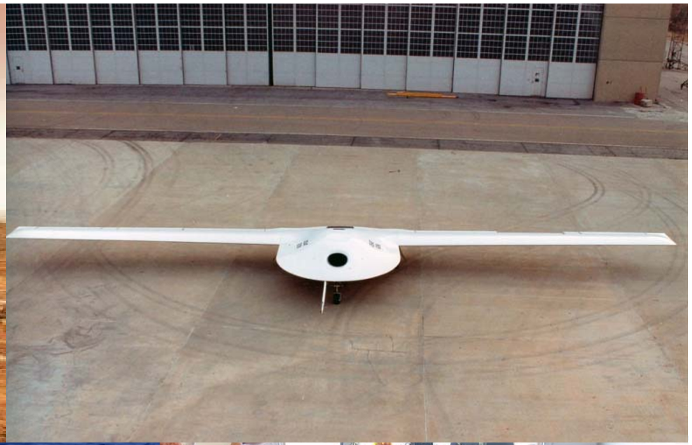
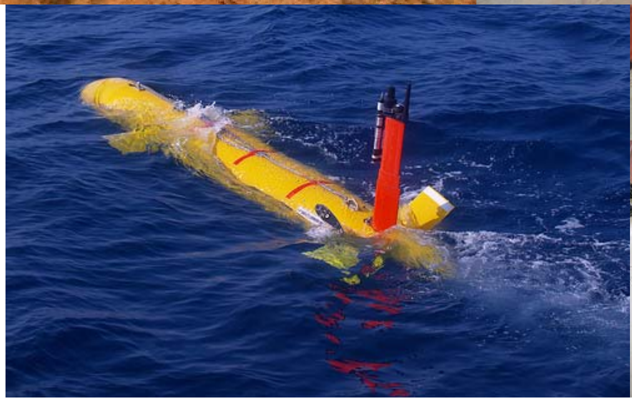
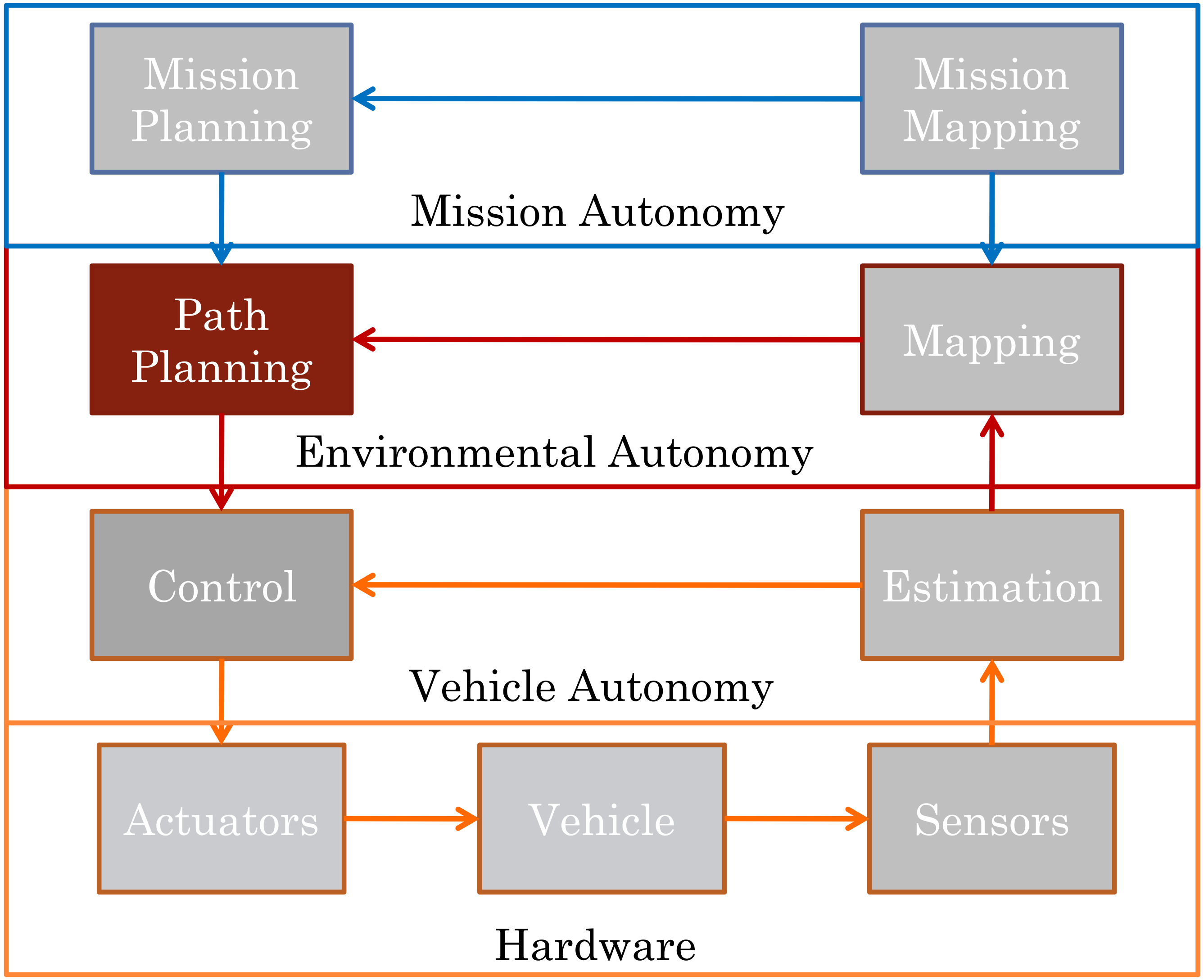


ME 597: Autonomous Mobile Robotics
Section 8 – Planning III

Prof. Steven Waslander

Components



Mission Planning
Mission Mapping
Mission Autonomy
Path Planning
Mapping
Environmental Autonomy
Control
Estimation
Vehicle Autonomy
Actuators
Vehicle
Sensors
Hardware

OUTLINE

- Optimal Planning
  - Motion Planning with Nonlinear Programming
  - Receding Horizon Planning

Optimal Planning

- Non-Linear Program (NLP)
  - (P) Convex problems are easy to solve
  - Non-convex problems harder, not guaranteed to find global optimum (local minima can occur)

$$
\begin{aligned}
\min_{x \in \mathbb{R}^n} \quad & f(x) \\
\text{s.t.} \quad & g(x) \leq 0 \\
& h(x) = 0
\end{aligned}
$$

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

$$g : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$h : \mathbb{R}^n \rightarrow \mathbb{R}^p$$

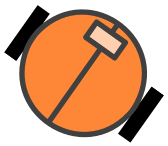
NONLINEAR PROGRAMMING

- Application to mobile robotics
  - It is possible to formulate motion planning with NLPs
    - However, a poorly formulated problem may not converge
    - Not guaranteed to find a global optimum, can be stuck in very poor solutions
  - Obstacles are particularly hard
    - Difficult for continuous algorithms to jump from one side to other
  - Initial feasible solution required, but impacts solution quality

Nonlinear Program

- Path Planning Example
  - Dynamics – our favorite two wheeled robot

$$\begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \end{bmatrix} = g(x_{t-1}, u_t) = \begin{bmatrix} x_{1,t-1} + u_{1,t} \cos x_{3,t-1} dt \\ x_{2,t-1} + u_{1,t} \sin x_{3,t-1} dt \\ x_{3,t-1} + u_{2,t} dt \end{bmatrix}$$



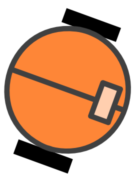
  - Initial feasible solution
    - Set velocity and turn rate to zero, hold initial position
    - Pick feasible inputs and propagate dynamics
      - Ensure constraints are not violated

Nonlinear Program

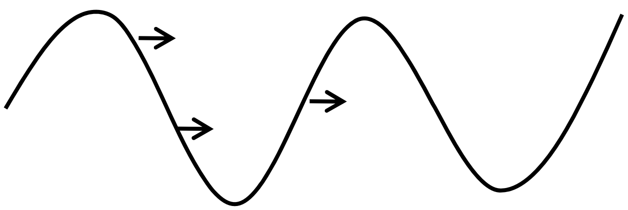
- Trajectory Tracking Example
  - Initial position

$$p_0 = [0 \quad 2 \quad 0]$$

  - Input bounds on velocity and turn rate
  - Desired trajectory



$$x^d(t) = \begin{bmatrix} t & \sin(0.3t) \end{bmatrix}^T$$



    - Heading not specified, so not penalized in cost

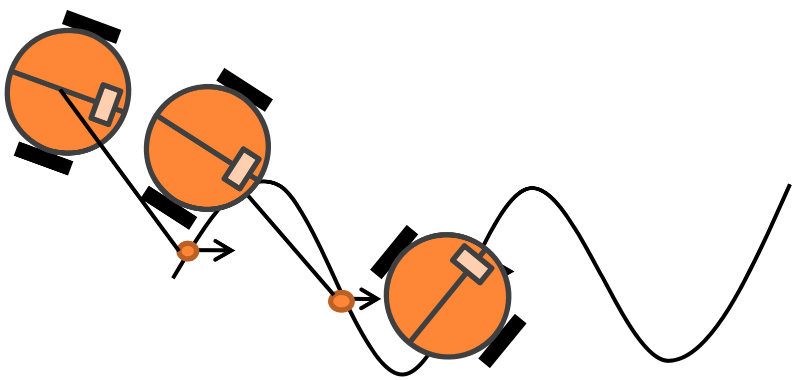
Nonlinear Program

- Trajectory Tracking Example
  - Costs
    - Quadratic deviation from desired trajectory

$$f(x) = K_d \sum_{t=1}^{T} \| x_t - x_t^d \|^2$$

    - Quadratic penalty on inputs

$$f(x) = \sum_{t=1}^{T} \left(K_{u_1} u_{1,t}{}^2 + K_{u_2} u_{2,t}{}^2 \right)$$



NONLINEAR PROGRAM

- Implementation in Matlab
  - Use fmincon function

```
[X,FVAL,EXITFLAG,OUTPUT,LAMBDA] =
  fmincon(@(x) cost(x),x0,A,B,Aeq,Beq,LB,UB,@(x)
          constraints(x), options);
```

  - Notation – defining functions for Matlab to use
    - `@(x) cost(x)` is a function handle to function `cost(x)`, which is a function of `x` (`@(x)`)

Nonlinear Program

- Implementation in Matlab
  - Must provide two functions for this optimization
    - Cost function that takes current x and returns cost

```
f = cost(x)
```

    - Nonlinear constraints function that takes x and returns g(x) and h(x) (g(x)<=0, h(x)=0)

```
[Gineq, Heq] = constraints(x)
```

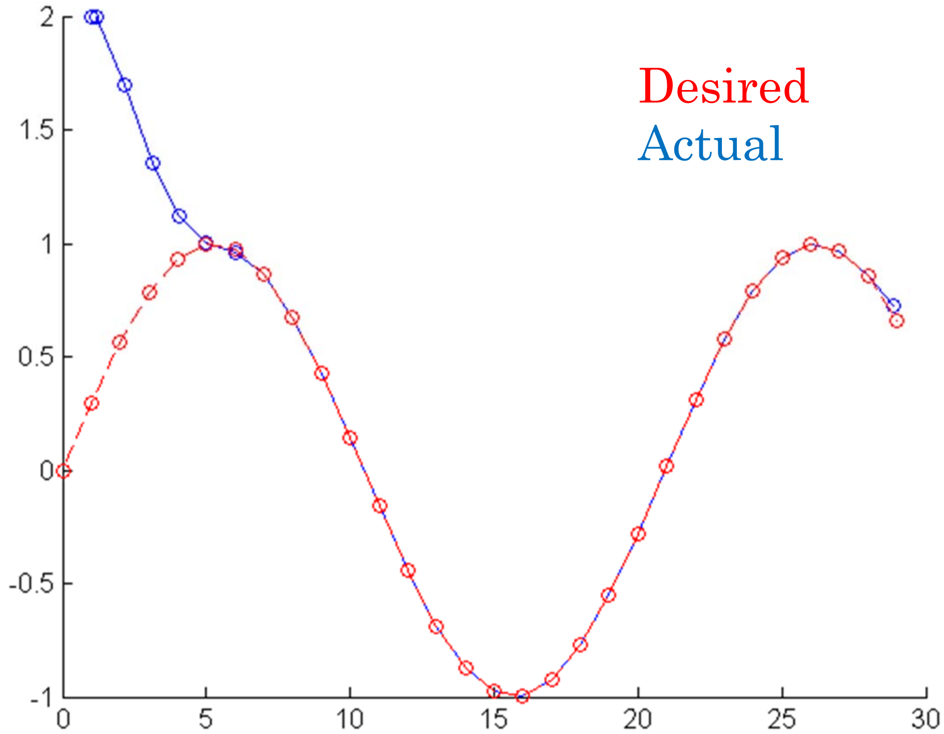
    - To provide information other than x,
      - Use global variables (declared at top of main and function)

```
global xd T dt
```

      - Pass in additional arguments to fmincon after options

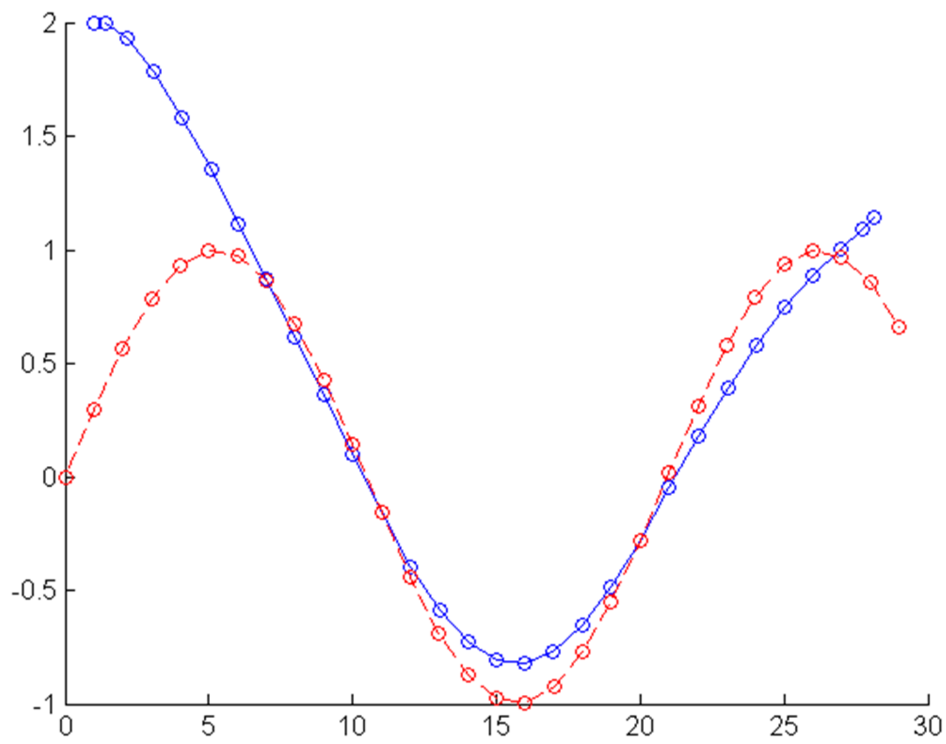
NONLINEAR PROGRAM

- Trajectory Tracking Example
  - Low weights on inputs
  - Tracks very well
  - Plans reconnect to desired trajectory nicely



NONLINEAR PROGRAM

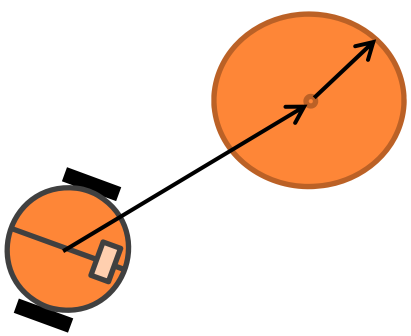
- Trajectory Tracking Example
  - Higher weights on turn rate input
  - Starts to trade off tracking and input
  - End condition has a big impact on solution

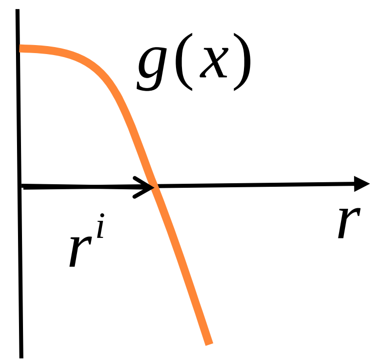


Nonlinear Program

- A big benefit of the NLP formulation is the ability to add nonlinear constraints
  - Obstacles
    - Must be defined so as to permit smooth derivatives
    - Circles work well for this
      - Define center x^i and radius r^i of circular obstacle i.

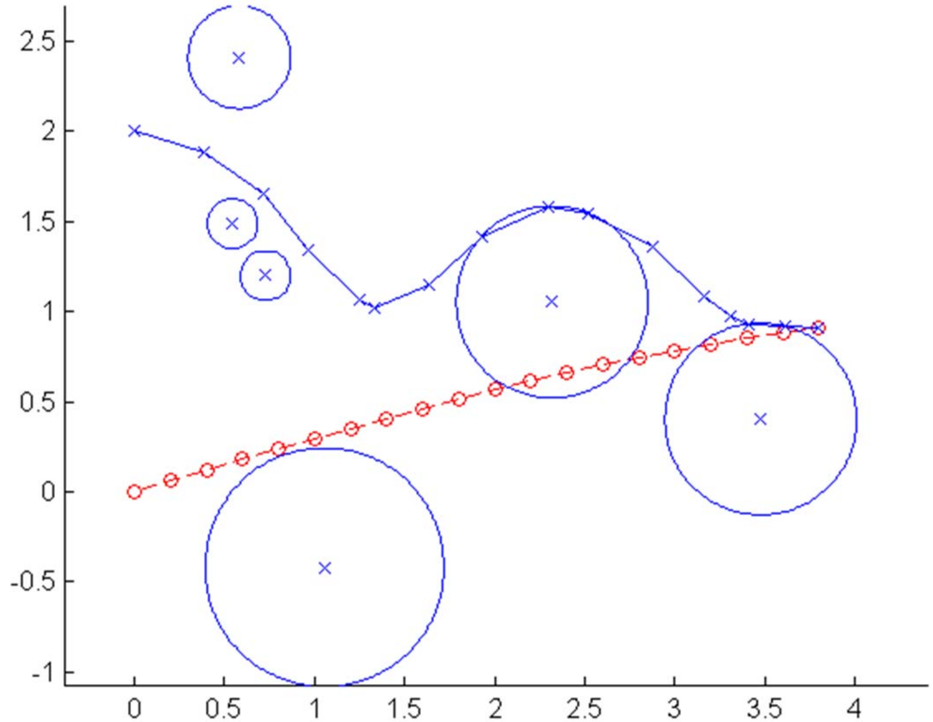
$$g(x) = \left(r^i\right)^2 - \| x_t - x^i \|^2 \leq 0$$





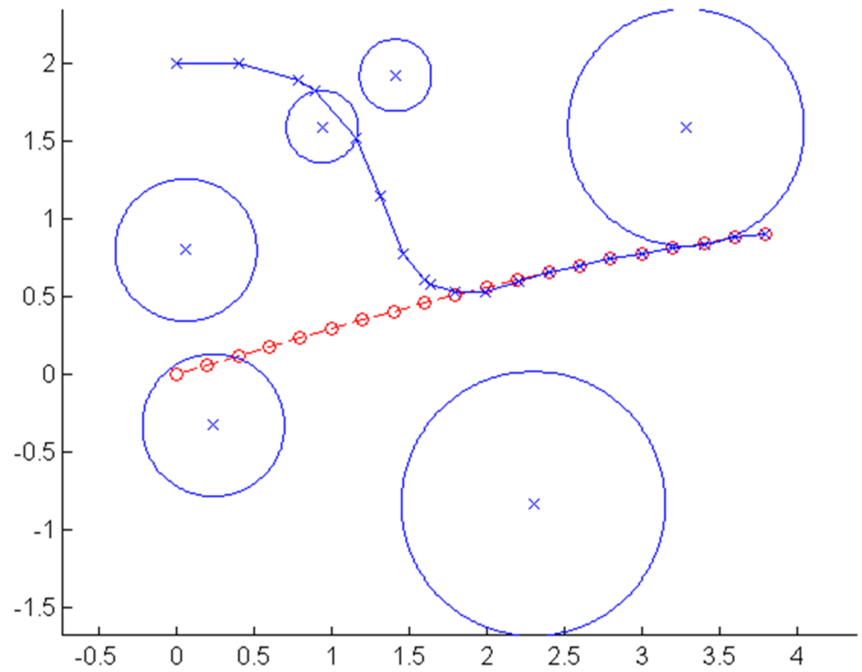
Nonlinear Program

- Trajectory Tracking with obstacles
  - 20 timesteps
  - 6 obstacles
  - Large input bounds
  - Initial conditions
    - Stay at x_0
      - v_0, $w_0 = 0$
  - Local minimum



Nonlinear Program

- Trajectory Tracking with obstacles
  - 20 timesteps
  - 6 obstacles
  - Large input bounds



- Issues
  - Discretization
  - Allowable inputs
- Solutions
  - Smaller discretization, longer computation time
  - Continuous formulation
    - single shooting, multiple shooting, collocation
    - Also enable minimum time problem formulations

RUN TIMES

- Very approximate run times
  - Based on small sample size
  - Highly dependent on problem instance for obstacles

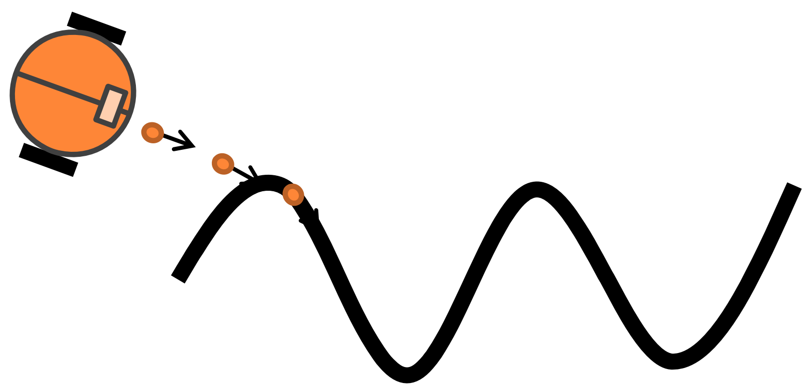
Problem	10	20	40	Comment
NLP	8 s	28 s	96 s	1 run
NLP - Obs	9 s	35 s	388 s	1 run

Outline

- Optimal Planning
  - Motion Planning with Nonlinear Programming
  - Receding Horizon Planning

Receding Horizon Approach

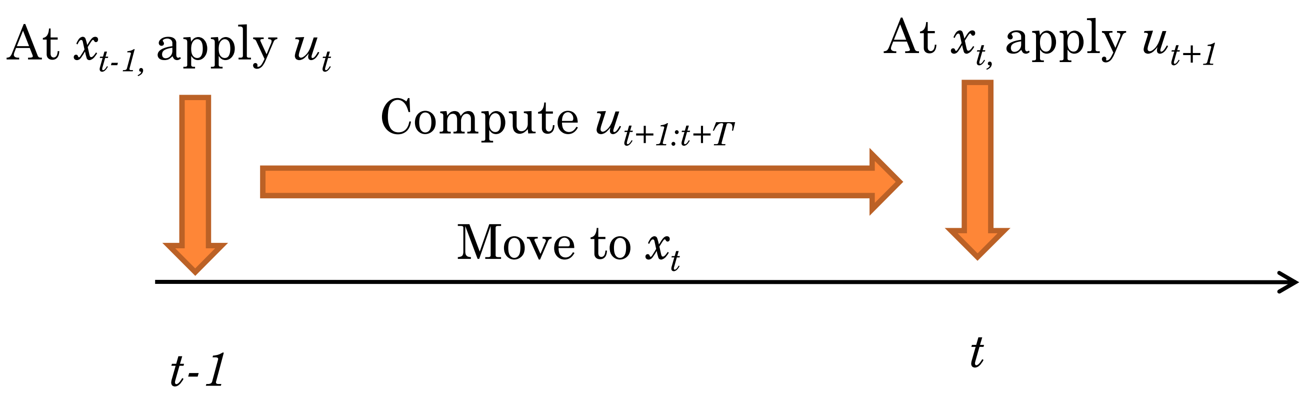
- Instead of solving for the entire plan, plan as you go along
  - Continuously use computation resources
  - Smaller optimization problem at each step
  - More susceptible to local minima
    - Escape from minima must be possible within horizon



  - Receding Horizon Control also called Model Predictive Control (MPC)

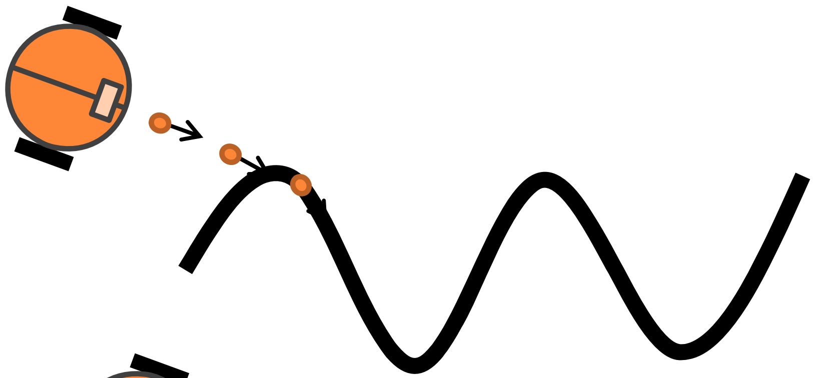
Receding Horizon Control

- Algorithm
  - Pick receding horizon length T
  - At each timestep
    - Set initial state to predicted state
    - Perform optimization over finite horizon
    - Apply control from first timestep of previous iteration
    - Predict state at next time step using motion model

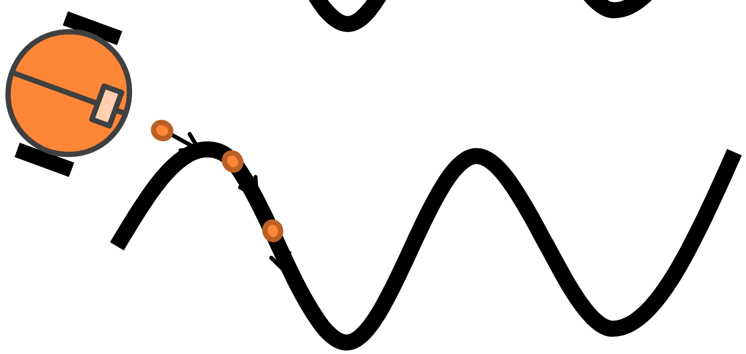


Receding horizon Control

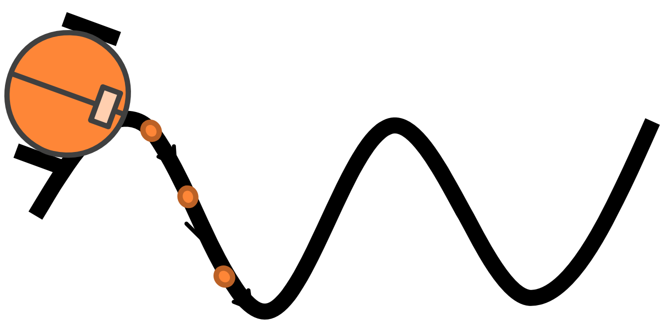
- Pictorially
  - 1)



  - 2)

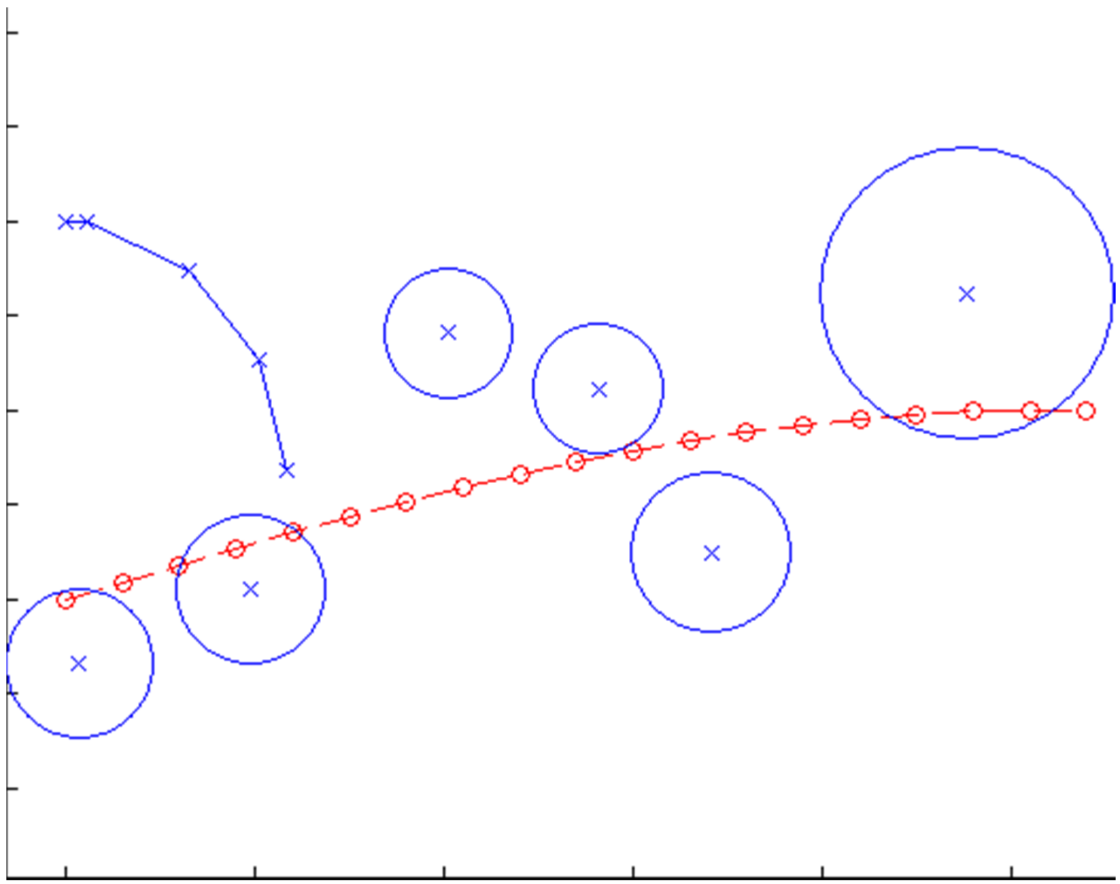


  - 3)



Receding Horizon Control

- NLP Example with RHC
  - Horizon T=5
    - 1-2 seconds per time step



Receding Horizon Control

- Comments
  - Originally developed for process control
    - 1-2 hour updates, trying to model complex chemical processes
  - Even more susceptible to local minima than full NLP
  - Since NLP complexity is roughly $O(n^3)$, this can be a big computational savings
  - All DARPA Grand and Urban challenge vehicles had some form of RHC for path planning
  - Similar to trajectory rollout
    - An optimization instead of a fixed discrete search

Extra Slides

Optimization Problem Types

- Linear Program (LP)
  - (P) Easy, fast to solve, convex

$$\begin{aligned} \min_{x \in X \subseteq \Re^n} \quad & f^T x \\ \text{s.t.} \quad & Ax \leq b \\ & A_{eq} x = b_{eq} \end{aligned}$$

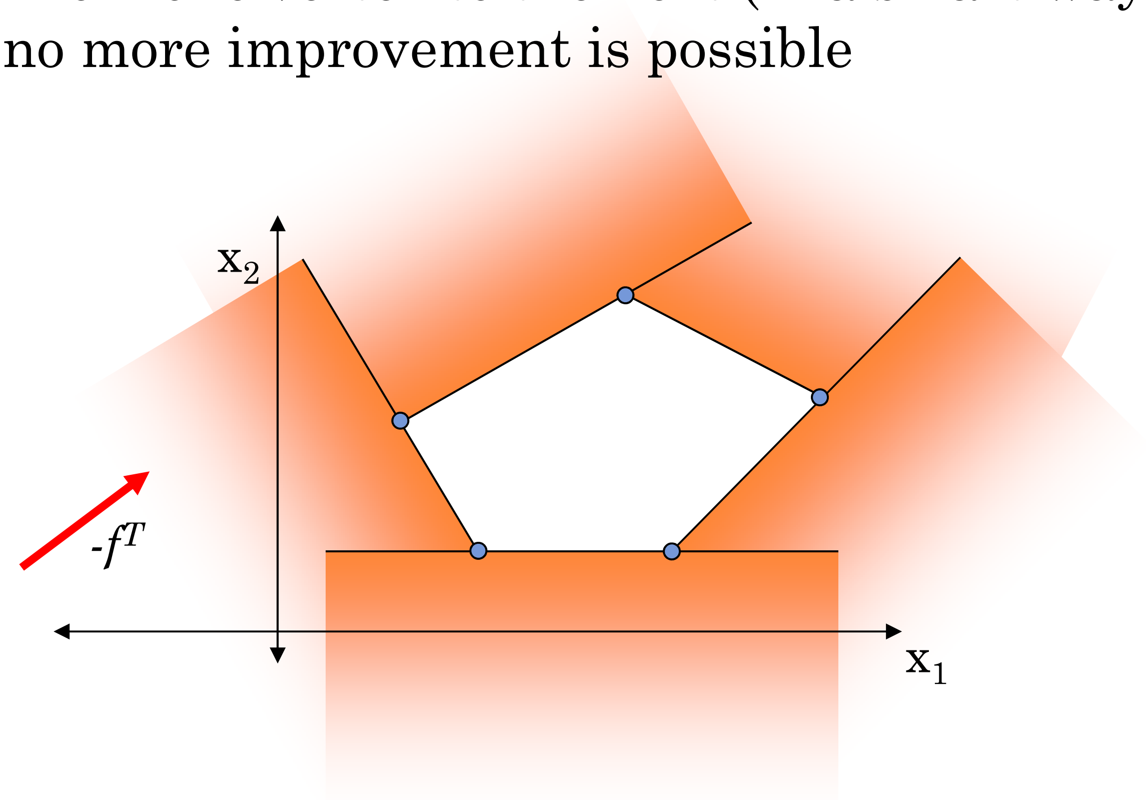
- Matlab command:

```
x = linprog(f, A, b, Aeq, beq, LB, UB, x0)
```

- Almost no planning problems are linear (trivial example in the extra slides)

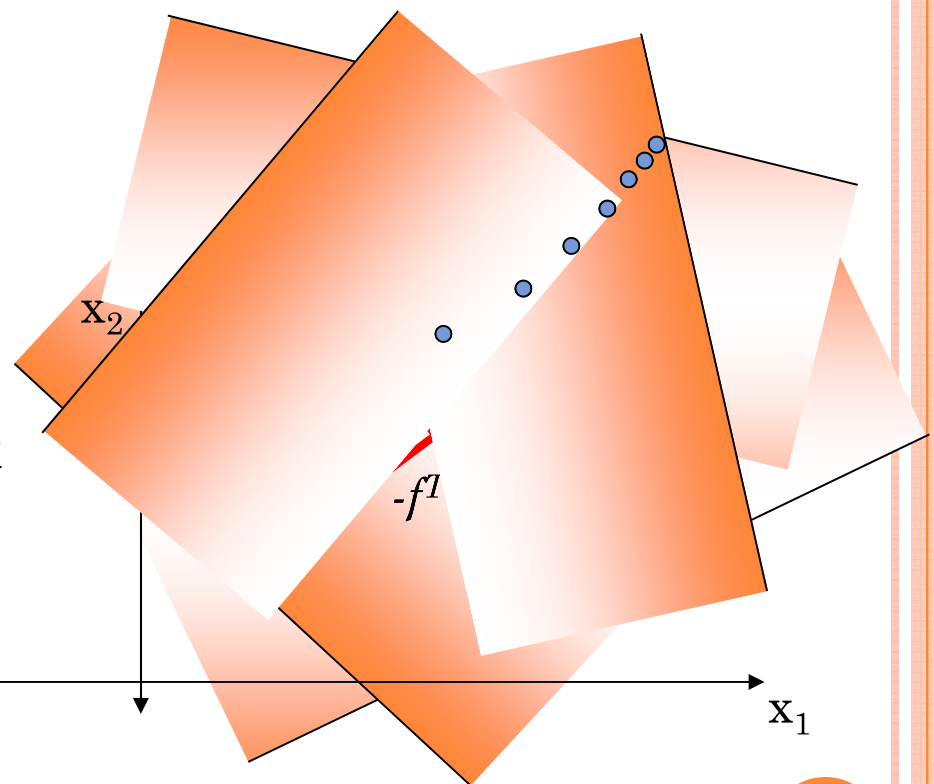
Solution Methods for Linear Programs

- Simplex Method
  - Optimum must be at the intersection of constraints
  - Intersections are easy to find, change inequalities to equalities, add slack variables
  - Jump from one vertex to the next (in a smart way), until no more improvement is possible



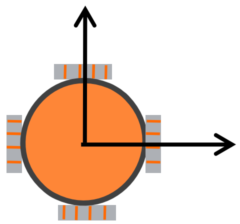
Solution Method for Linear Programs

- Interior Point Methods
  - Apply Barrier Function to each constraint and sum
  - Primal-Dual Formulation
  - Newton Step or other
  - At each iteration, increase slope of barriers
- Benefits
  - Scales better than Simplex
  - Certificate of Optimality
    - Stop whenever
    - Know how close to optimal the current solution is
    - Relies on duality



Linear Program

- Path Planning example
  - Note: It is difficult to devise a real world robotics problem that is an LP



  - Linear dynamics

$$x_t = \begin{bmatrix} 1 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 1 \end{bmatrix} x_{t-1} + \begin{bmatrix} 0 & 0 \\ dt & 0 \\ 0 & 0 \\ 0 & dt \end{bmatrix} u_t$$

$$x_t = Ax_{t-1} + Bu_t$$

Linear Program

- Path Planning example
  - Initial and final positions

$$x_0 = p_0 \qquad x_{t_F} = p_F$$

  - Minimum and maximum inputs

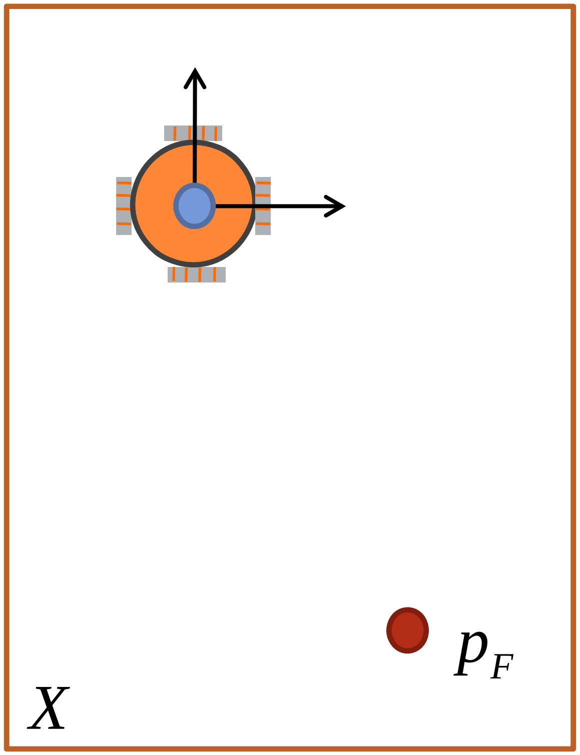
$$\underline{u} \le u_t \le \overline{u}$$

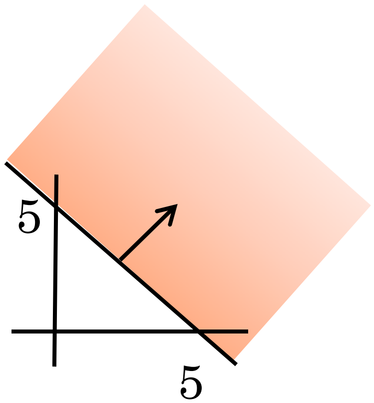
  - Minimum and maximum positions

$$x_{1:2,t} \in X$$

    - Define normal to line and offset

$$x + y \le 5$$





LINEAR PROGRAM

- Path Planning Example
  - Formulation as a Linear Program
    - Define time horizon: T
    - Define time step: dt

    - Number of states: n
    - Number of inputs: m

    - Number of optimization variables per timestep: N=n+m
    - Total number of optimization variables: M = N*T

    - Optimization vector:

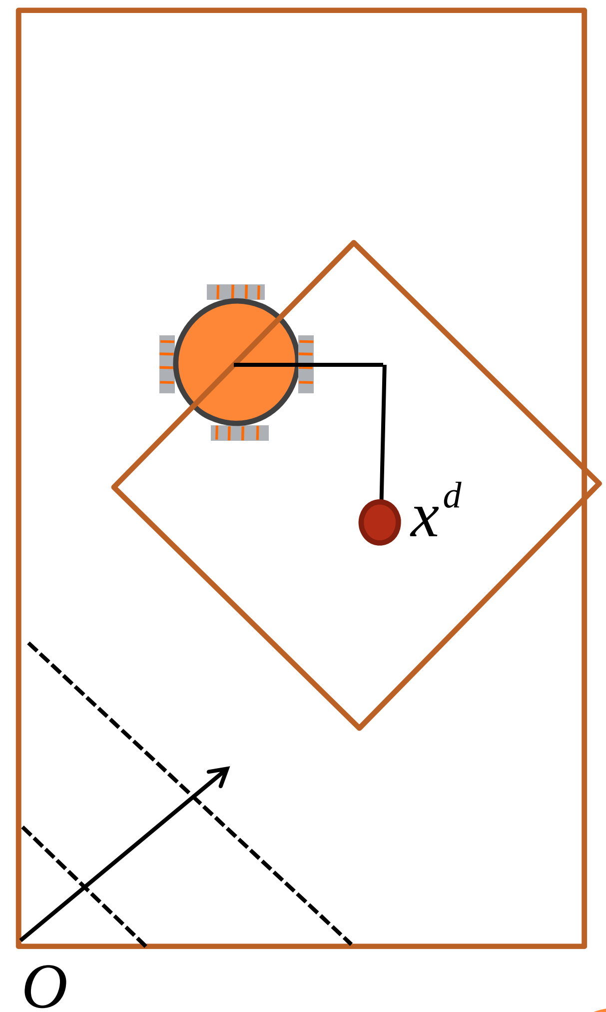
$$x = [x_0 \quad u_1 \quad \dots \quad \dots \quad x_{t_F} \quad u_{t_F+1}]^T$$

      - Extra set of inputs, constrain to zero

LINEAR PROGRAM

- Path Planning example
  - Costs
    - Must be a linear combination of states and inputs
    - Maximize x+y position (avoid origin)
    - Minimize speed
    - Minimize use of control inputs

$$f_t(x_{t-1},u_t)=\begin{bmatrix}-1 & 1 & -1 & 1 & 3 & 3\end{bmatrix}\begin{bmatrix}x_{t-1}\\ u_t\end{bmatrix}$$



    - For distance from desired, can use L_1 norm

$$\| x-x^d \|_1=\sum_{i=1}^{2}\left|x_i-x_i^d\right|$$

    - Requires transformation of variables

LINEAR PROGRAM

- Path Planning Example
  - Formulation as a Linear Program
    - Define cost:

$$f(x) = \sum_{t=1}^{T} f_t(x_{t-1}, u_t)$$

    - Define equality constraints for dynamics
      - Rewrite in standard form

$$Ax_{t-1} + Bu_t - x_t = 0$$

      - Specify for each timestep

$$Aeq = \begin{bmatrix} A & B & -I & 0 & 0 & 0 & & 0 & 0 & 0 & 0 \\ 0 & 0 & A & B & -I & 0 & & 0 & 0 & 0 & 0 \\ & & & & & & \ddots & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & & A & B & -I & 0 \end{bmatrix} \quad Beq = 0$$

Linear Program

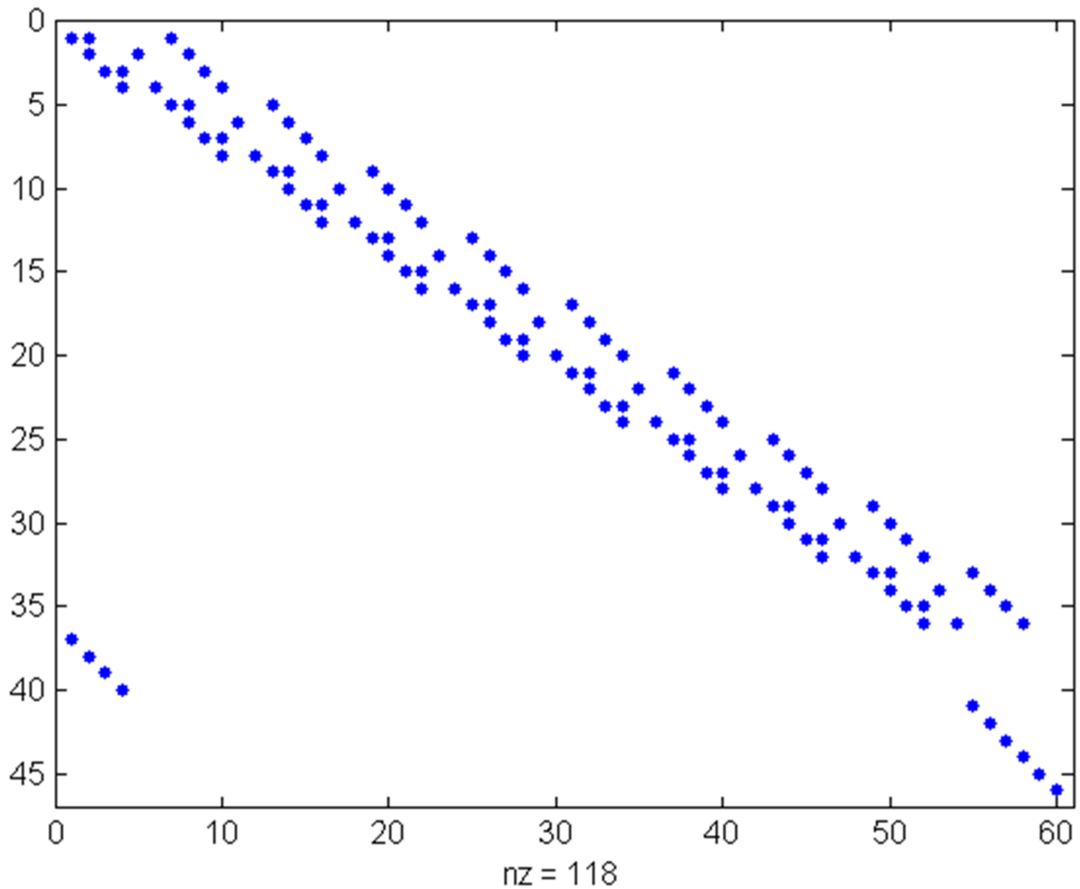
- Path Planning Example
  - Other equality constraints added to the bottom of the Aeq and Beq matrices

$$x_0 = p_0 \qquad x_{t_F} = p_F$$

  - Inequality constraints also compiled into a single Aineq, Bineq matrix pair
    - One set of constraints to add at each time step
      - Bounds on inputs
      - Bounds on state
      - Region definition

Linear Program

- Path Planning Example
  - The resulting Aeq sparsity pattern
    - spy(A) in Matlab



Linear Program

- Path Planning Example
  - Equality constraints code

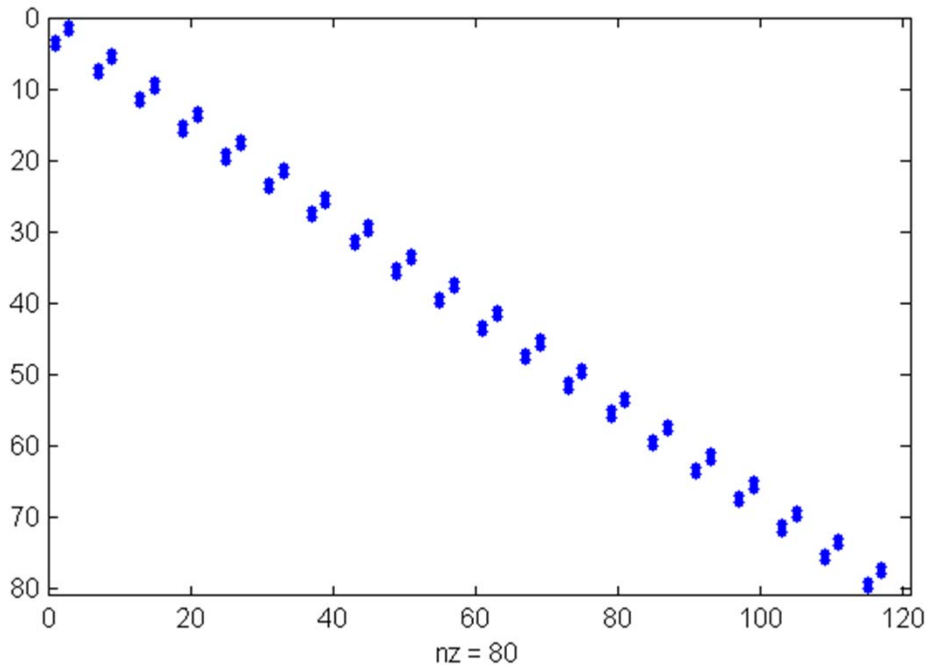
```
n = length(A(1,:));
m = length(B(1,:));

% Dynamics
for i=1:T-1
    Aeq(n*(i-1)+1:n*i, (n+m)*(i)+1:(n+m)*(i)+4) = -eye(n);
    Aeq(n*(i-1)+1:n*i, (n+m)*(i-1)+1:(n+m)*(i-1)+4) = A;
    Aeq(n*(i-1)+1:n*i, (n+m)*(i-1)+5:(n+m)*(i-1)+6) = B;
    beq(n*(i-1)+1:n*i) = zeros(n,1);
end

% Initial and Final Conditions
Aeq(n*(T-1)+1:n*(T-1)+n,1:n) = eye(n);
Aeq(n*(T-1)+n+1:n*(T-1)+2*n+m,(n+m)*(T-1)+1:(n+m)*T) = eye(n+m);
beq(n*(T-1)+1:n*(T-1)+2*n+m,1) = [p0'; pF'];
```

Linear Program

- Path Planning Example
  - The resulting Aineq sparsity pattern
    - spy(A) in Matlab



Linear Program

- Path Planning Example
  - Inequality constraint code
    - Could also be included in bounds on state/inputs

```
g = [ 0 1; 0 -1; 1 0; -1 0];
b = [4.2; 0; 4.2; 0];
q = length(g(:,1));


for i = 1:T
    Aineq(q*(i-1)+1:q*i, (n+m)*(i-1)+1:2:(n+m)*(i-1)+3) = g;
    bineq(q*(i-1)+1:q*i) = b;
end
```

Linear Program

- Path Planning Example
  - Once all of the setup is complete

```
[X,FVAL,EXITFLAG,OUTPUT,LAMBDA] =
     linprog(f,Aineq,bineq,Aeq,beq,LB,UB,x0,options);
```

```
  Residuals:   Primal     Dual     Upper    Duality     Total
               Infeas    Infeas    Bounds     Gap        Rel
               A*x-b   A'*y+z-w-f {x}+s-ub x'*z+s'*w   Error
  ---------------------------------------------------------------
  Iter    0:  6.22e+002 3.74e+001 8.02e+002 9.18e+004 2.69e+000
  Iter    1:  4.67e+000 9.61e-015 6.02e+000 2.27e+003 1.20e+000
  Iter    2:  7.64e-011 2.44e-013 0.00e+000 9.27e+001 3.17e-001
  Iter    3:  1.56e-010 4.43e-014 3.08e-015 2.19e+001 7.54e-002
  Iter    4:  5.00e-011 6.75e-015 0.00e+000 6.13e+000 2.12e-002
  Iter    5:  3.16e-011 6.30e-015 2.51e-015 9.12e-001 3.17e-003
  Iter    6:  5.20e-011 6.54e-015 2.51e-015 9.89e-003 3.44e-005
  Iter    7:  2.72e-011 6.48e-015 1.78e-015 4.95e-007 1.72e-009
Optimization terminated.
```

Linear Program

- Path Planning Example
  - Initial location

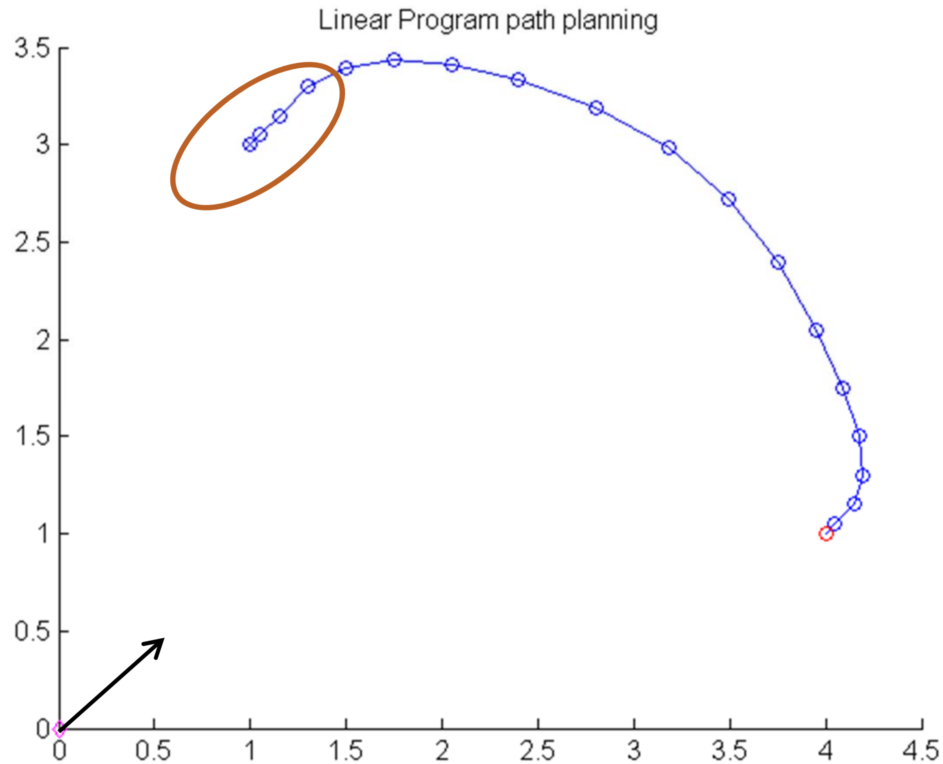
$$p_0 = [1 \quad 3]$$

  - Final location

$$p_F = [4 \quad 1]$$

  - Allowable region

$$X = \{(x, y) \mid x, y \in [0, 4.2]\}$$



  - Cost per timestep

$$f_t(x_{t-1}, u_t) = [-1 \quad 1 \quad -1 \quad 1 \quad 3 \quad 3]\begin{bmatrix} x_{t-1} \\ u_t \end{bmatrix}$$

Control bounds

$$|u| \le 5$$

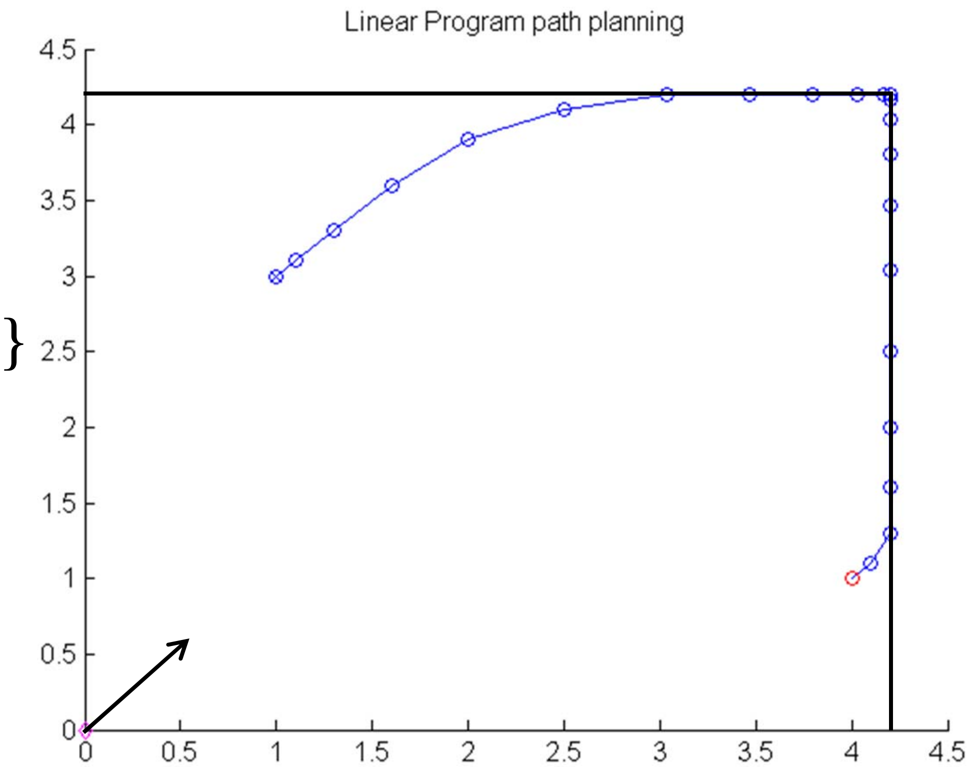
LINEAR PROGRAM

- Path Planning Example,
  - More time
    - T = 40
  - Allowable region

$$X = \{(x, y) \mid x, y \in [0, 4.2]\}$$

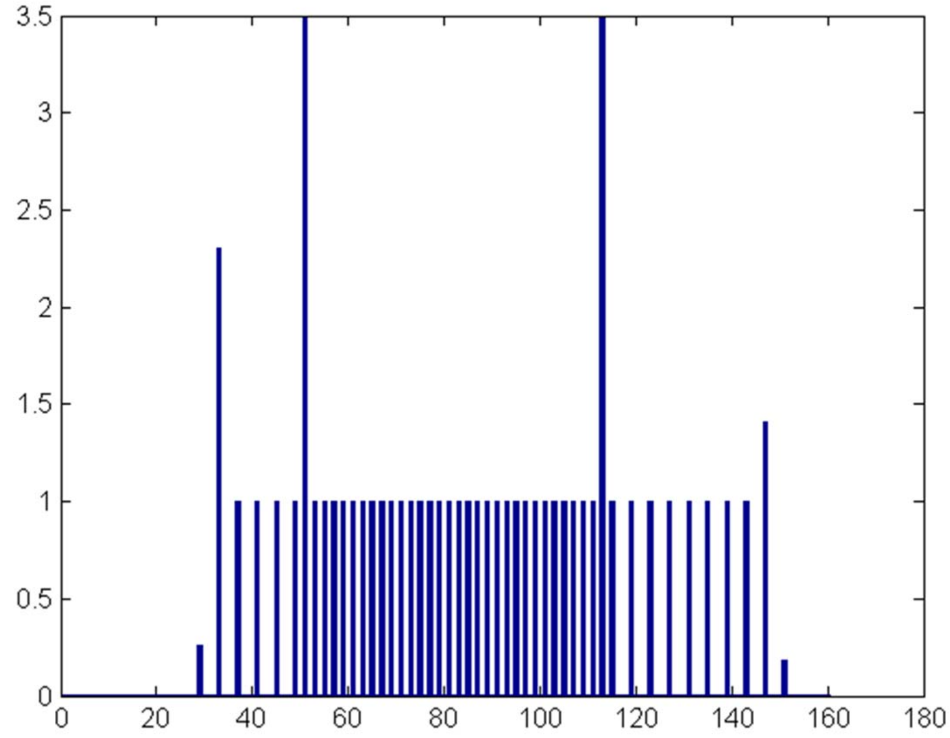
  - Control bounds

$$|u| \leq 5$$



Linear Program

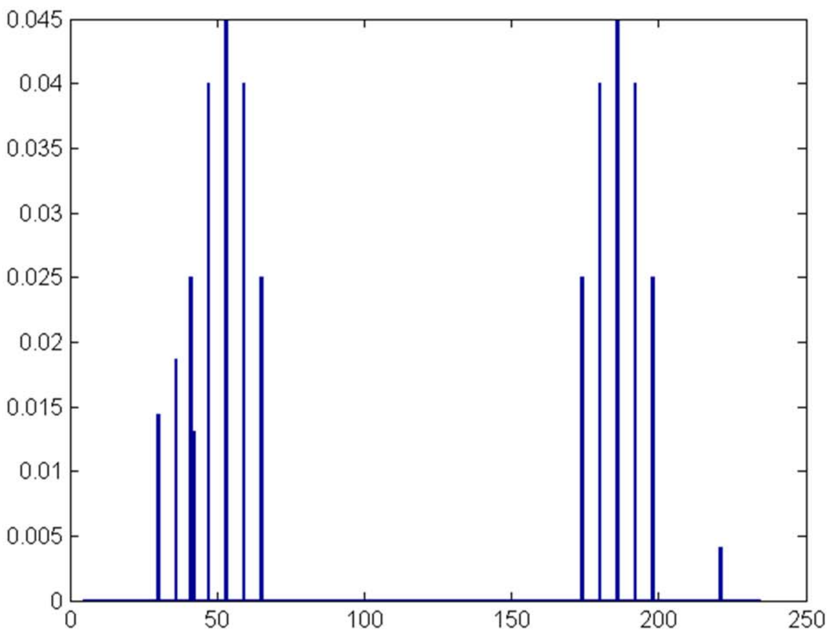
- Path planning example
  - Lagrange multipliers for all inequality constraints
    - All four sides of environment at each timestep



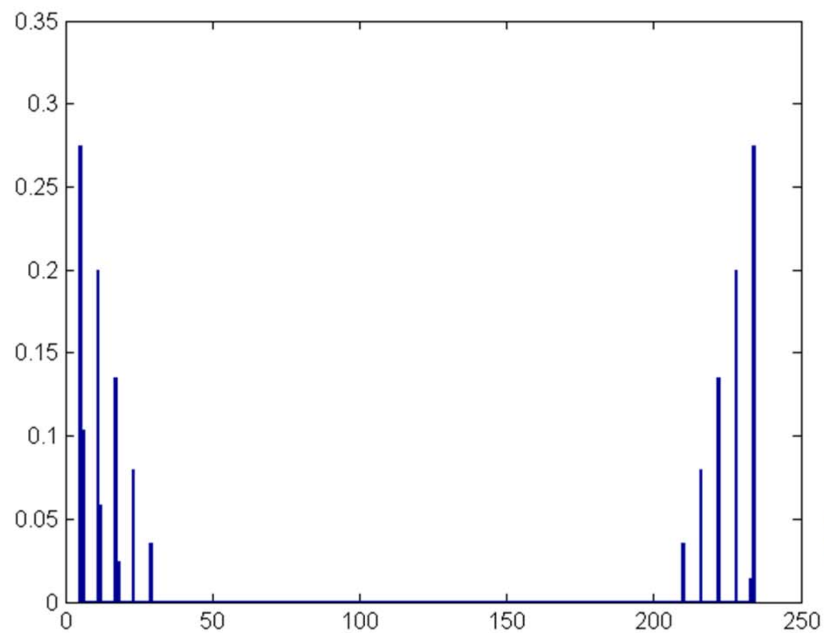
Linear Program

- Path planning example
  - Lagrange multipliers for all variable bounds
    - Four states and two inputs at each timestep

Lower



Upper



Optimization Problem Types

- Quadratic Program (QP)
  - (P) Quadratic cost with linear constraints $O(n^3)$
    - Still fairly easy, fast to solve and convex

$$
\begin{aligned}
\min_{x \in X \subseteq \Re^n} \quad & x^T Q x \\
\text{s.t.} \quad & Ax \leq b \\
& A_{eq} x = b_{eq}
\end{aligned}
$$

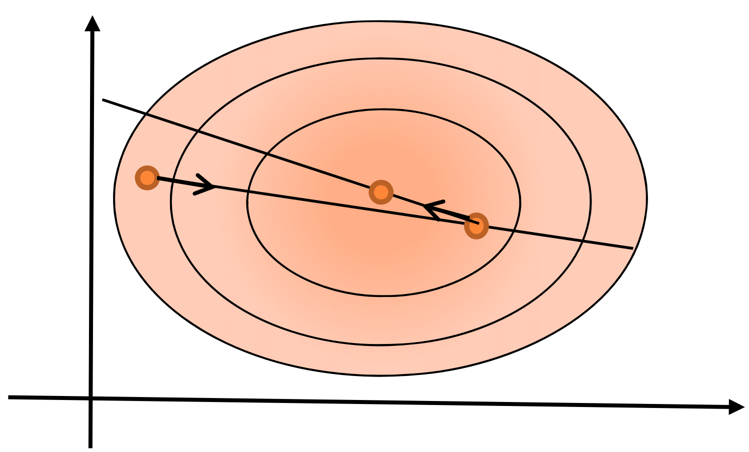
- Matlab command:

```
x = quadprog(Q, A, b, Aeq, beq, LB, UB, x0)
```

- Kalman filter, LQR (unconstrained)
- In fact, any convex problem can be solved quickly
  - Matlab toolbox: cvx

Solution Methods For NLPs

- Sequential Quadratic Programming
  - Also an interior point method
  - At each iteration, calculate gradient and Hessian of Lagrangian
  - If problem is a quadratic program, apply Newton step to optimal solution
  - If not, use Newton step direction as a descent direction and apply a line search
  - Finding Newton step involves inverse of Hessian



Mixed Integer Linear Programming

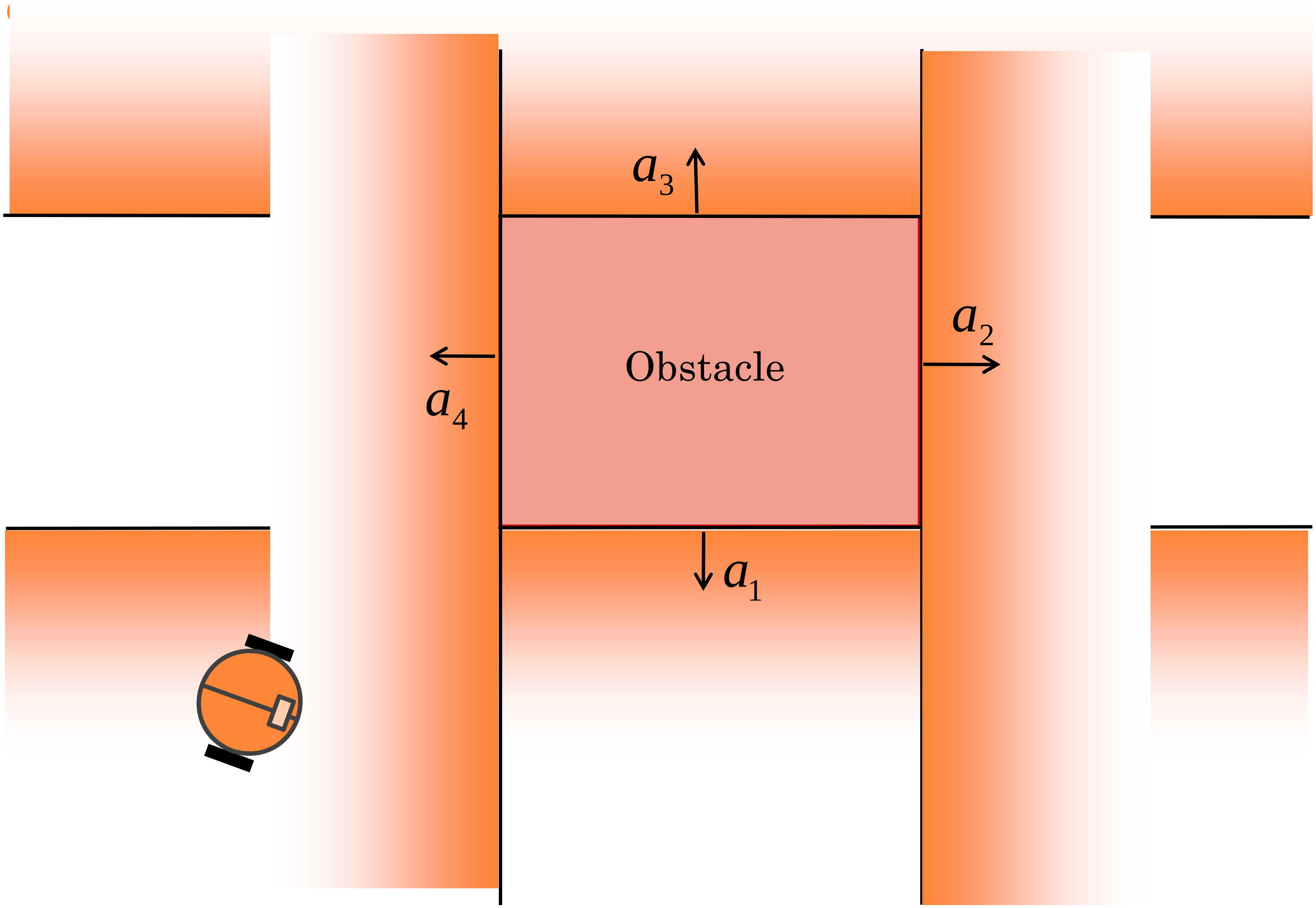
- Key insight into problem formulation
  - Since binary/integer variables are tied directly to complexity, use as few as possible

- Key formulation trick – Big-M constraints
  - A binary decision variable and large constant can be used to selectively relax a set of constraints

$$Ax - B + Mb \geq 0 \rightarrow \begin{cases} Ax - B \geq 0 & b = 0 \\ Mb \geq 0 & b = 1 \end{cases}$$

  - Expensive solvers can sometimes do this without numerical issues
    - CPLEX logical indicator constraints

Mixed Integer Linear Programming



MIXED INTEGER LINEAR PROGRAMMING

- Representing Obstacles in MILP
  - At each timestep, for each obstacle
    - Each edge requires a single constraint

$$a_{i,e}x_t - b_{i,e} - M\left(1 - o_{i,e}\right) \leq 0$$

    - $o_{i,e}$ is a binary decision variable
      - 0 – constraint is inactive
      - 1 – constraint is active
    - M is a large number that relaxes the constraint when not active
  - At each time step, for each obstacle
    - Must be satisfying at least one obstacle edge constraints

$$\sum_{e=1}^{N_e} o_{i,e} \geq 1$$

MIXED INTEGER LINEAR PROGRAMMING

- Minimum time formulation
  - Dynamics apply when not at end point

$$Ax_{t-1} + Bu_t - x_t = 0$$

  - Vehicle does not move once end point is reached

$$x_t - x_{t-1} = 0$$

    - This requires the end point to be consistent with dynamics
    - For the 4 state linear motion model, ensure end point velocities are 0.

  - Formulate big-M constraints to use dynamics while moving and fixed end point once arrived

Mixed Integer Linear Programming

- Minimum Time Formulation
  - To relax equality constraints, must convert to pairs of inequality constraints
    - For dynamics,

$$Ax_{t-1} + Bu_t - x_t + Md_t \geq 0$$
$$Ax_{t-1} + Bu_t - x_t - Md_t \leq 0$$

    - For end point, $x_{t_F} = x_F$

$$x_t - x_{t-1} + M(1-d_t) \geq 0$$
$$x_t - x_{t-1} - M(1-d_t) \leq 0$$

    - And ensuring we don't leave the end point once arrived

$$d_{t+1} - d_t \geq 0 \qquad d_{t_0} = 0 \qquad d_{t_F} = 1$$

Mixed Integer Linear Programming

- Minimizing the magnitude of inputs
  - Define new variable

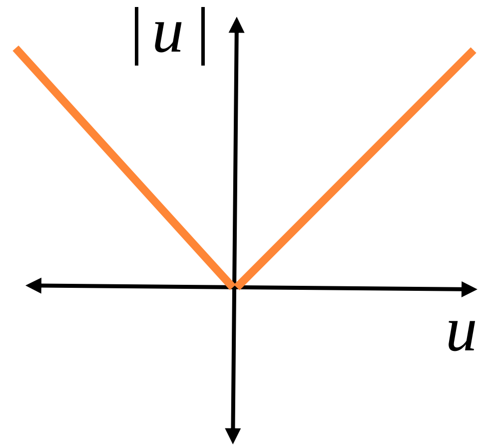
$$u^m$$

  - Add two sets of constraints

$$u^m \geq u$$

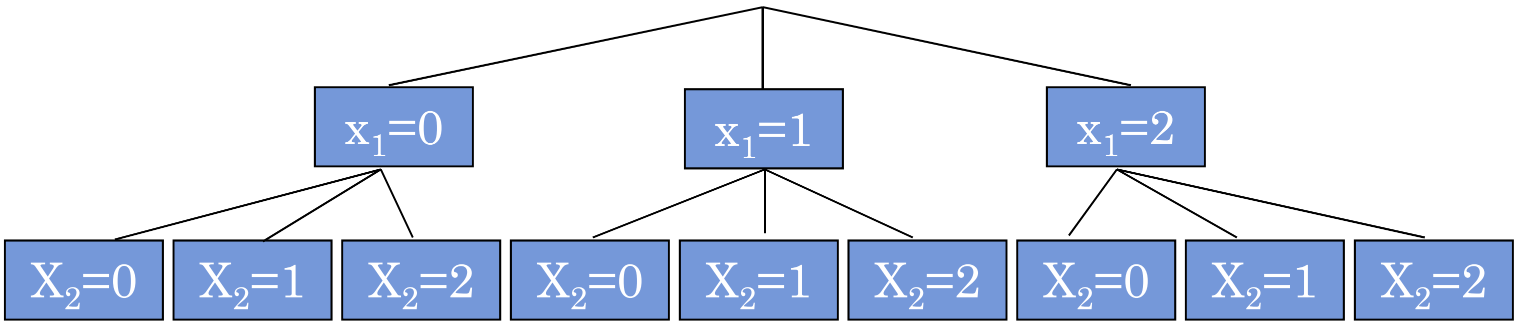
$$u^m \geq -u$$

  - Minimize u^m at each timestep
  - Works for LP, NLP as well
  - u^m referred to as a slack variable



Solution Methods for Integer Programs

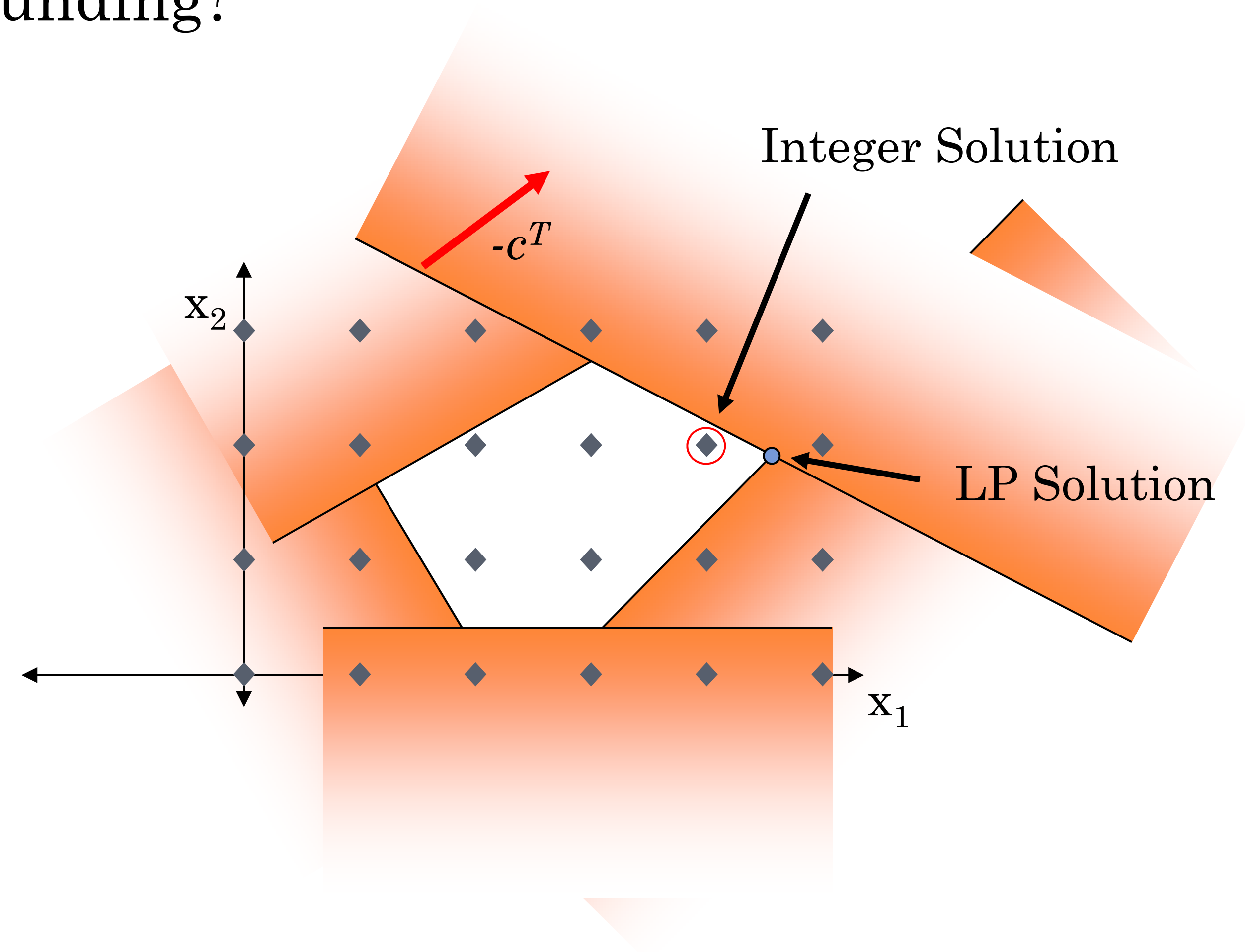
- Enumeration – Tree Search, Dynamic Programming etc.



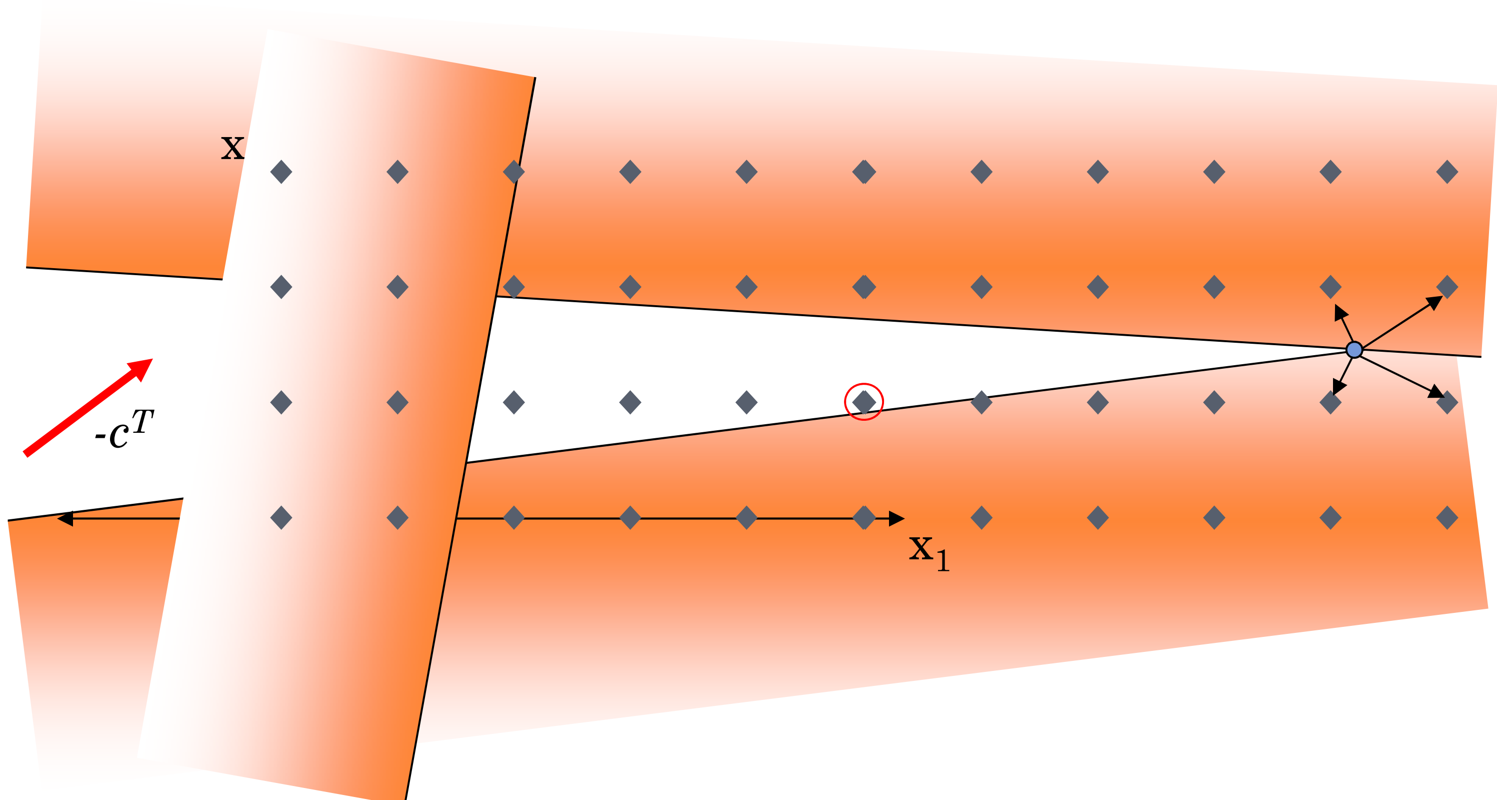
  - Guaranteed to find a feasible solution (only consider integers, can check feasibility (P))
  - But, guaranteed exponential growth in computation time

Solution Methods for Integer Programs

- How about solving LP Relaxation followed by rounding?



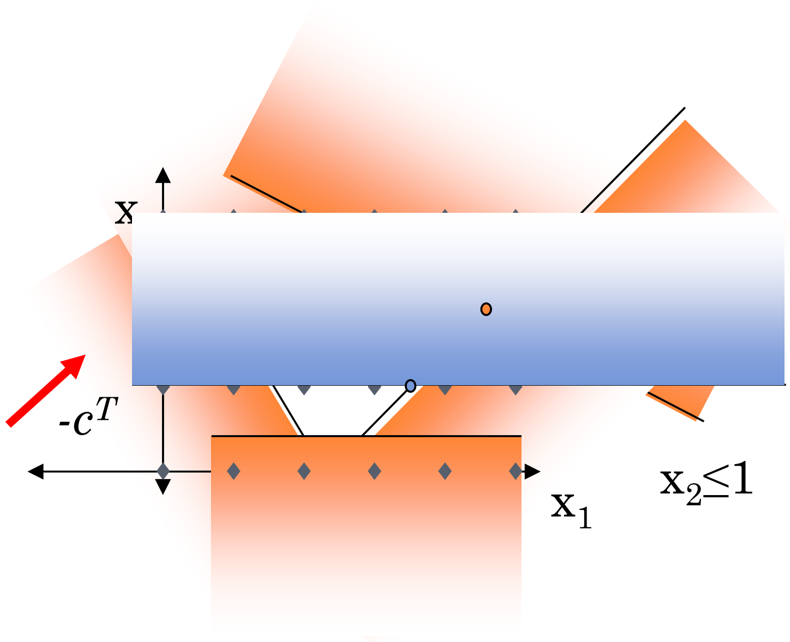
Integer Programs

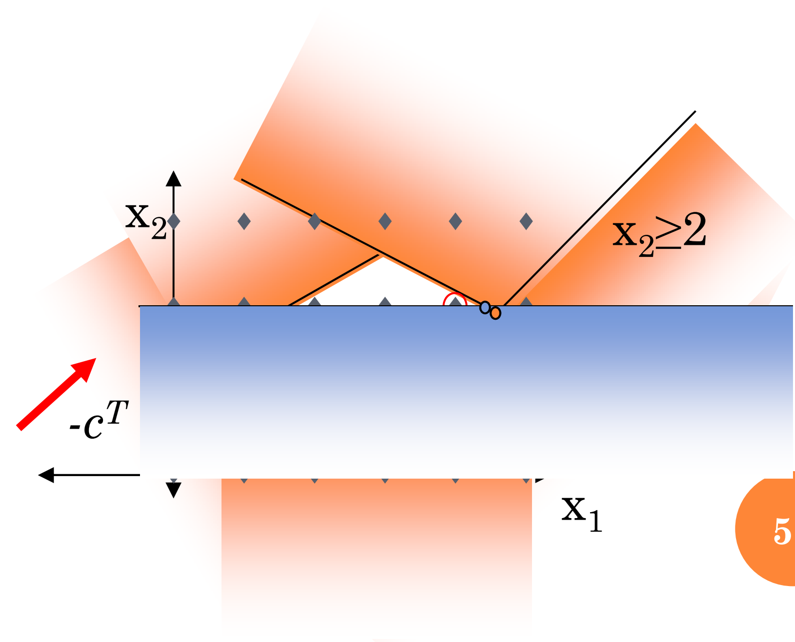


- LP solution provides lower bound on IP
- But, rounding can be arbitrarily far away from integer solution

Combined approach to Integer Programming

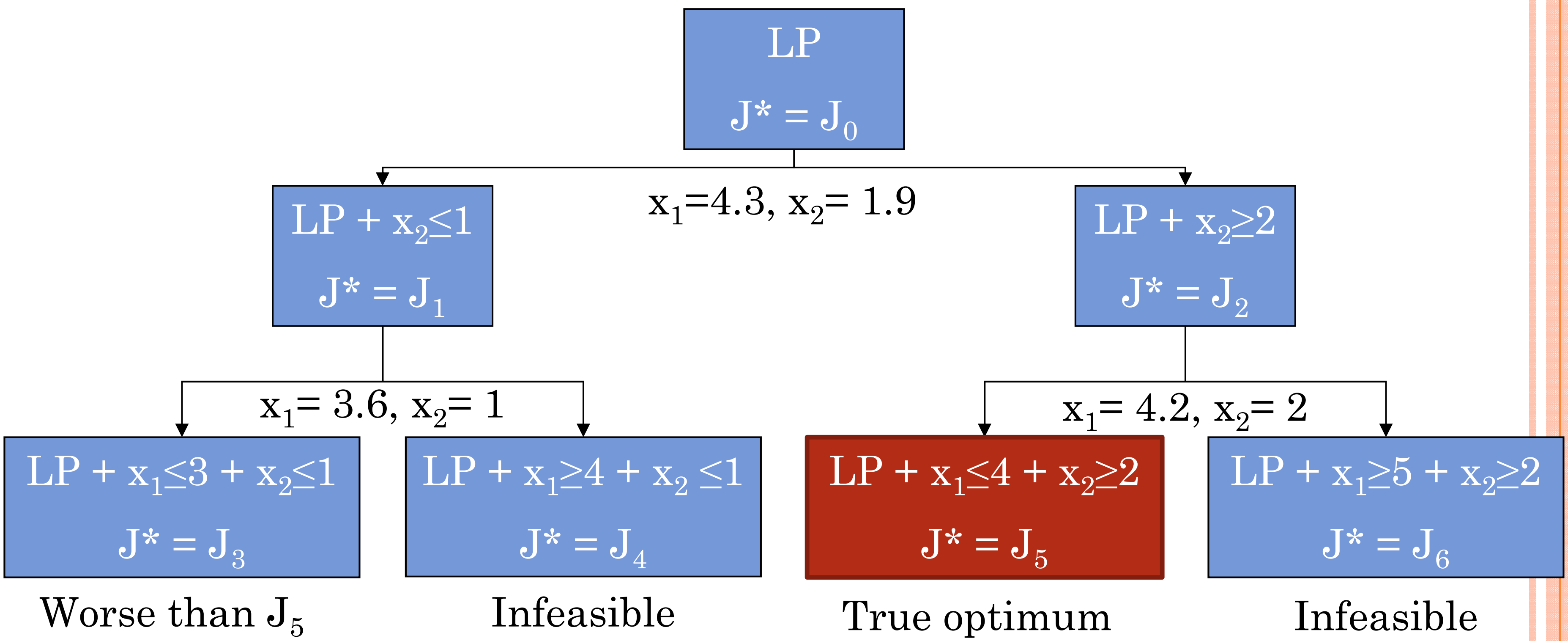
- Why not combine both approaches!
  - Solve LP Relaxation to get fractional solutions
  - Create two sub-branches by adding constraints





Solution Methods for Integer Programs

- Known as Branch and Bound
  - Branch as above
  - LP give lower bound, feasible solutions give upper bound

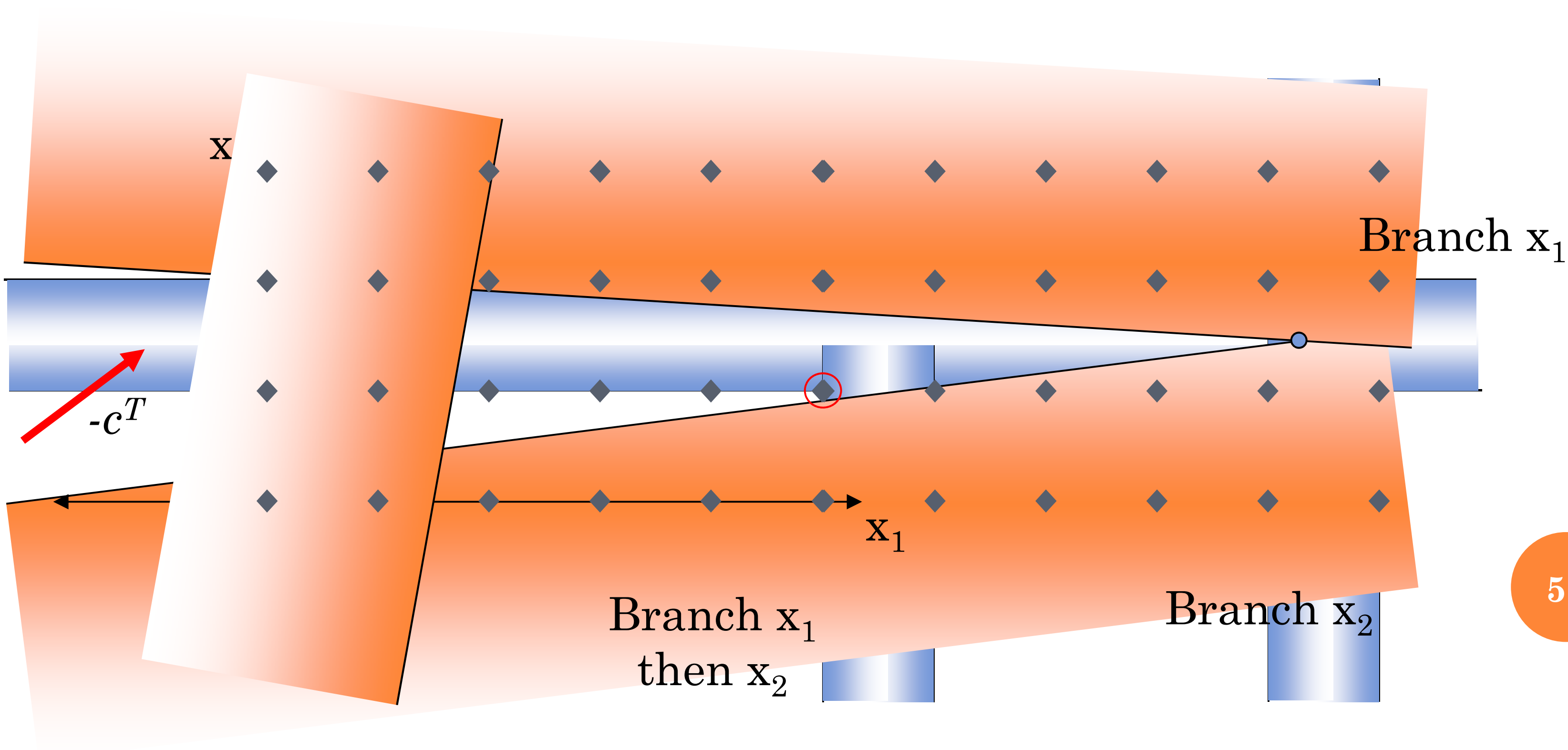


Branch and Bound Method

- Branch and Bound Algorithm
  1. Solve LP relaxation for lower bound on cost for current branch
     - If solution exceeds upper bound, branch is terminated
     - If solution is integer, replace upper bound on cost if lower
  2. Create two branched problems by adding constraints to original problem
     - Select integer variable with fractional LP solution
     - Add integer constraints to the original LP
  3. Repeat until no branches remain, return optimal solution.

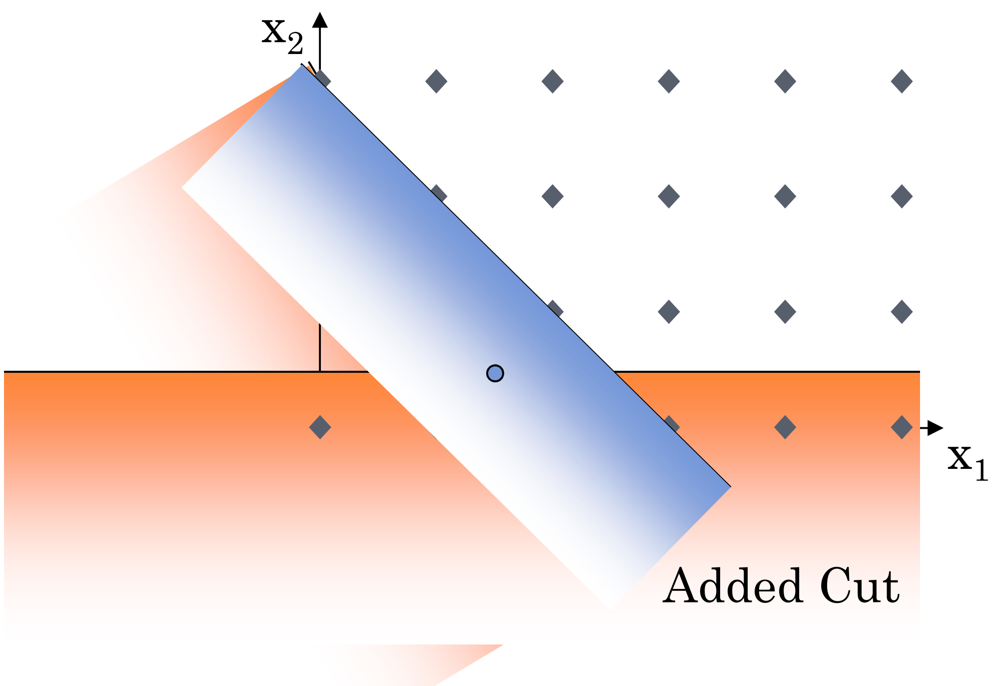
Integer Programs

- Order matters
  - All solutions cause branching to stop
  - Each feasible solution is an upper bound on optimal cost, allowing elimination of nodes



Additional Refinements – Cutting Planes

- Idea stems from adding additional constraints to LP to improve tightness of relaxation
- Combine constraints to eliminate non-integer solutions



- All feasible integer solutions remain feasible
- Current LP solution is not feasible

OUTLINE

- Optimal Planning
  - Motion Planning with Nonlinear Programming
  - Receding Horizon Planning
  - Motion Planning with Mixed Integer Linear Programming

Optimal Planning

- Mixed Integer Linear Program (MILP)
  - (NP-hard) computational complexity

$$\begin{aligned} \min_{x \in X} \quad & f^T x \\ \text{s.t.} \quad & Ax \leq b \\ & A_{eq} = b_{eq} \\ \text{where } X & \subseteq \mathbb{Z}^{n_i} \times \mathbb{R}^{n_r} \end{aligned}$$

  - Exponential growth in complexity
  - However, many problems can be solved surprisingly quickly
- MINLP, MILQP etc.

MIXED INTEGER LINEAR PROGRAMMING

- The core issue with NLPs are
  - Smooth obstacle definitions
  - Local minima
  - Difficulty evaluating alternative routes around obstacles
    - Continuous deformation can't jump over holes
- Alternative is to pose as MILP
    - Integer variable represents whether or not a constraint is active
    - Guaranteed to find optimal solution
    - Exponential complexity growth in number of binary decision variables
    - Limited to linear dynamics

Mixed Integer Linear Programs

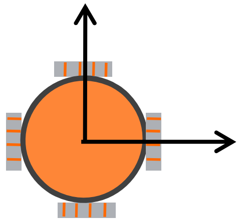
- Solved via branch and bound
  - Same concept as A* search
    - If lower bound on cost exceeds current best solution, no need to evaluate this branch of solutions further
    - The faster a good upper bound on the optimal cost is found, and the tighter the lower bounds on costs-to-go, the faster a solution can be proven optimal

- Optimization Packages
  - ILOG CPLEX: Gold standard of industry, expensive, but free for academics!

  - LU-solve: free, open source, easy to use, callable from Matlab, included in code library with a dll for Win 64.

Mixed Integer Linear Program

- Path Planning example
  - Note: It is difficult to devise a real world robotics problem that is a pure Linear Program, but with integer variables, things get more interesting!
  - Linear dynamics



$$x_t = \begin{bmatrix} 1 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 1 \end{bmatrix} x_{t-1} + \begin{bmatrix} 0 & 0 \\ dt & 0 \\ 0 & 0 \\ 0 & dt \end{bmatrix} u_t$$

$$x_t = Ax_{t-1} + Bu_t$$

Mixed Integer Linear Program

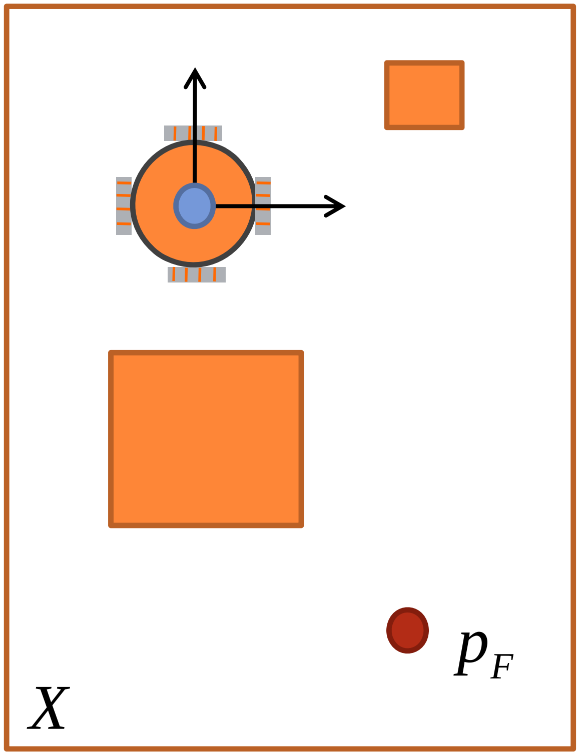
- Path Planning example
  - Initial and final positions

    $$x_0 = p_0 \qquad x_{t_F} = p_F$$
  - Minimum and maximum inputs

    $$\underline{u} \le u_t \le \overline{u}$$
  - Minimum and maximum positions

    $$x_{[1,3],t} \in X$$
  - Minimum and maximum velocities

    $$x_{[2,4],t} \in V$$
  - Obstacles



$$a_{i,e} x_t - b_{i,e} - M\left(1 - o_{i,e}\right) \le 0 \qquad \sum_{e=1}^{N_e} o_{i,e} \ge 1$$

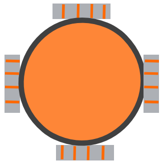
Mixed Integer Linear Program

- Path Planning example
  - Optimization variables

$$X = [x_0 \ldots x_T \quad u_0 \ldots u_T \quad u_0^m \ldots u_T^m \quad o_{1,1} \ldots o_{1,N_e} o_{2,1} \ldots o_{2,N_e} \ldots o_{M,N_e}]$$

  - Costs
    - Minimize control magnitudes (u^m)

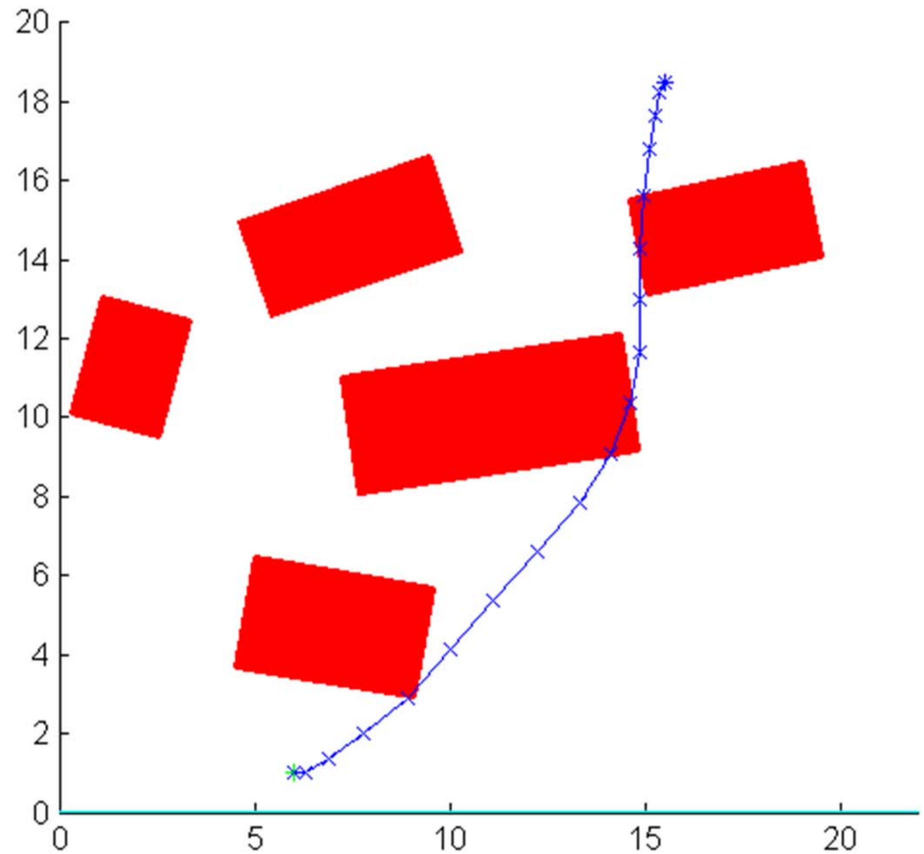
$$f = [\boldsymbol{0} \quad \boldsymbol{0} \quad \boldsymbol{-1} \quad \boldsymbol{0}]^T$$



  - Example is fixed time, can add in minimum time as well
    - Append T binary decision variables to indicate when end goal is reached
    - Replace dynamics equality constraints with four sets of inequalities

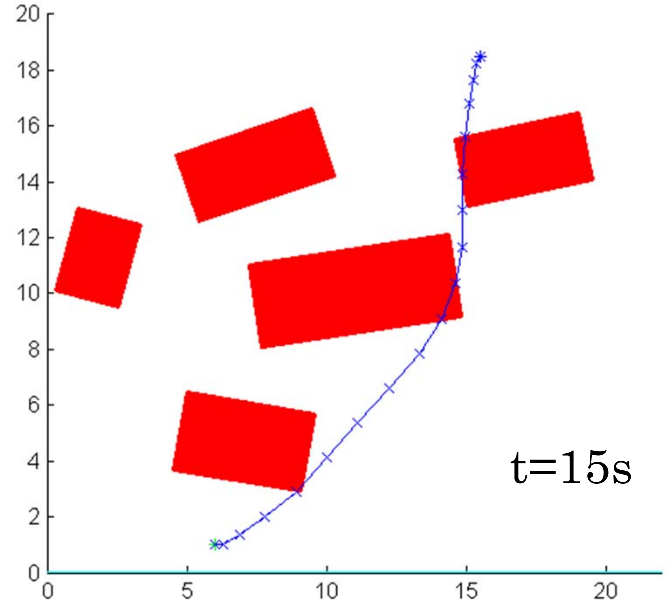
Mixed Integer Linear Program

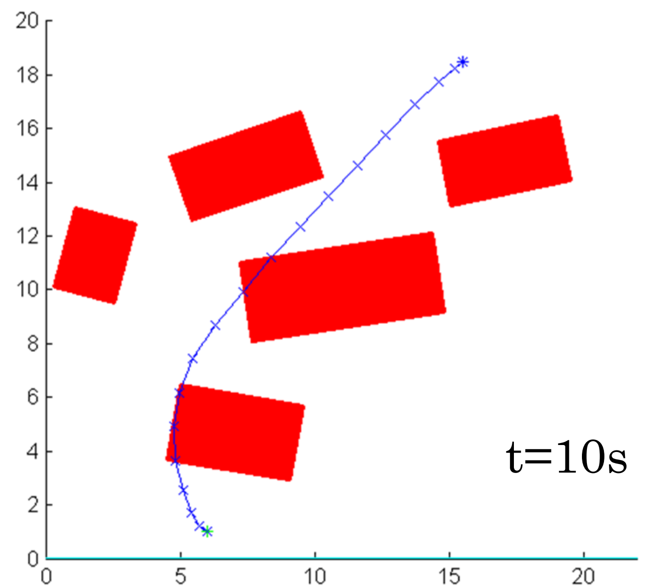
- Results
  - 5 Obstacles
  - 20 time steps
  - 2.5 minutes to compute solution



Mixed Integer Linear Programming

- Results
  - Best known solution after 15 seconds
  - After 10 seconds, a solution only 2.5% worse is found
  - Remainder of time spend ensuring this is truly the optimal solution!





MIXED INTEGER LINEAR PROGRAMMING

- Run time – 2 obstacles, 100 runs
  - 88 under a second, 96 under 2 seconds, 1 took 36 seconds.