15.094J/1.142J Robust Path Planning with Obstacles

Victor Gonzalez(victorgo@mit.edu) and Johannes Norheim(norheim@mit.edu)

1 Introduction

In 2004, Hessem introduces the idea of robust path planning for convex regions; moving an autonomous vehicle(i.e a rover or UAV), from an uncertain starting region to a goal region while minimizing a cost function(i.e. fuel or energy) and limiting the probability of crashing with the borders of a convex region to a threshold δ [1]. During execution of a planned trajectory the vehicle is also susceptible to noise from the environment(e.g. wind), and therefore has a controller to "return" to the reference planned path which relies on noisy readings of it's actual state(e.g. position). Additionally, the position of obstacles could be uncertain. It is important to avoid obstacles in these problems because a collision could result in high economic and environmental costs. The first part of the problem is often called "planning" and the second "control"(both research fields of significant size). In the references[1, 2] both problems are optimized together. Blackmore further expanded the work to allow for robust path planning under obstacles(non-convex feasability regions)[2]. A notional representation of the general setup as discussed by Blackmore is shown in Figure 7, where the plan is shown by the solid line labeled *reference path for controller*.

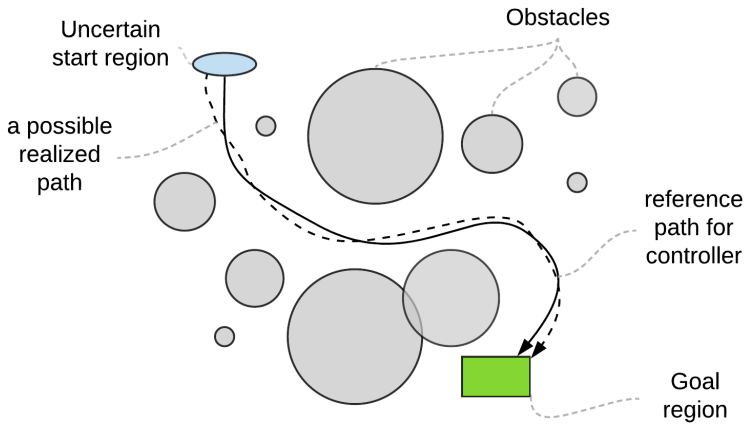

Figure 1: Schematic of *robust path planning with obstacles* problem set-up

For this project we approach the same problem from a refreshing perspective: adaptive robust optimization; the adaptivity will replace the "controller" and will be the *wait and see* decisions to be taken upon realization of the uncertainty; in the current setup we don't have any *here and now* variables, but this could be adjusted if we allowed for a more advanced scheme where we want to have a "planned" path in addition to control based on uncertain variables.

2 Problem Formulation and Model

2.1 System Dynamics

The governing equations used for the dynamics are linear:

$$\begin{aligned}\mathbf{x}_{k+1} &= A\mathbf{x}_k + B^w\mathbf{w}_k + B\mathbf{u}_k \\ \mathbf{y}_k &= C\mathbf{x}_k + D\mathbf{w}_k\end{aligned}$$

$\mathbf{x}$ refers to the position and velocity while $\mathbf{y}$ refers to the sensor readings(which is the only information available to the adaptive *controller*), $\mathbf{w}$ refers to the noise input, and $\mathbf{u}$ is the controller input(this is how we can affect the system in the optimization). For this problem, we will apply ARC bringing in uncertainty sets around the noise in the system and the starting point and use a parameter analogous to δ that controls the robustness to crashing. Our initial idea for these uncertainty sets is to use the approach based on the central limit theory. As for data, we will use the toy example developed by Blackmore in [2].

2.2 Linear Decision Rule

We use a simplified model where we set $C = I$ and $D = 0$, assuming that we have perfect information. In order to project future time steps, we use the linear decision rule for $k \geq 1$

$$\mathbf{u}_k = \alpha_k + \sum_{i=0}^{k-1} K_{k,i+1}\mathbf{w}_i$$

We do not need a linear decision rule for $\mathbf{u}_0$ because it is the first decision and does not have any information about previous uncertainty, so we will say

$$\mathbf{u}_0 = \alpha_0$$

for convenience. Plugging this into the system dynamics gives

$$\begin{aligned}\mathbf{x}_{k+1} &= A\mathbf{x}_k + B^w\mathbf{w}_k + B\left(\alpha_k + \sum_{i=0}^{k-1} K_{k,i+1}\mathbf{w}_i\right) \\ &= A\mathbf{x}_k + B^w\mathbf{w}_k + B\alpha_k + \sum_{i=0}^{k-1} BK_{k,i+1}\mathbf{w}_i\end{aligned} \tag{1}$$

for $k \geq 2$. Note that this is a recursive definition. We propose that this definition, for $k \geq 2$, is equivalent to:

$$\mathbf{x}_k = A^k\mathbf{x}_0 + \sum_{i=0}^{k-1} A^{k-i-1}B^w\mathbf{w}_i + \sum_{i=0}^{k-1} A^{k-i-1}B\alpha_i + \sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1} A^{k-j-1}BK_{j,i+1}\mathbf{w}_i \tag{2}$$

We can show this by using induction. First, observe that using (1) we get for $k = 2$:

$$\mathbf{x}_2 = A^2\mathbf{x}_0 + AB^w\mathbf{w}_0 + B^w\mathbf{w}_1 + AB\alpha_0 + B\alpha_1 + BK_{1,1}\mathbf{w}_0$$

By plugging in our recursive definition, we get:

$$\begin{aligned}\mathbf{x}_2 &= A\mathbf{x}_1 + B^w\mathbf{w}_1 + B\alpha_1 + BK_{1,1}\mathbf{w}_1 \\ &= A(A\mathbf{x}_0 + B^w\mathbf{w}_0 + B\alpha_0) + B^w\mathbf{w}_1 + B\alpha_1 + BK_{1,1}\mathbf{w}_0 \\ &= A^2\mathbf{x}_0 + AB^w\mathbf{w}_0 + AB\alpha_0 + B^w\mathbf{w}_1 + B\alpha_1 + BK_{1,1}\mathbf{w}_0\end{aligned}$$

Applying (2) for $k = 2$ we confirm that we get the same result. Now, assume that for some $k \geq 2$ that

$$\mathbf{x}_k = A^k\mathbf{x}_0 + \sum_{i=0}^{k-1} A^{k-i-1}B^w\mathbf{w}_i + \sum_{i=0}^{k-1} A^{k-i-1}B\alpha_i + \sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1} A^{k-j-1}BK_{j,i+1}\mathbf{w}_i$$

Then, we can compute

$$\begin{aligned}\mathbf{x}_{k+1} &= A\mathbf{x}_k + B^w\mathbf{w}_k + B\alpha_k + \sum_{i=0}^{k-1} BK_{k,i+1}\mathbf{w}_i \\ &= A\left(A^k\mathbf{x}_0 + \sum_{i=0}^{k-1} A^{k-i-1}B^w\mathbf{w}_i + \sum_{i=0}^{k-1} A^{k-i-1}B\alpha_i + \sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1} A^{k-j-1}BK_{j,i+1}\mathbf{w}_i\right) \\ &+ B^w\mathbf{w}_k + B\alpha_k + \sum_{i=0}^{k-1} BK_{k,i+1}\mathbf{w}_i \\ &= A^{k+1}\mathbf{x}_0 + \sum_{i=0}^{k-1} A^{k-i}B^w\mathbf{w}_i + \sum_{i=0}^{k-1} A^{k-i}B\alpha_i + \sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1} A^{k-j}BK_{j,i+1}\mathbf{w}_i + B^w\mathbf{w}_k + B\alpha_k + \sum_{i=0}^{k-1} BK_{k,i+1}\mathbf{w}_i \\ &= A^{k+1}\mathbf{x}_0 + \sum_{i=0}^{k} A^{k-i}B^w\mathbf{w}_i + \sum_{i=0}^{k} A^{k-i}B\alpha_i + \sum_{i=0}^{k-2}\left(\sum_{j=i+1}^{k} A^{k-j}BK_{j,i+1}\mathbf{w}_i\right) + BK_{k,k}\mathbf{w}_{k-1} \\ &= A^{k+1}\mathbf{x}_0 + \sum_{i=0}^{k} A^{k-i}B^w\mathbf{w}_i + \sum_{i=0}^{k} A^{k-i}B\alpha_i + \sum_{i=0}^{k-1}\sum_{j=i+1}^{k} A^{k-j}BK_{j,i+1}\mathbf{w}_i\end{aligned}$$

competing the proof by induction.

2.3 Obstacle model

Since we have obstacles in $\mathbb{R}^2$, our feasible region will be non-convex. In order to accommodate for this we use binary variables. Suppose that we have convex shaped obstacles $P^1, \ldots, P^m$ where obstacle P^i is defined by the intersection of half-spaces:

$$P^i = \left\{\mathbf{x} : \bar{\mathbf{a}}_{\mathbf{j}}^{(\mathbf{i})^{\mathrm{T}}}\mathbf{x} \leq b_j^{(i)} \quad \forall j = 1, ..., \left|P^i\right|\right\}$$

If we define binary variables $y_j^{(i)}$ and sufficiently sized M, we create a big-M formulation with binary variables $y_j^i \in \{0, 1\}$ to impose that $\mathbf{x}$ is in the non-convex feasible region:

$$\begin{aligned}\bar{\mathbf{a}}_{\mathbf{j}}^{(\mathbf{i})^{\mathrm{T}}}\mathbf{x} + My_j^{(i)} &\geq b^{(i)} \\ \sum_{j=1}^{|P^i|} y_j^{(i)} &\leq |P^i| - 1\end{aligned} \tag{3}$$

This relationship forces $\mathbf{x}$ to violate at least one of the constraints defining the space inside the obstacle, meaning that we are outside it, in the feasible region.

We can transform (3) to have the inequality constraint in the canonical direction seen for uncertainty analysis:

$$-\bar{\mathbf{a}}_{\mathbf{j}}^{(\mathbf{i})^{\mathrm{T}}}\mathbf{x} - My_j^{(i)} \leq -b_j^{(i)} \tag{4}$$

When we apply (4) to $\mathbf{x_k}$ under uncertainty $\mathbf{w_k}$ we get the robust constraint:

$$-\bar{\mathbf{a}}_{\mathbf{j}}^{(\mathbf{i})^{\mathrm{T}}}\mathbf{x_k}(\mathbf{w_1}, \ldots, \mathbf{w_k}) - My_j^{(i)} \leq -b_j^{(i)} \quad \forall \quad \mathbf{w}_l \in \mathcal{U} \quad \text{for} \quad l = 1, \ldots, \mathbf{k}$$

In order to be robust, we need

$$\max_{\mathbf{w_1},\ldots,\mathbf{w_k}} \left\{-\bar{\mathbf{a}}_{\mathbf{j}}^{(\mathbf{i})^{\mathrm{T}}}\mathbf{x_k}(\mathbf{w_1}, \ldots, \mathbf{w_k})\right\} - My_j^{(i)} \leq -b_j^{(i)} \quad \text{for} \quad k = 2, \ldots, N \tag{5}$$

2.4 Robustness

We choose for this problem the ball uncertainty set for all $\mathbf{w}_l$:

$$\mathcal{U} = \{\mathbf{w} : \|\mathbf{w}\|_2 \leq \rho\}$$

This choice lies in the intuition that at every point in time the wind could blow from any direction, and we don't want to bias our solution against wind with certain directionality unless we have further knowledge about the expected winds. Suppose that we have some vector $\bar{a}$. If we want to maximize $-\bar{\mathbf{a}}^{\mathrm{T}}\mathbf{x_k}$, we get

$$\begin{aligned}
\max_{\mathbf{w}} -\bar{a}^{\mathrm{T}}\mathbf{x_k} &= \max_{\mathbf{w}} -\bar{a}^{\mathrm{T}}\left(A^k\mathbf{x}_0 + \sum_{i=0}^{k-1} A^{k-i-1}B^w\mathbf{w}_i + \sum_{i=0}^{k-1} A^{k-i-1}B\alpha_i + \sum_{i=0}^{k-2}\sum_{j=i+1}^{k-1} A^{k-j-1}BK_{j,i+1}\mathbf{w}_i\right) \\
&= -\bar{a}^{\mathrm{T}}\beta_k + \max_{w}\left\{-\sum_{i=0}^{k-2}\left(\bar{a}^{\mathrm{T}}A^{k-i-1}B^w\mathbf{w}_i + \sum_{j=i+1}^{k-1}\bar{a}^{\mathrm{T}}A^{k-j-1}BK_{j,i+1}\mathbf{w}_i\right) - \bar{a}^{\mathrm{T}}B^w\mathbf{w}_{k-1}\right\} \\
&= -\bar{a}^{\mathrm{T}}\beta_k + \max_{w}\left\{-\sum_{i=0}^{k-2}\left(\bar{a}^{\mathrm{T}}A^{k-i-1}B^w + \sum_{j=i+1}^{k-1}\bar{a}^{\mathrm{T}}A^{k-j-1}BK_{j,i+1}\right)\mathbf{w}_i - \bar{a}^{\mathrm{T}}B^w\mathbf{w}_{k-1}\right\} \\
&= -\bar{a}^{\mathrm{T}}\beta_k + \rho\sum_{i=0}^{k-2}\left\|\bar{a}^{\mathrm{T}}A^{k-i-1}B^w + \sum_{j=i+1}^{k-1}\bar{a}^{\mathrm{T}}A^{k-j-1}BK_{j,i+1}\right\|_2 + \rho\left\|\bar{a}^{\mathrm{T}}B^w\right\|_2 \\
&\text{where } \beta_k = A^k\mathbf{x}_0 + \sum_{i=0}^{k-1} A^{k-i-1}B\alpha_i
\end{aligned}$$

Plugging this in (5) gives

$$-\bar{a}_j^{\mathrm{T}}A^k\mathbf{x}_0 - \sum_{i=0}^{k-1}\bar{a}^{\mathrm{T}}A^{k-i-1}B\alpha_i + \rho\sum_{i=0}^{k-2}\left\|\bar{a}^{\mathrm{T}}A^{k-i-1}B^w + \sum_{j=i+1}^{k-1}\bar{a}^{\mathrm{T}}A^{k-j-1}BK_{j,i+1}\right\|_2 + \rho\left\|\bar{a}^{\mathrm{T}}B^w\right\|_2 - My_j^{(i)} \leq -b_j^{(i)}$$

Constraints of this form were used for the of the obstacle constraints. Additionally, for the goal region consider that we want to satisfy the inequality

$$\mathbf{x_G} - \epsilon \leq \mathbf{x_N}(\mathbf{w_1}, \ldots, \mathbf{w_k}) \leq \mathbf{x_G} + \epsilon \quad \forall \quad \mathbf{w}_l \in \mathcal{U} \quad \text{for} \quad l = N, \ldots, \mathbf{k}$$

Then, we can replace these constraint with

$$\beta_k + \rho \sum_{i=0}^{N-2} \left\| A^{N-i-1} B^w + \sum_{j=i+1}^{N-1} A^{N-j-1} B K_{j,i+1} \right\|_2 + \rho \left\| B^w \right\|_2 \leq \mathbf{x_G} + \epsilon$$
$$-\beta_k + \rho \sum_{i=0}^{N-2} \left\| A^{N-i-1} B^w + \sum_{j=i+1}^{N-1} A^{N-j-1} B K_{j,i+1} \right\|_2 + \rho \left\| B^w \right\|_2 \leq -\mathbf{x_G} + \epsilon$$

2.5 Objective Function

For our optimization, we wanted to minimize the amount of fuel used. This objective is

$$\min_{u_k} \quad \sum_{k=0}^{N-1} u_k$$
$$s.t. \qquad \alpha_k + \sum_{i=0}^{k-1} K_{K,i+1} \mathbf{w}_i \leq u_k \quad \forall \quad \mathbf{w_i} \in \mathcal{U} \tag{6}$$
$$-\alpha_k - \sum_{i=0}^{k-1} K_{K,i+1} \mathbf{w}_i \leq u_k \quad \forall \quad \mathbf{w_i} \in \mathcal{U} \tag{7}$$

Where the robust counterparts for (6) and (7) are:

$$\alpha_k + \rho \sum_{i=0}^{k-1} \left\| K_{K,i+1} \right\|_2 \leq u_k$$
$$-\alpha_k + \rho \sum_{i=0}^{k-1} \left\| K_{K,i+1} \right\|_2 \leq u_k$$

2.6 Probabilistic Guarantee

Suppose that we have maximum wind w_{max}, and we want the maximum probability of failure to be ε. Then, if we take

$$\rho = w_{max} \sqrt{2 \log \frac{1}{\varepsilon}}$$

Then, we have a guarantee that the system will have a violation with probability less than or equal to ε.

2.7 Tractability of the model

Note that if we have N time steps this formulation will require $\mathcal{O}(N^2)$ second order cone polytopes which can be difficult to solve for larger problems. One alternative approach that we considered was the case where we have a limited horizon for K. In this case, the constraints remain mostly the same except the sum inside the 2-norm is reduced to the relative indices j. This reduces the problem to a formulation that is of order $\mathcal{O}(N)$. We investigate the effect of this formulation on solution speed, and how well the resulting path does at avoiding obstacles.

3 Experimental Results

3.1 Model Specifications

If we say that the location of the UAV is (x, y), we can define the system as

$$\mathbf{x} = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} \ddot{\mathrm{x}} \\ \ddot{\mathrm{y}} \end{bmatrix} \quad A = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ \Delta t & 0 \\ 0 & 0 \\ 0 & \Delta t \end{bmatrix} \quad B^w = B$$

We pick a reasonable end time T depending on the size of the map and a discretization N $\Delta t = T/N$. We chose ϵ for the goal in such a way that we are guaranteed to have a feasible solution(this is first checked before optimizing the main problem). We also pick w_{wind} on the order of 0.001m/s to 0.01m/s. Finally we simulate the stochastic path a large number of times: 1000 for the plots, and 100,000 for generating the numerical results.

3.2 Small obstacle map

Before exploring larger maps we started with the simplest obstacle map, defined by two lines as seen in Figure 2 and Figure 3. For this case we set $T = 10$ and we test for small $N = 10$ and a larger $N = 30$. For each case we adapt the $w_{wind} = 0.05/10$ to be *fair*. We plot the average robust path as a red dotted line, and the deterministic path as the green dotted line. The clusters of points result from 1000 different simulated cases.

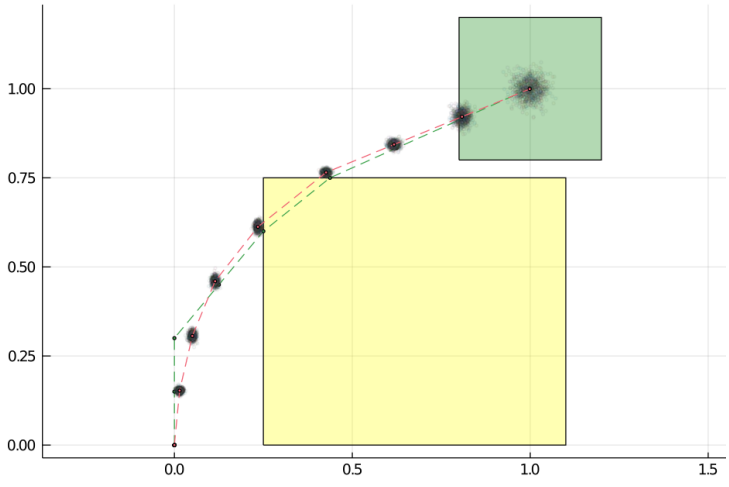

Figure 2: N=10

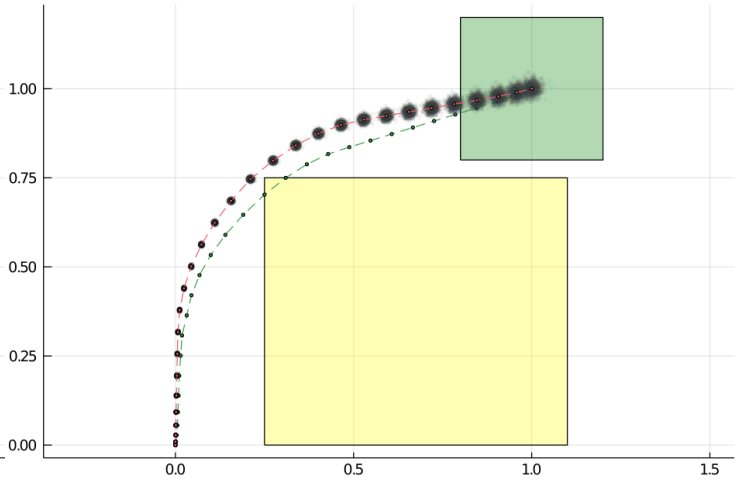

Figure 3: N=30

It is important to note that the line goes through the obstacles, which is referred to as *jumping*. However, the model satisfies the obstacle constraints at the discretization points. This is a shortcoming of the model. If we had more time, this is an area that we probably could have improved upon. On the redeeming side, as N increases, these "shortcuts" are not as big. This is because as N increases, the distance between any two discretization points decreases so it cannot make as big of a jump. Additionally, when constructing these problems researchers often add a "buffer zone" around the obstacles. Therefore, for N sufficiently large, the shortcuts should go through the buffer zone avoiding a collision. We can also observe that as we increase the discretization, the path taken by the robust solution is smoother, and actually overcomes the jumping problem described.

3.3 Larger obstacle Maps

We now take a larger map setting to test our robust formulation in a more "realistic setting". The test cases were taken from Blackmore[2], that proposed a very different approach and we wanted to be able to compare our results to his, even though the paper is now 12 years old, and computation has significantly improved, meaning that it would be unfair to compare computational times, we can still qualitatively compare our results. The only missing part between our model and Blackmore's is that they also considered uncertainty in the starting position, which we neglected in the beginning to simplify things, and did not have time to add at this later stage in the project. Here we take the analysis one step further, by also showing results when limiting the horizon to number of K's =4. Blackmore used two test cases: a simple case where we expect the robust path to be very close to the deterministic path, and a harder case, where we expect the robust path to be different from the deterministic case due to the presence of a narrow corridor on the shortest path.

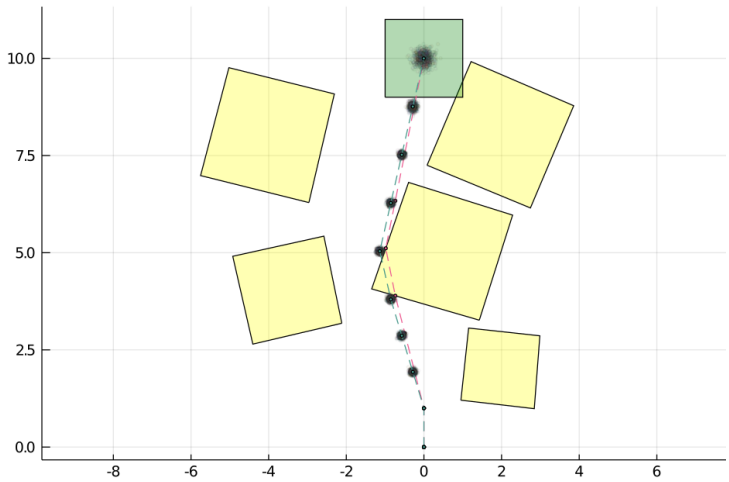

Figure 4: Simple case, based on Blackmore [2]

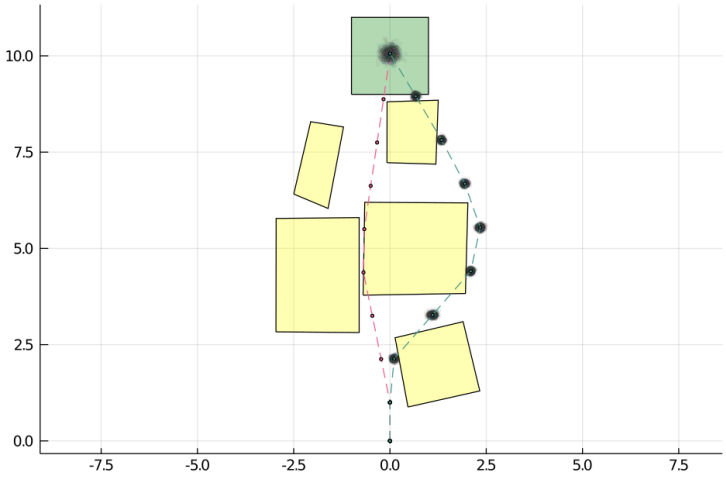

Figure 5: Harder case, with a narrow passage where it might be riskier to cross, based on Blackmore [2]

As can be seen in the figures, the robust case does in-deed take the longer, but safer route in Figure 5. These results align with the results that Blackmore produced. When we take the limited horizon approach we see something very different however:

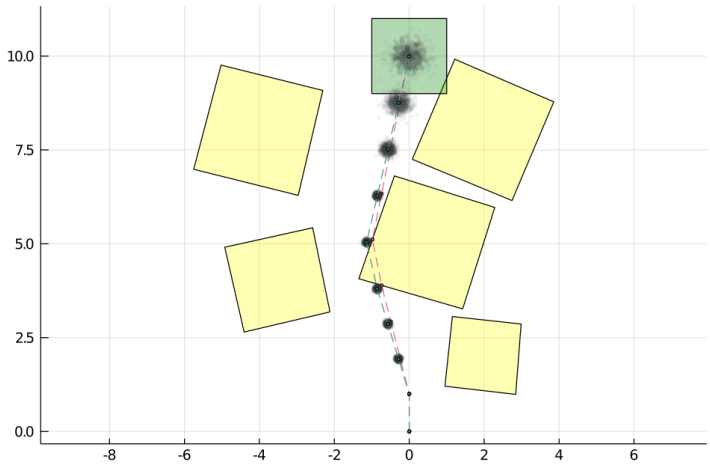

Figure 6: Simple case, now with K=4, here P(collision)=2.47×10^{-4}

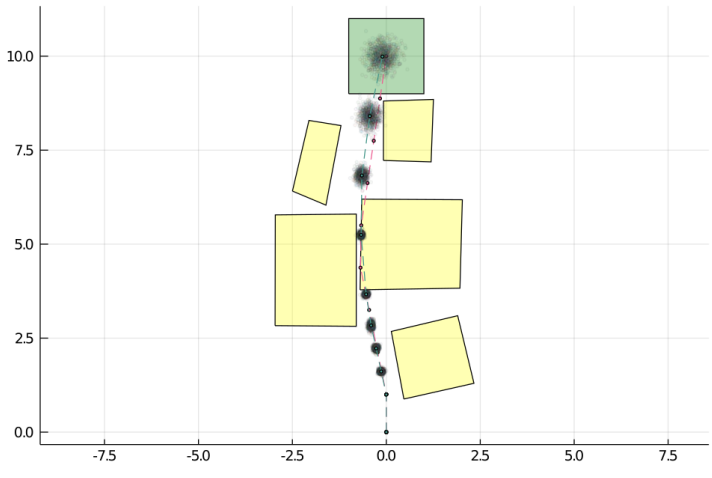

Figure 7: Harder case, now with K=4, here P(collision)=0.05 which comes from the choice of taking the riskier path

In Figure 7, the "robust" path goes through the narrow corridor to optimize the fuel use; since we are no longer using a fully adaptive linear decision rule our probabilistic guarantees are lost.

4 Computational Results

We compared the fuel requirement, run-time, and probability for the deterministic model, the full robust model, and the limited horizon model. Our results are given below

Simple case (N=10)	Min Fuel	Average case	Worst case	Run Time	Probability
Deterministic	0.269	-	-	.17s	0
Full Horizon (K=8)	1.99	0.269	0.84	95.73s	.000227
4-Horizon	1.74	0.44	0.73	30.72s	.000247
5-Horizon	1.80	0.48	0.77	182.02s	.000241
Hard case (N=10)	Min Fuel	Average case	Worst case	Run Time	Probability
Deterministic	0.19	-	-	.15s	0
Full Horizon (K=8)	2.41	0.91	1.20	283.82s	.000464
4-Horizon	2.01	0.69	1.08	68.15s	.052
5-Horizon	2.20	0.89	1.30	351.50s	.042

Some key takeaways are that the fuel and run time generally increase as we increase N, but the probability of failure goes down. We also observe that the robust case requires significantly more fuel than the deterministic case. Although we would expect runtime to always go down with shorter horizon, the reason it is not significant here is that the problem is still small enough (N=10) that it does not make a large difference.

5 Conclusions

We constructed methods that solves a path planning problem to be robust against disturbance uncertainty, in our case from wind. This method is fast and can converge to a global optimum. Some future directions include reducing the size and magnitude of jumps across obstacles. Additionally, it would be interesting to implement a folding horizon approach and see how the results compare, but we did not have the time to implement this approach.

6 Appendix

6.1 Code

6.1.1 dynamics.jl

```
using LinearAlgebra
# Create model for dynamics
function get_dynamic_matrices(T, N)
    dt = T/N
    A = zeros(4,4)+I
    A[1,2] = A[3,4] = dt
    B = zeros(4,2)
    B[2,1] = B[4,2] = dt
    #C = zeros(2,4)
    #C[1,2] = C[2,4] = 1
    #D = zeros(2,2)
    Bw = copy(B);

    Ak = zeros(N+1,size(A)...)
    Ak[1,:,:] = zeros(Int, size(A))+I
    for i=2:N+1
        Ak[i,:,:] = A*Ak[i-1,:,:]
    end
    Ai = i -> Ak[i+1,:,:]

    x_k_det = (x0, k, alpha) -> Ai(k-1)*x0+sum(Ai(k-i-2)*B*alpha[:,i+1] for i=0:k-2)
    x_k_stoc = (x0, k, alpha, K, w) -> (x_k_det(k,alpha)
        +sum((Ai(k-i-1) * Bw
            +sum(Ai(k-j-1) * B * K(j+1, i) for j=i+1:k-1))*w[:,i] for i=1:k-2)
        +Bw*w[:,k-1])

    function simulate_stoc(x0, alpha, K, w)
        x = zeros(4,N)
        x[:, 1] = x0
        for k = 2:N
            u = alpha[:, k - 1]
            for i = 1:k-2
                u += K(k, i)*w[:,i]
            end
            x[:, k] = A*x[:, k-1] + Bw*w[:,k-1] + B*u
        end
        return x
    end

    return x_k_det, simulate_stoc, Ai, B, Bw
end
```

6.2 detsolver.jl

```
Q = zeros(2,4)
Q[1,1] = Q[2,3] = -1
bigM = 100; #Big M

function detpathplanner(problem)
    Ai, Bw, B, N, x_k_det, P, q, obj, xN, epsilon_in, rho = problem
    PQ = P*Q
    Nobj = length(obj)
    m = length(q)

    model = Model(optimizer_with_attributes(
            () -> Gurobi.Optimizer(GRB_ENV), #"OutputFlag" => 0
        ));
    @variable(model, x[1:4, 1:N])
    @variable(model, u[1:2, 1:N-1])
    @variable(model, au[1:2, 1:N-1] >=0)
    @variable(model, z[1:N-1,1:m] >=0, Bin)
    @constraint(model, [k=2:N], PQ*x[:,k] .<= -q + bigM*z[k-1,:] )

    @constraint(model, [k=1:N-1, i=1:Nobj],
           sum(z[k,j] for j in obj[i]) == length(obj[i])-1) # Obstacles

    @constraint(model, x[:,1] .== x0)
    @constraint(model, x[:,N] .== xN)
    @constraint(model, [k=1:N-1], x[:,k+1] .== Ai(1)*x[:,k]+B*u[:,k])
    @constraint(model, u .<= au)
    @constraint(model, -au .<= u)
    @objective(model, Min, sum(au));
    optimize!(model)

    return value.(x)[1,:],value.(x)[3,:]
end
```

6.3 nksolver.jl

```
using JuMP, Gurobi, LinearAlgebra
GRB_ENV = Gurobi.Env();
include("utils.jl")

Q = zeros(2,4)
Q[1,1] = Q[2,3] = -1
bigM = 100; #Big M

function pathplanner(problem, npastK=0, feasability=false)
        Ai, Bw, B, N, x_k_det, P, q, obj, xN, epsilon_in, rho = problem
```

```
PQ = P*Q
Nobj = length(obj)
m = length(q)
if npastK == 0
        npastK = N-2
end
get_nkcone = get_ncone(npastK)
N_K = get_ncone(npastK)(N, npastK)

model = Model(optimizer_with_attributes(
        () -> Gurobi.Optimizer(GRB_ENV), #"OutputFlag" => 0
    ));

@variable(model, K[1:N_K, 1:2, 1:2])   # Control law linear
@variable(model, alpha[1:2, 1:N-1])      # Control law constant
@variable(model, au[1:2, 1:N-1] >=0) # Absolute value of u(control)
@variable(model, z[1:N-1, 1:m] >=0, Bin) # Binary vars for obstacles
if feasability
        @variable(model, minepsilon[1:4] >=0)
        epsilon = minepsilon
else
        epsilon = epsilon_in
end

# Variables for auxiliary second order cones:
@variable(model, t[1:N_K, 1:m] >= 0) # (we need one for each halfplane)
@variable(model, s[1:N_K, 1:2] >= 0); # (one for each control vector)
@variable(model, r[1:npastK, 1:4] >= 0) # (one for each state variable)

@constraint(model, [k=1:N-1, i=1:Nobj],
   sum(z[k,j] for j in obj[i]) == length(obj[i])-1) # Obstacles

function get_K_independent_vars(k, S=I, size=2)
        if npastK+1<=k-2
              return rho*mapslices(norm, sum(
                      (S*Ai(k-i-2)*Bw) for i=npastK+1:k-2), dims=2)[:]
        end
        return zeros(size)
end

# Obstacle free path
# Time step k=2
@constraint(model, (PQ*x_k_det(2,alpha)
        + rho*mapslices(norm, PQ*Bw, dims=2)[:]
        ) .<= -q+bigM*z[1,:])
# Time steps k=3 to N:
for k=3:N
        print("-----")
```

```
        print(size(PQ*x_k_det(k,alpha)))
        print("\n")
        print(size(get_K_independent_vars(k, PQ, 2)))
        print("\n")
        print(size(sum(rho*t[get_nkcone(k, i),:] for i=1:min(k-2,npastK))))
        print("\n")
        print(size(rho*mapslices(norm, PQ*Bw, dims=2)[:]))
        print("\n")
end
@constraint(model, [k=3:N], (PQ*x_k_det(k,alpha)
        + get_K_independent_vars(k, PQ, 2)
        + sum(rho*t[get_nkcone(k, i),:] for i=1:min(k-2,npastK))
        + rho*mapslices(norm, PQ*Bw, dims=2)[:]
        ) .<= -q+bigM*z[k-1,:])

# Maximum control (2 inequalities per k)
@constraint(model, alpha[:, 1] .<= au[:, 1])
@constraint(model, -au[:, 1] .<= alpha[:, 1])
@constraint(model, [k=2:N-1], (alpha[:, k]
    +rho*sum(s[get_nkcone(k, i),:] for i=1:min(npastK,k-2))
    ) .<= au[:,k])
@constraint(model, [k=2:N-1],  (-alpha[:,k]
    +rho*sum(s[get_nkcone(k, i),:] for i=1:min(npastK,k-2))
    ) .<= au[:,k])

# Goals (4 inequalities)
@constraint(model, (x_k_det(N, alpha)
        + get_K_independent_vars(N, I, 4)
        + sum(rho*r[i,:] for i=1:min(npastK, N-2))
        + rho*mapslices(norm, Bw, dims=2)[:]
        ) .<= xN+epsilon)
@constraint(model, (-x_k_det(N, alpha)
        + get_K_independent_vars(N, I, 4)
        + sum(rho*r[i,:] for i=1:min(npastK, N-2))
        + rho*mapslices(norm, Bw, dims=2)[:]
           ) .<= -xN+epsilon)

# Obstacles SOC
@constraint(model, [k=3:N, i=1:min(k-2, npastK), l=1:m], vcat(
        t[get_nkcone(k, i), l],
        (PQ * Ai(k-i-1) * Bw)[l,:]+sum(
        (PQ * Ai(k-j-1) * B * K[get_nkcone(j + 1, i),:,:])[l,:]
        for j=(i+1):(k-1))) in SecondOrderCone())

# Control SOC
@constraint(model, [k=3:N-1, i=1:min(k-2, npastK), l=1:2], vcat(
        s[get_nkcone(k, i), l], sum(
        K[get_nkcone(j + 1, i), l, :] for j=(i+1):(k-1)))
```

```
                in SecondOrderCone())

        # Goals SOC
        @constraint(model, [i=1:min(N-2, npastK), l=1:4], vcat(r[i, l],
                (Ai(N-i-1)*Bw)[l,:]+sum(
                (Ai(N-j-1)*B*K[get_nkcone(j + 1, i),:,:])[l,:] for j=(i+1):(N-1)))
                in SecondOrderCone())

        # Objective
        if feasability
                @objective(model, Min, sum(minepsilon))
                optimize!(model)
                return value.(minepsilon)
        else
                @objective(model, Min, sum(au))
                optimize!(model)
                K_val = value.(K)
                K_out = (i,j) -> j <= min(i-2, npastK) ?
                        K_val[get_nkcone(i, j), :, :] :
                        zeros(size(K_val)[[2, 3]])
                return K_out, value.(alpha)
        end
end
```

6.4 draw.jl

```
using Plots
rectangle(w, h, x, y) = Shape(ones(4)*x + [0,w,w,0], ones(4)*y + [0,0,h,h])

function plot_goal(xN, epsilon)
    plot!(rectangle(2*epsilon[1], 2*epsilon[3], xN[1]-epsilon[1], xN[3]-epsilon[3]),
        opacity=.5, legend=:none)
end

function plot_polygons(polygons)
    for p in polygons
        plot!(Shape(p), fillcolor = plot_color(:yellow, 0.3))
    end
end
```

6.5 utils.jl

```
accum_counter = n -> Int(n*(n+1)/2)
get_cone = (i,j) -> accum_counter(i-3)+j;
get_ncone = n -> (i,j) -> accum_counter(i-3)+j-accum_counter(max(3+n,i)-(3+n));

function check_collisions(x, y, P, q, obj)
```

```
        iterations = size(x, 1)
        Np = size(x, 2)
        faults = zeros(iterations)
        for i = 1:iterations
                for j = 1:Np
                        y_vec = [x[i,j]; y[i,j]]
                        for o in obj
                                if sum(P[o,:]*y_vec .<= q[o]) == size(o,1)
                                        faults[i] += 1
                                end
                        end
                end
        end
        return faults
end

ap = (p1,p2) -> [p1[2]-p2[2],p2[1]-p1[1]]
bp = (p1,p2) -> p2[1]*p1[2]-p1[1]*p2[2]

function makePq(points)
        s = size(points)
        s2 = zeros(Int, s[1])
        for i = 1:s[1]
                s3 = size(points[i])
                s2[i] = s3[1]
        end
        P = zeros(sum(s2), 2)
        q = zeros(sum(s2))
        idx = 1
        for i = 1:s[1]
                for (p1,p2) in zip(points[i],
                        vcat(points[i][2:end],points[i][1]))
                        #println(p1,p2)
                        P[idx,:] = ap(p1, p2)
                        q[idx] = bp(p1, p2)
                        idx += 1
                end
        end
        return (P, q)
end
```

References

[1] Hessem, D. H. V., Stochastic Inequality Constrained Closed-loop Model Predictive Control, Ph.D. thesis, Technische Universiteit Delft, Delft, The Netherlands, 2004

[2] Blackmore L., Robust Path Planning and Feedback Design under Stochastic Uncertainty, Proceedings of the AIAA Guidance, Navigation and Control Conference, 2008

[3] Deits, Robin, and Russ Tedrake. "Computing large convex regions of obstacle-free space through semidefinite programming." Algorithmic foundations of robotics XI. Springer, Cham, 2015. 109-124.