15-780 – Robotics

J. Zico Kolter

April 14, 2014

Outline

Robot kinematics

Motion planning

Robot dynamics

Control

Outline

Robot kinematics

Motion planning

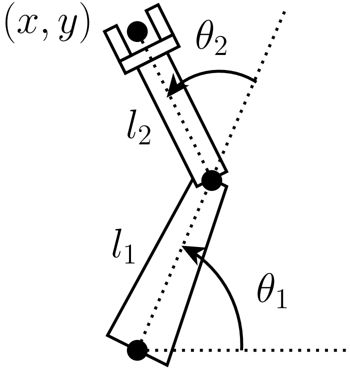
Robot dynamics

Control

Kinematics

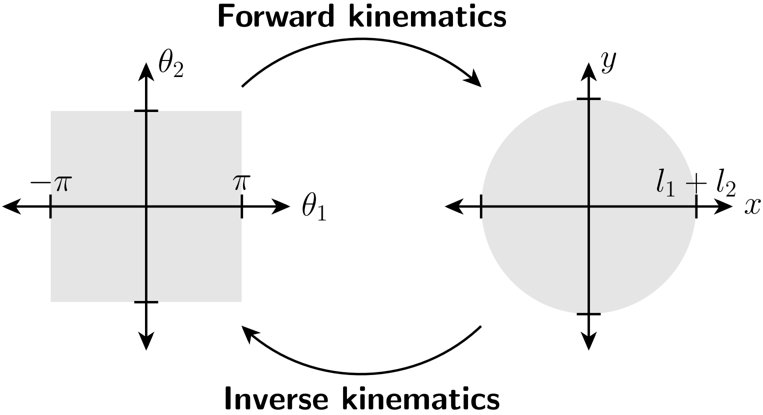
- Kinematics refers generally to the study of robot geometry
- Given a configuration of a robot (e.g., settings to joint angles), how does this affect the position of its parts?
- For a desired position of the robot end-effector, are there joint angles that achieve this position?

Two-link planar robot

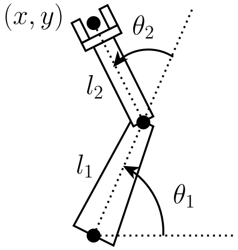


- θ_1, θ_2: joint angles of robot (configuration space, joint space)
- l_1, l_2: length of each link (robot parameters)
- x, y: position of end effector (task space)
- Kinematics is how we move back and forth between these representations

Kinematics of two-link robot



Forward kinematics of two-link robot



- Position of "elbow" x_0, y_0

$$x_0 = \ell_1 \cos(\theta_1)$$
$$y_0 = \ell_1 \sin(\theta_1)$$

- So, position of end effector x, y

$$x = \ell_1 \cos(\theta_1) + \ell_2 \cos(\theta_1 + \theta_2)$$
$$y = \ell_1 \sin(\theta_1) + \ell_2 \sin(\theta_1 + \theta_2)$$

- For simplicity, we'll write this as

$$x = \ell_1 c_1 + \ell_2 c_{12}$$
$$y = \ell_1 s_1 + \ell_2 s_{12}$$

Inverse kinematics of two-link robot

- Given x, y, can we find θ_1, θ_2 that achieve this position?

- This seems harder, there could be

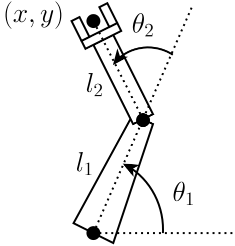
  - Infinite solutions ($x = 0, y = 0$)

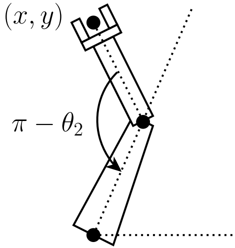
  - Two solutions ($\sqrt{x^2 + y^2} < \ell_1 + \ell_2$)

  - One solution ($\sqrt{x^2 + y^2} = \ell_1 + \ell_2$)

  - No solutions ($\sqrt{x^2 + y^2} > \ell_1 + \ell_2$)

- (Sometimes) can solve via inverse trigonometry functions

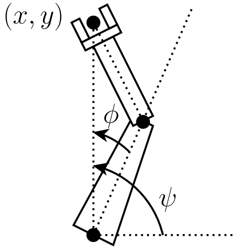

(x, y)
θ_2
l_2
l_1
θ_1



- From cosine rule

$$x^2 + y^2 = \ell_1^2 + \ell_2^2 - 2l_1 l_2 \cos(\pi - \theta_2)$$

$$\implies \theta_2 = \pm\cos^{-1}\left(\frac{x^2 + y^2 - \ell_1^2 - \ell_2^2}{2l_1 l_2}\right)$$



- From cosine rule

$$x^2 + y^2 = \ell_1^2 + \ell_2^2 - 2l_1 l_2 \cos(\pi - \theta_2)$$
$$\implies \theta_2 = \pm \cos^{-1}\left(\frac{x^2 + y^2 - \ell_1^2 - \ell_2^2}{2l_1 l_2}\right)$$

- Now solve for θ_1

$$\tan\psi = y/x$$
$$\sin\phi = \frac{\ell_2 \sin(\theta_2)}{x^2 + y^2}$$
$$\implies \theta_1 = \psi - \phi$$
$$= \tan^{-1}\left(\frac{y}{x}\right) - \sin^{-1}\left(\frac{\ell_2 \sin(\theta_2)}{x^2 + y^2}\right)$$

$$\theta_2 = \pm \cos^{-1}\left(\frac{x^2 + y^2 - \ell_1^2 - \ell_2^2}{2l_1 l_2}\right)$$

$$\theta_1 = \tan^{-1}\left(\frac{y}{x}\right) - \sin^{-1}\left(\frac{\ell_2 \sin(\theta_2)}{x^2 + y^2}\right)$$

- What happens when $\sqrt{x^2 + y^2} > \ell_1 + \ell_2$?

- For general manipulators (more on this shortly), we may not be able to find a closed form solution.

Inverse kinematics as optimization

- Define forward kinematics as the function

$$x = f(\theta), \quad x, \theta \in \mathbb{R}^n$$

- Inverse kinematics can be solved via the (non-convex) optimization problem

$$\operatorname*{minimize}_{\theta} \|f(\theta) - x^\star\|_2^2$$

  - Solve via gradient descent, other methods

  - For overdetermined systems (θ higher dimensional than x), can impose other penalties like smoothness

Jacobian

- *Jacobian* matrix contains derivatives of robot end effector with respect to joint angles

$$\left[\begin{array}{c} x \\ y \end{array}\right] = \left[\begin{array}{c} l_1 c_1 + l_2 c_{12} \\ l_1 s_1 + l_2 s_{12} \end{array}\right]$$

so

$$\begin{aligned} J &= \left[\begin{array}{cc} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} \end{array}\right] \\ &= \left[\begin{array}{cc} -\ell_1 s_1 - \ell_s s_{12} & -\ell_2 s_{12} \\ \ell_1 c_2 + \ell_2 c_{12} & \ell_2 c_{12} \end{array}\right] \end{aligned}$$

- Jacobian also provides (instantaneous) relationship between joint velocities and velocities of end effector

- Let $\theta_1(t), \theta_2(t)$ be time-varying angles

- Then by chain rule

$$\frac{\partial x(t)}{\partial t} = \frac{\partial x(t)}{\partial \theta_1(t)} \frac{\partial \theta_1(t)}{\partial t} + \frac{\partial x(t)}{\partial \theta_2(t)} \frac{\partial \theta_2(t)}{\partial t}$$

i.e.

$$\left[\begin{array}{c} \frac{\partial x(t)}{\partial t} \\ \frac{\partial y(t)}{\partial t} \end{array} \right] = J \left[\begin{array}{c} \frac{\partial \theta_1(t)}{\partial t} \\ \frac{\partial \theta_2(t)}{\partial t} \end{array} \right]$$

General manipulators

- Two-link planar robot is not that useful in practice

- To manipulate objects in 3D space, we typically want full control over 3D position *and* 3D orientation of end effector $\Longrightarrow$ at least 6 joint angles

- Forward kinematics still easy to solve (just be careful with representing 3D rotations)

- Inverse kinematics often solvable too, but much more complicated

Outline

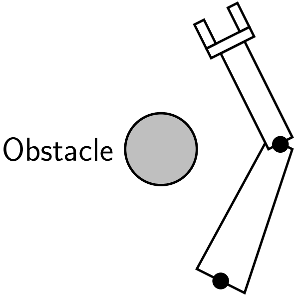
Robot kinematics

Motion planning

Robot dynamics

Control

Obstacles

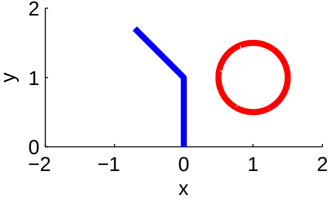


Obstacles in configuration space

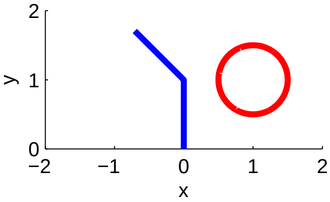
- Obstacles usually "naturally" described in the task space of the robot, but inverse kinematics often makes it less convinient to plan in task space

- Instead, want to determine which poses in the robot's *configuration space* (joint space) are non-colliding

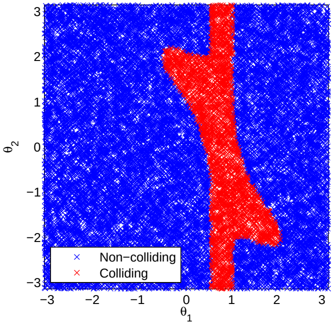
- Set of all non-colliding configurations is also called *free space*

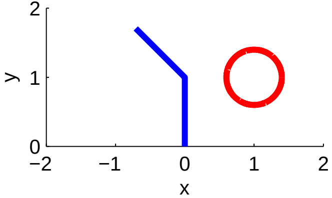


Obstacle with $r = 0.5$ at (-1,1)

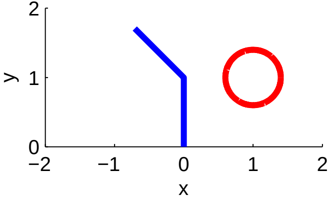


Obstacle with $r = 0.5$ at (-1,1)

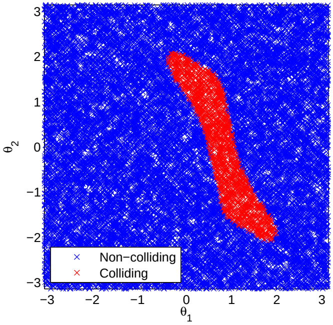




Obstacle with $r = 0.4$ at (-1,1)



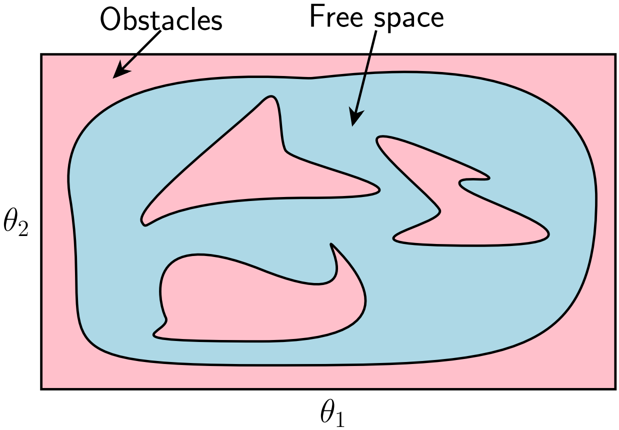
Obstacle with $r = 0.4$ at (-1,1)



Sample-based planning

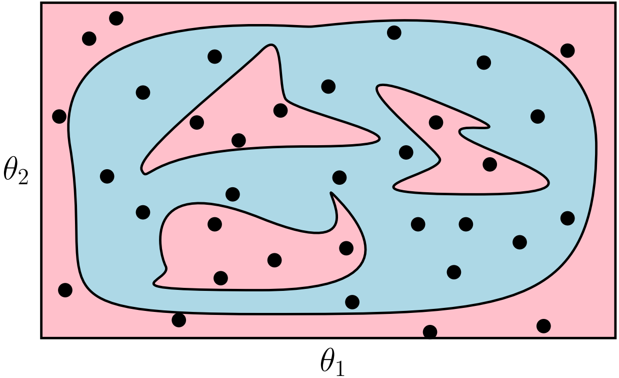
- In general, it's very difficult to analytically desribe the free space
- But we *can* (relatively) quickly check to see if a given configuration is colliding or not
- Motivated a class of algorithms that somehow sample points in configuration space, form paths over non-colliding samples

Probabilistic road maps (PRMs)



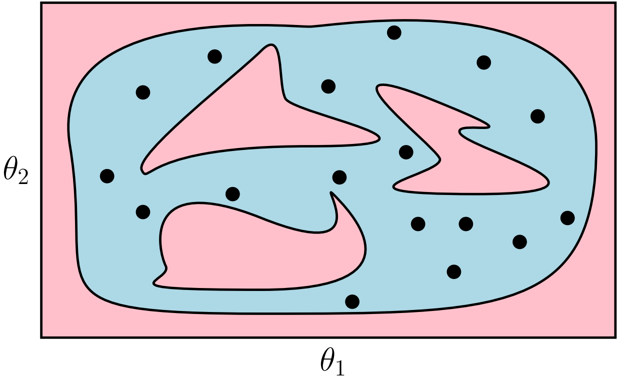
Plot of configuration space of robot

Probabilistic road maps (PRMs)



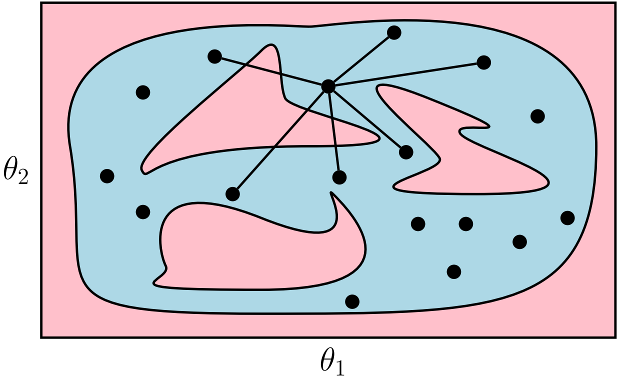
Randomly sample points in configuration space

Probabilistic road maps (PRMs)



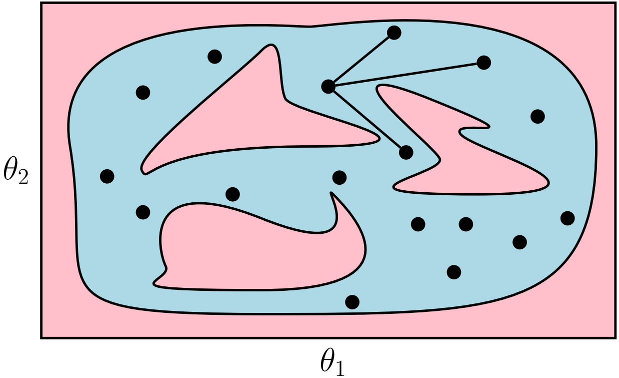
Throw out all points not in free space

Probabilistic road maps (PRMs)



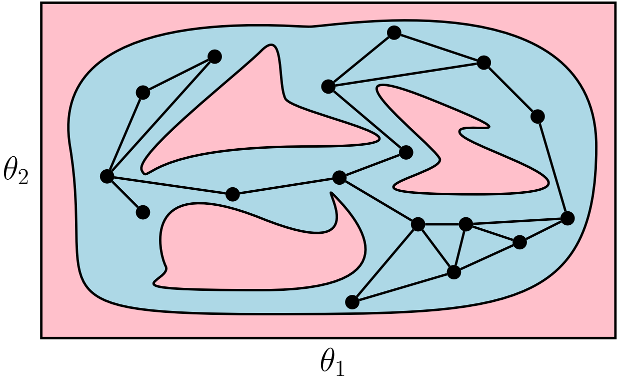
Connect each remaining point to its nearest neighbors

Probabilistic road maps (PRMs)



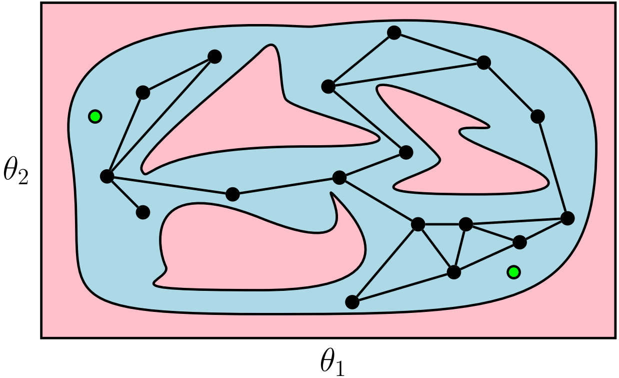
Remove all colliding paths

Probabilistic road maps (PRMs)



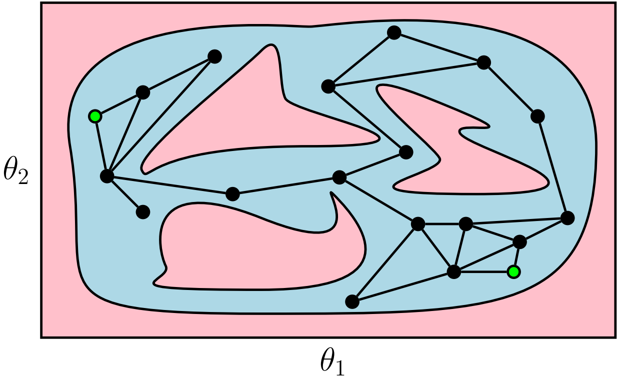
Do this for all the nodes to form a graph

Probabilistic road maps (PRMs)



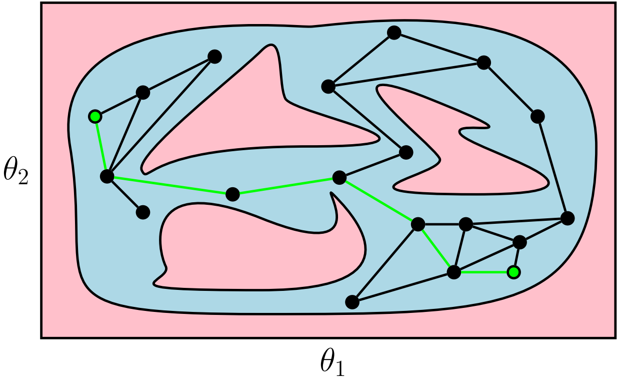
Now, given new start and end points

Probabilistic road maps (PRMs)



Add points to the graph

Probabilistic road maps (PRMs)



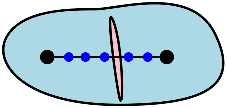
Plan motion using any graph search method

Challenges in PRMs

- How do we know if a path is non-colliding? (remember, we can only easily check if individual points in configuration space are non-colliding)
  - Check many points uniformly on line

Challenges in PRMs

- How do we know if a path is non-colliding? (remember, we can only easily check if individual points in configuration space are non-colliding)
  - Check many points uniformly on line
  - Looks good!



  - Need to ensure the discretization is smaller than narrowest obstacle (e.g. by adding "safety margin" to obstacles)

- Existence of “bottlenecks”
  - Sample more densely in areas that have narrow passages
- Random sampling in $[0, 1]^n$?
- Complexity of constructing graph?
- What about systems with dynamics, can’t move arbitrarily between points in configuration space (more on this next time)

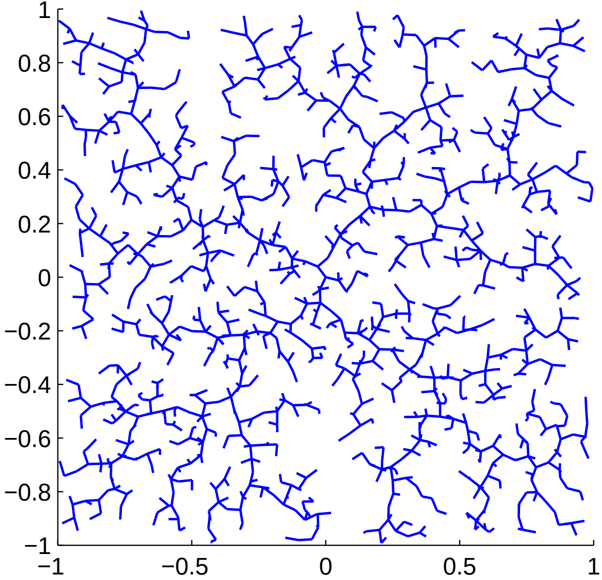
Rapidly-exploring random trees (RRTs)

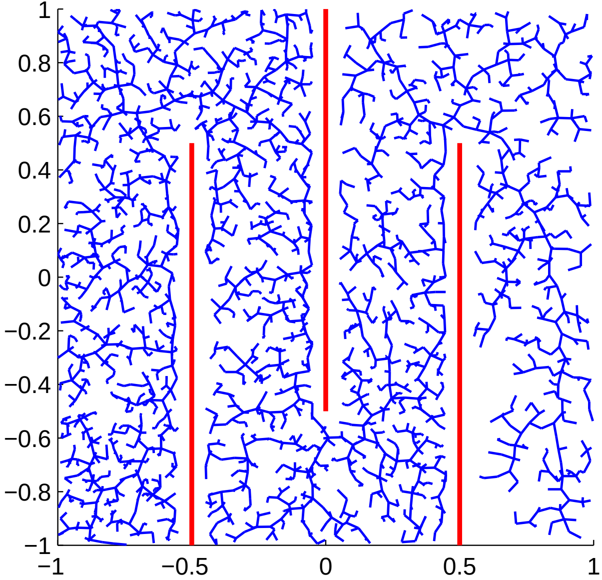
- (LaVallle, 1998)
- A method for generating sample points and graph (here just a tree) for a PRM
- Scales to higher dimensions better than random uniform sampling (but careful, still exponential complexity in dimension)
- Can incoporate dynamics (not discussed today)

function $T =$ Build_RRT$(x_{\text{init}}, \epsilon)$
T.add_vertex(x_{init})
For $i = 1, \ldots, m$
$x_{\text{rand}} \leftarrow$ Random_State()
$x_{\text{near}} \leftarrow$ Nearest_Neighbor(T, x_{rand})
$x_{\text{new}} \leftarrow$ Grow_Towards$(x_{\text{near}}, x_{\text{rand}}, \epsilon)$
T.add_vertex(x_{new})
T.add_edge$(x_{\text{near}}, x_{\text{new}})$

- Can account for obstacles by just not adding point of x_{new} colliding (as long as ϵ small enough)

- *Many* variants: forward-backward, dynamic versions, RRT*


1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
−1
−0.5
0
0.5
1


1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
−1
−0.5
0
0.5
1

Outline

Robot kinematics

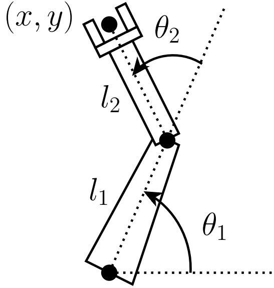
Motion planning

Robot dynamics

Control

Robot Dynamics

- Need to consider how robot's state evolves over time, and how physical laws effect this evolution



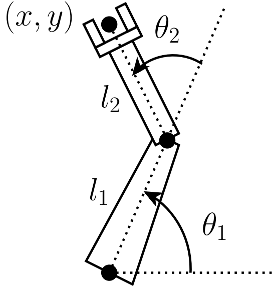
Kinematic system

State: θ_1, θ_2

Parameters: ℓ_1, ℓ_2

Robot Dynamics

- Need to consider how robot's state evolves over time, and how physical laws effect this evolution



Kinematic system

State: θ_1, θ_2

Parameters: ℓ_1, ℓ_2

Dynamic system

State: $\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2$

Parameters: ℓ_1, ℓ_2, m_1, m_2 (point masses at elbow, wrist)

- Given a current state $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2)$ we want to find a function that shows how the system evolves

- I.e., we want to find

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} = f(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2)$$

  called the *equations of motion* of the system

- To derive the equations of motion for this system, we'll use a generalization of Newton's laws $F = ma$

- We'll write a general form of this law (called the Euler-Lagrange equations) as

$$F_i = \frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}_i} - \frac{\partial L}{\partial \theta_i}$$

  where

  - $L = T - U$ is called the *Lagrangian* of the system, where T is equal to the kinetic energy and U equal to the potential energy

  - F_i is generalized force applied to ith coordinate of system (in our case, these would be torques applied to the joints, which we'll denote as τ_i)

- Consider applying these laws to a simple particle with coordinate x (experiencing no gravity) and mass m

- Then
$$T = \frac{1}{2}m\dot{x}^2, \;\; U = 0$$
so
$$F = \frac{d}{dt}\frac{\partial}{\partial \dot{x}}\frac{1}{2}m\dot{x}^2 - \frac{\partial}{\partial x}\frac{1}{2}\dot{x}^2 = \frac{d}{dt}m\dot{x} = m\ddot{x}$$

- If particle were being acted upon by gravity, then we would have $U = mgh$ where h is the height of the particle.

- Let's go through the process for the two-link robot (here x_1, y_1 will denote location of elbow, x_2, y_2 location of end effector)

- First, by forward kinematics, we have

$$\begin{aligned} x_1 = \ell_1 c_1 \quad &\Longrightarrow \quad \dot{x}_1 = -\ell_1 s_1 \dot{\theta}_1 \\ y_1 = \ell_1 s_1 \quad &\Longrightarrow \quad \dot{y}_1 = \ell_1 c_1 \dot{\theta}_1 \\ x_2 = x_1 + \ell_2 c_{12} \quad &\Longrightarrow \quad \dot{x}_2 = \dot{x}_1 - \ell_2 s_{12} \dot{\theta}_{12} \\ y_2 = y_1 + \ell_2 s_{12} \quad &\Longrightarrow \quad \dot{y}_2 = \dot{y}_1 + \ell_2 c_{12} \dot{\theta}_{12} \end{aligned}$$

- Then (after some algebra, and trigonometric identities)

$$\begin{aligned} T &= \frac{1}{2} m_1 (\dot{x}_1^2 + \dot{y}_1^2) + \frac{1}{2} m_2 (\dot{x}_2^2 + \dot{y}_2^2) \\ &= \frac{1}{2} (m_1 + m_2) \ell_1^2 \dot{\theta}_1^2 + \frac{1}{2} m_2 \ell_2^2 (\dot{\theta}_1 + \dot{\theta}_2)^2 + m_2 c_2 \ell_1 \ell_2 \dot{\theta}_1 (\dot{\theta}_1 + \dot{\theta}_2) \\ U &= m_1 g y_1 + m_2 g y_2 = (m_1 + m_2) g \ell_1 s_1 + m_2 g \ell_2 s_{12} \end{aligned}$$

- Taking derivatives and simplifying

$$
\begin{aligned}
\tau_1 &= \frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}_1} - \frac{\partial L}{\partial \theta_1} \\
&= (m_1 + m_2)\ell_1^2\ddot{\theta}_1 + m_2\ell_2^2(\ddot{\theta}_1 + \ddot{\theta}_2) + m_2 c_2 \ell_1 \ell_2 (2\ddot{\theta}_1 + \ddot{\theta}_2) \\
&\quad - m_2 s_2 \ell_1 \ell_2 (2\dot{\theta}_1 + \dot{\theta}_2)\dot{\theta}_2 + (m_1 + m_2) g \ell_1 c_1 + m_2 g \ell_2 c_{12} \\
\tau_2 &= \frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}_2} - \frac{\partial L}{\partial \theta_2} \\
&= m_2 \ell_2^2 (\ddot{\theta}_1 + \ddot{\theta}_2) + m_2 c_2 \ell_1 \ell_2 \ddot{\theta}_1 + m_2 s_2 \ell_1 \ell_2 \dot{\theta}_1^2 + m_2 g \ell_2 c_{12}
\end{aligned}
$$

- But, we still want a direct solution of $\ddot{\theta}_1$, $\ddot{\theta}_2$

- Putting the equations above in matrix forms

$$H(\theta)\begin{bmatrix}\ddot{\theta}_1\\ \ddot{\theta}_2\end{bmatrix}+C(\theta,\dot{\theta})+G(\theta)=\begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix}$$

where

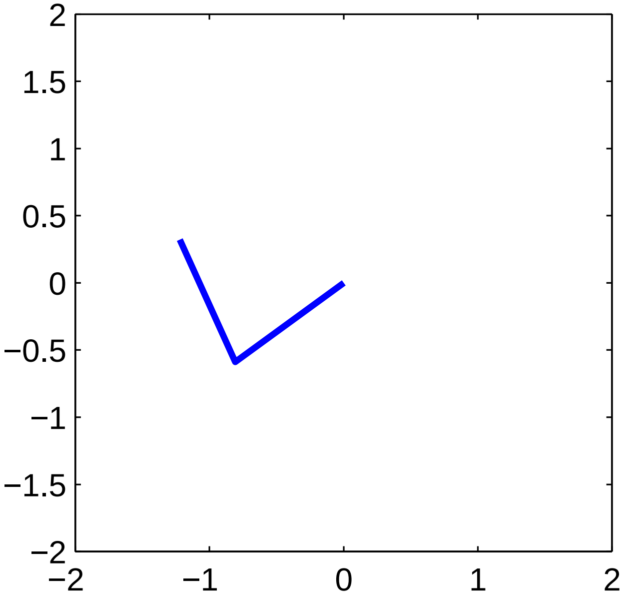
$$H(\theta)=\begin{bmatrix}(m_1+m_2)\ell_1^2+m_2\ell_2^2+2m_2c_2\ell_1\ell_2 & m_2\ell_2^2+m_2c_2\ell_1\ell_2\\ m_2\ell_2^2+m_2c_2\ell_1\ell_2 & m_2\ell_2^2\end{bmatrix}$$

$$C(\theta,\dot{\theta})=\begin{bmatrix}-m_2s_2\ell_1\ell_2(2\dot{\theta}_1+\dot{\theta}_2)\dot{\theta}_2\\ m_2s_2\ell_1\ell_2\dot{\theta}_1^2\end{bmatrix}$$

$$G(\theta)=\begin{bmatrix}(m_1+m_2)g\ell_1c_1+m_2g\ell_2c_{12}\\ m_2g\ell_2c_{12}\end{bmatrix}$$

- So, after all this, we finally have

$$\ddot{\theta}=H(\theta)^{-1}(\tau-C(\theta,\dot{\theta})-G(\theta))$$


2
1.5
1
0.5
0
−0.5
−1
−1.5
−2
−2
−1
0
1
2

Outline

Robot kinematics

Motion planning

Robot dynamics

Control

- How do we make robot behave as we want (e.g. reach a certain point, follow a certain trajectory) under the constraints of its dynamics?

PD Control

- For now, let's assume that each of the robot's joints has a motor that can apply torque (be careful, things change a lot when this is no longer the case)

- Suppose we want to bring robot to a desired state $\theta^\star$

- We could try to look into the detailed dynamics model, produce a sequence of torques, but this seems uncessarily complex

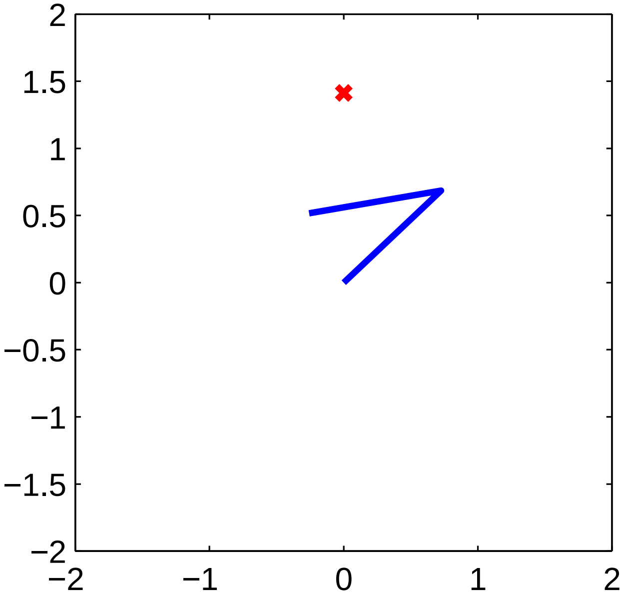
- **Proportional (P) control**: instead of trying to use our dynamics model, let's just use the intuitive control law

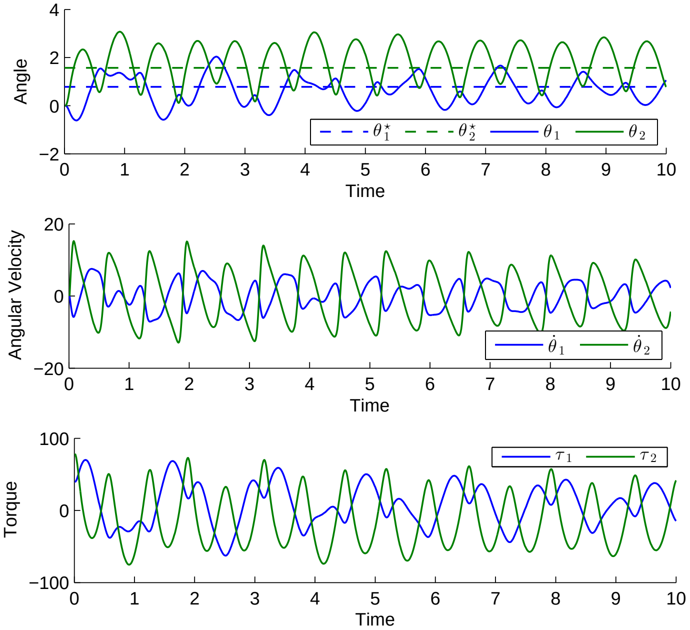
$$\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = k_P \begin{bmatrix} \theta_1 - \theta_1^\star \\ \theta_2 - \theta_2^\star \end{bmatrix}$$

  for some constant k_P

- Know as *proportional* control, it just applies a torque in relation to how far away we are from the desired location

- Let's look at the method applied to our two-link arm with $m_1 = m_2 = 1\text{kg}$, $\ell_1 = \ell_2 = 1\text{m}$, $k_P = -50$


2
1.5
1
0.5
0
−0.5
−1
−1.5
−2
−2
−1
0
1
2


4
2
0
−2
Angle
0
1
2
3
4
5
6
7
8
9
10
Time
$\theta_1^\star$
$\theta_2^\star$
θ_1
θ_2
20
0
−20
Angular Velocity
Time
$\dot{\theta}_1$
$\dot{\theta}_2$
100
0
−100
Torque
Time
τ_1
τ_2

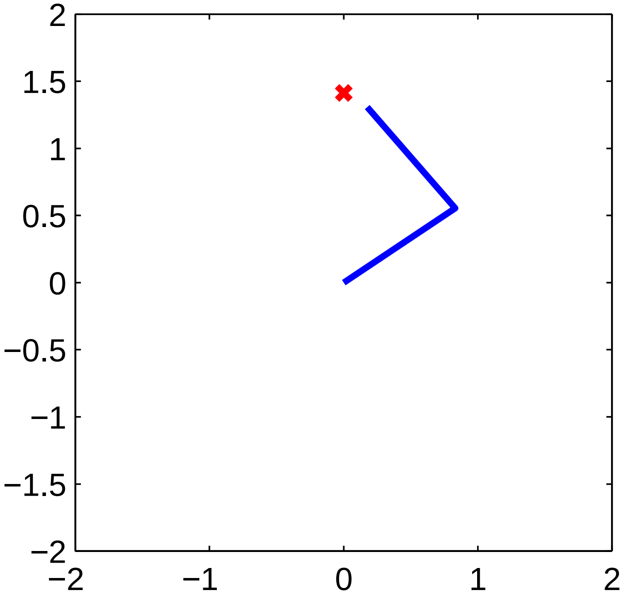
- The trouble with proportional control is that it "overshoots," by the time we reach the desired position we've already built up velocity, leads to oscilations

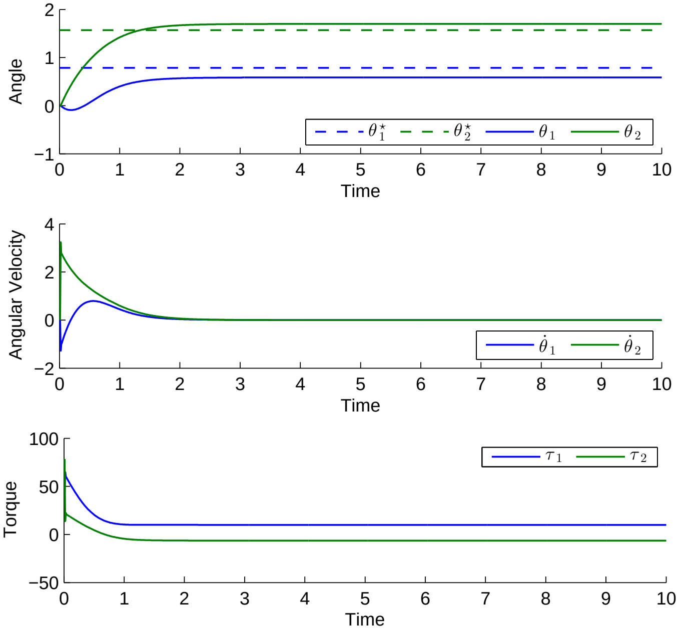
- Can overcome this by adding a term that penalizes deviation from desired velocity (in this case, $\dot{\theta}_i^\star = 0$)

- **Proportional Derivative (PD) control**

$$\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = k_P \begin{bmatrix} \theta_1 - \theta_1^\star \\ \theta_2 - \theta_2^\star \end{bmatrix} + k_D \begin{bmatrix} \dot{\theta}_1 - \dot{\theta}_1^\star \\ \dot{\theta}_2 - \dot{\theta}_2^\star \end{bmatrix}$$

- Using the same parameters as before, but $k_D = -20$


2
1.5
1
0.5
0
−0.5
−1
−1.5
−2
−2
−1
0
1
2



Angle
Time
$\theta_1^\star$
$\theta_2^\star$
θ_1
θ_2
Angular Velocity
Time
$\dot\theta_1$
$\dot\theta_2$
Torque
Time
τ_1
τ_2

- Still aren't reaching the desired location, because gravity is "fighting" the control (would work in zero gravity)

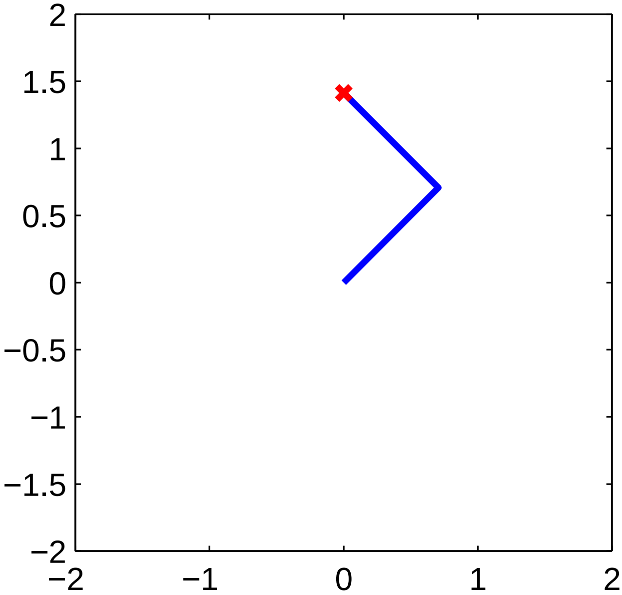
- Solution is to find "open loop" torques that would keep us at desired position

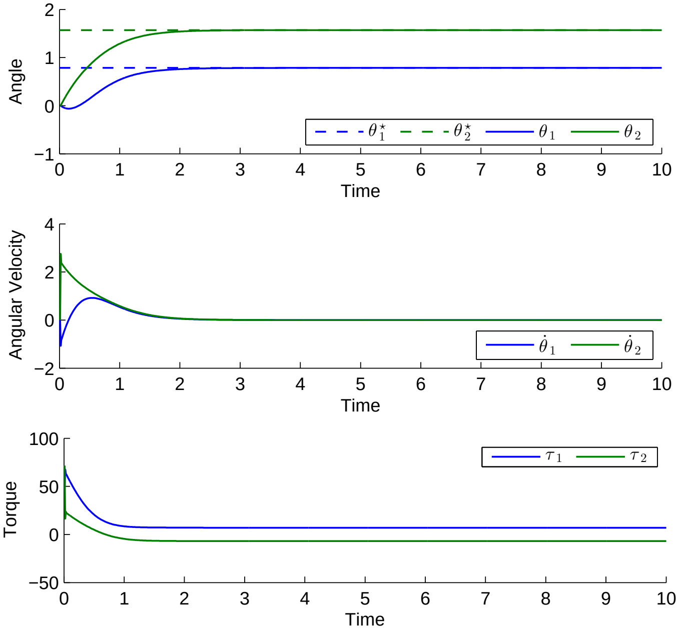
$$\tau^\star = H(\theta^\star)\ddot{\theta}^\star + C(\theta^\star, \dot{\theta}^\star) + G(\theta^\star)$$

- Since $\dot{\theta}^\star = 0$ in this case, for the two-link manipulator optimal torques are just $\tau^\star = G(\theta^\star)$ (i.e., the torques required to overcome gravity)

- **Feedforward PD control**

$$\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = \tau^\star + k_P \begin{bmatrix} \theta_1 - \theta_1^\star \\ \theta_2 - \theta_2^\star \end{bmatrix} + k_D \begin{bmatrix} \dot{\theta}_1 - \dot{\theta}_1^\star \\ \dot{\theta}_2 - \dot{\theta}_2^\star \end{bmatrix}$$


2
1.5
1
0.5
0
−0.5
−1
−1.5
−2
−2
−1
0
1
2



2
1
0
−1
Angle
$\theta_1^\star$
$\theta_2^\star$
θ_1
θ_2
0 1 2 3 4 5 6 7 8 9 10
Time
4
2
0
−2
Angular Velocity
$\dot{\theta}_1$
$\dot{\theta}_2$
0 1 2 3 4 5 6 7 8 9 10
Time
100
50
0
−50
Torque
τ_1
τ_2
0 1 2 3 4 5 6 7 8 9 10
Time

- Combining *feedforward* (open loop $\tau^\star$) and *feedback* (P and D terms) control laws lets us reach the desired position

- Can replace the feedforward term with an "integrator" that integrates the error in position, called *proporitonal integral derivative* (PID) control

$$\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = k_I \int_0^T \begin{bmatrix} \theta_1(t) - \theta_1^\star \\ \theta_2(t) - \theta_2^\star \end{bmatrix} dt + k_P \begin{bmatrix} \theta_1 - \theta_1^\star \\ \theta_2 - \theta_2^\star \end{bmatrix} + k_D \begin{bmatrix} \dot\theta_1 - \dot\theta_1^\star \\ \dot\theta_2 - \dot\theta_2^\star \end{bmatrix}$$

- But watch out, integrator term can be very finicky, especially when we talk about tracking motion

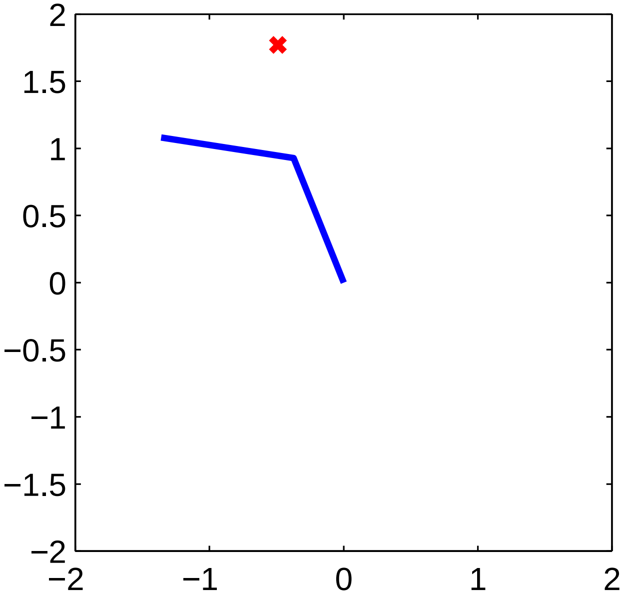
Trajectory following

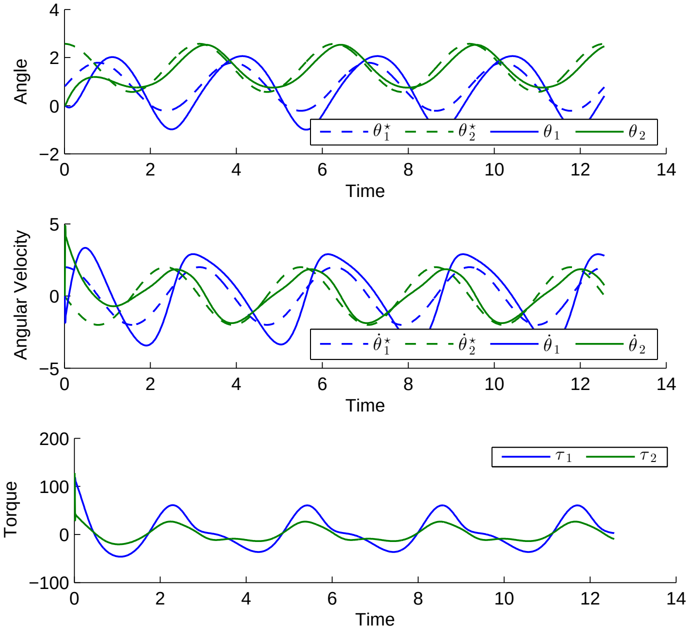
- The same concepts apply to following a desired trajectory $\theta^\star(t)$

- For instance, PD control in this case would take the form

$$\left[\begin{array}{c} \tau_1 \\ \tau_2 \end{array}\right] = k_P \left[\begin{array}{c} \theta_1 - \theta_1^\star(t) \\ \theta_2 - \theta_2^\star(t) \end{array}\right] + k_D \left[\begin{array}{c} \dot{\theta}_1 - \dot{\theta}_1^\star(t) \\ \dot{\theta}_2 - \dot{\theta}_2^\star(t) \end{array}\right]$$

- Same problem with pure PD control as before (don't reach the desired location), but this time it won't even work in zero gravity


2
1.5
1
0.5
0
−0.5
−1
−1.5
−2
−2
−1
0
1
2


Angle
Time
$\theta_1^\star$ $\theta_2^\star$ θ_1 θ_2
Angular Velocity
Time
$\dot{\theta}_1^\star$ $\dot{\theta}_2^\star$ $\dot{\theta}_1$ $\dot{\theta}_2$
Torque
Time
τ_1 τ_2

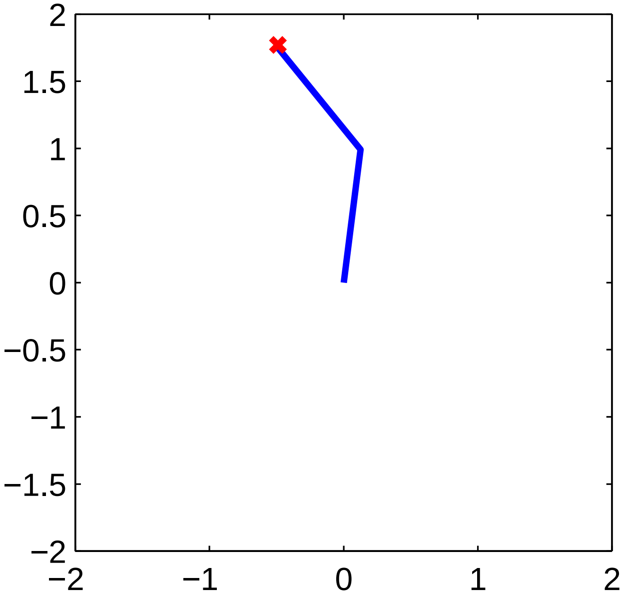
- Feedforward control works as before, but this time we'll need time-varying optimal torques, and all the terms in the dynamics

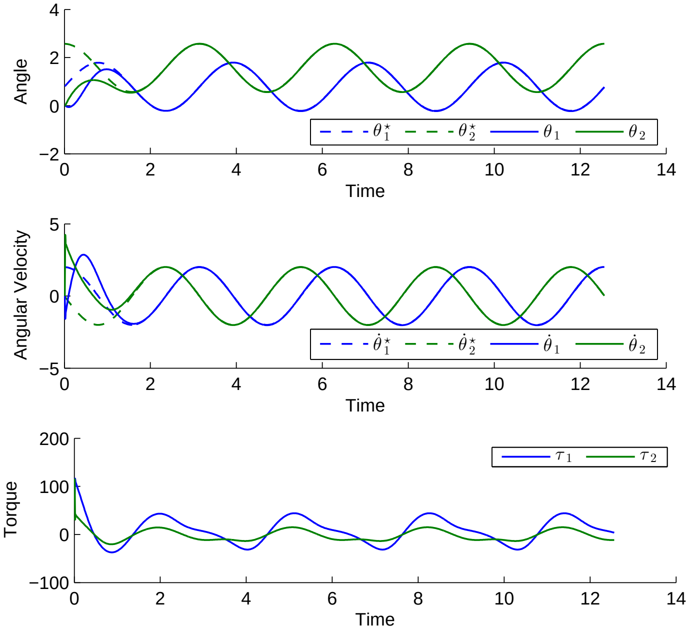
$$\tau^\star(t) = H(\theta^\star(t))\ddot{\theta}^\star(t) + C(\theta^\star(t), \dot{\theta}^\star(t)) + G(\theta^\star(t))$$

and control law

$$\left[\begin{array}{c} \tau_1 \\ \tau_2 \end{array}\right] = \tau^\star(t) + k_P \left[\begin{array}{c} \theta_1 - \theta_1^\star(t) \\ \theta_2 - \theta_2^\star(t) \end{array}\right] + k_D \left[\begin{array}{c} \dot{\theta}_1 - \dot{\theta}_1^\star(t) \\ \dot{\theta}_2 - \dot{\theta}_2^\star(t) \end{array}\right]$$

- Here, it's much trickier to get integral control to work well, since "open loop" term is time-varying


2
1.5
1
0.5
0
−0.5
−1
−1.5
−2
−2
−1
0
1
2



Angle
4
2
0
−2
0
2
4
6
8
10
12
14
Time
$\theta_1^\star$
$\theta_2^\star$
θ_1
θ_2
Angular Velocity
5
0
−5
Time
$\dot\theta_1^\star$
$\dot\theta_2^\star$
$\dot\theta_1$
$\dot\theta_2$
Torque
200
100
0
−100
Time
τ_1
τ_2

Underactuated robots

- The following examples were all "easy" in the sense that we had an actuator controlling each degree of freedom of the robot (and they could generate arbitrary torque)

- But, most robot aren't like this

  – Plane: 6DOF, 4 inputs

  – Heliopter: 6DOF, 4 inputs

  – Planar Car: 3DOF, 2 inputs

- How can we control these systems?

- First, let's accept the fact that we can no longer control the robot arbitrarily

- Example: two-link manipulator with only elbow controlled (acrobot)

$$H(\theta)\ddot{\theta} + C(\theta, \dot{\theta}) + G(\theta) = \begin{bmatrix} 0 \\ \tau_2 \end{bmatrix}$$

- Could we reach and maintain $\theta = (\pi/4, \pi/2)$?

- But, maybe we can control the system around "feasible" points

- For this to be possible at all, we'd require a point $\theta^\star$ and torque $\tau_2^\star$ such that
$$\begin{bmatrix} 0 \\ \tau_2 \end{bmatrix} = G(\theta^\star)$$
known as an equilibrium point of the system

- E.g., for $\theta^\star = (\pi/2, 0)$ (robot fully upright), $G(\theta^\star) = 0$, so an equilibrium point for $\tau = 0$

- But, how do we design a "PD-like" control law that can maintain this point? We need something like
$$\tau_2 = f(\theta_1 - \theta_1^\star, \theta_2 - \theta_2^\star, \dot{\theta}_1 - \dot{\theta}_1^\star, \dot{\theta}_2 - \dot{\theta}_2^\star)$$
but what is this function f?

Linear quadratic regulator (LQR)

- Given a (linear) system with dynamics

$$\dot{x} = Ax + Bu$$

where x denotes the state and u denotes the control inputs

- Want to find a feedback controller $u(t) = Kx(t)$ that forces state to $x = 0$ and maintains it there

- Cost of system is measured by

$$J = \int_0^\infty \left(x(t)^T Q x(t) + u(t)^T R u(t) \right) dt$$

for some (positive definite) matrices Q and R

- Somewhat surprisingly, it turns out we can solve this problem exactly, optimal K is given by

$$K = -R^{-1}B^T P$$

where P is the solution to the equation

$$A^T P + PA - PBR^{-1}B^T P = 0$$

- A non-linear set of equations, but there exist methods that will find this very efficiently (i.e., in MATLAB the command `lqr`, in Python there are a couple libraries that will do it)

Back to the Acrobot

- But the acrobot is a *non-linear* system: letting $x = (\theta_1, \theta_2, \dot\theta_1, \dot\theta_2)$, $u = \tau_2$, our dynamics can be written as

$$\dot{x} = f(x, u)$$

- A remarkable property of non-linear control: assume $x^\star, u^\star$ is an equilbrium point of f (i.e.,) $f(x^\star, u^\star) = 0$, and we have controller $u(t) = Kx(t)$ that stabilizes the linear approximation to this system

$$\dot{x} - \dot{x}^\star \approx A(x - x^\star) + B(u - u^\star)$$

  where

$$A \equiv \frac{\partial f(x^\star, u^\star)}{\partial x}, \quad B \equiv \frac{\partial f(x^\star, u^\star)}{\partial u}$$

- Then this controller also stabilizes the non-linear system in some region around $x^\star, u^\star$

Putting it all together

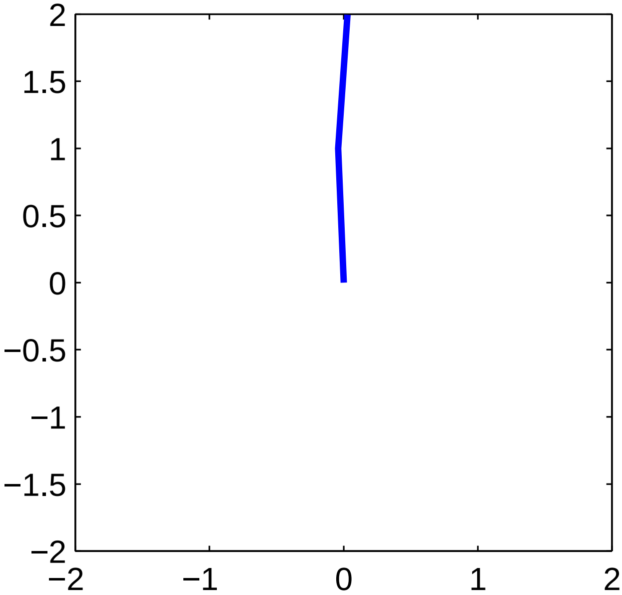
- So, the whole process is as follows:

  1. Find an equilibrium point of the system

  2. Compute linearization A and B around this equilibrium point

  3. Compute LQR controller for A and B (using some, probably hand-specified cost matrices Q and R)

  4. Execute the resulting controller on the non-linear system


2
1.5
1
0.5
0
−0.5
−1
−1.5
−2
−2
−1
0
1
2

LQR around trajectories

- All the considerations above also apply to tracking a (feasible) trajectory $\theta^\star(t)$ in an underactuated system
- As before, a few additional complications, need to compute time-varying LQR controllers
- Often, the most challenging piece is simply coming up with the feasible trajectory in the first place
  - Here's where we can use planning techniques like RRTs (extended to dynamical systems), optimization methods (shooting, direct collocation), etc