Dynamic Real-time Replanning in Belief Space: An Experimental Study on Physical Mobile Robots

Ali-akbar Agha-mohammadi, Saurav Agarwal, Aditya Mahadeval, Daniel Tomkins, Jory Denny, Suman Chakravorty, Nancy M. Amato

Abstract

Methods based on the POMDP (Partially-Observable Markov Decision Process) framework for planning under uncertainty rely on the knowledge about the system model and the noise models. However, in practical cases there are always discrepancies between the models used for computation and the real-world models. This paper proposes a dynamic replanning scheme which can perform real-time replanning as the system deviates from the desired plan or encounters any sign of such discrepancies. The main contribution of this paper is to implement a belief space planner on a physical robot and making POMDP methods one step closer to becoming a practical tool for robot motion planning. We demonstrate the planning results on an i-robot create equipped with a monocular camera. Moreover, we show that not only is the proposed method robust to model discrepancies, but also that it is robust to changes in the environment (both in the obstacle map and information sources), as well as unforeseen large deviations in robot's location.

I. Introduction

Motivating example: Consider an autonomous low-cost mobile robot, working in an office environment. Each time it visits a goal location (or accomplishes a task), a new goal location (task) is assigned to it. Therefore, the robot needs to change its plan according to each goal. Moreover, although in an office-like environment most of objects are stationary, there exist objects whose state may change discretely (such as office doors that may get open or closed). Therefore, robot needs to replan when it encounters such changes in the environment. However, low-cost robots are not able to follow the control commands exactly due to the motion noise and they do not have exact measurements due to the sensor noise. Threfore, this problem calls for real-time planning algorithms in uncertain, partially observable environments, where the state of some objects (e.g. doors) are subject to change over time. This is a typical scenario for many service and healthcare robots operating in indoor home or office-like environments. In a broader sense, this problem is an instance of the problem of decision making and control under uncertainty. However, what makes the problem more difficult is the need for a robust solution that is able to replan in real-time to cope with discrete changes in the environment, failures in sensory system, and large deviations from the nominal plan.

POMDPs: In general, decision making and control under uncertainty is a ubiquitous challenge in many engineering applications and, in particular, in robotics. For an autonomous robot to perform robust and reliably, it is crucial to be able to perceive measurements through its sensors, infer its situation (state) in the environment, and plan and take actions accordingly. However, in partially-observable environments the state of system cannot be determined exactly due to the imperfect and noisy measurements. However, knowing the system and sensors model as well as the probability distributions of the process and sensing noises, a filtering module (such as a Kalman filter) can provide an estimation of the state, i.e., a probability distribution function (pdf) over all possible system states. This pdf is also referred to as *belief* or *information-state*. Accordingly, actions can be taken based on the belief. To formalize this procedure and finding the optimal mapping between perceived observations and the taken action, the Partially-Observable Markov Decision Processes (POMDP) formulation is one of the most general frameworks that formalizes this problem [1], [2].

Challenges in solving POMDPs: There are a number of challenges in dealing with POMDPs, including the *curse of dimensionality* and *curse of history*. Curse of dimensionality refers to the high dimensions of the belief space. If the underlying robotic system evolves in a one dimensional discrete state space with n states, the corresponding belief space is an n-dimensional continuous space. Moreover, if the underlying space is continuous (which is the case for most real robotic applications), then the belief space is an infinite dimensional space. Methods such as [3], [4], [5], [6], [7], [8], alleviate such issues and aims at taking POMDPs to more challenging and realistic problems.

Model Discrepancies and Environment Changes: However, when dealing with real-world physical systems, there is another important challenge: *discrepancy between the real models with the models used for computation*. These discrepancies include discrepancy in the process model (state evolution model), the process noise model (distribution), sensor model, sensing noise model (distribution), and the environment map. In other words, the equation used for modelling the state evolution always is a simplification of system's physics or the distribution used for the process noise never exactly matches the true noise distribution. Such discrepancies can lead to deviations of the system from the desired plan. A plausible solution for this problem is an ability to replan dynamically in real-time as the system encounters such deviations or observes signs of discrepancy during the plan execution. Moreover, as mentioned, real-time replanning can handle discrete changes in the environment and make the system more robust to intermittent sensing failures.

Single-query vs. Multi-query: The main body of POMDP literature, in particular sampling-based methods, propose single-query solvers, i.e., the computed solution depends on the initial belief [9], [10], [11]. Therefore, in replanning (planning from a new initial belief) almost all the computations need to be reproduces, which limits their usage in cases where real-time replanning is needed. In particular, dynamic replanning schemes such as Receding Horizon Control (RHC), where the planning problem needs to be solved frequently from new start points, calls for real-time policy generation which restricts the usage of single-query methods. However, multi-query methods such as Information RoadMaps (IRM) [12], [13] provides a construction mechanism, independent of the initial belief of the system. As a result, they are suitable methods to be used in an RHC framework.

Contributions and highlights: In this paper we study the inclusion of IRM in the inner loop of an RHC planner, propose an IRM based version of RHC for the stochastic setting, and design a real-time replanning scheme in belief space. A primary emphasis of this paper is on the implementation of this belief space planner on a physical robot. Establishing connections between FIRM and rollout policy approach in the control theory, we propose a principled way of real-time replanning in belief space and implement it on a physical robot. In particular, we investigate how such a real-time replanning can generate a feedback plan that is robust to discrepancies between real models and computational models as well as robust to changes in the environment, failures in the sensory system, and large deviations from the nominal plan. We believe these results lay the ground work for further moving the theoretical POMDP framework toward practical applications, and achieving long-term autonomy in robotic systems.

Paper organization: In the next subsection, we first review the related work. Then, we start with describing the application of interest and the system set up we used in our implementations, and a statement of the problem. After a brief review of the underlying FIRM framework (Section III), in Section IV, we discuss how FIRM can be embedded in a rollout policy framework and propose the dynamic replanning framework in belief space. Section V investigates the performance of the belief space planner on a physical robot compared to deterministic planners. In Section VI, we demonstrate real-time replanning and show its behaviour on a physical robot. Finally, we conclude the paper in Section VII.

A. Related Work: Real-time Replanning in Belief Space

Real-time replanning in the belief space is a vital capability in many applications, for two main reasons: *(i)* the belief dynamics are usually much more random than the state dynamics, because the belief is directly affected by the system observation. Therefore, a spurious data association (detecting a wrong feature ID), which is a very common failure in many sensing systems (such as vision or laser range finders), can cause huge changes in the belief. Hence, a real-time replanning is necessary to recover from such failures. *(ii)* Moreover, in practical applications, discrepancies between real models with the models used for computation is a significant source of error and cause the belief to occasionally have behaviours different than its expected nominal behaviour. Again, by being able to replan, the robot can recover from such belief deviations. Besides these two points, real-time replanning can help the robot to cope better with changes in the environment as well as recover from large deviations that may occur in its location.

However, the majority of state-of-the-art methods for planning in belief space are not equipped with real-time replanning capabilities. Sampling methods for solving belief space planning such as Belief RoadMap (BRM) [9], LQG-MP [10], [14], or [15] are single query methods (the solution is valid for a given initial belief). Therefore, in case of replanning from a new belief all (or most) of the computation needs to be reproduced, which restrict their usage in cases where frequent real-time replanning is required. Similarly, point-based methods such as [3], [6], [16], [17] are rooted in a single initial belief.

On the other hand, trajectory optimization-based methods can be used for replanning in a Receding Horizon Control (RHC) scheme. In an RHC scheme (as will be detailed in Section IV), the optimal trajectory is computed within a limited horizon. Then only the first step of the trajectory is executed and the rest of it is discarded. From the new point, then, the optimal trajectory is recomputed and this process is repeated until the system reaches the goal region. The RHC framework is originally designed for deterministic systems and its extension to stochastic systems and belief space planning is still an open problem. A direct approach is to replace the uncertain quantities (such as noises) with their nominal values (e.g., zero, for zero-mean Gaussian noises), and then treat the stochastic system as a deterministic one and use it in a RHC framework. Methods such as [18], [19], [20], [21], and [22] fall into this category and are among the trajectory optimization-based methods that can be/are used in RHC framework, with an additional assumption that the future observations are deterministic. However, in such an approach the optimization is carried out only within limited horizon, and therefore the system may locally choose good actions but after a while find itself in a region with high costs. Moreover, removing the stochasticity of the system state (or belief) may lead to unreliable plans. In this paper, we propose a methods based on the rollout policy and FIRM framework that address these issues. In other words, in the proposed belief space planning method, we consider all possible future (random) observation. Moreover, we pick the FIRM plan as the base policy of the rollout policy method and thus incorporate the cost-to-go beyond the optimization horizon into the planning. We detail this approach in Section IV.

II. Target Application and System Set Up

We start by discussing the application of interest and the system set up used in our implementation. Accordingly, in later sections when we discus the theoretical framework, we can tie it to the physical system on which we are implementing the method.

The goal of the experiments: The ultimate goal of these experiments is to have a belief space planner that can handle the uncertainties associated with a typical low-cost robot in an office-like environment. We use an iRobot Create platform (Figure 2), on which a Dell Latitude laptop with an on-board camera and wireless networking capability is mounted. As will be discussed further below, landmarks are installed in the environment. Robot can get noisy measurements of the relative range and bearing to landmarks. The desired behaviour for the planner is to guide the robot to the goal through the parts of the environment where robot can better localize itself and hence better avoid collisions. However, most importantly, we

need the planner to be able to replan in real-time so that it can cope with deviations resulted from model discrepancies, changes in the environment, large disturbances, and sensor failures.

Scenario: To design and evaluate a planner with the mentioned properties, we consider a scenario, where the robot needs to operate in an office environment. We conduct an experiment where robots need to reach a goal, and each time it reaches a goal, a new goal is submitted by the user. During this long run the robustness of the method is investigated with respect to *(i)* changing obstacles, such as doors, and moving people, *(ii)* changes in the goal location, *(iii)* deviations due to missing information sources, and *(iv)* kidnap situations (significant sudden deviation in robots location). During the run, there are many situations where robot needs to replan: It needs to replan each time a new goal is submitted and moves toward the new goal. Also, the robot encounters changes in the obstacle map. For example it encounters doors that are in a different state than they were supposed to be. Similarly, it encounters moving people. Observing these changes, the robot updates its map and replans in real-time and updates its policy accordingly. Moreover, the robot may be "kidnapped" by a person to an unknown location during the run. Thus, the robot needs to recover from this catastrophic situation. Finally, the robot may deviate from its nominal location due to temporary failures in the sensing system. In all these cases a real-time replanning scheme can help robot to recover from the situation and accomplish toward its goal.

In the rest of this section, we first discuss the environment used for experiments. Then, we discuss the robot model and the sensory system.

A. Environment

The specific environment for conducting experiments is the fourth floor of the Harvey Bum Bright (HRBB) building at the Texas A&M University campus in College Station, TX. A floorplan can be seen in Fig. 1. The floor spans almost 40 meters of hallways whose width are almost 2 meters, which is distinguished in yellow and blue in Fig. 1. The main experiments are conducted in the region which is highlighted in blue in Fig. 1, part of which contains a large cluttered office (room 407). This area has interesting properties that makes the planning more challenging: *(i)* As is seen in Fig. 1, there are several doors in this office (407) which may be opened or closed. Two of these doors (front-door and back-door) are shown in Fig. 1. Besides, *(ii)* there are objects such as trash-cans in this environment which usually get displaced and block some of the landmarks in the environment. This is an instance of missing information sources. Finally, *(iii)* people are moving in this area. Therefore, a reactive behaviour may displace the robot from its planned path, which introduces another challenge for the high level planner.

B. Robot Model

The robot utilized in our experiments is the i-robot create mobile robot (See Fig. 2). The robot can be modelled as a unicycle whose kinematics is as follows:

$$x_{k+1} = f(x_k, u_k, w_k) = \begin{pmatrix} \mathsf{x}_k + (V_k \delta t + n_v \sqrt{\delta t}) \cos\theta_k \\ \mathsf{y}_k + (V_k \delta t + n_v \sqrt{\delta t}) \sin\theta_k \\ \theta_k + \omega_k \delta t + n_\omega \sqrt{\delta t} \end{pmatrix}, \tag{1}$$

where $x_k = (\mathsf{x}_k, \mathsf{y}_k, \theta_k)^T$ describes the robot state, in which $(\mathsf{x}_k, \mathsf{y}_k)^T$ is the 2D position of the robot and θ_k is the heading angle of the robot, at time step k. Control commands are the linear and angular velocities $u_k = (V_k, \omega_k)^T$. We use Player robot interface [23] to send these control commands to robot.

Motion noise: The motion noise vector is denoted by $w_k = (n_v, n_\omega)^T \sim \mathcal{N}(0, \mathbf{Q}_k)$, which mostly arose from uneven tiles on the floor, wheel slippage, and inaccuracy in the length of time control signals need to be applied. Experimentally, we found out that the in addition to the fixed uncertainty associated

Fig. 1. Floorplan of the environment, in which experiments are conducted.

with the control commands there exists a part of noise that is proportional to the signal strength. Thus, we model the variance of process noise at the k-th time step as

$$\mathbf{Q}_k = \begin{pmatrix} (\eta V_k + \sigma_b^V)^2 & 0 \\ 0 & (\eta \omega_k + \sigma_b^\omega)^2 \end{pmatrix} \tag{2}$$

where, in our implementations we have $\eta = 0.03$, $\sigma_b^V = 0.01$m/s, $\sigma_b^\omega = 0.001$rad $= 0.057$deg.

Fig. 2. A picture of robot (i-robot create) in the operating environment. Landmark can be seen on the walls.

Hardware-software interface: The connection between the planner and with the robot hardware is established through the Player robot interface [23].

C. Sensing Model

For sensing purposes, we use the on-board camera existing on the laptop. We perform a vision-based landmark detection based on ArUco (a minimal library for Augmented Reality applications) [24]. Each landmark is a black and white pattern printed on a letter-size paper. The pattern on each landmark follows a slight modification of the Hamming code, and has a unique id, so that it can be detected robustly and uniquely. Landmarks are placed on the walls in the environment (see Fig. 2) at the same height with the camera (robot is moving in a 2D space). The absolute position and orientation of each landmark in the environment is known. The ArUco library performs the detection process and presents the range and bearing relative to each visible landmark along with its id. Therefore, if we denote the j-th landmark position in the 2D global coordinates as ${}^{j}L$, we can model the observation as a range-bearing sensing system:

$$ {}^{j}z_k = [\|{}^{j}\mathbf{d}_k\|, \text{atan2}({}^{j}d_{2_k}, {}^{j}d_{1_k}) - \theta]^T + {}^{j}v, \quad {}^{j}v \sim \mathcal{N}(\mathbf{0}, {}^{j}\mathbf{R}), $$

where ${}^{j}\mathbf{d}_k = [{}^{j}d_{1_k}, {}^{j}d_{2_k}]^T := [\mathtt{x}_k, \mathtt{y}_k]^T - L_j$.

Measurement noise: Random vector ${}^{j}v$ models the measurement noise associated with the measurement of the j-th landmark. Experimentally, we found out that the intensity of measurement noise increases by the distance from the landmark and by the incident angle. Incident angle refers to the angle between the line connecting the camera to landmark and the wall, on which landmark is mounted. Denoting the incident angle by $\phi \in [-\pi/2, \pi/2]$, we model the sensing noise associated with the j-th landmark as a zero mean Gaussian, whose covariance is

$$ {}^{j}\mathbf{R}_k = \begin{pmatrix} (\eta_{r_d}\|{}^{j}\mathbf{d}_k\| + \eta_{r_\phi}|\phi_k| + \sigma_b^r)^2 & 0 \\ 0 & (\eta_{\theta_d}\|{}^{j}\mathbf{d}_k\| + \eta_{\theta_\phi}|\phi_k| + \sigma_b^\theta)^2 \end{pmatrix} \tag{3} $$

where, in our implementations we have $\eta_{r_d} = 0.1$, $\eta_{r_\phi} = 0.01$, $\sigma_b^r = 0.05$m, $\eta_{\theta_d} = 0.001$, $\eta_{\theta_\phi} = 0.01$, and $\sigma_b^\theta = 2.0$deg.

Full vector of measurements: At every step robot observes a subset of landmarks, which fall into its field of view. Suppose at a particular step robot can see r landmarks $\{L_{i_1}, \cdots, L_{i_r}\}$. The concatenation of visible landmarks is the total measurement vector that is denoted by $z = [{}^{i_1}z^T, \cdots, {}^{i_r}z^T]^T$ and due to the independence of measurements of different landmarks, the observation model for all landmarks can be written as $z = h(x) + v$, where $v = [{}^{i_1}v^T, \cdots, {}^{i_r}v^T]^T$. Thus, the full measurement noise vector is drawn from $v \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$, where $\mathbf{R} = \text{diag}({}^{i_1}\mathbf{R}, \cdots, {}^{i_r}\mathbf{R})$.

D. Planning Under Uncertainty

Available data for decision making: In the presence of noises w_k and v_k, the exact value of system state is not known. The only data that is available for decision making at the k-th time step (i.e., generating u_k) is the observations we have got and the controls we have applied up to that time step, i.e., $\mathcal{H}_k = \{z_{0:k}, u_{0:k-1}\} = \{z_0, z_1, \cdots, z_k, u_0, \cdots, u_{k-1}\}$.

State estimation: However, relying on the probability distribution of these noises, a filtering module can provide information about the system state in the form of the probability distribution over all possible system states given the observations and actions, $b_k = p(x_k|\mathcal{H}_k)$, which is referred to as *belief* or *information-state*. It is well-known that in Baysian filtering, belief can be computed recursively merely based on the last action and current observation $b_{k+1} = \tau(b_k, u_k, z_{k+1})$ [25],[26]:

$$ b_{k+1} = \alpha p(z_{k+1}|x_{k+1}) \int_{\mathbb{X}} p(x_{k+1}|x_k, u_k) b_k dx_k =: \tau(b_k, u_k, z_{k+1}). \tag{4} $$

where, $\alpha = p(z_{k+1}|\mathcal{H}_k, u_k)^{-1}$ is the normalization constant. As a result of filtering, the action u_k can be taken based on the belief b_k using a policy (planner) π_k, i.e., $u_k = \pi_k(b_k)$.

One-step-cost: To find the best planner, we need to define the objective of planning. We consider the localization uncertainty, control effort, and elapsed time as different elements constituting our cost function.

$$c(b_k, u_k) = \zeta_p P_k + \zeta_u \|u_k\| + \zeta_T(1). \tag{5}$$

where P is the estimation covariance as a measure of localization uncertainty, i.e., $P_k = \int_{\mathbb{X}} x^2 b_k dx - (\int_{\mathbb{X}} x b_k dx)^2$. Norm of control signal $\|u_k\|$ denotes the control effort, and the (1) is present in the cost to model the 1-step time elapse. Coefficients ζ_p, ζ_u, and ζ_T are user-defined task-dependent scalars to combine these costs toward the desirable behaviour. In the presence of constraints (such as obstacles in the environment), we first assume that the task fails if the robot violates these constraints (e.g., collides with obstacles). Therefore, in case of failure, the running-sum of costs (cost-to-go), i.e., $J(F) = \sum_{t'}^{\infty} c(b, u)$ is set to a suitably high cost-to-go.

POMDP problem and challenges: The problem of planning under uncertainty is defined as finding a sequence of policies $\pi_{0:\infty} = \{\pi_0(\cdot), \pi_1(\cdot), \pi_2(\cdot), \cdots\}$, where π_k maps any given b_k to the optimal action u_k. Stationary policy π_s is a special case in this notation, where $\pi_0 = \pi_1 = \pi_2 = \cdots = \pi_s$. This problem can be formally defined as:

$$\pi_{0:\infty} = \arg\min_{\Pi_{0:\infty}} \mathbb{E} \sum_{k=0}^{\infty} c(b_k, u_k) = \arg\min_{\Pi_{0:\infty}} \mathbb{E} \sum_{k=0}^{\infty} c(b_k, \pi_k(b_k)) \tag{6}$$

where, Π_k is the space of all possible policies at time step k, i.e., $\pi_k \in \Pi_k$. Note that the expectation operator $\mathbb{E}$ is present in the formulation since the belief evolves randomly (due to the randomness of the observation) and thus the cost is random. The problem in (6) is referred to as the POMDP (Partially-Observable Markov Decision Process) problem or more precisely belief-MDP (belief-Markov Decision Process). Solving the belief-MDP problem is a computationally intractable procedure, in particular over the continuous state, action, and observation spaces.

Solving POMDPs in real-time: The more difficult problem is to solve the POMDP problem in real-time as needed to cope with changes in the models or map (the case in our application). In the next section, we briefly review the Information RoadMap (IRM) to reduce POMDPs to a tractable problem, and later in Section IV, we discuss how IRM can be exploited to make it possible to re-solve POMDPs in real-time.

III. FIRM Overview

In this section, we briefly review the Information RoadMap (IRM) framework [12], [13]. We also discuss the concrete realization of the FIRM constructed for conducting the experiments.

Information RoadMaps (IRM): An Information RoadMap is a framework to reduce the intractable belief-MDP problem to a tractable belief-MDP problem over a representative graph in the belief space. IRM generates a graph in the belief space, and accordingly it reduces the policy space Π to a representative set of policies. Inspired by Probabilistic Roadmap Methods (PRMs) where the *solution path* is a concatenation of local paths, in Information RoadMaps (IRM) the *solution policy* is a concatenation of local policies. Each node in IRM is a small region in the belief space. We denote i-th node by $B^i \subset \mathbb{B}$. Each edge in IRM is a local controller. In FIRM (Feedback-based IRM), each edge (local controller) is a feedback controller, whose goal is to drive the belief into the target node of the edge. We denote the edge (controller) between nodes i and j by μ^{ij}.

Having such a graph in belief space, we can form the belief-MDP on the IRM graph:

$$\pi^g_{0:\infty} = \arg\min_{\Pi^g_{0:\infty}} \mathbb{E} \sum_{n=0}^{\infty} C^g(B_n, \mu_n)$$
$$= \arg\min_{\Pi^g_{0:\infty}} \mathbb{E} \sum_{n=0}^{\infty} C^g(B_n, \pi^g(B_n)) \tag{7}$$

where, B_n is the n-th FIRM node we visit during the plan execution, and μ_n is the local controller taken at B_n. $C^g(B, \mu)$ is the generalized cost, which is defined as the cost of taking the local controller μ at node B. In the following we discuss all these elements detailed for our specific implementation.

Stabilizers: To construct a FIRM graph, we first need to sample a set of stabilizers. Each stabilizers is a feedback controller whose role is to drive the belief into a FIRM node. It is a well known that a controller in a partially-observable environment (where our problem resides) consists of a filter and a separated controller [27]. Filter governs the belief evolution and the separated-controller generates the control signal based on the available belief at each time step [27]. In the following, we discuss the filter and the separated controller we use in the current implementations. In this work, to sample stabilizers, we first sample a set of points $\mathcal{V} = \{\mathbf{v}^i\}$ in the problems state space and then associated with each point we construct a stabilizer.

Filter of stabilizer: We use Kalman filter for localization (estimating the robot's state), which takes into account both uncertainty in robot's motion and uncertainty in sensory measurements. Consider the stabilizer associated with the sampled point $\mathbf{v}$. Since during stabilization the system is around the target point and aims at reaching the target point $\mathbf{v}$, we adopt the Stationary Kalman Filter (which is constructed by linearizing the system about the target point $\mathbf{v}$) as the stabilizer's filter. Let us denote the system linearized at the target point $\mathbf{v}$ by:

$$x_{k+1} = Ax_k + Bu_k + Gw_k, \quad w_k \sim \mathcal{N}(0, Q), \tag{8}$$
$$z_k = Hx_k + Mv_k, \quad v_k \sim \mathcal{N}(0, R), \tag{9}$$

Then, denoting Gaussian beliefs with their mean and covariance $b_k \equiv (\widehat{x}^+_k, P^+_k)$, the filtering equation in (4) during the stabilization process reduces to:

$$b_{k+1} \equiv \begin{bmatrix} \widehat{x}^+_{k+1} \\ P^+_{k+1} \end{bmatrix} = \begin{bmatrix} A\widehat{x}^+_k + Bu_k + K_{k+1}(z_{k+1} - H(A\widehat{x}^+_k + Bu_k)) \\ (I - K_{k+1}H)(AP^+_k A^T + GQG^T) \end{bmatrix} = \begin{bmatrix} \tau_1(\widehat{x}^+_k, u_k, z_{k+1}) \\ \tau_2(P^+_k) \end{bmatrix} \equiv \tau(b_k, u_k, z_{k+1}), \tag{10}$$

where K_k is called the Kalman gain at the k-th time step and is computed as follows:

$$K_{k+1} = (AP^+_k A^T + GQG^T)H^T(H(AP^+_k A^T + GQG^T)H^T + MRM^T)^{-1}.$$

An important observation one can make from Eq. (10) is that if the (A, H) pair is an observable pair [25], the difference equation evolving the covariance matrix, i.e. $P^+_{k+1} = \tau_2(P^+_k)$, has a fixed point. In other words, the covariance converges to a stationary covariance P^+_s that can be obtained through solving $P^+_s = \tau_2(P^+_s)$, which is indeed a Discrete Algabraic Riccati Equation (DARE), whose solution can be computed efficiently [28]. It is important to note that the stationary covariance P^+_s does not depend on the choice of the separated-controller as long as the separated controller can keep the system close enough to the target node such that the linearized model in Eq. (8) is valid. However, if due to a large noise or any other reason system goes far from the target point, the dynamic replanning will take care of the deviation as will be discussed in later sections.

Separated-controller of stabilizer: A separated-controller μ is responsible to generate the control signal based on the available belief, i.e., $u_k = \mu(b_k)$. The i-robot create is a nonholonomic robot and is modelled as a unicycle (see Section II-B). Since the covariance is already approaching to its stationary value, The

separated-controller needs to only act on the mean value and drives it toward the target point $\mathbf{v}$. Therefore, we can use a variety of controllers designed for stabilizing nonholonomic systems, e.g., [29], [30], and [31]. However, it is worth noting that the mean value gets affected by random observations (see τ_1 in Eq. (10)), and thus it is impossible to take it to an exact point in the belief space; However, defining a ball around the target point $\mathbf{v}$, the controller can take the mean value into this ball in a finite time (note that the observation noise has a zero mean). As a result, the adopted controller needs to perform well under such uncertainties, which limits our choices. In our experiments, we implemented different controllers such as polar coordinate-based controllers [32], or Dynamic Feedback Linearization-based controller [29], [33] and observed their behaviours under uncertainty. However, the best results we got was through a goal-seeking controller. The implemented goal-seeking controller computes a sequence of open-loop controls that can drive the system from the current mean value to the target node in the absence of noise. Then, we apply a truncated version of this control sequence (first five controls in our implementations - only one control is not enough due to the nonholonomicity of the system). After applying the truncated sequence of controls, the controller recomputes the open-loop control sequence and repeats this procedure (similar to RHC) until the mean value reaches the target region.

FIRM nodes: Since the covariance matrix approaches the stationary covariance P_s^+ and the mean value reaches to a region around the target point $\mathbf{v}$, we can define the belief $\grave{b} \equiv (\mathbf{v}, P_s^+)$, whose neighborhood is reachable under the constructed stabilizer. Therefore, we define the corresponding FIRM node as:

$$B = \{b : \|b - \grave{b}\| \leq \epsilon\} \tag{11}$$

where, $\|\cdot\|$ denotes a suitable norm on the belief space and ϵ defines the node size. Characterizing a FIRM node for each sampled node $\mathbf{v}^i$, we get a set of FIRM nodes $\{B^i\}$. Algorithm 1 summarizes the steps in sampling FIRM nodes.

Algorithm 1: Sample_FIRM_Node

input : A sample in state space $\mathbf{v}$
output : A FIRM node B
Construct (sample) the stabilizer (both filter $\tau(\cdot;\mathbf{v})$ and separated-controller $\mu(\cdot;\mathbf{v})$) associated with the state sample $\mathbf{v}$;
Solve $P_s^+ = \tau_2(P_s^+)$ to compute stationary covariance;
Construct belief sample $\grave{b} \equiv (\mathbf{v}, P_s^+)$;
Construct FIRM node $B = \{b : \|b - \grave{b}\| \leq \epsilon\}$;
return B;

FIRM Edges: FIRM edges are feedback controllers that take belief from a FIRM node to another FIRM node. If nodes are dense, nodes in the vicinity of the node $\mathbf{v}$ will fall into the basin of attraction of its associated stabilizer. Therefore, for a dense graph stabilizers are indeed the graph edges. However, for a sparse graph, a stabilizer may not be sufficient to take the belief from a node to its neighboring node. In such a cases, we precede the stabilizer with a feedback controller that is designed to track a nominal trajectory that connects underlying nodes. In our implementations, we use time-varying LQG controller to track a nominal trajectory (with a finite length N) that connects, say, $\mathbf{v}^i$ to $\mathbf{v}^j$, and when it finishes tracking (i.e., after N steps), we switch to stabilizer associated with $\mathbf{v}^j$ to drive the belief to the FIRM node B^j. We denote the concatenation of the (i,j)-th edge controller and the j-th stabilizer as μ^{ij} and call it (i,j)-th local controller or *(i,j)-th FIRM edge*. Algorithm 2 summarizes the steps in FIRM node connection phase.

Generalized Costs and Transition Probabilities: To find the costs associated with FIRM edges, we add up one-step costs along the edge (i.e., during the execution of the local controller). In other words, the

Algorithm 2: Connect_FIRM_Nodes

input : Two FIRM nodes B^i, B^j
output : FIRM edge μ^{ij}
Retrieve underlying PRM nodes $\mathbf{v}^i, \mathbf{v}^j$ and Sample PRM nodes $\mathcal{V} = \{\mathbf{v}^j\}_{j=1}^{N_v}$ and construct its edges $\mathcal{E} = \{\mathbf{e}^{ij}\}$;
Design the edge controller (time-varying LQG) $\overline{\mu}_k^{ij}$ along the edge $\mathbf{e}^{ij}$;
Construct the local controller μ^{ij} by concatenating edge controller $\overline{\mu}_k^{ij}$ and node controller μ_s^j;
Set $b_0 = b_c^i$;
Generate sample belief paths $b_{0:\mathcal{T}}$ and ground truth paths $x_{0:\mathcal{T}}$ induced by controller μ^{ij} invoked at B^i;
Compute the transition probabilities $\mathbb{P}^g(F|B^i, \mu^{ij})$ and $\mathbb{P}^g(B^j|B^i, \mu^{ij})$ and transition cost $C^g(B^i, \mu^{ij})$;

cost of (i, j)-th edge is defined as:

$$C^g(B^i, \mu^{ij}) = \mathbb{E} \sum_{k=1}^{\mathcal{T}^{ij}} c(b_k, \mu^{ij}(b_k)) \tag{12}$$

To compute this expectation and to find the transition probabilities associated with FIRM edges, we run the controller on an ensemble of particles. For the (i, j)-th edge, we sample M beliefs $\{b_0^m\}_{m=1}^M$ from the FIRM node B^i and run the controller on them. The average of particles' cost approximates the expectation in (12), and the number of successful particles divided by the total number of particles M approximates the success probability along the edge $\mathbb{P}(B^j|B^i, \mu^{ij})$. Algorithm 3 summarizes the FIRM graph generation phase.

Algorithm 3: Construct_FIRM_Graph

input : Two FIRM nodes B^i, B^j
output : FIRM graph $\mathcal{G}$
Generate a PRM in the state space with nodes $\mathcal{V} = \{\mathbf{v}_j\}_{j=1}^N$ and edges $\mathcal{E} = \{e^{ij}\}$;
$\mathbb{V} = \emptyset$;
$\mathbb{M} = \emptyset$;
forall the $\mathbf{v}_i \in \mathcal{V}$ **do**
    B^i =Sample_FIRM_node($\mathbf{v}_i$);
    $\mathbb{V} \leftarrow \mathbb{V} \cup \{B^i\}$;
forall the $e^{ij} \in \mathcal{E}$ **do**
    μ^{ij} =Connect_FIRM(e^{ij});
    $\mathbb{M} \leftarrow \mathbb{M} \cup \{\mu^{ij}\}$;
Construct $\mathbb{M} = \{\mu^{ij}\}$;
return $\mathcal{G} = \{\mathbb{V}, \mathbb{M}\}$;

Graph feedback: Constructing FIRM nodes $\{B^i\}$, FIRM edges $\{\mu^{ij}\}$, edge costs $C^g(B^i, \mu^{ij})$, and edge probabilities $\mathbb{P}(B^i, \mu^{ij})$, we can formulate a dynamic programming on the FIRM graph to solve the FIRM MDP defined in Eq. (7). Let us denote the optimal cost-to-go on the graph as $J^g(B_0) = \min_{\Pi^g} \mathbb{E} \sum_{n=0}^{\infty} C^g(B_n, \pi^g(B_n))$. Also, let us treat the failure event (collision with obstacles or any event that leads to not reaching to a FIRM node under the local controller) as reaching to a hypothetical FIRM

node B^0. Therefore, the failure probability on each edge is denoted by $\mathbb{P}(B^0|B^i,\mu^{ij})$ as follows:

$$J^g(B^i) = \min_{\mu}\{C^g(B^i,\mu) + \sum_{\gamma=0}^{N} \mathbb{P}(B^\gamma|B^i,\mu)J^g(B^\gamma)\} \tag{13}$$

where, the failure cost-to-go, i.e., $J^g(B^0)$ is set to a suitably high cost-to-go.

Algorithm 4: (Re)plan_from

1 **input** : Start belief b_0, Graph Cost-to-go $J^g(\cdot)$, FIRM nodes $\mathbb{V} = \{B^i\}$, Success probabilities $P^{success}(\cdot)$
2 **output** : Next Local Controller μ^*
3 Find r neighboring nodes $\mathfrak{N} = \{B^i\}_{i=1}^r$ to b_0;
4 Set $J^*(B) = \infty$;
5 **for** $B \in \mathfrak{N}$ **do**
6 Construct local planner μ from b_0 to B;
7 Compute the transition cost $C(b_0,\mu)$ and probability $\mathbb{P}(B|b_0,\mu)$;
8 **if** $C(b_0,\mu) + \mathbb{P}(B|b_0,\mu)J(B) + (1-\mathbb{P}(B|b_0,\mu))J(F) < J^*(B)$ **then**
9 $J^*(B) = C(b_0,\mu) + \mathbb{P}(B|b_0,\mu)J(B) + (1-\mathbb{P}(B|b_0,\mu))J(F)$;
10 $\mu^* = \mu$;
11 **return** μ^*;

Remark: If the desired factor is the success probability only, one can initialize the success probability from b_0 to 0; i.e., $P^*(b_0)$ before the `for` loop in Algorithm 4. Then, the one can replace the condition in line 7 of Algorithm 4 by $\mathbb{P}(B|b_0,\mu)P^{success}(B) > P^*(b_0)$ and add the condition update statement $P^*(b_0) = \mathbb{P}(B|b_0,\mu)P^{success}(B)$ into the `for` loop.

Algorithm 5: (Re)plan_to

1 **input** : Goal state $\mathbf{v}^{goal}$, FIRM Graph $\mathcal{G} = \{\mathbb{V},\mathbb{M}\}$
2 **output** : FIRM feedback π^g
3 B^{goal} = Sample_FIRM_node($\mathbf{v}^{goal}$);
4 Add B^{goal} to the FIRM graph; i.e., $\mathbb{V} \leftarrow \mathbb{V} \cup \{B^{goal}\}$;
5 Connect B^{goal} to its r nearest neighbors usign edges $\{\mu^{(i,goal)}\}$. Also, $\mathbb{M} \leftarrow \mathbb{M} \cup \{\mu^{(i,goal)}\}$;
6 Compute the cost-to-go J^g and feedback π^g over the FIRM nodes by solving the MDP in Eq. (14);
7 **return** π^g;

IV. Dynamic Replanning in Belief Space

In this section, we discuss how we can perform dynamic replanning in belief space utilizing Information RoadMaps (IRM). The dynamic replanning mechanism is based on the Limited Receding Horizon Control (RHC) and Rollout Policy (ROP) [25] framework.

A. Dynamic Replanning for Fully-Observable Systems

In this section, we aim to design a principled scheme for real-time replanning in the belief space that can cope with large deviations and changes in the environment map.

First, recall that in controlling uncertain systems, the goal is to find a sequence of policies $\pi_{0:\infty}(\cdot) = \{\pi_1(\cdot), \pi_2(\cdot), \pi_3(\cdot), \cdots\}$. (A stationary policy π_s is a special case where we have $\pi_1 = \pi_2 = \cdots = \pi_s$.) Therefore, the original problem of stochastic control with perfect state information is defined as:

Problem 1. ***(MDP)*** *The problem of stochastic control with perfect state information or the Markov Decision Process (MDP) problem is defined as the following optimization over the policy space:*

$$\begin{aligned} \pi_{0:\infty}(\cdot) &= \arg\min_{\Pi_{0:\infty}} \sum_{k=0}^{\infty} \mathbb{E}\left[c(x_k, \pi_k(x_k))\right] \\ s.t. \quad & x_{k+1} = f(x_k, \pi_k(x_k), w_k), \quad w_k \sim p(w_k | x_k, \pi_k(x_k)) \end{aligned} \tag{14}$$

RHC for fully observable stochastic systems: Receding horizon control (often referred to as rollinghorizon or modelpredictive control) was originally designed for deterministic systems (to cope with model discrepancy). For stochastic systems, where the closed-loop (feedback) control law is needed, formulation of the RHC scheme is an open problem [34], [35], [36], [20]. In the most common form of RHC [25] the stochastic system is approximated with a deterministic system by replacing the uncertain quantities with their typical values (e.g., maximum likelihood value.) For the zero-mean Gaussian noises, the maximum likelihood value for noise is zero. Therefore, for planning purposes, the approximated system is $x_{k+1} = f(x_k, u_k, 0)$. At every time step, the RHC scheme performs a two-stage computation. At the first stage, the RHC scheme for deterministic systems solves an open-loop control problem (i.e., returns a sequence of actions $u_{0:T}$) over a fixed finite horizon T as follows:

$$\begin{aligned} u_{0:T} &= \arg\min_{\mathbb{U}_{0:T}} \sum_{k=0}^{T} c(x_k, u_k) \\ s.t. \quad & x_{k+1} = f(x_k, u_k, 0). \end{aligned} \tag{15}$$

In the second stage, it executes only the first action u_0 and discards the rest of actions in the sequence $u_{0:T}$. However, due to the model discrepancy (f is not exact), system state may end up in a different location than its nominal trajectory. Subsequently, RHC performs these two stages from the new point (state is assumed to be observable in deterministic setting). In other words, RHC computes an open loop sequence $u_{0:T}$ from this new state, and this process continues until the state reaches the goal location. Following algorithms recaps this procedure.

Algorithm 6: RHC for fully-observable stochastic systems

```
input : Initial state x_current ∈ 𝕏, 𝒳_goal ⊂ 𝕏
while x_current ∉ 𝒳_goal do
    u_{0:T} = Solve optimization in Eq.(15) starting from x_0 = x_current;
    Apply the action u_0 to the system;
    Observe and update the current state x_current;
```

Issues with RHC: There are some obvious issues and pitfalls regarding the presented RHC framework: *(i)* First, due to the limited horizon and ignoring the cost-to-go beyond the horizon, the method may get stuck into pitfalls by choosing actions that guides the robot toward "favorable" states (with low cost) in the near future proceeding with a set of "unfavorable" states (with a high cost) in a long run. Also, *(ii)*, the presented form of RHC ignores the stochasticity of the system within horizon, which may lead to inaccurate approximation of the cost and unreliable control actions. To overcome these issues, researchers have proposed variants of RHC and different frameworks, such as Rollout policy, based on the idea of repeated planning [25].

Rollout policy for observable stochastic systems: Another class of methods to reduce the complexity of the MDP problem in Eq.(14) is the class of Rollout policies [25], which are more powerful methods than the described version of RHC for two reasons: *(i)* They search for a sequence of policies (instead of open-loop controls) within the horizon, and do not approximate the system with a deterministic one. *(ii)* Moreover, they use a suboptimal policy, called the based policy, to compute a cost-to-go function $\widetilde{J}$ that approximates the true cost-to-go beyond the horizon. In other words, at each step of rollout policy scheme, the following closed-loop optimization is solved:

$$\begin{aligned} \pi_{0:T}(\cdot) = \arg\min_{\Pi_{0:T}} \mathbb{E}\left[\sum_{k=0}^{T} c(x_k, \pi_k(x_k)) + \widetilde{J}(x_{T+1})\right] \\ s.t. \quad x_{k+1} = f(x_k, \pi_k(x_k), w_k), \quad w_k \sim p(w_k|x_k, \pi_k(x_k)) \end{aligned} \tag{16}$$

Then, only the first feedback control law π_0 is used to generate the control signal u_0 and the rest of policies are discarded. Similar to the RHC, after applying the first control, a new sequence of policies is computed from the new point. The rollout algorithm is detailed as follows:

Algorithm 7: Rollout algorithm for fully observable stochastic systems:

input : Initial state $x_{current} \in \mathbb{X}$, $\mathcal{X}_{goal} \subset \mathbb{X}$
while $x_{current} \notin \mathcal{X}_{goal}$ **do**
  $\pi_{0:T}$ = Solve optimization in Eq.(16) starting from $x_0 = x_{current}$;
  Apply the action $u_0 = \pi(x_0)$ to the system;
  Observe and update the current state $x_{current}$;

Although rollout policy efficiently reduces the computational cost compared to the original MDP problem, it is still formidable to solve since the optimization is carried out over the policy space (such an optimization is exceedingly difficult, in particular over continuous space, control, and observation spaces). Moreover there should be a base policy that provides a reasonable cost-to-go $\widetilde{J}$. These problems are more difficult in the belief space as we will discuss in the next subsection. Then, we provide an "IRM-based rollout policy scheme" that can be applied to the perfect information case as well.

B. Dynamic Replanning for Partially-Observable Systems

In this section, we discuss the extension of the RHC and Rollout policy to the belief space to design a principled scheme for real-time replanning in the belief space that can cope with large deviations and changes in the environment map.

First, recall that in controlling uncertain systems with perfect state information, the goal is to find a sequence of policies $\pi_{0:\infty}(\cdot) = \{\pi_1(\cdot), \pi_2(\cdot), \pi_3(\cdot), \cdots\}$. (A stationary policy π_s is a special case where we have $\pi_1 = \pi_2 = \cdots = \pi_s$.) Therefore, the original problem of stochastic control with imperfect state information is defined as:

Problem 2. ***(POMDP)*** *The problem of stochastic control with imperfect state information or the Partially-Observable Markov Decision Process (POMDP) problem is defined as the following optimization over the policy space:*

$$\begin{aligned} \pi_{0:\infty}(\cdot) = \arg\min_{\Pi_{0:\infty}} \sum_{k=0}^{\infty} \mathbb{E}\left[c(b_k, \pi_k(b_k))\right] \\ s.t. \quad b_{k+1} = \tau(b_k, \pi_k(b_k), z_k), \quad z_k \sim p(z_k|x_k) \\ x_{k+1} = f(x_k, \pi_k(b_k), w_k), \quad w_k \sim p(w_k|x_k, \pi_k(b_k)) \end{aligned} \tag{17}$$

RHC in belief space: Receding horizon control (often referred to as rollinghorizon or modelpredictive control) was originally designed for deterministic systems (to cope with model discrepancy). For stochastic systems, where the closed-loop (feedback) control law is needed, formulation of the RHC scheme is an open problem [34], [35], [36], [20]. In the most common form of RHC [25] the stochastic system is approximated with a deterministic system by replacing the uncertain quantities with their typical values (e.g., maximum likelihood value.) In belief space planning the quantity that injects randomness in belief dynamics is the observation. Thus, one can replace the random observations z_k with their deterministic maximum likelihood value z^{ml}, where $z_k^{ml} := \arg\max_z p(z_k|x_k^d)$ in which x^d is the nominal deterministic value for the state that results from replacing the motion noise w by zero, i.e., $x_{k+1}^d = f(x_k^d, \pi_k(b_k^d), 0)$. The deterministic belief b^d is then used for planning in the receding horizon window. At every time step, the RHC scheme performs a two-stage computation. At the first stage, the RHC scheme for deterministic systems solves an open-loop control problem (i.e., returns a sequence of actions $u_{0:T}$) over a fixed finite horizon T as follows:

$$\begin{aligned} u_{0:T} = \arg\min_{\mathbb{U}_{0:T}} & \sum_{k=0}^{T} c(b_k^d, u_k) \\ s.t. \quad & b_{k+1}^d = \tau(b_k^d, u_k, z_{k+1}^{ml}) \\ & z_{k+1}^{ml} = \arg\max_z p(z|x_{k+1}^d) \\ & x_{k+1}^d = f(x_k^d, u_k, 0) \end{aligned} \tag{18}$$

In the second stage, it executes only the first action u_0 and discards the rest of actions in the sequence $u_{0:T}$. However, since the actual observation is noisy and is not equal to the z^{ml}, the the belief b_{k+1} will be different that b_{k+1}^d. Subsequently, RHC performs these two stages from the new belief b_{k+1}. In other words, RHC computes an open loop sequence $u_{0:T}$ from this new belief, and this process continues until the belief reaches the desired belief location. Following algorithms recaps this procedure.

Algorithm 8: RHC for Partially-observable stochastic systems

input : Initial belief $b_{current} \in \mathbb{X}$, $B_{goal} \subset \mathbb{B}$
while $b_{current} \notin B_{goal}$ **do**
  $u_{0:T}$ = Solve the optimization in Eq.(18) starting from $b_0^d = b_{current}$;
  Apply the action u_0 to the system;
  Observe the actual z;
  Compute the belief $b_{current} \leftarrow \tau(b_{current}, u_0, z)$;

State-of-the-art methods such as [22] and [37] utilize the RHC-in belief space. This framework is also called Partially-closed loop RHC (PCLRHC) [22] since it partially exploits some information about the future observations (i.e., z^{ml}) and does not fully ignore them.

Issues with RHC: There are some issues regarding the presented RHC framework: *(i)* First, due to the limited horizon and ignoring the cost-to-go beyond the horizon, the method may get stuck into pitfalls by choosing actions that guides the robot toward "favorable" states (with low cost) in the near future followed by a set of "unfavorable" states (with a high cost) in the long run. Second, *(ii)* the presented form of RHC ignores the stochasticity of the system within horizon, which may lead to inaccurate approximation of the cost and unreliable control actions. To overcome these issues, researchers have proposed variants of RHC and different frameworks, such as the "rollout policy", based on the idea of repeated planning [25].

Rollout policy in belief space: Another class of methods that aim to reduce the complexity of the stochastic planning problem in Eq.(14) is the class of rollout policies [25], which are more powerful than the described version of RHC in the following senses: *(i)* They search for a sequence of policies (instead of open-loop controls) within the horizon, and do not approximate the system with a deterministic one. *(ii)* They use a suboptimal policy, called the "base policy", to compute a cost-to-go function $\widetilde{J}$ that approximates the true cost-to-go beyond the horizon. In other words, at each step of the rollout policy scheme, the following closed-loop optimization is solved:

$$\begin{aligned} \pi_{0:T}(\cdot) = \arg\min_{\Pi_{0:T}} \mathbb{E}\left[\sum_{k=0}^{T} c(b_k, \pi_k(b_k)) + \widetilde{J}(b_{T+1})\right] \\ s.t. \quad b_{k+1} = \tau(b_k, \pi_k(b_k), z_k), \quad z_k \sim p(z_k|x_k) \\ x_{k+1} = f(x_k, \pi_k(b_k), w_k), \quad w_k \sim p(w_k|x_k, \pi_k(b_k)) \end{aligned} \tag{19}$$

Then, only the first control law π_0 is used to generate the control signal u_0 and the rest of policies are discarded. Similar to the RHC, after applying the first control, a new sequence of policies is computed from the new point. Rollout algorithm is detailed as follows:

Algorithm 9: Rollout algorithm in Belief Space:

input : Initial belief $b_{current} \in \mathbb{B}$, $B_{goal} \subset \mathbb{B}$
while $b_{current} \notin B_{goal}$ **do**
  $\pi_{0:T}$ = Solve optimization in Eq.(19) starting from $b_0 = b_{current}$;
  Apply the action $u_0 = \pi(b_0)$ to the system;
  Observe the actual z;
  Compute the belief $b_{current} \leftarrow \tau(b_{current}, u_0, z)$;

Although the rollout policy in the belief space efficiently reduces the computational cost compared to the original POMDP problem, it is still formidable to solve since the optimization is carried out over the policy space. Moreover there should be a base policy that provides a reasonable cost-to-go $\widetilde{J}$. In the following, we propose a rollout policy in the belief space based on the IRM-based cost-to-go.

IRM-based Rollout Policy: In IRM-based rollout policy, we adopt the IRM policy as the base policy of the rollout algorithm. Accordingly, the cost-to-go of the IRM policy will be used as the cost-to-go beyond the horizon. Now, if we have a dense FIRM graph such that FIRM nodes partition the belief space, i.e., $\cup_i B^i = \mathbb{B}$, then at the end of horizon, the belief b_{T+1} belongs to an IRM node B, from which the IRM cost-to-go is available. However, in practice, when the IRM nodes cannot cover the entire belief space, we need to make sure that the truncated policy can drive the belief into an IRM node at the end of horizon. However, since the belief evolution is random, we may not be able guarantee the belief reaches an IRM node at the end of a deterministic horizon T. Therefore, instead of truncating the policy over time, we truncate the policy over the belief and leave the horizon length to be random (denoted by $\mathcal{T}$) as follows:

$$\begin{aligned} \pi_{0:\infty}(\cdot) = \arg\min_{\widetilde{\Pi}} \mathbb{E}\left[\sum_{k=0}^{\mathcal{T}} c(b_k, \pi_k(b_k)) + \widetilde{J}(b_{\mathcal{T}+1})\right] \\ s.t. \quad b_{k+1} = \tau(b_k, \pi_k(b_k), z_k), \quad z_k \sim p(z_k|x_k) \\ x_{k+1} = f(x_k, \pi_k(b_k), w_k), \quad w_k \sim p(w_k|x_k, \pi_k(b_k)) \\ b_{\mathcal{T}+1} \in \cup_j B^j, \end{aligned} \tag{20}$$

where for $b_{\mathcal{T}+1} \in B^j$ we have

$$\widetilde{J}(b_{\mathcal{T}+1}) = J^g(B^j) \tag{21}$$

where $\widetilde{\Pi}$ is a restricted set of policies under which the belief will reach an IRM node B^j in finite time (possibly random). More rigorously, if $\pi \in \widetilde{\Pi}$ and $\pi = \{\pi_1, \pi_2, \cdots\}$, then for finite $\mathcal{T}$, we have $\mathbb{P}(b_{\mathcal{T}+1} \in \cup_j B^j | \pi) = 1$, i.e., belief will enter into a FIRM node under π after finite time. In other words, the last constrained in (20) is redundant as it is already satisfied by the definition of $\widetilde{\Pi}$. However, it is explicitly written in (20) to emphasize this constraint. Also, it is worth noting that the IRM-based cost-to-go $J^g(\cdot)$ plays the role of the cost-to-go beyond the horizon $\widetilde{J}(\cdot)$ (Equation (21)).

Therefore, in solving IRM-based rollout policy problem, we aim at finding a sequence of policies that ends up in a FIRM node and minimizes the cost in (20). To find this optimal policy, we parametrize the policy space $\widetilde{\Pi}$ and perform the minimization over the parameter space. For our particular system, the policy is considered to be a concatenation of a tracker μ_k (a controller to track a nominal trajectory) and a stabilizer μ^s (a controller that stabilize the belief to a fixed belief):

$$\pi(\cdot; \{x^d_{0:n}, \mathbf{v}\}) = \{\mu_k(\cdot; x^d_{0:n}), \mu^s(\cdot; \mathbf{v})\} \tag{22}$$

The tracker is parametrized by a nominal deterministic trajectory $x^d_{0:n}$. The role of tracker is to drive the belief to the vicinity of a belief node (IRM node). The stabilizer is parametrized by a nominal state $\mathbf{v}$. The role of stabilizer is to drive the belief into the IRM node.

The particular tracker and stabilizer we have adopted in our implementation are the same as tracker and stabilizer used along the edges (described in Section III). If the current belief is $b_0 = (\widehat{x}^+_0, P_0)$ we fix $x^d_0 = \widehat{x}^+_0$. Then, for each IRM node B^j, we retrieve the corresponding state node $\mathbf{v}^j$ (based on $\grave{b}^j = (\mathbf{v}^j, P^j)$ (see Section III)) and fix $x^d_n = \mathbf{v}^j$. Accordingly, aim to find the optimal $\mathbf{v}^j$ and intermediate deterministic states and controls $x^d_{0:n}, u^d_{0:n}$ that satisfy system equations, $x^d_{k+1} = f(x^d_k, u^d_k, 0)$.

V. Experimental Results on Planning with PRM and FIRM

In this section, we discuss the results of PRM and FIRM-based motion planning on a low-cost i-robot create equipped with a laptop. The integrated web-camera (monocular) is used to observe the landmarks. The goal of this section is to show how FIRM, as a belief space planner, guides the robot through regions with less collision probability, and more information, to better localize the robot.

The environment is shown in Fig. 3. Blue regions are obstacles and black regions are free space. Landmarks are shown by small white diamonds. The start and goal locations for the motion planning problem are marked in Fig. 3. As can be seen, the goal location is inside room 407 (see Fig. 1) and the start is close to the front door. In the following, we compare the performance (success probability) of the MAPRM (Medial-Axis PRM), a conventional configuration space planne,r with the performance of FIRM.

MAPRM-based planning: As one of the best variants of PRM, when it comes to the collision avoidance, we construct Medial-Axis PRM (MAPRM) in this environment [38]. The resulting PRM is shown in Fig. 3. As can be seen in Fig. 3, there exists a homotopy class of paths through the front door of the room 407 as well as a homotopy class of paths through the back door of the room, between the start and goal nodes. From Fig. 3, it is obvious that the path through the front door is shorter. Moreover, the path through the front door has a larger obstacle clearance (larger minimum distance from obstacles along the path) compared to the path through the back door (since the back door is half open). Therefore, based on conventional metrics in deterministic settings, such as shortest path or maximum clearance, MAPRM chooses the path through the front door over the path through the back door. The feedback tree that results from solving DP in this case is shown in Fig. 4. As expected, the DP guides the robot to go through the front door.

MAPRM path execution: To execute the plan generated from PRM, we use time-varying LQG controllers to keep the robot close to the nominal path (solution of the PRM-based planning). However, due to the lack of enough information along the nominal path, the success rate of this plan is low, and the robot frequently collides with obstacles along the path as the robot is prone to drift. The success probability along the nominal path is computed by a Monte Carlo simulation (100 runs) and is equal to 27% (27 runs out of 100 runs were successful).

FIRM-based planning: As can be seen in Fig. 3, the distribution of information is not uniform in the environment. The density of landmarks (information sources) along the path through the back door is more than the landmarks along the path going through the front door. Incorporating the information distribution in the environment in planning leads to a better judgement of the narrowness of passages. For example in this experiment, the path through the front door seems to be shorter than the path through the back door. However, considering the information sources the success probability of traversing through the back door is more than the success probability of traversing through the front door. Such a knowledge about the environment is reflected in the FIRM cost-to-go and success probability in a principled rigorous framework. As a result, it generates a policy that suits the application, taking into account the uncertainty, and available information in the environment. Solving a DP problem on the FIRM graph gives a feedback as shown in Fig. 5, which results in an 88% success probability.

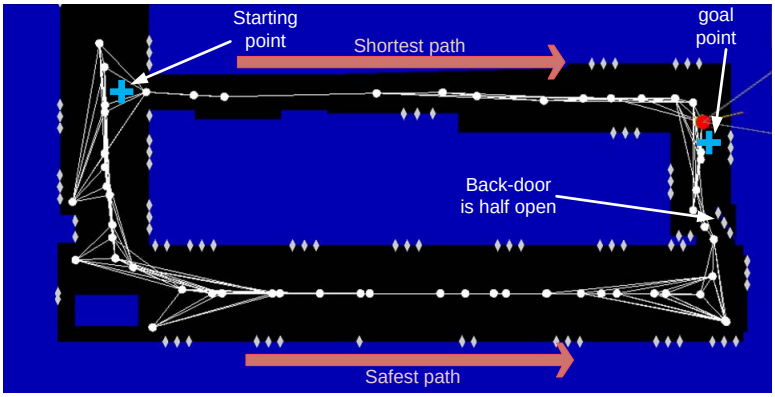


Fig. 3. The environment including obstacles (blue regions), free space (black region), and landmark (white diamonds) are shown in this figure. An MAPRM graph approximating the connectivity of free space is also shown. The start and goal points (for the experiments in this section) are distinguished by crosses in the light blue.

VI. Robustness Experiments

In this section, we examine and discuss the robustness properties of the FIRM-based RHC framework for stochastic systems. We first look into the case where the obstacle map is subject to change. Then, we investigate the robustness to missing information sources and large deviations. Finally, we consider repeated changes in the goal location, along with the types of changes mentioned above and demonstrate the method's performance on a more complex scenario.

A. Robustness to Changes in the environment: obstacles and information sources

In this section, we investigate the robustness of the proposed algorithm to the changes in the obstacles' map. In this experiment we demonstrate the robustness of the proposed belief space planner to changes in the environment, implementing the method on a physical robot. In this sense, it takes an important step toward making belief space planners a practical tool in robotics. Moreover, in this experiment, we also show the robustness of the method to missing information sources (landmarks).

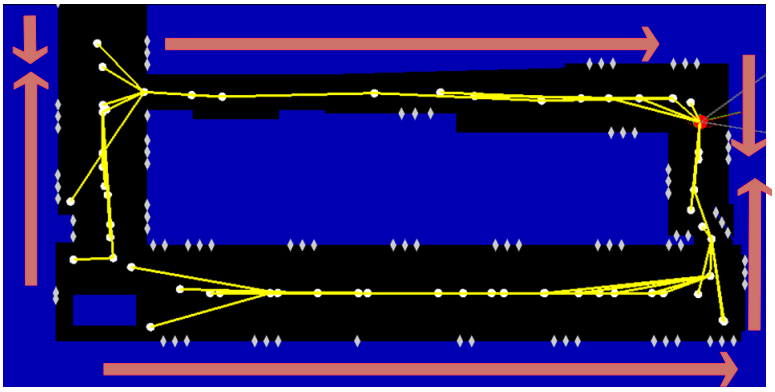


Fig. 4. The feedback tree generated by solving DP on MAPRM is shown in yellow. From each node there is only one outgoing edge (in yellow), computed by DP, guiding the robot toward the goal (See Fig. 3). Arrows in pink coarsely represent the direction on which the feedback guides the robot.

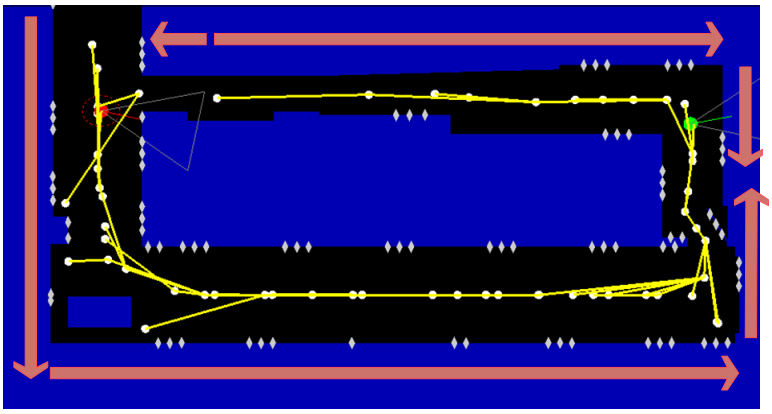


Fig. 5. The feedback tree generated by solving DP on FIRM is shown. As can be seen the computed feedback guides robots through the more informative regions that leads to more accurate localization and less collision probabilities. Arrows in pink coarsely represent the direction on which the feedback guides the robot.

Type of obstacles and map changes: In our experiments, we consider three types of obstacles. The first set of obstacles (most of the map) are static obstacles such as walls. The second class of obstacles are the ones that discretely change their state such as doors in the environment (open and closed state). The last class of objects are people standing or moving in the environment. It is worth noting that dealing with a fully dynamic environment is not a goal of this paper. To handle moving obstacles (people) we assume there exists a reactive planner at a lower level that suppresses the belief space planner in the vicinity of obstacles. Accordingly, after moving away from the moving obstacle, the robot may be deviated from its nominal plan and thus the belief space planner has to replan to recover from such deviations. The main focus of the following experiments is to demonstrate how our method can replan in real-time when encountered with changes in the environment map. Obviously, dealing with agile obstacles or fully dynamic environment requires development of the more sophisticated reactive planners and is beyond the scope of this paper.

Scenario: As the first experiment, we consider the environment shown in Fig. 1. The start and goal locations are shown in Fig. 6(a). We construct a PRM in the environment ignoring the changing obstacles (assuming all doors are open and there are no people in the environment). Leveraging PRM to FIRM and solving dynamic programming on it, we get the feedback tree shown in Fig. 6(a) that guides the robot

to go through the back door of room 407. However, the challenge is that the door may be closed when robot reaches it, and there may be people moving in the environment. Moreover, for different reasons (such as blur in the image or blocked landmarks by people or different objects), we may miss detecting landmarks temporarily during the run.

Obstacle detection: We assume that the robot is equipped with a sensor that detects the obstacles in the vicinity of the robot. Such a perception can be performed by a Laser Range Finder (LRF). However, designing the perception module is not a concern of this paper, and since our robot is not equipped with an LRF, we use a simple method in our experiments to detect objects. We stick a small marker with a specific ID on moving objects (doors or people shoes). When the robot observes these landmarks, it realizes that there is an obstacle in the vicinity of the robot. To handle such a change in the obstacle map and replan accordingly, we use the "lazy feedback evaluation" algorithm outlined below.

Lazy feedback evaluation: To adapt the proposed framework to the changing environment, we rely on lazy evaluation methods. Inspired by the lazy evaluation methods for PRM frameworks [39], we propose a variant of the lazy evaluation methods for evaluating the generated feedback tree. The basic idea is that at every node the robot re-evaluates *only* the next edge (or the next few edges up to a fixed horizon) that robot needs to take. By re-evaluation, we mean it needs to re-compute the collision probabilities along these edges. If there is a significant change in the collision probabilities, the dynamic programming is re-solved and a new feedback tree is computed. Otherwise, the feedback tree remains unchanged and the robot keeps following it. Such a lazy evaluation (computing the collision probabilities for a single edge or a small number of edges) can be performed in real-time. The method is detailed in Algorithm 10.

Algorithm 10: Lazy Feedback Evaluation (Lazy Replanning)

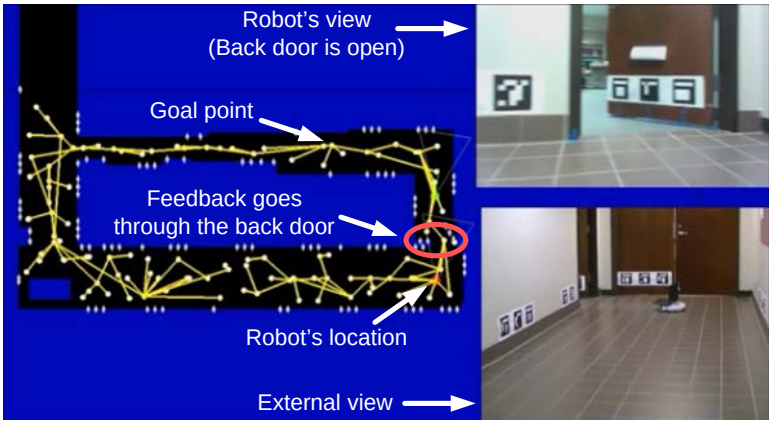
input : Feedback tree π^g, current belief $b_{current}$
output : Updated feedback tree, π^g
Update the obstacles map;
if *there is a change in map* **then**
    $\mathcal{F} \leftarrow$ Retrieve the sequence of nominal edges returned by feedback up to horizon l;
    forall the *edges* $\mu \in \mathcal{F}$ **do**
        Re-compute the collision probabilities $\mathbb{P}_{new}(B, \mu)$ from the start node B of edge;
    if *exists* $\mu \in \mathcal{F}$ *such that* $|\mathbb{P}_{new}(B, \mu) - \mathbb{P}(B, \mu)| > \alpha$ **then**
        $\mathbb{P}(B, \mu) \leftarrow \mathbb{P}_{new}(B, \mu)$;
        $\pi^g \leftarrow$ Replan($b_{current}$);
return π^g;

forgetting time: Imagine a case where the robot is in a room with two doors. Suppose after checking both doors, the robot realizes they are closed. In such cases to persuade the robot to recheck the state of doors, we reset the door state to "open" after a specific amount of time as if the robot forgets that the door was "closed". In our experiments, the forgetting time for doors is 10 minutes, and the forgetting time for other moving obstacles is about 10 seconds.

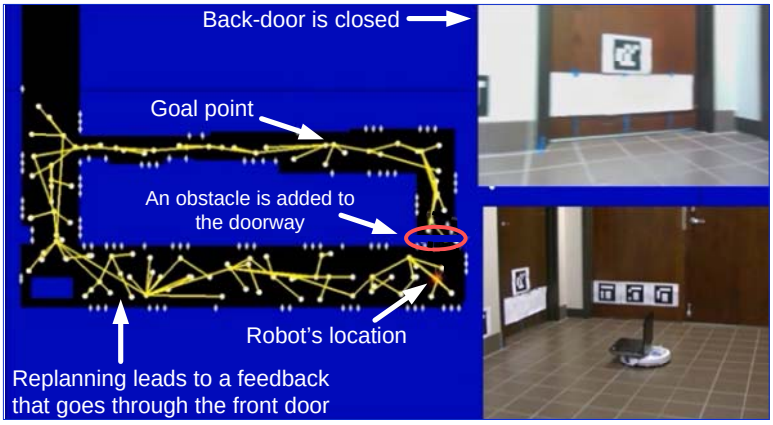
Results on physical robots: Figure 6(b) shows a snapshot of our run when the robot detects the change signal, i.e., detects the door is in a different situation than its recorded situation in the map. As a result robot updates the obstacle map as can be seen in Fig. 6(b) (Door is closed). Accordingly, robot replans; Figure 6(b) shows the feedback tree resulted from replanning. As can be seen, the new feedback guides the robot through the front door since it detects the back door is closed. The full video of this run provides much more detail and is available in Appendix I.

Comparison with state-of-the-art: It is important to note that it is the graph structure of FIRM that

makes such a replanning feasible in real-time. The graph structure of FIRM allows us to *locally* change the collision probabilities in the environment without affecting the collision probability of the rest of the graph (i.e., properties of different edges on the graph are independent of each other). It is important to note that such a property is not present in any other state-of-the-art belief space planner, such as BRM (Belief Roadmap Method) [9], or LQG-MP [10]. In those methods, collision probabilities and costs on *all* edges (number of possible edges is exponential in the size of underlying PRM) need to be re-computed.



(a)



(b)

Fig. 6. (a) The back door is open at this snapshot. The feedback guides the robot toward goal through the back door. (b) The back door is closed at this snapshot. Robot detects the door is closed and updates the obstacle map (adds door). Accordingly robot replans and computes the new feedback. The new feedback guides the robot through the front door.

Missing landmarks: We may also miss detecting some information sources. For example, people may block the landmarks temporarily. Also, in our physical experiments, we observed that the rugged parts of floor, which leads to a lot of jitter in robot's motion, can make the captured images blurry. Thus, in those regions we may miss some landmarks intermittently (this is an example of discrepancy in computational models and physical models). Also, we constantly encountered objects (such as trash cans) that have been moved and block some of the landmarks. This phenomenon can also be a common issue for service robots.

Replanning criteria: Experimentally, we found out that the effect of missing information sources in the environment usually manifest in two ways: *(i)* an increase in stabilization time, or *(ii)* a deviation from the underlying nominal PRM edge. Therefore, we check both these conditions at each step and if either of them is satisfied, we use the replanning algorithm from the current belief. The current belief

(initial belief for replanning) usually have a larger uncertainty due to the missing information sources, and thus replanning can take into account this growth in the uncertainty.

B. Robustness to large deviations

In this subsection, we investigate the robustness of the proposed framework in dealing with large deviations in robots position. As a more general form of this problem, we consider the *kidnapped robot problem*.

Kidnapped robot problem: "In robotics, the kidnapped robot problem commonly refers to a situation where an autonomous robot in operation is carried to an arbitrary location." [40]. "This problem is commonly used to test a robot's ability to recover from catastrophic localization failures" [40]. This problem introduces different challenges such as *(i)* how to detect kidnapping, *(ii)* how to localize the robot, and *(iii)* how to control the robot to accomplish its goal. Our main focus, here, is on the third part, i.e., how to replan in belief space from the new point in the belief space after recovering from being kidnapped.

Detecting kidnapped situation: To detect the kidnapped situation, we constantly monitor the innovation signal $\widetilde{z}_k = z_k - z_k^-$ (the different between the actual observations and predicted observation). To define the specific measure of innovation, we use in our implementation, recall that the observation at time step k from the j-th landmark is the relative range and bearing of the robot to the j-th landmark, i.e., ${}^j z_k = ({}^j r_k, {}^j \theta_k)$. The predicted version of this measurement is shown by ${}^j z_k^- = ({}^j r_k^-, {}^j \theta_k^-)$. We monitor the following measures of the innovation signal:

$$\widetilde{r}_k = \max_j(|{}^j r_k - {}^j r_k^-|), \quad \widetilde{\theta}_k = \max_j(d^\theta({}^j \theta_k, {}^j \theta_k^-)) \tag{23}$$

where $d^\theta(\theta, \theta')$ returns the absolute value of the smallest angle that maps θ onto θ'. Passing these signal through a low-pass filter, we filter out the outliers (temporary failures in the sensory reading). Denoting the filtered signals by $\overline{r}_k$ and $\overline{\theta}_k$, we monitor the conditions $\overline{r}_k < r_{max}$ and $\overline{\theta}_k < \theta_{max}$. If both of them are satisfied, we follow the FIRM feedback (i.e., we are in the *Feedback Following Mode* (FFM)). However, violation of any of these conditions means that the robot is constantly observing high innovations, and thus it is not in the location that it was supposed to be (i.e., it is kidnapped). Further below we show the innovation signal for a sample run on a physical robot. In our implementations, we consider $r_{max} = 1$ (meters) and $\theta_{max} = 50$ (degrees).

Information Gathering Mode (IGM): Once it is detected that the robot is kidnapped, we first replace the estimation covariance with a large covariance (to get an approximately uniform distribution over the state space). Then, we enter the Information Gathering Mode (IGM), where we take very small and conservative steps (e.g., turning in place or taking random actions with small velocities) to get some known measurements. Once the robot gets a few measurements, the localization module corrects the estimation value and innovation signal reduces. When conditions $\overline{r}_k < r_{max}$ and $\overline{\theta}_k < \theta_{max}$ are satisfied again, we quit the information gathering mode.

Post-IGM replanning: After recovering from being kidnapped, controlling the robot in belief space is a significant challenge as the system can be far from where it was supposed to be. However, using FIRM, the robot just needs to go to a neighboring node from this new point. Since the FIRM graph is spread in the belief space, there is no need for costly replanning procedure. Indeed, the only required computation is to evaluate the cost of edges that connect the new start point to the neighboring FIRM nodes.

Results on physical robots: Figure 7 shows a snapshot of a run that contains two kidnapping and illustrates the robustness of the planning algorithm to the kidnapping situation. The start and goal positions are shown in Fig. 7. Feedback tree (shown in yellow) guides the robot toward the goal through the front door. However, before reaching the goal point robot gets kidnapped in the hallway (cf. Fig. 7) and placed it in an unknown location within the 407 office (cf. Fig. 7). The first jump in 8 shows this deviation.

Once the robot recovers from being kidnapped (i.e., when both innovation signals in Fig. 8 fall below their corresponding thresholds), a replanning from the new point is performed. This time feedback guides the robot toward the goal point from withing room 407. However, again before robot reaches the goal point, it is kidnapped and placed in an unknown location (see Fig. 7). The second jump in the innovation signals in Fig. 8 corresponds to this kidnapping.

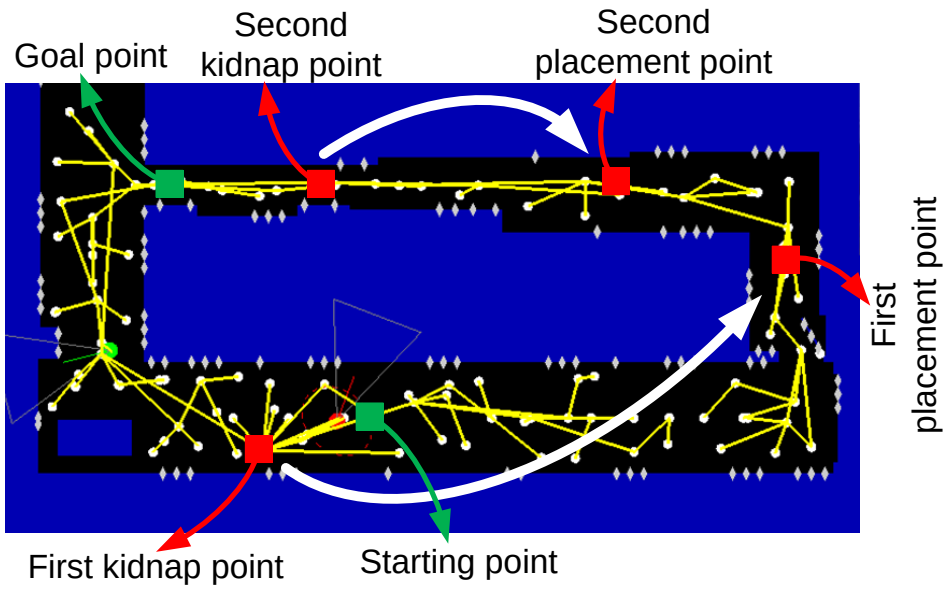


Fig. 7. This figure shows the set up for the experiment containing two kidnapping.

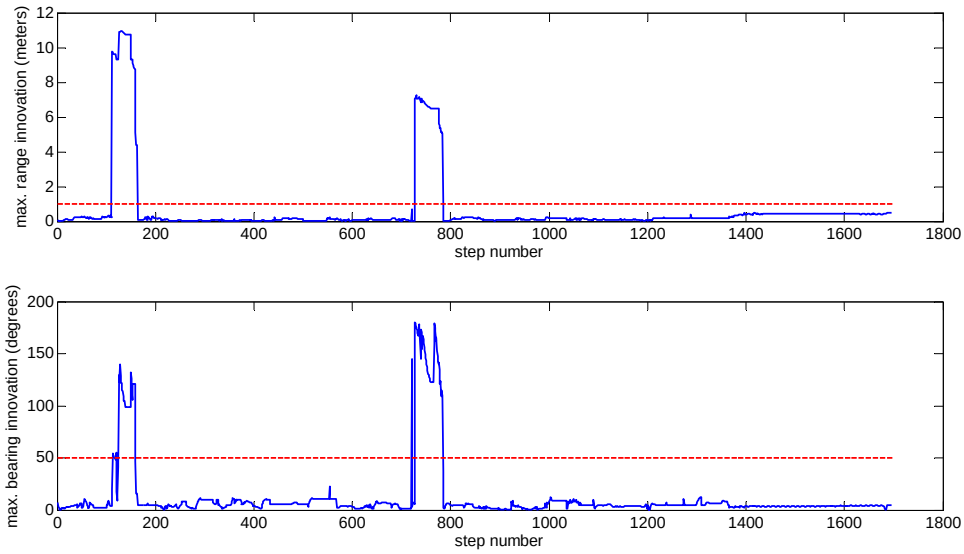


Fig. 8. This figure shows the innovation signals $\widehat{r}_k$ and $\widehat{\theta}_k$ during this run. When both of the signals are below their specified thresholds r_{max} and θ_{max} (dashed red lines), robot follows the FIRM feedback. Otherwise, the system enters the information gathering mode.

C. A Longer and more complex experiment: Robustness to changing goals, obstacles, and landmarks and to large deviations

In this section, we emphasize the ability of the system to perform long-term tasks consists of visiting several goals. The replanning ability allows us to change the plan in real-time as the goal location changes. In this experiment, we consider a scenario in which users submit a new goal for robot to reach after it reaches its currently assigned goal. The robot needs to visit these goals, while it frequently encounters changes in the obstacle map (open/closed doors and moving people) as well as missing information and kidnapped robot situations. Thus, the robot needs to perform a lot of real-time replannings in belief space to cope with such frequent changes. The video (as detailed in Appendix I) shows robots performance in this long and complex scenario.

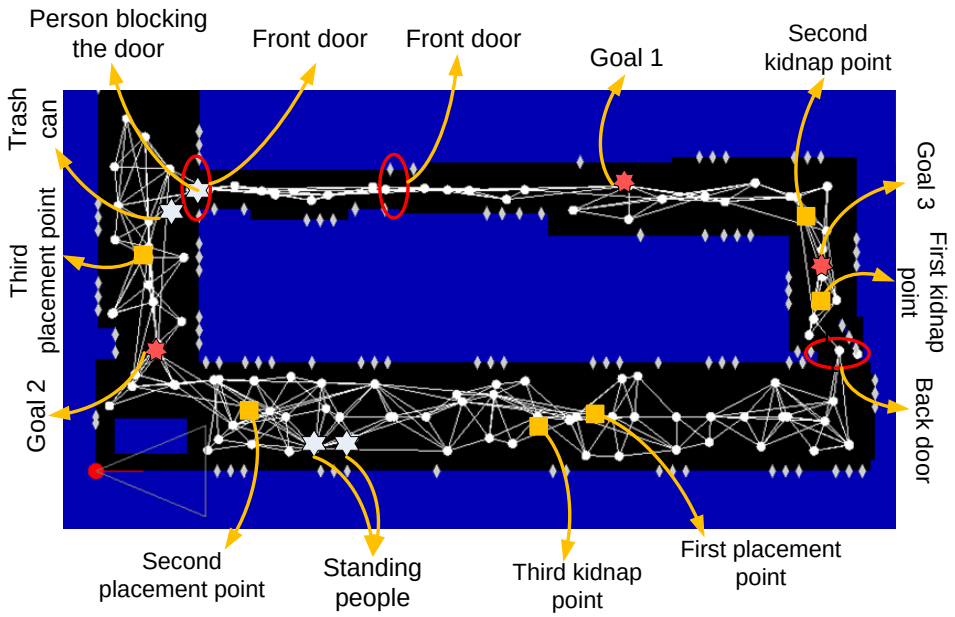


Fig. 9. This figure shows the set up for the longer experiment with a sequence of goals as well as intermediate events and changes in the environment map.

In the following, we provide an itemized description of the specific steps involved in this run based on Fig. 9. Also, we discuss different changes in the environment that robot needs to cope with, along the way to accomplishing its goals. All of the following steps can be seen more clearly in the accompanying video (see Appendix I).

1) Robot starts at the starting point shown in Fig. 9 and aims to reach goal 1 as shown in Fig. 9. Goal 1 is inside the room 407. FIRM returns a feedback tree that guides the robot through the back door of 407 (cf. Fig. 9).
2) Robot goes through the narrow passage introduced by the back door (it is half-open). However, before reaching the goal it gets kidnapped (the first kidnap point as shown in Fig. 9). The robot is placed in an unknown location (shown in Fig. 9 by first placement point.)
3) Observing new landmarks the robot detects that it has been kidnapped. Accordingly it adds a new node to graph and replans in real-time. As a result, the feedback guides the robot toward the goal point through the back door again.
4) However, in the meantime the back door has closed and when the robot reaches the vicinity of the back door, it detects that the door is closed. Therefore, it updates its map by closing the door (i.e.

putting an obstacle at the doorway). Note that the robot will open the door (remove the obstacle) in its map after the forgetting time of 10 minutes. Accordingly, robot replans a feedback tree that guides the robot through the front door toward the goal point.

5) Along the way people are moving in the hallway and inside the 407 office. Thus, robot replans accordingly in real-time as it encounters the people. Moving people are ignored but the standing people and static obstacles such as trash can (see Fig. 9) temporarily get added to the map as obstacles. Replanning several times to cope with such changes, the robot goes through the front and inner doors and reaches the goal point inside the 407 office.
6) After reaching the goal point, another goal (second goal in Fig. 9) is assigned to the robot.
7) Replanning for reaching this goal leads to a feedback tree that guides the robot through the inner door, and front door, toward goal 2.
8) However, as the robot reaches the vicinity of the inner door, it detects the door has been closed. Therefore, it updates its map and replans accordingly. The replanning leads to a feedback tree that guides the robot toward goal 2 through the back door. Again, along the way robot encounters moving people in the office 407 and in the hallway.
9) However, before reaching the goal point, robot gets kidnapped at the "second kidnap point" as shown in Fig. 9. The robot is placed at a really far-off point (the "second placement point"). Once the robot detects it is kidnapped, it replans and moves slower to gather information. Detecting landmarks, it reduces its uncertainty and continues going toward the goal point.
10) After reaching the goal point, the next goal (i.e., third goal) is assigned to the robot (see Fig. 9). Replanning for this goal, leads to a feedback that guides the robot through the front door.
11) However, when robot reaches the front door, it encounters a person standing in the doorway. Accordingly, it replans and decides to go through the back door.
12) On the way to the back door, it is again displaced at the "third kidnap point" and placed at the "third placement point".
13) This time due to the forgetting time, the replanning leads to a path through the front door (person is not there anymore).
14) Again robot follows the feedback and achieves its goal.

VII. Conclusion

This paper proposes a robust method for planning in belief space that can efficiently replan in real-time. Such replanning is a key ability in coping with *(i)* discrepancies between real world models and computational models, *(ii)* changes in the environment and obstacles, *(iii)* large deviations, and *(iv)* changes in the information sources. Implementing a belief space planner on physical robots and demonstrating the robustness to such discrepancies that occur in practice, this method takes an important step in making POMDP methods applicable to real world robotic applications.

References

[1] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable markov processes over a finite horizon," *Operations Research*, vol. 21, no. 5, pp. 1071–1088, 1973.

[2] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, pp. 99–134, 1998.

[3] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *International Joint Conference on Artificial Intelligence*, 2003, pp. 1025–1032.

[4] T. Smith and R. Simmons, "Point-based pomdp algorithms: Improved analysis and implementation," in *Proceedings of Uncertainty in Artificial Intelligence*, 2005.

[5] M. Spaan and N. Vlassis, "Perseus: Randomized point-based vallue iteration for pomdps," *Journal of Artificial Intelligence Research*, vol. 24, pp. 195–220, 2005.

[6] H. Kurniawati, D. Hsu, and W. Lee, "SARSOP: Efficient point-based pomdp planning by approximating optimally reachable belief spaces," in *Proceedings of Robotics: Science and Systems*, 2008.

[7] H. Kurniawati, Y. Du, D. Hsu, and W. S. Lee, "Motion planning under uncertainty for robotic tasks with long time horizons," *The International Journal of Robotics Research*, vol. 30, no. 3, pp. 308–323, 2011.

[8] D. Grady, M. Moll, and L. E. Kavraki, "Automated model approximation for robotic navigation with POMDPs," in *ICRA*, 2013.

[9] S. Prentice and N. Roy, "The belief roadmap: Efficient planning in belief space by factoring the covariance," *International Journal of Robotics Research*, vol. 28, no. 11-12, October 2009.

[10] J. van den Berg, P. Abbeel, and K. Goldberg, "LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 895–913, 2011.

[11] H. Kurniawati, Y. Du, D. Hsu, and W. S. Lee, "Motion planning under uncertainty for robotic tasks with long time horizons," *International Journal of Robotics Research*, vol. 30, pp. 308–323, 2010.

[12] A. Agha-mohammadi, S. Chakravorty, and N. Amato, "FIRM: Sampling-based feedback motion planning under motion uncertainty and imperfect measurements," *International Journal of Robotics Research*, To appear.

[13] ——, "FIRM: Feedback controller-based Information-state RoadMap -a framework for motion planning under uncertainty-," in *International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[14] A. Bry and N. Roy, "Rapidly-exploring random belief trees for motion planning under uncertainty." in *ICRA*, 2011, pp. 723–730.

[15] H. Kurniawati, T. Bandyopadhyay, and N. Patrikalakis, "Global motion planning under uncertain motion, sensing, and environment map," *Autonomous Robots*, pp. 1–18, 2012.

[16] H. Bai, D. Hsu, W. S. Lee, and V. A. Ngo, "Monte carlo value iteration for continuous-state pomdps." in *WAFR*, ser. Springer Tracts in Advanced Robotics, vol. 68. Springer, 2010, pp. 175–191.

[17] P. Chaudhari, S. Karaman, D. Hsu, and E. Frazzoli, "Sampling-based algorithms for continuous-time pomdps," in *the American Control Conference (ACC)*, Washington DC, 2013.

[18] T. Erez and W. D. Smart, "A scalable method for solving high-dimensional continuous pomdps using local approximation," in *the International Conference on Uncertainty in Artificial Intelligence*, 2010.

[19] R. Platt, R. Tedrake, L. Kaelbling, and T. Lozano-Perez, "Belief space planning assuming maximum likelihood observatoins," in *Proceedings of Robotics: Science and Systems (RSS)*, June 2010.

[20] S. Chakravorty and R. S. Erwin, "Information space receding horizon control," in *IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning (ADPRL)*, April 2011.

[21] R. He, E. Brunskill, and N. Roy, "Efficient planning under uncertainty with macro-actions," *Journal of Artificial Intelligence Research*, vol. 40, pp. 523–570, February 2011.

[22] N. D. Toit and J. W. Burdick, "Robotic motion planning in dynamic, cluttered, uncertain environments," in *ICRA*, May 2010.

[23] B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th international conference on advanced robotics*, vol. 1, 2003, pp. 317–323.

[24] R. Munoz-Salinas, "http://sourceforge.net/projects/aruco/."

[25] D. Bertsekas, *Dynamic Programming and Optimal Control: 3rd Ed.* Athena Scientific, 2007.

[26] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

[27] P. R. Kumar and P. P. Varaiya, *Stochastic Systems: Estimation, Identification, and Adaptive Control*. Englewood Cliffs, NJ: Prentice-Hall, 1986.

[28] W. F. Arnold III and A. J. Laub, "Generalized eigenproblem algorithms and software for algebraic Riccati equations," *Proceedings of the IEEE*, vol. 72, no. 12, pp. 1746–1754, 1984.

[29] G. Oriolo, A. De Luca, and M. Vandittelli, "WMR control via dynamic feedback linearization: design, implementation, and experimental validation," *IEEE Transactions on Control Systems Technology*, vol. 10, no. 6, pp. 835–851, 2002.

[30] R. M. Murray and S. S. Sastry, "Nonholonomic motion planning: Steering using sinusoids," *IEEE Transactions on Automatic Control*, vol. 38, no. 5, pp. 700–716, 1993.

[31] C. Samson and K. Ait-Abderrahim, "Feedback control of a nonholonomic wheeled cart in cartesian space," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*. IEEE, 1991, pp. 1136–1141.

[32] A. De Luca, G. Oriolo, and M. Vendittelli, "Control of wheeled mobile robots: An experimental overview," in *Ramsete*. Springer, 2001, pp. 181–226.

[33] A. Agha-mohammadi, S. Chakravorty, and N. Amato, "Nonholonomic motion planning in belief space via dynamic feedback linearization-based FIRM," in *International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[34] D. Li, F. Qian, and P. Fu, "Variance minimization approach for a class of dual control problems," *IEEE Trans. Aut. Control*, vol. 47, no. 12, pp. 2010–2020, 2002.

[35] D. van Hessem and O. Bosgra, "A full solution to the constrained stochastic closed-loop MPC problem via state and innovations feedback and its receding horizon implementation," in *Proceedings of the the Conference on Decision and Control (CDC)*, Maui, HI, December 2003, pp. 929–934.

[36] S. K. Shah, C. D. Pahlajani, N. A. Lacock, and H. G. Tanner, "Stochastic receding horizon control for robots with probabilistic state constraints," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[37] R. Platt, "Convex receding horizon control in non-gaussian belief space."

[38] S. A. Wilmarth, N. M. Amato, and P. F. Stiller, "MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space," in *IEEE International Conference on Robotics and Automation*, vol. 2, 1999, pp. 1024–1031.
[39] R. Bohlin and L. E. Kavraki, "Path planning using lazy prm," in *IEEE International Conference on Robotics and Automation*, vol. 1. IEEE, 2000, pp. 521–528.
[40] Wikipedia, "http://en.wikipedia.org/wiki/kidnapped_robot_problem," 2012.

APPENDIX I
INDEX TO MULTIMEDIA EXTENSION

The multimedia extensions to this article can be found on-line by following the hyperlinks in Table I.

TABLE I
INDEX TO MULTIMEDIA EXTENSION

Extension	Media type	Description	link
1	video	*Narrow passage:* Result video for executing the FIRM plan, corresponding to the environment in Fig.3	link
2	video	*Longer complex experiment:* Result video for a long operation of the robot with FIRM, which shows the robustness of the method to changes in the goal location, changes in doors, moving people, missing landmarks, and large disturbances.	link