Lecture 1, Sep 7, 2023

Lecture 2, Sep 12, 2023

Lecture 3, Sep 14, 2023

Taxonomy of Robotics

- A common paradigm is that of "sense, plan, act" which we will use in this course
  - Sometimes these stages are combined with no clear boundaries, such as neural networks or learning-based paradigms
- The typical chain of processing is perception → information processing → mapping → localization → planning → navigation → action
  - Perception is done through basic sensors such as odometry, gyroscopes, IMUs, etc and rich sensors such as cameras, lidars, tactile sensors, etc
  - Information processing involves reducing noise, antialiasing, and fusion of sensor measurements
  - Mapping involves locating features and landmarks and creating a map of the environment
  - Localization uses the map to determine where we are on the map with techniques such as Kalman filtering, particle filtering (Monte Carlo) or Bayesian localization
    * Localization and mapping can be combined into SLAM (simultaneous localization and mapping), which solves the chicken-and-egg problem
  - Planning involves pathfinding, with techniques such as Voronoi diagrams, cell decomposition and potential fields
  - Navigation involves following the planned path; it can be methodical and map-based or behaviour-based which is reactive (e.g. behaviour trees)
  - Action directly controls actuators, e.g. PID control
- Levels of autonomy:
  - Assistant: fully supervised and teleoperated
    * Unilateral teleoperation involves no feedback, bilateral has position feedback only and multilateral has force feedback as well
    * Teleoperation faces issues with interfacing, time delays and flexibility
  - Apprentice: able to execute low-level tasks unsupervised
  - Associate: able to execute elements of tasks autonomously (but cannot break down large tasks)
  - Agent: fully autonomous

Lecture 4, Sep 19, 2023

Motion of Robots

- We will concentrate primarily on rolling, i.e. robots with wheels
- Steering models:
  - Bicycle steering: traction wheel in rear, steering wheel in front
    * Tricycle steering: the rear wheel is unpowered, while the front steering wheel is powered
  - Differential steering: two independently controlled wheels, vary speed to get desired curvature
    * Tripod differential steering: differential steering with an unpowered omnidirectional wheel in front for support
    * Skid-steering: each side is controlled together
  - Directed differential steering: differential steering aided by a steering wheel
  - Ackermann steering: two differentially operated wheels at the back and two connected steering wheels at the front
- Wheels can also be compound:
  - Mecanum wheels: wheels with angled rollers on the surface
    * Through moving the wheels in different directions, motion in any of the 4 directions or rotation

  can be achieved
  * The wheels produce forces in diagonal directions; combining these forces results in a net force in the desired direction
- Omni wheels: like the Mecanum wheel, but the rollers are perpendicular instead of diagonal

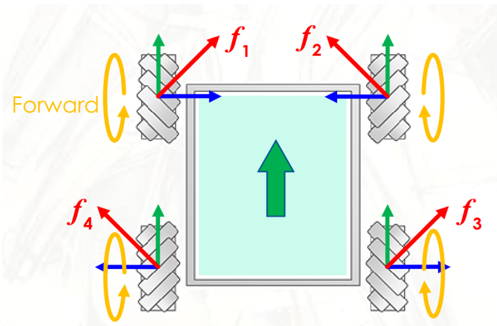

Figure 1: Mecanum wheels in translation.

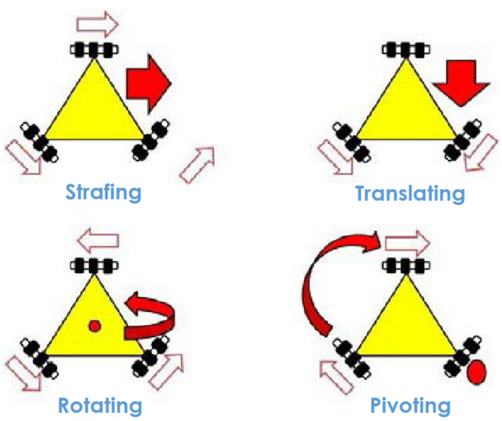

Figure 2: Omni wheels used on a vehicle.

- Holonomic constraint: a vehicle is holonomic if the vehicle's geometry does not constrain its motion (i.e. it can move in any direction, regardless of which direction it's facing)
  - Mecanum and omni drive are holonomic, but Ackermann is not
  - We will formally define this in AER301

Kinematical Models of Motion

- The *pose* of a robot is its position and orientation
  - For now we will work in 2D, with position being x, y and orientation being θ, so the pose is $\boldsymbol{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$
- For a simple unicycle model, θ is the angle of the wheel and x, y are the position of the wheel on the ground
  - $\dot{x} = v\cos\theta, \dot{y} = v\sin\theta, \dot{\theta} = \omega$
  - $\boldsymbol{x} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix}$

- $\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \implies \dot{\boldsymbol{x}} = \boldsymbol{B}\boldsymbol{u}$
- $\boldsymbol{x}$ is the state and $\boldsymbol{u}$ is the system input

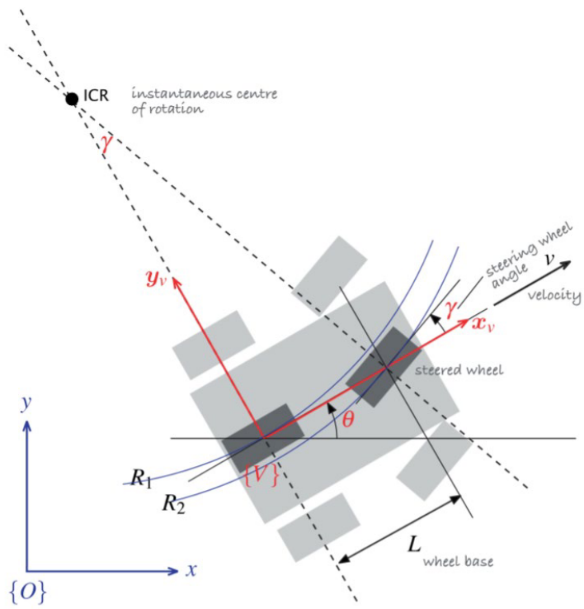

Figure 3: Bicycle model derivation.

- For a bicycle model, refer to the diagram above
  - We will fix our reference point to the rear wheel; θ is the angle the rear wheel makes with the horizontal axis
  - $\dot{x} = v\cos\theta, \dot{y} = v\sin\theta$ as usual
  - To find $\dot{\theta}$, we extend a perpendicular line from the wheels to intersect at the instantaneous center of rotation
  - $v = R_1\dot{\theta}, \frac{l}{R_1} = \tan\gamma \implies \dot{\theta} = \frac{v}{l}\tan\gamma$
  - The control inputs are v and γ
  - Notice that this model is now nonlinear due to the tangent on γ and multiplication by v

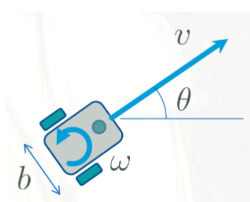

Figure 4: Differential steering derivation.

- For differential steering our control inputs are $\dot{\gamma}_r, \dot{\gamma}_l$, which are the rotational rates of the two wheels
  - $v = \frac{r(\dot{\varphi}_r + \dot{\varphi}_l)}{2}$ (velocity is simply the average)
  - $\omega = \frac{r(\dot{\varphi}_r - \dot{\varphi}_l)}{b}$
  - We can then put this into the unicycle model to obtain the final model

- $\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} r\cos\theta & \frac{1}{2} r\cos\theta \\ \frac{1}{2} r\sin\theta & \frac{1}{2} r\sin\theta \\ \frac{r}{b} & -\frac{r}{b} \end{bmatrix} \begin{bmatrix} \dot{\varphi}_r \\ \dot{\varphi}_l \end{bmatrix}$

Wheel Models

- To generalize our motion models, we want to derive the general model for a standard wheel

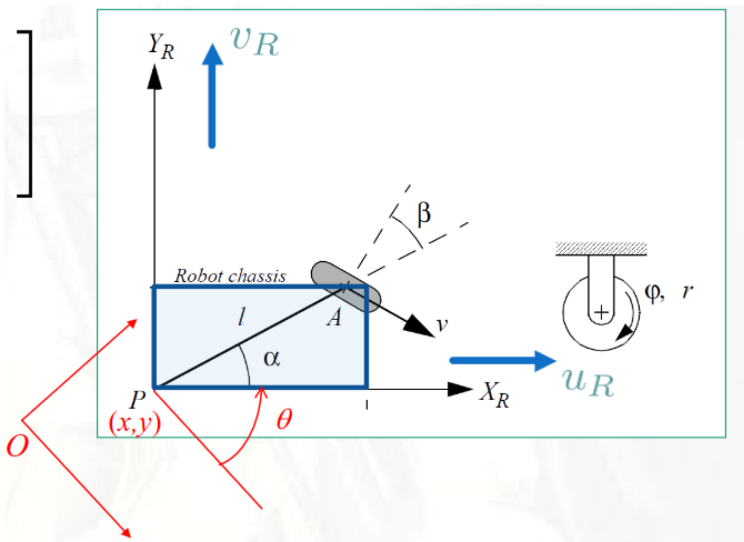

Figure 5: Derivation of the standard wheel model.

- Let $\underset{\rightarrow}{\mathcal{F}}_g$ be the global reference frame, let $\underset{\rightarrow}{\mathcal{F}}_r$ be the vehicle reference frame and $\underset{\rightarrow}{\mathcal{F}}_w$ be the wheel reference frame
  - For all 3 frames the 3rd vector points up
  - $\underset{\rightarrow}{r}_1$ is parallel to the vehicle and $\underset{\rightarrow}{r}_2$ is normal to it
  - $\underset{\rightarrow}{w}_1$ is normal to the wheel and $\underset{\rightarrow}{w}_2$ is parallel to it
  - Since everything is in the same plane, we have $\boldsymbol{C}_{rg} = \boldsymbol{C}_3(\theta), \boldsymbol{C}_{wr} = \boldsymbol{C}_3(\alpha + \beta)$
- We'll use the notation that $\underset{\rightarrow}{\rho}^{XY}$ being the position of point X measured in frame Y (if Y is omitted, it is the global frame)
- For any wheel, the kinematics of the vehicle is defined by the constraints of the wheel
  - We will assume that the wheel does not slip, so it cannot move in the direction of $\underset{\rightarrow}{w}_1$
    * This imposes a constraint $\underset{\rightarrow}{v}^A \cdot \underset{\rightarrow}{w}_1 = 0$, that is, $\underset{\rightarrow}{v}^A$ has no velocity in the direction of $\underset{\rightarrow}{w}_1$
  - The wheel can roll freely in the direction of $\underset{\rightarrow}{w}_2$
    * This means means that $\underset{\rightarrow}{v}^A \cdot \underset{\rightarrow}{w}_2 = -r\dot{\varphi}$
- We want $\underset{\rightarrow}{v}^A = \left.\frac{\mathrm{d}}{\mathrm{d}t}\underset{\rightarrow}{\rho}^A\right|_{\underset{\rightarrow}{\mathcal{F}}_g} = \left.\frac{\mathrm{d}}{\mathrm{d}t}\underset{\rightarrow}{\rho}^P\right|_{\underset{\rightarrow}{\mathcal{F}}_g} + \left.\frac{\mathrm{d}}{\mathrm{d}t}\underset{\rightarrow}{\rho}^{AP}\right|_{\underset{\rightarrow}{\mathcal{F}}_g} = \underset{\rightarrow}{v}^P + \left.\frac{\mathrm{d}}{\mathrm{d}t}\underset{\rightarrow}{\rho}^{AP}\right|_{\underset{\rightarrow}{\mathcal{F}}_r} + \underset{\rightarrow}{\omega}^{rg} \times \underset{\rightarrow}{\rho}^{AP}$
  - $\underset{\rightarrow}{v}^P = \underset{\rightarrow}{\mathcal{F}}_g^T \begin{bmatrix} \dot{x} \\ \dot{y} \\ 0 \end{bmatrix} = \underset{\rightarrow}{\mathcal{F}}_r^T \boldsymbol{C}_{rg} \begin{bmatrix} \dot{x} \\ \dot{y} \\ 0 \end{bmatrix} = \underset{\rightarrow}{\mathcal{F}}_r^T \begin{bmatrix} u_r \\ v_r \\ 0 \end{bmatrix}$
  - $\underset{\rightarrow}{\omega}^{rg}$ is the angular velocity of $\underset{\rightarrow}{\mathcal{F}}_r$ with respect to $\underset{\rightarrow}{\mathcal{F}}_g$ so it's simply $\underset{\rightarrow}{\mathcal{F}}_r^T \begin{bmatrix} 0 \\ 0 \\ \dot{\theta} \end{bmatrix}$
  - $\underset{\rightarrow}{\rho}^{AP} = \underset{\rightarrow}{\mathcal{F}}_r^T \begin{bmatrix} l\cos\alpha \\ l\sin\alpha \\ 0 \end{bmatrix}$ so it has a derivative of 0

- Therefore we can get $\underset{\rightarrow}{v}^A = \underset{\rightarrow}{v}^D + \underset{\rightarrow}{\omega}^{rg} \times \underset{\rightarrow}{\rho}^{AP}$ and then express it in frame $\underset{\rightarrow}{\mathcal{F}}_w^T$
  - This works out to be $\underset{\rightarrow}{\mathcal{F}}_w^T \begin{bmatrix} u_R \cos(\alpha+\beta) + v_R \sin(\alpha+\beta) + l\dot{\theta}\sin\beta \\ -u_R \sin(\alpha+\beta) + v_R \cos(\alpha+\beta) + l\dot{\theta}\cos\beta \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -r\dot{\varphi} \\ 0 \end{bmatrix}$ (due to the constraints) where u_R, v_R are the components of the robot's velocity along and normal to its frame
- The two equations $\begin{cases} u_R \cos(\alpha+\beta) + v_R \sin(\alpha+\beta) + l\dot{\theta}\sin\beta = 0 \\ -u_R \sin(\alpha+\beta) + v_R \cos(\alpha+\beta) + l\dot{\theta}\cos\beta = -r\dot{\varphi} \end{cases}$ define the wheel kinematics

Lecture 5, Sep 21, 2023

Introduction to Control Theory

Stability

- Consider the first-order linear time-invariant system $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}$
  - We diagonalize the system so that $\boldsymbol{P}^{-1}\boldsymbol{A}\boldsymbol{P} = \boldsymbol{\Lambda}$
  - Then we express $\boldsymbol{x}$ as a linear combination of eigenvectors: $\boldsymbol{x}(t) = \sum_{\alpha=1}^{n} \eta_\alpha(t)\boldsymbol{p}_\alpha$
    - η are the coordinates
  - Substituting it back into the equation of motion, we get $\dot{\eta}_\alpha = \lambda_\alpha \eta$ for $\alpha = 1, \dots, n$
  - Therefore we can solve it as $\eta_\alpha(t) = \eta_\alpha(0)e^{\lambda_\alpha t}$
- For this system, we know $\boldsymbol{x} = \boldsymbol{0}$ is a solution; when talking about stability, we consider the long-term behaviour of the differential equation and see if it goes back to 0
  - The $\eta_\alpha(t)$ are disturbances to the system, so we want them to be eliminated eventually
- If $\mathrm{Re}(\lambda_\alpha) < 0$ then as $t \to \infty$, we have all $\eta_\alpha \to 0 \implies \boldsymbol{x} \to \boldsymbol{0}$

Definition

A linear system $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}$ is *stable* if $\mathrm{Re}(\lambda_\alpha) \leq 0$ for all α; it is *asymptotically stable* if $\mathrm{Re}(\lambda_\alpha) < 0$. This works even for nondiagonalizable matrices by considering their Jordan forms.

For nonlinear systems $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$, we can consider local stability in the neighbourhood of a solution by linearizing the system using the Jacobian,

$$\boldsymbol{A} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}^T} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

PID Control

- PID control can be used to address two types of problems: the regulator problem (eliminating disturbances to the system) and the servo or tracking problem (tracking the output to a trajectory)
- In control theory the thing being controlled is referred to as the *plant*, a combination of actuators and processes
- Consider a simple single-variable, first-order linear system $\dot{x} + \sigma x = u, x(0) = x_0$ where x is the state variable and u is the control variable; we want to track the system to x_d
  - The eigenvalue for this system is $-\sigma$ (see this by $\dot{x} = -\sigma x$), so if $\sigma < 0$, this system is unstable
  - Define the error $e = x - x_d$ and let $u = -k_p e(t)$, so $\dot{x} + (\sigma + k_p)x = x_d$
  - For this system, $x_h = e^{-(\sigma+k_p)t}, x_p = \dfrac{k_p x_d}{\sigma + k_p}$ so the solution is $\dfrac{k_p x_d}{\sigma + k_p} + \left(x_0 - \dfrac{k_p x_d}{\sigma + k_p}\right)e^{-(\sigma+k_p)t}$

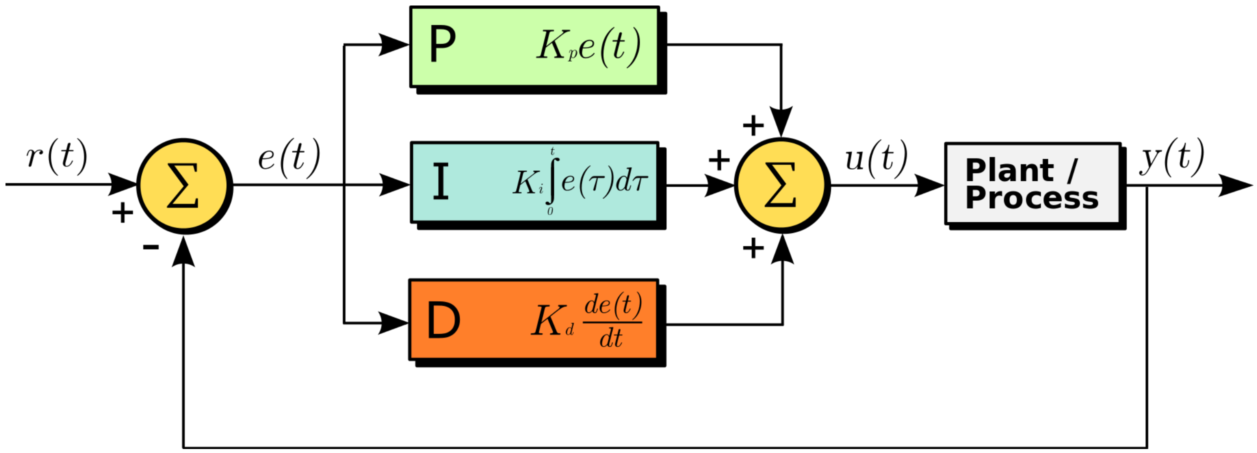

Figure 6: Diagram of PID control.

        * Therefore even if $\sigma < 0$, as long as we choose a sufficiently high k_p, the system can be stable
        * However if we let $t \to \infty$ we have $x = \dfrac{k_p x_d}{\sigma + k_p} \neq x_d$, so we have a steady-state error
    - Let's add an integral term: $u = -k_p e - k_i \displaystyle\int e(\tau)\,\mathrm{d}\tau$
        * Substituting in u and differentiating, we have $\ddot{x} + (\sigma + k_p)x + k_i x = k_i x_d$
        * The general solution is $x_h = c_1 e^{\lambda_1 t} + c_2 e^{\lambda_2 t}$
        * The particular solution is just $x = x_d$
        * The complete solution is $x(t) = x_d + c_1 e^{\lambda_1 t} + c_2 e^{\lambda_2 t}$
            - $\lambda = \dfrac{-(\sigma + k_p) \pm \sqrt{(\sigma + k_p)^2 - k_i}}{2}$
            - If $\mathrm{Re}(\lambda_i) < 0$, then as $t \to \infty$, $x(t) \to x_d$ and we have no steady-state error
            - Now λ might have an imaginary component, so our system may have oscillations; it could be underdamped, overdamped or critically damped depending on the gains
        * This system is stable if $k_i > 0, k_p + \sigma > 0$
- More generally, our state equation can be $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}$
    - Our feedback is $\boldsymbol{u} = -\boldsymbol{F}\boldsymbol{x}$ where $\boldsymbol{F}$ is the gain matrix
    - This gives $\dot{\boldsymbol{x}} = (\boldsymbol{A} - \boldsymbol{B}\boldsymbol{F})\boldsymbol{x}$
    - We can now make this system stable by finding an $\boldsymbol{F}$ that modifies the eigenvalues of $\boldsymbol{A}$
        * Whether we can always find such an $\boldsymbol{F}$ is related to the controllability of the system
- Even more generally, for nonlinear systems $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$, we can choose to linearize locally as before, or we can try heuristic feedback, with either linear or nonlinear control
- Now consider a second-order plant $\ddot{x} + \sigma\dot{x} + \eta x = u$
    - We will also add a derivative term: $u(t) = -k_p e(t) - k_d \dot{e}(t) - k_i \displaystyle\int e(\tau)\,\mathrm{d}\tau$
    - Substituting this and differentiating, we will get a third order differential equation
    - This gives us a new set of stability requirements
- Response characteristics:
    - Rise time: the amount of time for the output to approach the input
        * There is no set convention on this; often it's defined as the time from 0 to 100% of the desired output, sometimes it's 10% to 90%
    - Overshoot: the amount over the desired output that the maximum value is
    - Settling time: time to reach and stay within a certain band δ of the desired output
    - Steady-state error: remaining error as $t \to \infty$
- The gains change the characteristics of the response; depending on the system, different characteristics may be desired

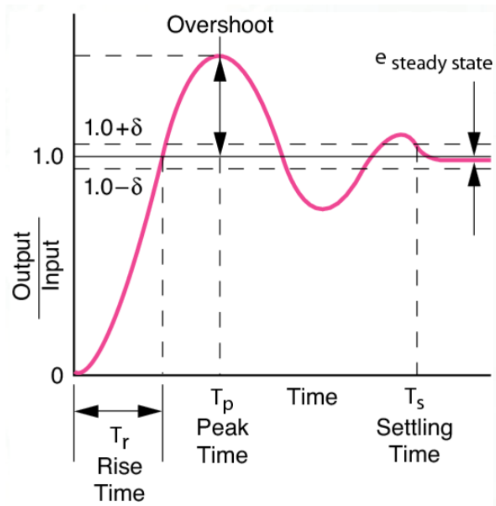

Figure 7: Example response of a PID controller.

Gain	Rise Time	Overshoot	Settling Time	Steady Error	Stability
k_p	Decrease	Increase	Little effect	Decrease	No effect
k_i	Decrease	Increase	Increase	Eliminate	Degrade
k_d	Little effect	Decrease	Decrease	No effect	Improve

Figure 8: Effect of increasing different gains on a PID controller.

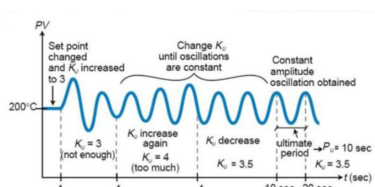

Figure 9: Ziegler-Nichols tuning sequence.

- The Ziegler-Nichols method is one among many methods to tune PID gains:
  1. Suppress the integral and derivative terms completely
  2. Create a small disturbance by suddenly changing the setpoint
  3. Increase k_p until the system is oscillating with constant amplitude
  4. Record the gain value as k_u, the oscillation period T_u, and refer to the table to set k_p, k_i, k_d

Type of Control	k_p	k_i	k_d
P	$0.50k_u$	---	---
PI	$0.45k_u$	$1.2k_u/T_u$	---
PD	$0.80k_u$	---	$0.125k_uT_u$
Classic PID	$0.60k_u$	$2.0k_u/T_u$	$0.125k_uT_u$

Figure 10: Ziegler-Nichols table of gains.

- PID control is prone to common problems:
  - Noise in the derivative: derivatives are typically numerically calculated and can be quite noisy
    * This can be mediated by attaching a low-pass filter on the signal to remove high-frequency components
  - Integral windup: error can build up in the integral term, making it overwhelm the other control terms
    * This can be mediated by removing the i term after the desired value is reached, capping the error integral, or reinitializing the i term
  - Deadband: the region where the control input does not affect the actuator (e.g. due to friction)
    * This can be mediated by commanding a minimum control input when in the deadband so the control is not useless

Applications in Robotics

- Consider robot with a bicycle model; we want to drive it to a desired goal point (x_d, y_d)
  - Proportional control: $v = -k_{p,v}\sqrt{(x-x_d)^2+(y-y_d)^2}, \theta_d = \tan\dfrac{y-y_d}{x-x_d}, \gamma = -k_{p,\gamma}(\theta-\theta_d)$
- What if we wanted to follow a line $ax + by + c = 0$?
  - We can measure the crosstrack error by $\delta = \dfrac{ax+by+c}{\sqrt{a^2+b^2}}$ (normal distance to line)
  - Then $\gamma_\delta = -k_{p,\delta}\delta$ makes us steer the robot towards the line
  - But now we want to keep the robot on the line, so let $\theta_d = \tan^{-1}\left(-\dfrac{a}{b}\right)$ and $\gamma_\theta = -k_{p,\theta}(\theta-\theta_d)$ steers us towards the line
  - These two terms are combined, and a fixed speed is added for this simple proportional control
- What if we wanted to follow a path?
  - Let $e = \sqrt{(x-x_d)^2+(y-y_d)^2} - d$, and then apply PI control on the velocity using this error
    * In effect this follows a set point at a distance d ahead all the time
    * This is because without the $-d$, e will always be positive and so we will get integral wind-up, where the integral term overwhelms the control
  - The steering can be controlled using the same way as when moving to a goal point
- Consider a robot with a unicycle model; we want to move it to a pose (x_d, y_d, θ_d)
  - We will transform our variables (x, y, θ) to (ρ, α, β), where ρ is the distance to the setpoint, α is the angle from the line that connects directly to the target
    * $\rho = \sqrt{\Delta_x^2 + \Delta_y^2}$
    * $\alpha = \tan^{-1}\dfrac{\Delta_y}{\Delta x} - \theta$
    * $\beta = -\theta - \alpha$
  - We want to regulate $(\rho, \alpha, \beta) = (0, 0, 0)$
    * Apply proportional control on v with ρ, and ω with α and β

Lecture 6, Sep 26, 2023

Dubin's Model

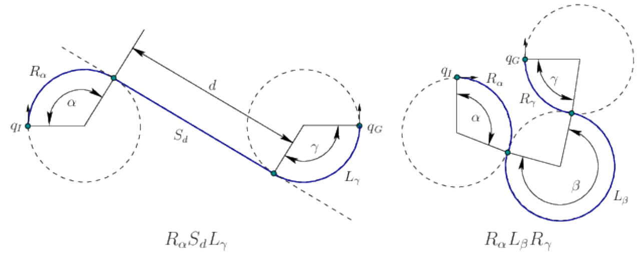

Figure 11: Example of Dubin's curves.

- Lester Eli Dubins proved that the shortest path between two points (with specified heading), with a minimum turning radius constraint, is a path with only straight and circular segments, corresponding to one of 6 types of curves:
  - LRL, RLR, LSL, LSR, RSL, RSR
  - Where L is a left turn of minimum radius, R is a right turn of minimum radius, and S is a straight line segment
  - This is the minimum-time path for a robot if the robot can only go forward at a constant velocity or stop
- To find the curves, we can draw two circles around each endpoint (one corresponding to a direction of rotation) and try to connect a circle on one side to a circle on the other side

Example: Wheel Constraint Model

- (see Xournal++ notes)

Lecture 7, Sep 28, 2023

Stability for Nonlinear Systems – Lyapunov's Method

- In general a nonlinear model is characterized by $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$
- One approach is locally linearizing using the Jacobian around a particular state and control input
  - $\Delta\dot{\boldsymbol{x}} = \boldsymbol{A}\Delta\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}$ where $\Delta\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{x}_d$ and $\boldsymbol{x}_d$ is the set point
  - $\boldsymbol{A} = \left.\dfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}^T}\right|_{\boldsymbol{x}=\boldsymbol{x}_d}, \boldsymbol{B} = \left.\dfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}^T}\right|_{\boldsymbol{u}=\boldsymbol{0}}$
  - With this we can apply the normal feedback methods with $\boldsymbol{u} = -\boldsymbol{F}\Delta\boldsymbol{x} \implies \Delta\dot{\boldsymbol{x}} = (\boldsymbol{A} - \boldsymbol{B}\boldsymbol{F})\Delta\boldsymbol{x}$ and choose $\boldsymbol{F}$ appropriately to put the poles in the left-half plane
  - Because this a local approximation, it will not work when the state is significantly different from the linearization point
- Another approach is *gain scheduling*, where we design a set of gains for a variety of different set points of the nonlinear system (i.e. "scheduling" the gains according to where you are in state space)
  - However this requires a lot of work and more importantly cannot guarantee stability
- To guarantee stability for a nonlinear system, we can use Lyapunov's method

Definition

The solution $\boldsymbol{x}(t; \boldsymbol{x}_0, t_0)$ to the system $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, t)$ is said to be *stable* in the Lyapunov sense (aka L-stable) if

$$\forall \varepsilon > 0, \exists \delta > 0 \text{ s.t. } \|\Delta \boldsymbol{x}_0\| < \delta \implies \forall t > t_0, \|\Delta \boldsymbol{x}\| < \varepsilon$$

$\boldsymbol{x}$ is *asympotitcally stable* if $\lim_{t\to\infty} \|\Delta \boldsymbol{x}\| = 0$; *exponential stability* further requires that $\|\Delta \boldsymbol{x}\|$ decreases exponentially.

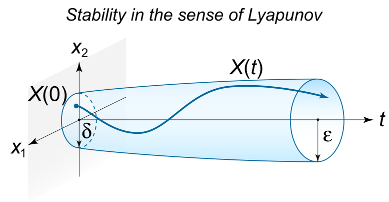

Figure 12: Illustration of Lyapunov stability.

Definition

A function $v(\boldsymbol{x})$ is *positive-definite* if

$$\forall \boldsymbol{x} \neq \mathbf{0}, v(\boldsymbol{x}) > 0 \text{ and } v(\mathbf{0}) = 0$$

and *negative-definite* if

$$\forall \boldsymbol{x} \neq \mathbf{0}, v(\boldsymbol{x}) < 0 \text{ and } v(\mathbf{0}) = 0$$

$v(\boldsymbol{x})$ is positive/negative-*semidefinite* if $v(\boldsymbol{x}) \geq 0 / v(\boldsymbol{x}) \leq 0$ for all $\boldsymbol{x}$.

Theorem

Let $\dot{\boldsymbol{x}} = \boldsymbol{f}(x)$ with an equilibrium at $\boldsymbol{x} = \mathbf{0}$; if we can find a positive-definite $v(\boldsymbol{x})$, and $\dot{v}(\boldsymbol{x})$ is negative-semidefinite, then $\boldsymbol{x} = \mathbf{0}$ is stable. If $\dot{v}(\boldsymbol{x})$ is negative-definite, then $\boldsymbol{x} = \mathbf{0}$ is asymptotically stable. Note that

$$\dot{v} = \frac{\partial v}{\partial \boldsymbol{x}^T}\dot{\boldsymbol{x}} = \frac{\partial v}{\partial \boldsymbol{x}^T}\boldsymbol{f}(\boldsymbol{x})$$

This function $v(\boldsymbol{x})$ is known as a *Lyapunov function.*

- $v(\boldsymbol{x})$ can be thought of as a potential energy surface; since $\dot{v}(\boldsymbol{x})$ is negative-(semi)definite, we always go down the surface, and since $v(\boldsymbol{x})$ is positive definite, we can't go down lower than 0, which is the location of the equilibrium
  - If $\dot{v}(\boldsymbol{x})$ is merely negative-semidefinite, we can get "stuck" before reaching the equilibrium (e.g. in a local minimum), but the solution is still stable
- Just because you can't find a Lyapunov function doesn't mean the system is unstable!
- However we can invert the result and find a positive-definite $\dot{v}(x)$, which would mean the system is unstable
- Example: $\dot{x}_1 = x_2 + \alpha x_1(x_1^2 + x_2^2), \dot{x}_2 = -x_1 + \alpha x_2(x_1^2 + x_2^2)$
  - We can take $v(x_1, x_2) = \frac{1}{2}(x_1^2 + x_2^2)$ which is clearly positive-definite
  - $\dot{v} = x_1\dot{x}_1 + x_2\dot{x}_2 = \alpha(x_1^2 + x_2^2)$
  - Therefore the system is asymptotically stable if $\alpha < 0$ or merely stable if $\alpha \leq 0$
  - For this example we can also say that if $\alpha > 0$, the system is unstable since $\dot{v}$ is positive-definite

Theorem

Lasalle's extension: If $\dot{v}$ is only negative-semidefinite, but the only solution to $\dot{v}(\boldsymbol{x}) = 0$ and $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$ is $\boldsymbol{x} = \boldsymbol{0}$, then $\boldsymbol{x} = \boldsymbol{0}$ is asymptotically stable.

- The idea is that Lyapunov's theorem considers all $\boldsymbol{x}$, but we only care about the ones that satisfy the equation of motion; so if $\dot{v}(\boldsymbol{x}) = \boldsymbol{0}$ is only possible at $\boldsymbol{x} = 0$ if the equation of motion must be satisfied, then the system is still asymptotically stable
  - Usually when Lasalle's extension applies, we have a $\dot{v}$ that is zero when only some of the x_i are zero, but does not require all of them to be zero; so if satisfying $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$ with these $x_i = 0$ requires all the other coordinates to be zero, then $\boldsymbol{x} = \boldsymbol{0}$ is still asymptotically stable

Example: Feedback Tracking Problem

- Consider a robot with a unicycle model; we want to track a path $(x_d(t), y_d(t), \theta_d(t))$
  - $\begin{bmatrix} \dot{x}_d \\ \dot{y}_d \\ \dot{\theta}_d \end{bmatrix} = \begin{bmatrix} u_d \cos\theta_d \\ u_d \sin\theta_d \\ \omega_d \end{bmatrix}$
  - u_d, ω_d are the control inputs that will get us exactly to the setpoint in a perfect world; however since we might have disturbances we need feedback control
- We will do a coordinate transform into the robot coordinate system with axes ξ parallel to the robot and η perpendicular to it
  - $\boldsymbol{\xi} = \begin{bmatrix} \xi \\ \eta \\ \theta \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$ corresponding to a rotation about the third axis
  - $\dot{\xi} = \dot{x}\cos\theta + \dot{y}\sin\theta + (-x\sin\theta + y\cos\theta)\dot{\theta} = \dot{x}\cos\theta + \dot{y}\sin\theta + \eta\dot{\theta}$
  - $\dot{\eta} = -\dot{x}\sin\theta + \dot{y}\cos\theta + (x\cos\theta + y\sin\theta)\dot{\theta} = -\dot{x}\sin\theta + \dot{y}\cos\theta + \xi\dot{\theta}$
- We can make the same transformation for the desired coordinates $(x_d, y_d, \theta_d) \to (\xi_d, \eta_d, \theta_d)$
  - $\xi_d = x_d \cos\theta + y_d \sin\theta$
  - $\eta_d = -x_d \sin\theta + y_d \cos\theta$
  - $\dot{\xi}_d = \dot{x}_d \cos\theta + \dot{y}_d \sin\theta + \eta_d\dot{\theta} = u_d \cos(\theta - \theta_d) + \eta_d\omega$
  - $\dot{\eta}_d = -\dot{x}_d \sin\theta + \dot{y}_d \cos\theta + \xi_d\dot{\theta} = u_d \sin(\theta - \theta_d) - \xi_d\omega$
- Let the error $e_x = \xi - \xi_d, e_y = \eta - \eta_d, e_\theta = \theta - \theta_d$
  - We've converted a tracking problem to a regulator problem
  - We want to send all these error terms to zero
- The error derivatives are $\dot{\boldsymbol{e}} = \begin{bmatrix} \dot{e}_x \\ \dot{e}_y \\ \dot{e}_\theta \end{bmatrix} = \begin{bmatrix} u_d \cos e_\theta + u + e_y\omega \\ u_d \sin e_\theta - e_x\omega \\ \omega - \omega_d \end{bmatrix}$
- Our control algorithm will be $u = -k_x e_x - u_d \cos e_\theta, \omega = \omega_d - k_\theta \sin e_\theta - u_d e_y$
  - In the end we get a nonlinear function $\dot{\boldsymbol{e}} = \boldsymbol{\Phi}(\boldsymbol{e})$
- Choose a candidate Lyapunov function $v(e_x, e_y, e_\theta) = \dfrac{1}{2}(e_x^2 + e_y^2) + (1 - \cos e_\theta)$
  - Notice that these terms are energy-like: the $\dfrac{1}{2}(e_x^2 + e_y^2)$ is spring energy in 2D and $1 - \cos e_\theta$ is the energy of a pendulum; this is usually a good guide to selecting candidate Lyapunov functions
  - $\dot{v} = -k_x e_x^2 - k_\theta \sin^2 e_\theta$
  - If $k_x, k_\theta > 0$, $\dot{v}$ is negative definite but only with respect to e_x and e_θ; this means it is negative-semidefinite
  - Lyapunov's theorem alone tells us only that the system is stable, but not necessarily asymptotically so
  - We can try applying Lasalle's extension, if we can show that $e_x = e_\theta = 0 \implies e_y = 0$ in order to satisfy the equation of motion
    * If we substitute back in $e_x = e_\theta = 0$ (and $\dot{e}_x = \dot{e}_\theta = 0$) we can prove that $e_y = 0$, so by Lasalle's extension this system is asymptotically stable

Lecture 8, Oct 3, 2023

Localization

- Localization is the process of determining where the robot is
  - Do we already have a map (i.e. landmarks) or do we need to build one?
  - How do we measure uncertainty arising from sensors and actuators?
  - How do we formulate the best estimate for localization from uncertain measurements?
- Any sensor measurement will invariably be corrupted by noise to some extent
  - Measurements are often distributed according to a Gaussian, due to the central limit theorem

Propagation of Error – Odometry Example

- How does uncertainty in measurements propagate?
- The covariance matrix generalizes variance to multiple dimensions
  - $\boldsymbol{\Sigma} = \mathbb{E}[(\boldsymbol{x} - \mathbb{E}(\boldsymbol{x}))(\boldsymbol{x} - \mathbb{E}(\boldsymbol{x}))^T]$

    $$= \begin{bmatrix} \sigma_{x_1}^2 & \sigma_{x_1 x_2} & \cdots \\ \sigma_{x_2 x_1} & \sigma_{x_2}^2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$
  - If we take the covariance between two different variables, it is known as the *cross-covariance*
  - $\operatorname{cov}(\boldsymbol{x}, \boldsymbol{y}) = \mathbb{E}[(\boldsymbol{x} - \boldsymbol{\mu}_x)(\boldsymbol{y} - \boldsymbol{\mu}_y)^T]$
- Note some important properties of covariance:
  1. $\boldsymbol{\Sigma} = \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T] - \boldsymbol{\mu}\boldsymbol{\mu}^T$
  2. $\boldsymbol{\Sigma}^T = \boldsymbol{\Sigma} \geq 0$, i.e. the covariance matrix is semi-definite
  3. $\operatorname{cov}(\boldsymbol{x}, \boldsymbol{y}) = \operatorname{cov}(\boldsymbol{y}, \boldsymbol{x})^T$
  4. $\operatorname{cov}(\boldsymbol{x}_1 + \boldsymbol{x}_2, \boldsymbol{y}) = \operatorname{cov}(\boldsymbol{x}_1, \boldsymbol{y}) + \operatorname{cov}(\boldsymbol{x}_2, \boldsymbol{y})$, i.e. covariance is bilinear
  5. $\operatorname{cov}(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{a}, \boldsymbol{B}\boldsymbol{y} + \boldsymbol{b}) = \boldsymbol{A} \operatorname{cov}(\boldsymbol{x}, \boldsymbol{y}) \boldsymbol{B}^T$
  6. $\operatorname{cov}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{0}$ if $\boldsymbol{x}$ and $\boldsymbol{y}$ are independent (but a zero covariance does not mean no correlation)
- Let $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$, then in general we can see how $\boldsymbol{\Sigma}_y$ relates to $\boldsymbol{\Sigma}_x$
  - By Taylor expansion $\boldsymbol{y} = \boldsymbol{y}_0 + (\vec{\nabla}\boldsymbol{f})_0(\boldsymbol{x} - \boldsymbol{x}_0)$
    * Then $(\vec{\nabla}\boldsymbol{f})_0\boldsymbol{x} = \boldsymbol{A}\boldsymbol{x}$ and $\boldsymbol{y}_0 - (\vec{\nabla}\boldsymbol{f})_0\boldsymbol{x}_0 = \boldsymbol{a}$
    * By property 5 above, $\boldsymbol{\Sigma}_y = (\vec{\nabla}\boldsymbol{f})_0\boldsymbol{\Sigma}_x(\vec{\nabla}\boldsymbol{f})_0^T$
- Consider the problem of determining pose using only odometry, i.e. movement of the wheels $\Delta\boldsymbol{s} = \begin{bmatrix} \Delta s_l \\ \Delta s_r \end{bmatrix}$
  - $\Delta\boldsymbol{x} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} = \begin{bmatrix} \dfrac{\Delta s_r + \Delta s_l}{2}\cos\theta \\ \dfrac{\Delta s_r + \Delta s_l}{2}\sin\theta \\ \dfrac{\Delta s_r - \Delta s_l}{b} \end{bmatrix} = \begin{bmatrix} \Delta s\cos\theta \\ \Delta s\sin\theta \\ \dfrac{\Delta s_r - \Delta s_l}{b} \end{bmatrix}$
- Our new position is given by $\Delta x' = \boldsymbol{f}(\boldsymbol{x} + \Delta\boldsymbol{s})$
  - Linearize: $\boldsymbol{x}' = \boldsymbol{x} + (\vec{\nabla}_{\Delta s}\boldsymbol{f})_0\Delta\boldsymbol{s}$ where $\vec{\nabla}_{\Delta s}\boldsymbol{f}$ is the Jacobian $\begin{bmatrix} \frac{1}{2}\cos\theta & \frac{1}{2}\cos\theta \\ \frac{1}{2}\sin\theta & \frac{1}{2}\sin\theta \\ -b^{-1} & b^{-1} \end{bmatrix}$
  - Assume uncorrelated odometry error and $\boldsymbol{\Sigma}_\Delta = \begin{bmatrix} \sigma_{\Delta,l}^2 & 0 \\ 0 & \sigma_{\Delta,r}^2 \end{bmatrix}$
  - Error propagates as $\boldsymbol{\Sigma}_{x'} = \boldsymbol{\Sigma}_x + (\vec{\nabla}_{\Delta s}\boldsymbol{f})\boldsymbol{\Sigma}_\Delta(\vec{\nabla}_{\Delta_s}\boldsymbol{f})^T$
  - Notice that the part we add is always positive due to the positive-semidefiniteness of the covariance, so the error always grows!
- This means using odometry alone, our estimate of where the robot is will get worse with time

1D Kalman Filtering

- If we have n measurements for a static variable x, how do we obtain the best estimate $\hat{x}$?
  - We can try to minimize $e = \sum_{k=1}^{n} w_k(\hat{x} - x_k)^2$, i.e. weighted least squares
  - The weight can be $w_k = \frac{1}{\sigma_k^2}$, so that measurements with higher variance (uncertainty) are weighted less
  - The solution is given by $\hat{x} = \frac{\sum_k \sigma_k^{-2} x_k}{\sum_k \sigma_k^{-2}}$
    * This is a weighted average of all the x_k with weights $\frac{1}{\sigma_k^2}$
- Consider the case where we have only 2 measurements, then $\hat{x} = \frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2} x_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} x_2$
  - Then the variance of $\hat{x}$ is $\operatorname{var} \hat{x} = \left(\frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2}\right)^2 \sigma_1^2 + \left(\frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}\right)^2 \sigma_2^2 = \frac{\sigma_1^2 \sigma_2^2}{\sigma_1^2 + \sigma_2^2}$
  - But note, this is less than both σ_1^2 and σ_2^2!
- Note also $\hat{x} = \frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2} x_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} x_2 = x_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}(x_2 - x_1)$
  - This is a much more convenient form for us, since we've turned it from batch form (needing all measurements at once) into a recursive form (where we can continuously update)
- If we have $\hat{x}_k, \hat{\sigma}_k$ as the previous estimate at the current timestep, and we get a new measurement x_{k+1} with variance σ_{k+1}^2 then we can update:
  - $\hat{x}_{k+1} = \hat{x}_k + \frac{\hat{\sigma}_k^2}{\hat{\sigma}_k^2 + \sigma_{k+1}^2}(x_{k+1} - \hat{x}) = \hat{x}_k + W_{k+1}(x_{k+1} - \hat{x}_k)$
  - $\hat{\sigma}_{k+1} = \frac{\hat{\sigma}_k^2 \sigma_{k+1}^2}{\hat{\sigma}_k^2 + \sigma_{k+1}^2} = \hat{\sigma}_k^2 - W_{k+1}\hat{\sigma}_k^2$
  - W is known as the *Kalman gain*
  - We can see that this is similar to a feedback control law – the correction to the state is the gain multiplied by the "error"
- Kalman filtering is a special case of Bayesian filtering, where the distribution is a Gaussian
- But we still haven't accounted for the fact that x may be dynamic, i.e. it can evolve over time; to account for this, we will predict what the new state should be based on the old estimate, and then compute the error from the measurement of the new state
  - Consider the 1D state update equation $x_{k+1} = x_k + u_k + v_k$ where u_k is the control input and v_k is some noise
    * $v_k \sim \mathcal{N}(0, \varsigma_k^2)$, i.e. normally distributed, zero-mean with variance ς_k^2
    * Assume that u_k can be accurately delivered, i.e. there is no noise
  - Let $\hat{x}_{k|k}$ be the estimate of x at step k, given measurements $\{ x_0, x_1, \dots, x_k \}$
  - Let $\hat{x}_{k+1|k}$ be the prediction of x at step $k+1$, given measurements $\{ x_0, x_1, \dots, x_k \}$
    * $\hat{x}_{k+1|k} = \hat{x}_{k|k} + u_k$ (note since the noise is zero-mean, we can disregard it)
  - Now to get $\hat{x}_{k+1|k+1}$, we can use the same Kalman update formula as above
    * $\hat{\sigma}_{k+1|k}^2 = \hat{\sigma}_{k|k}^2 + \varsigma_{k+1}^2$
    * $W_{k+1} = \frac{\sigma_{k+1|k}^2}{\sigma_{k+1|k}^2 + \sigma_{k+1}^2}$
    * $\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + W_{k+1}(x_{k+1} - \hat{x}_{k+1|k})$
    * $\hat{\sigma}_{k+1|k+1}^2 = \hat{\sigma}_{k+1|k}^2 - W_{k+1}\hat{\sigma}_{k+1|k}^2$
- Intuitively, Kalman filters combine an estimate and a new measurement, both of which have some uncertainty, and finds the most likely new state according to the distributions of error in both

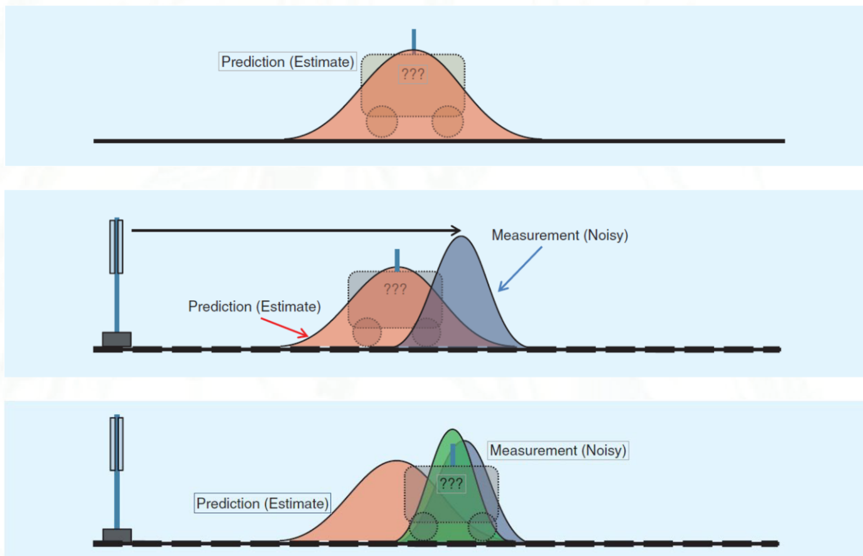

Figure 13: Diagram of Kalman filtering.

Lecture 9, Oct 5, 2023

Multidimensional Kalman Filtering

- We can generalize our model to multiple degrees of freedom with a separate measurement relation:
  - $\boldsymbol{x}_{k+1} = \boldsymbol{A}_k\boldsymbol{x}_k + \boldsymbol{B}_k\boldsymbol{u}_k + \boldsymbol{v}_k$ (the *state*, or *process equation*)
  - $\boldsymbol{z}_k + \boldsymbol{D}_k\boldsymbol{x}_k + \boldsymbol{w}_k$ (the *measurement model*)
  - Where $\boldsymbol{A}_k$ is the state update matrix, $\boldsymbol{B}_k$ is the control matrix, $\boldsymbol{z}_k$ is the measurement, $\boldsymbol{D}_k$ is the measurement matrix, $\boldsymbol{v}_k$ is the process (or model) noise and $\boldsymbol{w}_k$ is the measurement noise
  - Again assume $\boldsymbol{v}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}_k), \boldsymbol{w}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}_k)$, i.e. zero-mean noise with covariances $\boldsymbol{Q}_k, \boldsymbol{R}_k$
  - In practice we can find $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$ through testing and characterization of the system; depending on the model, we may be able to estimate it mathematically
- Let $\hat{\boldsymbol{x}}_{k|j}$ be the estimate of $\boldsymbol{x}_k$ given measurements $\{\, \boldsymbol{z}_0, \dots, \boldsymbol{z}_j \,\}$, with $\boldsymbol{P}_{k|j}$ as its covariance
- Kalman filtering is a two-branch process divided into state and covariance estimations:
  - Given $\hat{\boldsymbol{P}}_{k|k}$ (the previous best estimate), $\boldsymbol{u}_k$ (the control input), $\boldsymbol{P}_{k|k}$ (the previous covariance)
  - State estimation:
    1. Predict the next state: $\hat{\boldsymbol{x}}_{k+1|k} = \boldsymbol{A}_k\hat{\boldsymbol{x}}_{k|k} + \boldsymbol{B}_k\boldsymbol{u}_k$
    2. Predict the next measurement: $\hat{\boldsymbol{z}}_{k+1|k} = \boldsymbol{D}_{k+1}\hat{\boldsymbol{x}}_{k+1|k}$
    3. Calculate the measurement residual: $\boldsymbol{s}_{k+1} = \boldsymbol{z}_{k+1} - \hat{\boldsymbol{z}}_{k+1|k}$
    4. Update the state estimate: $\hat{\boldsymbol{x}}_{k+1|k+1} = \hat{\boldsymbol{x}}_{k+1|k} + \boldsymbol{W}_{k+1}\boldsymbol{s}_{k+1}$
  - Covariance estimation:
    1. Predict the state covariance: $\boldsymbol{P}_{k+1|k} = \boldsymbol{A}_k\boldsymbol{P}_{k|k}\boldsymbol{A}_k^T + \boldsymbol{Q}_k$
    2. Predict the measurement covariance: $\boldsymbol{S}_{k+1} = \boldsymbol{D}_{k+1}\boldsymbol{P}_{k+1|k}\boldsymbol{D}_{k+1}^T + \boldsymbol{R}_{k+1}$
    3. Calculate the Kalman gain: $\boldsymbol{W}_{k+1} = \boldsymbol{P}_{k+1|k}\boldsymbol{D}_{k+1}^T\boldsymbol{S}_{k+1}^{-1}$
    4. Update the state covariance: $\boldsymbol{P}_{k+1|k+1} = \boldsymbol{P}_{k+1|k} - \boldsymbol{W}_{k+1}\boldsymbol{S}_{k+1}\boldsymbol{W}_{k+1}^T$
- The state covariance estimation part can be combined into a single equation, which is known as the *Ricatti equation*
- This only works if we have noise – if $\boldsymbol{R}_{k+1} = \boldsymbol{0}$, often $\boldsymbol{S}_{k+1}$ is not invertible; however if $\boldsymbol{R}_{k+1}$ is invertible, then due to the positive-definiteness of $\boldsymbol{P}_{k+1|k}$, we are guaranteed that $\boldsymbol{S}$ is invertible
- Example: body in free fall
  - Model: $\begin{bmatrix} x_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ v_k \end{bmatrix} + \begin{bmatrix} -\frac{1}{2}g \\ -g \end{bmatrix}$

- We will measure the height: $z_k = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}x_k \\ v_k\end{bmatrix} + w_k$
- Take noise to be $\boldsymbol{Q}_k = \boldsymbol{0}, \boldsymbol{R}_k = 1$

Optimality of Kalman Filtering

- If we expand the update relations we get:
  - $\hat{\boldsymbol{x}}_{k+1} = \hat{\boldsymbol{x}}_{k+1|k} + \boldsymbol{W}_{k+1}(\boldsymbol{D}_{k+1}\boldsymbol{x}_{k+1} + \boldsymbol{w}_{k+1} - \boldsymbol{D}_{k+1}\hat{\boldsymbol{x}}_{k+1|k})$
  - $P_{k+1|k+1} = (\boldsymbol{1} - \boldsymbol{W}_{k+1}\boldsymbol{D}_{k+1})(\boldsymbol{A}_k\boldsymbol{P}_{k|k}\boldsymbol{A}_k + \boldsymbol{Q}_k)(\boldsymbol{1} - \boldsymbol{W}_{k+1}\boldsymbol{D}_{k+1})^T + \boldsymbol{W}_{k+1}\boldsymbol{R}_{k+1}\boldsymbol{W}_{k+1}^T$
- We would want to minimize $\varepsilon_{k+1}^2 = \mathbb{E}\left[\|\hat{\boldsymbol{x}}_{k+1|k+1} - \boldsymbol{x}_{k+1}\|^2\right]$, i.e. the expected error
  - $$\begin{aligned}\varepsilon_{k+1}^2 &= \mathbb{E}\left[\|\hat{\boldsymbol{x}}_{k+1|k+1} - \boldsymbol{x}_{k+1}\|^2\right] \\ &= \mathbb{E}\left[(\hat{\boldsymbol{x}}_{k+1|k+1} - \boldsymbol{x}_{k+1})^T(\hat{\boldsymbol{x}}_{k+1|k+1} - \boldsymbol{x}_{k+1})\right] \\ &= \operatorname{tr}\mathbb{E}\left[(\hat{\boldsymbol{x}}_{k+1|k+1} - \boldsymbol{x}_{k+1})(\hat{\boldsymbol{x}}_{k+1|k+1} - \boldsymbol{x}_{k+1})^T\right] \\ &= \operatorname{tr}\boldsymbol{P}_{k+1|k+1}\end{aligned}$$
  - This means that to minimize the expected error, we should minimize the covariance
- To minimize the error, we solve for $\dfrac{\partial\varepsilon_{k+1}^2}{\partial\boldsymbol{W}_{k+1}} = \boldsymbol{0}$ to get the optimal $\boldsymbol{W}$
  - Note: $\dfrac{\partial\operatorname{tr}\boldsymbol{AB}}{\partial\boldsymbol{B}} = \boldsymbol{A}^T$ for matrices $\boldsymbol{A}, \boldsymbol{B}$
  - If we do this, we get $\dfrac{\partial\operatorname{tr}\boldsymbol{P}_{k+1|k+1}}{\partial\boldsymbol{W}_{k+1}} = -2\boldsymbol{P}_{k+1|k+1}\boldsymbol{D}_{k+1}^T + 2\boldsymbol{W}_{k+1}\boldsymbol{S}_{k+1} = \boldsymbol{0}$ where $\boldsymbol{S}_{k+1}$ is defined above
  - Assuming $\boldsymbol{S}_{k+1}$ is invertible, solve to get $\boldsymbol{W}_{k+1} = \boldsymbol{P}_{k+1|k}\boldsymbol{D}_{k+1}^T\boldsymbol{S}_{k+1}^{-1}$
- Hence Kalman filtering is an *optimal* estimator

Extended Kalman Filtering (EKF)

- What if we didn't have a linear process/measurement model?
- In general, we can have $\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) + \boldsymbol{v}_k, \boldsymbol{z}_{k+1} = \boldsymbol{h}(\boldsymbol{x}_{k+1}) + \boldsymbol{w}_{k+1}$
  - Note we are assuming that noise is additive right now
- We simply linearize the system with the Jacobian
- For the predictions, we can directly do $\hat{\boldsymbol{x}}_{k+1|k} = \boldsymbol{f}(\hat{\boldsymbol{x}}_{k|k}, \boldsymbol{u}_k)$ and $\hat{\boldsymbol{z}}_{k+1|k} = \boldsymbol{h}(\hat{\boldsymbol{x}}_{k|k})$
- For the state covariance estimate, we will linearize about $\hat{\boldsymbol{x}}_{k+1|k}, \boldsymbol{u}_k$:
  - $\boldsymbol{A}_k = \left.\dfrac{\partial\boldsymbol{f}}{\partial\boldsymbol{x}_k^T}\right|_{\hat{\boldsymbol{x}}_{k+1|k}, \boldsymbol{u}_k}, \boldsymbol{B}_k = \left.\dfrac{\partial\boldsymbol{f}}{\partial\boldsymbol{u}_k^T}\right|_{\hat{\boldsymbol{x}}_{k+1|k}, \boldsymbol{u}_k}, \boldsymbol{D}_{k+1} = \left.\dfrac{\partial\boldsymbol{h}}{\partial\boldsymbol{x}_k^T}\right|_{\hat{\boldsymbol{x}}_{k+1|k}}$
    - Note that now the matrices such as $\boldsymbol{A}_k$ are dependent on our state!
- The procedure is identical to that of normal Kalman filtering, except the nonlinear model is used for prediction and measurement, while the linearized Jacobians are used for covariance estimation

Lecture 10, Oct 10, 2023

- Example: consider the problem $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}$; given that constant $P \in \mathbb{R}^{n\times n}$ is positive-definite and $\boldsymbol{A}^T\boldsymbol{P} + \boldsymbol{P}\boldsymbol{A}$ is negative-definite, show that $\boldsymbol{x} = \boldsymbol{0}$ is asymptotically stable
  - Candidate Lyapunov function: $v(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}$, which is positive-definite since $\boldsymbol{P}$ is positive-definite
  - $\dot{v}(\boldsymbol{x}) = \dfrac{\partial}{\partial t}\boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x} = \dot{\boldsymbol{x}}^T\boldsymbol{P}\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{P}\dot{\boldsymbol{x}} = \boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{P}\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{x}^T(\boldsymbol{A}^T\boldsymbol{P} + \boldsymbol{P}\boldsymbol{A})\boldsymbol{x}$ which is negative-definite since $\boldsymbol{A}^T\boldsymbol{P} + \boldsymbol{P}\boldsymbol{A}$ is negative-definite
  - Therefore by Lyapunov's method the solution $\boldsymbol{x} = \boldsymbol{0}$ is asymptotically stable
  - The condition $\boldsymbol{A}^T\boldsymbol{P} + \boldsymbol{P}\boldsymbol{A} = -\boldsymbol{L}$, where $\boldsymbol{P}, \boldsymbol{L}$ are both semi-definite, is called *Lyapunov's equation*; this condition is equivalent to saying that $\boldsymbol{A}$ has only eigenvalues with negative real parts

Lecture 11, Oct 12, 2023

Control (Actuator) Noise

- Consider the control input having some noise, so $\boldsymbol{u}_k = \boldsymbol{u}^*_{k|k} + \boldsymbol{n}_k$ where $\boldsymbol{u}_k$ is the actual control input delivered, $\boldsymbol{u}^*_{k|k}$ is the requested control input, and $\boldsymbol{n}_k$ is some zero-mean, Gaussian noise with covariance $\boldsymbol{N}_k$
- After linearization $\boldsymbol{x}_{k+1} = \hat{\boldsymbol{x}}_{k+1|k} + \boldsymbol{A}_k(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k}) + \boldsymbol{B}_k(\boldsymbol{u}_k - \boldsymbol{u}^*_{k|k}) + \boldsymbol{v}_k$
- In this case our a priori covariance estimate is $\boldsymbol{A}_k\boldsymbol{P}_{k|k}\boldsymbol{A}_k^T + \boldsymbol{B}_k\boldsymbol{N}_k\boldsymbol{B}_k^T + \boldsymbol{Q}_k$
- The rest stays unchanged

Kalman Filtering Example – Observability, Controllability, Detectability and Stabilizability

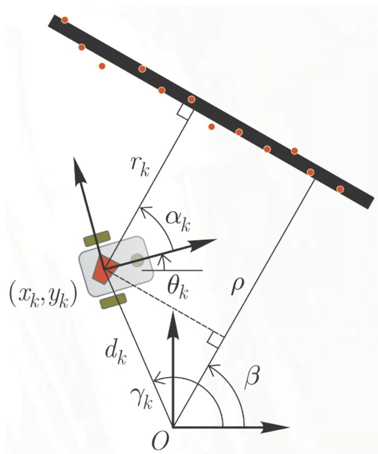

Figure 14: Example scenario.

- Consider a differentially steered robot: $\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) = \boldsymbol{A}_k\boldsymbol{x}_k + \underbrace{\begin{bmatrix} \frac{1}{2}\cos\theta_k & \frac{1}{2}\cos\theta_k \\ \frac{1}{2}\sin\theta_k & \frac{1}{2}\sin\theta_k \\ -b^{-1} & -b^{-1} \end{bmatrix}}_{\boldsymbol{B}_k} \underbrace{\begin{bmatrix} \Delta s_{l,k} \\ \Delta s_{r,k} \end{bmatrix}}_{\boldsymbol{u}_k} + \boldsymbol{v}_k$
- The robot measures the angle and perpendicular distance to a wall; the wall is specified in the map as an angle β and distance ρ
- The measurement model is: $\boldsymbol{z}_k = \begin{bmatrix} \alpha_k \\ r_k \end{bmatrix} = \begin{bmatrix} \beta - \theta_k \\ \rho - x_k\cos\beta - y_k\sin\beta \end{bmatrix} + \boldsymbol{w}_k = \boldsymbol{h}(\boldsymbol{x}_k) + \boldsymbol{w}_k$
  - Linearize: $\boldsymbol{D}_{k+1} = \dfrac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}_{k+1}^T} = \begin{bmatrix} 0 & 0 & -1 \\ -\cos\beta & -\sin\beta & 0 \end{bmatrix}$
- Note that for Kalman filters to work, the system has to be *observable*, that is, using sufficient measurements of $\boldsymbol{z}$, we can reconstruct $\boldsymbol{x}$
  - A system $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}, \boldsymbol{z} = \boldsymbol{D}\boldsymbol{x}$ is *observable* if the observability matrix $\boldsymbol{O} = \begin{bmatrix} \boldsymbol{D} \\ \boldsymbol{D}\boldsymbol{A} \\ \vdots \\ \boldsymbol{D}\boldsymbol{A}^{n-1} \end{bmatrix}$ has rank n
- The dual of observability is *controllability*, the ability to achieve any system state by using a sequence of control inputs $\boldsymbol{u}$
  - The controllability matrix is $\boldsymbol{C} = \begin{bmatrix} \boldsymbol{B} & \boldsymbol{A}\boldsymbol{B} & \boldsymbol{A}^2\boldsymbol{B} & \cdots & \boldsymbol{A}^n\boldsymbol{B} \end{bmatrix}$, which needs to be rank n for the system to be controllable

- Our system is not observable, since $\boldsymbol{A}$ is the identity and $\boldsymbol{D}$ has rank 2
  - This corresponds to the fact that we don't get enough information by just looking at the wall; we could be anywhere along the wall and still get the same measurement
  - In this case, the output of the filter is not guaranteed to be correct (but we have no way of telling this)
- *Detectability* is a weaker form of observability which requires an $\boldsymbol{L}$ to exist such that $\boldsymbol{A} + \boldsymbol{L}\boldsymbol{D}$ is stable
  - This means that the unobservable states are stable according to their own dynamics
- *Stabilizability* is the dual of detectability, which requires $\boldsymbol{K}$ to exist such that $\boldsymbol{A} + \boldsymbol{B}\boldsymbol{K}$ is stable
  - This means that the uncontrollable states are stable according to their own dynamics
- If a system is not observable, but it is still detectable and stabilizable, then the Kalman filter will still converge
- Note that stability in a discrete system requires that all $|\lambda| < 1$ (since we have $\boldsymbol{x}_n = \boldsymbol{A}^n \boldsymbol{x}_0$), whereas in a continuous system we require the eigenvalues to have negative real parts
  - This is known as *Schur stability*

Mapping

- Before, we assumed that we had the location of landmarks; how do we get those landmark locations in the first place?
  - To build a map, we need to localize; to localize, we need a map, leading to a chicken-and-egg problem
  - For now, we will assume we have perfect localization, and see how we can build a map of landmarks
- Suppose we have m landmarks each with coordinate $\boldsymbol{\xi}^{(i)}$; we want to estimate $\boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{\xi}^{(1)} \\ \vdots \\ \boldsymbol{\xi}^{(m)} \end{bmatrix}$
- We can try to use Kalman filtering!
- Since landmarks don't move, $\boldsymbol{\xi}_{k+1} = \boldsymbol{\xi}_k \implies \boldsymbol{A} = \boldsymbol{1}, \boldsymbol{B} = \boldsymbol{0}$, and there is no noise
- Measurements are modelled by $\boldsymbol{\zeta}_k = \boldsymbol{\eta}(\boldsymbol{x}_k, \boldsymbol{\xi}_k) + \boldsymbol{\varpi}_k$
  - Example: if we treat landmarks as points $(\xi^{(i)}, \eta^{(i)})$ and we measure their bearing ρ and distance ϕ, then:
    * $\rho_k^{(i)} = \sqrt{(\xi_k^{(i)} - x_k)^2 + (\eta_k^{(i)} - y_k)^2}$
    * $\phi_k^{(i)} = \tan^{-1} \dfrac{\eta_k^{(i)} - y_k}{\xi_k^{(i)} - x_k} - \theta_k$
  - This can then be linearized to obtain $\boldsymbol{D}_{k+1}^{(i)}$
- Apply EKF:
  - State estimation:
    1. $\hat{\boldsymbol{\xi}}_{k+1|k} = \hat{\boldsymbol{\xi}}_{k|k}$
    2. $\boldsymbol{\zeta}_{k+1|k} = \boldsymbol{\eta}(\boldsymbol{x}_k, \hat{\boldsymbol{\xi}}_{k+1|k})$
    3. $\boldsymbol{\nu}_{k+1} = \boldsymbol{\zeta}_{k+1} - \hat{\boldsymbol{\zeta}}_{k+1|k}$
    4. $\hat{\boldsymbol{\zeta}}_{k+1|k+1} = \hat{\boldsymbol{\zeta}}_{k+1|k} + \boldsymbol{W}_{k+1}\boldsymbol{\nu}_{k+1}$
  - Covariance estimation:
    1. $\boldsymbol{P}_{k+1|k} = \boldsymbol{P}_{k|k}$
    2. $\boldsymbol{S}_{k+1} = \boldsymbol{D}_{k+1}\boldsymbol{P}_{k+1|k}\boldsymbol{D}_{k+1}^T + \boldsymbol{R}_{k+1}$
    3. $\boldsymbol{W}_{k+1} = \boldsymbol{P}_{k+1|k}\boldsymbol{D}_{k+1}^T\boldsymbol{S}_{k+1}^{-1}$
    4. $\boldsymbol{P}_{k+1|k+1} = \boldsymbol{P}_{k+1|k} - \boldsymbol{W}_{k+1}\boldsymbol{S}_{k+1}\boldsymbol{W}_{k+1}^T$
- However, unlike state, with landmarks we may wish to add new ones during the course of estimation
  - Suppose we want to add new variables to $\boldsymbol{\xi}, \boldsymbol{P}$; we do this at the a priori stage
  - The new landmark state is $\boldsymbol{\xi}_{k|k}^{\text{new}} = \begin{bmatrix} \boldsymbol{\xi}_{k|k} \\ \boldsymbol{\xi}_{k|k}^{(m+1)} \end{bmatrix}$
  - To kick start the state: $\boldsymbol{\xi}_{k|k}^{(m+1)} = \boldsymbol{\gamma}^{(m+1)}(\boldsymbol{x}_k, \boldsymbol{\zeta}_k^{(m+1)})$, where $\boldsymbol{\gamma}$ is the inverse of $\boldsymbol{\eta}$, so we initialize

the new state by inverting the new measurement

  - For $\boldsymbol{P}$, we have $\boldsymbol{P}_{k|k}^{\text{new}} = \begin{bmatrix} \boldsymbol{P}_{k|k} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{P}_{k|k}^{(m+1)} \end{bmatrix}$
  - Kick start with $\boldsymbol{P}_{k|k}^{(m+1)} = \boldsymbol{G}_k^{(m+1)} \boldsymbol{\Sigma}_\zeta^{(m+1)} \boldsymbol{G}_k^{(m+1)^T} + \boldsymbol{R}_k^{(m+1)}$ where $\boldsymbol{G}^{(m+1)} = \dfrac{\partial \boldsymbol{\gamma}^{(m+1)}}{\partial \boldsymbol{\zeta}^{(m+1)^T}}$ and $\boldsymbol{R} = \text{cov}(\boldsymbol{\varpi}, \boldsymbol{\varpi})$

Simultaneous Localization and Mapping (SLAM)

- If we need to localize and map at the same time, we need to estimate the robot pose and landmark position simultaneously
- Our overall state just becomes the combination of the robot state $\boldsymbol{x}$ and map $\boldsymbol{\xi}$
- $\begin{bmatrix} \boldsymbol{x}_{k+1} \\ \boldsymbol{\xi}_{k+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_k & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_k \\ \boldsymbol{\xi}_k \end{bmatrix} + \begin{bmatrix} \boldsymbol{B}_k \\ \boldsymbol{0} \\ \boldsymbol{u}_k \end{bmatrix} + \begin{bmatrix} \boldsymbol{v}_k \\ \boldsymbol{0} \end{bmatrix}$
- $\begin{bmatrix} \boldsymbol{z}_k \\ \zeta_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{\xi}_k) \\ \boldsymbol{\eta}(\boldsymbol{x}_k, \boldsymbol{\xi}_k) \end{bmatrix} + \begin{bmatrix} \boldsymbol{w}_k \\ \boldsymbol{\varpi}_k \end{bmatrix}$
- Many SLAM approaches are available, but this is the essence of SLAM

Lecture 12, Oct 17, 2023

Kalman Filter (Discretization) Example

- Consider a system modelled by $\dfrac{\mathrm{d}v}{\mathrm{d}t} = \dfrac{u}{m}$, where the state is $\boldsymbol{x} = \begin{bmatrix} x \\ v \end{bmatrix}$
- First we need to discretize the system and bring it into standard form $\boldsymbol{x}_{k+1} = \boldsymbol{A}_k \boldsymbol{x}_k + \boldsymbol{B}_k \boldsymbol{u}_k + \boldsymbol{s}_k$ and $\boldsymbol{z}_k = \boldsymbol{D}_k \boldsymbol{x}_k + \boldsymbol{w}_k$
- $\dfrac{x_{k+1} - x_k}{\Delta t} = v_k + r_k, \dfrac{v_{k+1} - v_k}{\Delta t} = \dfrac{u_k}{m} + s_k$ where r_k and s_k are noise terms
  - This gives $x_{k+1} = x_k + v_k \Delta t + r_k \Delta t, v_{k+1} = v_k + \dfrac{\Delta t}{m} u_k + s_k \Delta t$
- Take some timestep Δt, then $\begin{bmatrix} x_{k+1} \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ v_k \end{bmatrix} + \begin{bmatrix} 0 \\ \Delta t/m \end{bmatrix} u_k + \Delta t \boldsymbol{s}_k$ and $z_k = v_k = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ v_k \end{bmatrix} + w_k$
  - Therefore $\boldsymbol{A}_k = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}, \boldsymbol{B}_k = \begin{bmatrix} 0 \\ \Delta t/m \end{bmatrix}, \boldsymbol{D}_k = \begin{bmatrix} 0 & 1 \end{bmatrix}$ (note $\boldsymbol{s}_k = \begin{bmatrix} r_k \\ s_k \end{bmatrix}$)
  - Therefore $\boldsymbol{Q}_k = \Delta t^2 \mathbb{E}[\boldsymbol{s}_k \boldsymbol{s}_k^T]$; note the Δt^2, since the noise is scaled by Δt

Lecture 13, Oct 19, 2023

Bayesian Localization

- Bayesian localization is a localization technique based on probability
  - Kalman filtering is a form of this for Gaussian distributions
- Let $p(x)$ be the probability that the robot is at location x
  - x can represent a number of things, including a point in continuum state space (e.g. pose), discretized state space (e.g. a cell), or some descriptive location (e.g. a room in a building)
  - The first two are examples of ordered sets, which Kalman filters can do; the last is an unordered set, which we can do using a Bayesian Filter
- We will have a probability distribution described by $p(x)$; for ordered sets we can take the mean or media, for unordered sets we can take the mode
- Recall that for conditional probability, $p(x|z) = \sum_{\forall y} p(x|y,z)p(y|z)$

- We will be making heavy use of Bayes' rule, $p(x|y) = \dfrac{p(y|x)p(x)}{p(y)}$

- For localization, we will assume the Markov property, $p(x_{k+1}|x_k, x_{k-1}, \ldots, x_0) = p(x_{k+1}|x_k)$, i.e. the probability of being in a state depends only on the previous state (and input) and not any of the states prior to that
- Let $z_{0:k} = z_0, z_1, \ldots, z_k$ be a sequence of measurements up to and including time step k; the prediction of state x_{k+1} given measurements $z_{0:k}$ is denoted $p(x_{k+1}|z_{0:k})$; the control inputs are υ_k, u_k
- Start by predicting the state probabilities at $k+1$ given the state at time k
  - $p(x_{k+1}|z_{0:k}) = \displaystyle\sum_{\upsilon_k \in \Upsilon} p(x_{k+1}|\upsilon_k, z_{0:k})p(\upsilon_k|z_{0:k}) = p(x_{k+1}|\upsilon u_k, z_{0:k})$
  - If we assume we can deliver our desired control with certainty, $p(\upsilon_k|z_{0:k})$ is only 1 when $\upsilon_k = u_k$ and zero elsewhere, which is why we can get rid of the sum
- $p(x_{k+1}|z_{0:k}) = p(x_{k+1}|u_k, z_{0:k}) = \displaystyle\sum_{x_k \in \Lambda} p(x_{k+1}|x_k, u_k, z_{0:k})p(x_k|z_{0:k})$
  - This considers all possible positions in the previous state, where Λ is the entire state space
  - Assume $p(x_{k+1}|x_k, u_k, z_{0:k}) = p(x_{k+1}|x_k, u_k)$, that is, what the robot is doing is independent of the measurements
  - $p(x_{k+1}|x_k, u_k)$ is just our state model that describes x_{k+1} in terms of x_k and u_k
- The *a priori* state estimate is given by $p(x_{k+1}|z_{0:k}) = \displaystyle\sum_{x_k \in \Lambda} p(x_{k+1}|x_k, u_k)p(x_k|z_{0:k})$
  - By Bayes' rule, $p(x_{k+1}|z_{0:k+1}) = p(x_{k+1}|z_{0:k}, z_{k+1}) = \dfrac{p(z_{k+1}|x_{k+1}, z_{0:k})p(x_{k+1}|z_{0:k})}{p(z_{k+1}|z_{0:k})}$
  - Assume $p(z_{k+1}|x_{k+1}, z_{0:k}) = p(z_{k+1}|x_{k+1})$, i.e. the measurement has no dependence on previous measurements
    - This is our measurement model expressed probabilistically
- The *a posteriori* estimate is then $p(x_{k+1}|z_{0:k+1}) = \dfrac{p(z_{k+1}|x_{k+1})p(x_{k+1}|z_{0:k})}{p(z_{k+1}|z_{0:k})}$
  - The denominator is a normalization factor
- Therefore:
  - State prediction: $p(x_{k+1}|z_{0:k}) = \displaystyle\sum_{x_k \in \Lambda} p(x_{k+1}|x_k, u_k)p(x_k|z_{0:k})$
    - i.e. we take the state distribution we currently have, and we use the state prediction model to see what that distribution transforms into
  - State update: $p(x_{k+1}|z_{0:k+1}) = \dfrac{p(z_{k+1}|x_{k+1})p(x_{k+1}|z_{0:k})}{\sum_{\xi_{k+1} \in \Lambda} p(z_{k+1}|\xi_{k+1})p(\xi_{k+1}|z_{0:k})}$
    - i.e. we take the predicted state distribution, and use the measurement model to see how likely each of the predicted states would yield the measurement that we got
  - Unlike Kalman filtering, now we get the entire probability distribution of the state instead of just the mean; however now we need to consider the entire possible state space

Bayes	Kalman
$p(x_{k+1}\|x_k, u_k)$	$\mathbf{x}_{k+1} = \mathbf{A}_k\mathbf{x}_k + \mathbf{B}_k\mathbf{u}_k + \mathbf{v}_k$
$p(z_k\|x_k)$	$\mathbf{z}_k = \mathbf{D}_k\mathbf{x}_k + \mathbf{w}_k$
$p(x_{k+1}\|z_{0:k})$	$\hat{\mathbf{x}}_{k+1\|k}$
$p(x_{k+1}\|z_{0:k+1})$	$\hat{\mathbf{x}}_{k+1\|k+1}$

Figure 15: Comparison of Bayesian and Kalman filtering.

Particle Filtering

- Bayesian localization requires us to update all possible states at the same time; what if state space was continuous, or really large?

- The summations would become integrals for continuous probability distributions, but this is hard to compute
- Instead of treating the probabilities as continuous, we can instead use sampling
  - This is referred to as *particle filtering* or *Monte Carlo filtering*
  - We draw a set of discrete points $\Lambda_k = \left\{ \boldsymbol{x}_k^{[1]}, \boldsymbol{x}_k^{[2]}, \dots, \boldsymbol{x}_k^{[p]} \right\}$ from $p(\boldsymbol{x}_k)$ to represent the distribution; each of these points is called a *particle*
  - The basic idea is to follow each particle as if it describes the robot's pose, and hope that all particles converge on the robot's true pose
  - The pose at any given time can be estimated as $\hat{\boldsymbol{x}}_k = \sum_{i=1}^{p} w_k^{[i]} \boldsymbol{x}_k^{[i]}$
  - Now the question is how to calculate the weights
- Particle filter procedure:
  - At each time k, draw a set of p particles Λ_k from $p(\boldsymbol{x}_k)$
    * If we know the initial location, we can sample the particles around it, otherwise can choose to evenly distribute the particles
  - For each particle calculate the prediction as $p(\boldsymbol{x}_{k+1}^{[i]} | \boldsymbol{z}_{0:k}) = p(\boldsymbol{x}_{k+1}^{[i]} | \boldsymbol{x}_k^{[i]}, \boldsymbol{u}_k) p(\boldsymbol{x}_k^{[i]} | \boldsymbol{z}_{0:k})$
  - Then update the state as $p(\boldsymbol{x}_{k+1}^{[i]} | \boldsymbol{z}_{0:k+1}) = \dfrac{p(\boldsymbol{z}_{k+1} | \boldsymbol{x}_{k+1}^{[i]}) p(\boldsymbol{x}_{k+1}^{[i]} | \boldsymbol{z}_{0:k})}{\sum_{\boldsymbol{\xi}_{k+1}^{[j]} \in \Lambda_{k+1}} p(\boldsymbol{z}_{k+1} | \boldsymbol{\xi}_{k+1}^{[j]}) p(\boldsymbol{\xi}_{k+1}^{[j]} | \boldsymbol{z}_{0:k})}$
  - Now we can estimate the state as $\hat{\boldsymbol{x}}_{k+1} = \sum_{i=1}^{p} w_{k+1}^{[i]} \boldsymbol{x}_{k+1}^{[i]}$, with the weight of each particle being its (normalized) probability
  - Update the probability distribution as $p(\boldsymbol{x}_{k+1} | \boldsymbol{z}_{0:k+1}) = p(\boldsymbol{x}_{k+1}^{[i]} | \boldsymbol{z}_{0:k+1}) \sim \sum_{i=1}^{p} w_{k+1}^{[i]} \phi(\boldsymbol{x}_{k+1} - \boldsymbol{x}_{k+1}^{[i]})$
    * This is combining the distributions of the individual particles
- One advantage of the particle filter is that it works on any probability distribution of states

Lecture 14, Oct 24, 2023

Lecture 15, Oct 31, 2023

Path Planning

- The path planning problem is about determining a path from a start pose to an end pose while being subject to constraints
  - Constraints can be created by obstacles, barriers, proscribed areas, etc
- The *configuration manifold* C is the set of all possible states that the robot can exist in (given the robot's geometric constraints)
  - $C = \left\{ \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \middle| \, x, y \in \mathbb{R}, \theta \in S^1 \right\}$
  - Note S^1 is the set of all points on a circle
- Let Ω as the parts of the configuration manifold occupied by obstacles, barriers, and prohibited areas
- The *free-world manifold* $W = C \backslash \Omega$ is then all the points we are allowed to be
  - Note $A \backslash B = \{ x \mid x \in A, x \notin B \}$
- For manipulators, we can map the geometric workspace constraints to the configuration manifold
- We will examine 3 basic strategies in detail:
  1. Road-map method: identify a set of discrete routes within the free manifold
  2. Cell-decomposition method: discretizing the map and identifying free and occupied cells
  3. Potential-field method: imposes a field of resistance over obstacles, barriers, and prohibited areas that pushes back the robot

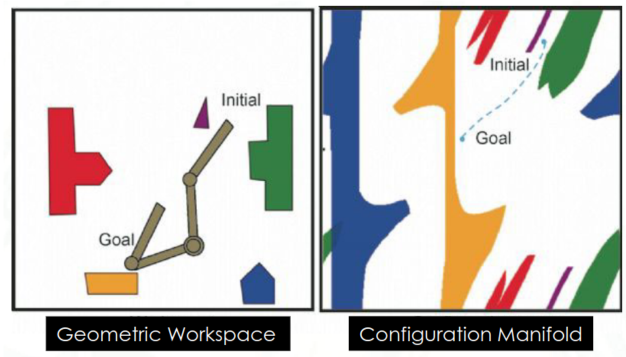

Figure 16: Mapping geometric workspace to configuration manifold.

Road-Map Method

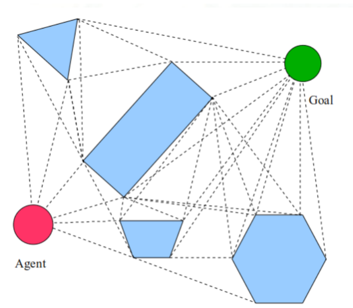

Figure 17: A visibility graph.

- Visibility graph method: We can draw polygons around the obstacles in the configuration manifold and connect vertices with lines that don't cross any polygon
  - Using this set of connecting lines we can find an ideal path between two points
  - However this will slow down in cluttered environments as the number of vertices grow
  - This gives the shortest path, but will come as close as possible to obstacles

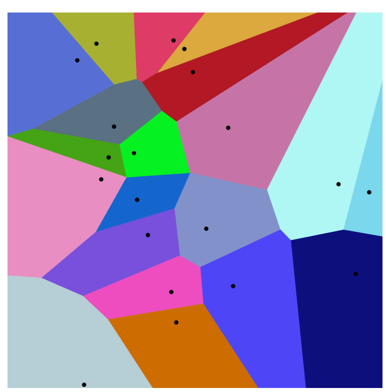

Figure 18: A Voronoi diagram.

- The Voronoi diagram method is the opposite and tries to pick a path that maximizes distance to obstacles
  - We essentially divide the space into regions that are closest to one point
    - Formally the cells are defined as $V_k = \{ x \in S \mid d(x, P_k) \leq d(x, P_j) \forall j \neq k \}$

- We then use the boundaries between cells as possible routes, since this maximizes distance to obstacles
  - To build the diagram, we discretize the map into points,and for each point we compute the distance to the nearest obstacle and check which cell it belongs in
  - If obstacles are points, routes appear as edges of Voronoi cells; if obstacles are polygons, routes consist of straight and parabolic lines
  - The routes we get are usually far from shortest
  - Moving as far as possible from obstacles makes it difficult to localize if sensor ranges are short (e.g. sonar)
- Dijkstra's algorithm can be used once we discretize the space
  - Algorithm:
    1. Assign each node a tentative distance (infinite for undiscovered nodes)
    2. When visiting each node, calculate distance to all neighbouring nodes and update the their distance if it's shorter
    3. Choose the unvisited node with the shortest distance as the next node
  - Given V vertices and E edges, the complexity is $O(E + V \log V)$
  - This is guaranteed to find the shortest path
- To improve the search time we can use the A* algorithm, using a heuristic to direct the search towards the goal
  - Define the cost function $F(k) = G(k) + H(k)$ where $G(k)$ is the actual cost to the node and $H(k)$ is the heuristic estimate
  - The heuristic can be e.g. straight-line Euclidean distance to target
  - We want to ensure the heuristic never overestimates the actual cost, otherwise the algorithm will waste time
- Another way is to use Rapidly Expanding Random Trees (RRTs)
  - These were developed to deal particularly with high-dimensional planning problems
  - The solution space is explored by generating an expanding tree from the initial point towards the goal point; a kind of discretization is performed
  - Repeat until goal is reached:
    1. Begin with an initial point
    2. Select a random point in the free-world space
    3. Find the nearest point on the tree using some metric, e.g. shortest distance
    4. From the nearest point, determine a control input that takes the system towards the direction of the random point
    5. Apply this control input for one step and include the resulting point in the tree
  - The randomized discretization allows us to avoid the grid-like discretization of the graph search methods and generally generate smoother paths

Cell Decomposition

- The general idea is to decompose the configuration space into free areas and occupied areas
  - The world is divided into cells, and then using adjacent free cells we construct a connectivity graph
  - We use the connectivity graph to find a path from start to finish
  - The final path is assembled by passing through the edge midpoints or following barriers

- Exact cell decomposition decomposes space into exact shapes surrounding obstacles
  - The number of cells is small for sparse environments
  - However the implementation can be quite complex
- Approximate cell decomposition (i.e. *occupancy grid* approach) discretizes the world into a fixed-size grid and determines whether each cell is free
  - The number of cells is much larger but implementation is easier
  - A graph search algorithm can be used to find the path
  - We need a large number of cells to make them fine enough to get good resolution
- Adaptive cell decomposition uses an adaptive cell size; we start with a coarse grid, then occupied cells are decomposed further recursively using smaller cell sizes

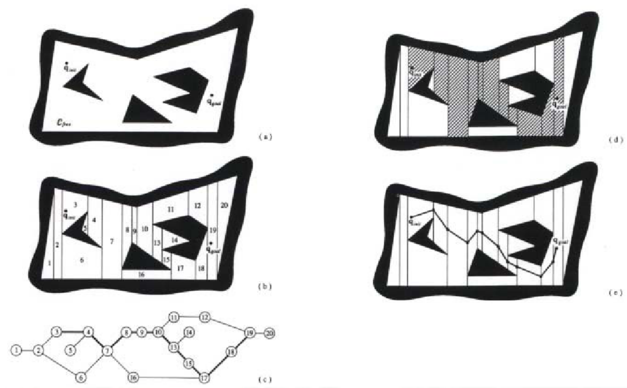

Figure 19: Exact cell decomposition.

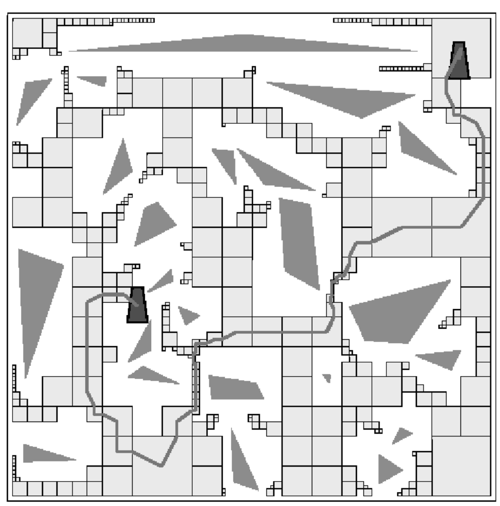

Figure 20: Adaptive cell decomposition.

Potential Field Method

- The robot is treated as a point under the influence of a potential field
  - The goal acts as an attractive force while obstacles act as repulsive forces
  - Robot travels towards the goal like a ball rolling down a hill
- In general $U(\boldsymbol{x}) = \boldsymbol{U}_{\text{goal}}(\boldsymbol{x}) + \sum_i U_{\text{obs},i}(\boldsymbol{x})$
  - $\boldsymbol{U}_{\text{goal}}(\boldsymbol{x})$ has a minimum when we reach the goal
  - $U_{\text{obs},i}(\boldsymbol{x})$ increases when we get closer to the targets
  - We can then find the force as $\boldsymbol{f}(\boldsymbol{x}) = -\vec{\nabla} u(\boldsymbol{x})$
  - Example:
    * Goal potential: $U_{\text{goal}} = \frac{1}{2} k_{\text{goal}}[(x - x_{\text{goal}})^2 + (y - y_{\text{goal}})^2]$
    * Obstacle potentials: $U_{\text{obs},i} = \begin{cases} \frac{1}{2} k_{\text{obs}} \left(\frac{1}{r_i} - \frac{1}{r_0} \right)^2 & r_i(\boldsymbol{x}) \leq r_0 \\ 0 & r_i(\boldsymbol{x}) > r_0 \end{cases}$ where r_0 is some radius of influence
- Now to find the optimal path we can just use gradient descent
- However, we can get stuck in local minima and never reach the goal
  - This can be counteracted by adding a "momentum term"

Lecture 16, Nov 2, 2023

Real-Time Obstacle Avoidance

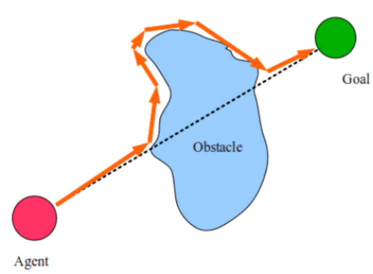

Figure 21: Bug algorithm.

- The bug algorithm is the simplest obstacle avoidance algorithm; the robot just follows the perimeter of the obstacle and resumes on the desired path when possible
  - This assumes we can trace the edge of the obstacle, which is not realistic for non-holonomic robots
  - Only the most recent sensor reading is used so this is very susceptible to noise
- The bubble-band technique creates a bubble of free space around the robot; the bubble is elastic and sensor and obstacle uncertainty can be accounted for by adjusting the bubble
  - This requires a map
- The vector field histogram (VFH) algorithm builds a local probabilistic occupancy grid around the robot, and transforms it into polar coordinates; valleys where obstacle probability is low are identified as potential paths
  - The selected path is usually based on minimization of a path function
  - $G \sim c_1 \Delta\phi_k + c_2 \Delta\theta_k + c_3 \Delta\theta_{k-1}$
  - $\Delta\phi_k$ is the difference between the candidate path direction and the robot's preplanned desired path
  - $\Delta\theta_k$ is the difference in the candidate path direction and current direction
  - $\Delta\theta_k$ is the difference in the candidate path direction and previous directions
- The dynamic window approach (DWA) creates a search space for motion in terms of (linear and angular)

velocities

- Constraints and obstacles are represented as unfeasible areas in the velocity space
- This allows us to take into account nonholonomic constraints
- Like vector field histogram but in velocity space, whereas vector field histogram was about position
- The path is approximated as a circular arc at each instant in time; each arc is defined by (v_k, ω_k) and limited to admissible velocities (i.e. velocities that allow us to stop in time)
- Maximize a cost function, e.g. $G \sim c_1 \frac{1}{\Delta\phi_k} + c_2\Delta d_k + c_3\|v_k, \omega_k\|$ where $\Delta\phi$ is the change fro m the desired path, Δd_k is the distance to obstacles

Navigation and Control Architecture

- How do we design a navigation architecture?
- There are two ways to break down the architecture:
  - Temporal decomposition: distinguish processes having different levels of real-time demands
    * We can consider a hierarchy of processes that require increasing levels of temporal constraints
    * Offline planning (no temporal constraints), strategic decision making (few temporal constraints), quasi-real-time decision making (immediate action), real-time decision making (servo-level control)
  - Control decomposition: distinguish processes having different roles and running at different frequencies
    * Consider a hierarchy of processes that require increasing frequencies
    * Path planner, obstacle avoidance, emergency stop, low-level PID control
- There are 2 major paradigms:
  - Deliberative: traditional sense-plan-act; top-down approach
    * Some algorithm is used to determine the action to take
  - Reactive: parallelized planning with multiple concurrent, independent behaviours; most important action takes precedent; bottom-up approach
    * Action with the highest hierarchy gets executed
    * Subsumption architecture: higher levels of behaviour subsume lower levels

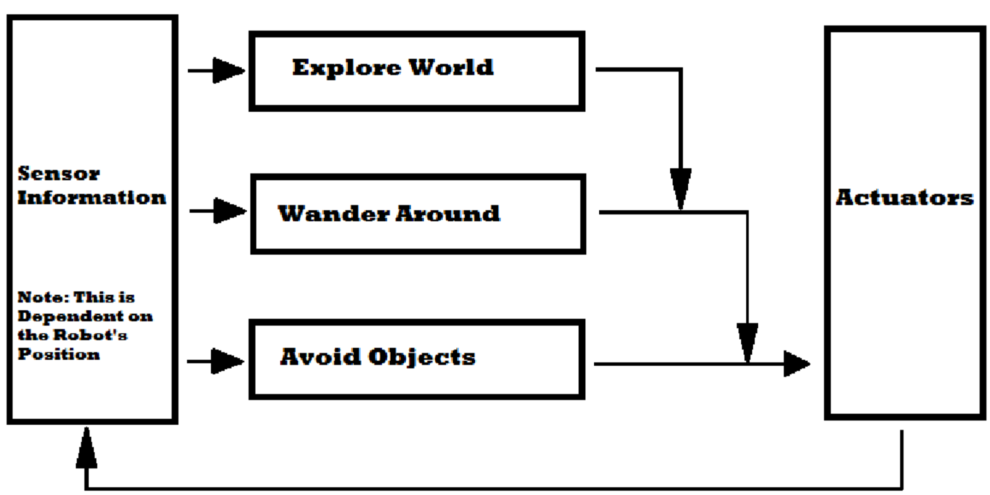

Figure 22: Subsumption architecture is a type of reactive paradigm.

- Subsumption philosophy has 4 principles:
  - Situatedness: “the world is its own best model”
    * The robot interacts with the environment directly without a world model
  - Embodiment: “the world grounds regress”
    * Using real physical systems, not theoretical or simulation models

    * Behaviours are grounded in the real world
  - Intelligence: "intelligence is determined by the dynamics of interaction with the world"
    * Perceptual and mobility skills are necessary for intelligence
  - Emergence: "intelligence is in the eye of the observer"
    * Individual low-level behaviors are not intelligent, but intelligence emerges from interaction of behaviours with each other and the world

Lecture 17, Nov 14, 2023

Manipulators

- Two types of topologies: *open chains*, where there are no closed loops in the system, and *closed chains*
- We consider 2 types of joints: *revolute* (i.e. rotational) and *prismatic* (i.e. translational, extending/contracting)
  - Consider only 1 degree of freedom per joint

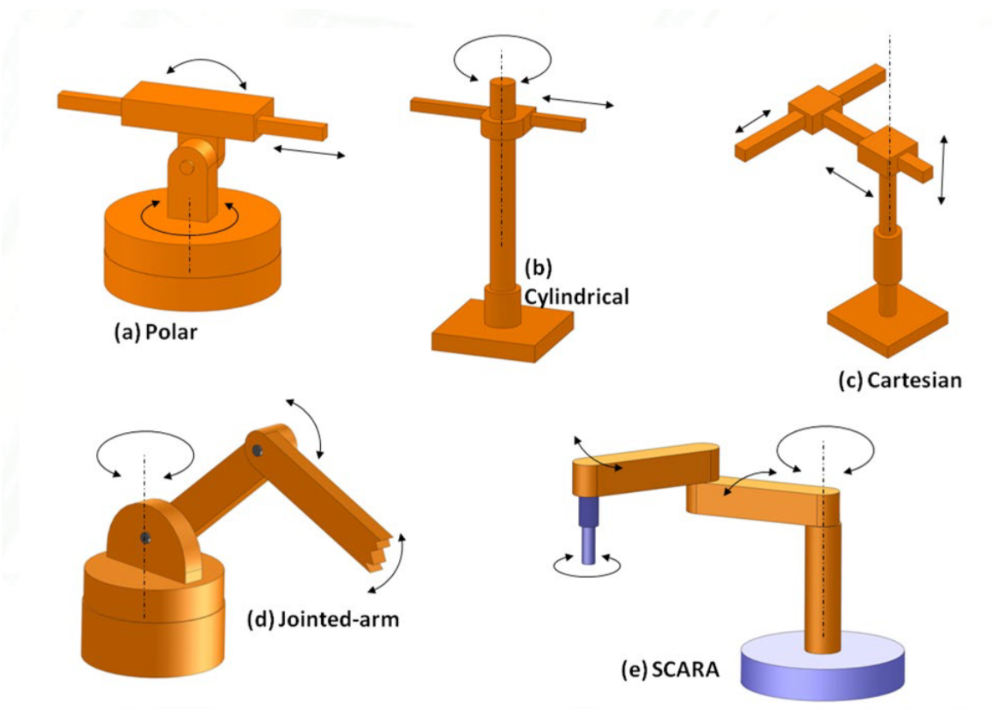

Figure 23: Types of manipulators.

- We are particularly interested in 3-DoF manipulators, because 3 independent degrees of freedom lets us place the end-effector anywhere in 3D translational space
  - Attaching another 3 degrees of freedom via a wrist will get us the rotations as well
- 3-DoF manipulators:
  - Cartesian: PPP (prismatic-prismatic-prismatic)
    * e.g. a 3D printer
    * Each degree of freedom covers a Cartesian coordinate
    * Workspace shape is a cube
  - Revolute/anthropomorphic: RRR (revolute-revolute-revolute)
    * e.g. ABB IRB1400
    * The joints are referred to as body, shoulder, forearm
  - SCARA (Selective Compliant Articulated Robot for Assembly): RRP (revolute-revolute-prismatic)
    * e.g. Epson E2L653S

- Spherical/polar: RRP (revolute-revolute-prismatic)
  * e.g. the Stanford arm
  * Unlike SCARA the second revolute joint is rotated
- Cylindrical: RPP (revolute-prismatic-prismatic)
  * e.g. Seiko RT3300

Manipulator Geometry

- Rotation matrices form the *special orthogonal group*, $SO(3) = \left\{ \boldsymbol{C} \in \mathbb{R}^{3\times 3} \,\middle|\, \boldsymbol{C}^T\boldsymbol{C} = \mathbf{1}, \det \boldsymbol{C} = 1 \right\}$
  - Recall that a group is a set of elements G and a binary operation xy that is closed, associative, has an identity and inverse
  - Commutative groups (aka Abelian groups) have a commutative binary operation ($SO(3)$ is not Abelian)
  - $SO(3)$ is a *Lie group*, i.e. it is differentiable
- Given a point $\underrightarrow{w}$ expressed in $\underrightarrow{\mathcal{F}}_b$ relative to O_b, we may want to express it in $\underrightarrow{\mathcal{F}}_a$ relative to O_a; O_b has position $\underrightarrow{\rho}$ relative to O_a
  - $\underrightarrow{v} = \underrightarrow{w} + \underrightarrow{\rho} \iff \boldsymbol{v}_a = \boldsymbol{C}_{ab}\boldsymbol{w}_b + \boldsymbol{\rho}_a$
  - We can combine this as $\begin{bmatrix} \boldsymbol{v}_a \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{C}_{ab} & \boldsymbol{\rho}_a \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_b \\ 1 \end{bmatrix} \implies \boldsymbol{u}_a = \boldsymbol{T}_{ab}\boldsymbol{u}_b$
    * Note this only works for position vectors
  - $\boldsymbol{T}_{ab}$ is a 4×4 matrix that generalizes rotations
  - $\boldsymbol{T}$ forms the *special Euclidean group* $SE(3) = \left\{ \boldsymbol{T} = \begin{bmatrix} \boldsymbol{C} & \boldsymbol{\rho} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \,\middle|\, \boldsymbol{C}^T\boldsymbol{C} = \mathbf{1}, \det \boldsymbol{C} = 1 \right\}$
    * This is also a Lie group but not a commutative group
  - Note $\boldsymbol{T}^{-1} = \begin{bmatrix} \boldsymbol{C}^T & -\boldsymbol{C}^T\boldsymbol{\rho} \\ \mathbf{0}^T & 1 \end{bmatrix}$, and the identity of $SE(3)$ is $\mathbf{1}_{4\times 4}$
  - In $SE(3)$, $\dot{\boldsymbol{T}}_{ab} = -\boldsymbol{\Omega}_a^{ab}\boldsymbol{T}_{ab}$ where $\boldsymbol{\Omega}_a^{ab} = \begin{bmatrix} \boldsymbol{\omega}_a^{ab\times} & \boldsymbol{v}_a^{ab} \\ \mathbf{0}^T & 0 \end{bmatrix}$
    * This is a generalized form of Poisson's kinematical equation

Denavit-Hartenberg Parameters

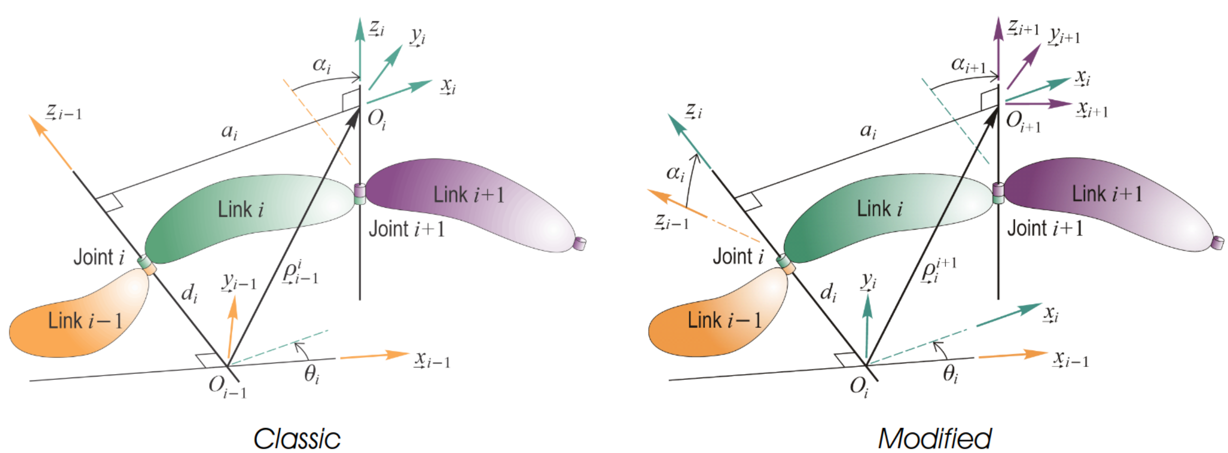

Figure 24: Denavit-Hartenberg Parameters.

- We can describe any series link manipulator with revolute and prismatic joints using *Denavit-Hartenberg parameters*
  - The DH parameters consist of 4 parameters per joint:
    1. Link length (a_i)

    * This is the length of a line segment normal to and joining $\underrightarrow{z}_i, \underrightarrow{z}_{i+1}$ (direction $\underrightarrow{x}_i = \underrightarrow{z}_i \times \underrightarrow{z}_{i+1}$)
        - This could be longer than the actual physical length of the link due to the orientation of axes
    * The $\underrightarrow{z}_i$ are the axes of each joint – axis of rotation for revolute joints, axis of translation for prismatic joints
    * The intersection of this line and the link axes are the reference points O_i
        - Note if $\underrightarrow{z}_i$ and $\underrightarrow{z}_{i-1}$ are parallel, this reference point can be anywhere
        - Note O_i is not fixed with respect to link i, but link $i-1$ instead (for a prismatic joint, O_i can shift)
  2. Link twist (α_i)
    * This is the angle between $\underrightarrow{z}_{i-1}$ and $\underrightarrow{z}_i$
  3. Link offset (d_i)
    * This is the distance along $\underrightarrow{z}_i$ from O_i to the intersection of $\underrightarrow{x}_i, \underrightarrow{z}_i$
    * This is a variable if the joint is prismatic, fixed if the joint is revolute
  4. Joint angle (θ_i)
    * This is the angle between $\underrightarrow{x}_i$ and $\underrightarrow{x}_{i-1}$
    * This is a variable if the joint is revolute, fixed if the joint is prismatic
- Note that this is referred to as the *modified* DH parameters

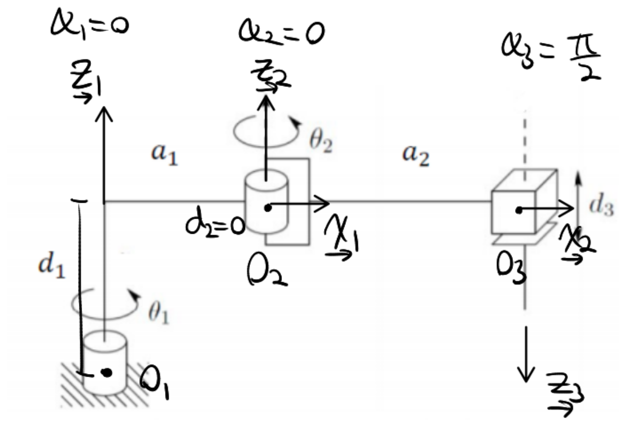

Figure 25: Example: SCARA manipulator DH parameters.

- The relative position of O_{i+1} from O_i is $\rho_i^{i+1} = d_i \underrightarrow{z}_i + a_i \underrightarrow{x}_i$
  - In $\underrightarrow{F}_i$ we have $\underrightarrow{\rho}_i^{i+1} = \begin{bmatrix} a_i \\ 0 \\ d_i \end{bmatrix}$
- Consider an arbitrary point P with position $\underrightarrow{v}_i$ relative to O_i; then $\underrightarrow{v}_i = \underrightarrow{v}_{i+1} + \underrightarrow{\rho}_i^{i+1} = \underrightarrow{v}_{i+1} + d_i \underrightarrow{z}_i + a_i \underrightarrow{x}_i$
  - The rotation matrix from $\underrightarrow{\mathcal{F}}_{i-1}$ to $\underrightarrow{\mathcal{F}}_i$ is $\boldsymbol{C}_{i,i-1} = \boldsymbol{C}_3(\theta_i)\boldsymbol{C}_1(\alpha_i)$ (first rotate about $\underrightarrow{x}_{i-1}$, then rotate about $\underrightarrow{z}_i$)
  - Therefore $\boldsymbol{T}_{i,i+1} = \begin{bmatrix} \boldsymbol{C}_{i,i+1} & d_i \mathbf{1}_3 + a_i \mathbf{1}_1 \\ \mathbf{0}^T & 1 \end{bmatrix}$
  - Expanded out: $\boldsymbol{T}_{i,i+1} = \begin{bmatrix} \cos(\theta_{i+1}) & -\sin(\theta_{i+1}) & 0 & a_i \\ \sin(\theta_{i+1})\cos(\alpha_{i+1}) & \cos(\theta_{i+1}\cos(\alpha_{i+1}) & -\sin(\alpha_{i+1}) & 0 \\ \sin(\theta_{i+1})\sin(\alpha_{i+1}) & \cos(\theta_{i+1})\sin(\alpha_{i+1}) & \cos(\alpha_{i+1}) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Lecture 18, Nov 16, 2023

Manipulator Jacobians

Velocity

- Each joint gives us one degree of freedom $q_i = \begin{cases} \theta_i & \text{joint is revolute} \\ d_i & \text{joint is prismatic} \end{cases}$
- We want to know how, given a desired velocity of the end-effector, we can set the joint rates to achieve that velocity
- The manipulator Jacobian relates the end-effector velocity and angular velocity: $\boldsymbol{v} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$
  - $\boldsymbol{v} = \begin{bmatrix} \boldsymbol{v}_0^{ee} \\ \boldsymbol{\omega}_0^{ee} \end{bmatrix}$ is the velocity (including translational and angular) velocity of the end-effector
    * Note that this is expressed in frame 0, which is our world/inertial frame
  - $\dot{\boldsymbol{q}}$ are the joint rates
  - $\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{m \times n}$ where $m \leq 6$ and n is the number of joints; in general it is a function of the current joint states
- Partition the Jacobian as $\boldsymbol{J}(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{J}^{(v)}(\boldsymbol{q}) \\ \boldsymbol{J}^{(\omega)}(\boldsymbol{q}) \end{bmatrix}$, where one part is for linear velocity and the other part is for angular
  - Given an expression for the end-effector position we can simply differentiate it to get the translational velocity Jacobian
  - $\boldsymbol{J}^{(v)}(\boldsymbol{q}) = \dfrac{\partial \boldsymbol{r}_0^{ee}}{\partial \boldsymbol{q}^T}$ where $\boldsymbol{r}_0^{ee}$ is the position of the end-effector
  - Angular velocity however is more complicated since it's not the direct derivatives of the orientation variables
- For angular velocity $(\boldsymbol{\omega}_0^{ee})^\times \boldsymbol{C}_{0,n}\dot{\boldsymbol{C}}_{0,n}^T = \sum \boldsymbol{C}_{0,n}\dfrac{\partial \boldsymbol{C}_{0,n}^T}{\partial q_i}\dot{q}_i \equiv \sum_i (\boldsymbol{\nu}_i^{ee})^\times \dot{q}_i$
  - $\boldsymbol{C}_{0,i}$ is the rotation matrix from frame i to the world frame
  - Therefore $\boldsymbol{\omega}_0^{ee} = \sum_i \boldsymbol{\nu}_i^{ee}\dot{q}_i$ and so $\boldsymbol{J}^{(\omega)} = \begin{bmatrix} \boldsymbol{\nu}_1^{ee} & \cdots & \boldsymbol{\nu}_n^{ee} \end{bmatrix}$
- Using DH parameters:
  - Let $\underset{\rightarrow}{\rho}_i^j = \sum_{k=i}^{j-1} \underset{\rightarrow}{\rho}_k^{k+1}$ be the relative position of O_j from O_i
  - Let $\underset{\rightarrow}{\omega}_i^j = \sum_{k=i}^{j-1} \underset{\rightarrow}{\omega}_k^{k+1}$ be the angular velocity of link i with respect to link j
  - Let $\boldsymbol{C}_{ij} = \prod_{k=i}^{j-1} \boldsymbol{C}_{k,k+1}$ be the rotation matrix from frame j to frame i
- Then $\underset{\rightarrow}{v}^{ee} - \underset{\rightarrow}{\rho}_0^{n+1^{\bullet}}$ and $\underset{\rightarrow}{\omega}^{ee} = \underset{\rightarrow}{\omega}_0^n$
  - Note the velocity is to $n+1$ because we want the velocity of the end-effector (i.e. end of the last link), but the angular velocity is of the last link so it's to n
  - Note $\underset{\rightarrow}{\rho}_i^{i+1} = \underset{\rightarrow}{\boldsymbol{\mathcal{F}}}_i^T \boldsymbol{\rho}_i^{i+1}, \underset{\rightarrow}{\omega}_{i-1}^i = \underset{\rightarrow}{\boldsymbol{\mathcal{F}}}_i^T \omega_{i-1}^i$, i.e. $\boldsymbol{\rho}_i^{i+1}$ and $\boldsymbol{\omega}_{i-1}^i$ are both expressed in frame i
- For the angular velocity part:
  - $\underset{\rightarrow}{\omega}_{i-1}^i = \begin{cases} \dot{\theta}_i \underset{\rightarrow}{z}_i & \text{revolute joint} \\ \underset{\rightarrow}{0} & \text{prismatic joint} \end{cases}$
  - $\underset{\rightarrow}{\omega}^{ee} = \sum_{i=1}^n \varepsilon_i \dot{\theta} \underset{\rightarrow}{z}_i$
    * Note ε_i is 1 if the joint is revolute, otherwise 0
  - $\underset{\rightarrow}{z}_i = \underset{\rightarrow}{\boldsymbol{\mathcal{F}}}_i^T \boldsymbol{1}_3 \implies \boldsymbol{\omega}_0^{ee} = \sum_{i=1}^n \varepsilon_i \boldsymbol{C}_{0,i}\boldsymbol{1}_3\dot{\theta}_i$

    - The Jacobian is then $\boldsymbol{J}^{(\omega)} = \begin{bmatrix} \boldsymbol{j}_1^{(\omega)} & \cdots & \boldsymbol{j}_n^{(\omega)} \end{bmatrix}$ where $\boldsymbol{j}_i^{(\omega)} = \varepsilon_i \boldsymbol{C}_{0,i}\mathbf{1}_3$
- For the translational velocity part:
    - $\underrightarrow{v}^{ee} = {\underrightarrow{\rho}_0^{n+1}}^{\bullet} = \sum_{i=0}^{n} {\underrightarrow{\rho}_i^{i+1}}^{\bullet}$
    - ${\underrightarrow{\rho}_i^{i+1}}^{\bullet} = {\underrightarrow{\rho}_i^{i+1}}^{\circ} + \underrightarrow{\omega}_0^i \times \underrightarrow{\rho}_i^{i+1}$
        - Recall $\underrightarrow{\rho}_i^{i+1} = d_i \underrightarrow{z}_i + a_i \underrightarrow{x}_i \implies {\underrightarrow{\rho}_i^{i+1}}^{\circ} = (1-\varepsilon_i)\dot{d}_i \underrightarrow{z}_i + d_i \underrightarrow{z}_i^{\circ} + a_i \underrightarrow{x}_i^{\circ}$
        - But $\underrightarrow{x}_i^{\circ} = \underrightarrow{z}_i^{\circ} = \underrightarrow{0}$
    - Therefore ${\underrightarrow{\rho}_i^{i+1}}^{\bullet} = (1-\varepsilon_i)\dot{d}_i \underrightarrow{z}_i + \underrightarrow{\omega}_0^i \times \underrightarrow{\rho}_i^{i+1}$
    - Substitute $\underrightarrow{\omega}_0^i = \sum_{k=1}^{i} \varepsilon_k \dot{\theta}_k \underrightarrow{z}_k$
    - So $\underrightarrow{v}^{ee} = \sum_{i=1}^{n} \left[(1-\varepsilon_i)\dot{d}_i \underrightarrow{z}_i + \sum_{k=1}^{i} \varepsilon_k \dot{\theta}_k \underrightarrow{z}_k \times \underrightarrow{\rho}_i^{i+1} \right]$
    - This reduces to $\underrightarrow{v}^{ee} = \sum_{i=1}^{n} \left[(1-\varepsilon_i)\dot{d}_i \underrightarrow{z}_i + \varepsilon_i \dot{\theta}_i \underrightarrow{z}_i \times \underrightarrow{\rho}_i^{n+1} \right]$
    - In the world frame, $\boldsymbol{v}_0^{ee} = \sum_{i=1}^{n} \left[(1-\varepsilon_i)\dot{d}_i \boldsymbol{C}_{0,i}\mathbf{1}_3 + \varepsilon_i \dot{\theta}_i \boldsymbol{C}_{0,i}\mathbf{1}_3^{\times} \boldsymbol{\rho}_i^{n+1} \right]$
    - Therefore $\boldsymbol{J}^{(v)} = \begin{bmatrix} \boldsymbol{j}_1^{(v)} & \cdots & \boldsymbol{j}_n^{(v)} \end{bmatrix}$ where $\boldsymbol{j}_i^{(v)} = (1-\varepsilon_i)\boldsymbol{C}_{0,i}\mathbf{1}_3 + \varepsilon_i \boldsymbol{C}_{0,1}\mathbf{1}_3^{\times} \boldsymbol{\rho}_i^{n+1}$
- $\boldsymbol{J} = \begin{bmatrix} \boldsymbol{j}_1 & \cdots & \boldsymbol{j}_n \end{bmatrix}$ where $\boldsymbol{j}_i = \begin{bmatrix} \boldsymbol{j}_i^{(v)} \\ \boldsymbol{j}_i^{(\omega)} \end{bmatrix} = \begin{cases} \begin{bmatrix} \boldsymbol{C}_{0,i}\mathbf{1}_3^{\times} \boldsymbol{\rho}_i^{n+1} \\ \boldsymbol{C}_{0,i}\mathbf{1}_3 \end{bmatrix} & \text{revolute joint} \\ \begin{bmatrix} \boldsymbol{C}_{0,i}\mathbf{1}_3 \\ \mathbf{0} \end{bmatrix} & \text{prismatic joint} \end{cases}$

Force

- Define the joint control force/torque as $\underrightarrow{\eta}_{i-1}^i = \eta_i \underrightarrow{z}_i = \begin{cases} \tau_i \underrightarrow{z}_i & \text{revolute joint} \\ f_i \underrightarrow{z}_i & \text{prismatic joint} \end{cases}$
    - This force or torque is between links $i-1$ and i
- We can obtain the actual control input force by taking the dot product of the joint forces with $\underrightarrow{z}_i$, since only 1 of 6 degrees of freedom of force is due to the input and the other are due to constraints
- How do we relate the control input force to the force delivered at the end-effector?
- Consider a free-body segment between links i and n; assume this is static (i.e. ignoring inertial forces)
    - The interlink force and torque are $\underrightarrow{\tau}_{i-1}^i = \underrightarrow{\tau}^{ee} + \underrightarrow{\rho}_i^{n+1} \times \underrightarrow{f}^{ee}$ and $\underrightarrow{f}_{i-1}^i = \underrightarrow{f}^{ee}$, derived from the FBD
- The control inputs are therefore $\eta_i = \begin{cases} \tau_i = \underrightarrow{z}_i \cdot \underrightarrow{\tau}^{ee} + \underrightarrow{z}_i \cdot \rho_i^{n+1} \times \underrightarrow{f}^{ee} & \text{revolute joint} \\ f_i = \underrightarrow{z}_i \cdot \underrightarrow{f}^{ee} & \text{prismatic joint} \end{cases}$
    - Expressed in world frame: $\eta_i = \begin{cases} \tau_i = (\boldsymbol{C}_{0,i}\mathbf{1}_3)^T \boldsymbol{\tau}_0^{ee} + (\boldsymbol{C}_{0,i}\mathbf{1}_3^{\times} \boldsymbol{\rho}_i^{n+1})^T \boldsymbol{f}_0^{ee} & \text{revolute joint} \\ f_i = (\boldsymbol{C}_{0,i}\mathbf{1}_3)^T \boldsymbol{f}_0^{ee} & \text{prismatic joint} \end{cases}$
- This gives $\boldsymbol{\eta} = \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{f}$ where $\boldsymbol{f} = \begin{bmatrix} \boldsymbol{f}_0^{ee} \\ \boldsymbol{\tau}_0^{ee} \end{bmatrix}$, where the Jacobian is the same as before

Acceleration

- $\boldsymbol{a} = \dot{\boldsymbol{v}} = \begin{bmatrix} \dot{\boldsymbol{v}}_0^{ee} \\ \dot{\boldsymbol{\omega}}_0^{ee} \end{bmatrix}$
- So $\boldsymbol{a} = \boldsymbol{J}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}(\boldsymbol{q})\dot{\boldsymbol{q}}$
- We can write $\dot{\boldsymbol{J}(\boldsymbol{q})}\dot{\boldsymbol{q}} = \text{col}\left[\sum_{j=1}^{n}\sum_{k=1}^{n} \frac{\partial J_{ik}}{\partial q_j}\dot{q}_j \dot{q}_k \right]$

Kinematics

- Forward kinematics is finding $\boldsymbol{v}$ given $\dot{\boldsymbol{q}}$ and $\boldsymbol{q}$; this is easy if we have the Jacobian
- Inverse kinematics is the problem of finding $\dot{\boldsymbol{q}}$ given $\boldsymbol{v}$ (and integrating for $\boldsymbol{q}$); in general this is much more challenging
- If the Jacobian is square and invertible, then we can simply find $\dot{\boldsymbol{q}} = \boldsymbol{J}^{-1}(\boldsymbol{q})\boldsymbol{v}$
- If it is not invertible, assuming $m < n$ (i.e. we have more joints/DoF than spacial dimensions), we can try using the *pseudoinverse*
  - Provided rank $\boldsymbol{J} = m$, $\boldsymbol{J}\boldsymbol{J}^T$ is invertible
  - Define the *(Moore-Penrose) pseudoinverse* $\boldsymbol{J}^\dagger = \boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}$, so $\boldsymbol{J}\boldsymbol{J}^\dagger = \mathbf{1} \in \mathbb{R}^{m\times m}$
- Then in general if rank $\boldsymbol{J} = m$, $\dot{\boldsymbol{q}} = \boldsymbol{J}^\dagger\boldsymbol{v} + (1 - \boldsymbol{J}^\dagger\boldsymbol{J})\boldsymbol{b}$, for any $\boldsymbol{b} \in \mathbb{R}^n$ (i.e. we have an infinite number of solutions)
  - Note that $(\mathbf{1} - \boldsymbol{J}^\dagger\boldsymbol{J})\boldsymbol{b} \in \ker \boldsymbol{J}$
  - Take $\boldsymbol{b} = \mathbf{0}$ if we want to minimize the joint rates
- What about square but non-invertible $\boldsymbol{J}$ or rank $\boldsymbol{J} < m$?
  - In this case we have a *singularity* – we cannot solve for $\dot{\boldsymbol{q}}$ given an arbitrary $\boldsymbol{v}$
- At singularities, configurations with motion in certain directions may be unattainable
  - These often occur at boundaries of the workspace
  - Finite end-effector rates might imply infinite joint rates
  - Finite joint forces/torques might imply infinite end-effector forces and torques

Definition

A *singularity* occurs at $\boldsymbol{q}$ when rank $\boldsymbol{J}(\boldsymbol{q}) < m$, or equivalently $\det(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})^T) = 0$, where m is the dimension of the workspace.

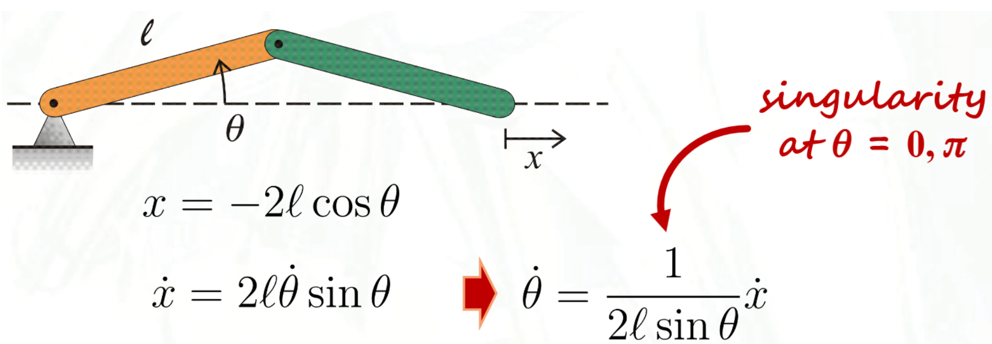

Figure 26: Simple singularity example.

- Singularities for the above arm occur at $\theta_3 = 0, \pi$ or $\theta_3 = -2\theta_2$
  - At $\theta_3 = -2\theta_2$, the end-effector will be on the z axis, so any θ_1 gives us the same end-effector position; therefore we can't solve for θ_1
  - At $\theta_3 = 0$, the links are in a straight line, so we can't get any motion along that line
  - At $\theta_3 = \pi$, the arm is folded back on itself, so again we can't get any motion on that line
- Translation and rotation of an end-effector can be theoretically uncoupled if we have a *wrist-partitioned arm*, if:
  - Last 3 joints are revolute with axes passing through the common centre E
  - Successive axes are not parallel
  - E can be placed arbitrarily in position space
- Practically however the end-effector is always displaced from E

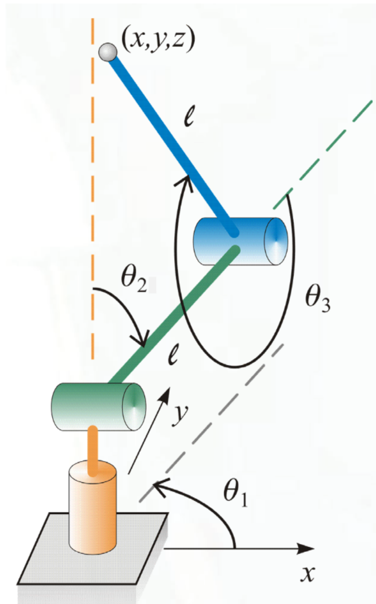

Figure 27: Singularity example.

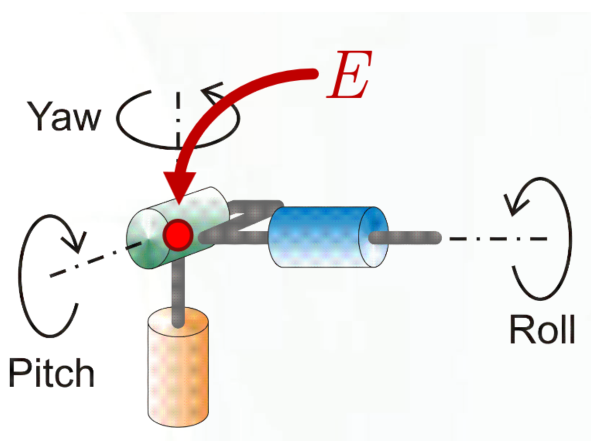

Figure 28: A wrist-partitioned arm.

Lecture 19, Nov 21, 2023

Manipulator Examples

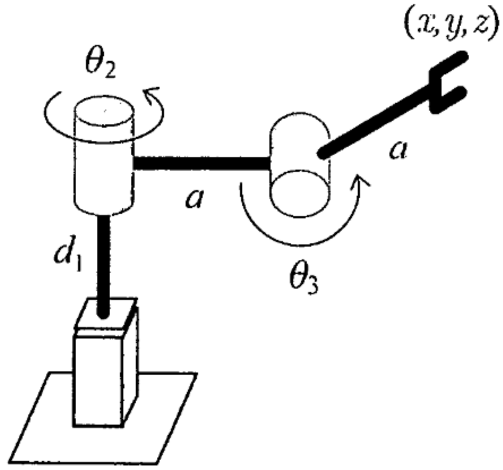

Figure 29: Example question.

- For the PRR manipulator above, with joint variables d_1, θ_2, θ_3 (where θ_3 is measured relative to the second link), determine:
  - Displacement (x, y, z) of the end-effector
    * We can do this by inspection from the geometry
    * $\boldsymbol{r} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a(1+\cos\theta_3)\cos\theta_2 \\ a(1+\cos\theta_3)\sin\theta_2 \\ d_1 + a\sin\theta_3 \end{bmatrix}$
  - The Jacobian $\boldsymbol{J}^{(v)}$, with the translational velocity only
    * Since we have $\boldsymbol{r}(d_1, \theta_2, \theta_3)$ we can directly differentiate to find the Jacobian
    * $\boldsymbol{J}^{(v)} = \begin{bmatrix} 0 & -a(1+\cos\theta_3)\sin\theta_2 & -a\sin\theta_3\cos\theta_2 \\ 0 & a(1+\cos\theta_3)\cos\theta_2 & -a\sin\theta_3\sin\theta_2 \\ 1 & 0 & a\cos\theta_3 \end{bmatrix}$
  - Singularities of the system
    * The singularity condition is $\det(\boldsymbol{J}\boldsymbol{J}^T) = 0$, which for a square $\boldsymbol{J}$ is equivalent to $\det \boldsymbol{J} = 0$
    * Since the first column has only a single 1, the determinant is given by the determinant of the 2x2 matrix at the top right
    * $\det \boldsymbol{J} = a(1+\cos\theta_3)\sin\theta_2 a\sin\theta_3\sin\theta_2 + a\sin\theta_3\cos\theta_2 a(1+\cos\theta_3)\cos\theta_2$
      $= a^2(1+\cos\theta_3)\sin\theta_3\sin^2\theta_2 + a^2(1+\cos\theta_3)\sin\theta_3\cos^2\theta_2$
      $= a^2\sin\theta_3(1+\cos\theta_3)$
    * This gives us $\theta_3 = 0, \pi$
    * Intuitively, at $\theta_3 = 0$, the last 2 links are aligned; this means we need an infinite $\dot{\theta}_3$ to get a finite EE velocity; at $\theta_3 = \pi$ the last links are folded on each other, so any angle of θ_2 results in the same EE position, and we also have the infinite velocity issue; additionally $\dot{d}_1$ results in the same $\dot{z}$ as $\dot{\theta}_3$
      - Notice $\theta_3 = \pi$ is a double root, which corresponds to the two different interpretations
  - The required joint force and torques required to deliver a force f^{ee} applied in the downward direction, when $d_1 = a, \theta_2 = 0, \theta_3 = -45°$
    * Recall: $\boldsymbol{\eta} = \boldsymbol{J}^T\boldsymbol{f}$

* In this configuration, $\boldsymbol{J} = \begin{bmatrix} 0 & 0 & \dfrac{a\sqrt{2}}{2} \\ 0 & a\left(1 + \dfrac{\sqrt{2}}{2}\right) & 0 \\ 1 & 0 & a\dfrac{\sqrt{2}}{2} \end{bmatrix}$
* $\boldsymbol{f} = \begin{bmatrix} 0 \\ 0 \\ -f^{ee} \end{bmatrix}$
* Therefore $\boldsymbol{\eta} = \begin{bmatrix} -f^{ee} \\ 0 \\ -\dfrac{a\sqrt{2}}{2} f^{ee} \end{bmatrix}$

Lecture 20, Nov 23, 2023

Geometry in $SE(3)$

- We can represent the position of the end-effector using $SE(3)$ transformations
  - $\boldsymbol{u}^{ee} = \left(\prod_{i=1}^{n} \boldsymbol{T}_{i-1}\right) \boldsymbol{u}_n^{n+1}$ (note the $\boldsymbol{u}_n^{n+1}$ brings us from the last joint to the end-effector)
  - In matrix form: $\begin{bmatrix} \boldsymbol{r}^{ee} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{C}_{i-1,i} & \boldsymbol{\rho}_{i-1}^{i} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{\rho}_n^{n+1} \\ 1 \end{bmatrix}$
- The orientation of the end-effector is given by $\boldsymbol{C}^{ee} = \prod_{i=1}^{n} \boldsymbol{C}_{i-1,i}$
- We can combine both into the pose: $\boldsymbol{T}^{ee} = \prod_{i=1}^{n+1} \boldsymbol{T}_{i-1,i}$
  - $\boldsymbol{T}^{ee} = \begin{bmatrix} \boldsymbol{C}_{0,n} & \boldsymbol{r}^{ee} \\ \boldsymbol{0}^T & 1 \end{bmatrix}$
  - $\boldsymbol{T}_{n,n+1} = \begin{bmatrix} \boldsymbol{1} & \boldsymbol{\rho}_n^{n+1} \\ \boldsymbol{0}^T & 1 \end{bmatrix}$
    * We added this to bring us from the last joint to the end-effector

Inverse Kinematics

- Technically inverse "geometry"
- In general, $\boldsymbol{r}^{ee} = \boldsymbol{f}_r(\boldsymbol{q}), \boldsymbol{\theta}^{ee} = \boldsymbol{f}_\theta(\boldsymbol{q}) \implies \boldsymbol{p}^{ee} = \boldsymbol{f}(\boldsymbol{q})$ where $\boldsymbol{p}^{ee}$ is the end-effector pose
  - Given $\boldsymbol{q}$, solving for $\boldsymbol{p}^{ee}$ is the problem of *forward kinematics*
  - Given $\boldsymbol{p}^{ee}$, solving for $\boldsymbol{q}$ is the problem of *inverse kinematics*
- Solving inverse kinematics often requires numerical techniques, and often has multiple (possibly infinite) solutions

- For the example above, we can solve for the angles using the cosine law; the $\cos^{-1}$ gives two possible solutions, one for positive θ_2 and one for negative θ_2
- A 6-DoF robotic arm with revolute joints can have as many as 16 solutions depending on the link lengths
- Incremental solution technique: given a solution at $\boldsymbol{q}$ corresponding to $\boldsymbol{p}^{ee}$, what if we changed $\boldsymbol{p}^{ee}$ by a small $\Delta\boldsymbol{p}^{ee}$?
  - $\begin{bmatrix} \boldsymbol{v}^{ee} \\ \boldsymbol{\omega}^{ee} \end{bmatrix} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \implies \begin{bmatrix} \Delta t\boldsymbol{v}^{ee} \\ \Delta t\boldsymbol{\omega}^{ee} \end{bmatrix} = \boldsymbol{J}(\boldsymbol{q})\Delta q$
  - Therefore $\Delta\boldsymbol{p}^{ee} = \begin{bmatrix} \Delta\boldsymbol{r}^{ee} \\ \Delta\boldsymbol{\phi}^{ee} \end{bmatrix} = \boldsymbol{J}(\boldsymbol{q})\Delta\boldsymbol{q}$ (since for small angular displacements only, we can directly multiply by Δt to get $\Delta\boldsymbol{\phi}$)

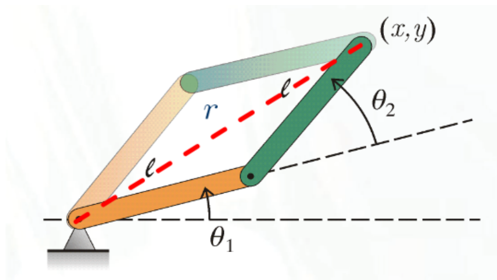

Figure 30: Example of a two-link system where multiple solutions exist.

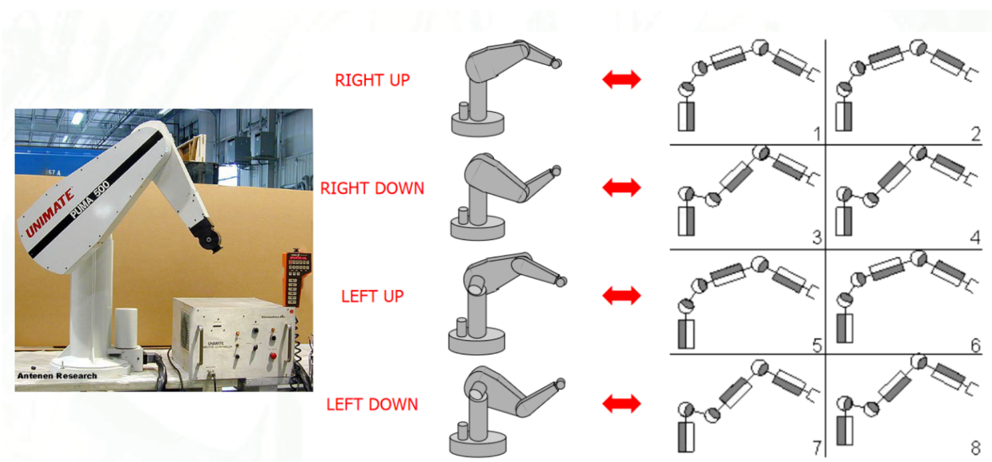

Figure 31: Example of a system with even more solutions.

- Notice that this look exactly like the kinematical relation; we can now use the pseudoinverse to solve for it

- $\Delta\boldsymbol{q} = \boldsymbol{J}^{\dagger}(\boldsymbol{q})\Delta\boldsymbol{p}^{ee} + (\mathbf{1} - \boldsymbol{J}^{\dagger}\boldsymbol{J})\boldsymbol{b}$ where $\boldsymbol{J}^{\dagger} = \boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1}$
    - But this doesn't quite do it because the inverse can be big, even when Δp^{ee} is small
- The *damped least-squares technique* (aka Levenberg-Marquardt method) is a variation on the incremental technique
    - Minimize $\|\Delta\boldsymbol{p}^{ee} - \boldsymbol{J}(\boldsymbol{q})\Delta\boldsymbol{q}\|^2 + \lambda^2\|\Delta\boldsymbol{q}\|^2$
    - λ is a damping term which makes sure that our Δqs are small – this is known as *regularization*
        * If we're talking about a pose, we can use $\boldsymbol{T}$ and use a matrix norm
    - This is equivalent to minimizing $\left\|\begin{bmatrix}\Delta\boldsymbol{p}^{ee} \\ \mathbf{0}\end{bmatrix} - \begin{bmatrix}\boldsymbol{J} \\ \lambda\mathbf{1}\end{bmatrix}\Delta\boldsymbol{q}\right\|$, which is like a linear regression minimizing $\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|$
        * Therefore this is satisfied by $\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b} \implies \begin{bmatrix}\boldsymbol{J} \\ \lambda\mathbf{1}\end{bmatrix}^T\begin{bmatrix}\boldsymbol{J} \\ \lambda\mathbf{1}\end{bmatrix}\Delta\boldsymbol{q} = \begin{bmatrix}\boldsymbol{J} \\ \lambda\mathbf{1}\end{bmatrix}^T\begin{bmatrix}\Delta\boldsymbol{p}^{ee} \\ \mathbf{0}\end{bmatrix}$
    - This reduces to $\Delta\boldsymbol{q} = (\boldsymbol{J}^T\boldsymbol{J} + \lambda^2\mathbf{1})^{-1}\boldsymbol{J}^T\Delta\boldsymbol{p}^{ee}$
        * The addition of the $\lambda\mathbf{1}$ term regularizes the solution and keeps the inverse small even when $\boldsymbol{J}^T\boldsymbol{J}$ is close to singular

Planning

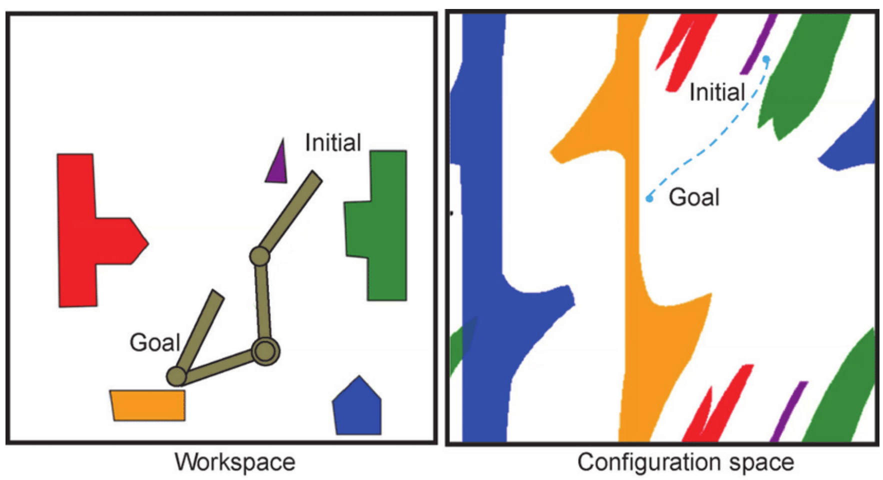

Figure 32: Mapping from workspace to configuration space.

- How do we get from work (task) space to configuration space?
- Recall that C, the configuration manifold, is the set of all possible points for the manipulator; Ω is all the parts of the configuration manifold occupied by obstacles, barriers and prohibited areas; then the free world manifold is $W = C \backslash \Omega$
- For a simple manipulator like the 2-link manipulator in 2 dimensions, we can calculate exactly where the links are and check that points on the links do not overlap obstacles
- In general, an analytical expression for this might be impossible to obtain, so we must resort to numerical methods
- The simplest way is to test point by point whether the manipulator at a given point in configuration space intersects obstacles
    - Take a point $\boldsymbol{q}$ in C and determine all the points in the manipulator, $\mathcal{M}(\boldsymbol{q})$
    - Make sure that $\mathcal{M}(\boldsymbol{q}) \cap \mathcal{O}_{i,\text{work}} = \varnothing$, then $\boldsymbol{q}$ is accessible in C
- Path planning techniques for mobile robots can also be used for manipulators in configuration space, e.g. road-map methods, Dijkstra's/A^*, potential fields, RRTs

Manipulability

- How can we measure quantitatively the ability of a manipulator to undertake a task? Can we provide a measure of the maneuverability or manipulability for a manipulator?
- We can do this kinematically or dynamically
- Consider an n-link manipulator; taking just the velocity partition, we have $\boldsymbol{v} = \boldsymbol{J}^{(v)}(\boldsymbol{q})\dot{\boldsymbol{q}}$ (we will drop the superscript from here on)
- Consider the set of all possible end-effector velocities $\boldsymbol{v}$ realizable by joint rates contained by $\|\dot{\boldsymbol{q}}\|^2 \leq 1$; intuitively, the larger this set, the more "manipulable" the manipulator is
  - Note this requires some weighting and/or non-dimensionalization if both revolute and prismatic joints are involved
  - This set will turn out to be an ellipsoid, which is called the *manipulability ellipsoid*

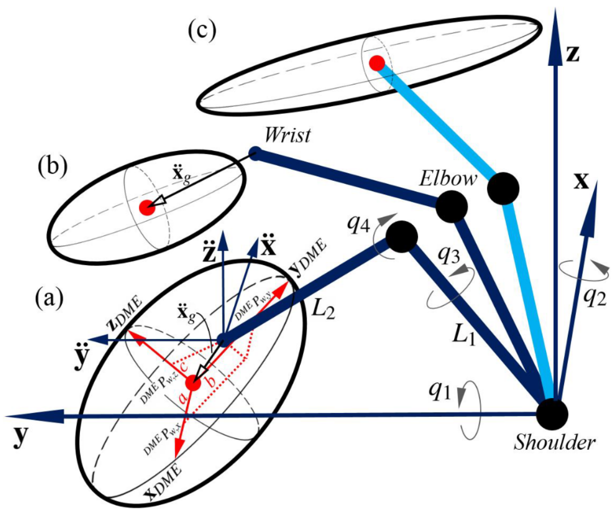

Figure 33: Manipulability ellipsoid.

- Recall that away from a singularity, $\dot{\boldsymbol{q}} = \boldsymbol{J}^\dagger \boldsymbol{v} + (\mathbf{1} - \boldsymbol{J}^\dagger \boldsymbol{J})\boldsymbol{b}$
  - This gives $\|\dot{\boldsymbol{q}}\|^2 = \dot{\boldsymbol{q}}^T \dot{\boldsymbol{q}} \geq \boldsymbol{v}^T (\boldsymbol{J}^\dagger)^T \boldsymbol{J}^\dagger \boldsymbol{v}$
  - Therefore the manipulability ellipsoid is $\boldsymbol{v}^T (\boldsymbol{J}^\dagger)^T \boldsymbol{J}^\dagger \boldsymbol{v} \leq 1$
- The principal axes of this ellipsoid represent how fast the ellipsoid can move
  - The size is given by the eigenvalues of $(\boldsymbol{J}^\dagger)^T \boldsymbol{J}^\dagger$ (like the energy/momentum ellipsoid derivation)
  - Note that substituting in the definition for $\boldsymbol{J}^\dagger$, we have $(\boldsymbol{J}^\dagger)^T \boldsymbol{J}^\dagger = (\boldsymbol{J}\boldsymbol{J}^T)^{-1}$
- Consider the SVD of $\boldsymbol{J}$: $\boldsymbol{J} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$
  - For us, $m < n$ so $\boldsymbol{\Sigma}$ has several zero columns at the end
  - Note that the singular values are the square roots of the eigenvalues of $\boldsymbol{J}\boldsymbol{J}^T$
  - Then $\boldsymbol{J}^\dagger = \boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1} = \boldsymbol{V}\boldsymbol{\Sigma}^T\boldsymbol{U}^T(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{\Sigma}^T\boldsymbol{U}^T)^{-1} = \boldsymbol{V}\boldsymbol{\Sigma}^T(\boldsymbol{\Sigma}\boldsymbol{\Sigma}^T)^{-1}\boldsymbol{U}^T$
  - And $(\boldsymbol{J}^\dagger)^T\boldsymbol{J}^\dagger = (\boldsymbol{J}\boldsymbol{J}^T)^{-1} = \boldsymbol{U}(\boldsymbol{\Sigma}\boldsymbol{\Sigma}^T)^{-1}\boldsymbol{U}^T$
  - So $\boldsymbol{v}^T(\boldsymbol{J}^\dagger)\boldsymbol{J}^\dagger\boldsymbol{v} = \boldsymbol{v}^T\boldsymbol{U}(\boldsymbol{\Sigma}\boldsymbol{\Sigma}^T)^{-1}\boldsymbol{U}^T\boldsymbol{v} \leq 1$ gives the ellipsoid
  - Let $\boldsymbol{z} = \boldsymbol{U}^T\boldsymbol{v}$, then we have $\boldsymbol{z}^T(\boldsymbol{\Sigma}\boldsymbol{\Sigma}^T)^{-1}\boldsymbol{z} = 1$
  - So in terms of z, we get $\dfrac{z_1^2}{\sigma_1^2} + \dfrac{z_2^2}{\sigma_2^2} + \dfrac{z_3^2}{\sigma_3^2} = 1$ – an ellipsoid with axes $\sigma_1, \sigma_2, \sigma_3$
- Given the ellipsoid, we can define several different measures of manipulability:
  - $w_1(\boldsymbol{q}) = \sigma_1\sigma_2\sigma_3 = \sqrt{\det(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}^T(\boldsymbol{q}))}$ (ellipsoid volume)
  - $w_2(\boldsymbol{q}) = \dfrac{\sigma_{\min}}{\sigma_{\max}}$ (ellipsoid stretching)
  - $w_3(\boldsymbol{q}) = \sigma_{\min}$ (length of shortest axis)
  - $w_4(\boldsymbol{q}) = (\sigma_1\sigma_2\sigma_3)^{\frac{1}{3}} = w_1^{\frac{1}{3}}(\boldsymbol{q})$ (geometric mean of the axes)
- For the 2-link manipulator, we have $w = |\det(\boldsymbol{J})| = l^2|\sin\theta_2|$

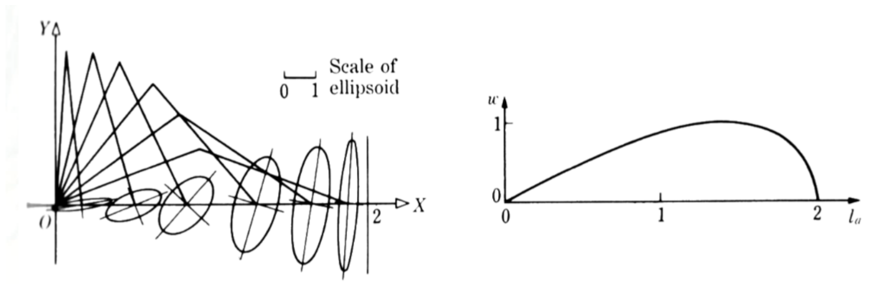

Figure 34: Manipulability ellipsiods for the 2-link manipulator.

  - Notice that this is not dependent on θ_1
  - The ellipsoid is rounder when we have intermediate values of θ_2
  - At the singularities, the ellipsoid collapses to a line, so the manipulability also drops to 0
- Recall that $\boldsymbol{\eta} = \boldsymbol{J}^T\boldsymbol{f}$, so we can address manipulability from a force/torque perspective using this dual relation
  - Consider $\|\boldsymbol{\eta}\|^2 \leq 1$
  - Going through the same steps yields $\boldsymbol{f}^T\boldsymbol{J}\boldsymbol{J}^T\boldsymbol{f} = 1 \implies \sigma_1^2 f_1^2 + \sigma_2^2 f_2^2 + \sigma_3^2 f_3^2 = 1$
  - Notice that this flips the intercepts of the ellipsoid
  - e.g. in the diagram below, larger kinematics ellipsoids result in smaller force ellipsoids

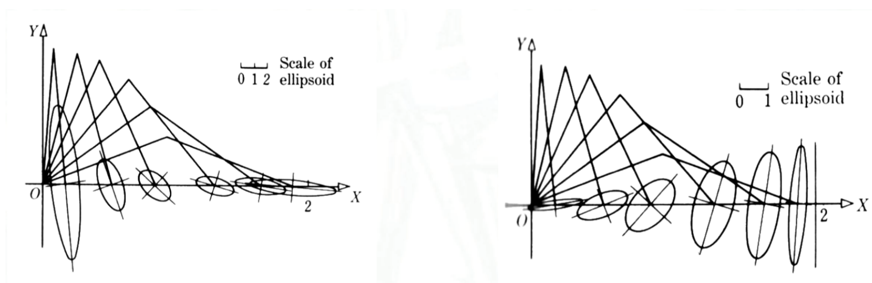

Figure 35: Force and kinematic ellipsiods for the 2-link manipulator.

Lecture 21, Nov 28, 2023

Manipulator Dynamics

- For typical mobile robots, dynamics aren't all too important to their motion
- But for manipulator systems, we may need fast movements, making dynamics important
- We would like to derive the equations of motion for manipulators, using the Lagrangian formulation

- Consider the 2-link manipulator with coordinates θ_1, θ_2, both links with length l, masses m_1, m_2, and moments of mass a bout joints c_1, c_2, I_1, I_2, and centres of mass at midlink
  - $T_1 = \frac{1}{2}m_1\left(\frac{1}{2}l\dot{\theta}_1\right)^2 + \frac{1}{2}I_{\odot,1}\dot{\theta}_1^2 = \frac{1}{2}\left(I_{\odot,1} + \frac{1}{4}m_1 l^2\right)\dot{\theta}_1^2 = \frac{1}{2}I_1\dot{\theta}_1^2$
    * Note we started with the kinetic energy relative to the centre of mass, where we have both a translational and a rotational component
    * This is equivalent to applying the parallel axis theorem
  - The speed of link 2's centre of mass is $v_{\odot,2}^2 = (l\dot{\theta}_1)^2 + \left(\frac{1}{2}l(\dot{\theta}_1 + \dot{\theta}_2)\right)^2 + l^2\dot{\theta}_1(\dot{\theta}_1 + \dot{\theta}_2)\cos\theta_2$
    * Note we can obtain this by the cosine law

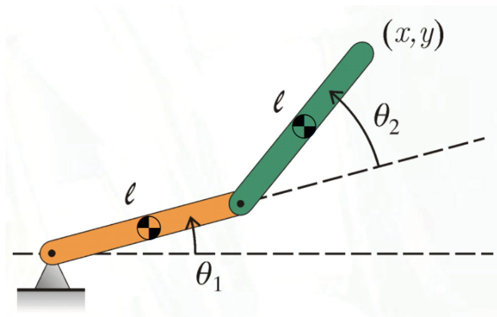

Figure 36: Two-link manipulator.

- $T_2 = \frac{1}{2}m_2\left((l\dot{\theta}_1)^2 + \left(\frac{1}{2}l(\dot{\theta}_1+\dot{\theta}_2)\right)^2 + l^2\dot{\theta}_1(\dot{\theta}_1+\dot{\theta}_2)\cos\theta_2\right) + \frac{1}{2}I_{\otimes,2}(\dot{\theta}_1+\dot{\theta}_2)^2$
  $= \frac{1}{2}m_2(l\dot{\theta}_1)^2 + c_2(l\dot{\theta}_1)(\dot{\theta}_1+\dot{\theta}_2)\cos\theta_2 + \frac{1}{2}I_2(\dot{\theta}_1+\dot{\theta}_2)^2$
  * $c_2 = \frac{1}{2}m_2 l, I_2 = I_{\otimes,2} + \frac{1}{4}m_2 l^2$
  * c_2 is the first moment of mass of link 2 about its joint
- $V_1 = \frac{1}{2}m_1 g l\sin\theta_1, V_2 = m_1 g l\left(\sin\theta_1 + \frac{1}{2}\sin(\theta_1+\theta_2)\right)$
- Virtual work done at joints: $\widehat{\delta W}_2 = \tau_1\delta\theta_1, \widehat{\delta W}_2 = \tau_2\delta\theta_2$
- $(I_1+I_2+m_2l^2+2c_2l\cos\theta_2)\ddot{\theta}_1 + (I_2+c_2l\cos\theta_2)\ddot{\theta}_2 - c_2l(2\dot{\theta}_1+\dot{\theta}_2)\dot{\theta}_2\sin\theta_2 + \left(\frac{1}{2}m_1+m_2\right)gl\cos\theta_1 + \frac{1}{2}m_2gl\cos(\theta_1+\theta_2) = \tau_1$
- $(I_2+c_2l\cos\theta_2)\ddot{\theta}_1 + I_2\ddot{\theta}_2 + c_2l\dot{\theta}_1^2\sin\theta_2 + \frac{1}{2}m_2gl\cos(\theta_1+\theta_2) = \tau_2$
- We can cast this in a general form: $\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) - \boldsymbol{f}^g(\boldsymbol{q}) = \boldsymbol{u}(t)$
  * $\boldsymbol{q} = \begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}$ are the coordinates
  * $\boldsymbol{u} = \begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix}$ are the applied forces
  * $\boldsymbol{f}^g(\boldsymbol{q}) = \begin{bmatrix}-\left(\frac{1}{2}m_1+m_2\right)gl\cos\theta_1 - \frac{1}{2}m_2gl\cos(\theta_1+\theta_2)\\ -\frac{1}{2}m_2gl\cos(\theta_1+\theta_2)\end{bmatrix}$
  * $\boldsymbol{M}(\boldsymbol{q}) = \begin{bmatrix}I_1+I_2+m_2l^2+2c_2l\cos\theta_2 & I_2+c_2l\cos\theta_2\\ I_2+c_2l\cos\theta_2 & I_2\end{bmatrix}$
    - This is the mass matrix, and it's symmetric positive definite
  * $\boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \begin{bmatrix}-c_2l(2\dot{\theta}_1+\dot{\theta}_2)\dot{\theta}_2\sin\theta_2\\ c_2l\dot{\theta}_1^2\theta_2\end{bmatrix}$
- In general, the kinetic energy of link i is $T_i = \frac{1}{2}m_i\mathbf{v}_i^T\mathbf{v}_i - \mathbf{v}_i^T\boldsymbol{c}_i^\times\boldsymbol{\omega}_i + \frac{1}{2}\boldsymbol{\omega}_i^T\boldsymbol{I}_i\boldsymbol{\omega}_i$
  - Note this uses O_i as a reference point
  - We may write this as $T_i = \frac{1}{2}\boldsymbol{v}_i^T\boldsymbol{M}_i\boldsymbol{v}_i$
  - $\boldsymbol{v}_i = \begin{bmatrix}\mathbf{v}_i\\ \boldsymbol{\omega}_i\end{bmatrix}, \boldsymbol{M}_i = \begin{bmatrix}\boldsymbol{m_i 1} & -\boldsymbol{c}_i^\times\\ \boldsymbol{c}_i^\times & \boldsymbol{I}_i\end{bmatrix}$
  - Note $\boldsymbol{M}_i = \int\begin{bmatrix}\boldsymbol{1}\\ \boldsymbol{s}^\times\end{bmatrix}\begin{bmatrix}\boldsymbol{1}\\ \boldsymbol{s}^\times\end{bmatrix}^T \mathrm{d}m$
  - These are the generalized velocity and mass matrices
- We can build a Jacobian for link i like the end-effector: $\boldsymbol{v}_i = \boldsymbol{J}_1(q_1,\ldots,q_i)\dot{\boldsymbol{q}}$

- Note that this is a function of only the coordinates up to q_i, since we have a serial manipulator
- Therefore columns of $\boldsymbol{J}_i$ after column I are zero
- Also, this Jacobian is expressed in the link frame, instead of the world frame
  * i.e. $\boldsymbol{J}_i = \begin{bmatrix} \boldsymbol{C}_{i,0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{C}_{i,0} \end{bmatrix} \boldsymbol{J}_i'$, where the latter is in the world frame
- Note for a prismatic joint, $\boldsymbol{M}_i$ is dependent on d_i (since the reference point O_i changes with d_i)
  * With the moving reference point, $\boldsymbol{s}$ changes with d_i
  * Suppose we pick some reference point O_i' fixed to the link, with position $\boldsymbol{r}$ relative to O_i, then $\boldsymbol{s} = \boldsymbol{r} + d_i \boldsymbol{1}_3$
  * $\begin{bmatrix} \boldsymbol{1} \\ \boldsymbol{s}^\times \end{bmatrix} = \begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} \\ d_i \boldsymbol{1}_3^\times & \boldsymbol{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{1} \\ \boldsymbol{r}^\times \end{bmatrix} = \boldsymbol{D} \begin{bmatrix} \boldsymbol{1} \\ \boldsymbol{r}_\times \end{bmatrix}$
  * Therefore $\boldsymbol{M}_i = \boldsymbol{D}\boldsymbol{M}_{i,O_i'}\boldsymbol{D}^T$, where $\boldsymbol{M}_{i,O_i'}$ is independent of d_i, but $\boldsymbol{D}$ is
  * This doesn't really matter in the end since we need to add up all the kinetic energies in the end

- $T_i = \frac{1}{2}\dot{\boldsymbol{q}}^T \boldsymbol{J}_i^T(\boldsymbol{q}) \boldsymbol{M}_i \boldsymbol{J}_i(\boldsymbol{q}) \dot{\boldsymbol{q}}$
  - For the whole manipulator, $T = \sum_{i=1}^{n} T_i = \frac{1}{2}\dot{\boldsymbol{q}}^T \left[\sum_{i=1}^{n} \boldsymbol{J}_i^T(\boldsymbol{q}) \boldsymbol{M}_i \boldsymbol{J}_i(\boldsymbol{q})\right] \dot{\boldsymbol{q}}$
    * We can define the middle part as the mass matrix for the whole arm
    * $\boldsymbol{M}(\boldsymbol{q}) = \sum_{i=1}^{n} \boldsymbol{J}_i^T(\boldsymbol{q}) \boldsymbol{M}_i \boldsymbol{J}_i(\boldsymbol{q})$
  - $T = \frac{1}{2}\dot{\boldsymbol{q}}^T \boldsymbol{M}(\boldsymbol{q}) \dot{\boldsymbol{q}}$
    * Note that this $\boldsymbol{M}$ is symmetric and positive definite, since if any joint is moving, we will have some amount of kinetic energy
- For gravitational potential energy, the centre of mass of link i is $\boldsymbol{r}_0^{i,\otimes} = \begin{bmatrix} \boldsymbol{\rho}_0^{i,\otimes} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{C}_{0,i} & \boldsymbol{\rho}_0^i \\ \boldsymbol{0}^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{\rho}_i^{i,\otimes} \\ 1 \end{bmatrix} = \boldsymbol{T}_{0,i} \boldsymbol{r}_i^{i,\otimes}$
  - The height is then $h^{i,\otimes} = \boldsymbol{k}^T \boldsymbol{r}_0^{i,\otimes} = \boldsymbol{k}^T \boldsymbol{T}_{0,i} \boldsymbol{r}_i^{i,\otimes}$
  - $\boldsymbol{k} = \begin{bmatrix} \boldsymbol{1}_3 \\ 0 \end{bmatrix}$
- Therefore $V_i = m_i g \boldsymbol{k}^T \boldsymbol{T}_{0,i} \boldsymbol{r}_i^{i,\otimes}$ so $\boldsymbol{V} = \sum_{i=1}^{n} V_i = \sum_{i=1}^{n} m_i g \boldsymbol{k}^T \boldsymbol{T}_{0,i} \boldsymbol{r}_i^{i,\otimes}$
- The virtual work done is $\widehat{\delta W}_i^{\text{con}} = u_i \delta q_i$ so $\widehat{\delta W}^{\text{con}} = \sum_i u_i \delta q_i = \delta \boldsymbol{q}^T \boldsymbol{u}$
  - If we have friction, $\widehat{\delta W}_i^f = f_i(q_i, \dot{q}_i) \delta q_i$
  - Then $\Delta \widehat{W}^f = \sum_i f_i \delta q_i = \delta \boldsymbol{q}^T \boldsymbol{f}^f(\boldsymbol{q}, \dot{\boldsymbol{q}})$
  - If we have forces at the end-effector, we also have $\widehat{\delta W}^{ee} = \delta \boldsymbol{q}^T \boldsymbol{J}^T(\boldsymbol{q}) \boldsymbol{f}^{ee}$
- The total non-conservative virtual work is then $\widehat{\delta W} = \delta \boldsymbol{q}^T (\boldsymbol{u} + \boldsymbol{f}^f + \boldsymbol{J}^T(\boldsymbol{q}) \boldsymbol{f}^{ee})$
- The resulting equation of motion is $\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{f}^f(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{f}^g(\boldsymbol{q}) - \boldsymbol{J}^T(\boldsymbol{q}) \boldsymbol{f}^{ee} = \boldsymbol{u}(t)$
  - $\boldsymbol{f}^f$ are the frictional forces, $\boldsymbol{f}^g$ are the gravitational forces and $\boldsymbol{f}^{ee}$ are the forces at the end-effector
  - $\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is a nonlinear inertial term
  - $h_k = \sum_{j=1}^{n} \left(\dot{M}_{kj} - \frac{1}{2} \sum_{i=1}^{n} \frac{\partial M_{ij}}{\partial q_k} \dot{q}_i \right) \dot{q}_j$
- Like kinematics, we have 2 problems: inverse dynamics (given a trajectory $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$, solve for $\boldsymbol{u}(t)$), and forward or simulation dynamics (given control forces $\boldsymbol{u}(t)$, solve for the motion $\boldsymbol{q}$)
  - Inverse dynamics is much easier than forward dynamics
  - The problem is even more complex if the manipulator links are elastic, which is an issue for space manipulators especially

- Flexibility/compliance at the joints also complicates the problem

Lecture 22, Nov 30, 2023

Manipulator Control

- Recall the manipulator dynamics: $\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{f}^f(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{f}^g(\boldsymbol{q}) - \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{f}^{ee} = \boldsymbol{u}(t)$
  - $\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}}$ are the linear (in $\ddot{\boldsymbol{q}}$) terms
  - $\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ are the nonlinear (in $\dot{\boldsymbol{q}}$) terms
- In general there are 3 approaches to control: independent joint control, computed torque control, and general feedback control

Independent Joint Control

- Assumes joints are independent; each joint is controlled by its own PID controller
- $u_k(t) = -K_{D,k}(\dot{q}_k - \dot{q}_{d,k}) - K_{P,k}(q_k - q_{d,k}) - K_{I,k}\int(q_k - q_{d,k})\,\mathrm{d}t$
  - $q_{d,k}(t)$ is the desired trajectory of joint k
- Most simple and most commonly used; does not take into account the system dynamics at all
- Since the system is highly nonlinear, there is no guarantee that this will work
- In practice gain scheduling might be used to improve results

Computed Torque Control

- Using the equations of motion, solve for the forces to effect the desired motion, and use PD control to correct for errors
- $\boldsymbol{u}(t) = \boldsymbol{M}(\boldsymbol{q})[\ddot{\boldsymbol{q}}_d - \boldsymbol{K}_D(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_d) - \boldsymbol{K}_P(\boldsymbol{q} - \boldsymbol{q}_d)] + \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{f}^g(\boldsymbol{q})$
  - Note in the following discussion we will neglect friction and end-effector force terms
- This uses a combination of feedback and feedforward control
- This requires that we know all parts of the system dynamics fairly well
- We can show that this is asymptotically stable; substitute $\boldsymbol{u}(t)$ in the manipulator dynamics, then:
  - $\boldsymbol{M}(\boldsymbol{q})[\ddot{\boldsymbol{q}}_d - \boldsymbol{K}_D(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_d) - \boldsymbol{K}_P(\boldsymbol{q} - \boldsymbol{q}_d)] + \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{f}^g(\boldsymbol{q}) = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{f}^g(\boldsymbol{q})$
  - $\boldsymbol{M}(\boldsymbol{q})[(\dot{\boldsymbol{q}} - \ddot{\boldsymbol{q}}_d) - \boldsymbol{K}_D(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_d) - \boldsymbol{K}_P(\boldsymbol{q} - \boldsymbol{q}_d)] = \boldsymbol{0}$
  - Since $\boldsymbol{M}$ is positive definite, this reduces to $\ddot{\boldsymbol{e}} + \boldsymbol{K}_D\dot{\boldsymbol{e}} + \boldsymbol{K}_P\boldsymbol{e} = \boldsymbol{0}$, where $\boldsymbol{e} = \boldsymbol{q} - \boldsymbol{q}_d$
  - This is asymptotically stable if $\boldsymbol{K}_D, \boldsymbol{K}_D$ are both positive definite
- $\boldsymbol{K}_D, \boldsymbol{K}_P$ can be chosen to be e.g. diagonal matrices, in which case this would be similar to independent joint control, but with feedforward to take into account manipulator dynamics
  - If $\boldsymbol{K}_D = \mathrm{diag}[2\zeta_i\omega_i], \boldsymbol{K}_P = \mathrm{diag}[\omega_i^2]$, then the error equation is $\ddot{e}_i + 2\zeta_i\omega_i\dot{e}_i + \omega_i^2 e_i = 0$

General Feedback Control

- PD controller with some feedforward for gravity, but not inertia
- $\boldsymbol{u}(t) = -\boldsymbol{K}_D(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_d) - \boldsymbol{K}_P(\boldsymbol{q} - \boldsymbol{q}_d) - \boldsymbol{f}^g(\boldsymbol{q})$
  - Notice that there is no $\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ term or inertia matrix
  - This requires much less knowledge of the system than the computed-torque approach
- Here we will analyze only the regulator problem (i.e. constant $\boldsymbol{q}_d$)
- We can prove that this is stable using Lyapunov theory:
  - Candidate Lyapunov function: $v(\boldsymbol{e}, \dot{\boldsymbol{q}}) = \frac{1}{2}\dot{\boldsymbol{q}}^T\boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{q}} + \frac{1}{2}\boldsymbol{e}^T\boldsymbol{K}_P\boldsymbol{e}$
    - Since $\boldsymbol{M} > 0$, assuming $\boldsymbol{K}_P > 0$, this is clearly positive definite
  - $\dot{v}(\boldsymbol{e}, \dot{\boldsymbol{q}}) = \dot{\boldsymbol{q}}^T\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \frac{1}{2}\dot{\boldsymbol{q}}^T\dot{\boldsymbol{M}}\dot{\boldsymbol{q}} + \dot{\boldsymbol{q}}^T\boldsymbol{K}_P\boldsymbol{e}$
    - From the equation of motion, $\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} = -\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{K}_D\dot{\boldsymbol{q}} - \boldsymbol{K}_P\boldsymbol{e}$ (obtain by substituting in control policy)
    - $\dot{v}(\boldsymbol{e}, \dot{\boldsymbol{q}}) = \frac{1}{2}\dot{\boldsymbol{q}}^T\dot{\boldsymbol{M}}(\boldsymbol{q})\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}^T\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \dot{\boldsymbol{q}}^T\boldsymbol{K}_D\dot{\boldsymbol{q}}$

    * The last term is negative semi-definite, but what about the rest?
  - $\frac{1}{2}\dot{\boldsymbol{q}}^T\dot{\boldsymbol{M}}(\boldsymbol{q})\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}^T\boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \sum_{k=1}^{n}\sum_{j=1}^{n}\left(\frac{1}{2}\dot{M}_{kj}\dot{q}_j - h_k\right)\dot{q}_k$
    * Recall $h_k = \sum_{j=1}^{n}\left(\dot{M}_{kj} - \frac{1}{2}\sum_{i=1}^{n}\frac{\partial M_{ij}}{\partial q_k}\dot{q}_i\right)\dot{q}_j$
    * Therefore $\frac{1}{2}\dot{\boldsymbol{q}}^T\dot{\boldsymbol{M}}(\boldsymbol{q})\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}^T\boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) = -\frac{1}{2}\sum_{k=1}^{n}\sum_{j=1}^{n}\left(\dot{M}_{kj} - \sum_{i=1}^{n}\frac{\partial M_{ij}}{\partial q_k}\dot{q}_i\right)\dot{q}_j\dot{q}_k$
    * Note $\dot{M}_{kj} = \sum_{i=1}^{n}\frac{\partial M_{kj}}{\partial q_i}\dot{q}_i$
    * Therefore $\frac{1}{2}\dot{\boldsymbol{q}}^T\dot{\boldsymbol{M}}(\boldsymbol{q})\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}^T\boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\left(\frac{\partial M_{kj}}{\partial q_i} - \frac{\partial M_{ij}}{\partial q_k}\right)\dot{q}_i\dot{q}_j\dot{q}_k$
    * But $\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\frac{\partial M_{kj}}{\partial q_i}\dot{q}_i\dot{q}_j\dot{q}_k = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\frac{\partial M_{ij}}{\partial q_k}\dot{q}_i\dot{q}_j\dot{q}_k$ if we rename the indices
    * Therefore this entire term reduces to 0
  - Hence $\dot{v}(\boldsymbol{e},\dot{\boldsymbol{q}}) = -\dot{\boldsymbol{q}}^T\boldsymbol{K}_D\dot{\boldsymbol{q}}$
    * Provided $\boldsymbol{K}_D > 0$, this is negative definite with respect to $\dot{\boldsymbol{q}}$, but not $\boldsymbol{e}$
    * We need to use Lasalle's extension
  - Consider the equation of motion: $\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{K}_D\dot{\boldsymbol{q}} + \boldsymbol{K}_P\boldsymbol{e} = \boldsymbol{0}$
    * When $\ddot{\boldsymbol{q}} = \boldsymbol{0}$, we also have $\ddot{\boldsymbol{q}} = 0$, so the equation of motion reduces to $\boldsymbol{K}_P\boldsymbol{e} = \boldsymbol{0}$
    * Since $\boldsymbol{K}_P$ is positive definite, it is also full rank, so the only solution is $\boldsymbol{e} = \boldsymbol{0}$
    * Therefore when $\dot{v} = 0$, we are forced to have $\boldsymbol{e} = \boldsymbol{0}$, so Lasalle's extension applies
  - Hence this system is asymptotically stable if $\boldsymbol{K}_P, \boldsymbol{K}_D > 0$

Lecture 23, Dec 5, 2023

Exam Review

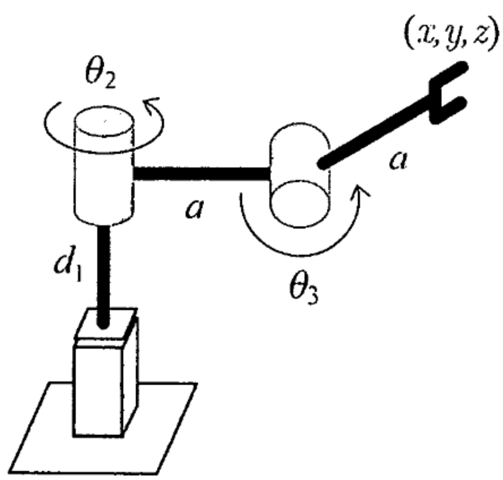

Figure 37: Example question 1.

- Refer to the manipulator example from Lecture 19
- What is the manipulability index of this manipulator, based on the volume of the manipulability ellipsoid?

– We need to find $w = \sigma_1\sigma_2\sigma_3$, where σ_i are the singular values of $\boldsymbol{J}^{(v)}$
– Since σ_i^2 are the eigenvalues of $\boldsymbol{J}\boldsymbol{J}^T$, $w = \sqrt{\lambda_1\lambda_2\lambda_3} = \sqrt{\det(\boldsymbol{J}\boldsymbol{J}^T)} = |\det \boldsymbol{J}|$, since $\boldsymbol{J}$ is square
– Recall from Lecture 19: $|\det \boldsymbol{J}| = a^2|\sin\theta_3|(1+\cos\theta_3)$

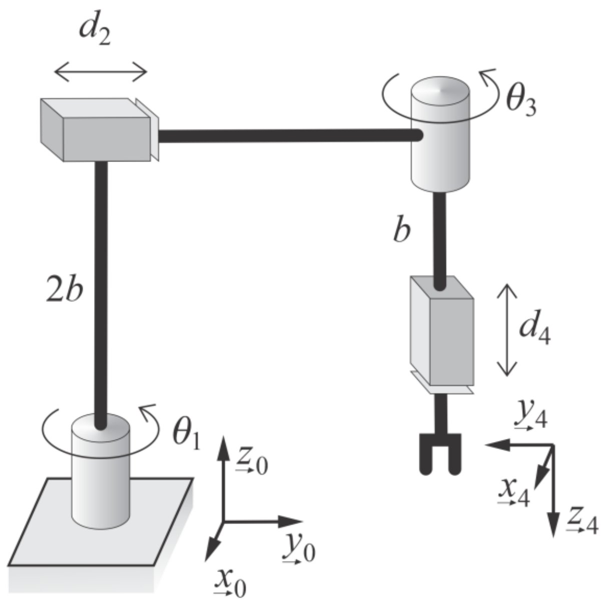

Figure 38: Example question 2.

- Consider the RPRP manipulator above (note the direction of θ_3) is reversed
  - Determine the DH transformation in $SE(3)$, $\boldsymbol{T}_{04}$
    * $\alpha_1 = 0, \alpha_2 = 270°, \alpha_3 = 270°, \alpha_4 = 0$
    * $\boldsymbol{T}_{01} = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
      - This is a rotation about the $\underset{\rightarrow}{z}_1$ axis by an angle θ_1
    * $\boldsymbol{T}_{12} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 2b \\ 0 & 0 & 0 & 1 \end{bmatrix}$
      - This is a rotation plus a translation
    * $\boldsymbol{T}_{23} = \begin{bmatrix} c_3 & 0 & -s_3 & 0 \\ 0 & 0 & 1 & 0 \\ -s_3 & -c_3 & 0 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
    * $\boldsymbol{T}_{34} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & b \\ 0 & 0 & 0 & 1 \end{bmatrix}$
    * $\boldsymbol{T}_{04} = \begin{bmatrix} c_{13} & s_{13} & 0 & -d_2 s_1 \\ s_{13} & -c_{13} & 0 & d_2 c_1 \\ 0 & 0 & -1 & b \\ 0 & 0 & 0 & 1 \end{bmatrix}$
    * Note this only takes us to link 4, but not the position of the end-effector since we need an additional length d_4

- Considering the pose of the end-effector to be only (x, y, z, ϕ), where ϕ is the rotation angle about the vertical axis, determine the theoretical singularities of the manipulator, if any
  * We can determine the manipulator pose in terms of joint variables by inspection
  * $\begin{bmatrix} x \\ y \\ z \\ \phi \end{bmatrix} = \begin{bmatrix} -d_2 \sin\theta_1 \\ d_2 \cos\theta_1 \\ b - d_4 \\ \theta_1 - \theta_3 \end{bmatrix}$
    - Note when $\theta_1 = 0$ the prismatic joint is aligned with $\underset{\rightarrow}{y}_0$
  * $\boldsymbol{J} = \begin{bmatrix} -d_2 \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ -d_2 \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & -1 & 0 \end{bmatrix}$
  * Singularities occur when $\det(\boldsymbol{J}\boldsymbol{J}^T) = 0 \iff \det \boldsymbol{J} = 0$
  * $\det \boldsymbol{J} = -(-1)(-1)(-d_2 \cos^2\theta_1 - d_2 \sin^2\theta_1) = d_2$
  * Hence the singularity is at $d_2 = 0$ – however this is theoretical, since for a real manipulator we can never collapse d_2 to exactly 0
- If there's only a force f_y^{ee} acting in the y direction (in the global frame) at the end effector, what must the joint control forces and torques be to balance it?
  * $\boldsymbol{\eta} = \begin{bmatrix} \tau_1 \\ f_2 \\ \tau_3 \\ f_4 \end{bmatrix} = \boldsymbol{J}^T \boldsymbol{f}^{ee} = \boldsymbol{J}^T \begin{bmatrix} f_x \\ f_y \\ f_z \\ \tau_z \end{bmatrix} = \boldsymbol{J}^T \begin{bmatrix} 0 \\ f_y^{ee} \\ 0 \\ 0 \end{bmatrix}$
  * We need to be careful here since we're working in 4-dimensional space instead of 6-dimensional space
  * Therefore $\boldsymbol{\eta} = f_y^{ee} \begin{bmatrix} -d_2 \sin\theta_1 \\ \cos\theta_1 \\ 0 \\ 0 \end{bmatrix}$