CS491/691: Introduction to Aerial Robotics

Topic: Recapitulate

Dr. Kostas Alexis (CSE)

Contents

- We will recapitulate selected topics in:
  - Coordinate Systems Transformations (CST)
  - MAV Dynamics (MAVD)
  - State Estimation (SE)
  - Flight Control Systems (FCS)
  - Motion Planning (MP)

  - With an ***emphasis*** on FCS and MP

CS491/691: Introduction to Aerial Robotics

Topic: Coordinate Systems Transformations

Dr. Kostas Alexis (CSE)

CST: Inertial Frame to Body Frame

- Let:

$$\mathcal{R}_v^b(\phi,\theta,\psi) = \mathcal{R}_{v_2}(\phi)\mathcal{R}_{v_1}^{v_2}(\theta)\mathcal{R}_v^{v_1}(\psi)$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ s_\phi s_\theta c_\psi - c_\phi s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & s_\phi c_\theta \\ c_\phi s_\theta c_\psi + s_\phi s_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\phi c_\theta \end{bmatrix}$$

- Then:

$$\mathbf{p}^b = \mathcal{R}_v^b \mathbf{p}^v$$

CST: Rotation of Reference Frame

- Rotation around the i-axis

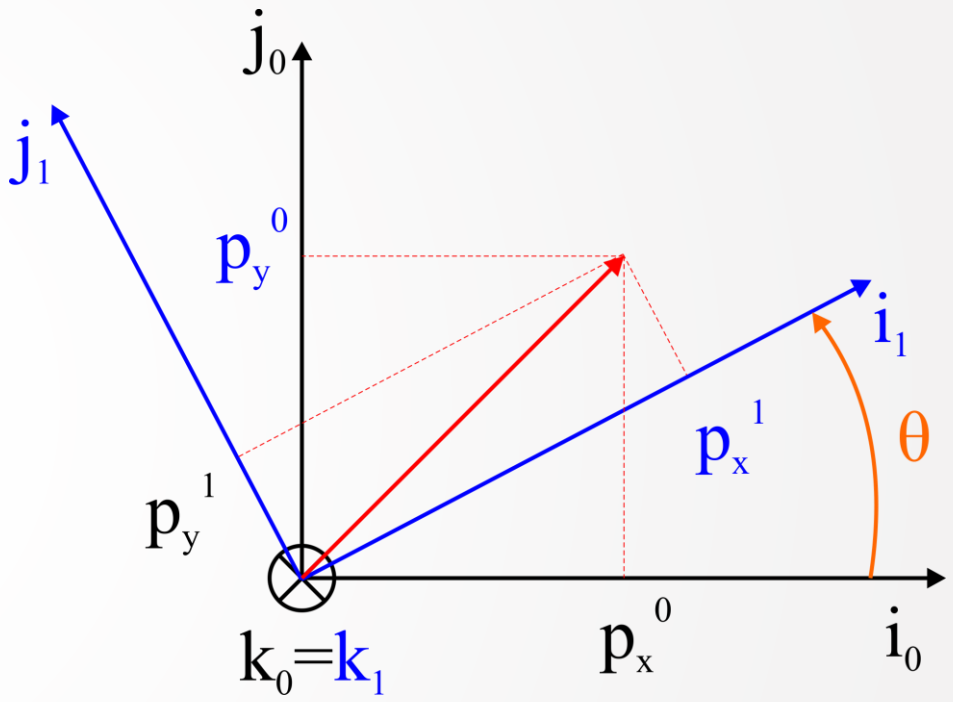
$$\mathcal{R}_0^1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}$$

- Rotation around the j-axis

$$\mathcal{R}_0^1 = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}$$

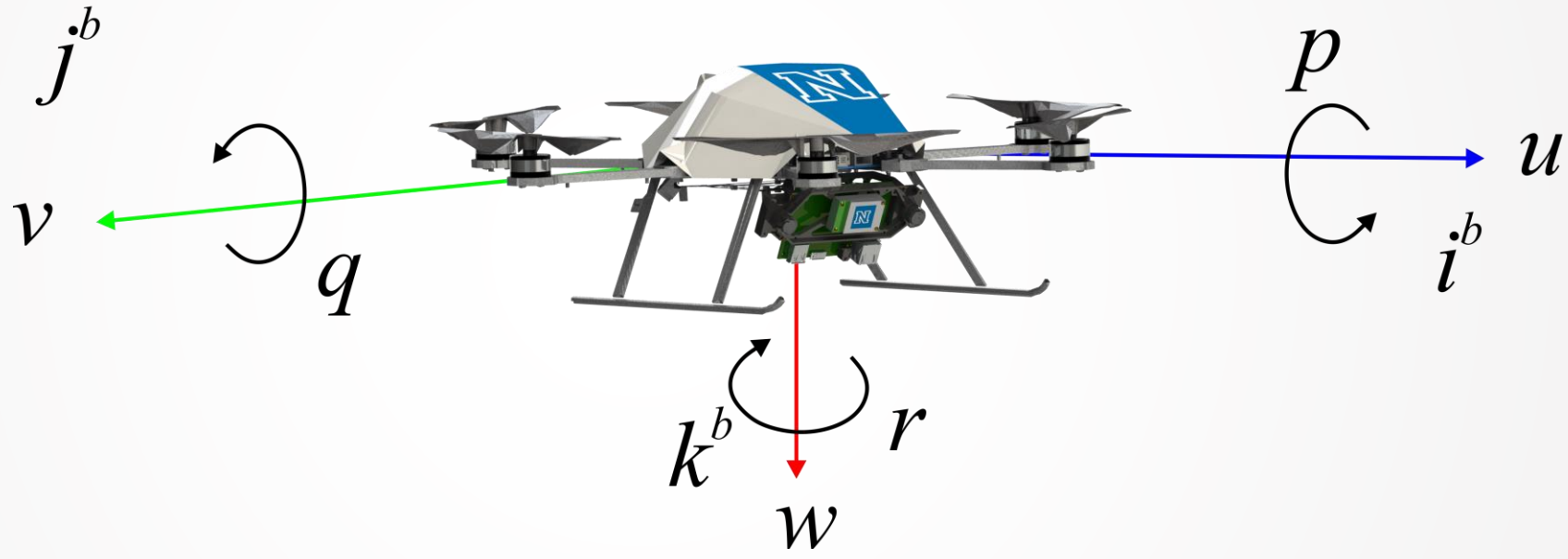
- Rotation around the k-axis

$$\mathcal{R}_0^1 = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



- Orthonormal matrix properties
  - $(\mathcal{R}_a^b)^-1 = (\mathcal{R}_a^b)^T = \mathcal{R}_b^a$
  - $\mathcal{R}_b^c \mathcal{R} a^b = \mathcal{R}_a^c$
  - $\det(\mathcal{R}_a^b) = 1$

CST: Application to Robot Kinematics



- *[p,q,r]* : body angular rates
- *[u,v,w]* : body linear velocities

CST: Relate Translational Velocity-Position

- Let *[u,v,w]* represent the body linear velocities

$$\frac{d}{dt}\begin{bmatrix} p_n \\ p_e \\ p_d \end{bmatrix} = \mathcal{R}_b^v \begin{bmatrix} u \\ v \\ w \end{bmatrix} = (\mathcal{R}_v^b)^T \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

- Which gives:

$$\begin{bmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \end{bmatrix} = \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\phi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_{c\psi} \\ -s_\theta & s_\phi c_\theta & c_{c\theta} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

CST: Body Rates – Euler Rates

- Let *[p,q,r]* denote the body angular rates

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \mathcal{R}^{b}_{v_2}(\phi) \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \mathcal{R}^{b}_{v_2}(\phi)\mathcal{R}^{v_2}_{v_1}(\theta) \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} =$$

$$\begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

- Inverting this expression:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

CST: Indicative questions

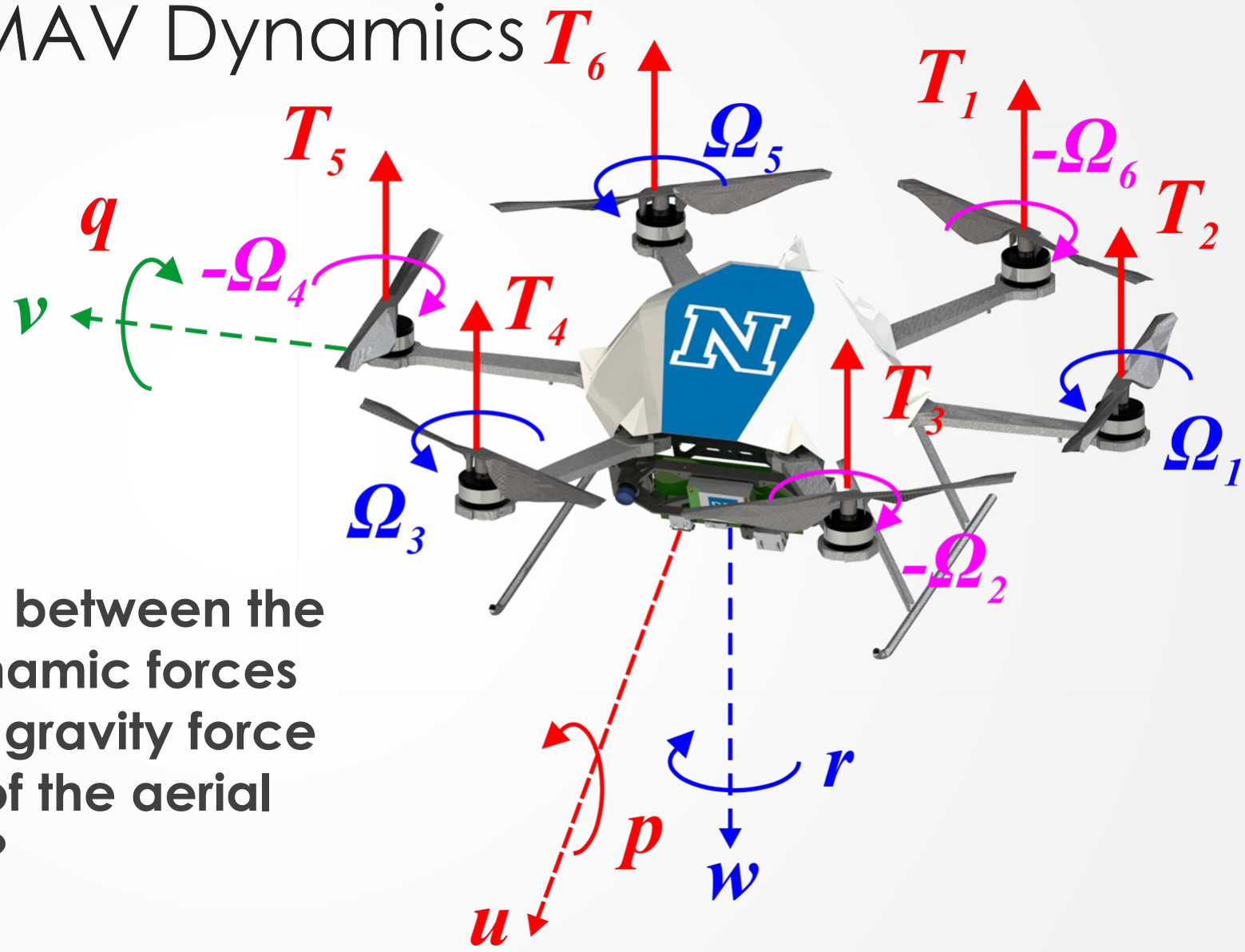
- What is the rotation matrix for a roll motion of π/6 and a pitch motion of π/4?
- Correlate the body rates and the Euler angles derivatives for a roll motion of π/6 and a pitch motion of π/4
- Where do the Euler angles present singularity?
- Explain the chain of operations required to conduct a roll, pitch and yaw rotation

CS491/691: Introduction to Aerial Robotics

Topic: MAV Dynamics

Dr. Kostas Alexis (CSE)

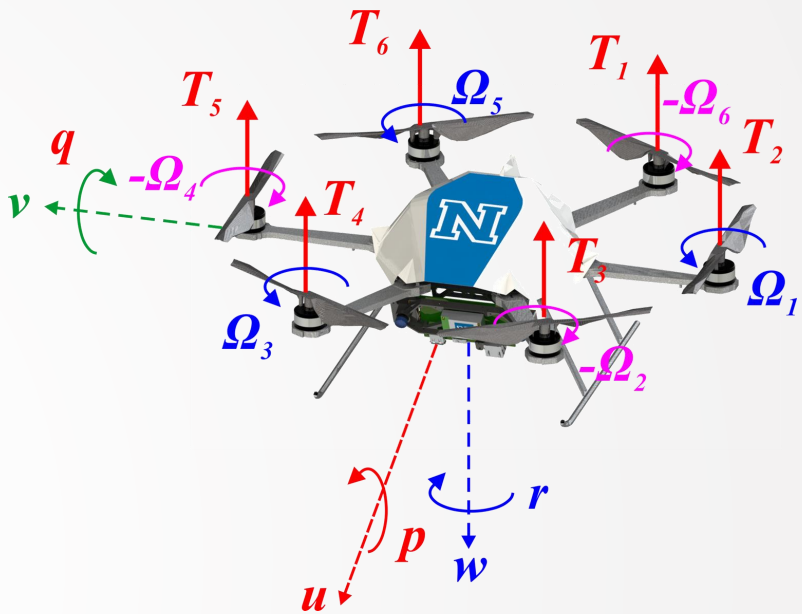
MAVD: MAV Dynamics



What is the relation between the propeller aerodynamic forces and moments, the gravity force and the motion of the aerial robot?

MAVD: MAV Dynamics

- Assumption 1: the Micro Aerial Vehicle is flying as a rigid body with negligible aerodynamic effects on it – for the employed airspeeds.
- The propeller is considered as a simple propeller disc that generates thrust and a moment around its shaft.



- Recall:

$$F_T = T = C_T \rho A (\Omega R)^2$$

$$M_Q = Q = C_Q \rho A (\Omega R)^2 R$$

- And let us write:

$$T_i = k_n \Omega_i^2$$

$$M_i = (-1)^{i-1} k_m T_i$$

MAVD: MAV Dynamics

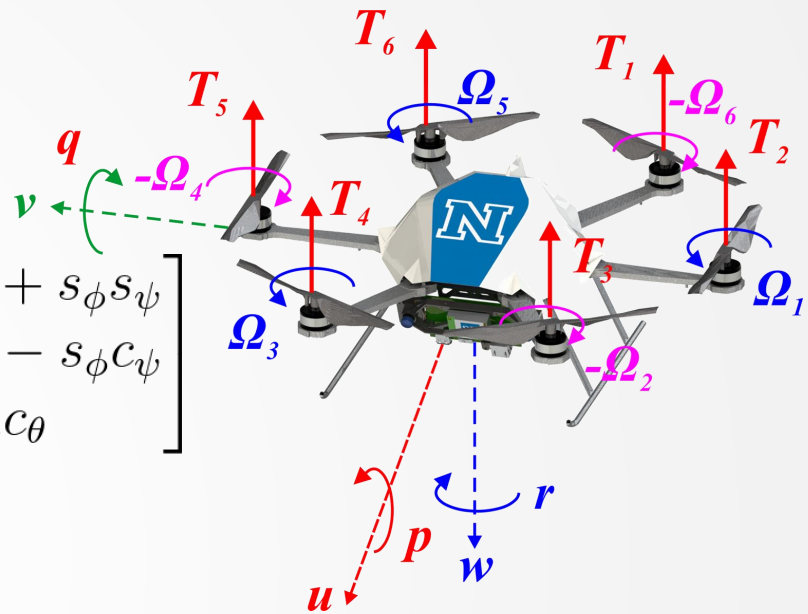
- To append the forces and moments we need to combine their formulation with

$$\begin{bmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \end{bmatrix} = \mathcal{R}_b^v \begin{bmatrix} u \\ v \\ w \end{bmatrix}, \quad \mathcal{R}_b^v = \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix}$$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{bmatrix} + \frac{1}{m} \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi \tan\theta & \cos\phi \tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi \sec\theta & \cos\phi \sec\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{J_y - J_z}{J_x} qr \\ \frac{J_z - J_x}{J_y} pr \\ \frac{J_x - J_y}{J_z} pq \end{bmatrix} + \begin{bmatrix} \frac{1}{J_x} M_x \\ \frac{1}{J_y} M_y \\ \frac{1}{J_z} M_z \end{bmatrix}$$

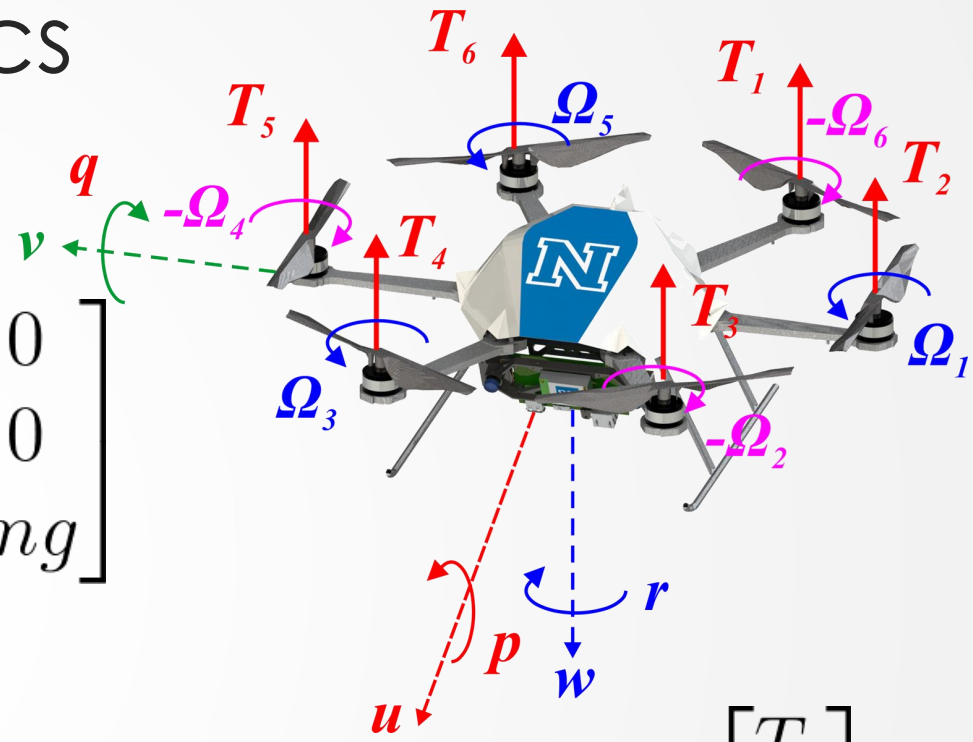


- ***Next step: append the MAV forces and moments***

MAVD: MAV Dynamics

- MAV forces in the body frame:

$$\mathbf{f}_b = \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{6} T_i \end{bmatrix} - \mathcal{R}_v^b \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}$$



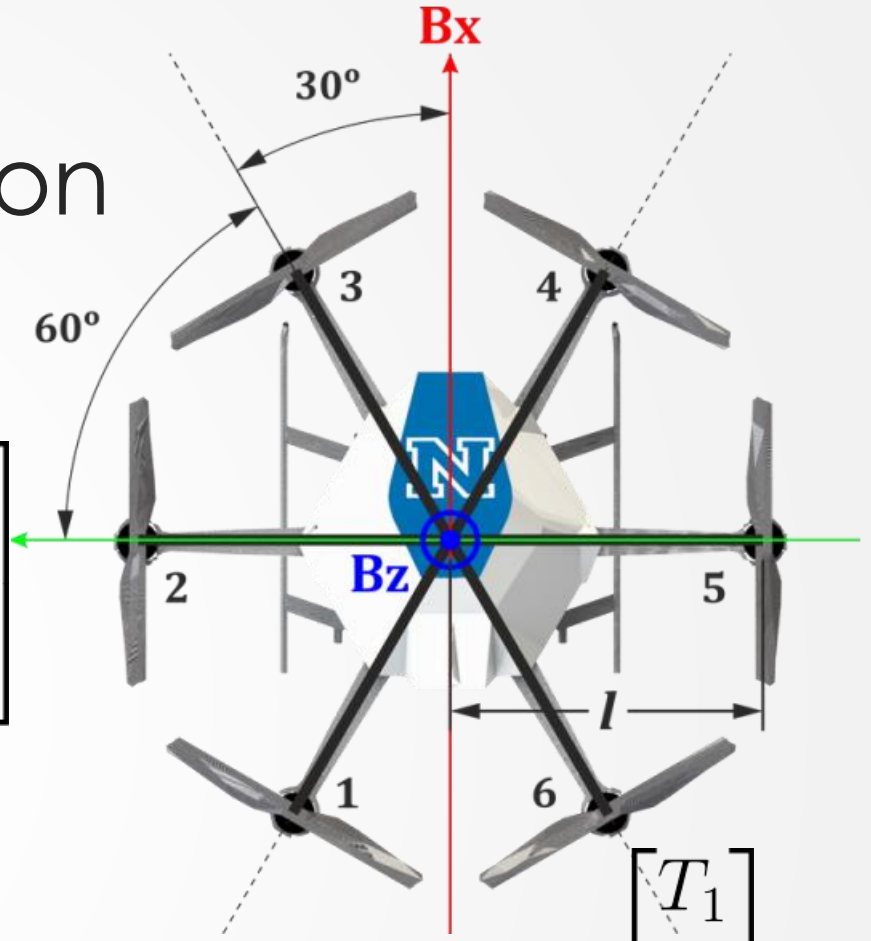
- Moments in the body frame:

$$\mathbf{m}_b = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} lc_{60} & l & lc_{60} & -lc_{60} & -l & -lc_{60} \\ -ls_{60} & 0 & ls_{60} & ls_{60} & 0 & -ls_{60} \\ -k_m & k_m & -k_m & k_m & -k_m & k_m \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{bmatrix}$$

MAVD: Control Allocation

- MAV forces in the body frame:

$$\mathbf{f}_b = \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{6} T_i \end{bmatrix} - \mathcal{R}_v^b \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}$$



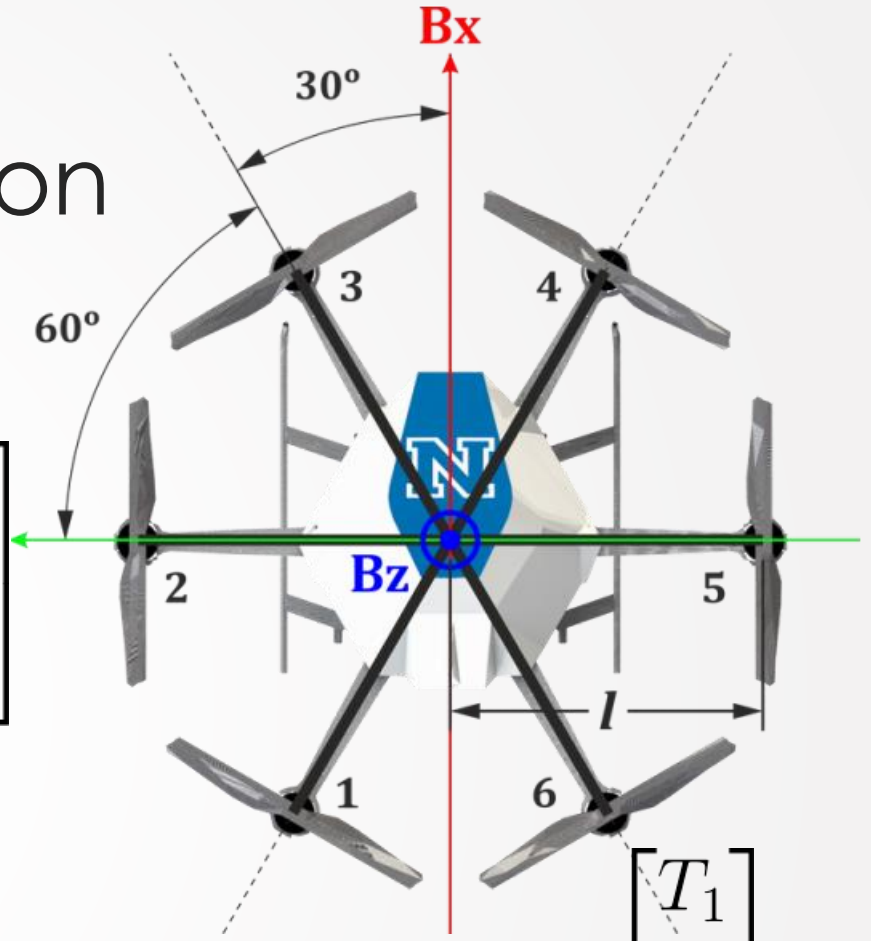
- Moments in the body frame:

$$\mathbf{m}_b = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} lc_{60} & l & lc_{60} & -lc_{60} & -l & -lc_{60} \\ -ls_{60} & 0 & ls_{60} & ls_{60} & 0 & -ls_{60} \\ -k_m & k_m & -k_m & k_m & -k_m & k_m \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{bmatrix}$$

MAVD: Control Allocation

- MAV forces in the body frame:

$$\mathbf{f}_b = \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{6} T_i \end{bmatrix} - \mathcal{R}_v^b \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}$$



- Moments in the body frame:

$$\mathbf{m}_b = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} lc_{60} & l & lc_{60} & -lc_{60} & -l & -lc_{60} \\ -ls_{60} & 0 & ls_{60} & ls_{60} & 0 & -ls_{60} \\ -k_m & k_m & -k_m & k_m & -k_m & k_m \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{bmatrix}$$

MAVD: Indicative questions

- What is the control allocation matrix for an octarotor with arm length equal to 0.3m? Describe how the control methodology of this control allocation actuates the roll, pitch and yaw rotations as well as how the aerial robot moves in the x,y,z axis.
- What is the thrust force necessary to compensate for the weight given any possibly orientation of the MAV that can be continuously defined using Euler angles?
- Describe the series of motions such that the system manages to translate along the x-axis.
- If the system is rotated around the yaw axis, how can the system achieve translation along a single inertial axis?

CS491/691: Introduction to Aerial Robotics

Topic: State Estimation

Dr. Kostas Alexis (CSE)

SE: The State Estimation problem

- We want to estimate the world state x from:
  - Sensor measurements z and
  - Controls u

- We need to model the relationship between these random variables, i.e:

$$p(\mathbf{x}|\mathbf{z}) \qquad p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$$

SE: Causal vs. Diagnostic Reasoning

$P(\mathbf{x}|\mathbf{z})$ Is diagnostic

$P(\mathbf{z}|\mathbf{x})$ Is causal

- Diagnostic reasoning is typically what we need.
- Often causal knowledge is easier to obtain.
- Bayes rule allows us to use causal knowledge in diagnostic reasoning.

SE: Bayes rule

- Definition of **conditional probability**:

$$P(x, z) = P(x|z)P(z) = P(z|x)P(x)$$

- **Bayes rule:**

Observation likelihood

Prior on world state

$$P(x|z) = \frac{P(z|x)P(x)}{P(z)}$$

Prior on sensor observations

Markov Assumption

- Observations depend only on current state

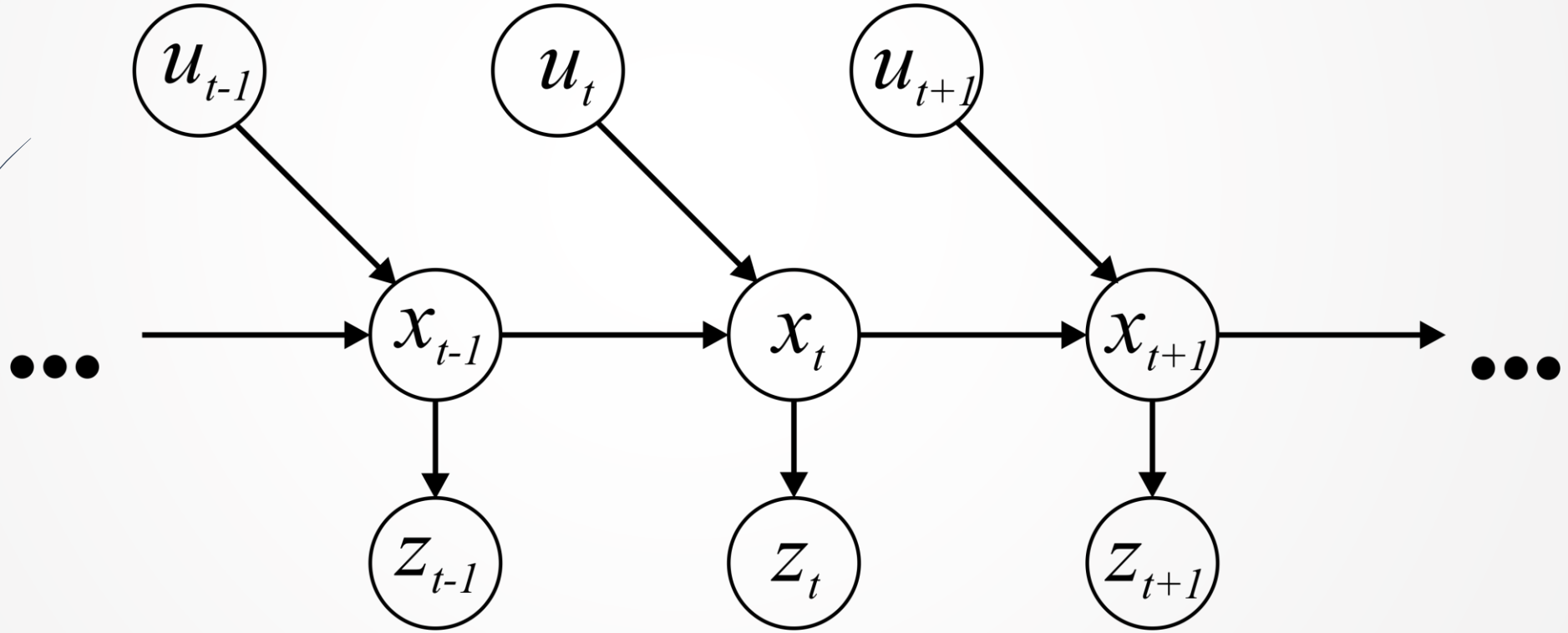
$$P(z_t|x_{0:t}, z_{1:t-1}, u_{1:t}) = P(z_t|x_t)$$

- Current state depends only on previous state and current action

$$P(x_t|x_{0:t}, z_{1:t}, u_{1:t}) = P(x_t|x_{t-1}, u_t)$$

Markov Chain

- A Markov Chain is a stochastic process where, given the present state, the past and the future states are independent.



Underlying Assumptions

- Static world
- Independent noise
- Perfect model, no approximation errors

Bayes Filter

- **Given**
  - Sequence of observations and actions: z_t, u_t
  - Sensor model: $P(z|x)$
  - Action model: $P(x'|x,u)$
  - Prior probability of the system state: $P(x)$
- **Desired**
  - Estimate of the state of the dynamic system: x
  - Posterior of the state is also called belief:

$$Bel(x_t) = P(x_t|u_1, z_1, \ldots, u_t, z_t)$$

Bayes Filter Algorithm

- **For each time step, do:**
  - Apply motion model:

$$\overline{Bel}(x_t) = \sum_{x_t - 1} P(x_t | x_{t-1}, u_t) Bel(x_{t-1})$$

  - Apply sensor model:

$$Bel(x_t) = \eta P(z_t | x_t) \overline{Bel}(x_t)$$

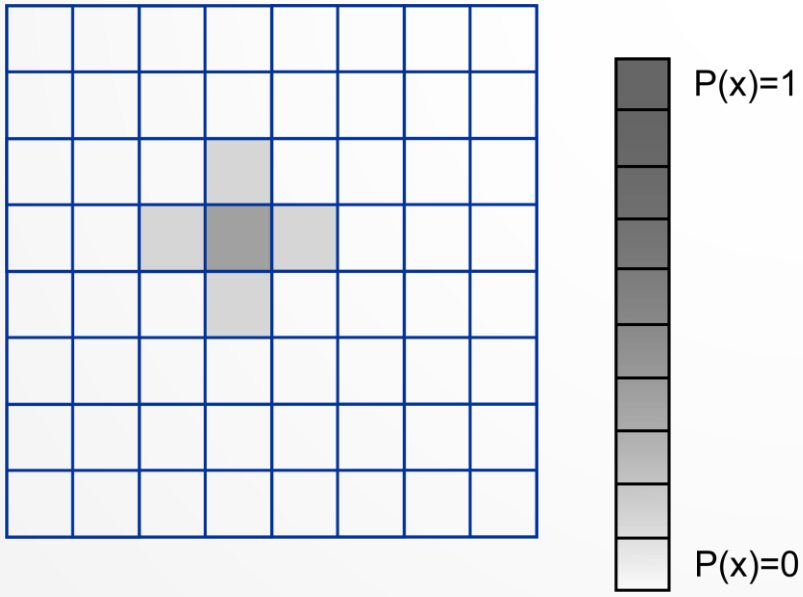
  - η is a normalization factor to ensure that the probability is maximum 1.

Notes

- Bayes filters also work on continuous state spaces (replace sum by integral).
- Bayes filter also works when actions and observations are asynchronous.

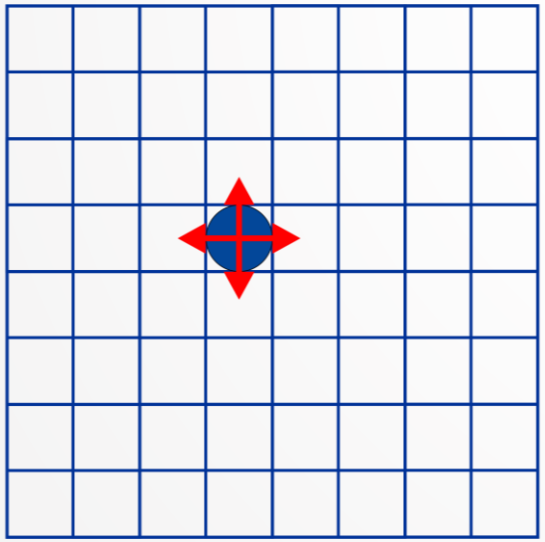
Example: Localization

- Discrete state: $x \in \{1, 2, \ldots, w\} \times \{1, 2, \ldots, h\}$
- Belief distribution can be represented as a grid
- This is also called a **historigram filter**



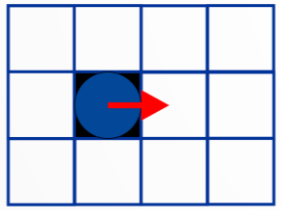
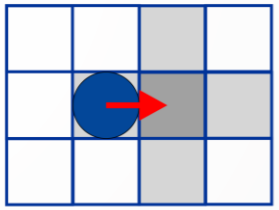
Example: Localization

- Action: $u \in \{north, east, south, west\}$
- Robot can move one cell in each time step
- Actions are not perfectly executed



Example: Localization

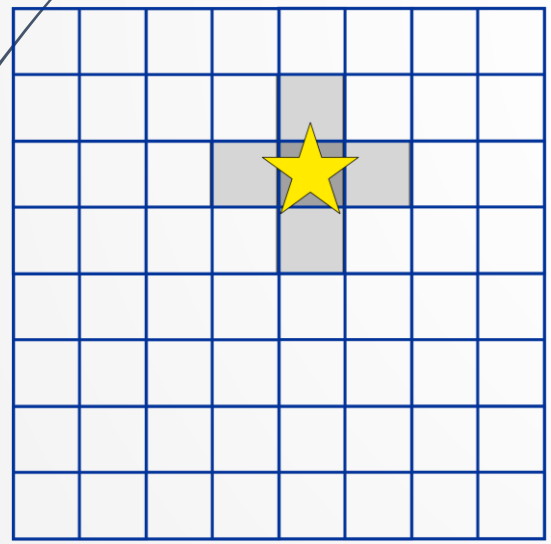
- Action
- Robot can move one cell in each time step
- Actions are not perfectly executed
- Example: move east

$x_{t-1} =$  $u = east$ 

- 60% success rate, 10% to stay/move too far/ move one up/ move one down

Example: Localization

- Binary observation: $z \in \{marker, \bar{marker}\}$
- One (special) location has a marker
- Marker is sometimes also detected in neighboring cells

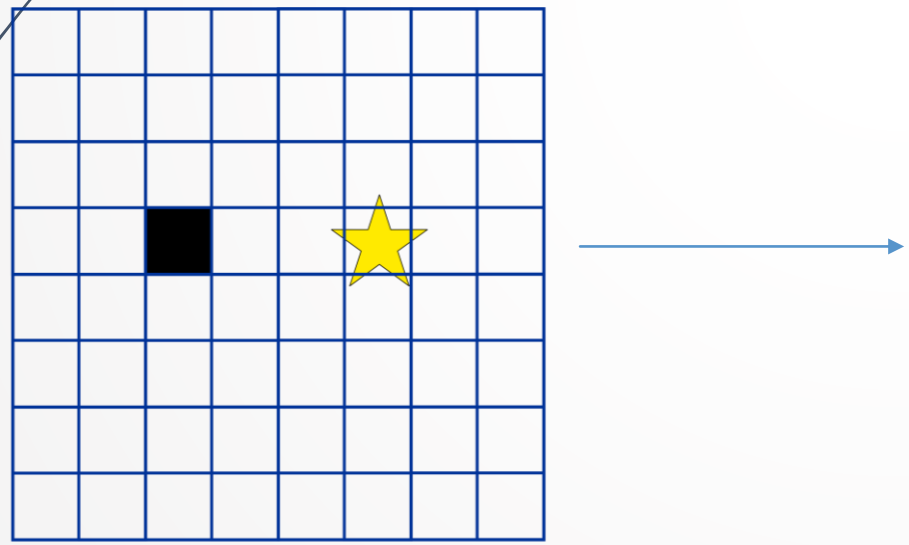


Example: Localization

- Let's start a simulation run…

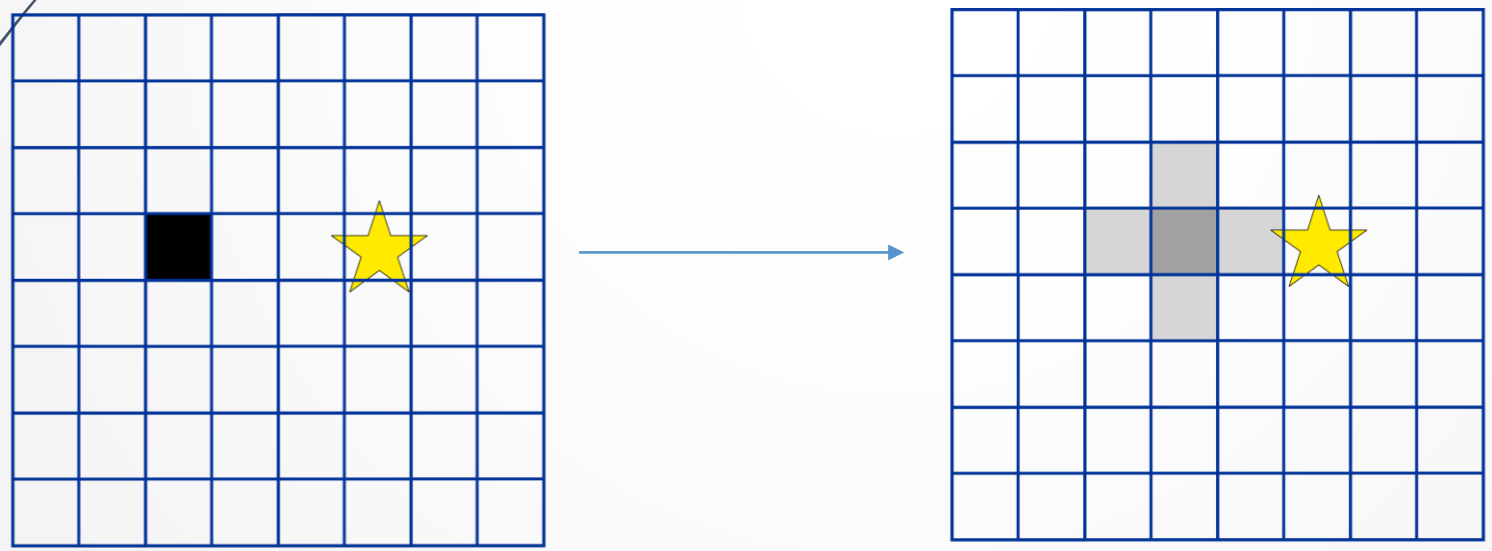
Example: Localization

- t=0
- Prior distribution (initial belief)
- Assume that we know the initial location (if not, we could initialize with a uniform prior)



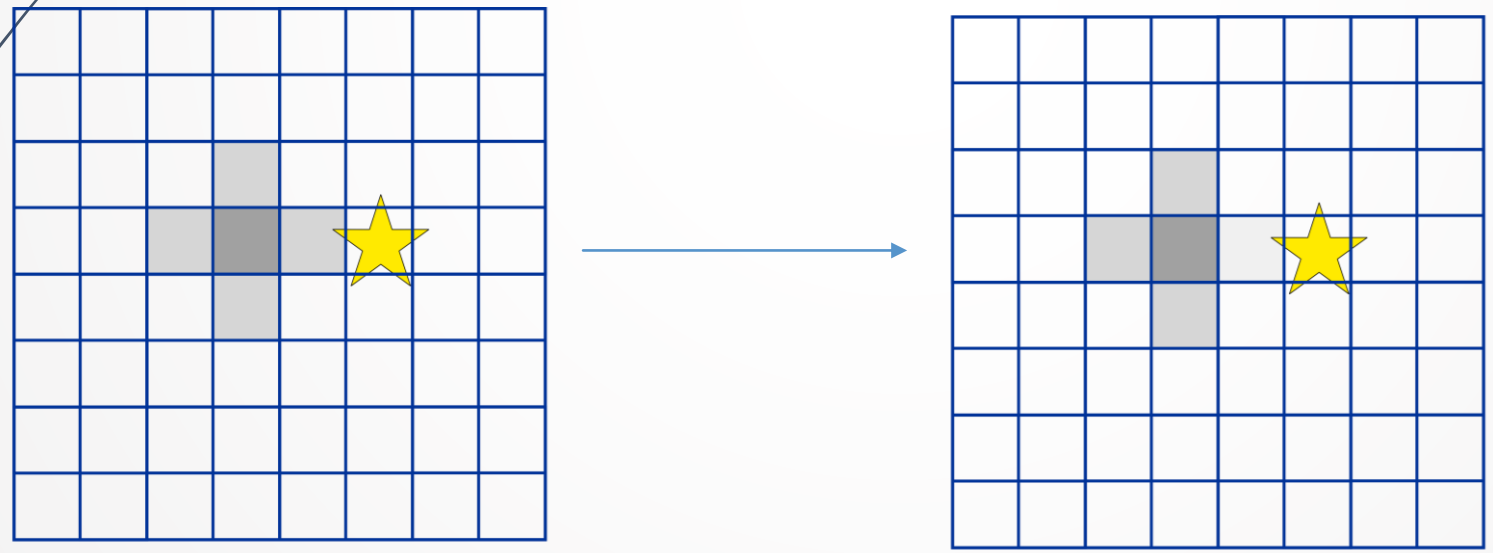
Example: Localization

- t=1, u =east, z=no-marker
- Bayes filter step 1: Apply motion model



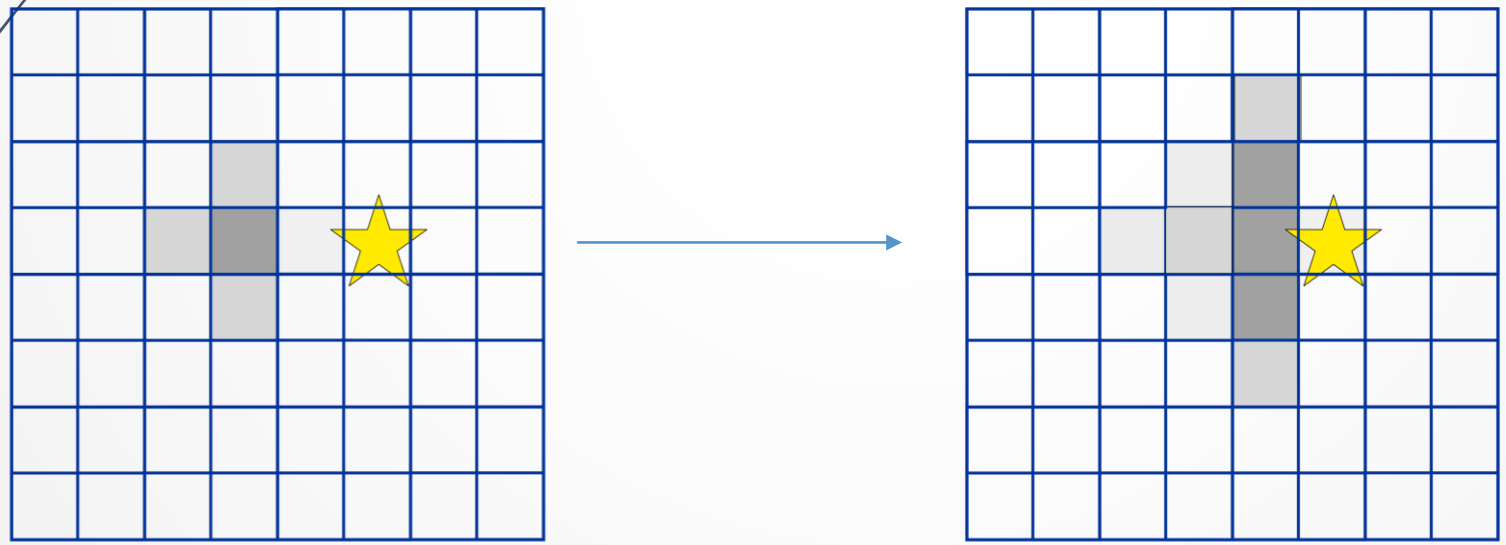
Example: Localization

- t=1, u =east, z=no-marker
- Bayes filter step 2: Apply observation model



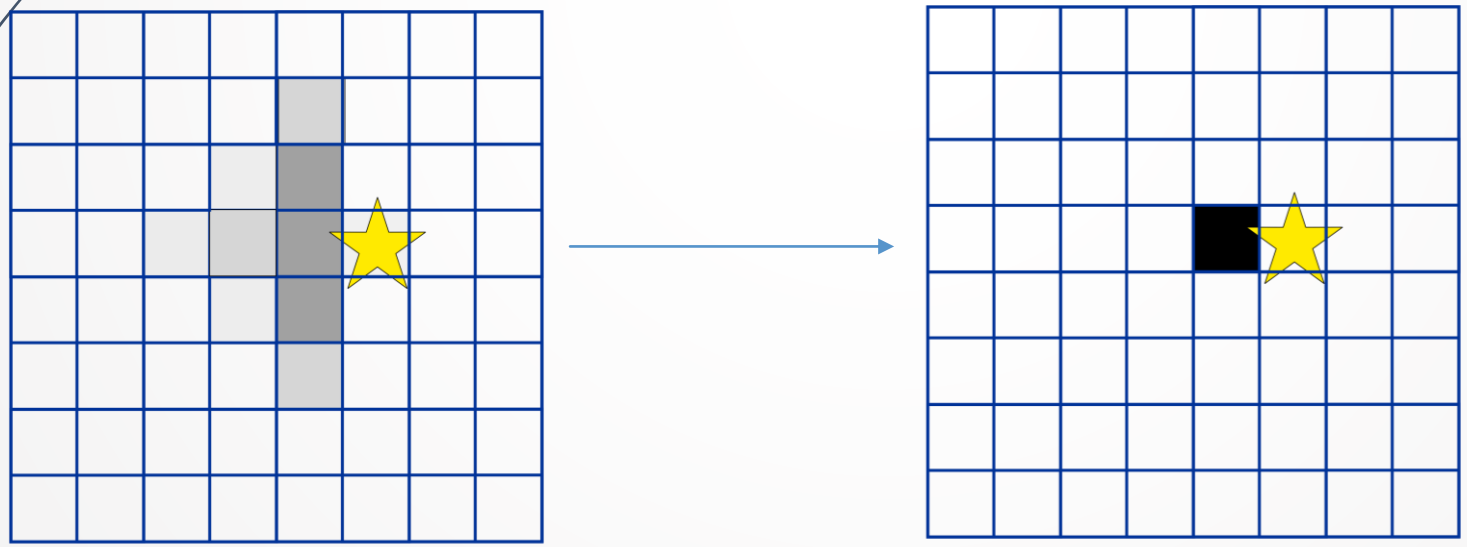
Example: Localization

- t=2, u =east, z=marker
- Bayes filter step 1: Apply motion model



Example: Localization

- t=2, u =east, z=marker
- Bayes filter step 2: Apply observation model
- Question: where is the robot?



SE: Indicative questions

- Explain the Bayes rule.
- Solve a localization problem of a robot operating in a grid-space, able to move E-W, N-S and getting a measurement that is true or false depending on the possible detection of a marker.
- What are the underlying assumptions of Markov processes used for localization?
- Would we be able to localize the robot using the Bayes filter if the marker was moving and we had no knowledge of its motion pattern and values over time?

CS491/691: Introduction to Aerial Robotics

Topic: Flight Control Systems

Dr. Kostas Alexis (CSE)

Controlling a Multirotor along the x-axis

- Assume a single-axis multirotor.
  - The system has to coordinate its pitching motion and thrust to move to the desired point ahead of its axis.
  - Roll is considered to be zero, yaw is considered to be constant. No initial velocity. No motion is expressed in any other axis.
  - A system of only two degrees of freedom.

Controlling a Multirotor along the x-axis

- Simplified linear dynamics

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 1/J_y \end{bmatrix} M_y$$

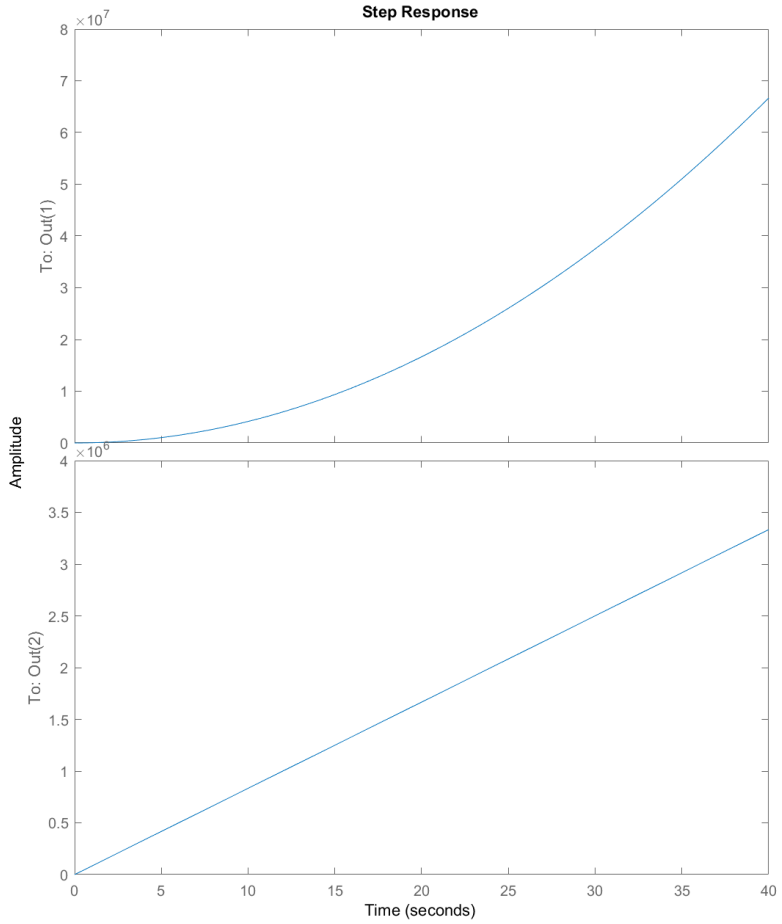
$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ -g \end{bmatrix} \theta$$

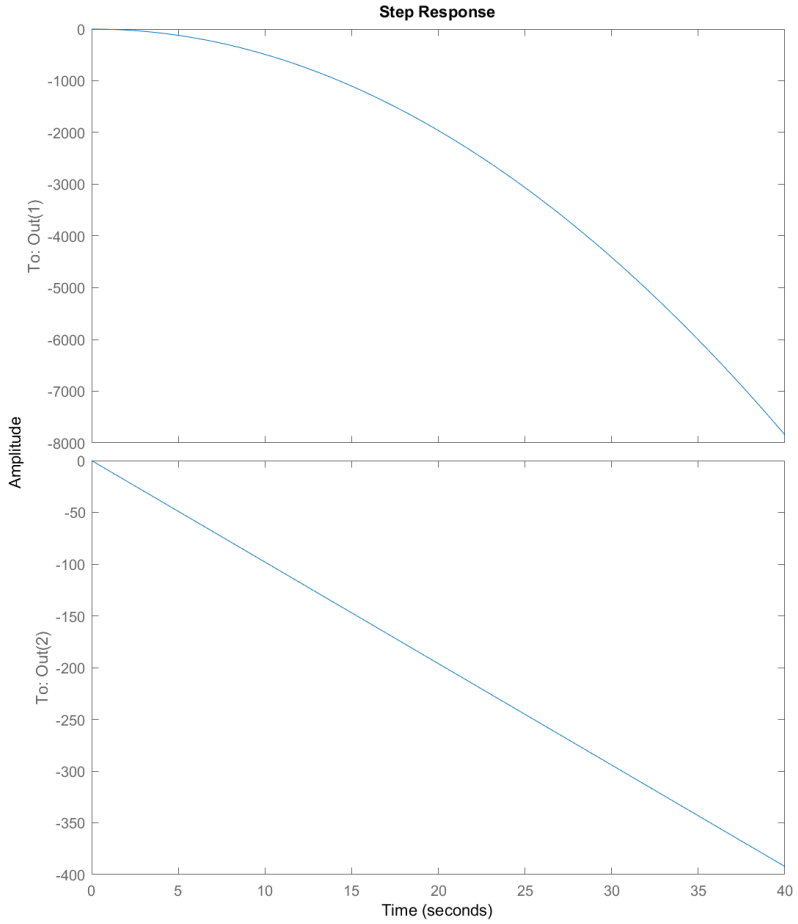
How does this system behave?

Controlling a Multirotor along the x-axis

```
%%  Simple Modeling and Control study
clear;
J_y = 1.2e-5;
g = 9.806; mass = 1.2;

%   Pitch Linear Model
A_p = [0 1; 0 0]; B_p = [0; 1/J_y];
C_p = eye(2); D_p = zeros(2,1);
ss_pitch = ss(A_p,B_p,C_p,D_p);

%   x Linear Model
A_x = [0 1; 0 0]; B_x = [0; -g];
C_x = eye(2); D_x = zeros(2,1);
ss_x = ss(A_x,B_x,C_x,D_x);

%   Observe the Step responses of the system
subplot(1,2,1); step(ss_pitch);
subplot(1,2,2); step(ss_x);
```

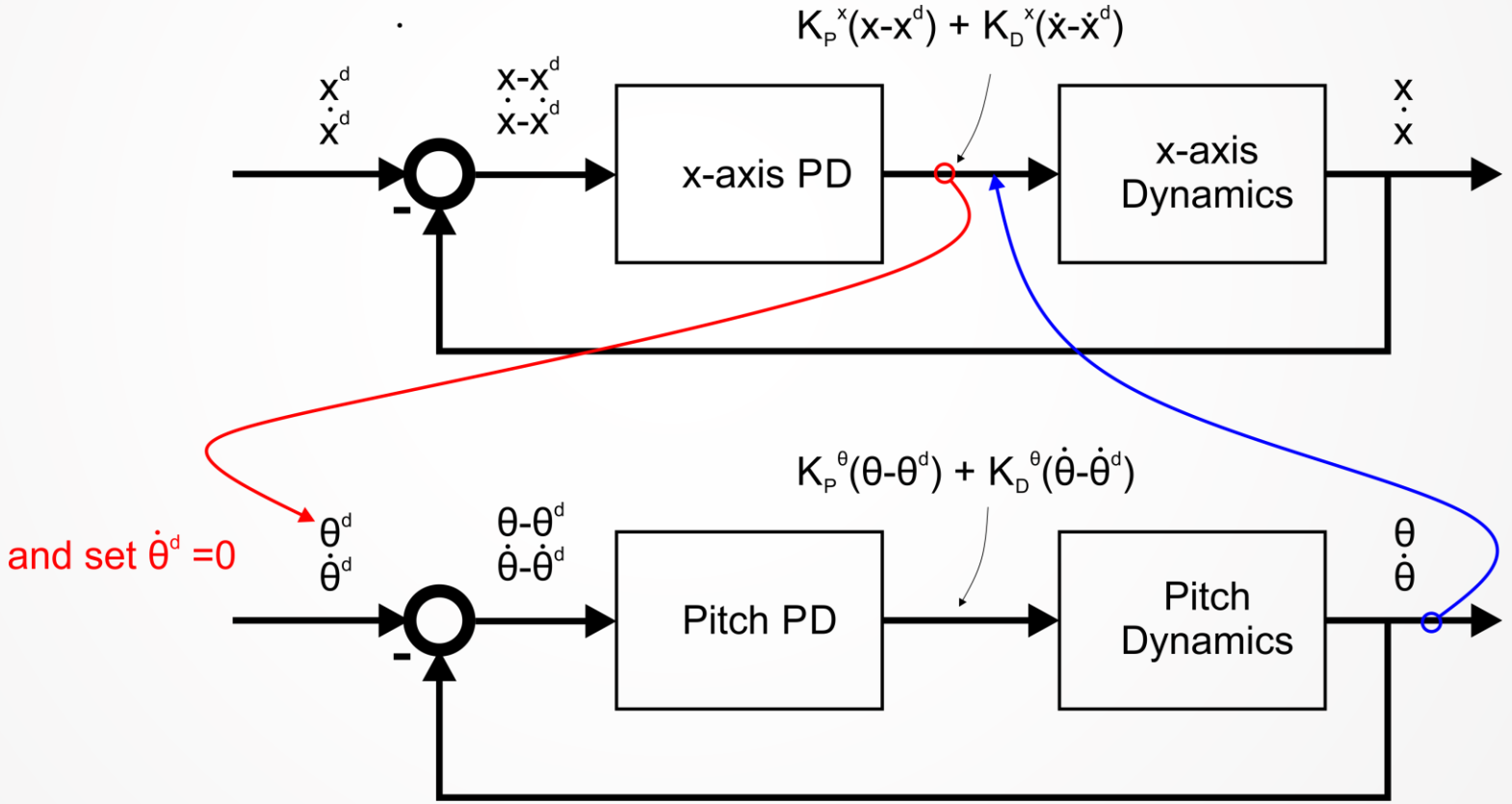
Controlling a Multirotor along the x-axis





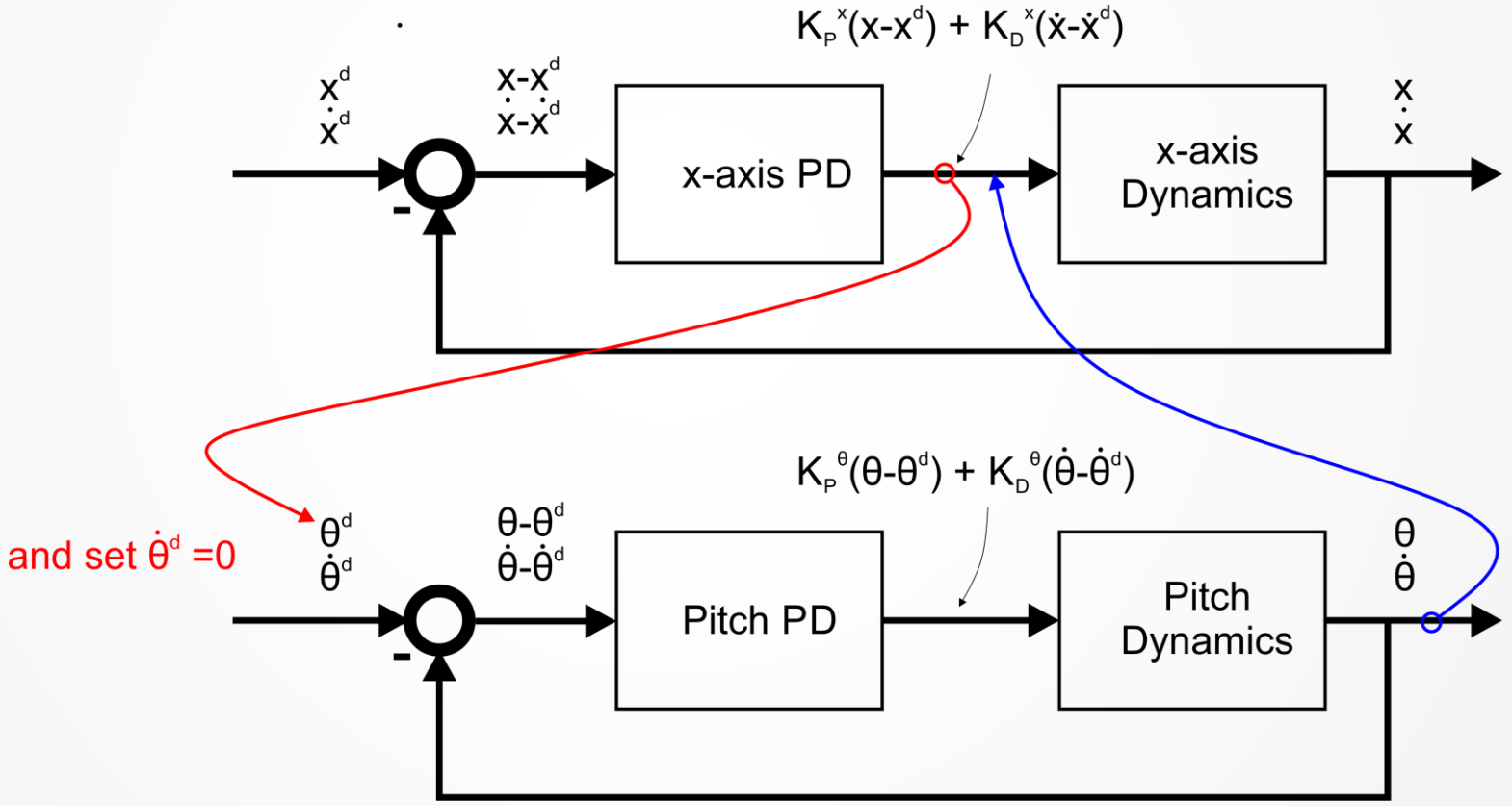
Controlling a Multirotor along the x-axis

- Decoupled Control Structure



Controlling a Multirotor along the x-axis

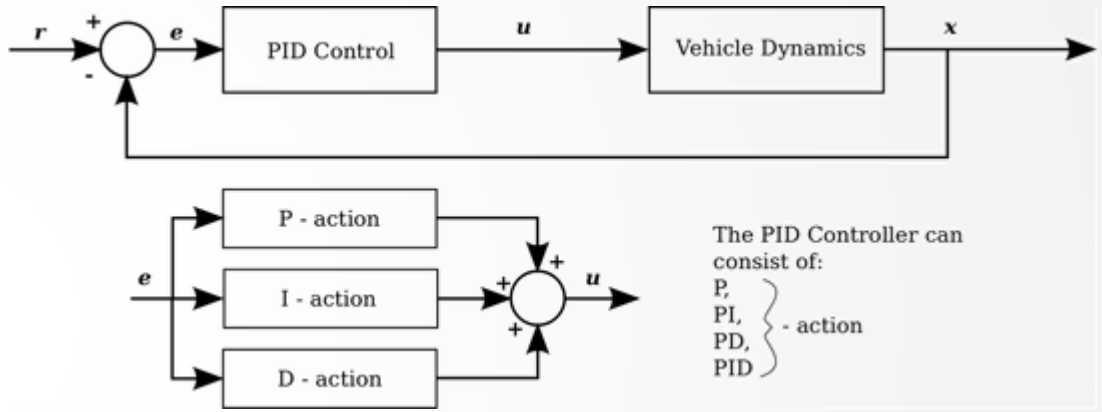
- Decoupled Control Structure



How to select the PD gains?

A mini introduction to PID Control

- PID Control stands for Proportional-Integral-Derivative feedback control and corresponds to one of the most commonly used controllers used in industry.
- It's success is based on its capacity to efficiently and robustly control a variety of processes and dynamic systems, while having an extremely simple structure and intuitive tuning procedures.
- Not comparable in performance with modern control strategies, but still the most common starting point



How to select the PD gains?

Controlling a Multirotor along the x-axis

- MATLAB Implementation

```
%%  Simple Modeling and Control study
clear;
J_y = 1.2e-5;
g = 9.806; mass = 1.2;

%   Pitch Linear Model
A_p = [0 1; 0 0]; B_p = [0; 1/J_y];
C_p = eye(2); D_p = zeros(2,1);
ss_pitch = ss(A_p,B_p,C_p,D_p);

%   x Linear Model
A_x = [0 1; 0 0]; B_x = [0; -g];
C_x = eye(2); D_x = zeros(2,1);
ss_x = ss(A_x,B_x,C_x,D_x);

%   Observe the Step responses of the system
subplot(1,2,1); step(ss_pitch);
subplot(1,2,2); step(ss_x);
```

```
%%  Design the PD Controller for Pitch

ss_pitch_tf = tf(ss_pitch); ss_pitch_tf = ss_pitch_tf(1);
pidTuner(ss_pitch_tf,'PD')
```

```
%%  Design the PD Controller for X translational dynamics

ss_x_tf = tf(ss_x); ss_x_tf = ss_x_tf(1);
pidTuner(ss_x_tf,'PD')
```

```
%%  Verification
close all;

K_P_pitch = 25.8e-5; K_D_pitch = 9.82e-5;
PD_PITCH_GAINS = [K_P_pitch 0; 0  K_D_pitch];

ss_pitch_cl = feedback(PD_PITCH_GAINS*ss_pitch,[1 1]);

K_P_x = -1.17; K_D_x = -0.823;
PD_X_GAINS = [K_P_x 0; 0 K_D_x];
ss_x_cl = feedback(PD_X_GAINS*ss_x,[1 1]);

subplot(1,2,1); step(ss_pitch_cl);
subplot(1,2,2); step(ss_x_cl);
```

Controlling a Multirotor along the x-axis

- MATLAB Implementation

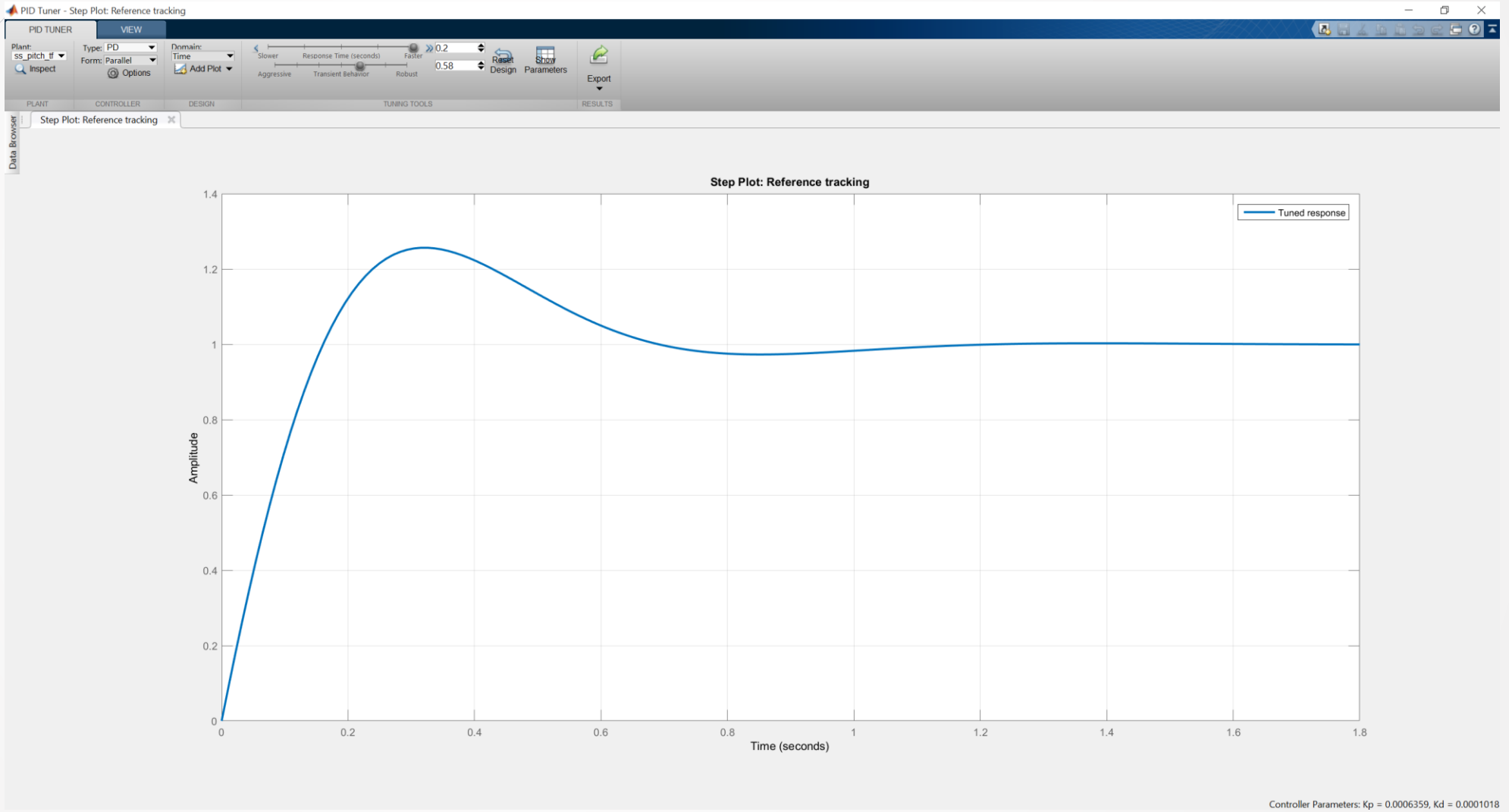
```
%%  Simple Modeling and Control study
clear;
J_y = 1.2e-5;
g = 9.806; mass = 1.2;

%   Pitch Linear Model
A_p = [0 1; 0 0]; B_p = [0; 1/J_y];
C_p = eye(2); D_p = zeros(2,1);
ss_pitch = ss(A_p,B_p,C_p,D_p);

%   x Linear Model
A_x = [0 1; 0 0]; B_x = [0; -g];
C_x = eye(2); D_x = zeros(2,1);
ss_x = ss(A_x,B_x,C_x,D_x);

%   Observe the Step responses of the system
subplot(1,2,1); step(ss_pitch);
subplot(1,2,2); step(ss_x);
```

```
%%  Design the PD Controller for Pitch

ss_pitch_tf = tf(ss_pitch); ss_pitch_tf = ss_pitch_tf(1);
pidTuner(ss_pitch_tf,'PD')
```

```
%%  Design the PD Controller for X translational dynamics

ss_x_tf = tf(ss_x); ss_x_tf = ss_x_tf(1);
pidTuner(ss_x_tf,'PD')
```

```
%%  Verification
close all;

K_P_pitch = 25.8e-5; K_D_pitch = 9.82e-5;
PD_PITCH_GAINS = [K_P_pitch 0; 0  K_D_pitch];

ss_pitch_cl = feedback(PD_PITCH_GAINS*ss_pitch,[1 1]);

K_P_x = -1.17; K_D_x = -0.823;
PD_X_GAINS = [K_P_x 0; 0 K_D_x];
ss_x_cl = feedback(PD_X_GAINS*ss_x,[1 1]);

subplot(1,2,1); step(ss_pitch_cl);
subplot(1,2,2); step(ss_x_cl);
```

Controlling a Multirotor along the x-axis



Controller Parameters: Kp = 0.0006359, Kd = 0.0001018

Controlling a Multirotor along the x-axis

- MATLAB Implementation

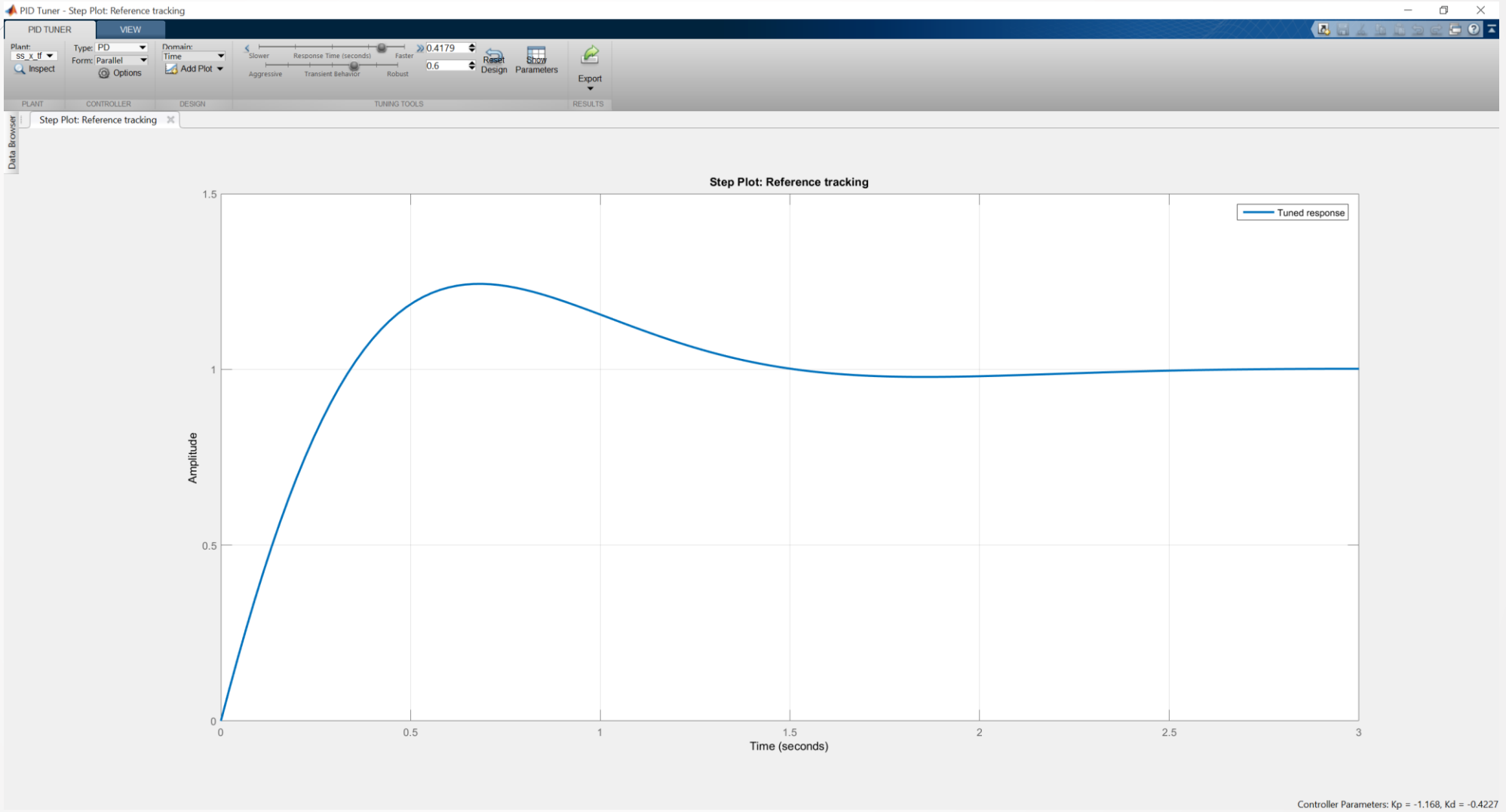
```
%%  Simple Modeling and Control study
clear;
J_y = 1.2e-5;
g = 9.806; mass = 1.2;

%   Pitch Linear Model
A_p = [0 1; 0 0]; B_p = [0; 1/J_y];
C_p = eye(2); D_p = zeros(2,1);
ss_pitch = ss(A_p,B_p,C_p,D_p);

%   x Linear Model
A_x = [0 1; 0 0]; B_x = [0; -g];
C_x = eye(2); D_x = zeros(2,1);
ss_x = ss(A_x,B_x,C_x,D_x);

%   Observe the Step responses of the system
subplot(1,2,1); step(ss_pitch);
subplot(1,2,2); step(ss_x);
```

```
%%  Design the PD Controller for Pitch

ss_pitch_tf = tf(ss_pitch); ss_pitch_tf = ss_pitch_tf(1);
pidTuner(ss_pitch_tf,'PD')
```

```
%%  Design the PD Controller for X translational dynamics

ss_x_tf = tf(ss_x); ss_x_tf = ss_x_tf(1);
pidTuner(ss_x_tf,'PD')
```

```
%%  Verification
close all;

K_P_pitch = 25.8e-5; K_D_pitch = 9.82e-5;
PD_PITCH_GAINS = [K_P_pitch 0; 0  K_D_pitch];

ss_pitch_cl = feedback(PD_PITCH_GAINS*ss_pitch,[1 1]);

K_P_x = -1.17; K_D_x = -0.823;
PD_X_GAINS = [K_P_x 0; 0 K_D_x];
ss_x_cl = feedback(PD_X_GAINS*ss_x,[1 1]);

subplot(1,2,1); step(ss_pitch_cl);
subplot(1,2,2); step(ss_x_cl);
```

Controlling a Multirotor along the x-axis



Controlling a Multirotor along the x-axis

- MATLAB Implementation

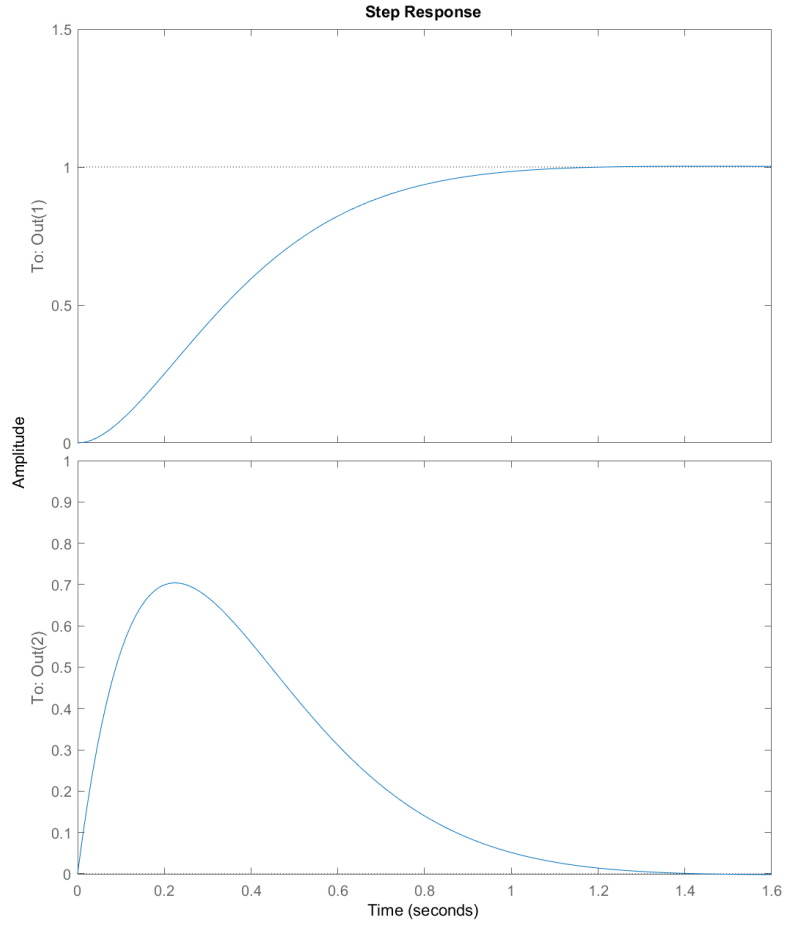
```
%%  Simple Modeling and Control study
clear;
J_y = 1.2e-5;
g = 9.806; mass = 1.2;

%   Pitch Linear Model
A_p = [0 1; 0 0]; B_p = [0; 1/J_y];
C_p = eye(2); D_p = zeros(2,1);
ss_pitch = ss(A_p,B_p,C_p,D_p);

%   x Linear Model
A_x = [0 1; 0 0]; B_x = [0; -g];
C_x = eye(2); D_x = zeros(2,1);
ss_x = ss(A_x,B_x,C_x,D_x);

%   Observe the Step responses of the system
subplot(1,2,1); step(ss_pitch);
subplot(1,2,2); step(ss_x);
```

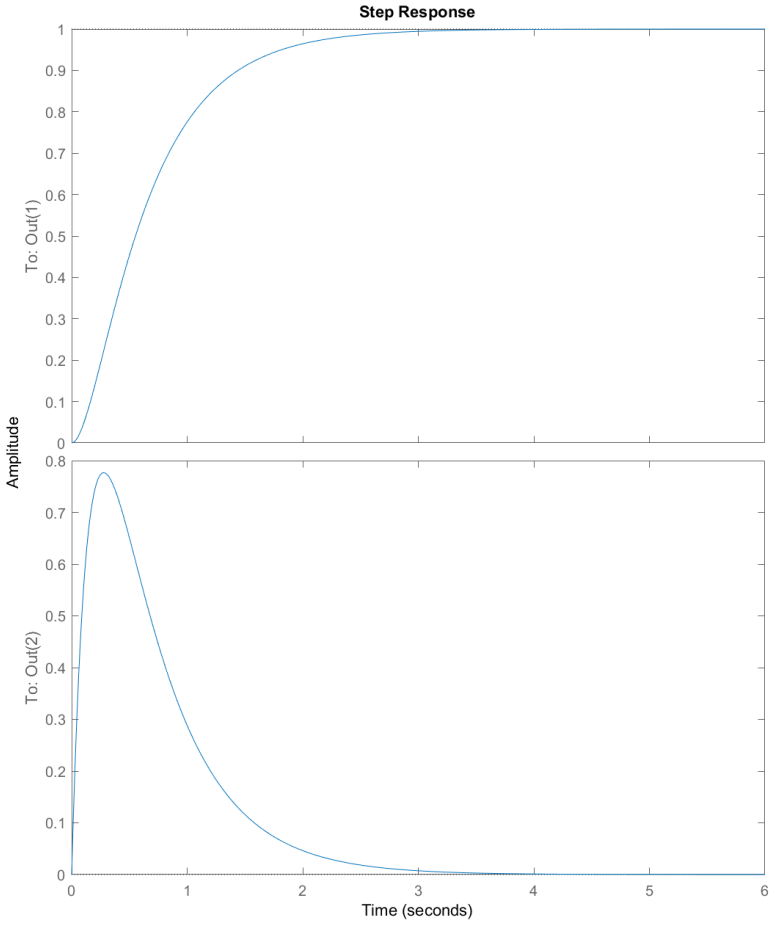
```
%%  Design the PD Controller for Pitch

ss_pitch_tf = tf(ss_pitch); ss_pitch_tf = ss_pitch_tf(1);
pidTuner(ss_pitch_tf,'PD')
```

```
%%  Design the PD Controller for X translational dynamics

ss_x_tf = tf(ss_x); ss_x_tf = ss_x_tf(1);
pidTuner(ss_x_tf,'PD')
```

```
%%  Verification
close all;

K_P_pitch = 25.8e-5; K_D_pitch = 9.82e-5;
PD_PITCH_GAINS = [K_P_pitch 0; 0  K_D_pitch];

ss_pitch_cl = feedback(PD_PITCH_GAINS*ss_pitch,[1 1]);

K_P_x = -1.17; K_D_x = -0.823;
PD_X_GAINS = [K_P_x 0; 0 K_D_x];
ss_x_cl = feedback(PD_X_GAINS*ss_x,[1 1]);

subplot(1,2,1); step(ss_pitch_cl);
subplot(1,2,2); step(ss_x_cl);
```

Controlling a Multirotor along the x-axis





FCS: Indicative questions

- Which of the following controllers can ensure stability of the open loop, linearized, decoupled pitch dynamics of a quadrotor aerial robot? a)Proportional control, b) Proportional-Derivative control, c) Proportional-Integral-Derivative
- Can a proportional gain-only controller that relies on the absolute position error, stabilize the open-loop, linearized, decoupled x-axis dynamics a quadrotor aerial robot?
- Can a proportional gain-only controller that relies on the absolute position error, stabilize the open-loop, linearized, decoupled pitch-rate dynamics a quadrotor aerial robot?
- Describe the role of each of the proportional, integral, derivative gains

CS491/691: Introduction to Aerial Robotics

Topic: Motion Planning

Dr. Kostas Alexis (CSE)

Fundamental motion planning problem

- Consider a dynamical control system defined by an ODE of the form:

$$\frac{dx}{dt} = f(x,u), x(0) = x_{init} \ (1)$$

- Where is x the state, u is the control.
- Given an obstacle set X_{obs}, and a goal set X_{goal}, the objective of the motion planning problem is to find, if it exists, a control signal u such that the solution of (1) satisfies $x(t) \notin X_{obs}$ for all $t \in R^+$, and $x(t) \in X_{goal}$ for all $t > T$, for some finite $T \geq 0$. Return failure if no such control signal exists.

- Basic problem in robotics
- Provably hard: a basic version of it (the Generalized Piano Mover's problem) is known to be PSPACE-hard.

Sampling-based algorithms

- A recently proposed class of motion planning algorithms that has been very successful in practice is based on (batch or incremental) sampling methods:
  - S**olutions are computed based on samples drawn from some distribution.** Sampling algorithms retain some form of completeness, e.g., probabilistic or resolution completeness.
- **Incremental sampling methods are particularly attractive:**
  - Incremental sampling algorithms lend themselves easily to real-time, on-line implementation.
  - Applicable to very generic dynamical systems.
  - Do not require the explicit enumeration of constraints.
  - Adaptively multi-resolution methods (i.e. make your own grid as you go along, up to the necessary resolution).

Rapidly-exploring Random Trees

- Introduced by LaValle and Kuffner in 1998.
- Appropriate for single-query planning problems.
- **Idea:** build (online) a tree, exploring the region of the state space that can be reached from the initial condition.

- **At each step:** sample one point from X_{free}, and try to connect it to the closest vertex in the tree.
- Very effective in practice, "Voronoi bias".

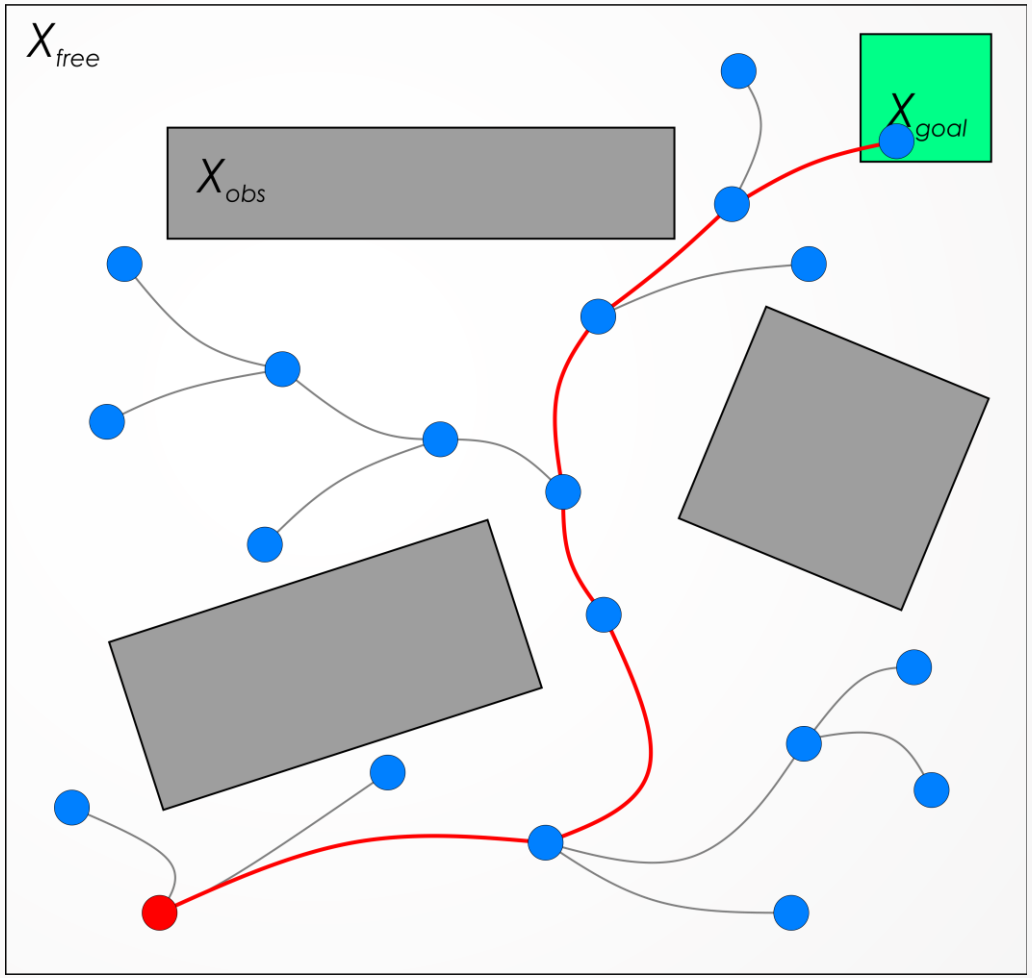
Rapidly-exploring Random Trees

RRT

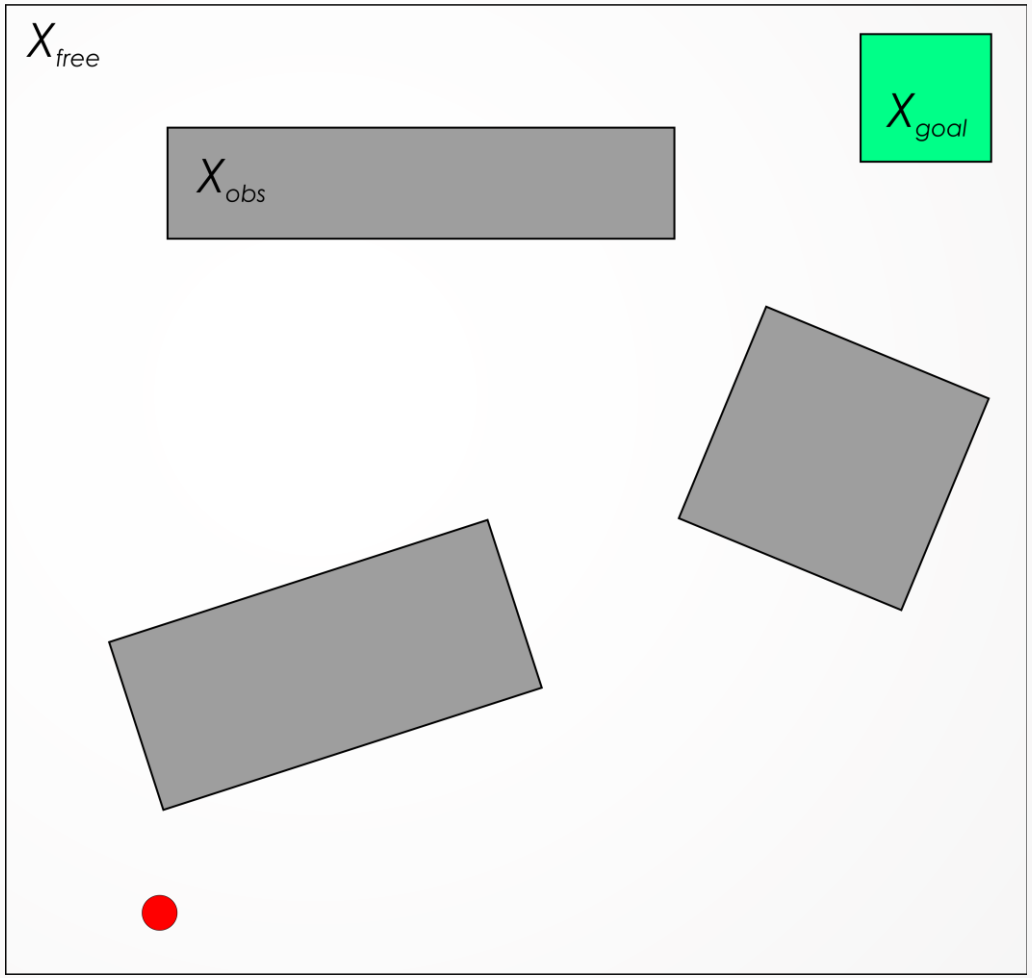
$V \leftarrow \{x_{init}\}; E \leftarrow 0;$
for i=1,…,N **do**:
    $x_{rand} \leftarrow SampleFree;$
    $x_{nearest} \leftarrow Nearest(G = (V, E), x_{rand});$
    $x_{new} \leftarrow Steer(x_{nearest}, x_{rand});$
    if $ObstacleFree(x_{nearest}, x_{new})$ **then**:
        $V \leftarrow V \cup \{x_{new}\}; E \leftarrow E \cup \{(x_{nearest}, x_{new})\};$
return $G = (V, E);$

- **The RRT algorithm is probabilistically complete**
- The probability of success goes to 1 exponentially fast, if the environment satisfies certain "good visibility" conditions.

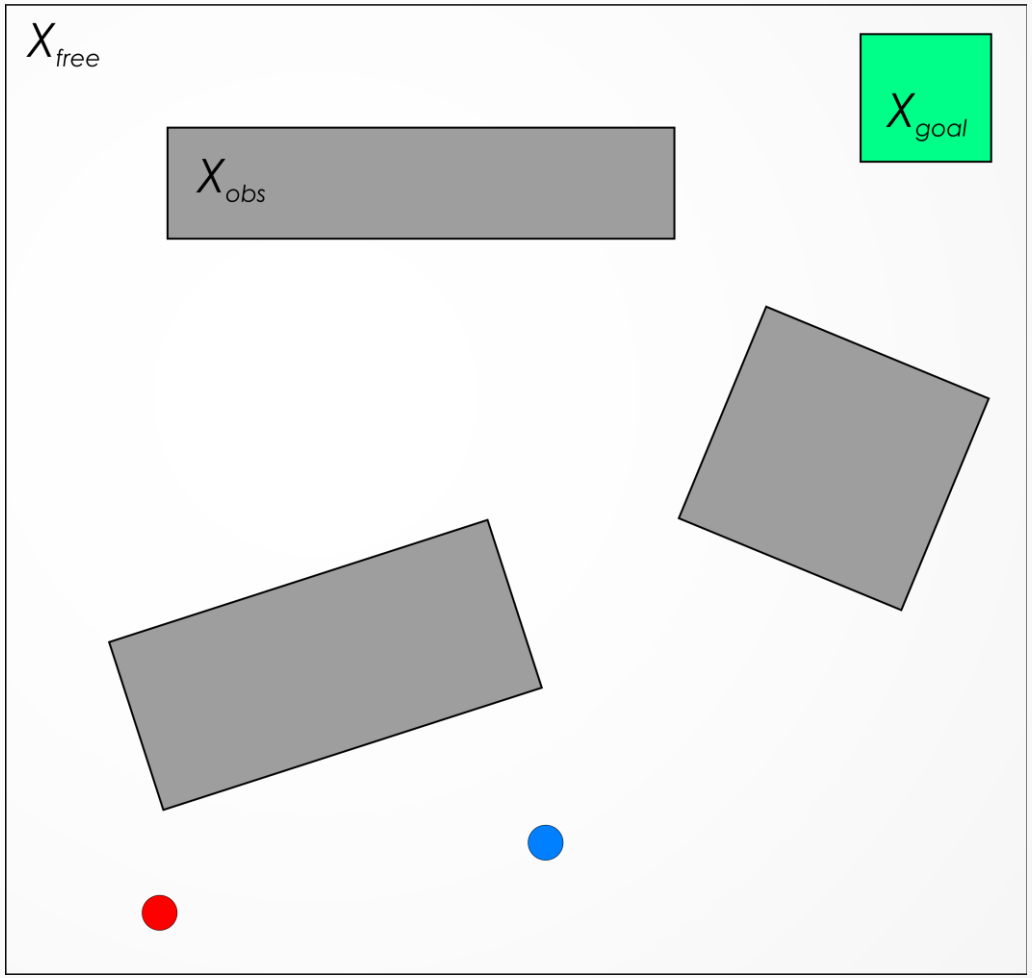
Rapidly-exploring Random Trees (RRTs)



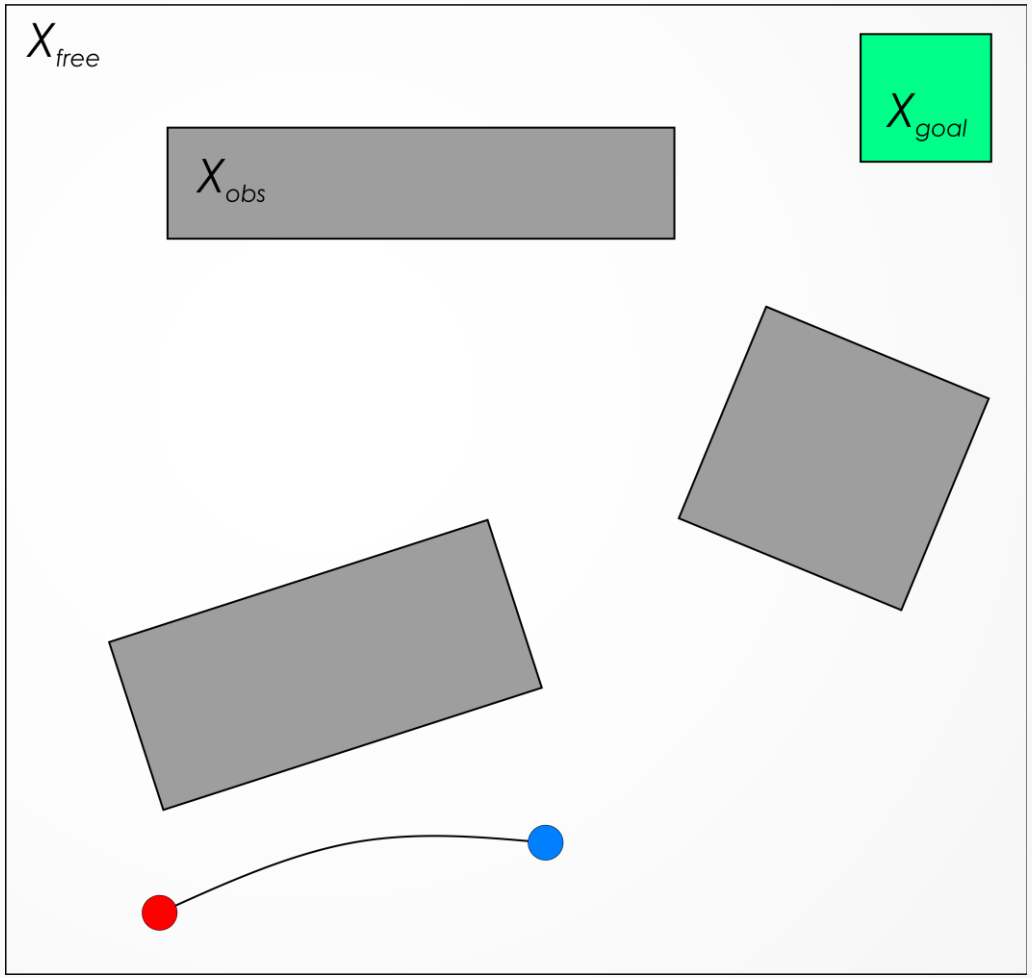
Rapidly-exploring Random Trees (RRTs)



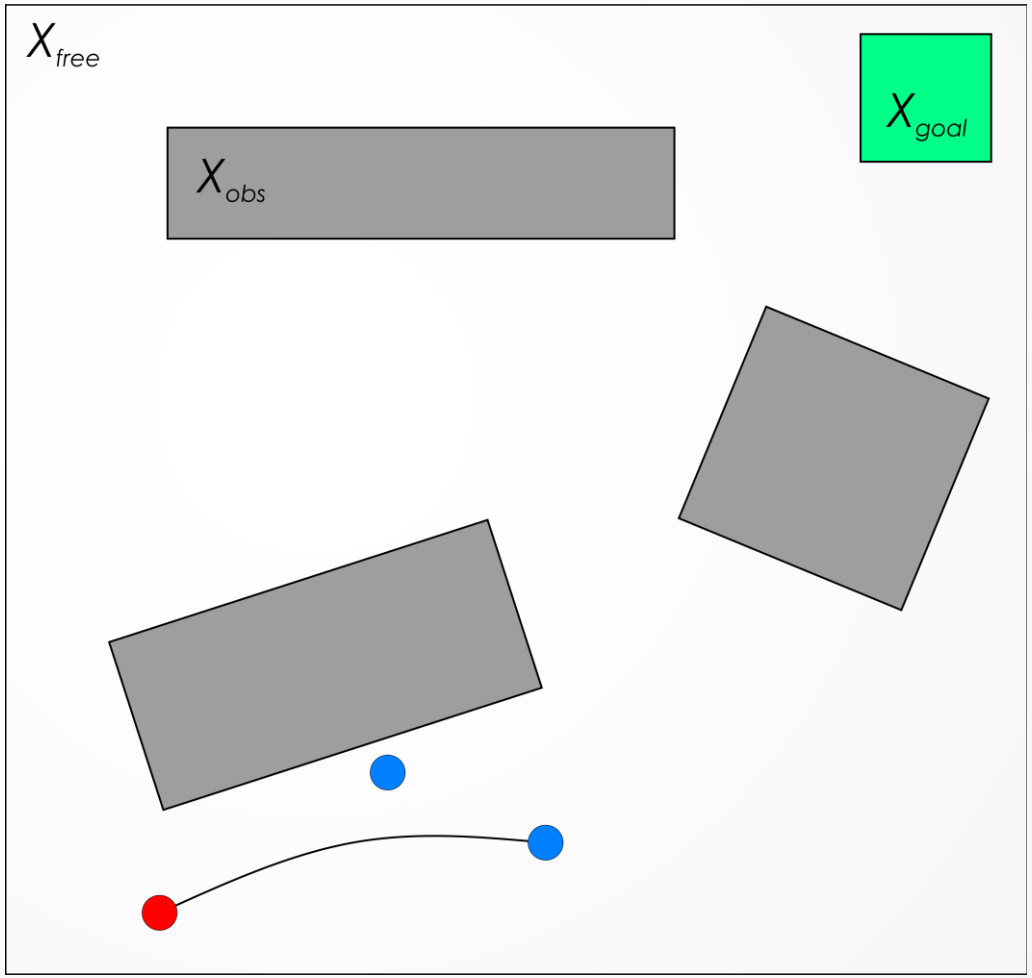
Rapidly-exploring Random Trees (RRTs)



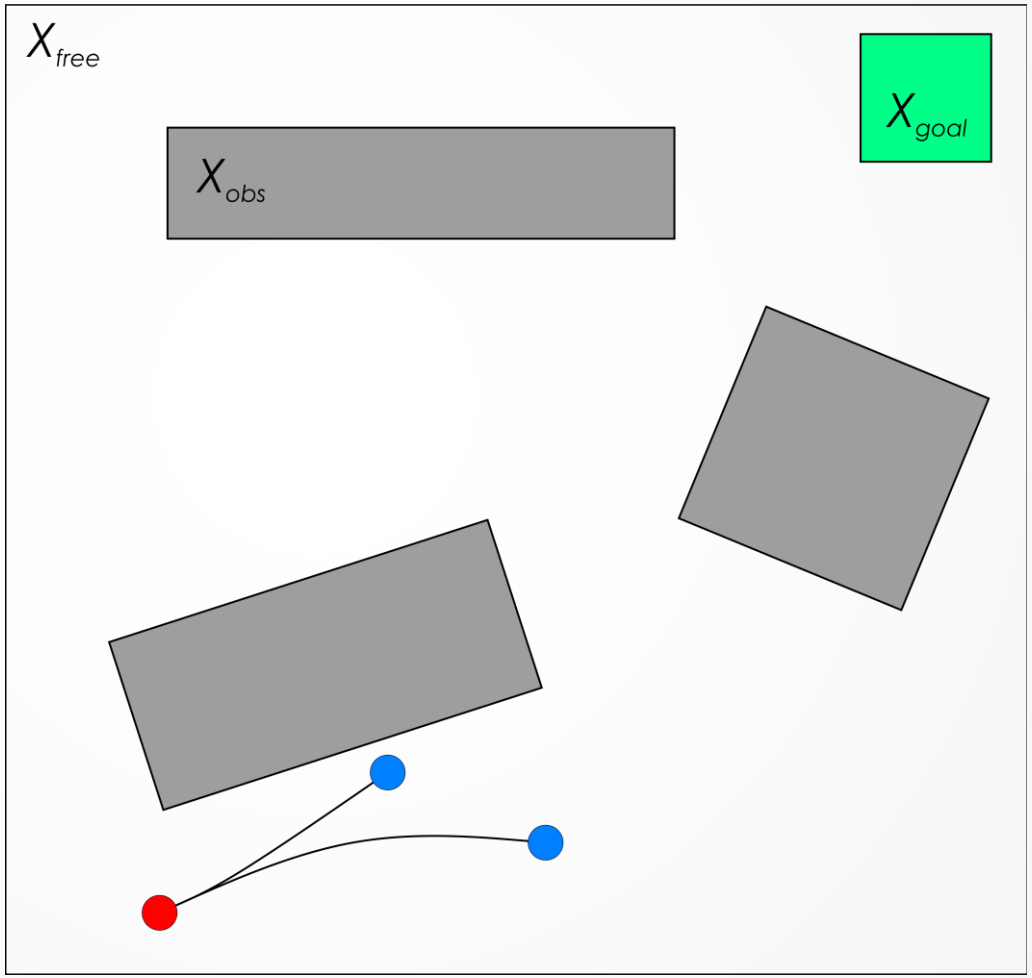
Rapidly-exploring Random Trees (RRTs)



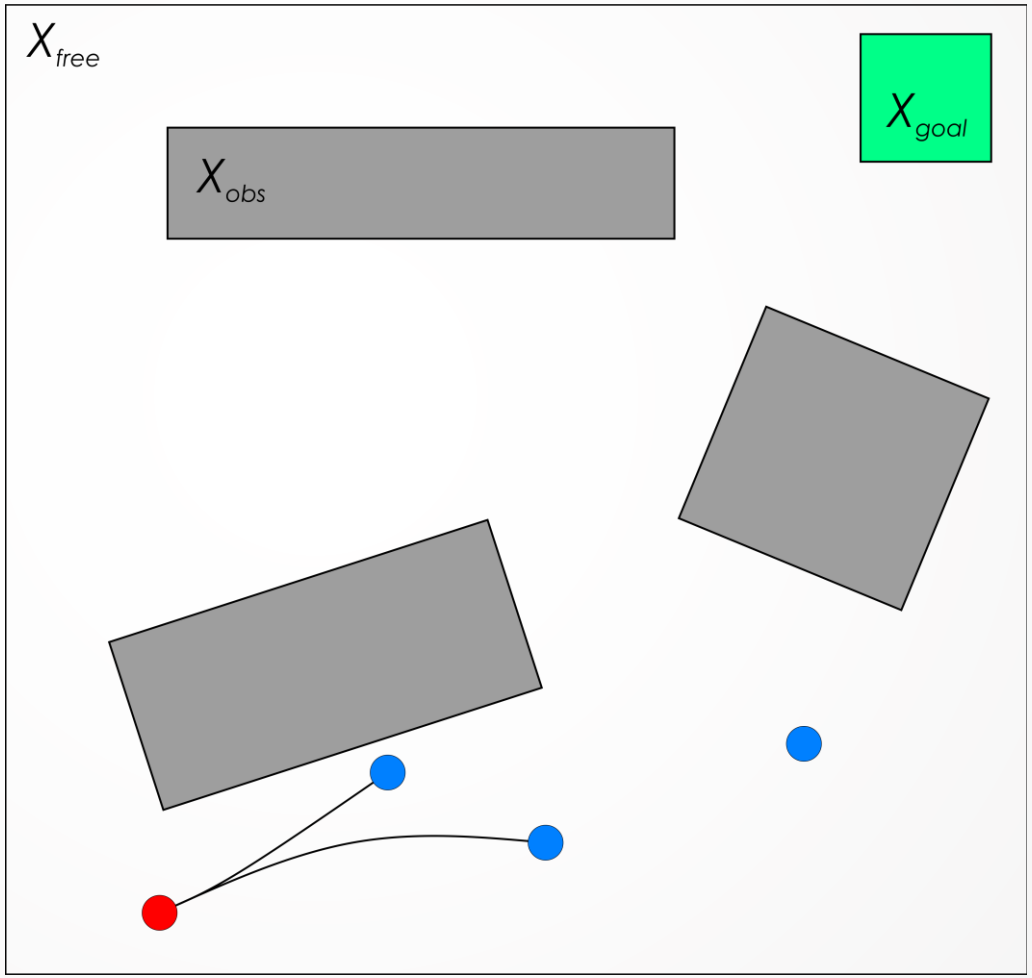
Rapidly-exploring Random Trees (RRTs)



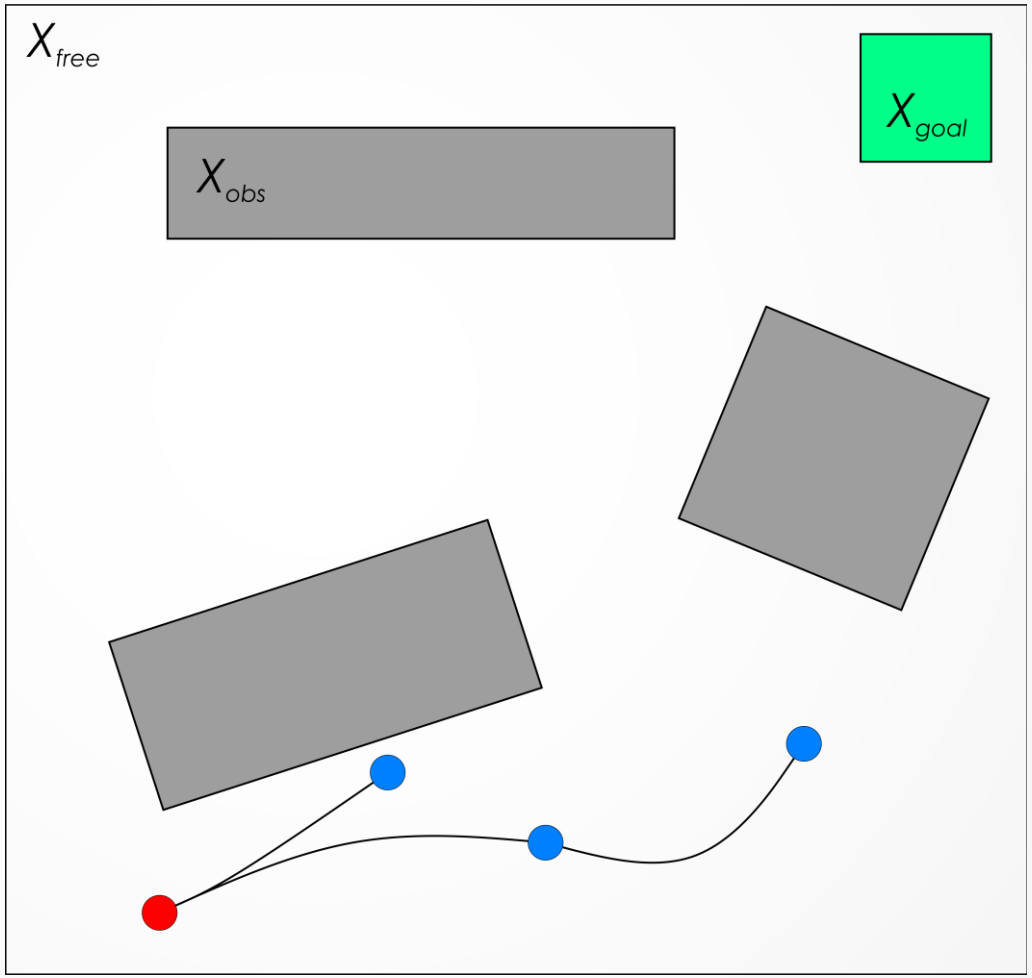
Rapidly-exploring Random Trees (RRTs)



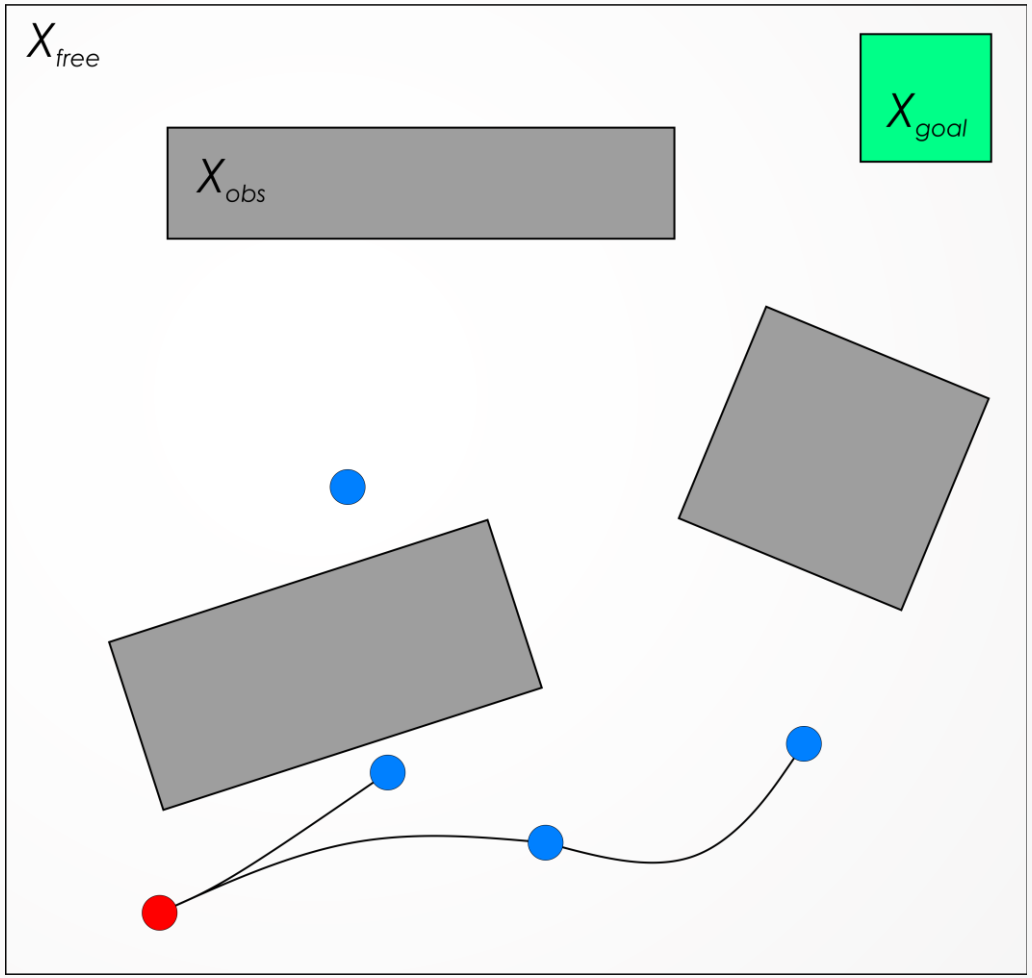
Rapidly-exploring Random Trees (RRTs)



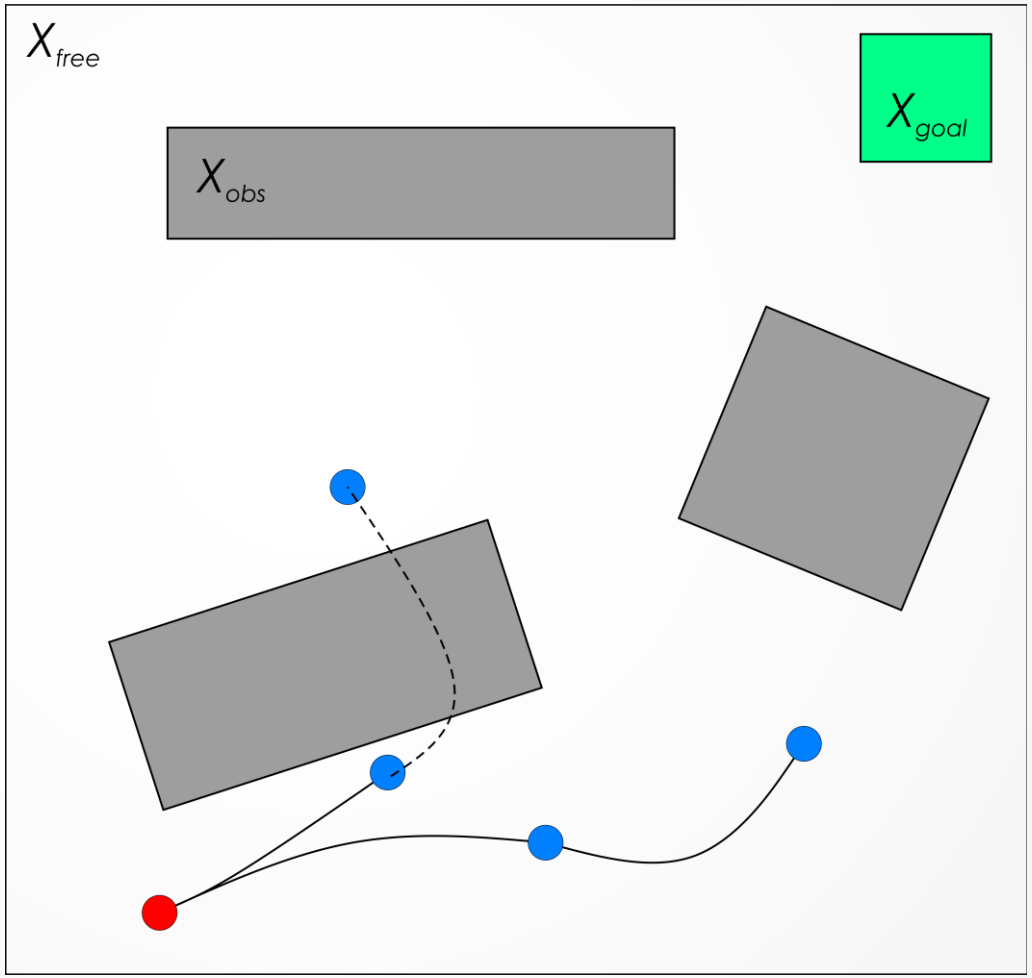
Rapidly-exploring Random Trees (RRTs)



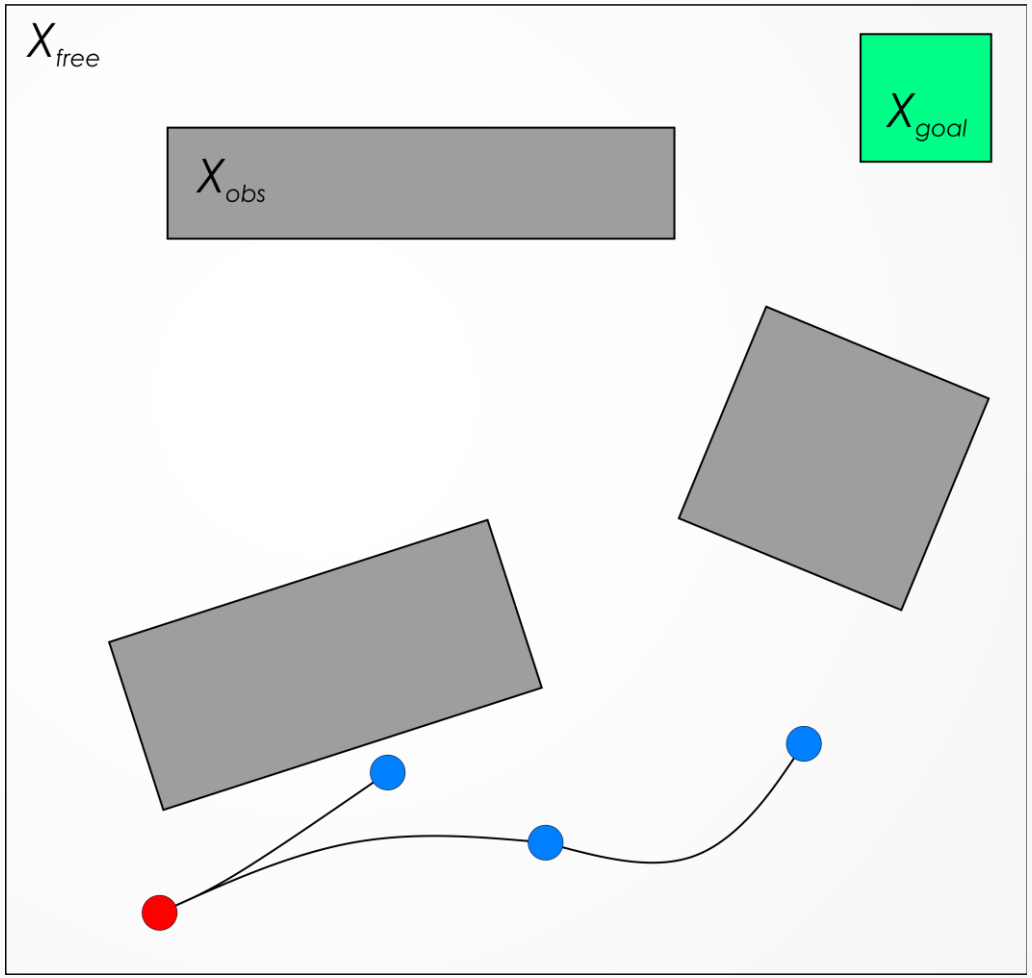
Rapidly-exploring Random Trees (RRTs)



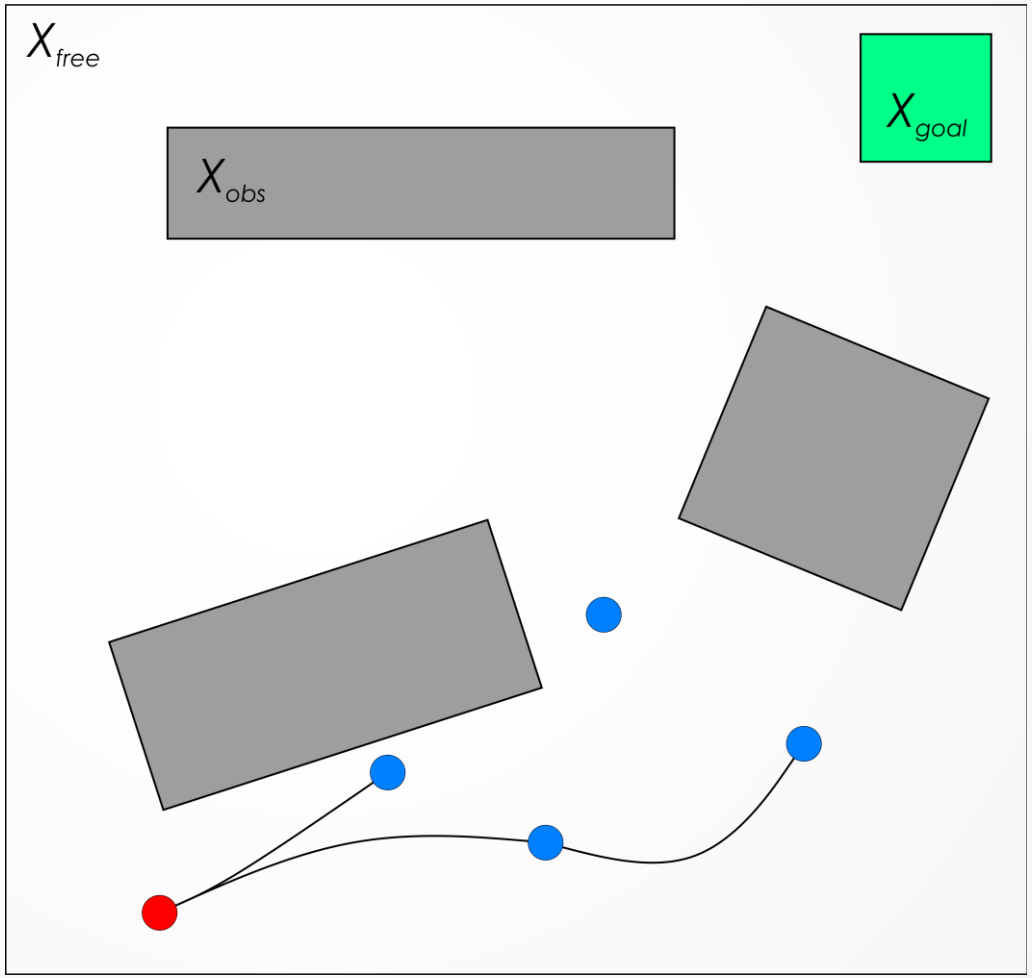
Rapidly-exploring Random Trees (RRTs)



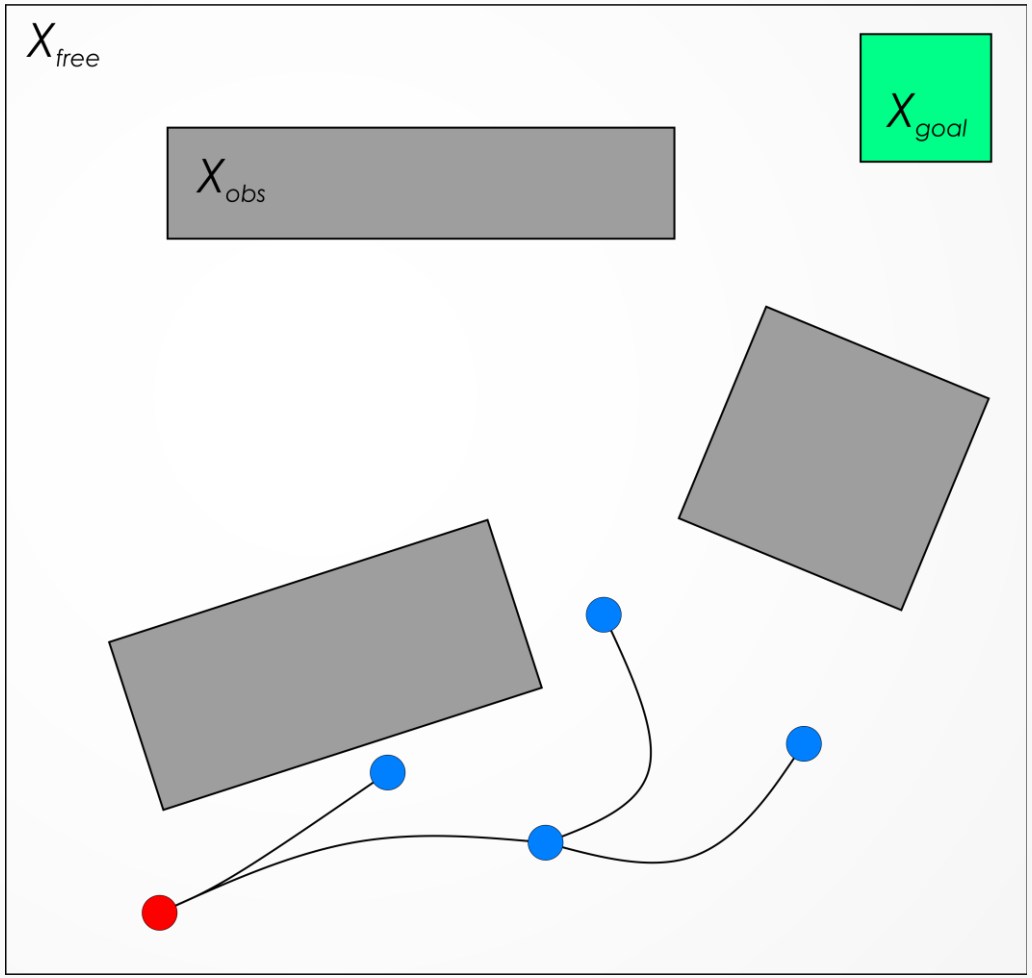
Rapidly-exploring Random Trees (RRTs)



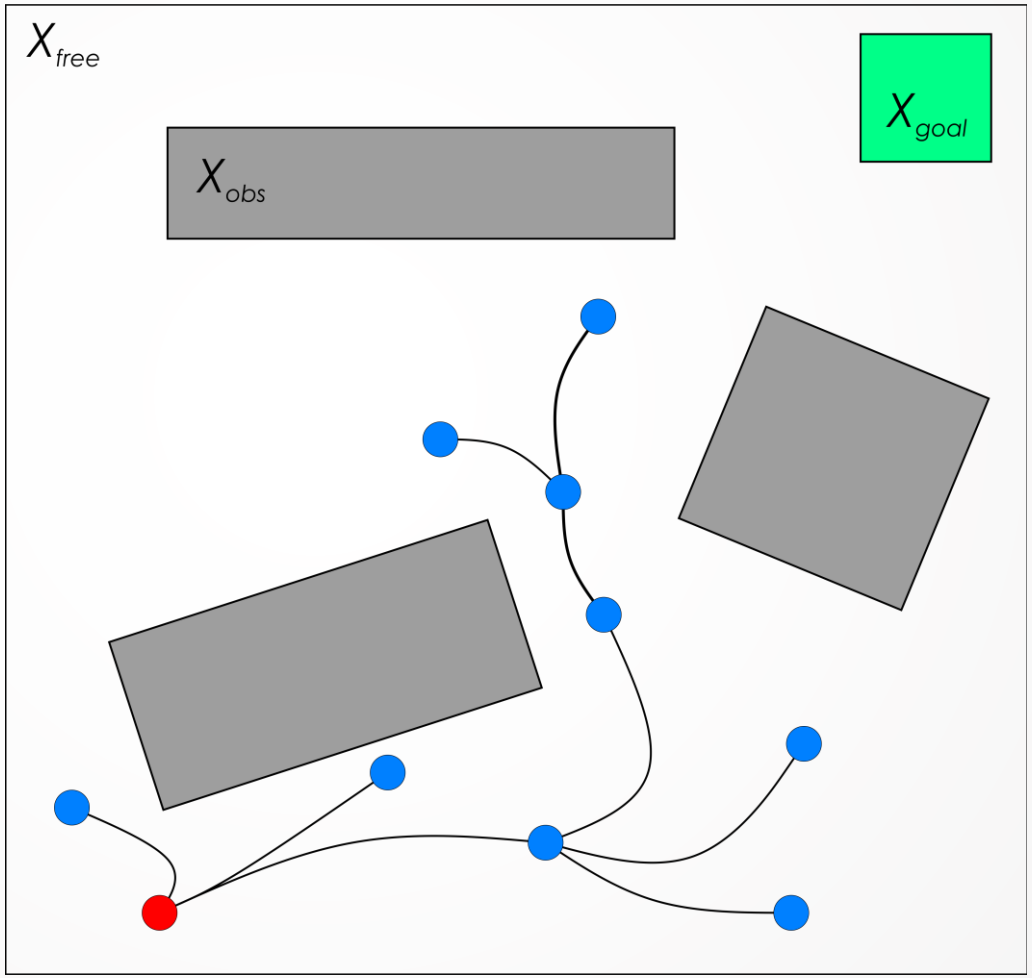
Rapidly-exploring Random Trees (RRTs)



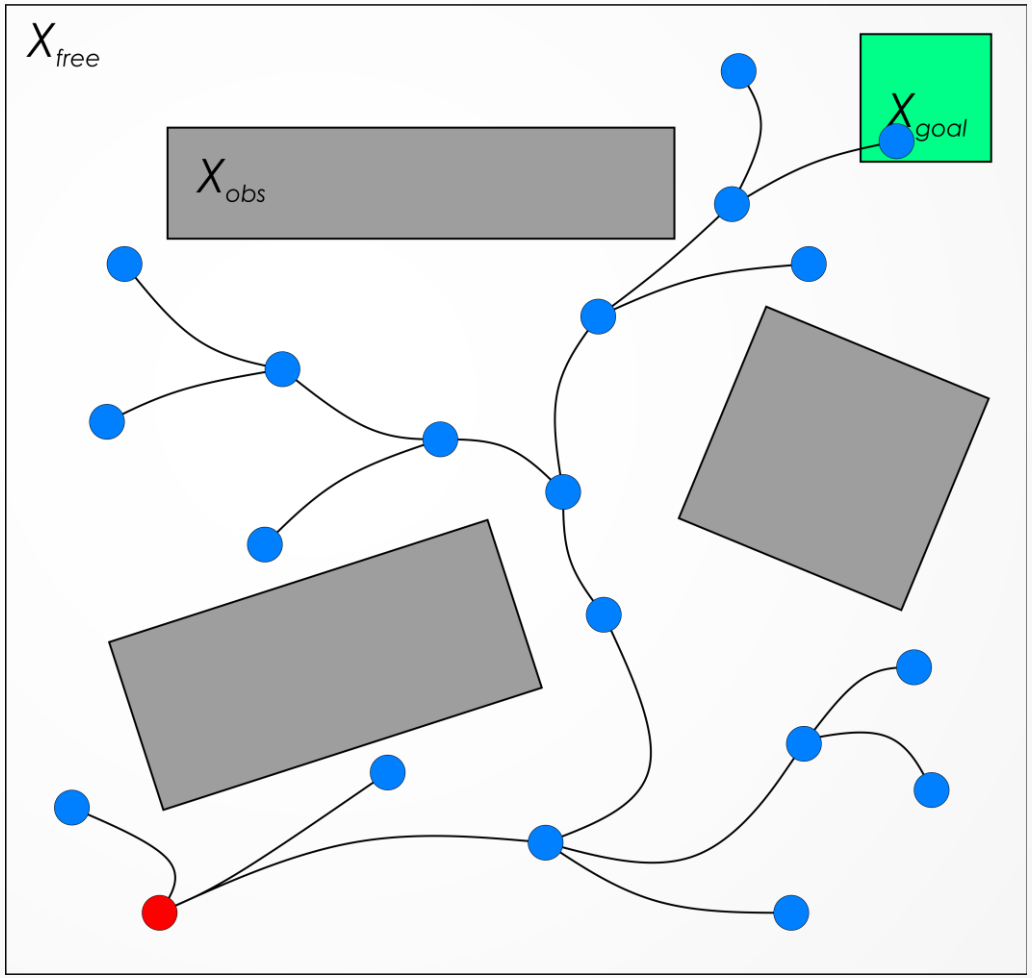
Rapidly-exploring Random Trees (RRTs)



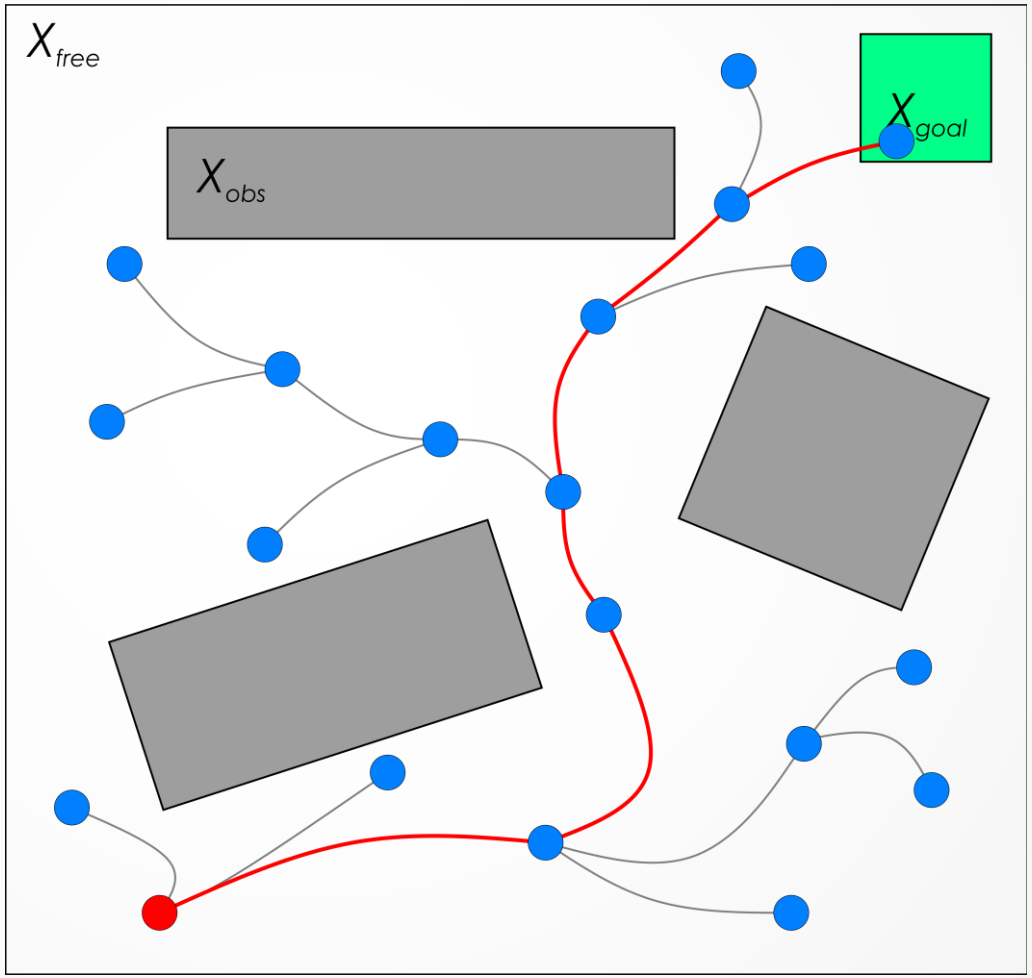
Rapidly-exploring Random Trees (RRTs)



Rapidly-exploring Random Trees (RRTs)



Rapidly-exploring Random Trees (RRTs)

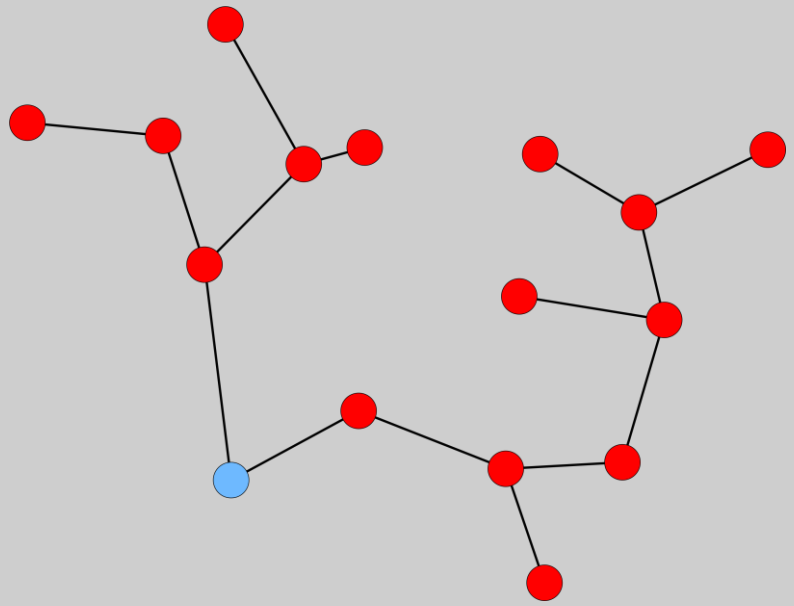


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic systems.

RRT* Algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

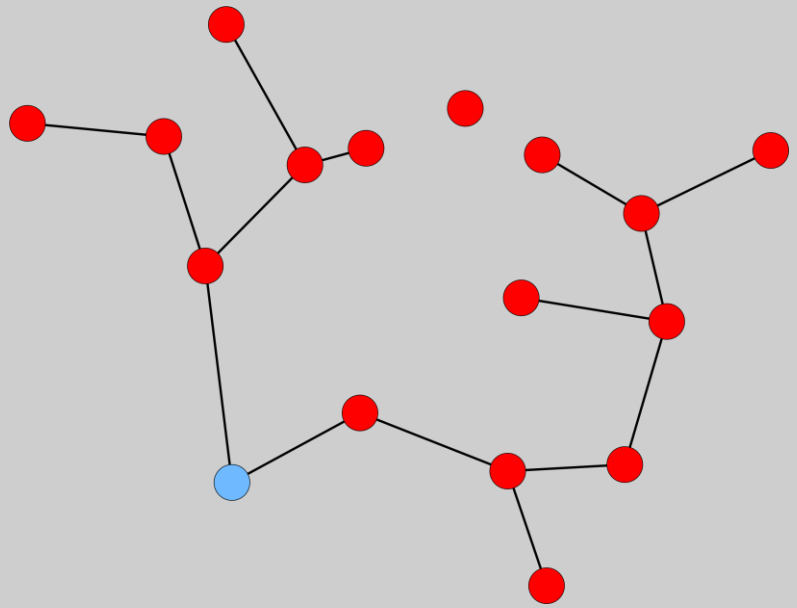


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic systems.

RRT* Algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

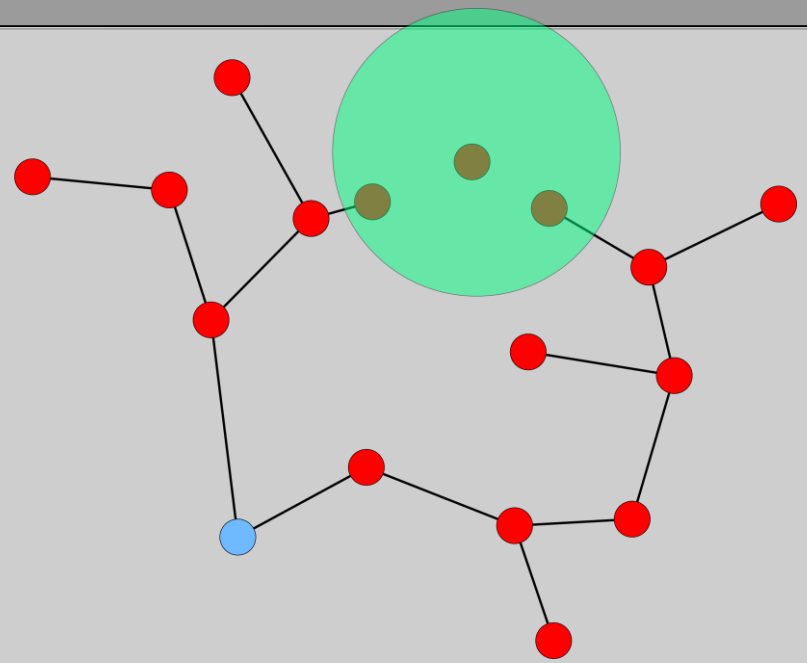


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic systems.

RRT* Algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

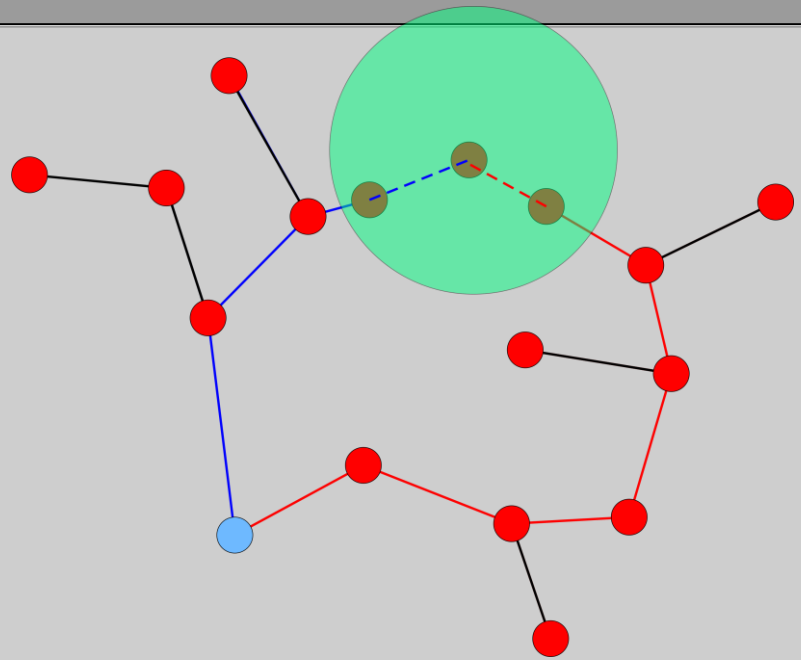


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic systems.

RRT* Algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

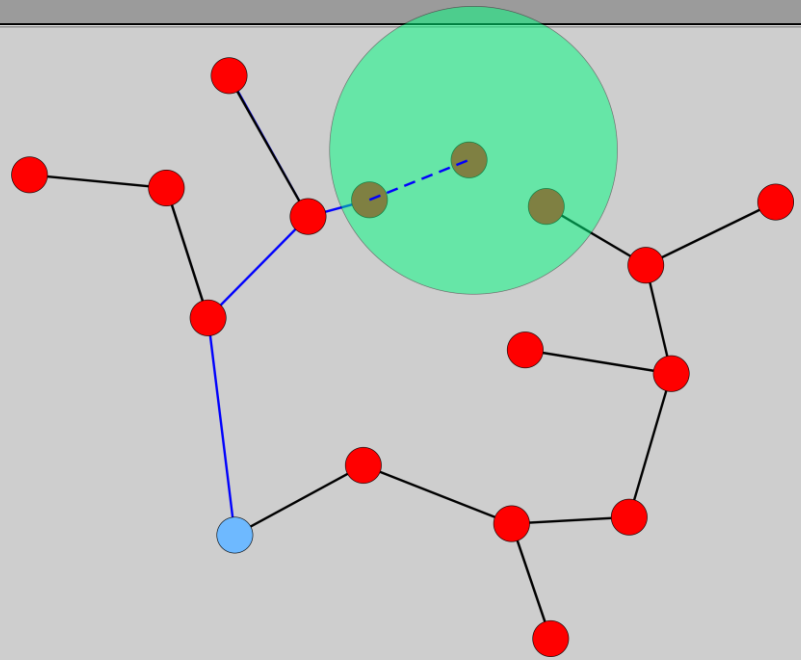


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic systems.

RRT* Algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

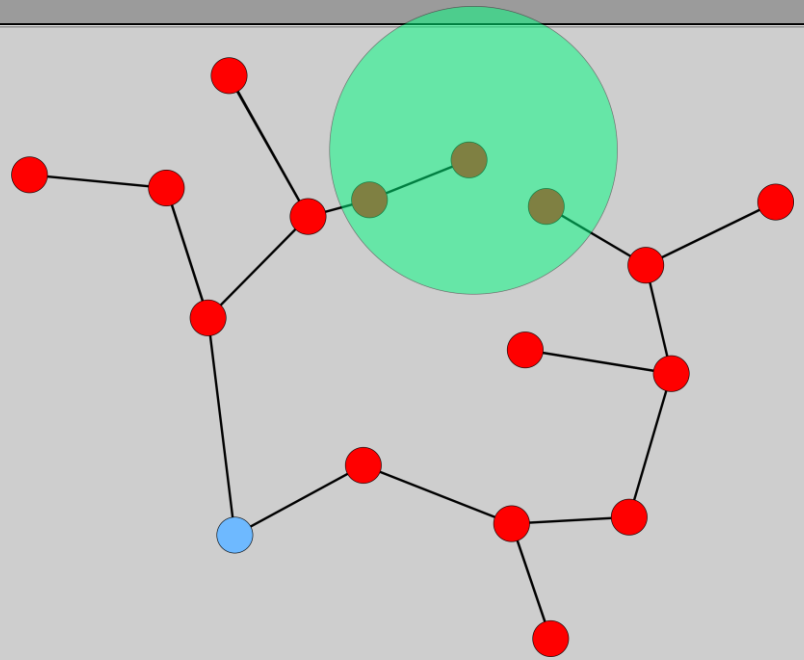


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic systems.

RRT* Algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

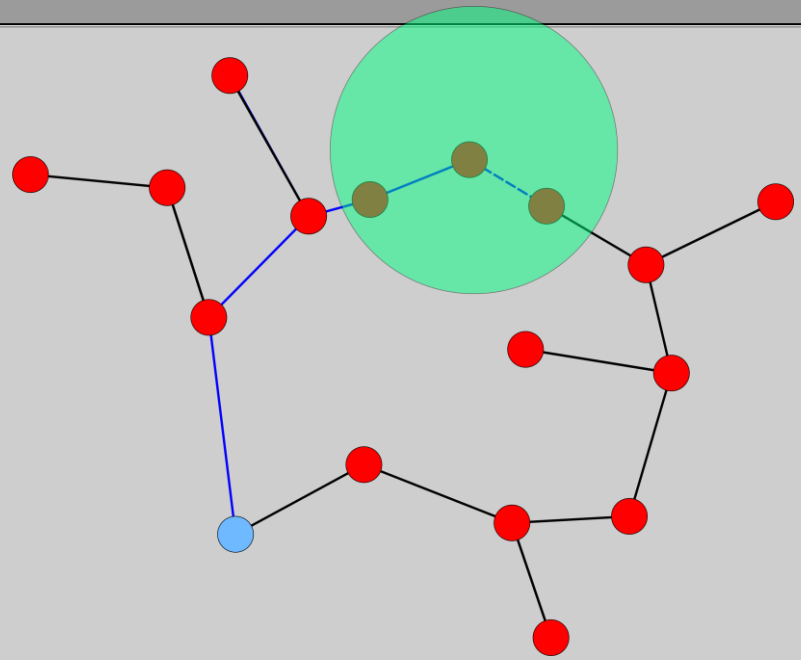


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic systems.

RRT* Algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

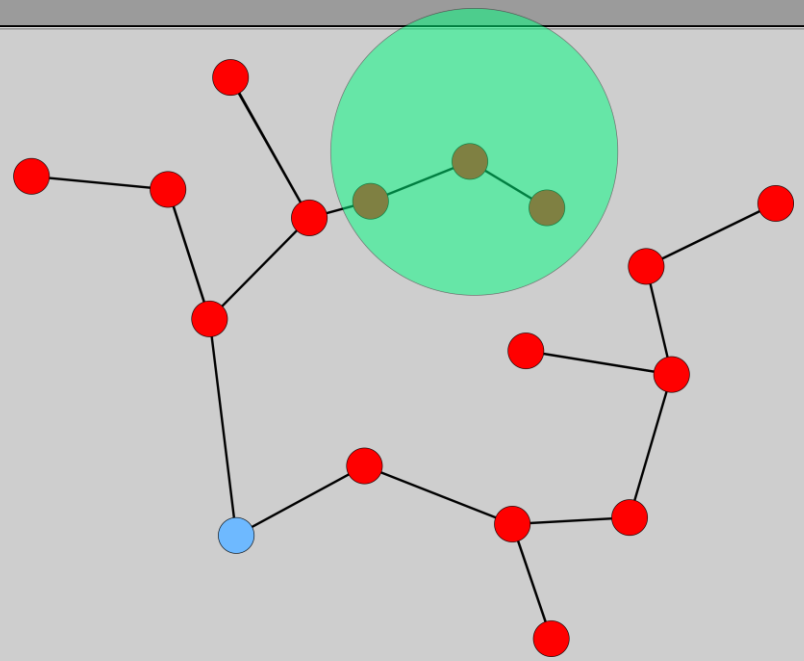


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic systems.

RRT* Algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

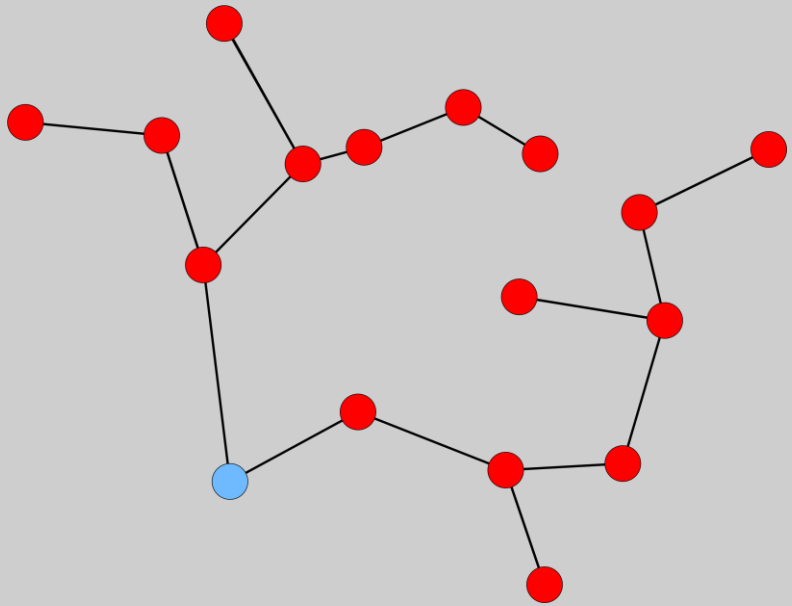


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic systems.

RRT* Algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.



Rapidly-exploring Random Tree-star (RRT*)

RRT* Algorithm

$V \leftarrow \{x_{init}\}; E \leftarrow 0;$
for i=1,…,N **do**:
$x_{rand} \leftarrow SampleFree;$
$x_{nearest} \leftarrow Nearest(G = (V, E), x_{rand});$
$x_{new} \leftarrow Steer(x_{nearest}, x_{rand});$
if $ObstacleFree(x_{nearest}, x_{new})$ **then**:
$X_{near} \leftarrow Near\left(G = (V, E), x_{new}, \min\{\gamma_{RRG}\left(\frac{\log(card\ V)}{card\ V}\right)^{1/d}, \eta\}\right);$
$V \leftarrow V \cup \{x_{new}\};$
$x_{min} \leftarrow x_{nearest};\ c_{\min} \leftarrow Cost(x_{nearest}) + c(Line(x_{nearest}, x_{new}))$
foreach $x_{near} \in X_{near}$ **do:**
if $CollisionFree(x_{near}, x_{new}) \wedge Cost(x_{near}) + c\big(Line(x_{near}, x_{new})\big) < Cost(x_{near})$ **then:**
$x_{parent} \leftarrow Parent(x_{near});$
$E \leftarrow E \backslash \{(x_{parent}, x_{near})\} \cup \{(x_{new}, x_{near})\};$
return $G = (V, E);$

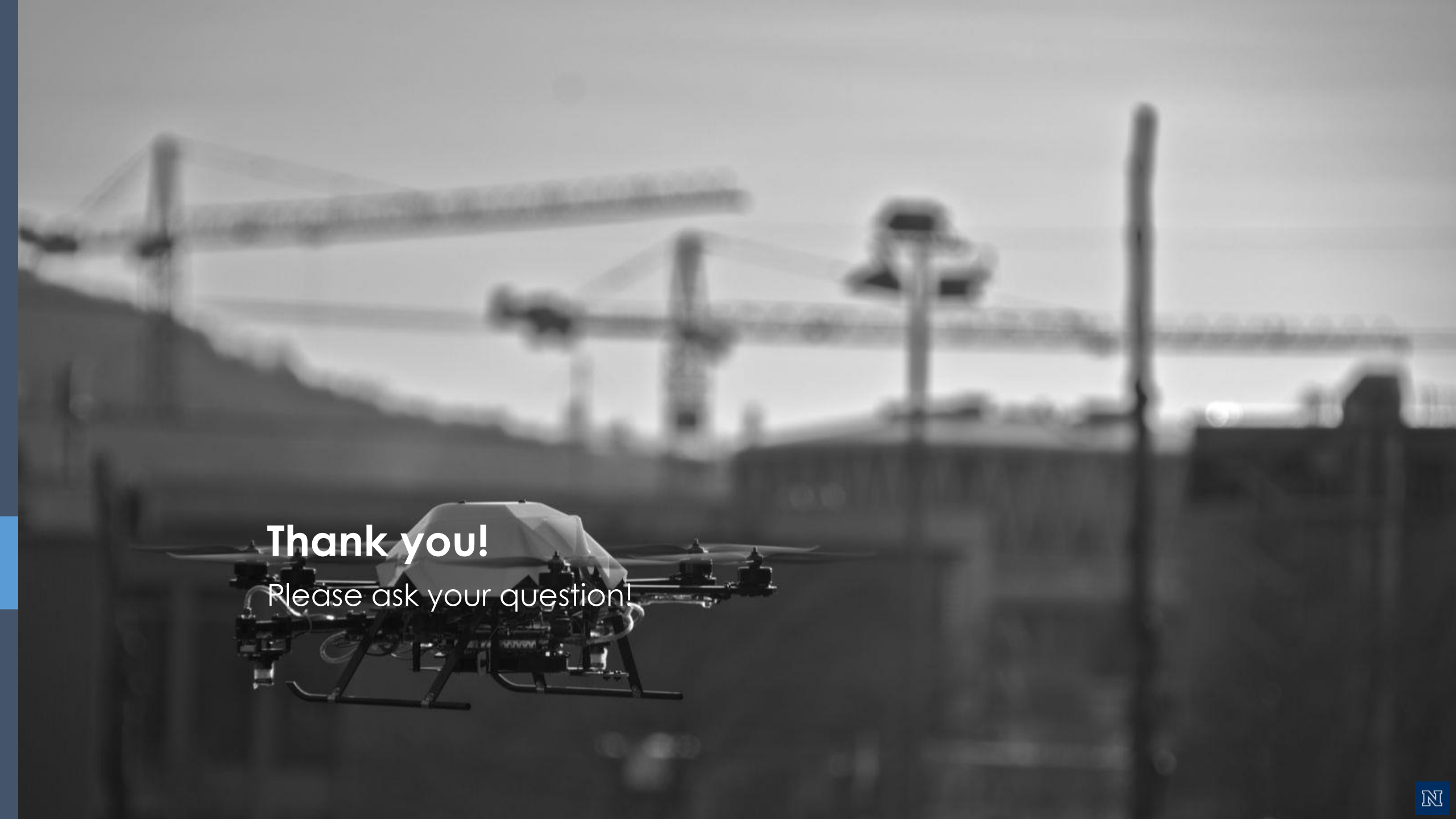
Summary

- **Key idea in RRG/RRT*:** to combine optimality and computational efficiency, it is necessary to attempt connection to $\Theta(\log N)$ nodes at each iteration.
  - Reduce volume of the "connection ball" as $\log(N)/N$;
  - Increase the number of connections as $\log N$
- These principles can be used to obtain "optimal" versions of PRM etc.

Algorithm	Probabilistic Completeness	Asymptotic Optimality	Computational Complexity
sPRM	YES	YES	$O(N)$
k-nearest PRM	NO	NO	$O(logN)$
RRT	YES	NO	$O(logN)$
PRM*	YES	YES	$O(logN)$
k-nearest PRM*	YES	YES	$O(logN)$
RRG	YES	YES	$O(logN)$
k-nearest RRG	YES	YES	$O(logN)$
RRT*	YES	YES	$O(logN)$
k-nearest RRG	YES	YES	$O(logN)$

MP: Indicative questions

- What is the difference between RRT and RRT* ?
- Considering a robot that starts from initial configuration x_0 and wants to arrive to a final configuration x_f , while navigating in the obstacle-free world, provide a visualization of the steps conducted by RRT to achieve this motion planning problem.
- Considering a robot that starts from initial configuration x_0 and wants to arrive to a final configuration x_f , while navigating in the obstacle-free world, provide a visualization of the steps conducted by RRT* to achieve this motion planning problem.
- Explain the rewiring process of RRT*



Thank you!

Please ask your question!