Handling Constrained Optimization in Factor Graphs for Autonomous Navigation

Barbara Bazzana, Graduate Student Member, IEEE, Tiziano Guadagnino, Member, IEEE, and Giorgio Grisetti, Member, IEEE

Abstract—Factor graphs are graphical models used to represent a wide variety of problems across robotics, such as Structure from Motion (SfM), Simultaneous Localization and Mapping (SLAM) and calibration. Typically, at their core, they have an optimization problem whose terms only depend on a small subset of variables. Factor graph solvers exploit the locality of problems to drastically reduce the computational time of the Iterative Least-Squares (ILS) methodology. Although extremely powerful, their application is usually limited to unconstrained problems. In this letter, we model constraints over variables within factor graphs by introducing a factor graph version of the Augmented Lagrangian (AL) method. We show the potential of our method by presenting a full navigation stack based on factor graphs. Differently from standard navigation stacks, we can model both optimal control for local planning and localization with factor graphs, and solve the two problems using the standard ILS methodology. We validate our approach in real-world autonomous navigation scenarios, comparing it with the de facto standard navigation stack implemented in ROS. Comparative experiments show that for the application at hand our system outperforms the standard nonlinear programming solver Interior-Point Optimizer ( IPOPT) in runtime, while achieving similar solutions.

Index Terms—Localization, Integrated Planning and Control, Optimization and Optimal Control.

I. INTRODUCTION

Factor graphs are a general graphical formalism to model several problems. The robotics community extensively used factor graphs to approach estimation problems, such as SfM, SLAM and calibration [8], [17], [18], [27]. On the one hand, they provide a graphical representation that exposes the general structure of the problem which, if exploited, allows the design of efficient algorithms. On the other hand, factor graphs allow to naturally couple problems sharing common variables by simply joining them. The solution to the joined factor graph will be then the optimum of the coupled problem. Well-known methodologies to solve large factor graphs use variants of Iterative Least Squares (ILS) [16], [19], [20] that do not support constrained optimization in its original formulation.

Fig. 1. Left: a trajectory executed by our real robot avoiding an unforeseen obstacle, a schematic factor graph is superposed to it where some robot states are represented (pink circles). Both the localization problem (orange square) and the optimal control are depicted. States are linked by the motion model (yellow squares) and subject to obstacle avoidance and trajectory tracking (green squares). Right: our custom made unicycle.

Albeit mapping and control are coupled in the reality, they are usually addressed separately due to their complexity. However, extending existing ILS solvers to constrained optimization provides a common framework where both the above-mentioned properties of factor graphs can be exploited. In particular, the two problems could consistently share common information such as system dynamics, external disturbances, and uncertainty models. Further, the sparsity pattern of both problems allows designing efficient algorithms to maintain high computational performance in joint optimization.

Leveraging on the results of [29], the contribution of this paper is an Augmented Lagrangian-version of factor graph optimization. It was applied to the development of a factor graph-based Model Predictive Control (MPC) framework for unicycle navigation. The optimal control problem we consider is that of generating a collision-free trajectory for a differential drive robot with actuation limits in a dynamic environment. To this extent, we introduce constraint factors and obstacle avoidance factors: the latter ones model the minimization of an artificial potential function [3], [28]. Our approach was validated both in simulation and on a custom-made robot, where all software runs on a Raspberry Pi4. Fig. 1 shows a real-world trajectory and a simplistic representation of the related factor graph. In this paper, we focus on the development of the navigation module, assuming that the map is given. Localization is also addressed in the framework of ILS on factor graphs, in particular the likelihood of the current estimate is evaluated through the distance between the laser scan and the map.

We conducted comparative experiments with ROS navigation stack, comprising amcl for localization and move_base with
teb_local_planner\(^2\) for planning. Experiments show that (i) our approach gives on-pair and repeatable results (ii) the CPU consumption of our approach is lower. Further, comparative experiments with the standard nonlinear programming solver IPOPT show that (iii) when approaching constrained optimization problems on the models we analyzed, our generic solver outperforms IPOPT in runtime while producing comparable trajectories and final cost value. This gives a glimpse of the computational performances of factor graphs motivating their use also for very large problems.

II. RELATED WORK

Factor graphs are a very powerful tool to model a great variety of unconstrained optimization problems, spacing from SLAM to SfM \cite{18, 27}. In the recent literature, they have been applied to model optimal control problems as well \cite{14, 21, 30, 31, 32}. To this extent, extensions of factor graph solvers have been proposed to handle constraints arising from optimal control modeling. Formalizing optimal control problems using factor graphs allows unifying the dual problems of estimation and control under the same representation. Thanks to this unification, common information and variables can be shared consistently in both estimation and control processes. Further, these two aspects can be addressed jointly taking advantage of efficient factor graph solvers \cite{16, 19, 20}.

To the best of our knowledge, Mukadam et al. \cite{21} were the first to optimize a unique factor graph for both trajectory estimation and motion planning, where trajectories are modeled using continuous Gaussian Processes \cite{13}. However, they were only relying on soft constraints. Later on, the problem of addressing constrained optimization received increasing attention, starting from Ta et al. \cite{30}. They present a factor graph version of Sequential Quadratic Programming (SQP) to handle nonlinear equality constraints and apply their approach to the development of an MPC framework on unmanned aerial vehicles. Our work can be viewed as an extension of this method, where we model both equality and inequality constraints using factor graphs. In place of SQP, we rely on the AL method because, differently from SQP, no quadratic programming solver is required for the internal iterations when addressing inequalities.

Yang et al. \cite{32} include control inputs in a factor graph to solve a Linear Quadratic Regulator problem subject to auxiliary equality constraints. During the variable elimination process, when constrained variables are eliminated, a specialized solver is used for solving the constrained sub-problem separately. Differently from them, our method can handle both equality and inequality constraints and does not require a specialized solver, as it uses a standard ILS algorithm.

To the best of our knowledge, only Xie et al. \cite{31} introduce both equality and inequality constraints in factor graphs. In particular, they present a factor graph version of a barrier-based approach to constrained optimization similar to the well-known Interior-Point Method. Our work can be viewed as complementary to Xie et al., in that we present an implementation of the AL method, as an alternative to the Interior-Point Method.

Sodhi et al. \cite{29} proposes an approach to refine the estimate of past trajectories: to accomplish collision-free trajectories, states are subject to inequality constraints. To the best of our knowledge they have been the first to introduce the AL method in the context of incremental smoothing. We rely on the same concepts to develop a factor-graph version of this method. We propose a different application than Sodhi et al: instead of state estimation, we address optimal control problems to develop a factor graph-based MPC controller that can be used as an elegant and compact local planner. In addition, we address the problem of obstacle avoidance by designing a factor that relies on a distance cost function inspired by scan registration techniques. The resulting factor graph is solved by our generic factor graph solver \cite{16}.

III. FACTOR GRAPH OPTIMIZATION

In this paper, we model both localization and MPC using factor graphs, which are bipartite graphs with two kinds of nodes: variables and factors. Variables represent the state of our system, while factor nodes model measurements connecting the set of variables from which they depend.

More formally, let \( x = x_{0:N} \) be the set of all variables which can span over arbitrary continuous domains, with \( x_k \in \mathbb{R}^n \). If the measurements \( z = z_{0:N} \) are affected by Gaussian noise, we can represent the \( k \text{-th} \) factor as the tuple \( \langle z_k, \Omega_k, h_k(\cdot) \rangle \) comprising the mean \( z_k \), the information matrix \( \Omega_k \), and the prediction function \( h_k(x_k) \), depending on a subset of variables \( x_k = x_{k_0:k_k} \). In this setting the negative log-likelihood of the factor graph becomes:

\[
F(x) = \sum_{k=0}^{K} ||h_k(x^k) - z_k||^2_{\Omega_k} = \sum_{k=0}^{K} ||e_k(x^k)||^2_{\Omega_k} \tag{1}
\]

Solving a factor graph means finding the \( x \) which minimizes (1), i.e. the \( x \) which is maximally consistent with the measurements. We refer the reader to \cite{11, 16} for more details on formulating robotics problems with factor graphs.

ILS minimizes (1) by using Gauss-Newton (GN) algorithm that iteratively refines the current solution \( \hat{x} \) by solving the quadratic approximation of (1). The first-order Taylor expansion of the errors \( e_k(x^k) \) around \( \hat{x} \) given the perturbation \( \Delta x^k \) is computed as:

\[
e_k(\hat{x}^k + \Delta x^k) \simeq e_k(\hat{x}^k) + J_k^T \Delta x^k \tag{2}
\]

By substituting (2) in (1), we obtain a quadratic form that approximates the cost function around \( \hat{x} \):

\[
F(\hat{x} + \Delta x) \simeq c + 2b^T \Delta x + \Delta x^T H \Delta x \tag{3}
\]

where

\[
b = \sum_{k=0}^{K} J_k^T \Omega_k e_k \quad H = \sum_{k=0}^{K} J_k^T \Omega_k J_k \tag{4}
\]
Algorithm 1: Augmented Lagrangian-Iterative Least Squares.

\begin{algorithm}
\hspace{1cm} F total number of factors
\hspace{1cm} 〈 x_k, \Omega_k, b_k (·) 〉 error factor
\hspace{1cm} 〈 \lambda_k, \rho_k, c_k (·) 〉 constraint factor
\hspace{1cm} while I converged do
\hspace{1cm}     H ← 0_{n×n}
\hspace{1cm}     b ← 0_{n×1}
\hspace{1cm}     for k = 0, …, F do
\hspace{1cm}         if error factor then
\hspace{1cm}             b += J_k^T \Omega_k \hat{e}_k
\hspace{1cm}             H += J_k^T \Omega_k J_k
\hspace{1cm}         else if constraint factor then
\hspace{1cm}             b += \frac{\rho_k}{2} C_k^T \hat{e}_k + \frac{1}{2} C_k^T \lambda_k
\hspace{1cm}             H += \frac{\rho_k}{2} C_k^T C_k
\hspace{1cm}         end if
\hspace{1cm}     end for
\hspace{1cm}     \Delta x ← solve(H\Delta x = -b)
\hspace{1cm}     \hat{x} ← \hat{x} + \Delta x
\hspace{1cm}     for k = 0, …, F do
\hspace{1cm}         if constraint factor then
\hspace{1cm}             update \lambda_k according to (8) or (10)
\hspace{1cm}         end if
\hspace{1cm}     end for
\end{algorithm}

The minimum \( \Delta x \) of the quadratic form of (3) is the solution of the linear system \( H \Delta x = -b \), while the next estimate is updated by applying the perturbation \( \hat{x} ← \hat{x} + \Delta x \).

IV. EXTENDING ITERATIVE LEAST SQUARES SOLVERS FOR CONstrained OPTIMIZATION

Lagrange Multipliers [2] is perhaps the most known method to deal with constrained optimization. Although the method tackles nonlinear constraint functions as well, we will focus on the linear case to give a simple overview. Consider the problem:

\[
F(x) = \min_x \sum_{k=0}^{K} \left\| e_k(x_k) \right\|_{\Omega_k}^2 \quad \text{s.t.} \quad Cx + c = 0
\]

where \( C \in \mathbb{R}^{c×n} \). The augmented Lagrangian of (5) is defined as:

\[
\mathcal{L}(x, \lambda^c; \rho^c) = F(x) + (\lambda^c)^T (Cx + c) + \frac{\rho^c}{2} \left\| Cx + c \right\|_2^2.
\]

Here, \( \lambda^c \in \mathbb{R}^c \) is a vector of Lagrange Multipliers and the term \( \frac{\rho^c}{2} \left\| \cdot \right\|_2^2 \) is a penalty term. It can be shown [4] that the minimum of (5) can be found by iteratively computing:

\[
x ← \arg\min_x \mathcal{L}(x, \lambda^c; \rho^c)
\]

\[
\lambda^c ← \lambda^c + \rho^c (Cx + c).
\]

This method can be generalized to deal also with inequality constraints \( Dx + d ≤ 0 \) with \( D \in \mathbb{R}^{d×n} \) by adding an additional multiplier \( \lambda^d \in \mathbb{R}^d \) and an additional penalty term weighted by \( \rho^d \in \mathbb{R} \). The new augmented Lagrangian then becomes:

\[
\mathcal{L}(x, \lambda^c, \lambda^d; \rho^c, \rho^d) = F(x) + (\lambda^c)^T (Cx + c) + \frac{\rho^c}{2} \left\| Cx + c \right\|_2^2 + (\lambda^d)^T (Dx + d) + \frac{\rho^d}{2} \left\| Dx + d \right\|_2^2.
\]

The current state estimate is the \( x \) which minimizes (9). Further, the update-law of the inequality multiplier \( \lambda^d \) is the following:

\[
\lambda^d ← \max(0, \lambda^d + \rho^d (Dx + d))
\]

The AL method belongs to the class of primal-dual methods which split the optimization in two steps: a primal step (7) and a dual step (8, 10). Implementing the dual update requires extending the constrained variables with the multipliers. An iteration proceeds by updating the constrained variable \( x \) while keeping the multipliers fixed through the primal step; once \( x \) is refined, the multipliers \( \lambda^c \) and \( \lambda^d \) are updated with \( x \) fixed through the dual step.

The primal step (7) is a minimization that can be solved through ILS since the multipliers are fixed. Let \( \hat{x} \) be the current solution, \( \hat{c} = Cx + c \) and \( \hat{d} = Dx + d \) be the constraint values computed at \( \hat{x} \). During the linearization step of ILS, the quadratic form approximating (9) around \( \hat{x} \) then becomes:

\[
\mathcal{L}(\hat{x} + \Delta x, \lambda^c, \lambda^d; \rho^c, \rho^d) ≈ c + 2b^T \Delta x + \Delta x^T H \Delta x
\]

where

\[
b = \sum_{k=0}^{K} J_k^T \Omega_k \hat{e}_k + \frac{\rho^c}{2} C^T \hat{c} + \frac{\rho^d}{2} D^T \hat{d}
\]

\[
H = \sum_{k=0}^{K} J_k^T \Omega_k J_k + \frac{\rho^c}{2} C^T C + \frac{\rho^d}{2} D^T D.
\]

By comparing (12) and (4), it turns out that the primal update can be implemented in ILS solvers by introducing constraint factors. They are represented by tuples as \( 〈 \lambda_k, \rho_k, c_k (·) 〉 \), and contribute to the \( H \) and \( b \) matrices respectively through \( H^c \) and \( b^c \), or \( H^d \) and \( b^d \) depending on the type of constraint. The proposed approach is summarized in Algorithm 1. In our implementation, we use constant \( \rho_k = 1 \) but an adaptive scheme as in [29] can be introduced. Our derivation of constraint factors is made possible by the Lagrangian being a simple extension of the usual cost function. As a result, the primal update is similar to the usual update of ILS, which then triggers the update of the multipliers through the dual update.

V. PROBLEM DESCRIPTION

Our factor-graph based navigation framework consists of two modules: MPC controller for local planning Section V-A and scan-matched based localization Section V-B. Both problems are modeled through factor graphs and tackled by the generic ILS solver [16]. This confirms the generality of our approach.
A. MPC with Obstacle Avoidance for Unicycle

MPC is a well-established technique for optimally controlling systems subject to constraints. At each time step, an MPC controller computes the best feasible trajectory which minimizes the assigned objective function, while avoiding obstacles and satisfying the constraints. Only the first control input is applied to the system; at the next time step, a new trajectory embedding information coming from the most recent measurements is computed. More in detail, let \( \mathbf{u} = u_{0:N-1} \) be the sequence of controls, \( \mathbf{x} = x_{0:N} \) the resulting chain of states, \( \mathbf{u}^{\text{ref}} = u_{0:N-1}^{\text{ref}} \) and \( \mathbf{x}^{\text{ref}} = x_{0:N}^{\text{ref}} \) the reference values for controls and states respectively, a suitable objective is the Quadratic Regulator (QR) cost function \( \Phi_{QR}(\mathbf{u}, \mathbf{x}) \) [1]

\[
\Phi_{QR}(\mathbf{u}, \mathbf{x}) = \sum_{n=0}^{N} \left\| \mathbf{x}_n - \mathbf{x}_n^{\text{ref}} \right\|^2 \Omega_n^{\text{r}} + \sum_{n=0}^{N-1} \left\| \mathbf{u}_n - \mathbf{u}_n^{\text{ref}} \right\|^2 \Omega_n^{\text{u}}
\]

subject to the robot kinematics and the actuation limits. The QR cost function in (13) is an instance of the standard Least Squares problem in (1), which is addressed by general factor graph solvers, when the control chain becomes part of the state. As the task of MPC is to compute both the trajectory and the sequence of controls, in the language of factor graphs controls will become variable nodes as well. Hence, each term in the above equation can be modeled as the single-variable factor \( (\mathbf{x}_n^{\text{ref}}, \Omega_n^{x}, \mathbf{I}) \), with \( \mathbf{I} \) identity matrix. In contrast, obstacle avoidance and constrained optimization require more effort to be translated into the same formalism. Fig. 2 illustrates the factor graph formulation of an optimal control problem.

A feasible trajectory can be found by minimizing a cost function \( g(\mathbf{x}) \) that decreases with the distance from obstacles, being zero at safe locations. According to the theory of Artificial Potential Fields [3], [28], having denoted as \( k \) a positive real scalar, as \( \mu \) a low-distance threshold, and as \( \rho \) a high-distance threshold, we designed \( g(\mathbf{x}) \) as:

\[
g(\mathbf{x}) = \begin{cases} 
    k \left( \frac{1}{\mu} - \frac{1}{\rho} \right) & \text{if } d(\mathbf{x}) < \mu \\
    k \left( \frac{1}{d(\mathbf{x})} - \frac{1}{\rho} \right) & \text{if } \mu < d(\mathbf{x}) < \rho \\
    0 & \text{otherwise}
\end{cases}
\]

(14)

The above equation imposes a finite maximum value to \( g(\mathbf{x}) \) by clamping it to its maximum where \( d(\mathbf{x}) < \mu \), and shapes a valley of zero-potential where \( d(\mathbf{x}) > \rho \). Points in the region \( d(\mathbf{x}) > \rho \) are safe and do not need to be pushed away by the optimization process. In our implementation of (14) we use \( k = 0.075, \rho = 0.8 \text{ m}, \mu = 0.05 \text{ m}. \) Therefore we model the obstacle avoidance through a factor whose error is:

\[
e_{\text{o}}^{n}(x_{n}) = g(x_{n}).
\]

(15)

It is well known that Artificial Potential Fields are affected by local minima. In our implementation, we have modified the gradient of the repulsive field generated by (14) to overcome such minima. Using the idea of vortex fields [10], we add to the repulsive gradient a vector which is tangent to the equipotential contours of the repulsive field, as it can be seen in Fig. 3. By applying the obstacle avoidance factor to each robot state, the vortex field acts on the whole trajectory rather than on a single state [10], to deflect it from unexpected obstacles.

As suggested by Rösmann et al. [26], the kinematic model \( f(\cdot) \) mapping current state \( x_n \) and control \( u_n \) to next state \( x_{n+1} \) is incorporated into the objective function as a quadratic penalty for faster convergence of the algorithm. Such a penalty can be modeled by a factor with error function \( e_{\text{o}}^{n}(x_{n}, u_{n}, x_{n+1}) = x_{n+1} - f(x_{n}, u_{n}) \). The weighting matrix \( \Omega^{n}_{\text{o}} \) was tuned by trial and error to preserve the correctness of the result despite such approximation of the constraint. In an MPC fashion, at each time step a graph is constructed starting from the current localization estimate which becomes the first (fixed) variable \( x_0 \). Our robot is a unicycle, controlled in translational and rotational velocities, denoted respectively with \( v_n \) and \( \omega_n \). Using Runge-Kutta integration [5] with \( T_s \) as integration interval and being \( (x_n, y_n)^T \) the position and \( \theta_n \) the orientation of the \( n^{th} \) pose \( x_n \), the kinematics is as follows:

\[
x_{n+1} = x_n + v_n T_s \cos \left( \theta_n + \frac{\omega_n T_s}{2} \right) \tag{16}
\]

\[
y_{n+1} = y_n + v_n T_s \sin \left( \theta_n + \frac{\omega_n T_s}{2} \right) \tag{17}
\]

\[
\theta_{n+1} = \theta_n + \frac{\omega_n T_s}{2} \tag{18}
\]

Applying (16) in the solution scheme of Section III might result in controls \( u = (v, \omega)^T_{0:N-1} \) that exceed the actuation limits. Hence we need to enforce inequality constraints over
the variables $u_n$, which are not addressed by regular ILS. In the next section, we introduce constraint factors and constrained variables to model both inequality and equality constraints.

Summarizing, the problem can be described through the set of equations

$$\begin{align*}
\min_{x,u} & \Phi_{\text{MPC}}(u,x) \\
\text{s.t.} & \forall n, |v_n| \leq v_{\text{max}}, |\omega_n| \leq \omega_{\text{max}}
\end{align*}$$

(19)

where

$$\Phi_{\text{MPC}}(u,x) = \Phi_{\text{QR}}(u,x) =$$

$$+ \sum_{n=0}^{N-1} \left[ \|e_p^o(x_n,u_n,x_{n+1})\|_{\Omega^p_n}^2 + \|e_p^o(x_n)\|_{\Omega^o_n}^2 \right]$$

(20)

$$+ \|e_n^v(x_N)\|_{\Omega^v_n}^2$$

Let $u_{\text{max}} = (v_{\text{max}}, \omega_{\text{max}})^T$, the Lagrangian function is

$$L(u,x,\lambda_1,\lambda_2,\rho_1,\rho_2) = \Phi_{\text{MPC}}(u,x) +$$

$$\lambda_1^T(u - u_{\text{max}}) + \frac{\rho_1}{2} \|u - u_{\text{max}}\|_2^2$$

(21)

$$\lambda_2^T(-u - u_{\text{max}}) + \frac{\rho_2}{2} \|-u - u_{\text{max}}\|_2^2$$

By imposing the variable ordering $x_0, u_0, x_1, \ldots$, we obtain the maximally sparse structure of the linear system shown in Table I for the case of $N = 4$.

### B. Localization as Registration on a Distance Matrix

Localization aims at estimating the most likely robot pose in the world frame $X^W_R \in SE(3)$, given the robot measurements $z = z_{0:K}$ and the odometry prior $z_O$. In particular, our robot moves on a plane $X^W_R \in SE(2)$ and is equipped with a 2D lidar. The map is given and stored in memory as a 2D matrix $M$ where each cell contains the distance $d(p) : \mathbb{R}^2 \to \mathbb{R}$ of point $p$ from the closest obstacle in the map. Factor graph in Fig. 4 summarizes an instance of such problem.

Let us indicate as $z_k \in \mathbb{R}^2$ a generic laser endpoint expressed in the robot frame. A suitable localization factor associated to the $k$-th measurement would minimize the error function:

$$e^l_k(X^W_R) = d(X^W_R z_k)$$

(22)

where the distance from the laser endpoint to the closest obstacle can be retrieved in $O(1)$ by querying the matrix $M$ at the global coordinates of the laser endpoint $z^W_k = X^W_R z_k$. Should the robot be perfectly localized in a static environment, all distances will be close to zero, and error in (22) would be zero as well. Summing the contribution of all laser endpoints in the laser scan, the localization graph has therefore the following cost function:

$$G(X^W_R) = \sum_{k=0}^{K} e^l_k(X^W_R z_k) \|_{\Omega^l_k}$$

(23)

$$X^W_R = \arg\min_{X^W_R} G(X^W_R)$$

(24)

In the implemented navigation stack, the same extended ILS solver addresses both MPC and localization as described above. Albeit we implemented the error function in (22) because of its compactness, it can be replaced with more elaborated state-of-the-art error functions [9], [22], [33] without changing the solver implementation, as done in [12].

### VI. Experiments

We tested our system both in simulation and on a real custom made unicycle robot equipped with a 2D lidar (Mikrotron LD06), running on a Raspberry Pi 4 board at our Department, whose map is shown in Fig. 5. Comparative experiments have been conducted in simulation with a similar robot configuration, on a laptop Intel(R) Core(TM) i7-10750H CPU running at 2.60 GHz with 16 GB of RAM.
We compare our method with the most recent version of the ROS navigation stack consisting of amcl for localization [23] and move_base with Timed Elastic Band (TEB) [25] local planner for control. The two navigation stacks consist of a global and a local planner. Both ROS and our global planner compute the obstacle-free path by Dijkstra’s algorithm relying on the knowledge of the map. In our navigation stack, the map is given as a 2D occupancy grid map obtained by running a standard SLAM algorithm [15] beforehand. This path is then fed to the local planner that calculates the optimal control, and considers potential dynamic obstacles detected through the lidar. The robot estimates its position by using the factor graph-based localizer of Section V-B.

Further, we feed sample instances of our optimization problem to the well-known state-of-the-art general purpose nonlinear programming solver IPOPT [24]. Our experiments are designed to show the validity of our approach and support our key claims (i) our approach gives repeatable results, on-pair with ROS navigation stack at a (ii) lower CPU consumption, (iii) when approaching constrained optimization problems on the models we analyzed, our generic solver outperforms IPOPT in runtime while producing comparable trajectories and final cost value.

A. Comparison With ROS Navigation Stack

We report here the results obtained by running our navigation stack and the ROS one to travel across 8 goals 30 times within the map of our Department shown in Fig. 5, and 5 goals 30 times within the map shown in Fig. 6. While map in Fig. 6 is better for overall evaluation as it has more corners, map in Fig. 5 alternates long straight corridors and very narrow passages as those in the upper left and right corners. For each path we store the ground truth poses published by ROS stage and the duration in seconds. The path length is computed by summing the distance between subsequent ground truth poses. Fig. 7 summarizes the obtained results, using bars for data, and error bars for standard deviation. Both approaches safely drove the robot during the whole 12 hours of simulation.

As it can be seen in Fig. 7(a)-(c), path length is comparable, being only slightly larger using our approach due to the global planner providing more conservative paths in narrow passages. The repeatability of our approach is also on-par with move_base, giving an average standard deviation of 0.62%, compared to 0.63% of our competitor.

Concerning path duration, Fig. 7(d) shows our approach performs better than move_base in the map of Fig. 6. Instead, Fig. 7(b) shows that the duration is larger for our method on the map of Fig. 5, and depends on the path itself: in long paths, where narrow passages account only for a small portion of the whole path, ROS duration is on average 95% of ours, while on
Fig. 8. Initial guess in green, iterations whitening pink, optimal path in blue: (a) backward maneuver from right to left; (b) control inputs; (c) obstacle avoidance maneuver from left to right; (d) corresponding control inputs.

paths mostly consisting of narrow passages ROS duration is 81% of ours. As already mentioned, this is due to the global planner being more conservative in very narrow passages. However, our approach is on average more repeatable with a standard deviation of 1.57%, compared to 4.35% of our competitor.

The comparable results shown in Fig. 7 come at a considerably lower CPU consumption: our factor graph-based local planner accounts for 3.8%, our global planner for 1.1%, and finally our localization module for 2.0%, giving a total of 6.9%. Differently, move_base accounts for 13.5%, and amcl for additional 2.8%, giving a total of 16.3%. For efficient computation, TEB method addresses the optimal control problem as multi-objective optimization within factor graphs, and uses the popular solver by Kümmerle et al. [19]. TEB relies on polynomial penalty functions to enforce constraints, and augments the state by incorporating the time intervals between two consecutive poses. Since our time step is fixed, the number of variables is lower in our case which contributes in reducing CPU consumption. At the same time, our implementation of AL method is competitive with a penalty-based approach to constrained optimization.

Finally, for the sake of completeness, Fig. 8 reports the internals of an MPC iteration in whitening pink, with associated control input profiles.

B. Comparison with IPOPT

We evaluated the performance of our solver compared to the well-known solver IPOPT, by feeding the optimal control problem in (19) to it, with same weights, limits and initial guess. Four different goals were considered starting from a given initial pose, resulting in both forward and backward trajectories. Table II shows the results obtained, while Fig. 9 plots control inputs optimized with the two methods corresponding to the last row of Table II. As a termination criterion for our solver, we consider the norm of the perturbation vector and that of the constraint violations. Optimization is stopped when:

\[ ||\Delta x||_2 < \epsilon_x, \quad ||Cx + c||_\infty < \epsilon_c, \quad \max(Dx + d, 0)||_\infty < \epsilon_d, \]

where 0 is the vector of all zeros, and \( \epsilon_x, \epsilon_c, \epsilon_d = 1e^{-4} \).

Our solver is around 2 orders of magnitude faster than IPOPT at converging. This makes our solver preferable for online onboard applications. Two aspects make this possible. The first is that, given the same objective functions and constraints, AL iterations are cheaper than those of Interior Point. Thanks to the split in primal and dual update, the dimension of the linear system we solve at each step is equal to the number of actual variables. Instead, in Interior Point the linear system is larger comprising also the set of auxiliary variables, as big as twice the number of inequality constraints plus once that of equality constraints. Further, a line-search procedure is implemented to choose the weight of the step. The second aspect motivating the difference with IPOPT is that the factor graph solver [16], [20] we use is highly optimized for sparse optimization. We can choose proper variable ordering so as to maximize the sparsity pattern, which results in easy inverse computations for linear system solving to compute the update step of ILS. The final cost of our solution is slightly higher, however both a quantitative comparison with the initial value reported in Table II and a graphical comparison based on Fig. 9 show that the two solutions are very similar. In summary, our evaluation suggests that our approach is considerably more efficient than standard non-linear programming solvers.
VII. CONCLUSION

We presented an approach to embedding constraints on variables as factors of a graph, based on the AL method. Having introduced an Iterative Least Squares solver for constrained optimization, we have contributed to extending the variety of problems across robotics that can be modeled using factor graphs.

In this letter, we focus on optimal control for MPC as an application field of constrained optimization to support the validity of our approach. In particular, we develop a full navigation stack based on factor graphs: both localization and MPC are modeled here through factors. Experiments show that our navigation stack compares favorably to the ROS navigation stack. Moreover, experiments conducted with IPOPT confirm that our solver gives the advantage of both an improvement in computational time and a unified formulation for estimation and control problems.

While our navigation stack is a working proof of concept, our method is general and can be easily used to model other classes of problems. As future work in the field of optimal control, different kinematic and dynamic models can be considered. Active SLAM [6], [7] is an example application where both estimation and control are coupled. The two problems share the knowledge of the map and the robot position estimate and contribute to the goal of having a robot that autonomously builds an accurate map in the shortest possible time. In the near future we envision future applications of our findings in this domain.

REFERENCES

[1] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, “The explicit linear quadratic regulator for constrained systems,” Automatica, vol. 38, no. 1, pp. 3–20, 2002.
[2] D. P. Bertsekas, Constrained Optimization and Lagrange Multiplier Methods. New York, NY, USA: Academic press, 2014.
[3] F. Bounini, D. Gingras, H. Pollart, and D. Gruyer, “Modified artificial potential field method for online path planning applications,” in Proc. IEEE Intell. Veh. Symp., 2017, pp. 180–185.
[4] S. P. Boyd, N. Parikh, E. K. W. Chu, B. Peleato, and J. Eckstein, “Distributed optimization and statistical learning via the alternating direction method of multipliers,” Found. Trends Mach. Learn., vol. 3, pp. 1–122, 2011.
[5] J. C. Butcher, “A history of Runge-Kutta methods,” Appl. Numer. Math., vol. 20, no. 3, pp. 247–260, 1996.
[6] H. Carrillo and J. A. Castellanos, “Navigation under uncertainty based on active SLAM concepts,” in Handing Uncertainty and Networked Structure in Robot Control, Berlin, Germany: Springer, 2015, pp. 209–235.
[7] Y. Chen, S. Huang, and R. Fitch, “Active SLAM for mobile robots with area coverage and obstacle avoidance,” IEEE/ASME Trans. Mechatron., vol. 25, no. 3, pp. 1182–1192, Jun. 2020.
[8] M. D. Cicco, B. D. Corte, and G. Grisetti, “Unsupervised calibration of wheeled mobile platforms,” in Proc. IEEE Int. Conf. Robot. Automat., 2016, pp. 4328–4334.
[9] L. Danatanrayana, R. Ranasinghe, and G. Dissanayake, “C-LOG: A chamber distance based method for localisation in occupancy grid-maps,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2013, pp. 376–381.
[10] A. De Luca and G. Oriolo, “Local incremental planning for nonholonomic mobile robots,” in Proc. IEEE Int. Conf. Robot. Automat., 1994, pp. 104–110.
[11] F. Delllaert, “Factor graphs: Exploiting structure in robotics,” Annu. Rev. Control. Robot., Auton. Syst., vol. 4, no. 1, pp. 141–166, 2021.
[12] L. D. Giammarino, I. Aloise, C. Stachniss, and G. Grisetti, “Visual place recognition using LiDAR intensity information,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2021, pp. 4382–4389.
[13] J. Dong, M. Mukadam, B. Boots, and F. Dellaert, “Sparse gaussian processes on matrix lie groups: A unified framework for optimizing continuous-time trajectories,” in Proc. IEEE Int. Conf. Robot. Automat., 2018, pp. 6497–6504.
[14] J. Dong and M. Mukadam, F. Dellaert, and B. Boots, “Motion planning as probabilistic inference using gaussian processes and factor graphs,” in Proc. 12th Robot.: Sci. Syst., Ann Arbor, MI, USA, Jun. 2016, doi: 10.15607/RSS.2016.XII.001.
[15] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, “A tutorial on graph-based SLAM,” IEEE Trans. Intell. Transp. Syst. Mag., vol. 2, no. 4, pp. 31–43, Winter 2010.
[16] G. Grisetti, T. Guadagnino, I. Aloise, M. Colosi, B. D. Corte, and D. Schlegel, “Least squares optimization: From theory to practice,” Robotics, vol. 9, no. 3, Jul. 2020, Art no. 51, doi: 10.3390/robotics9030051.
[17] G. Grisetti, R. Kümmerle, and K. Ni, “Robust optimization of factor graphs by using condensed measurements,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2012, pp. 581–588.
[18] V. Hal, L. Polok, M. Solony, and P. Svoboda, “SLAM—a highly efficient and temporally scalable incremental SLAM framework,” Int. J. Robot. Res., vol. 36, no. 2, pp. 210–230, 2017.
[19] M. Kaess, A. Ranganathan, and F. Dellaert, “iSAM: Incremental smoothing and mapping,” IEEE Trans. Robot., vol. 24, no. 6, pp. 1365–1378, Dec. 2008.
[20] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, “G2O: A general framework for graph optimization,” in Proc. IEEE Int. Conf. Robot. Automat., 2011, pp. 3607–3613.
[21] M. Mukadam, J. Dong, F. Dellaert, and B. Boots, “STEAP: Simultaneous trajectory estimation and planning,” Auton. Robots, vol. 43, pp. 415–432, 2019.
[22] H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. I. Nieto, and R. Y. Siegwart, “Signed distance fields: A natural representation for both mapping and planning,” in Proc. Workshop: Geometry Beyond-Representations Phys. Scene Understanding Robot., Univ. Michigan, 2016.
[23] P. Pfaff, W. Burgard, and D. Fox, “Robust monte-carlo localization using adaptive likelihood models,” in Proc. Eur. Robot. Symp., 2006, pp. 181–194.
[24] F. A. Potra and S. J. Wright, “Interior-point methods,” J. Comput. Appl. Math., vol. 124, no. 1, pp. 281–302, 2000.
[25] C. Rönnlmann, W. Feiten, T. Wösch, F. Hoffmann, and T. Bertram, “Efficient trajectory optimization using a sparse model,” in Proc. Eur. Conf. Mobile Robots, 2013, pp. 138–143.
[26] C. Rönnlmann, F. Hoffmann, and T. Bertram, “Kinodynamic trajectory optimization and control of car-like robots,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2017, pp. 5681–5686.
[27] L. J. Schonberger and J. Frahm, “Structure-from-motion revisited,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 4104–4113.
[28] J. Steir, M. Saad, and H. Salahi-Hassane, “An improved artificial potential field approach to real-time mobile robot path planning in an unknown environment,” in Proc. IEEE Int. Symp. Robot. Sensors Environ., 2011, pp. 206–213.
[29] P. Sadhii, S. Choudhury, J. G. Mangelson, and M. Kaess, “ICS: Incremental constrained smoothing for state estimation,” in Proc. IEEE Int. Conf. Robot. Automat., 2020, pp. 279–285.
[30] D. Ta, M. Kobilarov, and F. Dellaert, “A factor graph approach to estimation and model predictive control on unmanned aerial vehicles,” in Proc. IEEE Int. Conf. Unmanned Aircr. Syst., 2014, pp. 181–188.
[31] M. Xie, A. Escontrela, and F. Dellaert, “A factor-graph approach for optimization problems with dynamics constraints,” 2020, arXiv:2011.06194.
[32] S. Yang, G. Chen, Y. Zhang, H. Choset, and F. Dellaert, “Equality constrained linear optimal control with factor graphs,” in Proc. IEEE Int. Conf. Robot. Automat., 2021, pp. 9717–9723.
[33] M. Zhang, Y. Chen, and M. Li, “SDF-Loc: Signed distance field based 2D relocalization and map update in dynamic environments,” in Proc. IEEE Amer. Control Conf., 2019, pp. 1997–2004.