

# Sampling-Based Gaussian Estimation of Probability of Collision for Safe Planning

Ajaay Chandrasekaran,  
*Department of Electrical and Computer Engineering*  
*University of Michigan*  
 Ann Arbor, Michigan 48109, U.S.A.  
 Email: ajaay@umich.edu

**Abstract**—This is the abstract.

**Index Terms**—collision probability, motion planning, safety, uncertainty.

## I. INTRODUCTION

**T**He use of robots in elderly care is becoming increasingly likely throughout the world, as the population of elderly people is beginning to overtake the number of potential caregivers.

## II. RELATED WORK

### III. ESTIMATING PROBABILITY OF COLLISION

#### A. Problem Statement

We consider a robot operating in an environment with obstacles. The motion model of the robot is given as:

$$x_{t+1} = g(x_{t-1}, u_{t-1}, m), m \sim \mathcal{N}(0, R_t)$$

where  $x_t \in \mathcal{C}$  is the configuration of the robot at time  $t$ ,  $u_t \in \mathbb{R}^{n_u}$  is the applied control input, and  $m$  is a zero-mean Gaussian noise variable with variance  $R_t$ .

During execution of the motion plan, the robot obtains measurements of surrounding landmarks. The sensor model of the robot is given as:

$$z_t = h(x_t, l, n), n \sim \mathcal{N}(0, Q_t)$$

where  $z_t \in \mathbb{R}^{n_z}$  is the measurement obtained at time  $t$ ,  $l \in \mathbb{R}^{n_l}$  is the location of the landmark that is observed, and  $n$  is a zero-mean Gaussian noise variable with variance  $Q_t$ .

We assume the existence of a nominal plan computed by a motion planner. The nominal plan is defined by:

$$[x_0^*, u_0^*, \dots, x_N^*, u_N^*]$$

where  $x_t^* = g(x_{t-1}^*, u_{t-1}^*, 0)$  for  $0 < t \leq N$ , where  $N$  is the number of configurations in the motion plan.

With the preliminary notation established, we express our problem statement. Our goal is to estimate the probability of collision  $p_c$  for the nominal plan. More formally, this probability is represented as:

$$p_c = p \left( \bigvee_{t=0}^N x_t^* \notin \mathcal{C}_F \right)$$

where  $\mathcal{C}_F \subset \mathcal{C}$  represents the space of non-colliding configurations.

#### B. Formulation and Assumptions

Much of existing literature on this subject assumes that we can represent obstacles in the environment as known constraints on the robot's feasible locations in the workspace. However, we do not make that assumption, and we assume that there instead exists a collision checker  $\phi : \mathcal{C} \rightarrow \{\text{TRUE}, \text{FALSE}\}$ , which can determine if a configuration is in collision. If  $x \notin \mathcal{C}_F$ ,  $\phi(x) = \text{TRUE}$ ; otherwise  $\phi(x) = \text{FALSE}$  [1].

The assumption of an available collision checker contrasts our formulation from other literatures, which tend to only consider point robots through the formulation of known robot location constraints based on obstacles. Our formulation lets us consider the possibility of non-point robots, whose locations in the workspace may cause collisions with obstacles due to the robots having 3D-geometries and being situated with complex orientations.

We assume that the Extended Kalman Filter (EKF) localization algorithm [5] is used to compute  $\hat{x}_t$ , an estimate of the robot's true configuration,  $x_t$ , at time  $t$ . The covariance of the EKF estimate of  $x_t$  is given by  $\Sigma_t$ . We assume that there is a set of already known landmarks in the environment,  $L = l_1, \dots, l_{n_l}$ , where  $n_l$  is the number of landmarks, and that data association between sensor measurements and landmarks is known.

Due to the Gaussian noise in the motion model, we expect that the robot will deviate from the nominal motion plan during its execution. To compensate for the motion uncertainty, we assume that the robot operates with a closed-loop feedback controller [3]. We assume that there exists a linear feedback control law that operates on the EKF estimate of the robot configuration and attempts to keep the robot close to the nominal plan. This feedback policy is given as:

$$\bar{u}_t = L_{t+1}(\hat{x}_t - x_t^*)$$

where  $L_t$  is the control gain matrix, which is contingent on the choice of feedback controller [3] and  $\bar{u}_t$  is the necessary deviation from the nominal control input  $u_t^*$  to account for the robot's deviation from the nominal plan. It follows that  $u_t = \bar{u}_t + u_t^*$ , where  $u_t$  represents the true control input to be executed by the robot—given the deviation of the robot's estimated configuration  $\hat{x}_t$  from the nominal configuration  $x_t^*$ .

### C. Sources of Difficulty

Computing the probability of collision for a motion plan in practice is a difficult task. There are three key sources of the difficulty:

- 1) Uncertainty in robot motion and sensor models
- 2) Dependent individual events of collision
- 3) Computational time of Monte-Carlo simulations

We proceed to describe each of these sources.

1) *Model Uncertainty*: Robot motion is often corrupted by noise. We find that when we command a robot to follow a nominal control input, it often does not follow the command exactly due to various factors—usually wheel slippage due to lack of uniformity of surfaces.

Furthermore, sensor measurements that are used to localize a robot are also corrupted by natural noise and inaccuracy. Using sensor measurements for localization naturally causes uncertainty in the robot’s known position. When using a state estimate as an input to a feedback control loop to correct the robot’s position, there is a subsequent uncertainty in the accuracy of the desired controller adjustment—despite the desired intention of following the nominal motion plan.

2) *Dependence between Collision Events*: A key feature of this problem is that the event of a collision not occurring at a configuration  $x_t$  of a motion plan is not independent of the event of a collision not occurring at previous configurations of the motion plan. More formally:

$$p(x_t \in \mathcal{C}_F) \neq p\left(x_t \in \mathcal{C}_F \mid \bigwedge_{i=0}^{t-1} x_i \in \mathcal{C}_F\right)$$

Prior approaches to solving this problem often consider the events of collisions along the motion plan to be independent of each other, which can lead to an overly conservative estimate of the probability of collision for a motion plan. Incorporating the fact that the collision events (and non-collision events) are dependent on each other increases the difficulty of computing an accurate estimate of the probability of collision.

3) *Computational Time of Monte-Carlo Simulations*: Running a large number of Monte-Carlo simulation, each having a large number of particles, can—in theory—give the true probability of collision for a motion plan. The idea of a Monte-Carlo simulation is to simulate the motion of thousands of particles, each particle representing a potential configuration of a robot’s configuration given uncertainty in the robot’s true configuration  $x_t$ . By finding the proportion of particles that did not collide with obstacles at all during the entirety of a motion plan, we can have an estimate of the probability of collision.

However, a single Monte-Carlo simulation is generally not sufficient, and we would need to average the proportions of colliding-particles across hundreds or even thousands of Monte-Carlo simulations to obtain an accurate estimate of the true probability of collision for the motion plan.

Furthermore, running a single Monte-Carlo simulation with thousands of particles can be very computationally expensive due to the compounding of time required for collision checking. Running thousands of Monte-Carlo simulations

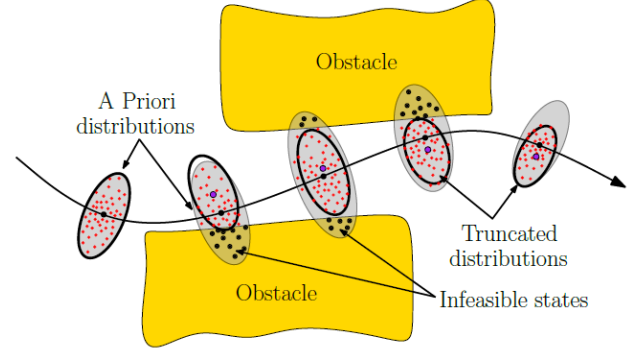


Fig. 1. Propagation and Truncation of State Estimate Distribution. Figure borrowed from Patil, van den Berg, Alterovitz 2012. “The probabilities of collision at each stage of the plan are conditioned on the previous stages being collision free. We truncate a priori distributions with respect to obstacles to discount plan executions that collide with obstacles (black disks). Propagating the truncated distributions (black ellipses) accounts for only the collision free samples (red disks), resulting in accurate estimation of the probability of collision. Using the unconditional distributions (gray ellipses) to estimate the collision probability results in a very conservative estimate.”[2]

compounds the computation time even further, causing this approach to be a poor idea in practice.

### D. Metric Evaluation

Given the sources of difficulty for this problem, we primarily strive to compute an accurate estimate of the true probability of collision for a motion plan. We also seek to compute the probability of collision for the plan in an amount of time that is approximately less than or equal to the time required for a single Monte-Carlo simulation while within the error bounds of the average proportion of colliding particles that would be computed from executing hundreds of Monte-Carlo simulations with many particles in each simulation instance.

Ideally, running our method should on average produce an estimate that is closer to the true probability of collision than the estimate that may be obtained from running a single Monte-Carlo simulation.

## IV. METHOD

### A. Method Basis

We propose a sampling-based method paired with the use of a Gaussian Mixture Model (GMM) to estimate the probability of collision for a motion plan. Our method adopts ideas from the the approach proposed by Patil, van den Berg, and Alterovitz [2]. However, our approach differs in that we do not assume that the environment obstacles can be represented as constraints on the robot location in the workspace, and we assume that a collision checker exists instead.

Figure 1 portrays the approach recognized in [2], which estimates the probability of collision for a motion plan by propagating a Kalman Filter (KF) estimate of a robot’s state and truncating the distribution of the estimate with respect to obstacles. Patil et al. make the observation that the distribution of the robot’s configuration at a timestep  $t$ —conditioned on the

configuration  $x_i \notin \mathcal{C}_F$ ,  $\forall i, 0 \leq i < t$  being collision-free, can be represented by truncating the KF estimate at  $x_t$  with respect to the obstacles. They also note that the proportion of the distribution that has been truncated by the obstacles would represent the conditional probability of collision for that state of the motion plan.

The approach of Patil et al. is purely analytical. It truncates the KF estimate of the robot state by sequentially applying each obstacle constraint to truncate the Gaussian distribution of the state estimate using analytical techniques from existing literature. They assume that the truncation of a Gaussian distribution by the obstacle constraints subsequently yields another Gaussian distribution.

The approach of Patil et al. makes two more important assumptions that are important to note when they truncate the KF estimate with respect to obstacles:

- 1) The obstacles in the environment can be expressed as linear constraints on the robot's location in the workspace
- 2) The free region containing the robot's configuration at time  $t$  is convex

We base our approach on the framework of [2], and attempt to release these assumptions as we define our sampling-based GMM approach to estimate the probability of collision.

### B. Method Overview and Justification

An important note to make is that the approach in [2] may be sufficient if we could express linear constraints in the configuration space  $\mathcal{C}$  of a robot. Knowing the obstacle constraints in  $\mathcal{C}$  rather than as constraints on location in the robot's 3D-workspace could allow us to apply the approach of Patil et al. to truncate the distribution that represents a KF estimate of the robot's configuration. However, in general, computation of  $\mathcal{C}_{OBS} \subset \mathcal{C}$ , the space of all robot configurations that lead to collision with an obstacle, is computationally expensive in practice. It follows that learning configuration constraints in  $\mathcal{C}$  rather than in the workspace may be very difficult in practice, as well. Furthermore, the constraints of  $\mathcal{C}$  may also not be linear since  $\mathcal{C}$  can have a complex topology.

Knowing (or learning) constraints for a robot's set of feasible configurations in  $\mathcal{C}$  does not seem computationally efficient; however, we assume that we have access to the collision checker  $\phi$ , which asserts whether an arbitrary  $x_t \in \mathcal{C}_{OBS}$ . Given the existence of a collision checker, we believe that a sampling-based approach for truncation of Gaussian distributions is necessary for a decent attempt of a solution to estimating the probability of collision. Other approaches assume the existence of a distance-to-obstacle checker [6], which may allow us to quantify the maximum clearance from an obstacle for a robot configuration to not collide. These approaches utilize this clearance in their computation of the probability of collision.

We observe that we need a method to truncate the robot state distribution computed from the KF estimate with respect to obstacles, where the proportion of the distribution that is truncated would represent the probability of collision for

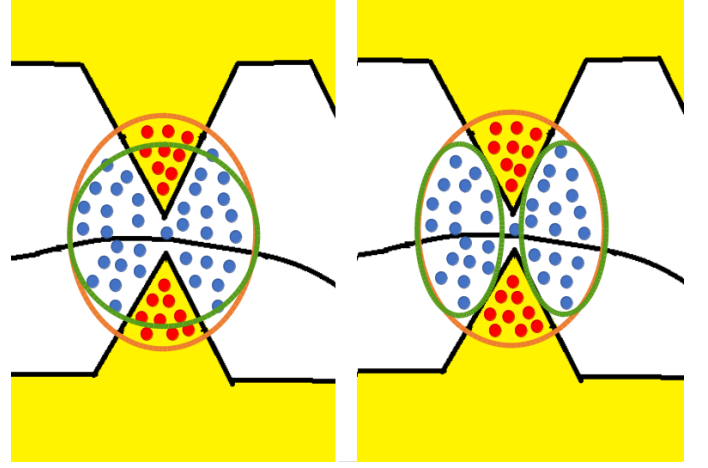


Fig. 2. Our GMM approach attempts to handle situations where the distribution obtained by the truncation of the robot's configuration distribution by obstacles may not necessarily be represented sufficiently by a single Gaussian. [Left] When the EKF estimate of the robot's configuration (orange ellipse) is truncated, a single Gaussian distribution (green ellipse) may not best represent the subsequent collision-free distribution. [Right] A mixture of two Gaussians may represent the truncated collision-free distribution better than a single Gaussians.

that waypoint of the motion plan—contingent on the the prior states being collision-free.

A straightforward approach using the collision checker involves sampling random configurations from the KF estimate of the robot's state and taking note of:

- 1) the proportion of sampled configurations in collision
- 2) the set of configurations that are not in collision

Given 1, we would have an estimate,  $p_{ct}$ , for the probability of collision for that waypoint of the motion plan, and  $1 - p_{ct}$  would represent the conditional probability of no collision, given that the previous states of the motion plan were collision-free. Given 2, we would be able to fit a distribution to the set of configurations that are not in collision, effectively truncating the original robot state distribution.

The left image of Figure 2 portrays this approach where we attempt to fit a single Gaussian distribution to the set of robot configurations that are not in collision. We find that in extreme cases like this, a single Gaussian distribution may not best represent the distribution of non-colliding particles. This can be especially common if the feasible region in  $\mathcal{C}$  containing  $x_t$  is non-convex. Our approach recognizes this and proposes to truncate the distribution and represent the distribution of the subsequent set of non-colliding points as a mixture of Gaussians, or a GMM. We believe that the use of a GMM would more accurately capture the distribution of non-colliding configurations. The idea of this approach is demonstrated in the right image of Figure 2.

We also note that the approach in [2] degenerates when the free space containing the robot configuration is non-convex, as they use a greedy method to convexify the free space and obtain and overestimate of the probability of collision. This stems from their assumption that a single Gaussian distribution best represents the truncated distribution. We construct our GMM approach with the intention that it can handle these

occurrences, representing the truncated distribution, as in the case of Figure 2, more accurately.

---

**Algorithm 1** Sampling-Based GMM Collision Estimation
 

---

**Input:**  $[x_0^*, u_0^*, \dots, x_N^*, u_N^*], \mu_0, \Sigma_0, N_G, N_p$   
**Output:**  $p_c$

```

1:  $\mu^* \leftarrow \mu_0$ 
2:  $\Sigma^* \leftarrow \Sigma_0$ 
3:  $p_f \leftarrow 1$ 
4:  $[\mu_1^*, \Sigma_1^*, \dots, \mu_{N_G}^*, \Sigma_{N_G}^*] \leftarrow \text{INIT\_GMM}(N_G, \mu_0, \Sigma_0)$ 
5:  $[\lambda_1, \dots, \lambda_{N_G}] \leftarrow [\frac{1}{N_G}, \dots, \frac{1}{N_G}]$ 
6: for  $i = 0$  to  $i = N - 1$  do
7:    $\bar{u}_i \leftarrow L_{t+1}(\mu^* - x_i^*)$ 
8:    $u_i \leftarrow u_i^* + \bar{u}_i$ 
9:    $\bar{\mu}^*, \bar{\Sigma}^* \leftarrow \text{EKF\_PREDICT}(\mu^*, \Sigma^*, u_i)$ 
10:   $z_i \leftarrow \text{SENSOR\_READ}()$ 
11:   $\mu^*, \Sigma^* \leftarrow \text{EKF\_CORRECT}(\bar{\mu}^*, \bar{\Sigma}^*, z_i)$ 
12:  for  $j = 1$  to  $j = N_G$  do
13:     $\bar{\mu}_i^*, \bar{\Sigma}_i^* \leftarrow \text{EKF\_PREDICT}(\mu_i^*, \Sigma_i^*, u_i)$ 
14:     $\mu_i^*, \Sigma_i^* \leftarrow \text{EKF\_CORRECT}(\bar{\mu}_i^*, \bar{\Sigma}_i^*, z_i)$ 
15:  end for
16:   $[\mu_1^*, \Sigma_1^*, \dots, \mu_{N_G}^*, \Sigma_{N_G}^*], p_i \leftarrow \text{TRUNCATE\_GMM}()$ 
17:   $p_f \leftarrow p_f * p_i$ 
18: end for
19:  $p_c \leftarrow 1 - p_f$ 

```

---



---

**Algorithm 2** TRUNCATE\_GMM
 

---

**Input:**  $[\mu_1^*, \Sigma_1^*, \dots, \mu_{N_G}^*, \Sigma_{N_G}^*], N_P$   
**Output:**  $[\mu_1^*, \Sigma_1^*, \dots, \mu_{N_G}^*, \Sigma_{N_G}^*], p_i$

```

1:  $[\chi_1, \dots, \chi_{N_G}] \leftarrow \text{SAMPLE\_GMM}(N_P)$ 
2:  $[\bar{\chi}_1, \dots, \bar{\chi}_{N_G}] \leftarrow [\emptyset, \dots, \emptyset]$ 
3:  $[F_1, \dots, F_{N_G}] \leftarrow [0, \dots, 0]$ 
4:  $C_c \leftarrow 0$ 
5: for  $j = 1$  to  $j = N_G$  do
6:   for  $q$  in  $\chi_j$  do
7:     if  $\text{CHECK\_COLLISION}(q) == \text{TRUE}$  then
8:        $C_c \leftarrow C_c + 1$ 
9:     else
10:       $\bar{\chi}_j \leftarrow \{\bar{\chi}_j, q\}$ 
11:       $F_j \leftarrow F_j + 1$ 
12:    end if
13:  end for
14: end for
15: for  $j = 1$  to  $j = N_G$  do
16:   $\mu_j^* \leftarrow \text{MEAN}(\bar{\chi}_j)$ 
17:   $\Sigma_j^* \leftarrow \text{COVARIANCE}(\bar{\chi}_j)$ 
18:   $\lambda_j \leftarrow \frac{F_j}{\sum_{i=1}^{N_G} F_i}$ 
19: end for
20:  $p_i \leftarrow \frac{C_c}{N_P}$ 
21:  $p_i \leftarrow 1 - p_i$ 

```

---

### C. Algorithm

Algorithm 1 portrays our sampling-based GMM collision probability estimation algorithm. The algorithm takes as input the nominal motion plan computed by a motion planner,

$[x_0^*, u_0^*, \dots, x_N^*, u_N^*]$ , and the initial configuration estimate  $\mu_0$  and its covariance  $\Sigma_0$ . The algorithm also receives the number of Gaussians to use  $N_G$ , and the number of samples  $N_p$  to take at each time step. The algorithm is also depicted pictorially in Figure 3.

The algorithm begins by initializing the EKF estimate mean and covariance. Generally,  $\mu_0 = x_0^*$ . Line 3 initializes a running proportion  $p_f$  for the probability that all configurations in the motion plan are configuration free. Line 4 initializes the GMM; the INIT\_GMM procedure simply assigns each mixture's  $\mu^*$  and  $\Sigma^*$  to be the same as  $\mu_0$  and  $\Sigma_0$  respectively. Furthermore, Line 5 initializes the weight,  $\lambda_i$ , of each mixture component to  $\frac{1}{N_G}$ .

Lines 6-18 consist of a main loop that propagates the EKF estimate of the robot's state, and the GMM representation of the collision-free conditional distribution, through the motion plan.

In Lines 7-8, the linear feedback control gain  $L_{t+1}$  is used to adjust the nominal control input  $u_i^*$  to attempt to fix the deviation between the KF estimate of the robot's current state  $\mu^*$  and the nominal state  $x_i^*$  in the motion plan [3].

The functions EKF\_PREDICT and EKF\_CORRECT represent the prediction and correction steps of the Extended Kalman Filter respectively [5]. Lines 9-11 apply the prediction and update to the KF estimate parameters  $\mu^*, \Sigma^*$  based on the adjusted control input and a set of sensor measurements  $z$  with known data association between measurements and landmarks. Furthermore, Lines 12-15 apply the EKF\_PREDICT and EKF\_UPDATE steps on each component of the GMM.

Line 16 applies the TRUNCATE\_GMM algorithm, which uses our sampling-based approach to truncate each GMM component. The algorithm takes as input the current means and covariances of the GMM and returns the updated means and covariances following truncation.

The TRUNCATE\_GMM algorithm also returns  $p_i$ , the proportion of samples drawn from the GMM mixture that did not collide with obstacles, which ultimately estimates the conditional probability that the current waypoint of the motion plan will be collision-free given that the previous waypoints are collision-free. This value is then used in Line 17 to update  $p_f$ , the running product representing the probability that all waypoints in the motion plan so far are collision-free.

Following the end of the main loop through all of the motion plan waypoints (Line 18),  $p_f$  stores an estimate of the the probability that all configurations in the motion plan will be collision-free. Line 19 computes  $p_c \leftarrow 1 - p_f$ , which represents the probability that there existed a configuration in the motion plan that experienced a collision.

We proceed to discuss the TRUNCATE\_GMM subprocedure, which is portrayed in Algorithm 2. Given the GMM represented by  $[\mu_1^*, \Sigma_1^*, \dots, \mu_{N_G}^*, \Sigma_{N_G}^*]$  and the number of samples  $N_P$ , the algorithm attempts to update the GMM representation of the robot's state distribution by discounting all colliding configurations from the distribution.

On Line 1,  $N_p$  configurations are sampled from the GMM using the weights  $\lambda_1, \dots, \lambda_{N_G}$ . For the mixture component parameterized by  $i, \Sigma_i$ , the set  $\chi_i$  stores the configurations that were obtained from that component during the sampling.

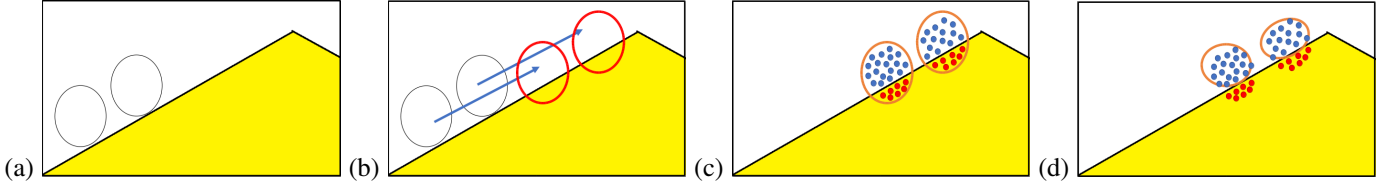


Fig. 3. Sampling-Based GMM Collision Estimation Algorithm applied to Mixture of 2 Gaussians. (a) Initialize mixture components with same initial mean  $x_0^*$  and covariance  $\Sigma_0$ , as well as equal weights (Means shown as different here to highlight distinction). Gaussians are represented by blue ellipses. (b) Propagate Gaussians in mixture to next state of motion plan via EKF prediction step. Updated Gaussians are represented by red ellipses. (c) Update Gaussians via EKF correction step based on measurements (orange ellipses). Then, sample  $N_p$  configurations from GMM based on weights. Check which configurations are in collision. Red dots represent colliding configurations and blue dots represent collision-free configurations. (d) Update mean and covariance of each Gaussian analytically based on the set of collision-free configurations contained in it. Update each mixture component's weight to be the proportion of the total non-colliding configurations that it contains.

Line 2 initializes a new set  $\bar{\chi}_i$  for each mixture component, which ultimately will store the configurations associated with that component's sample set,  $\chi_i$ , which are not in collision. In other words:

$$\forall i, 1 \leq i \leq N_G, (\bar{\chi}_i = \{q \in \chi_i | q \in \mathcal{C}_F\})$$

The loop during lines 5-14 computes each  $\bar{\chi}_j$ , which is updated on Line 10 whenever a new configuration from  $\chi_j$  is found to not be in collision.

The TRUNCATE\_GMM procedure also maintains several counts. Line 3 initializes the counts,  $F_i$ , for the number of non-colliding particles in each mixture sample. In other words:

$$\forall i, 1 \leq i \leq N_G, (F_i = |\bar{\chi}_i|)$$

Furthermore,  $C_c$ , which is initialized on Line 4, represents the count of all configurations across all of the samples  $[\chi_1, \dots, \chi_{N_G}]$  that were in collision. More formally:

$$C_c = N_P - \sum_{i=1}^{N_G} F_i$$

Lines 15-19 update the GMM components' means and covariances to be the sample mean and covariance of the sets  $\bar{\chi}_j$  of non-colliding points in each mixture sample. Line 18 also updates the weight of each mixture component to be the proportion of non-colliding configurations in the mixture sample across the sum of the counts of all non-colliding configurations from all the samples.

For a pictorial representation of the overall algorithm, refer to Figure 3.

#### D. Algorithm Analysis

We recognize that our algorithm may suffer computationally due to extensive time required to sample from  $\mathcal{C}$  and check for collisions on each configuration with the environment for each waypoint of a motion plan. Furthermore, the random sampling-based nature and the noise in motion and sensor information may also yield a different estimated probability of collision  $p_c$  during each time that the algorithm is executed.

Our goal in applying this algorithm is to obtain a reasonable estimate of the true probability of collision within a certain error-bound. Executing hundreds of Monte-Carlo simulations would potentially yield the true probability of collision; however, pursuing that approach would be very computationally expensive in practice.

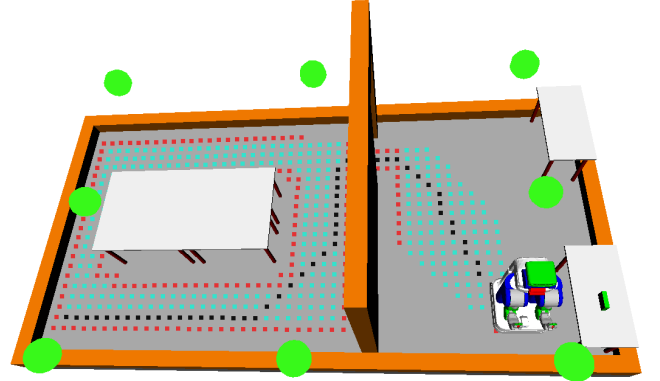


Fig. 4. Experimental Setup for Our Method with PR2 Robot. The black dots represent the nominal motion plan to be executed by the PR2. The green dots represent landmarks for the EKF localization.

Ideally, our sampling-based method would be able to execute in about the same amount of time as a single Monte-Carlo simulation while yielding the accuracy of averaging the probability of collision from running hundreds of Monte-Carlo simulations.

Suppose that  $N_p$  represents the number of samples (or the number of particles used in a Monte-Carlo simulation),  $N$  represents the number of waypoints in our motion plan, and  $N_m$  represents the number of Monte-Carlo simulations we might execute to compute the probability of collision. Our method would execute in  $\mathcal{O}(NN_p)$  time. Executing the Monte-Carlo simulations, however, would require  $\mathcal{O}(NN_pN_m)$  time. If our method is successful, it could be much more efficient than the Monte-Carlo simulation method while still retaining the same accuracy.

## V. RESULTS

We evaluate our method on a motion plan for the PR2 robot to gauge how well it performs in general. The PR2 robot navigates its environment under the guidance of noisy sensor measurements and with noise in its motion.

We design our experiments to answer the question: Can our algorithm execute in about as much time as a single Monte-Carlo simulation and obtain, on average, a reasonable estimate of the probability of collision without having to average results from executing hundreds of Monte-Carlo simulations?



In other words, we would like to see if perhaps running one execution of our method can on average be more accurate than running a single Monte-Carlo simulation.

We initialize our method with a nominal motion plan computed using an A\* planner. We tested our C++ implementation on a 4-core Intel 2.20GHz i5<sup>TM</sup> PC. We use the OpenRave Motion Planning framework and the Armadillo linear algebra library [4] for our experiments.

The experimental setup, including the environment and nominal motion plan, is depicted in Figure 4.

#### A. Models and Parameters

We consider the PR2 robot to be navigating in a 2D environment with obstacles (Figure 4). The state of the robot  $\mathbf{x} = [x, y, \theta]^T \in \mathbb{R}^3$ , consists of its position  $[x, y]$ , and its orientation  $\theta$ . The control input  $\mathbf{u} = [\delta_{rot1}, \delta_{trans}, \delta_{rot2}]^T$  represents the robot's odometry, which is encoded as a sequence of commands: a rotation, a straight-line translation, and a second rotation [5] [Chapter 5]. The motion noise is represented as  $\mathbf{m} = [\tilde{x}, \tilde{y}, \tilde{\theta}]^T \sim \mathcal{N}(0, R)$ , where  $R \in \mathbb{R}^{3 \times 3}$  represents the covariance of noise of the robot's motion in the configuration space.

The motion model of the robot is given as:

$$g(\mathbf{x}, \mathbf{u}, \mathbf{m}) = \begin{bmatrix} x + \delta_{trans} \cos(\theta + \delta_{rot1}) \\ y + \delta_{trans} \sin(\theta + \delta_{rot1}) \\ \theta + \delta_{rot1} + \delta_{rot2} \end{bmatrix} + \begin{bmatrix} \epsilon_{m1} \\ \epsilon_{m2} \\ \epsilon_{m3} \end{bmatrix}$$

where the notation  $\epsilon_i$  represents a zero-mean Gaussian noise variable with variance  $i$ . Following the notation in Thrun, 2005 [5], we can specify parameters  $[\alpha_1, \alpha_2, \alpha_3, \alpha_4]$  to encode the amount of noise in the robot's motion. Furthermore, we can compute the matrix  $R = VMV^T$ , where  $V = \frac{\partial g}{\partial \mathbf{x}}$ , and  $M = \text{diag}(\alpha_1 \delta_{rot1}^2 + \alpha_2 \delta_{trans}^2, \alpha_3 \delta_{trans}^2 + \alpha_4 \delta_{rot1}^2 + \alpha_4 \delta_{rot2}^2, \alpha_1 \delta_{rot2}^2 + \alpha_2 \delta_{trans}^2)$ .

The PR2 robot localizes itself using noisy distance measurements from eight landmarks, each landmark  $i$  being described as  $l_i = [l_x, l_y]^T$ . The sensor noise is represented as  $\mathbf{n} = [\tilde{z}] \sim \mathcal{N}(0, Q)$ , where  $Q \in \mathbb{R}^{1 \times 1}$  represents the sensor noise. These measurements are represented by the following sensor model:

$$h(\mathbf{x}, \mathbf{l}, \mathbf{n}) = \sqrt{(l_x - x)^2 + (l_y - y)^2} + \epsilon_Q$$

#### B. Analysis

We validate the accuracy of our method by first establishing a ground truth probability of collision for our nominal motion plan via a large number of Monte-Carlo simulations.

We executed 200 Monte-Carlo simulations, each simulation propagating 10,000 particles through the motion plan. The proportion of particles that experienced a collision during a single simulation would represent the estimate of the probability of collision for that simulation. The Monte-Carlo simulation results are shown in Figure 5. You can see that the range of the proportion of particles that experience a collision varies across a spectrum from 0.84 to 1. The mean of the distribution

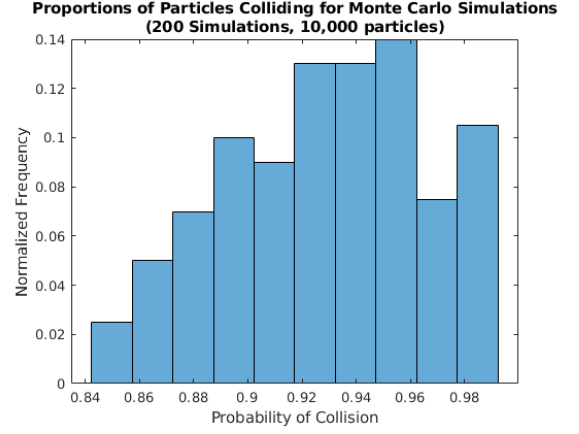


Fig. 5. Distribution of the proportions of particles colliding across 200 Monte-Carlo simulations, each simulation propagating 10,000 particles.

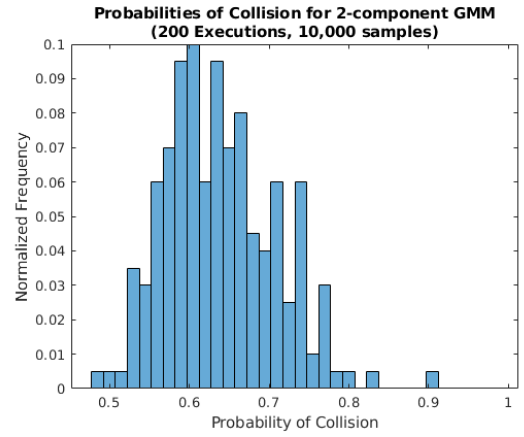


Fig. 6. Distribution of the probability of collision across 200 Monte-Carlo executions of 2-GMM, each execution taking 10,000 samples for GMM truncation.

is 0.9348, which we consider as our baseline ground-truth probability of collision. Each Monte-Carlo simulation requires 81.93 seconds on average.

We then executed our algorithm with 1, 2, and 3 mixture components. We refer to these executions as 1-GMM, 2-GMM, and 3-GMM. For each of these configurations of mixture components, we executed our algorithm 200 times and with  $N_p = 10,000$  samples and recorded the probability of collision  $p_c$  returned for each execution. Figure 6 shows the distribution of probabilities of collision obtained from executing 2-GMM. The distributions for 1-GMM and 3-GMM are similar.

TABLE I  
PROBABILITY OF COLLISION ESTIMATE FOR VARIOUS METHODS  
(200 EXECUTIONS, 10,000 SAMPLES)

	1-GMM	2-GMM	3-GMM	MC Simulation
Mean	0.6364	0.6393	0.6424	0.9348
Std.Dev	0.0699	0.0696	0.0686	0.0406
% Error	-31.92	-31.61	-31.28	0
Avg. Time(sec)	72.58	71.67	72.79	81.93

Each of the different GMM configurations are compared

in Table I alongside the Monte-Carlo simulation estimate of the probability of collision. Unfortunately, we find that our method performs poorly compared to the ground truth probability of collision deduced from the Monte-Carlo simulations. We see that our GMM method estimates the probability of collision on average to be 0.63 and this estimate does not change at all even if we alter the number of GMM components. This equates to a 31% average error from the ground truth probability of collision estimated from the Monte-Carlo simulations, 0.9348.

Our method requires slightly less time on average than a single Monte-Carlo simulation, which is expected since our algorithm has approximately the same computational complexity.

However, our method unfortunately does not provide a more accurate estimate on average compared to running a single Monte-Carlo simulation. We attempt to analyze shortcomings of our algorithm and sources of error in the following discussion section.

## VI. DISCUSSION

We note that our algorithm had attempted to estimate the probability of collision for a motion plan in about the same amount of time as a single Monte-Carlo simulation while having the accuracy of the result obtained from averaging hundreds of Monte-Carlo simulations. We assumed that propagating a GMM representation of the robot's configuration through the motion plan and truncating the GMM representation could permit us to accurately compute the conditional probability that a state of the motion plan is collision-free given that prior states in the motion plan were collision-free.

An interesting element of our results is that varying the number of components in the GMM does not seem to change the accuracy of our method. You can see in Table I that by varying the number of components from 1 to 3, the average estimated probability of collision remains at approximately 0.63. The case of 1-GMM represents the specific case of using only 1 Gaussian with a weight  $\lambda = 1$ . It seems that our method produces a consistent result despite the number of components in the GMM. This makes it evident that the GMM approach may not be the root cause of the high percent error from the MC-Simulation assessed probability of collision...as 1 Gaussian still performs as well as using 2 or 3 Gaussians for our mixture.

A key difference from our approach and the approach used by Patil et al. in [2] is that they produce a KF estimate of the deviation of the robot's configuration from the nominal configuration at a time step  $t$ . On the other hand, we use our KF to estimate the robot's true configuration. We have heard recently from a respected PhD candidate that using a KF to estimate deviations of the robot's state rather than the true robot state can be more accurate in practice due to the linearization of the deviation-expressed sensor and motion models being more stable than the linearization of the regular sensor and motion models. It could be possible that a poor KF estimate of the robot's state may be the cause of failure for our method. However, from trial and error, we tuned the

$[\alpha_1, \alpha_2, \alpha_3, \alpha_4]$  parameters from the odometry motion model and the sensor noise  $Q$  so that the KF covariance ellipses seemed to decently estimate the robot's true configuration at any time step. It is difficult to assess whether inaccuracy in the EKF estimation is the root cause of the method's poor performance.

## VII. CONCLUSION

We have presented an attempt for an algorithm for accurately and efficiently estimating the probability of collision for a motion plan. Our approach samples configurations of a robot's state from a GMM, which attempts to represent the conditional distribution of collision-free states for the robot given that prior states in the motion plan were collision-free.

Though our approach did not seem to be successful in practice, we hope that the attempt of using GMM's to represent the distribution of collision-free states can be used as inspiration for future approaches, as we believe that a single Gaussian distribution may not best represent this distribution in various scenarios. We also believe that our work provides a meaningful contribution to this field in recognizing that we do not always have knowledge of obstacle constraints and that we may only be able to use a collision checker to estimate the probability of collision.

For future work, we would hope to investigate sources of error in our algorithm. Primarily, we would look at attempts to produce a KF estimate of the deviation of a robot's configuration from the nominal configuration, rather than a KF estimate of the robot's configuration. We believe that this could be a likely issue in our approach.

We also would attempt to alter our algorithm slightly to obtain better performance. One approach in mind is to alter how we assign weights to the components of the GMM, perhaps considering looking at another metric aside from proportion of non-colliding configurations across all GMM samples to assign the  $\lambda$  weights. Another idea is to utilize approaches from machine learning, i.e. the EM algorithm, to learn a GMM representing the conditional distribution of collision-free configurations at a waypoint of a motion plan.

## ACKNOWLEDGMENT

The author would like to acknowledge Dmitry Berenson, Maani Ghaffari, and Valerie Chen for their helpful feedback.

## REFERENCES

- [1] S. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.
- [2] S. Patil, J. van den Berg, and R. Alterovitz, Estimating probability of collision for safe planning under Gaussian motion and sensing uncertainty, in ICRA, 2012, pp. 3238-3244.
- [3] R. F. Stengel, *Optimal Control and Estimation*. Dover Publications, 1994.
- [4] C. Sanderson, R. Curtin. *Armadillo: a template-based C++ library for linear algebra*. Journal of Open Source Software, Vol. 1, pp. 26, 2016.
- [5] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.
- [6] J. van den Berg, P. Abbeel, and K. Goldberg, LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information, in Int. Journal of Robotics Research, vol. 30, no. 7, 2011, pp. 895913.

**Ajaay Chandrasekaran** received the BSE degree in computer science from the University of Michigan, Ann Arbor, MI, in 2017. He is currently pursuing the MSE degree in electrical and computer engineering from the University of Michigan.