vol. 24, no. 6, pp. 1365–1378, 2008.

[10] M. Kaess, V. Ila, R. Roberts, and F. Dellaert, "The bayes tree: An algorithmic foundation for probabilistic robot mapping," in *Algorithmic Foundations of Robotics IX*. Springer, 2010, pp. 157–173.

[11] J. Dong, M. Mukadam, F. Dellaert, and B. Boots, "Motion planning as probabilistic inference using Gaussian processes and factor graphs." in *Robotics: Science and Systems (RSS)*, vol. 12, 2016, p. 4.

[12] F. Dellaert, "Factor graphs: Exploiting structure in robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, pp. 141–166, 2021.

[13] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time gaussian process motion planning via probabilistic inference," *The International Journal of Robotics Research*, vol. 37, no. 11, pp. 1319–1340, 2018.

[14] S. Yang, G. Chen, Y. Zhang, H. Choset, and F. Dellaert, "Equality constrained linear optimal control with factor graphs," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2021, pp. 9717–9723.

[15] M. Xie, A. Escontrela, and F. Dellaert, "A factor-graph approach for optimization problems with dynamics constraints," *arXiv preprint arXiv:2011.06194*, 2020.

[16] D.-N. Ta, M. Kobilarov, and F. Dellaert, "A factor graph approach to estimation and model predictive control on unmanned aerial vehicles," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2014, pp. 181–188.

[17] A. Cunningham, M. Paluri, and F. Dellaert, "DDF-SAM: Fully distributed SLAM using constrained factor graphs," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2010, pp. 3025–3030.

[18] I. D. D. Jimenez Rodriguez, "A factor graph approach to constrained optimization," 2017.

[19] S. Choudhary, L. Carlone, H. I. Christensen, and F. Dellaert, "Exactly sparse memory efficient SLAM using the multi-block alternating direction method of multipliers," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2015, pp. 1349–1356.

[20] F. Bai, T. Vidal-Calleja, and S. Huang, "Robust incremental SLAM under constrained optimization formulation," *IEEE Robotics and Automation Letters*, no. 2, pp. 1207–1214, 2018.

[21] F. Dellaert, "Factor graphs and GTSAM: A hands-on introduction," Georgia Institute of Technology, Tech. Rep., 2012.

[22] K.-T. Yu and A. Rodriguez, "Realtime state estimation with tactile and visual sensing. application to planar manipulation," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2018, pp. 7778–7785.