IOWA

Optimization based Receding Horizon Trajectory Generation using Bernstein Polynomials

By Rachit Singhvi

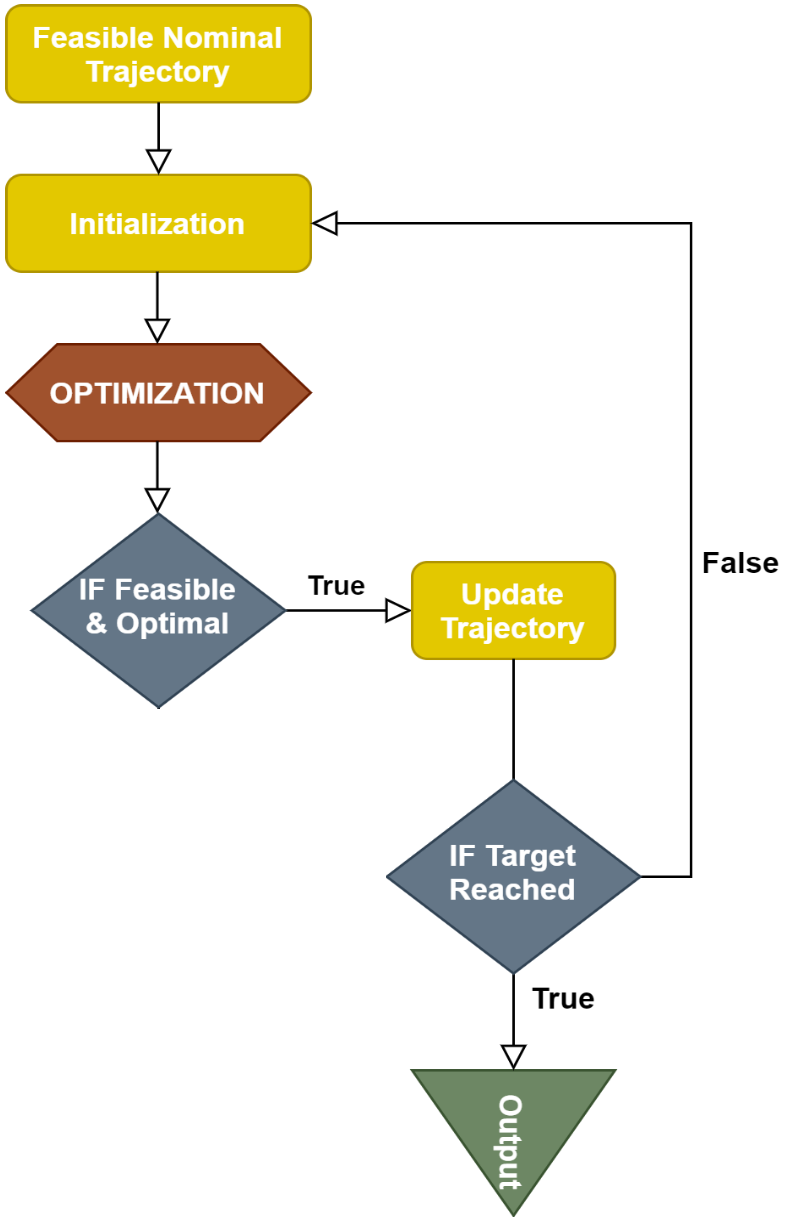
December 9, 2020

Motivation

- Trajectory generation for non-agile systems
- Limited sensing range
- Guarantees on convergence, optimality and feasibility
- Advantages of Bernstein polynomials in trajectory generation:
    - Convex hull property
    - End point property

Optimization Approach

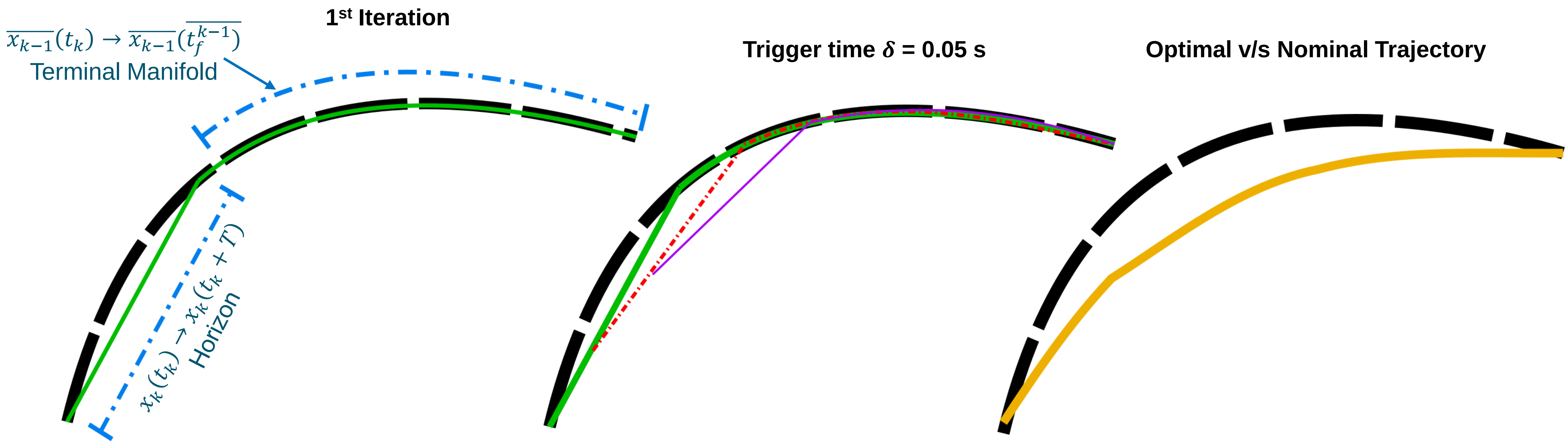
- Feasible nominal trajectory for "warm starting"
- Horizon constitutes small segment of trajectory
- ↑ Horizon size ↑ Computational expense
- Iterative optimization follows feasibility constraints
- Off-the-shelf optimizer such as fmincon, SciPy



IOWA

Receding Horizon Planner

- $[t_k, t_h]$ → Time bounds for current horizon
- $[t_h, t_f]$ → Remaining Nominal Trajectory



IOWA

Ground Robot Differential Flatness

$$\begin{cases} \dot{p}_x = v\, cos(\theta) \\ \dot{p}_y = v\, sin(\theta) \\ \quad \dot{\theta} = \ \omega \end{cases}$$

Let $y(t) = [p_x, p_y]$

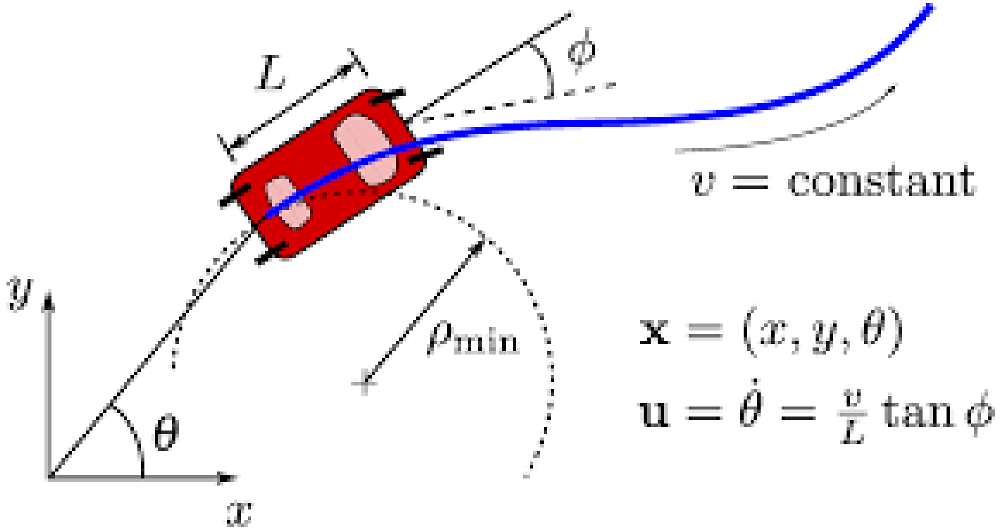
$$\frac{\dot{y}_2}{\dot{y}_1} = \frac{\cos(\theta)}{\sin(\theta)} \Rightarrow \theta = \arctan(\frac{\dot{y}_2}{\dot{y}_1})$$

Differentiating the above equation,

$$\omega(t) = \frac{\dot{y}_1(t)\ddot{y}_2(t) - \dot{y}_2(t)\ddot{y}_1(t)}{(\dot{y}_1(t))^2 + (\dot{y}_2(t))^2}$$

And,

$$v = \sqrt{\dot{y}_1 + \dot{y}_2}$$



Ground Robot Optimal Control Problem

OCP: Find $x : [t_k, t_k + T] \rightarrow \mathbb{R}^{n_x}, t \in [t_k, t_k + T]$

Cost Function

That solves $\boxed{\min_{x_k(t)} \quad J_{tot}(x_{cur}) = \int_{t_k}^{t_k+T} l(x_k(t)))dt}$

Subject to

$$\boxed{\begin{array}{ll} \dot{x}_k(t) = f(x_k(t)) & \forall t \in [t_k, t_k + t] \\ e(x_k(t_k), x_k(t_k + T), t_k + T) = 0 & \\ h(x_k(t)) \leq 0 & \forall t \in [t_k, t_k + t] \end{array}}$$

dynamic constraint (Diff Flat)

equality constraints

inequality constraints

Constraints

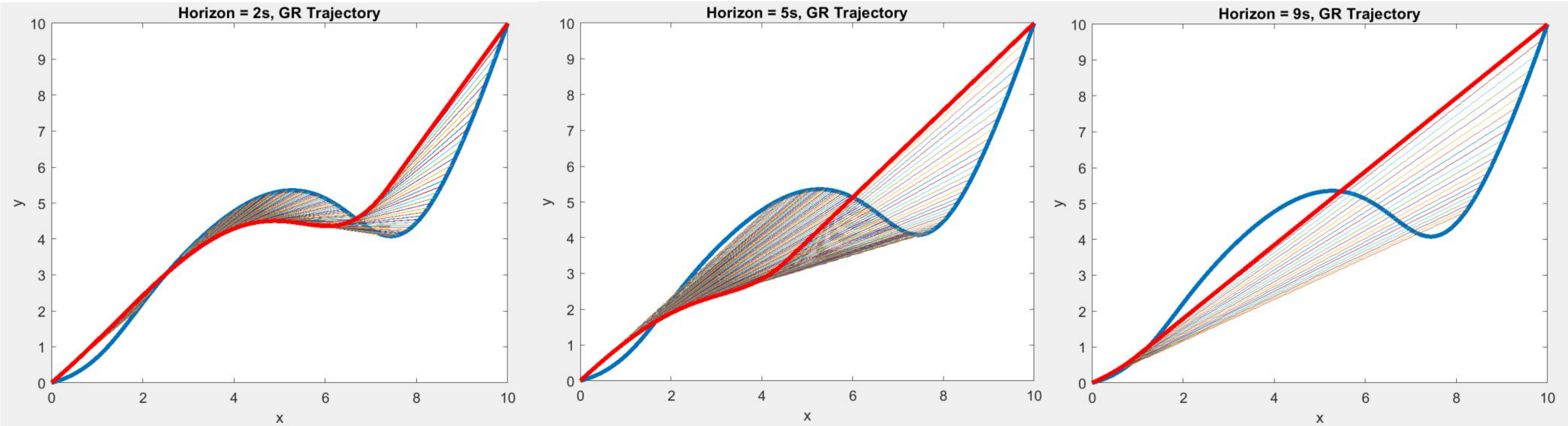
Nominal Trajectory

- Nominal trajectory created using random control points
- Feasible with respect to obstacle avoidance
- Time to complete motion on nominal trajectory: $t_f = 10\ s$

```
function [x,y] = get_nom()

%initial values
a = [0 0];
b = [10 10];

%x = linspace(a(1),b(1),4);
x = [0 1.5 2 2.5 3 3.5 6 6.5 7 7.5 10];

%y = linspace(a(2),b(2),4);
y = [0 0.5 2 5 1 7 7 7 2 1 10];

t = 0:0.1:10;

 c1 = [BernsteinPoly(x, t); BernsteinPoly(y, t)];

 plot(c1(1, :), c1(2, :));
end
```

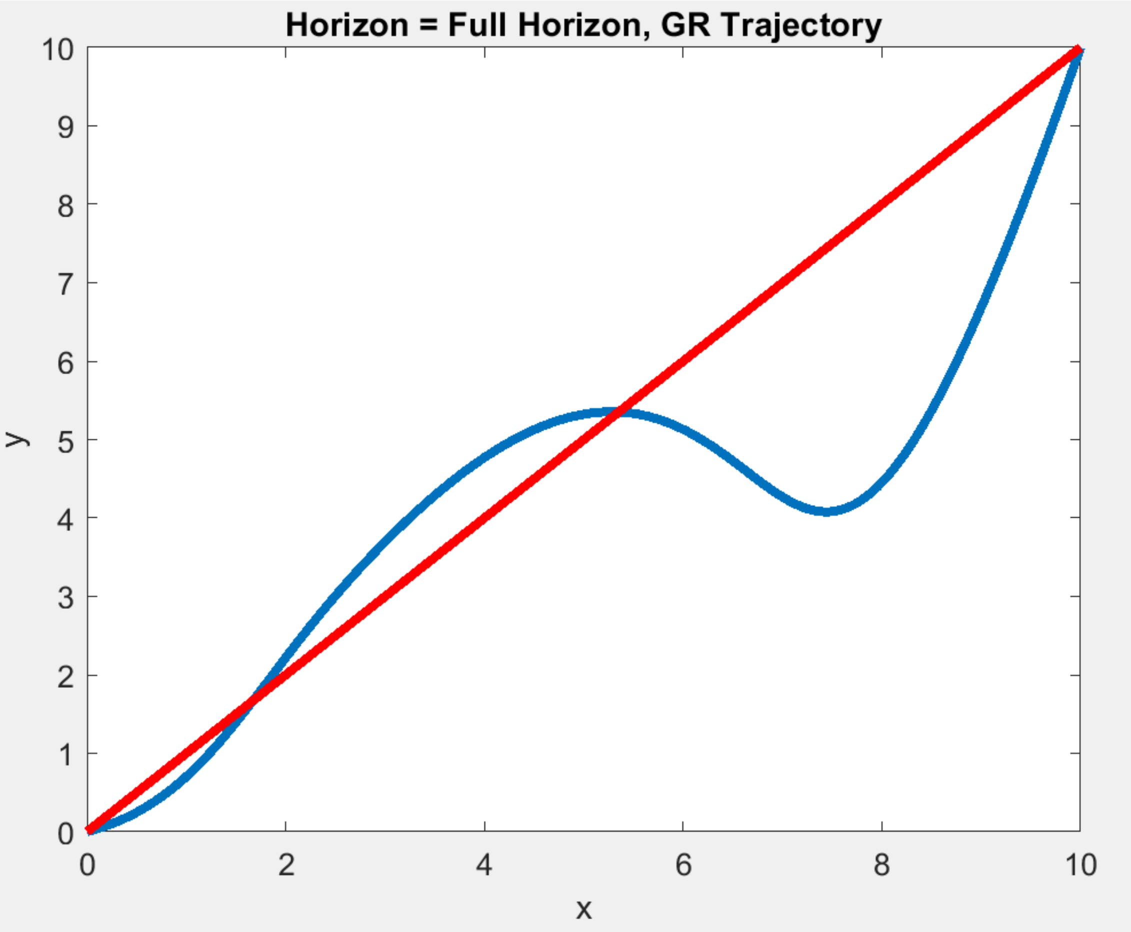
Receding Horizon Planner Trajectories

	T = 2s	T = 5s	T = 9s	Full Horizon
Trigger δ = 0.05s	**8.5015s**	**5.8258s**	**2.2645s**	**1.4642**



IOWA

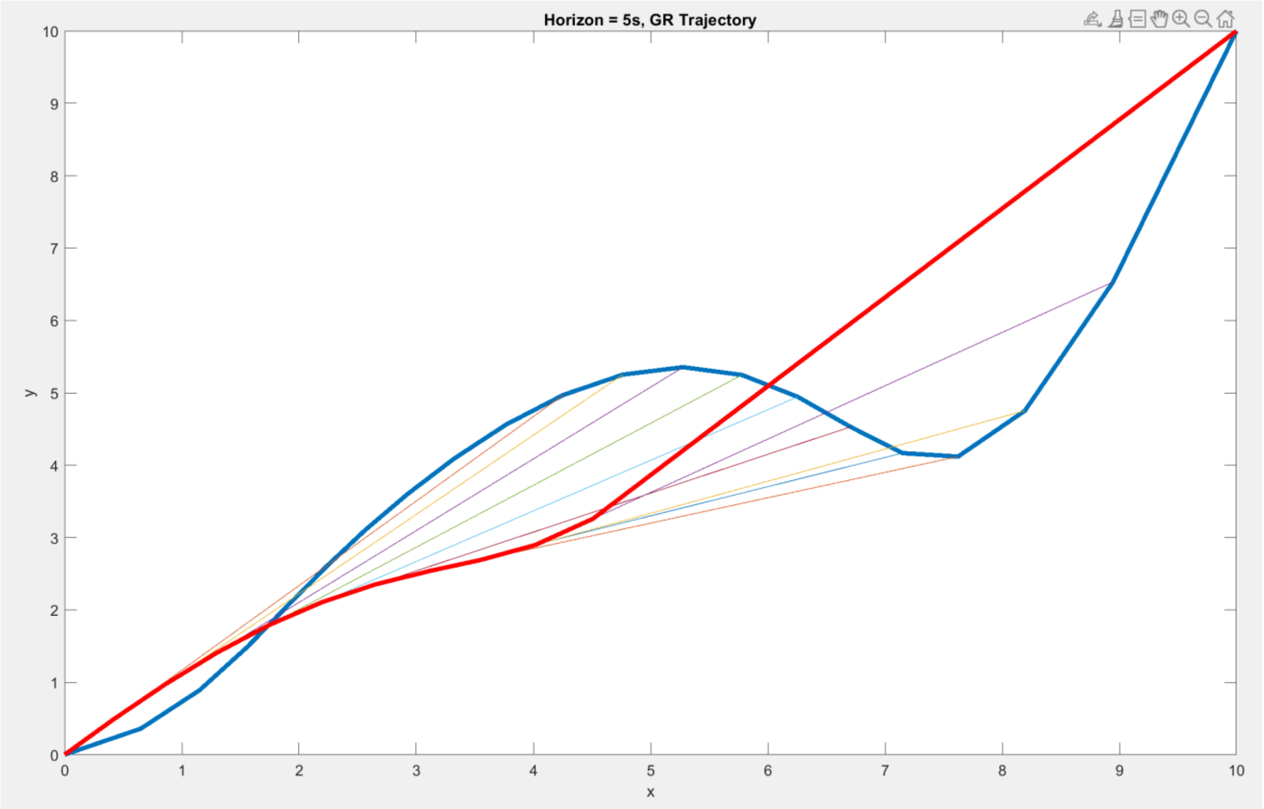
Full Horizon Trajectory



IOWA

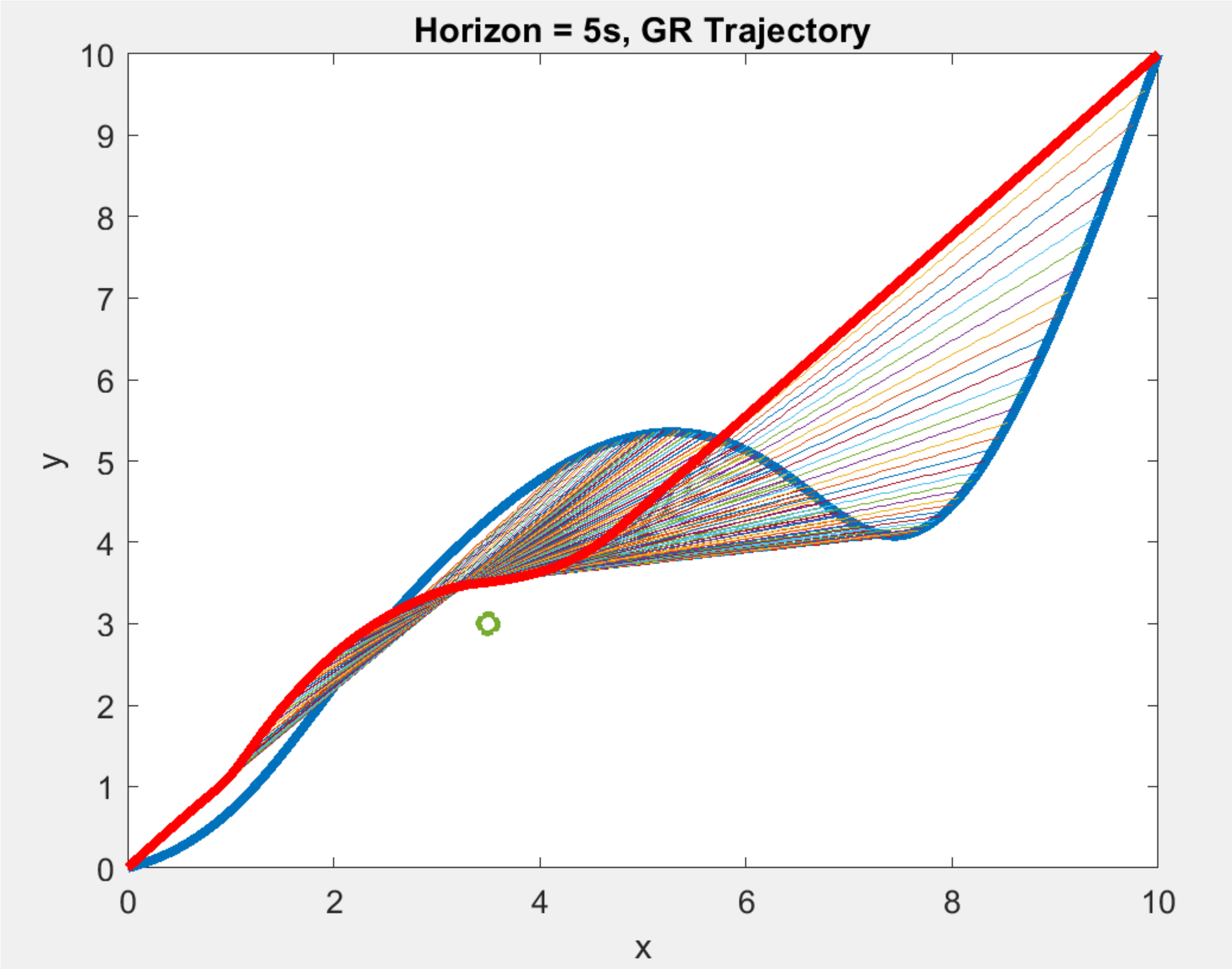
Increasing Trigger Time

- Nominal trajectory final time = 10s
- Horizon length = 5s
- Trigger time = 0.5s
- Optimal final time = 5.6126
- Number of iterations = 52



Obstacle Avoidance

- Nominal trajectory final time = 10s
- Horizon length = 5s
- Trigger time = 0.05s
- Obstacle Location = [3.5 3.0]
- Optimal final time = 5.7530



IOWA

Future Work

- Improve Obstacle avoidance
- Include guaranteed feasible nominal trajectory
- Include acceleration constraints to control smoothness at end points
- Compare FH planner against RH planner for complex paths
- Explore other horizon options such as control point based and time based

Receding Horizon Planner

```
while t_h ~= t_hp

    %Initial guess for control points and time
    x1 = linspace(P.pinit(1), P.pfin(1), N+1);
    y1 = linspace(P.pinit(2), P.pfin(2), N+1);
    T = 3;
    x0 = [x1';y1';T];

    %Fmincon Optimization
    A = []; b = []; Aeq = []; beq = []; lb = []; ub = [];
    options = optimoptions(@fmincon,'Algorithm','sqp'...
        ,'MaxFunctionEvaluations',300000);
    [x,f] = fmincon(@(x)costfun(x, P),x0,A,b,Aeq,beq,lb,ub...
        ,@(x)nonlcon(x,P),options);

    x1 = x(1:N+1)';
    y1 = x(N+2:2*N+2)';
    T = x(end);

    t = t_k:delta_t:t_h;
    c3 = [BernsteinPoly(x1, t); BernsteinPoly(y1, t)];
    plot(c3(1, :), c3(2, :)); hold on

    %Reinitialize timing variables
    [t_k,t_h,t_hp,t_total] = get_init(t_k,t_h,t_hp,t_total, delta_t);

    %Reinitialize initial and final position for next horizon
    P.pinit = [BernsteinPoly(x1,delta_t,0,t_h-t_k),...
        BernsteinPoly(y1,delta_t,0,t_h-t_k)];
    P.pfin = [BernsteinPoly(x_nom, t_h,0,t_f),...
        BernsteinPoly(y_nom, t_h,0,t_f)];

    i = i+1
    %Store data points
    points(:,i) = [BernsteinPoly(x1,delta_t,0,t_h-t_k),...
        BernsteinPoly(y1,delta_t,0,t_h-t_k)]';
end
```