Collision Checking for Path-Planning in Stochastic Environment

1 Problem formulation

Let C_1 and C_2 be two nodes in an n dimensional configuration space χ. The covariances at the nodes are P_1 and P_2 respectively. Let χ_{obs} be the obstacle region. If an obstacle lies within the convex set as shown in figure 1 then the robot could collide with it while traveling from C_1 to C_2. The convex set shown in figure 1 can be formulated as follows:

$$(y - C_z)^T P_z^{-1} (y - C_z) \leq 1 \tag{1}$$

where

$$\begin{gathered} y \in \mathbb{R}^n \\ z \in \mathbb{R} \\ 0 \leq z \leq 1 \\ C_z = (1 - z)C_1 + zC_2 \\ P_z = (1 - z)P_1 + zP_2 \end{gathered}$$

Using Schur complement, (1) can be written as

$$\begin{bmatrix} 1 & (y - C_z)^T \\ y - C_z & P_z \end{bmatrix} \geq 0 \tag{2}$$

The obstacle region is given by

$$\begin{gathered} Ay_{obs} \leq b \\ y_{obs} \in \chi_{obs} \\ A \in \mathbb{R}^{m \times n} \\ b \in \mathbb{R}^m \end{gathered} \tag{3}$$

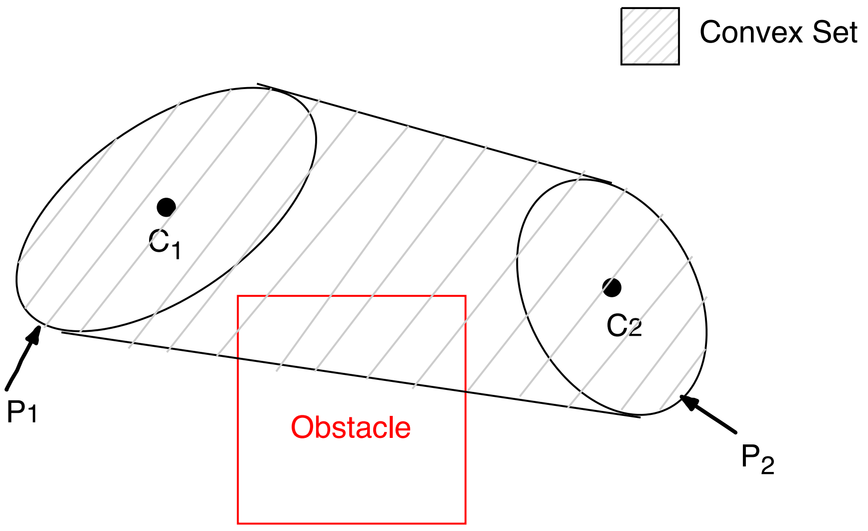

Figure 1: The convex set between two nodes

The collision checking problem can be formulated as a problem of determining feasibility of a system of inequalities as follows:

$$\begin{aligned} \begin{bmatrix} -1 & (1-z)C_1^T + zC_2^T - y^T \\ (1-z)C_1 + zC_2 - y & -(1-z)P_1 - zP_2 \end{bmatrix} &\preceq 0 \\ Ay - b &\le 0 \\ -z &\le 0 \\ z - 1 &\le 0 \end{aligned} \tag{4}$$

The first inequality is a generalized inequality w.r.t. the PSD cone. We can formulate (4) as an optimization problem as follows:

$$\begin{aligned} & min_{y,z}\ 0 \\ & subject\ to \begin{bmatrix} -1 & (1-z)C_1^T + zC_2^T - y^T \\ (1-z)C_1 + zC_2 - y & -(1-z)P_1 - zP_2 \end{bmatrix} \preceq 0 \\ & \qquad Ay - b \le 0 \\ & \qquad -z \le 0 \\ & \qquad z - 1 \le 0 \end{aligned} \tag{5}$$

The problem has optimal value $p^* = 0$ if constraints are feasible and $p^* = \infty$ if constraints are infeasible.

2 Method

We solve this problem using a particular interior-point algorithm, the Barrier Method. Barrier method requires a strictly feasible initial point. Since we don't know such a point, the barrier algorithm is preceded by a preliminary stage, called phase I, in which a strictly feasible point is computed (or the constraints are found to be infeasible). The strictly feasible point found during phase I is then used as the initial point for the barrier method [1].

2.1 Phase I formulation

We change (5) as follows:

$$\begin{aligned} & min_{y,z,s} \ s \\ & subject\ to \begin{bmatrix} -1 & (1-z)C_1^T + zC_2^T - y^T \\ (1-z)C_1 + zC_2 - y & -(1-z)P_1 - zP_2 \end{bmatrix} \leq s \\ & \qquad Ay - b \leq s \\ & \qquad -z \leq s \\ & \qquad z - 1 \leq s \end{aligned} \tag{6}$$

The variable $s \in \mathbb{R}$ can be interpreted as a bound on the maximum infeasibility of the inequalities. Our goal is to drive the maximum infeasibility below zero. We can choose initial $s = s_0$ such that (6) is always strictly feasible for the initial points y_0 and z_0. We can therefore apply the barrier method to solve (6). The algorithm checks the following 3 conditions:

1. If (y, z, s) is feasible for (6) with $s < 0$ then the system of inequalities (4) is feasible and collision happens

2. If for a dual feasible point of (6), the dual objective function is positive then the dual of (4) is feasible. Hence from the theorem of alternatives, (4) is infeasible and collision doesn't happen

3. If the duality gap reduces below a certain threshold after some number of iterations then we stop the algorithm and assume on the safer side that (4) is feasible and collision happens.

2.2 Barrier method

This method reformulates an inequality constrained problem as an equality constrained problem, to which Newton's method can be applied. The basic idea is to rewrite the problem (6), making the inequality constraints implicit in the objective function [1]. Let's introduce a block diagonal matrix M

$$M = diag(M_1, M_2, M_3, M_4) \tag{7}$$

where

$$M_1 = \begin{bmatrix} 1+s & y^T - (1-z)C_1^T - zC_2^T \\ y - (1-z)C_1 - zC_2 & (1-z)P_1 + zP_2 + sI \end{bmatrix} \tag{8}$$

$$M_2 = \underset{i}{diag}(s - a_i y + b_i) \tag{9}$$

$$M_3 = s + z \tag{10}$$

$$M_4 = s - z + 1 \tag{11}$$

a_i is the i^{th} row of A and b_i is the i^{th} component of b. Let

$$x = \begin{bmatrix} s \\ y \\ z \end{bmatrix} \tag{12}$$

Next, we define the logarithmic barrier function for (6) as

$$\phi(x) = -log\, det\, M \tag{13}$$

Hence the approximate reformulation of problem (6) is

$$min_x\, ts + \phi(x) \tag{14}$$

where $t > 0$ is a parameter of the barrier method that sets the accuracy of the approximation.

$$\lambda^* = \frac{1}{t}\nabla(log\, det\, M) \tag{15}$$

is the dual feasible for the problem (6). The Lagrangian

$$L(x, \lambda^*(t)) = s - \lambda^*(t)^T M \tag{16}$$

is minimized over x by $x = x^*(t)$. The dual function g evaluated at $(\lambda^*(t))$ is therefore equal to

$$\begin{aligned} g(\lambda^*(t)) &= s^*(t) - \lambda^*(t)^T M^*(t) \\ &= s^*(t) - (1/t)\nabla(log\,det\,M^*(t))^T M^*(t) \\ &= s^*(t) - (1/t)\overline{\theta} \end{aligned} \tag{17}$$

where $\overline{\theta}$ is the degree of $log\,det\,M$. Hence,

$$\overline{\theta} = (1+n) + m + 2 \tag{18}$$

$M^*(t)$ is M evaluated at $x^*(t)$.
And in the last line, we use the fact that $M^T \nabla(log\,det\,M) = \overline{\theta}$.
The duality gap is $(1/t)\overline{\theta}$. Hence, $x^*(t)$ is no more than $(1/t)\overline{\theta}$-suboptimal. $x^*(t)$ converges to an optimal point as $t \to \infty$. On the other hand, when the parameter t is large, the objective function in (14) is difficult to minimize by Newton's method, since its Hessian varies rapidly near the boundary of the feasible set. This problem can be circumvented by solving a sequence of problems of the form (14), increasing the parameter t (and therefore the accuracy of the approximation) at each step, and starting each Newton minimization at the solution of the problem for the previous value of t. We increase t until $t \geq \overline{\theta}/\epsilon$, where ϵ is a specified accuracy.

2.3 Newton's method

Now that we have converted the original problem (5) to an unconstrained optimization problem (14) we solve this problem using the Newton's method. Let

$$f(x) := ts + \phi(x) \tag{19}$$

Let's find the second-order approximation of $f(x)$ at x. Let $\overline{x}$ be a small perturbation at x and $\overline{M}$ be the corresponding perturbation at M.

$$\begin{aligned}\phi(x+\overline{x}) &= -log\,det(M+\overline{M}) \\ &= -log\,det(M^{\frac{1}{2}}(I+M^{-\frac{1}{2}}\overline{M}M^{-\frac{1}{2}})M^{\frac{1}{2}}) \\ &= -log\,det\,M - log\,det\,(I+M^{-\frac{1}{2}}\overline{M}M^{-\frac{1}{2}}) \\ &= -log\,det\,M - \sum_i log(1+\lambda_i)\end{aligned} \tag{20}$$

where λ_i is the i^{th} eigenvalue of $M^{-\frac{1}{2}}\overline{M}M^{-\frac{1}{2}}$. Using the second-order approximation in the expression above we get

$$\begin{aligned}\phi(x+\overline{x}) &\approx -log\,det\,M - \sum_i(\lambda_i - \frac{1}{2}\lambda_i^2) \\ &= -log\,det\,M - Tr(M^{-\frac{1}{2}}\overline{M}M^{-\frac{1}{2}}) + \frac{1}{2}Tr((M^{-\frac{1}{2}}\overline{M}M^{-\frac{1}{2}})(M^{-\frac{1}{2}}\overline{M}M^{-\frac{1}{2}})) \\ &= -log\,det\,M - Tr(M^{-1}\overline{M}) + \frac{1}{2}Tr(M^{-1}\overline{M}M^{-1}\overline{M})\end{aligned} \tag{21}$$

Hence,

$$\begin{aligned}f(x+\overline{x}) &\approx f(x) + t\overline{s} - Tr(M^{-1}\overline{M}) + \frac{1}{2}Tr(M^{-1}\overline{M}M^{-1}\overline{M}) \\ &:= F(\overline{x})\end{aligned} \tag{22}$$

Newton step $\overline{x}$ is the unique minimizer of the quadratic function F($\overline{x}$). Let Δx be a small perturbation at $\overline{x}$. Using the first-order approximation in the expression above we get

$$\begin{aligned}F(\overline{x}+\Delta x) &\approx F(\overline{x}) + t\Delta s - Tr(M^{-1}\Delta M) + Tr(M^{-1}\overline{M}M^{-1}\Delta M) \\ &= t\Delta s + Tr((M^{-1}\overline{M}M^{-1} - M^{-1})\Delta M)\end{aligned} \tag{23}$$

Let E, F_i and G be matrix coefficients such that

$$\overline{M} = \overline{s}E + \sum_i \overline{y}_i F_i + \overline{z}G \tag{24}$$

Hence,

$$\begin{aligned} F(\overline{x}+\Delta x) \approx F(\overline{x}) &+ [t + Tr((M^{-1}\overline{M}M^{-1} - M^{-1})E)]\Delta s \\ &+ \sum_i Tr((M^{-1}\overline{M}M^{-1} - M^{-1})F_i)\Delta y_i \\ &+ Tr((M^{-1}\overline{M}M^{-1} - M^{-1})G)\Delta z \end{aligned} \tag{25}$$

To find the minimizer of $F(\overline{x})$ we will make the first order coefficients in (23) zero. Hence,

$$t + Tr((M^{-1}\overline{M}M^{-1} - M^{-1})E) = 0 \tag{26}$$

$$Tr((M^{-1}\overline{M}M^{-1} - M^{-1})F_i) = 0 \quad \forall i \tag{27}$$

$$Tr((M^{-1}\overline{M}M^{-1} - M^{-1})G) = 0 \tag{28}$$

(24)-(26) can be written as (27)-(29)

$$\begin{aligned} &\overline{s}Tr(M^{-1}EM^{-1}E) + \sum_j \overline{y}_j Tr(M^{-1}F_jM^{-1}E) \\ &+\overline{z}Tr(M^{-1}GM^{-1}E) + t - Tr(M^{-1}E) = 0 \end{aligned} \tag{29}$$

$$\begin{aligned} &\overline{s}Tr(M^{-1}EM^{-1}F_i) + \sum_j \overline{y}_j Tr(M^{-1}F_jM^{-1}F_i) \\ &+\overline{z}Tr(M^{-1}GM^{-1}F_i) - Tr(M^{-1}F_i) = 0 \quad \forall i \end{aligned} \tag{30}$$

$$\begin{aligned} &\overline{s}Tr(M^{-1}EM^{-1}G) + \sum_j \overline{y}_j Tr(M^{-1}F_jM^{-1}G) \\ &+\overline{z}Tr(M^{-1}GM^{-1}G) + t - Tr(M^{-1}G) = 0 \end{aligned} \tag{31}$$

which can also be written as

$$\tilde{A}\overline{x} = \tilde{b} \tag{32}$$

where

$$\tilde{A} = \begin{bmatrix} Tr(M^{-1}EM^{-1}E) & Tr(M^{-1}F_1M^{-1}E) & \cdots & Tr(M^{-1}F_nM^{-1}E) & Tr(M^{-1}GM^{-1}E) \\ Tr(M^{-1}EM^{-1}F_1) & Tr(M^{-1}F_1M^{-1}F_1) & \cdots & Tr(M^{-1}F_nM^{-1}F_1) & Tr(M^{-1}GM^{-1}F_1) \\ \vdots & \vdots & & \vdots & \vdots \\ Tr(M^{-1}EM^{-1}F_n) & Tr(M^{-1}F_1M^{-1}F_n) & \cdots & Tr(M^{-1}F_nM^{-1}F_n) & Tr(M^{-1}GM^{-1}F_n) \\ Tr(M^{-1}EM^{-1}G) & Tr(M^{-1}F_1M^{-1}G) & \cdots & Tr(M^{-1}F_nM^{-1}G) & Tr(M^{-1}GM^{-1}G) \end{bmatrix} \tag{33}$$

$$\tilde{b} = \begin{bmatrix} Tr(M^{-1}E) - t \\ Tr(M^{-1}F_1) \\ \vdots \\ Tr(M^{-1}F_n) \\ Tr(M^{-1}G) \end{bmatrix} \tag{34}$$

and the solution $\overline{x}$ is

$$\overline{x} = \begin{bmatrix} \overline{s} \\ \overline{y_1} \\ \vdots \\ \overline{y_n} \\ \overline{z} \end{bmatrix} \tag{35}$$

3 Algorithm

Algorithm 1: Collision checking

```
given strictly feasible $x$, $t > 0$, $\mu > 1$,
tolerances $\epsilon_{in} > 0, \epsilon_{out} > 0, \lambda^2 > 2\epsilon_{in}, \alpha \in (0, 0.5), \beta \in (0, 1)$
while *duality gap* $= \overline{\theta}/t < \epsilon_{out}$ do
    1. Centering step: minimize $f(x) = ts - \phi(x)$:
    while $\lambda^2/2 \leq \epsilon_{in}$ do
        (i) Compute the Newton step $\overline{x}$ using (30)
        (ii) Line search:
        $r := 1$
        while $f(x + r\overline{x}) > f(x) + \alpha r \nabla f^T \overline{x}$ do
            $r := \beta r$
        end
        (iii) Update $x := x + r\overline{x}$
        (iv) if $s < 0$ then
            Collision happens; stop the algorithm
        end
    end
    2. Check the value of the dual objective function:
    if $s - (1/t)\overline{\theta} > 0$ then
        Collision doesn't happen; stop the algorithm
    end
    3. Increase $t$: $t := \mu t$
end
```

Since we start our algorithm form a primal feasible point x, at each inner step we get a primal feasible point. Hence we check if condition 1 in 2.1 is satisfied inside the inner loop. However we get a dual feasible point only at the end of each outer (centering) step. Hence we check if condition 2 in 2.1 is satisfied inside the outer loop after the centering step is completed.

4 Numerical Experiments

4.1 Accuracy

For testing the accuracy, we compared the results of our algorithm with the off-the-shelf SDP solver *sdpt3*. We generated a bunch of random collision checking problems similar to the one shown in figure 1. C_1 and C_2 are chosen as random coordinates in the space $(0,1)^2$. P_1 and P_2 are chosen as random positive-definite matrices. And the obstacle is a rectangle whose coordinates are also chosen randomly. Out of a hundred thousand problems, the results of only 2 cases did not match. This was because these 2 problems fall under the condition 3 in 2.1. But since for this condition we assume that the collision happens, the algorithm doesn't run into the danger of collision.

4.2 Speed

We ran our algorithm and *sdpt3* for solving a thousand randomly generated collision checking problems. The CPU time required for our algorithm is 41.50 *sec* whereas for *sdpt3* it is 157.33 *sec*. The SDP solvers require a Matlab interface, YALMIP from which the solvers are called. This could be one of the reasons for *sdpt3* being almost 3.8 times slower than our algorithm.

References:

1. Stephen Boyd and Lieven Vandenberghe, "Convex Optimization"