Path Planning under Malicious Injections and Removals of Perceived Obstacles: a Probabilistic Programming Approach

Jacopo Banfi[1*], Yizhou Zhang[2*], G. Edward Suh[1], Andrew C. Myers[1], and Mark Campbell[1]

***Abstract*—An autonomous mobile robot may encounter adversarial environments in which an attacker tries to influence its decisions. Through physical or software-level attacks, some of the robot's sensors might be compromised—a special concern for self-driving vehicles. Motivated by this scenario, this letter introduces and studies the problem of planning kinematically feasible (and possibly efficient) paths with bounded collision probability in adversarial settings where the obstacles perceived online by the robot display two layers of uncertainty. The first is the "usual" Gaussian uncertainty one would obtain from a standard object tracker (e.g., an Extended Kalman Filter); the second is an additional layer of uncertainty that captures possible sensor attacks and describes the actual existence of groups of obstacles in the environment. We study the complexity of the problem and propose a general sampling-based solution framework that uses the Sequential Probability Ratio Test (SPRT) to check collision probability constraints along the computed trajectory. We also show how probabilistic programming languages (PPLs) can simplify programming common algorithms (such as RRT and Hybrid A*) for mixed uncertainty. In addition to providing an easy-to-use programming framework, our approach is shown to plan safer paths compared to a Naive Monte Carlo baseline when both approaches are allowed to use at most the same given number of samples to perform collision checks.**

***Index Terms*—Motion and Path Planning; Robot Safety; Control Architectures and Programming**

I. Introduction

SINCE the dawn of mobile robotics, much research has aimed to increase robot autonomy. Classically, it is assumed that all the sensors the robot has at its disposal are not malicious, and that the only source of uncertainty for the robot is random noise in the sensor readings. Motivated by safety issues arising in autonomous vehicles applications—above all, self-driving cars—recent research has shown that an attacker might easily tamper with the robots' perception pipeline [1]. For example, attackers might make the robot collide with obstacles by setting all its lidar readings to very large values. Or they might trap the robot at its current spot by surrounding it with fake obstacles.

A principled planning method is needed to allow mobile robots to behave safely despite such adversarial compromise.

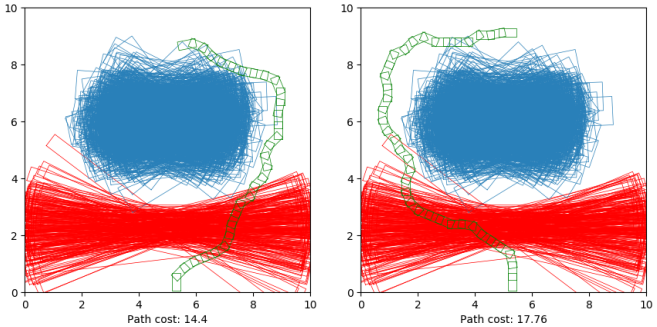
Manuscript received April 9, 2020; accepted August 15, 2020.

This letter was recommended for publication by Associate Editor Jyh-Ming Lien and Editor Nancy Amato upon evaluation of the Reviewers' comments. This work was supported by NASA under Grant NNX16AB09G.

[1]Cornell University, Ithaca NY, USA (e-mail: jb2639@cornell.edu; suh@ece.cornell.edu; andru@cs.cornell.edu; mc288@cornell.edu).

[2]Harvard University, Cambridge MA, USA and University of Waterloo, Waterloo ON, Canada (e-mail: yizhou.zhang@uwaterloo.ca).

*Equal contribution.



Example RRT path Example Hybrid A* path

Fig. 1: Map containing two possible scenarios shown with the corresponding samples: the red obstacle exists with $p = 0.15$, the blue one with $p = 0.85$. Gaussian uncertainty is also present. The proposed sampling-based planning framework is able to plan paths willing to pass through the red obstacle to reach the goal.

Motivated by this scenario, this letter introduces and studies the problem of planning kinematically feasible, reasonably efficient paths with bounded collision probability in adversarial settings where the obstacles perceived online by the robot display two layers of uncertainty. The first is the "usual" Gaussian uncertainty one would obtain from a standard object tracker (e.g., an Extended Kalman Filter [2]); the second is an additional layer of uncertainty describing the actual existence of groups of obstacles in the environment. In this initial work, all the obstacles are assumed to be static.

This letter contains a number of novel contributions. First, we formalize this new path planning problem and show that even the simplest path-planning task (planning a safe feasible path for a holonomic robot on a 4-connected grid) becomes NP-complete. Then, we propose a general sampling-based solution framework that uses the Sequential Probability Ratio Test (SPRT) [3] to check satisfaction of collision probability constraints. Finally, we show how probabilistic programming languages (PPLs) can help in dealing with such resolution approach. In particular we show how, by leveraging a customized PPL, it is possible to present variants of common planning algorithms—here we consider Rapidly Exploring Random Trees (RRTs) [4] and Hybrid A* [5]—that are not only effective in these contexts but can be written with minimal programmer effort. In a nutshell, our framework allows

programmers to implement their favorite planning algorithms as if these algorithms were operating in a trusted, completely deterministic setting—while obtaining from the framework a planning implementation that safely handles probabilistic collision checks and adversarial uncertainty about the environment. We show that our approach provides a principled way of planning without leveraging the simplifying assumption of treating all the obstacles as real, which could prevent the planner from even finding a feasible path (see Fig. 1). Moreover, our results show that our approach allows planning safer paths compared to a Naive Monte Carlo baseline when both approaches are allowed to use at most the same given number of samples to perform collision checks.

II. Related Work

A. Planning Under Uncertainty

The planning problems introduced in this paper can be framed in a more general context: planning in presence of uncertainty. Following the taxonomy outlined by Agha-Mohammadi et al. [6], uncertainty in planning stems from three sources: *motion*, *sensing* (also called *imperfect state information*), and *map*. We briefly discuss their differences, as related to the concepts presented in the paper.

Motion uncertainty refers to the noise affecting the dynamics of the system. Because the problem is so complex, planning is often simplified by focusing on techniques for tracking a nominal trajectory (obtained in a previous stage) with bounded collision probability [7]. In our work, we also decouple the problem of planning a nominal trajectory from the tracking problem, but focus on devising a method that bounds collision probability even in the presence of adversarial compromise.

Sensing uncertainty, also known as imperfect state information, refers to the noise affecting the sensor measurements that are fed to the robot's state estimation algorithm. As previously observed [6], this kind of uncertainty in planning is often considered along with motion uncertainty. Planning in this context is often referred to as *planning in belief space*, and in general amounts to solving a complicated Partially Observable Markov Decision Process via some reasonable approximation [6], [8]. The term "sensing uncertainty", however, is actually misleading in the context of this paper, as we assume that the robot can rely on a robust and precise state estimation procedure (such as the one that can be obtained outdoor with the help of GPS): what we consider is *map uncertainty*.

In the literature, map uncertainty has been studied under the assumption that the static map of the environment (obtained, for example, using the SLAM algorithm [9]) is subject to uncertainty [10]. In this work, instead, we assume that the static map of the environment contains negligible noise and is trusted as if provided by a trusted map server; our novelty consists in focusing instead on the uncertainty in the obstacles discovered at run time, considering both their existence and their precise location to be uncertain.

B. Collision Probability Estimation

The major challenge in tackling the problems introduced in this paper is to devise an efficient technique for estimating the collision probability—CP from now on—for any potential trajectory, as no closed form exists. Some work facing this issue (in problems involving "standard" uncertainties) prefers to enforce CP constraints *at each step* along the planned trajectory (with a given discretization), and to reason about approximations [11]. This approach works well when one would like to enforce an extremely small CP at each step along a short trajectory, because the CP along the entire trajectory will also likely be small. In our case, however, the robot may have to plan against a relatively high CP along a potentially long trajectory in order to find a feasible plan. Focusing on the computation of the CP of the entire trajectory, some work tries to simplify the problem by leveraging simplifying assumptions (like considering collision events as independent [12]). However, as discussed in [7], these methods might result in CPs that are off by a large margin. Other work resorts to Monte Carlo techniques [7]. In this work, we observe that it is not necessary to precisely compute CPs, *as long as there is high confidence in the satisfaction of the corresponding constraint*. Hence, we propose to leverage another sampling-based method, the Sequential Probability Ratio Test, to detect *with high confidence* whether the CP constraint is satisfied along the entire trajectory. To the best of our knowledge, ours is the first work to leverage the SPRT in path planning.

C. Probabilistic Programming

Several probabilistic programming languages have been developed with the aim of simplifying statistical modeling and reasoning. They allow for describing statistical models as programs and provide support for automated statistical inference. Languages like Pyro [13] are especially popular because, unlike in others, statistical models can be expressed directly in a general-purpose language like Python, inheriting expressive power from the host language and also making it convenient to use probabilistic programming within larger software projects.

However, languages like Pyro currently lack support for hypothesis testing. Hypothesis testing is important for cyber-physical systems in particular, because they need not only operate on uncertain data, but also do it in a calibrated way. Uncertain<T> [14] supports hypothesis testing, but unlike Pyro, it does not support automated Bayesian inference. As a domain-specific language (embedded in C#), it is also more limited in the kind of probabilistic models it can express.

III. Problem Formulation

A. Environment Model

We consider a mobile robot operating in a 2-D environment specified by a bounded region $X \subset \mathbb{R}^2$. W.l.o.g., it is assumed that X is partitioned into a (static) untraversable region $X_{obs} \subset X$ and free space $X_{free} \subseteq X$. The region X is provided by a map server, which is assumed to be trusted. For example, X can be the map obtained by means of a SLAM algorithm, or can be the map provided by an external entity.

We assume that the robot is equipped with sensors able to detect the presence of obstacles in the environment at run time. Furthermore, we assume that the robot is equipped with

an *inference engine* that receives raw input data coming from the sensors, and outputs a set of additional *independent* maps $M = \{M_i\}$ representing a probabilistic description of some obstacles that are *detected* and *tracked* at run time by one or more sensors. Each map can describe the environment as seen by some non-overlapping sensors (e.g., front and back lidars/cameras), or can contain group of obstacles consistently detected by all sensors. More specifically, each map M_i specifies a categorical distribution defined over a set of possible *scenarios*. We use $S_i = \{s_{ij}\}$ to denote the set of scenarios of map M_i. Each scenario is completely specified by a (possibly uncertain) description of a set of obstacles that are located in the environment, plus the probability that those obstacles are actually in the environment.

In particular, each scenario s_{ij} is defined as a pair $s_{ij} = \langle O_{ij}, p_{ij} \rangle$:

- $O_{ij} = \{o_{ij}^k\}$ represents a set of independent obstacles (possibly empty). Each obstacle o_{ij}^k is completely described by a tuple $\langle \mathbf{p}_{ij}^k, \mathcal{N}(\mu_{ij}^k, \Sigma_{ij}^k) \rangle$ where $\mathbf{p}_{ij}^k$ is the obstacle's 2-D polygonal shape, and $\mathcal{N}(\mu_{ij}^k, \Sigma_{ij}^k)$ is a multivariate Gaussian distribution which specifies the (x_{ij}^k, y_{ij}^k) coordinates in $\mathbb{R}^2$ of the centroid of $\mathbf{p}_{ij}^k$, along with its rotation θ_{ij}^k w.r.t. a given reference frame. (In this work it is assumed that the obstacle's shape is known without uncertainty, but this can be easily generalized [15]). This uncertain specification of obstacle o_{ij}^k holds assuming that it is actually present in the environment (see the next point).
- $p_{ij} \in [0, 1]$ is the probability that, in map M_i, scenario j is "the true one", while all the others are false. In other words, if a scenario is true, all the corresponding obstacles are actually present in the environment, and *vice versa*. Note that $\sum_{j=1}^{|S_i|} p_{ij} = 1$.

This model is completely agnostic about how these maps are actually obtained. As an example of a simple yet effective heuristic way of building such maps, consider a robot equipped with two completely identical and equally trusted sensors on the front. Under this setup, it is possible to run two independent standard object trackers, each one having as input the data coming from a different sensor. Given the two sets of detected objects, the inference engine can first filter out obstacles not having a close counterpart in the other object tracker. Then, the engine can try to match pairs of the remaining obstacles while minimizing a given distance metric (by solving an assignment problem [16]). The inference engine can finally use a decision table to handle situations where the result of the above procedure shows large discrepancies between the two sets of perceived obstacles, likely the result of adversarial attacks. For example, a situation related to the detection of an obstacle according to only one of the sensors could be dealt with by creating two scenarios in the same map M_1 with 0.5 probability each (recall that the sensors are identical), under the assumption that at least one sensor has not been compromised. If both sensors agree on the presence of other obstacles, whose estimated positions are very close (within a given small tolerance), one can create an additional map M_2 with a single scenario having probability 1. A slightly more complicated situation involving the presence of two identical sensors on the robot's back can simply be dealt with by adding other (independent) maps.

An interesting direction to explore, which we leave for future work, is related to devising a complete Bayesian framework in which the sensors' trustworthiness could be updated in a principled probabilistic way.

B. Path Planning Problems

We now present a formulation of the considered path planning problems based on the search of (feasible or optimal) paths that can be obtained by subsequent applications of suitable motion primitives chosen from a finite set. Although these problems could be defined more generally (for example, by extending Formulation 14.1 in [17]), we believe that the formulation given below is representative of most cases that may be of practical interest while, at the same time, capturing well the novelty introduced by this work.

Let us start from the feasibility version of the problem. The problem input is a tuple $\langle X, M, \mathbf{r}, q_I, Q_G, A, p_{\max} \rangle$ where:

- X and M are the planning region and obstacles' maps defined in Section III-A.
- $\mathbf{r}$ is a polygon representing the robot's footprint.
- q_I and Q_G are, respectively, the robot's *initial configuration* and *goal configuration set*. A robot configuration is a tuple $q = \langle x, y, \theta \rangle$ which fully specifies the space occupied by $\mathbf{r}$ in X w.r.t a given reference frame. A configuration q is said to be *legal* if the footprint $\mathbf{r}$, positioned as specified by q, does not overlap with X_{obs}.
- A is a discrete set of motion primitives available to the robot. Each motion primitive $a \in A$ moves the robot from a configuration q to a new configuration q' following a (continuous) trajectory $\overline{qq'}$.
- $p_{\max} \in [0, 1]$ is the maximum CP that the mission planner can tolerate for the robot.

Given a generic problem input as specified above, the set of feasible paths $\mathcal{P}$ can be defined as follows. Each $\pi \in \mathcal{P}$ specifies a sequence of n motion primitives

$$\pi = [a_1, a_2, \ldots, a_n]$$

leading the robot from the initial configuration q_I to any configuration $q \in Q_G$ via consecutive legal configurations. We introduce the notation $p_{\text{coll}}(\pi)$ to denote the CP of π, which is defined as the probability of obtaining a non-empty intersection in X_{free} between the planar region spanned by the subsequent robots' configurations (obtained by iteratively applying the motion primitives specified by π) and any of the random obstacles described by the set of maps M. Note that this definition implicitly assumes that the robot does *not* collide against any obstacle during the motion primitive execution; $p_{\text{coll}}(\pi)$ is hence an approximation of $p_{\text{coll}}(\overline{\pi})$, where $\overline{\pi} \subseteq X$ denotes the set of points of X spanned by the robot's footprint $\mathbf{r}$ when the corresponding centroid and orientation evolve along the continuous trajectory associated with π. Let also $p_{\text{coll},i}(\pi)$ be the CP of π when only map M_i is considered, and let $p_{\text{coll},ij}^k(\pi)$ be the CP of π restricted to

obstacle o_{ij}^k in scenario s_{ij}, *assuming that scenario* s_{ij} *is true*. The following relations hold:

$$p_{\text{coll}}(\pi) = 1 - \prod_{M_i \in M} \left[1 - p_{\text{coll},i}(\pi)\right] \tag{1}$$

$$p_{\text{coll},i}(\pi) = \sum_{s_{ij} \in S_i} p_{ij} \cdot \left[1 - \prod_{o_{ij}^k \in O_{ij}} \left(1 - p_{\text{coll},ij}^k(\pi)\right)\right] \tag{2}$$

Eq. (1) follows from the mutual independence of the maps in M, while Eq. (2) follows from the fact that —in a single map— scenarios are mutually exclusive, coped with the independence assumption on the obstacles belonging to the same scenario (which is rather standard [10]).

We define the *Mixed-Uncertainty Feasible Path Problem* as follows:

Problem 1 (Mixed-Uncertainty Feasible Path Problem – MUFPP): Given the tuple $\langle X, M, \mathbf{r}, q_I, Q_G, A, p_{\max}\rangle$, compute a path $\pi \in \mathcal{P}$ such that $p_{\text{coll}}(\pi) \leq p_{\max}$, or determine that no such path exists.

In practice, one might also be interested in computing a feasible path that optimizes a given performance metric. To this aim, we enrich the previous formulation with a cost function $c : A \to \mathbb{Q}_0^+$ describing the cost the robot accrues when executing a given motion primitive (e.g., time, distance). The *Mixed-Uncertainty Minimum Cost Path Problem* is the following:

Problem 2 (Mixed-Uncertainty Minimum Cost Path Problem – MUMCPP): Given the tuple $\langle X, M, \mathbf{r}, q_I, Q_G, A, p_{\max}, c\rangle$, compute a path $\pi \in \mathcal{P}$ such that $p_{\text{coll}}(\pi) \leq p_{\max}$ minimizing $\sum_{i=1}^{n} c(a_i)$, or determine that no such path exists.

C. Complexity

To get an idea of the additional challenges offered by MUFPP and MUMCPP compared to a "standard" path planning problem, let us consider a simplified version where:

- Planning takes place on a 4-connected grid.
- The robot is holonomic, and its footprint $\mathbf{r}$ can be expressed as the union of a finite number of neighboring grid cells. The initial configuration q_I is simply specified by a set of cells initially spanned by $\mathbf{r}$. The available motion primitives A correspond to moving the footprint's cells in one of the four cardinal directions.
- In each scenario S_{ij} of each map M_i, all obstacles o_{ij}^k are simply expressed as the union of a finite number of neighboring grid cells, with no Gaussian uncertainty (in Eq. (2), $p_{\text{coll},ij}^k(\pi)$ can be either 0 or 1).

The above simplification leads to a legitimate way of tackling the MUFPP and MUMCPP in presence of a holonomic robot and negligible Gaussian uncertainty on the obstacles. Let us focus on the MUFPP, and call MUFPP-S its simplified variant. The theorem below shows that the presence of uncertainty in the obstacles' existence turns a problem widely known to be in P into a likely intractable one.

Theorem 1: MUFPP-S is NP-complete.[1][2]

A proof sketch is reported in the Appendix. We now turn our attention on the more general (and interesting) problems, leaving the development of *ad hoc* algorithms for MUFPP-S for future works.

IV. A Sampling-Based, PPL-Aided Framework for Programming Search Algorithms

As discussed in Section II, the first and biggest challenge in developing algorithms for the MUFPP and MUMCPP is represented by the need of checking the satisfaction of the given collision-probability constraint along the trajectory that is being constructed. On top of that, Theorem 1 shows that the additional layer of uncertainty likely precludes the existence of polynomial-time algorithms even when no Gaussian uncertainty is present. Therefore, an approach decoupling the computation of $p_{\text{coll},ij}^k(\pi)$ from the closed-form part of Eq. (2) would have to simultaneously deal with two intractable problems. For this reason, we focus our attention on sampling-based methods, where one sample represents one of the many possible "true" *worlds* (as defined in Section III-A; note that the obstacle region X_{obs} must be present in each world sample). However, because collision checking is computationally expensive, a naive Monte Carlo approach offering a guaranteed bounded error rate (by performing collision checking against a fixed, usually large number of samples) would be too slow (see Section VI). Luckily, this performance penalty can be reduced significantly using a sequential approach. In particular, the one we propose is based on the usage of a hypothesis test called Sequential Probability Ratio Test (SPRT) [3].

A. Sequential Probability Ratio Test

The SPRT is a sequential hypothesis test. As such, the samples needed to confirm/reject the hypothesis do not need to be drawn all in advance; instead, they should be collected on-demand, until enough evidence to confirm/reject the hypothesis is collected. In particular, the SPRT starts as usual with a null hypothesis H_0, an alternative hypothesis H_1, and the desired type-I and type-II error rates α and β. The test statistic is the log-likelihood ratio, computed over all the samples seen so far. As new samples become available, the cumulative log-likelihood ratio Λ_i is simply computed as

$$\Lambda_i = \Lambda_{i-1} + \lambda_i,$$

with $\Lambda_0 = 0$ and λ_i the log-likelihood ratio of observing the i-th sample under the hypotheses.

Once Λ_i is computed, there are three possible outcomes:

(O1) $\Lambda_i \leq \log \frac{\beta}{1-\alpha}$: Accept H_0.

(O2) $\Lambda_i \geq \log \frac{1-\beta}{\alpha}$: Accept H_1.

(O3) $\log \frac{\beta}{1-\alpha} < \Lambda_i < \log \frac{1-\beta}{\alpha}$: The test statistic is not significant enough yet; keep drawing samples.

[1] Under the customary assumption that all the probabilities given in input are rational numbers [18].

[2] As an immediate corollary, we have that the simplified version of MUMCPP—a combinatorial optimization problem—does not belong to the exp-APX complexity class unless P=NP (see [19], Chapter 8)

Outcome O3 means the samples drawn so far do not provide enough evidence either for or against rejecting the null hypothesis H_0. So the SPRT goes on to draw more samples, compute Λ_{i+1}, and check the conditions of O1–O3 again. The SPRT does not guarantee an upper bound on the number of samples needed; in order to retain the guarantees on error rates, the above process must be repeated until a conclusion can be reached.

B. SPRT for Probabilistic Collision Checks

In the setting of an MUFPP or MUMCPP instance, we are given a path $\pi \in \mathcal{P}$, and we want to decide whether $p_{\text{coll}}(\pi) \leq p_{\max}$. Define the Bernoulli random variable Z_π as follows:

$$Z_\pi \sim \text{Bernoulli}(p_{\text{coll}}(\pi)). \tag{3}$$

The hypotheses to test are then predicates on the parameter of this distribution: the null hypothesis is $p_{\text{coll}}(\pi) > p_{\max}$, and the alternative $p_{\text{coll}}(\pi) \leq p_{\max}$. Type-I errors occur when collision-inducing paths are mistakenly accepted, while type-II errors occur when collision-avoiding paths are mistakenly rejected. Because this test makes Bernoulli trials, it is standard to approximate the *composite hypotheses* above using the following *simple hypotheses* [3]:

$$\begin{aligned} H_0 &\stackrel{def}{=} p_{\text{coll}}(\pi) = p_0 \\ H_1 &\stackrel{def}{=} p_{\text{coll}}(\pi) = p_1 \end{aligned}$$

where $p_0 > p_{\max} > p_1$ and the small interval (p_1, p_0) is called the *indifference region*—we are okay with accepting either H_0 or H_1 when the true value of p_{coll} is in the indifference region. The log-likelihood ratio for the Bernoulli trials can be computed as follows:

$$\begin{aligned} \Lambda_i &= \Lambda_{i-1} + LLR\,(n_i, N_i) \\ LLR(n, N) &= n \log \tfrac{p_1}{p_0} + (N - n) \log \tfrac{1-p_1}{1-p_0}. \end{aligned}$$

where N_i is the total number of samples $\{z_\pi\}_i$ drawn since last observing outcome O3, and n_i is the number of those samples that take the value $true$.

The samples $\{z_\pi\}_i$ are not drawn directly from the Bernoulli distribution (3) because $p_{coll}(\pi)$ is unknown. Instead, they are drawn from the nondeterministic function f:

$$f(\pi) \stackrel{def}{=} \begin{cases} true, & \pi \text{ collides with obstacles,} \\ false, & \text{otherwise.} \end{cases} \tag{4}$$

The nondeterminism comes from obstacles' being probabilistic. Function f is our generative probabilistic model for the random variable Z_π. Drawing samples from this probabilistic model requires drawing samples from the underlying probabilistic distribution describing the world.

Algorithm 1 shows the pseudocode of the hypothesis test described above. A typical implementation of the SPRT either runs indefinitely before reaching a conclusion, or returns a default value when all sampling resources have been used up. Instead, our implementation balances this trade-off by allowing the programmer to specify an error type to be conservative about (lines 18–23): when the guarantee on type-I (or type-II) error rates is desired, the algorithm terminates early if there would not be enough evidence to support accepting H_1 (or H_0) even when all remaining samples tested false (or true). Safety is concerned with controlling the first type of errors, so we invoke the algorithm with the parameter t being I.

Algorithm 1 Testing hypothesis $\Pr(f(\pi) = true) \leq p_{\max}$

Input: probabilistic model f of path π, CPs p_0 and p_1 where $p_0 > p_{\max} > p_1$, error rates α and β, maximum number of samples N_{max}, number of samples between checks N_{step}, type of error to be conservative about t (I or II).
Output: **true** (resp. **false**) if it can be concluded that, with an error rate at most α (resp. β), the chance of π colliding with the probabilistic obstacles is at most (resp. at least) $p_{\max}$.

1: $\Lambda \leftarrow 0$ *// cumulative log-likelihood*
2: $n_{total} \leftarrow 0$ *// total number of samples drawn*
3: $n_{true} \leftarrow 0$ *// number of samples testing true*
4: **while true do**
5: $\quad m_{true} \leftarrow 0$ *// number of samples testing true since last check*
6: $\quad$ **for** k **where** $0 \leq k < N_{step}$ **do**
7: $\quad\quad$ **if** $f(\pi)$ **then**
8: $\quad\quad\quad m_{true} \leftarrow m_{true} + 1$
9: $\quad\quad$ **end if**
10: $\quad$ **end for**
11: $\quad n_{true} \leftarrow n_{true} + m_{true}$
12: $\quad n_{total} \leftarrow n_{total} + N_{step}$
13: $\quad \Lambda \leftarrow \Lambda + LLR\,(m_{true}, N_{step})$
14: $\quad$ **if** $\Lambda \leq \log \frac{\beta}{1-\alpha}$ **then**
15: $\quad\quad$ **return false** *// Condition O1 met; accept* H_0
16: $\quad$ **else if** $\Lambda \geq \log \frac{1-\beta}{\alpha}$ **then**
17: $\quad\quad$ **return true** *// Condition O2 met; accept* H_1
18: $\quad$ **else if** $t =$ I **and** $LLR(n_{true}, N_{max}) \leq \log \frac{\beta}{1-\alpha}$ **then**
19: $\quad\quad$ *// Unable to accept* H_1 *even if all remaining samples tested false*
20: $\quad\quad$ **return false**
21: $\quad$ **else if** $t =$ II **and** $LLR(n_{true} + N_{max} - n_{total}, N_{max}) \geq \log \frac{1-\beta}{\alpha}$ **then**
22: $\quad\quad$ *// Unable to accept* H_0 *even if all remaining samples tested true*
23: $\quad\quad$ **return true**
24: $\quad$ **end if**
25: **end while**

Remark 1: As the reader might have noticed, the use of the SPRT introduces an additional dimension in the definition of the MUFPP and MUMCPP, namely, the choice of the parameters α, β, p_0 and p_1. However, we argue that the complexity of the problem justifies solution approaches dealing with such a slightly relaxed form.

Remark 2: The log-likelihood ratio thresholds $\log \frac{\beta}{1-\alpha}$ and $\log \frac{1-\beta}{\alpha}$ are approximations primarily designed for situations where α and β are small [3]. This approximation ensures that the "true" type-I and type-II errors α' and β' are bounded as $\alpha' \leq \alpha/(1-\beta)$ and $\beta' \leq \beta/(1-\alpha)$. Moreover, it can be shown that at least one of the inequalities $\alpha' \leq \alpha$ and $\beta' \leq \beta$ must hold. Preliminary experiments showed that setting both α and β small (≤ 0.05) requires a number of samples too large for our Python-based prototype implementation. We found a good trade-off in setting $\alpha = 0.05$ and $\beta = 0.2$: although this might result in an actual type-I error of 0.0625, our validation campaign shows that this might have a significant impact only for small $p_{\max}$ (0.01, where the actual CP might be around 0.012 in our most challenging planning scenario).

Remark 3: The usage of the SPRT is not limited to the particular probability distribution defined in Sec. III. For that distribution, however, we show that SPRT works better than a Naive Monte Carlo baseline (see Sec. VI.)

C. Searching for a Plan Using SPRT

The development in section IV-B can be integrated easily into a generic search algorithm that constructs a path iteratively. The key idea of virtually any search algorithm for a path-planning problem based on the use of a set of motion primitives A is to build a start–goal trajectory in an iterative fashion. This is accomplished by "extending" a suitably chosen partial trajectory π (generated in a previous iteration) to a new partial trajectory π' obtained by applying the primitive $a \in A$ on π. In a typical deterministic setting, one only has to make sure that π' does not overlap with X_{obs}. In our setting, we also need to make sure that π', with high probability, does not break the constraint on the CP. To this end, we simply apply the test in Algorithm 1 to π'. Note that the same partial trajectory π' should not be checked for collisions more than once during the search, since this could bias the result.

D. Further Speeding up the Search

Although the use of the SPRT helps reduce the number of samples needed, a naive implementation would still result in more computations than necessary. Notice that the hypothesis test is performed in each iteration of the search algorithm (i.e., every time the current configuration is extended). Also notice that π' collides with a specific sample of the probabilistic obstacles when π collides with the obstacles or when the extension from π to π' collides with the obstacles. So it is possible to speed up search by reusing the result of collision checkings on π in a previous iteration for collision checkings on π'. This optimization can be implemented by modifying only the probabilistic model f in (4); Algorithm 1 is transparent to the implementation of the probabilistic model.

We pre-generate a pool of world samples. Each time f requests a world sample, a uniformly random draw from the pool is made, and the result of computing f with that sample memoized to speed up search later.

E. Programming Support

We have extended the popular probabilistic programming language Pyro [13] with support for SPRT[3]. Thanks to this clean interface to the SPRT as well as to Pyro's programming support for writing probabilistic models, it is straightforward—as sections IV-C and IV-D shows—to integrate the reasoning about highly uncertain probabilistic worlds into common search algorithms for path-planning problems.

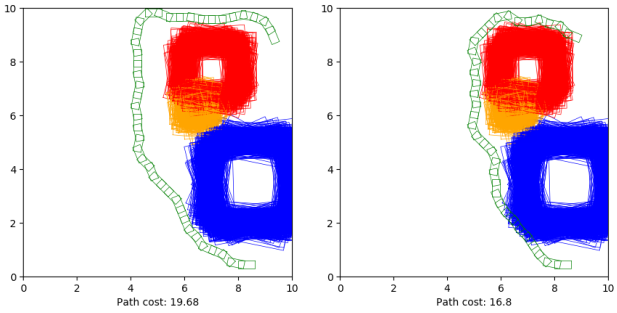
V. Example: RRT and Hybrid-A* Implementation

The proposed framework is used to implement two well-known planners, RRT [4] and Hybrid-A* [5]. Due to space constraints, the reader is referred to the original papers for the description of the standard versions of these algorithms.

We use RRT, a sampling-based planner designed to find feasible solutions quickly, to tackle the MUFPP. Our framework implicitly augments each configuration q in the RRT that is being constructed with a set of world samples that are used to determine whether the trajectory leading from q_I to q satisfies the given CP constraint. The programmer, after having defined a suitable world sampling scheme, can implement a collision check function pretending that the world is completely deterministic—i.e., as if she/he were only dealing with a fixed, deterministic obstacle region X_{obs}. The programmer could also be given access to the samples associated with each configuration, to implement more elaborate configuration sampling strategies. This is left for future work.

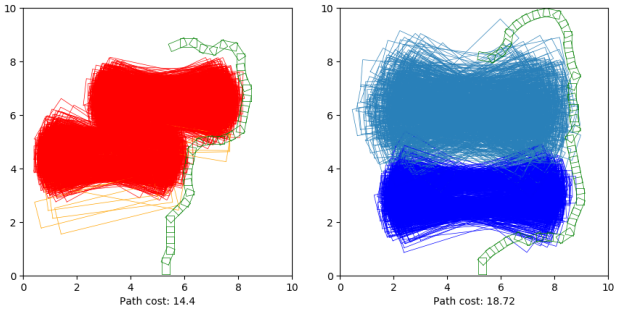
We use Hybrid-A* to tackle a MUMCPP where the accrued cost is the traveled distance. As with the RRT implementation, our framework implicitly augments each configuration q with a set of world samples. Configurations are associated to grid cells as in the original Hybrid-A* algorithm, regardless of how close they are to the violation of the CP constraint. It could also be possible to modify Hybrid-A* into variants able to leverage an estimate of configurations' CPs (obtained from SPRT samples drawn so far) in heuristic ways. This direction is left for future work.

[3]The source code is available at https://github.com/jacoban/mu_planner with detailed instructions to replicate our results.



$p_{\max} = 0.01$ $p_{\max} = 0.25$

Fig. 2: Example Hybrid A* paths for MG-HU.



MG-SU, $p_{\max} = 0.1$ HG-HU, $p_{\max} = 0.1$

Fig. 3: Example RRT paths for MG-SU and HG-HU.

VI. Results

We consider a differential drive robot with a set of motion primitives A built as follows: apply a constant forward speed of 0.8 m/s and an angular speed in $\{0.0, \pm\pi/8, \pm\pi/4, \pm 7/8\pi, \pm\pi/2\}$ rad/s for 0.6 s. We test the proposed framework in four representative planning problems, labeled MG-HU, MG-SU, HG-HU, and BL. MG-HU (moderate Gaussian uncertainty with high scenario uncertainty, see Fig. 2) contains two independent probabilistic maps, where the

first has a single square obstacle existing with probability 1 (blue), and the second has two scenarios, each one containing a square obstacle of a different size; the scenario with the red obstacle has probability 0.7 (the orange has 0.3). MG-SU (moderate Gaussian uncertainty with small scenario uncertainty, see Fig. 3 left) contains a single map with two scenarios having two rectangular obstacles each. The upper red ones are associated with a scenario with probability 0.99 (the orange ones to a scenario with probability 0.01). In both MG-HU and MG-SU, the Gaussian uncertainty on the obstacles is described by a diagonal matrix $\text{diag}(0.05, 0.05, 0.03)$ (the last element refers to the rotation). HG-HU (high Gaussian and scenario uncertainty, see Fig. 3 right) contains a single map with two equally probable scenarios with one obstacle each. The lower obstacle has the same Gaussian uncertainty of MG-HU and MG-SU, while the upper one has a larger uncertainty described by the diagonal matrix $\text{diag}(0.25, 0.25, 0.1)$. BL ("blocking" scenario) contains again a single map with two scenarios as explained in Fig. 1. The upper obstacle has larger Gaussian uncertainty, as in HG-HU. This scenario can only be planned in for values of $p_{\max} > 0.15$. In all cases, X_{obs} delimits a planning region confined to a square with side length 10 m, and goals are $1m \times 1m$ regions located in the upper part of the environment. We set $\alpha = 0.05$ and $\beta = 0.2$ for the SPRT (see Remark 2). The maximum number of samples checked at each new configuration is set to 300 for $p_{\max} = 0.01$ and 250 for $p_{\max} = 0.1, 0.25$. The planners are run on a computer equipped with an i7-6850K processor and 32 GB RAM. Figs 1-3 show some example paths obtained with RRT and Hybrid A*. Each path is shown with the obstacle samples pool (1000) used to check the satisfaction of the corresponding CP constraint.

We also compared our SPRT based approach against an alternative one that computes CP estimates with a Naive Monte Carlo (MC) approach, and performs a z-test to check if it can be concluded, with 95% confidence, that the trajectory leading to the current configuration satisfies the given CP constraint (we followed [20], Chapter 2). As in the SPRT, the z-test is iteratively performed on an increasing number of samples until (a) the trajectory is proved to satisfy the CP constraint, (b) the trajectory is proved to violate the CP constraint or (c) the procedure hits the maximum number of samples. To make the comparison fair, we use as maximum number of samples the same used by the SPRT. The number of initial MC samples is set to 300 for $p_{\max} = 0.01$ (the minimum number of samples needed by MC to declare "no collision" when none of the samples collides), and to 128 and 110 for $p_{\max} = 0.1$ and $p_{\max} = 0.25$, resp. (the minimum number of samples needed by the SPRT to declare "no collision"). We run 200 planning instances for each scenario. The results are shown in Table I, where values are reported as average $\pm$ standard deviation.

For $p_{\max} = 0.01$, the SPRT always allows planning safer paths compared to MC, with a CP constraint violation of at most 0.002 in average (in HG-HU with RRT). However, in one case (MG-HU with Hybrid A*) MC provides an interesting performance: the CP is in the indifference region (although higher than the SPRT one), but 14% more instances are solved. Collision checks and computation times are always lower with the SPRT, sometimes by around 50%.

$p_{\max} = 0.01$		%	Checks ($\times 10^6$)	Time (s)	$p_{\text{coll}}(\pi)$
MG-HU					
RRT	SPRT	39	1.22 $\pm$ 0.52	64.7 $\pm$ 26.8	0.011 $\pm$ 0.005
	MC	46	2.07 $\pm$ 0.77	104.5 $\pm$ 42.3	0.014 $\pm$ 0.007
HA*	SPRT	58	3.65 $\pm$ 0.89	115.2 $\pm$ 29.5	0.007 $\pm$ 0.003
	MC	72	3.96 $\pm$ 1.14	126.7 $\pm$ 38.8	0.011 $\pm$ 0.004
MG-SU					
RRT	SPRT	82	0.75 $\pm$ 0.38	31.4 $\pm$ 20.9	0.006 $\pm$ 0.004
	MC	88	1.31 $\pm$ 0.65	58.6 $\pm$ 34.2	0.010 $\pm$ 0.006
HA*	SPRT	100	1.57 $\pm$ 0.21	42.7 $\pm$ 5.8	0.005 $\pm$ 0.003
	MC	100	2.01 $\pm$ 0.22	58.9 $\pm$ 6.5	0.008 $\pm$ 0.004
HG-HU					
RRT	SPRT	45	0.52 $\pm$ 0.18	30.4 $\pm$ 14.3	0.012 $\pm$ 0.005
	MC	48	1.25 $\pm$ 0.50	62.2 $\pm$ 28.7	0.017 $\pm$ 0.007
HA*	SPRT	91	0.84 $\pm$ 0.20	20.7 $\pm$ 5.0	0.008 $\pm$ 0.004
	MC	94	1.36 $\pm$ 0.24	38.0 $\pm$ 7.4	0.013 $\pm$ 0.004
$p_{\max} = 0.1$		%	Checks ($\times 10^6$)	Time (s)	$p_{\text{coll}}(\pi)$
MG-HU					
RRT	SPRT	61	1.25 $\pm$ 0.57	61.2 $\pm$ 31.6	0.069 $\pm$ 0.015
	MC	59	1.15 $\pm$ 0.50	54.7 $\pm$ 27.7	0.095 $\pm$ 0.025
HA*	SPRT	96	1.98 $\pm$ 0.63	53.2 $\pm$ 17.7	0.059 $\pm$ 0.014
	MC	99	1.78 $\pm$ 0.50	45.3 $\pm$ 13.2	0.088 $\pm$ 0.018
MG-SU					
RRT	SPRT	96	0.63 $\pm$ 0.34	23.9 $\pm$ 16.8	0.054 $\pm$ 0.023
	MC	96	0.65 $\pm$ 0.34	26.1 $\pm$ 17.8	0.078 $\pm$ 0.032
HA*	SPRT	100	1.45 $\pm$ 0.13	36.0 $\pm$ 3.1	0.053 $\pm$ 0.013
	MC	100	1.39 $\pm$ 0.09	35.2 $\pm$ 2.2	0.079 $\pm$ 0.016
HG-HU					
RRT	SPRT	67	0.73 $\pm$ 0.33	34.0 $\pm$ 20.9	0.069 $\pm$ 0.015
	MC	72	0.72 $\pm$ 0.34	34.2 $\pm$ 21.5	0.102 $\pm$ 0.021
HA*	SPRT	99	1.36 $\pm$ 0.17	32.5 $\pm$ 4.5	0.062 $\pm$ 0.010
	MC	100	1.35 $\pm$ 0.14	30.5 $\pm$ 3.6	0.094 $\pm$ 0.014
$p_{\max} = 0.25$		%	Checks ($\times 10^6$)	Time (s)	$p_{\text{coll}}(\pi)$
MG-HU					
RRT	SPRT	65	1.23 $\pm$ 0.61	61.7 $\pm$ 36.1	0.159 $\pm$ 0.036
	MC	74	1.00 $\pm$ 0.52	47.0 $\pm$ 30.2	0.240 $\pm$ 0.038
HA*	SPRT	96	2.03 $\pm$ 0.67	58.3 $\pm$ 19.8	0.159 $\pm$ 0.019
	MC	97	1.56 $\pm$ 0.54	40.9 $\pm$ 14.6	0.228 $\pm$ 0.028
MG-SU					
RRT	SPRT	95	0.69 $\pm$ 0.40	28.6 $\pm$ 20.9	0.129 $\pm$ 0.057
	MC	98	0.56 $\pm$ 0.30	23.0 $\pm$ 15.8	0.177 $\pm$ 0.084
HA*	SPRT	100	1.42 $\pm$ 0.13	37.3 $\pm$ 3.2	0.138 $\pm$ 0.029
	MC	100	1.33 $\pm$ 0.13	34.5 $\pm$ 3.1	0.201 $\pm$ 0.036
HG-HU					
RRT	SPRT	77	0.76 $\pm$ 0.42	35.6 $\pm$ 25.3	0.168 $\pm$ 0.030
	MC	82	0.68 $\pm$ 0.33	32.4 $\pm$ 21.3	0.238 $\pm$ 0.046
HA*	SPRT	100	1.48 $\pm$ 0.18	37.2 $\pm$ 5.4	0.160 $\pm$ 0.014
	MC	100	1.47 $\pm$ 0.20	35.9 $\pm$ 5.5	0.236 $\pm$ 0.018
BL					
RRT	SPRT	85	0.94 $\pm$ 0.45	39.3 $\pm$ 24.5	0.164 $\pm$ 0.013
	MC	99	0.60 $\pm$ 0.31	21.2 $\pm$ 14.0	0.212 $\pm$ 0.035
HA*	SPRT	91	2.18 $\pm$ 0.34	56.5 $\pm$ 9.3	0.160 $\pm$ 0.010
	MC	100	1.57 $\pm$ 0.18	36.7 $\pm$ 4.2	0.224 $\pm$ 0.021

TABLE I: SPRT vs Naive MC. Green: $p_{\text{coll}}(\pi)$ avg. + std. dev. $\leq$ CP constraint. Orange: $p_{\text{coll}}(\pi)$ avg. + std. dev. in indifference region. Red: $p_{\text{coll}}(\pi)$ avg. + std. dev. outside indifference region. "%" is the percentage of instances solved within 5000 iterations.

For $p_{\max} = 0.1$, the SPRT always provides a superior performance in the satisfaction of the CP constraint. Computation times and collision checks are essentially equivalent. For $p_{\max} = 0.25$, the results are similar to those obtained with $p_{\max} = 0.1$, with the SPRT still superior overall. However, in the BL scenario, MC is always able to respect the CP constraint (although with a higher CP), solves almost all the instances, and provides lower computation times. This is related to the fact that the SPRT might run out of samples and declare that a configuration is unsafe while in the middle of traversing the red obstacle. A slightly larger number of samples would make this situation less likely. In general we noticed, as expected, that Hybrid A* is able to generate shorter

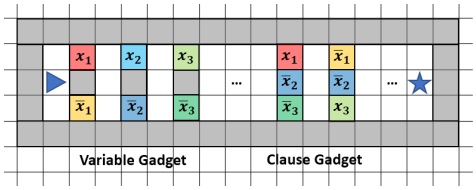


Fig. 4: Reduction example. The 3-SAT instance has two clauses $c_1 = x_1 \vee \bar{x}_2 \vee \bar{x}_3$ and $c_2 = \bar{x}_1 \vee \bar{x}_2 \vee x_3$. Start and goal cells are denoted by the triangle and the star, respectively. Gray cells denote X_{obs}, colored cells probabilistic obstacles.

paths compared to RRT.

With respect to memory usage, the planning instance requiring the largest number of collision checks (MG-HU with Hybrid A*) takes up to 500 MiB, which should not be a problem for most robotic platforms.

VII. Conclusions

This paper introduces and studies two path-planning problems arising in adversarial scenarios where an attacker has tampered with robot sensor readings in order to create fake obstacles or to remove real ones. To ease the programming task, we proposed a general programming framework based on SPRT and implemented it as an extension of Pyro, a well-known probabilistic programming language. Although it performs better overall than Naive MC, our current pipeline is not fast enough to be used in real time. Python is only part of the problem; code may spend a quarter of its computation time in the collision-checking library, which is written in C. Therefore, we envision future research directed toward (a) allowing the user to specify how often the SPRT should be performed, (b) implementing this pipeline in a lower-level probabilistic programming language, (c) leveraging the Gaussian part of the distribution by introducing a hierarchical 2-D grid structure for grouping samples, and (d) implementing collision checks on specialized hardware, such as GPUs and FPGAs.

Appendix
Proof of the NP-completeness of the MUFPP-S

NP membership follows from the fact that the robot's footprint never needs to occupy the same cells twice along a feasible path. NP-hardness is proved with a polynomial-time reduction from the NP-complete problem 3-Satisfiability [21]:

3-SAT

INSTANCE: set U of Boolean variables, set C of disjunctive clauses defined on them with exactly three literals each.

QUESTION: does there exist a truth assignment for U that satisfies all the clauses in C?

From a generic instance of 3-SAT, one can construct in polynomial time a particular instance of MUFPP-S following the scheme shown in Fig. 4, in which the robot's footprint covers a single cell. The MUFPP-S instance contains $|U|$ maps, one for each variable x_i, with two scenarios each having probability 0.5. Each scenario is associated with the appearance of x_i as a positive or negative literal in C, and contains $1 + N(l_i)$ obstacles spanning one single cell, where $N(l_i)$ denotes the number of times literal l_i appears in C. For each scenario, the first obstacle appears in the *variable gadget*, while the remaining $N(l_i)$ in the *clause gadget*. To pass the variable gadget, the robot has to make a choice about a scenario to traverse for each map. The clause gadget contains $|C|$ "barriers", one for each clause, containing the remaining obstacles of each scenario. The reduction is completed by setting $p_{\max} = 1 - 0.5^{|U|}$. It is easy to see that the 3-SAT instance is satisfiable iff the robot is able to reach the goal while respecting $p_{\max}$ (an unsatisfiable SAT instance would lead to collision with probability 1). ■

Acknowledgments

The authors would like to thank Mulong Luo for useful discussions.

References

[1] J. Liu, J. Corbett-Davies, A. Ferraiuolo, A. Ivanov, M. Luo, G. E. Suh, A. C. Myers, and M. Campbell, "Secure autonomous cyber-physical systems through verifiable information flow control," in *Proc. CPS-SPC*, 2018.
[2] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation With Applications to Tracking and Navigation*, Wiley, Ed., 2001.
[3] A. Wald, "Sequential tests of statistical hypotheses," *Ann Math Stat*, vol. 16, no. 2, 1945.
[4] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.
[5] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," in *Proc. STAIR*, 2008.
[6] A. Agha-Mohammadi, S. Chakravorty, and N. Amato, "Firm: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements," *Int J Robot Res*, vol. 33, no. 2, pp. 268–304, 2014.
[7] E. Schmerling and M. Pavone, "Evaluating trajectory collision probability through adaptive importance sampling for safe motion planning," in *Proc. RSS*, 2017.
[8] A. Bry and N. Roy, "Rapidly-exploring random belief trees for motion planning under uncertainty," in *Proc. ICRA*, 2011, pp. 723–730.
[9] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT Press, 2005.
[10] P. Missiuro and N. Roy, "Adapting probabilistic roadmaps to handle uncertain maps," in *Proc. ICRA*, 2006, pp. 1261–1267.
[11] J. Hardy and M. Campbell, "Contingency planning over probabilistic obstacle predictions for autonomous road vehicles," *IEEE Trans. Robot.*, vol. 29, no. 4, pp. 913–929, 2013.
[12] C. Park, J. Park, and D. Manocha, "Fast and bounded probabilistic collision detection in dynamic environments for high-dof trajectory planning," in *Proc. WAFR*, 2016.
[13] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman, "Pyro: Deep universal probabilistic programming," *J Mach Learn Res*, vol. 20, no. 1, 2019.
[14] J. Bornholt, T. Mytkowicz, and K. S. McKinley, "Uncertain<T>: A first-order type for uncertain data," in *Proc. ASPLOS*, 2014.
[15] K. Wyffels and M. Campbell, "Precision tracking via joint detailed shape estimation of arbitrary extended objects," *IEEE Trans. Robot.*, vol. 33, no. 2, pp. 313–332, 2016.
[16] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*. Society for Industrial and Applied Mathematics, 2009.
[17] S. LaValle, *Planning algorithms*. Cambridge University Press, 2006.
[18] O. Madani, S. Hanks, and A. Condon, "On the undecidability of probabilistic planning and related stochastic optimization problems," *Artif Intell*, vol. 147, no. 1-2, pp. 5–34, 2003.
[19] G. Ausiello, M. Protasi, A. Marchetti-Spaccamela, G. Gambosi, P. Crescenzi, and V. Kann, *Complexity and approximation*. Springer-Verlag, 1999.
[20] A. B. Owen, *Monte Carlo theory, methods and examples*, 2013.
[21] M. Garey and D. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*. W.H. Freeman, 1979.