Estimation and planning differential drive robot simultaneously - Technical Report*

Tai Hoang[1]
University of Information Technology
Thu Duc District, Ho Chi Minh City, Viet Nam
thobotics@gmail.com

***Abstract*— Estimation and planning play a vital role in the construction of an autonomous navigation framework. However, these problems are often considered separately, while planning gives robot a free-collision path towards the desired goal, estimation algorithm presents the executed trajectory in the sense that it has to be closed to the ground truth path as much as possible. Recently, a unified probabilistic framework, which supports solving these problems simultaneously, dubbed STEAP has been proposed. Nevertheless, its current version is only designated for an omni wheels robot, which allows robot to move and turn in vertical direction. Differential drive robot, on the other hand, though limited to move along only one direction, has been used in various situations due to its flexibility and lower cost in hardware designing. Thus, in this extension, our aim is to control a differential drive robot via STEAP. Moreover, in a more complicated environment such as labyrinth or maze, the original STEAP sometimes fails to find a path. Indeed, this problem is mainly caused by the poor initialization and the non-linearity in optimizer constraints. In our implementation, instead of dealing with these constraints, we employ a global planner algorithm such as Dijkstra or RRT to treat STEAP as an effective local planner module that focus on following the global path. Consequently, the experimental results show that the extended STEAP not only able to navigate a differential drive robot but also in a more complicated and unstructured environment.**

***Index Terms*— SLAM, Trajectory Optimization, Motion Planning, Gaussian Process, STEAP, GPMP2, Mobile Robot, Differential Drive Robot**

I. INTRODUCTION

In recent years, Simultaneous Localization and Mapping (SLAM) has received a remarkable attention from mobile robot communities. It plays an indispensable role in attempting to autonomously navigate a mobile robot. In fact, it enables ones to explore and navigate in a completely unknown environment which is useful for different applications such as urban reconstruction or search-and-rescue operations. Specifically, while mapping and localization have to be computed sequentially, the uncertainty generated in real-world scenarios such as limited sensing capabilities, model inaccuracy and noisy movement even makes SLAM become more challenging to generate an accurate map. Thus, to account for those unexpected behaviours, SLAM is often modelled as an iterative probabilistic inference problem. Although there exist several efficient solutions based on this modelling, running SLAM in a complicated and large-scale environment is a slightly different story. The larger the map the harder the problem is, to efficiently solve it, Smoothing and Mapping (SAM) [1] introduced a new inference strategy. By using factor graphs and then exploiting the sparsity of the linear systems which is generated by the aforementioned modelling, SAM not only able to generate a more accurate map but also in an acceptable responded time. Next, [2] further extends SAM to even deal with higher dimensional problems such as mapping in an uneven terrain, instead of discretizing the whole states, they reformulate the trajectory as a continuous-time function that maps any given time to a corresponded robot state. To be specific, this work heavily relies on Gaussian Process (GP), which is a popular non-parametric probabilistic framework to model these continuous-time trajectories. Hence, it obviously benefits from the advantages of GP, especially GP interpolation. As a result, lots of computational burdens, which are caused by numerous parameters from the original method, witnessed a remarkable reduction.

On the other hand, in robot manipulation communities, motion planning is a fundamental skill for robot that helps them search for a feasible and optimal trajectory towards goal in the robot's configuration space. The found trajectory must be free-collision and enforcing the robot's physical limitations (dynamic constraints). In general, there are two popular research orientations that aim to tackle this problem, include sampling-based and trajectory optimization based approaches. The former algorithm; for example, Probabilistic Road Map [3] and Rapidly exploring Random Tree [4] able to find an optimal and feasible trajectory in complex high-dimensional configuration spaces; however, this trajectory often lacks quality and may result in a jerky movement. The latter approach, in turn, attempts to generate smooth and free-collision trajectories throughout the cost function optimization while satisfying different criteria such as task, physical limitations and environment. Take CHOMP [5, 6] as an example, covariant gradient descent is employed to optimize a cost functional while ensuring smoothness and obstacle avoidance constraints. A novel method STOMP [7] based on this and is developed further to deal with non-differentiable constraints throughout stochastic sampling. However, one of the main drawbacks of these approaches is due to the discretization, which results in a significant increase of needed optimization parameters, may eventually

*This work was not supported by any organization lead it to an unsolvable problem. In recent years, building upon the work of [2], [8] discovered that by using Gaussian Process to model the continuous-time trajectory distribution, the original trajectory optimization can be treated as a probabilistic inference problem. Thus, it obviously reaps the benefits from GP area included GP interpolation and faster replanning strategy (iSAM2 [9]).

As mentioned earlier, although probabilistic inference is initially used in mobile robot area, recent researchers, throughout Gaussian Process and factor graphs, has bridged the gap between it and motion planning problem. In particular, [10] further extends this idea and propose a novel method in which instead of solely planning, they combine estimation and planning into one bigger single problem and dubbed Simultaneous Trajectory Estimation and Planning (STEAP). Although STEAP has shown a capability of efficiently navigating a high-dimensional mobile manipulator robot, it often fails to find a feasible path in a more complicated and unstructured environment. This can be explained by the poor initialization which often leads the optimizer to a bad local optima area.

To deal with it, in this paper, we employ the potential function from global planner algorithm to guide the optimizer to global optimum point. That is, STEAP can now be seen as analogous to a local planner in general mobile navigation framework. Furthermore, due to the considerable amount of cost in hardware designing of an omni-wheels robot, in most case, the differential drive robot is more preferred. In this extension, we also proved that STEAP can be further extended to safely navigate this kind of robot in an unstructured environment. Interestingly, instead of sending raw velocity commands from local planner to robot, we adopt a provably stable controller [11] to make it gratefully and accurately navigate robot towards a planned waypoint.

II. BACKGROUND

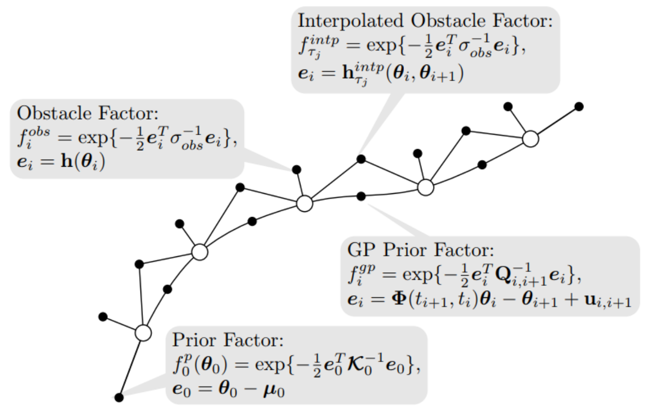
A. *Gaussian Process Motion Planning*

This section will briefly discuss the basic concept of Gaussian Process Motion Planning 2 (GPMP2) [8] and its ability on effective re planning or online planning execution throughout the use of iSAM2 [9] framework.

Intuitively, by using GPMP2, one can treat the problem of estimating or optimizing continuous-time trajectories as a probabilistic inference problem. By representing a trajectory as a mapping from time t to robot states $\theta(t) : t \to \mathbb{R}^D$, where D is the dimension of state. The goal is to find the *maximum a posterior* (MAP) continuous-time trajectory given a prior distribution and a likelihood function.

First, let assume a prior distribution over trajectories defined as a vector-valued Gaussian Process $\theta(t) \sim \mathcal{GP}(\mu(t), \mathcal{K}(t, t'))$, where $\mu(t)$ is a vector-value mean function, and $K(t, t')$ is represented as a matrix-value covariance function.

Next, one can define the prior distribution throughout the GP framework



Interpolated Obstacle Factor:
$f^{intp}_{\tau_j} = \exp\{-\frac{1}{2}\boldsymbol{e}_i^T \sigma_{obs}^{-1} \boldsymbol{e}_i\}$,
$\boldsymbol{e}_i = \mathbf{h}^{intp}_{\tau_j}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1})$
Obstacle Factor:
$f^{obs}_i = \exp\{-\frac{1}{2}\boldsymbol{e}_i^T \sigma_{obs}^{-1} \boldsymbol{e}_i\}$,
$\boldsymbol{e}_i = \mathbf{h}(\boldsymbol{\theta}_i)$
GP Prior Factor:
$f^{gp}_i = \exp\{-\frac{1}{2}\boldsymbol{e}_i^T \mathbf{Q}_{i,i+1}^{-1} \boldsymbol{e}_i\}$,
$\boldsymbol{e}_i = \boldsymbol{\Phi}(t_{i+1}, t_i)\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i+1} + \mathbf{u}_{i,i+1}$
Prior Factor:
$f^p_0(\boldsymbol{\theta}_0) = \exp\{-\frac{1}{2}\boldsymbol{e}_0^T \boldsymbol{\mathcal{K}}_0^{-1} \boldsymbol{e}_0\}$,
$\boldsymbol{e}_0 = \boldsymbol{\theta}_0 - \boldsymbol{\mu}_0$

Fig. 1. This figure illustrates how a trajectory can be represented as a factor graph, which includes support states (white circles) and four kinds of factors (black dots). While start and goal factors are respectively placed at the head and tail of the trajectory, GP prior factors serve as a connection between two consecutive support states, and finally the environmental information is conveyed via obstacle and interpolated obstacle factors.

$$p(\theta) \propto \exp\left\{-\frac{1}{2}\|\theta - \mu\|^2_K\right\}$$

As mentioned in [8], this prior distribution can be used for encoding the system dynamics via a structured GP in robot state estimation problems. Another important distribution need to be considered is the likelihood function $l(\theta; e) = p(e|\theta) \propto \exp\left\{-\frac{1}{2}\|h(\theta, e)\|_\Sigma\right\}$, where e is a binary event or observation value, and $h(\theta, e)$ can be any vector-value cost function with covariance matrix Σ. Combined together, our final problem for finding the desired trajectories $\theta(t)$ can be solved via the *maximum a posterior* (MAP) computing of θ

$$\theta^* = \text{argmax}_\theta p(\theta|e) = \text{argmax}_\theta p(\theta) l(\theta; e) \qquad (1)$$

To effectively solve the optimization problem in Equation 1, the authors further exploit the use of *factor graph* to decompose the posterior distributions into different components which correspond to different aims of the robot. For example, let turn the original posterior distribution into a factorized form

$$p(\theta|e) \propto \prod_{m=1}^{M} f_m(\Theta_m), \qquad (2)$$

where f_m are factors on variable subsets Θ_m.

In particular, one can treat the objectives of trajectories optimization problem as different factors which is illustrated in Figure 1. First factor would be *Prior Factor*, $f^{start}(\theta_0)$ and $f^{goal}(\theta_N)$ which defines the prior distributions on start and goal states respectively

$$f^p_i(\theta_i) = \exp\left\{-\frac{1}{2}\|\theta_i - \mu_i\|^2_\Sigma\right\}, i = 0 \text{ or } N \qquad (3)$$

Then, *GP Prior Factor* $f^{gp} = \prod_i f^{gp}_i(\theta_i, \theta_{i+1})$ is used to encodes the robot's dynamical system and are generated by a *Linear Time Variant - Stochastic Differential Equation*

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 11 February 2019 doi:10.20944/preprints201902.0084.v1

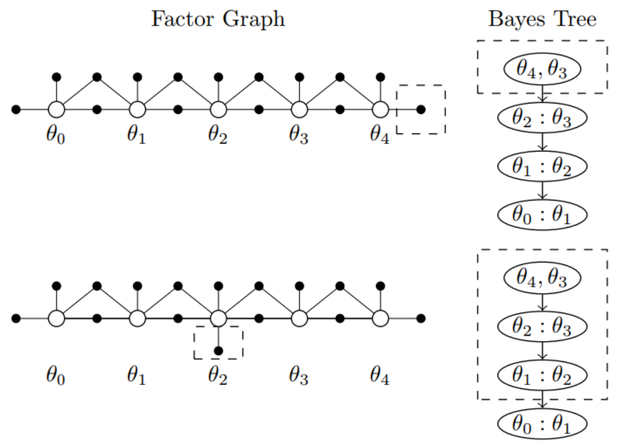


Fig. 2. Examples of Bayes Trees in replanning procedure. Parts of the factor graphs and its correspondingly affected leaves of Bayes Tree are indicated in dashed boxes.

(LTV-SDE), further details can be found on [8]. Finally, to make the robot move safely towards a goal, two types of obstacle cost are also employed, *Obstacle Factor* f_i^{obs} and *Interpolated Obstacle Factor* $f_{\tau_i}^{intp}$ which can be represented as one big obstacle factor

$$\begin{aligned} f^{obs} &= \prod_{i=0}^{N} \left\{ f_i^{obs}(\theta_i) \prod_{j=1}^{n_{ip}} f_{\tau_i}^{intp}(\theta_i, \theta_{i+1}) \right\} \\ &= \prod_{i=0}^{N} \left\{ \exp\left(-\frac{1}{2}\|h(\theta_i)\|^2_{\sigma_{obs}}\right) \right. \\ &\qquad \left. \prod_{j=1}^{n_{ip}} \exp\left(-\frac{1}{2}\|h_{\tau_i}^{intp}(\theta_i, \theta_{i+1})\|^2_{\sigma_{obs}}\right) \right\} \end{aligned} \tag{4}$$

The final form of the trajectory optimization problem after factorizing can be seen as

$$\begin{aligned} \theta^* &= \text{argmax}_\theta p(\theta|e) = \text{argmax}_\theta p(\theta) l(\theta; e) \\ &= \text{argmin}_\theta \{-\log(p(\theta) l(\theta; e))\} \\ &= \text{argmin}_\theta \left\{ -\frac{1}{2}\|\theta - \mu\|^2_{\mathcal{K}} + \frac{1}{2}\|h(\theta)\|^2_{\Sigma} \right\} \end{aligned} \tag{5}$$

The apparent construction of the posterior distribution is reduced to a nonlinear least squares problem and then can be solved with any non linear solver like Gaussian Newton or Levenberg-Marquardt. However, using such solver is computational inefficient due to the uncertainty generated in real world scenarios which requires a robot re-solving the optimization problem several time.

To help the re-planning execution more effectively, iSAM2 [9] is applied due to the aforementioned duality between trajectories optimization and probabilistic inference throughout the use of *factor graph*. iSAM2 is an efficient incremental inference framework which was first proposed to solve the SLAM problem. Intuitively, throughout the use of a new data structure *Bayes tree*, instead of updating all factors on the factor graph, only neighboured factor will be updated in the re-planning execution which results in a much faster planning in comparison to the naive approach. Figure 2 shows that the graph updating of the top figure only affects θ_4, θ_3 factor while any factors between θ_2 except the last one will be updated on the remaining figure.

B. Simultaneous Trajectories Estimation and Planning

To be able to autonomously navigate in an unknown environment, it requires solving both the problem of trajectory estimation (localization) and planning. These two have been successfully tackled in a separate manner for a decade. However, while trajectory estimation look like a backward-looking problem where an executed trajectory need to be re-estimated due to incomplete sensor data, the work of finding a clear path to the desired goal seems to be matched with the forward-looking problem. In *Simultaneous Trajectories Estimation and Planning* (STEAP) article, the authors propose a new method which tackle both problems simultaneously instead of the traditional two-step approach.

Intuitively, similar to previous section, STEAP is heavily relied upon the successful of *factor graph*. In fact, they first treat both the trajectory estimation and planning problem into one single problem and then apply the exact solver and re planning algorithm to tackle this problem. Thus, the whole work of STEAP seems to be inclined on the designing of a suitable *factor graph*.

Figure 3 shows an example of STEAP as a simple solution for navigation problem. First, the graph is constructed with five different robot states, equivalent to five different time step which are connected via *GP factors* that collectively form a prior distribution of a continuous-time trajectory. Then, *start and goal factor* in addition with a set of *obstacle factor* are added to make a mobile robot navigate from current position to the desired goal without collided. Start, goal and obstacle factors together form the likelihood or planed path, correspond to the blue line in Figure 3. Next, in each execution step, the robot incrementally updates the state with current *measurement factor*, form a new factor graph and eventually resolve the MAP problem by updating the Bayes Tree via iSAM2 algorithm.

III. Methodologies

A. GP Priors

As mentioned earlier, *GPMP2* can be seen as a general framework for tackling different trajectory optimization problems. Two has been solved in the previous sections which are *Motion Planning* and *Mobile Navigation*, thus by designing a suitable *factor graph*, one can benefit from the efficient in planning and execution of the aforementioned framework without further modification. In fact, while both GPMP2 and STEAP aim to solve the same problem - moving a robot towards the desired goal without collided, GPMP2 focus on *manipulation task*, and STEAP tackle the *mobile robot* navigation problem. The main difference between these two is due to the difference from dynamical systems, thus, *GP prior factor* was employed to serve this purpose. Instead

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 11 February 2019 doi:10.20944/preprints201902.0084.v1

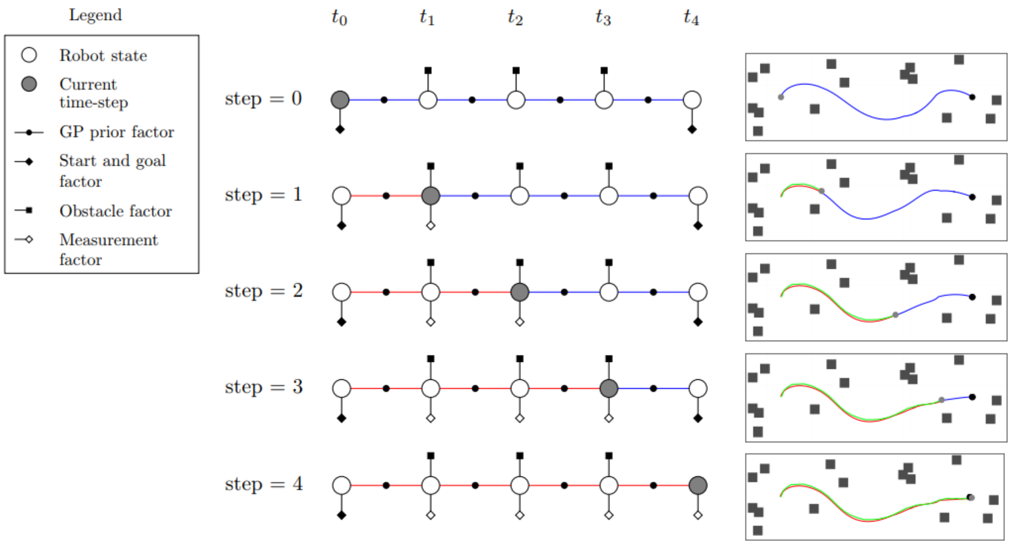


Fig. 3. This figure illustrates how a navigation procedure is represented throughout STEAP, a gray and black circle are represented for the robot and the desired goal, respectively. At each step, three different trajectories are drawn in both factor graphs and the simulated environment, includes ground-truth (green), estimated (red), and replanned (blue) trajectories.

of using a trivial Gaussian kernel such as RBF, both STEAP and GPMP2 are relied upon the linear time varying (LTV) stochastic differential equations (SDEs) GP priors which is proposed in [2]. These priors are highly structured and factorized according to

$$f^{gp} = \prod_i f_i^{gp}(\theta_i, \theta_{i+1}) \tag{6}$$

where any GP prior factor connects to only its two neighboring states, forming a (Gauss-Markov) chain. In specific, a *constant-velocity* model $\ddot{p}(t) = w(t)$ with N-dimensional position variable $p(t)$, is employed to define a LTV-SDE

$$\dot{\theta}(t) = A(t)\theta(t) + u(t) + F(t)w(t) \tag{7}$$

where $\theta(t) = \begin{bmatrix} p(t) & \dot{p}(t) \end{bmatrix}^T$, $u(t)$ is the control input, $A(t)$ and $F(t)$ are time-varying system matrices, and eventually the white process noise $w(t)$ which is represented by

$$w(t) \sim GP(0, Q_C\delta(t - t')) \tag{8}$$

where Q_c is the hyperparameter power-spectral density matrix, $\delta(t - t')$ is the Dirac delta function. We refer the reader to [2] for further details. Next, to form an exact *constant-velocity* model, $u(t)$, $A(t)$ and $F(t)$ need to be defined as

$$A(t) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, u(t) = 0, F(t) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{9}$$

Once formed the LTV-SDE final formulation, according to [2], one can define the GP factor between any two states at t_i and t_{i+1} as follow

$$f_i^{gp}(\theta_i, \theta_{i+1}) = \exp\left\{ -\frac{1}{2} \|\Phi(t_{i+1}, t_i)\theta_i - \theta_{i+1}\|^2_{Q_{i,i+1}} \right\} \tag{10}$$

where the covariance matrix $Q_{i,i+1} = \begin{bmatrix} \frac{1}{3}\Delta t_i^3 Q_c & \frac{1}{2}\Delta t_i^2 Q_c \\ \frac{1}{2}\Delta t_i^2 Q_c & \Delta t_i Q_c \end{bmatrix}$, and transition function $\Phi(t_{i+1}, t_i)$ is defined below if $p(t)$ is a purely value-vector (configuration vector) as the case defined in GPMP2 framework

$$\Phi(t, s) = \begin{bmatrix} 1 & (t-s)1 \\ 0 & 1 \end{bmatrix} \tag{11}$$

On the other hand, not all robots' configuration spaces can be well represented in vector spaces. For example, the orientation of mobile robot in planar space cannot be treated as vectors without singularity (Euler angle) or extra degrees (quaternion). Thus, to be able to apply the *constant velocity* model directly, STEAP adopt the linearization on the *Lie group* to transform the state $\theta(t) \in$ SE2 to $\xi(t) \in \mathbb{R}^N$. However, it not easy to employs this method to control a differential drive robot due to the non-holonomic constraint on the kinematic system. Therefore, in this implementation, to embed such constraint, we incline to use a simpler approach where the pose of robot on canonical coordinates $p(t) = \begin{bmatrix} x(t) & y(t) & \phi(t) \end{bmatrix}^T \in \mathbb{R}^3$ is represented as system state, and then employs the use of differential drive kinematic equation to form the velocities $\dot{p}(t)$

$$\dot{p}(t) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \cos\phi & 0 \\ \sin\phi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{12}$$

where v and w is linear and angular velocities command, respectively.

On the contrary, in the execution process, by treating the system state as a combined vector between position and velocity states $\theta(t) = \begin{bmatrix} p(t) & \dot{p}(t) \end{bmatrix}^T$, a naive approach would use the returned velocity to control the robot. This approach only works well if the number of *support states* is large, which will cause the tradeoff between computational efficiency and feasibility. Thus, instead of using the raw velocities, we apply the controller which provably stable via *Lyapunov function* and computational efficiency from the work [11].

$$v = \frac{v_{max}}{1 + \beta \|\kappa\|^{\lambda}}, \quad \omega = \kappa v \tag{13}$$

where v_{max}, λ, β are hyperparameters, and κ is defined as follow

$$\kappa = -\frac{1}{r}\left[k_2(\delta - \arctan(-k_1\theta)) + \left(1 + \frac{k_1}{1+(k_1\theta)^2}\right)\sin\delta\right] \tag{14}$$

where k_1 and k_2 are shape and gain parameters, respectively; and $\begin{bmatrix} r(t) & \theta(t) & \delta(t) \end{bmatrix}^T$ is the transformed state on egocentric polar coordinate of two consecutive *support state* $\begin{bmatrix} x(t) & y(t) & \phi(t) \end{bmatrix}^T$ and $\begin{bmatrix} x(t') & y(t') & \phi(t') \end{bmatrix}^T$.

B. Global potential factor

The current implementation of STEAP[1] poses many challenges which may produce an unstable or infeasible solution when dealing with the unstructured in real-world environment. First, STEAP currently uses a very simple initialization strategy where the whole initial trajectory is just the difference between the start and goal pose; and due to heavily relied on the iterative optimization engine, a poor initial solution may lead the optimization algorithm to a bad local optimum area. Next, a subsequent problem is caused by the use of *Prior factor* at goal state in Equation 3. In fact, due to the local update procedure of the iterative based optimization, experimentally, terminating at the exact goal pose becomes difficult or infeasible if the desired goal is located at a narrow place or surrounded by walls. This difficulty is also partly caused by the use of hinge loss in *Obstacle factor* in Equation 4 due to the fact that hinge loss is fundamentally a non-smooth function, which exists an indifferentiable point in the function domain.

To deal with these dilemmas, instead of planning and executing the mobile robot directly via STEAP, we treated it as a *local planner* module by adopting the results from *global planner* to guide the optimization procedure, like the traditional approach. While the final discrete path which is given by *global planner* (Dijkstra) is used for better initialization, to guide the robot to the near-goal region without collided, we employ the *potential function* factor

$$f_i^{pot}(\theta_i) = \exp\left\{-\frac{1}{2}\|\text{Potential}(\theta_i)\|_{\Sigma}^2\right\} \tag{15}$$

Potential function return the cost at any given pose, which follows the rule "the lower the cost the nearer the goal". In addition, by imposing a higher cost for occupied regions, it ensures that no intervened obstacle appears on the returned path. *Potential function* is also a continuous differentiable function, thus obviously eliminate the potential problem which is caused by hinge loss. However, one of the main drawbacks of this method is the tradeoff between accuracy and stability it causes, by guiding the robot towards a low-cost region instead of the exact goal, it ensures that the optimizer must generate the trajectory that follows the *Global planner*, thus more stable but low accuracy. Fortunately, this tradeoff can be mitigated by adjusting the resolution of the potential function. The higher resolution would strictly penalize the optimizer if the robot is situated in high-cost regions rather than the near-goal one.

[1]https://github.com/gtrll/piper

C. Pseudo code

In this section, the whole *extended STEAP* framework for differential drive robot navigation is described as follow

Algorithm 1 Extended STEAP

```
Input: planner_path: Planner path, potential: Potential function, θ: trajectory, f: factor, T: Bayes tree
    Initialization :
 1: $\theta_{init}$ = initializeTrajectory(planner_path)
 2: FG = createFactorGraph($f^{gp}$, $f^{pot}$, potential, $f^{fix}$)
 3: $\theta$ = inference(FG, $\theta_{init}$)
 4: $T$ = createBayesTree(FG, $\theta$)
    Execution :
 5: for $i = 0$ to $N - 1$ do
 6:     $u$ = controlLaw($\theta$, $i$, $i + 1$)
 7:     execute($u$)
 8:     $f_{i+1}^{meas}$ = localize()
 9:     $\theta$, $T$ = incrementalInference($\theta$, $f_{i+1}^{meas}$, $T$)
10: end for
11: return success
```

As illustrated, the main difference between two version of STEAP is the initial *factor graph* and the *executing velocities*. First, at line 2 of Algorithm 1, the obstacle factor f^{obs} is replaced by the potential one f^{pot} which is described in Equation 15. Meanwhile, instead of computing the command output $u = \begin{bmatrix} v & \omega \end{bmatrix}^T$ throughout interpolation, at line 6, we apply the controlLaw formula from Equation 13 at current time step of two consecutive support state θ_i and θ_{i+1} and send this to robot via *execute* function in the following line.

IV. Experimental results

A. Experimental setup

In this section, two experiments are conducted. While the first experiment shows the power of the potential factor in planning, the latter focus on illustrating how the aforementioned Lyapunov provably stable controller [11] serves for the purpose of an efficient execution. In addition, Robot Operation System[2] (ROS) is heavily employed in this implementation. It is chosen due to the flexibility in transferring between simulation and real-world robot. For simplicity, all experiments are run throughout the STAGE[3] which in

[2]https://ros.org
[3]http://wiki.ros.org/stage

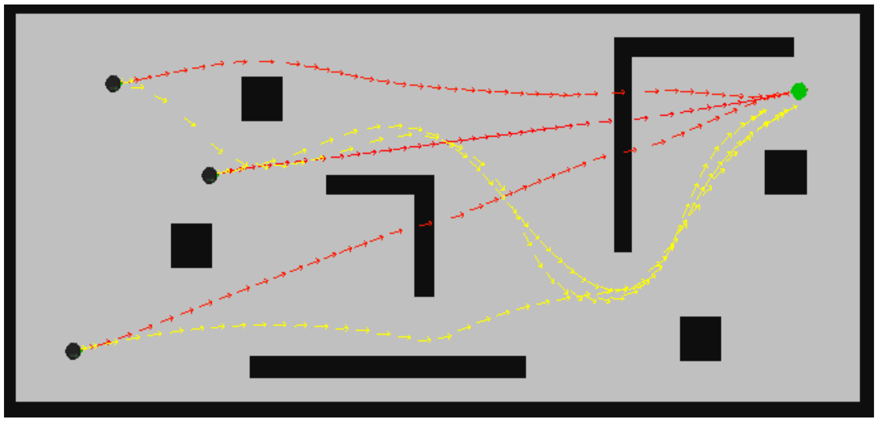


Fig. 4. An experimental result shows the planned trajectories resulted from the original STEAP (red) and our extended version (yellow) in a complicated and unstructured environment. A robot is initially located at three different position (black circle) are searching for a free-collision path towards the goal (green circle).

turn is one of the most popular simulation frameworks for robot navigation task in planar terrain. This implementation is based on the open source framework GTSAM[4] where the non-linear solver Levenberg-Marquardt and replanning iSAM2 algorithms are employed. Meanwhile, throughout the implementation of our algorithm, Dijsktra was used as a global planner under ROS navigation_stack framework. Beside those qualitative comparisons, it also worth to mention the computational time of the aforementioned method. Thus, to be specific, all the experiments are run on a desktop with CPU Xeon v1350, 16 Gb RAM, and GPU NVIDIA GTX 1060.

B. Planning

To begin with, the aim of this experiment is to show how the potential factor benefits in making a feasible plan toward a goal in an unstructured environment like the one in Figure 4. As illustrated, a differential drive robot is situated in three different start position, corresponding to the filled black circle in the figure, are all planning toward the green goal location. The resulted trajectories found by STEAP and our extended version are shown under red and yellow set of arrows, respectively. As shown, in all three case, due to the unstructured and complicated of the environment, it obviously shows that without the help of a global planner, STEAP failed to find a path toward the desired goal. Meanwhile, our method not only successfully gives the robot a free-collision path, but also in a reasonable time which is described in Table I.

TABLE I
COMPUTATIONAL TIME OF PLANNERS

	STEAP	Our
Top	6.9954	0.6498
Middle	9.2135	0.4751
Bottom	7.0078	0.1862

[4]https://bitbucket.org/gtborg/gtsam/

In Table I, with three different start position - top, middle and bottom, it clearly shows that our method outperformed STEAP in term of time complexity. While in all case, STEAP requires a large number of iteration to reach the final trajectory (but still infeasible), our version successfully found a feasible one in only about half of second.

C. Executing

Next experiment will show how much important the control law [11] is in the execution of differential drive mobile robot. For simplicity, instead of the previously complicated environment, a simpler scenario[5] will be considered in this case. That is, a simple trajectory which is a result of our planning strategy requires the robot to reach the desired goal in 20 seconds with 40 support states. To illustrate the power of *estimation* and *planning*, two evaluation metrics are employed in this experiment. While the former (Equation 16) shows the Euclidean distance between the ground-truth terminated pose and the oracle goal (from our planner), the accumulated distance between estimated and ground-truth state (Equation 17) is evaluated in the latter case.

$$\text{error}_{plan} = \|\theta_{end} - \text{gt}_{end}\|_2 \tag{16}$$

$$\text{error}_{est} = \sum_i \|\theta_i - \text{gt}_i\|_2 \tag{17}$$

Moreover, in order to deal with uncertainty, we ran the robot 5 times and report the average, min and max error in these following table

TABLE II
GOAL ERRORS

	Raw	Our
Average	1.1487	0.2144
Max	1.4633	0.3511
Min	0.8768	0.0943

[5]https://youtu.be/0G3FmNgEhMs

TABLE III
ACCUMULATED ESTIMATION ERRORS

	Raw	Our
Average	28.1654	8.6189
Max	30.4067	10.8772
Min	24.3155	4.7663

As illustrated, without the control law, both the estimated and goal-based error are significantly larger than the used one. To be specific, Table II shows that the generated error between oracle and true goal of the one uses control law command, on average, is five times lesser than the raw one. Meanwhile, the estimation accuracy between the estimated and ground-truth path also witness a remarkable improvement. That is, in Table III, it shows that the average estimation error is significantly different (about three times smaller) between two execution strategy. This proves that the accurate execution towards planned ahead way points significantly contribute to the reduction of backward state estimation error.

V. CONCLUSIONS

This work has shown a potential research direction for efficiently controlling and navigating a highly dimensional robot (9 DOF mobile manipulation) in a complex and unstructured environment. To be successfully manipulated, it includes two sequential phase; first, the robot needs to search for a free-collision path towards the goal and then effectively executing it. This can be seen as an analogy to the global and local planner procedure in general navigation framework. Specifically, in this work, while the optimizer is initialized and guided by global planner via potential factor, the iSAM2 [9] re-planning technique plays a role as a local planner where only needed factors are updated instead of resolving the entire optimization problem from scratch. Interestingly, rather than applying the raw velocities resulted from the optimizer, we proved that the robot can gratefully and accurately reach a planned ahead waypoint throughout the provably stable controller. However, for simplicity, Dijkstra serves as a global planner throughout this paper, thus, only the efficiency of planar navigation is shown. In future work, we would explore the capability of different sampling-based planners [4] to control the whole body (included attached arm) in a cluttered environment. Generally speaking, one of the main drawbacks of motion planning is about lacking skills from kinesthetic teaching rather than the manually designed cost. Thus, another interesting direction would be discovered the possibility of factor graphs when dealing with different objective functions. One of such is the potential of combining reinforcement learning to account for demonstrated trajectory from human [12].

REFERENCES

[1] F. Dellaert and M. Kaess, "Square root sam: Simultaneous localization and mapping via square root information smoothing," *Int. J. Rob. Res.*, vol. 25, no. 12, 2006.

[2] S. Anderson, T. D. Barfoot, C. H. Tong, and S. Sarkka, "Batch nonlinear continuous-time trajectory estimation as exactly sparse gaussian process regression," *Autonomous Robots*, vol. 39, no. 3, pp. 221–238, 2015.

[3] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[4] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, IEEE.

[5] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *2009 IEEE International Conference on Robotics and Automation*, IEEE, 2009.

[6] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[7] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011.

[8] 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.

[9] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the bayes tree," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 216–235, 2011.

[10] M. Mukadam, J. Dong, F. Dellaert, and B. Boots, "STEAP: Simultaneous trajectory estimation and planning," *Autonomous Robots*, 2018.

[11] J. J. Park and B. Kuipers, "A smooth control law for graceful motion of differential wheeled mobile robots in 2d environment," in *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011.

[12] P. Englert and M. Toussaint, "Learning manipulation skills from a single demonstration," *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 137–154, 2017.