From Last Class

• Actuator & Sensor overview
• Linear algebra review
  ➢ Vectors
  ➢ Matrices
    - Matrix Inverse
      \[
      \begin{pmatrix}
        a_{11} & a_{12} \\
        a_{21} & a_{22}
      \end{pmatrix}^{-1} = \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \begin{pmatrix}
        a_{22} & -a_{12} \\
        -a_{21} & a_{11}
      \end{pmatrix}
      \]
    - Vector Norms
    - Eigenvalues & Eigenvectors

Intro to Robot Kinematics

• Aim
  ➢ Description of mechanical behavior of the robot for design and control
  ➢ Unlike manipulators, mobile robots move within their workspaces
    - Generally hard to measure a robot’s position
    - Positions are generally integrated over time
    - Leads to …
  ➢ Understanding/Controlling/Predicting robot motion requires us to be able to relate inputs to the motion

Overview on Numerical Integration

• Given a system described by

\[
\dot{x} = f(x, \dot{x}) + g(x, \dot{x}) \Delta x
\]

Analytically Numerically

The Algorithm

• In steps of \( \Delta x \)

\[
f(x_{\text{new}}) = f(x_{\text{old}}) + \frac{df(x_{\text{old}})}{dx} \Delta x
\]
Numerical Integration of ODEs

\[ \frac{dy}{dx} = f(x) \]
\[ h = \Delta x \]

- Initial value problem: Given the initial state at \( y_0 = y(x_0) \), to compute the whole trajectory \( y(x) \)

**Explicit Euler Solution**

\( y' = 1 - 2y \), \( y(0) = 0 \)

**Truncation errors**

- **Local truncation error**
- **Global truncation error**

**Stability**

- A numerical method is **stable** if errors occurring at one stage of the process do not tend to be magnified at later stages.
- A numerical method is **unstable** if errors occurring at one stage of the process tend to be magnified at later stages.
- In general, the stability of a numerical scheme depends on the step size. Usually, large step sizes lead to unstable solutions.

**Euler’s Method**

- **Explicit**: evaluate derivative using values at the beginning of the time step
  \[ y_{i+1} = y_i + h \cdot f(x_i) + O(h^2) \]
  - Not very accurate (global accuracy \( O(h) \)) & requires small time steps for stability
- **Implicit**: evaluate derivative using values at the end of the time step
  \[ y_{i+1} = y_i + h \cdot f(x_{i+1}) + O(h^2) \]
  - May require iteration since the answer depends upon what is calculated at the end.
  - Still not very accurate (global accuracy \( O(h) \)).
  - Unconditionally stable for all time step sizes.
  - Implicit methods are in general more stable than explicit methods.
Second-order Runge-Kutta (midpoint method)

- Second-order accuracy is obtained by using the initial derivative at each step to find a midpoint halfway across the interval, then using the midpoint derivative across the full width of the interval.
- In the above figure, filled dots represent final function values, open dots represent function values that are discarded once their derivatives have been calculated and used.
- A method is called $n$th order if its error term is $O(h^{n+1})$.

Classic 4th-order R-K method

\[ y_{n+1} = y_n + \frac{h}{6} (k_1 + 2k_2 + 2k_3 + k_4) \]

\[ k_1 = f(x, y) \]

\[ k_2 = f \left( x + \frac{h}{2}, y + \frac{h}{2}k_1 \right) \]

\[ k_3 = f \left( x + \frac{h}{2}, y + \frac{h}{2}k_2 \right) \]

\[ k_4 = f \left( x + h, y + hk_3 \right) \]

where $h = x_{n+1} - x_n$ is the step size.

Local error is of $O(h^4)$. Global error is of $O(h^5)$.

Stiff ODEs

- Stiff systems are characterized by some system components which combine very fast and very slow behavior.
- Requires efficient step size control that adapt the step size dynamically, as only in certain phases they require very small step sizes.
- Implicit method is the cure!
  - Nonlinear systems: solving implicit models by linearization (semi-implicit methods)
  - Rosenbrock – generalizations of RK method
  - Bader-Deuflhard – semi-implicit method
  - Predictor-corrector methods

Matlab Solvers for Stiff Problems

- `ode15s`
  - Variable-order solver based on the numerical differentiation formulas (NDFs). Optionally it uses the backward differentiation formulas (BDFs).
  - Multistep solver.
  - If you suspect that a problem is stiff or if `ode45` failed or was very inefficient, try `ode15s` first.
- `ode23s`
  - Based on a modified Rosenbrock formula of order 2
  - One-step solver
  - May be more efficient than `ode15s` at crude tolerances
- `ode23t`
  - An implementation of the trapezoidal rule using a “free” interpolant
  - Use this solver if the problem is only moderately stiff and you need a solution without numerical damping
Matlab Solvers for Nonstiff Problems

- **ode45**
  - Explicit Runge-Kutta (4,5) formula
  - One-step solver
  - Best function to apply as a "first try" for most problems

- **ode23**
  - Explicit Runge-Kutta (2,3)
  - One-step solver
  - May be more efficient than ode45 at crude tolerances and in the presence of mild stiffness.

- **ode113**
  - Variable order Adams-Bashforth-Moulton PECE solver
  - Multistep solver
  - It may be more efficient than ode45 at stringent tolerances and when the ODE function is particularly expensive to evaluate.

References

- Numerical Recipes in C: The Art of Scientific Computing
  William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling,
- Dr. Vijay Kumar – University of Pennsylvania and Dr. Peng Song – Rutgers University

Intro to Robot Kinematics

- **Aim**
  - Description of mechanical behavior of the robot for design and control
  - Unlike manipulators, mobile robots move within their workspaces
    - Generally hard to measure a robot’s position
    - Positions are generally integrated over time
    - Leads to inaccuracies in position (or motion) estimates
  - Understanding/Controlling/Predicting robot motion requires us to be able to relate inputs to the motion

Robot Configurations and Configuration Spaces

- **Configuration**
  - Specification of position for all points on the robot
    \[ q = (q_1, q_2, \ldots, q_n) \]

- **Configuration Space**
  - Set of all possible configurations of the robot

For a planar rigid body:

\[ C = \mathbb{R} \times \mathbb{R} \times S^1 = \mathcal{S}(2) \]
Robot Position as a Vector

• In general,

\[ q = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{bmatrix} \]

Robot position at \( t_1 \)

Degrees of Freedom

• The minimum number of parameters to define a configuration

In 3-D, a rigid link has 6 DOF
Constraints remove DOF
Revolute & Prismatic joints impose 5 constraints

Manipulator Kinematics

• Coordinates

Relative vs. Absolute Joint Angles

Joint angles can be defined relative to the previous link
Joint angles can be defined absolutely with respect to the world coordinate frame

Joint Coordinates \( \theta_1, \theta_2, \theta_3 \)
Task Coordinates \( x, y, z \)
Position Kinematics

The forward kinematics define the end-effector position for a given set of joint values.

The inverse kinematics determine the joint values for a given end-effector position (when such a solution exists).

Forward Kinematics via Trigonometry

\[ x = l_1 \cos \theta_1 + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) \]
\[ y = l_1 \sin \theta_1 + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3) \]

\[ \alpha = \theta_1 + \theta_2 + \theta_3 \]

Mobile Robot Position Kinematics

- Robot movement as a vector

Robot movement from \( t_1 \) to \( t_2 \)

\[ p_{1\rightarrow2} = (\Delta x, \Delta y, \Delta z) \]

Robot Position After Movement

Robot position at \( t_2 \)

\[ p_2 = (x_2', y_2', z_2') \]

\[ = (x_1 + \Delta x, y_1 + \Delta y, z_1 + \Delta z) \]
Example
A robot is at point A, moving with velocity \( \mathbf{w} \), and you need to move toward point B. Assuming that you can turn instantaneously, by what angle should you turn the robot?

\[
\theta = \cos^{-1} \left( \frac{\mathbf{P}_1 \cdot \mathbf{P}_{1 \rightarrow 2}}{\|\mathbf{P}_1\| \|\mathbf{P}_{1 \rightarrow 2}\|} \right)
\]

Projection: Inner (dot) product
• If \( \mathbf{v} \) is a unit vector, then \( \mathbf{v} \cdot \mathbf{u} \) yields the projection of \( \mathbf{u} \) in the direction of \( \mathbf{v} \)

\[
\text{proj}_{\mathbf{v}}(\mathbf{u}) = \mathbf{v} \cdot \mathbf{u} = \|\mathbf{u}\| \cos \theta
\]

The result is a SCALAR quantity

Forward Kinematics via Trigonometry
We use linear matrices to generalize the solution process

Rotations
Homogeneous Transformations

Coordinate Frames & Robot Positions
• Definition: coordinate frame
  > A set of orthonormal basis vectors spanning \( \mathbb{R}^n \)
  > For example,

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

• When representing a point \( p \), we need to specify a coordinate frame
  > With respect to \( O_i \)
  > With respect to \( O_j \)

\[
\begin{bmatrix}
-2.8 \\
6 \\
4.2
\end{bmatrix}
\]

Note: \( v_{x}^{0} \) and \( v_{y}^{0} \) are invariant geometric entities

But the representation is dependant upon choice of coordinate frame
Rotations

- 2D rotations
  - Representing one coordinate frame in terms of another
    \[ R^0 = [x'_0, y'_0] \]
  - Where the unit vectors are defined as:
    \[ x'_0 = \begin{bmatrix} \cos \theta \\ -\sin \theta \end{bmatrix}, \quad y'_0 = \begin{bmatrix} \sin \theta \\ \cos \theta \end{bmatrix} \]
  - This is a rotation matrix

Properties of rotation matrices

- Inverse rotations:
  \[ R^{-1} = \begin{bmatrix} x_0 & y_0 \\ y_0 & x_0 \end{bmatrix} \]
  \[ R^{-1} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \]
  - Or, another interpretation uses odd/even properties:
    \[ R^{-1} = \begin{bmatrix} \cos (-\theta) & -\sin (-\theta) \\ \sin (-\theta) & \cos (-\theta) \end{bmatrix} \]

Alternate approach

- Rotation matrices as projections
  - Projecting the axes of from \( o_1 \) onto the axes of frame \( o_0 \)

Properties of rotation matrices

- Inverse of a rotation matrix:
  \[ (R^0)^{-1} = \begin{bmatrix} x_0 & y_0 \\ y_0 & x_0 \end{bmatrix} \]
  \[ (R^0)^{-1} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \]
  - The determinant of a rotation matrix is always ±1
    - +1 if we only use right-handed convention
Properties of rotation matrices

• Summary:
  ➢ Columns (rows) of R are mutually orthogonal
  ➢ Each column (row) of R is a unit vector
  \[ R^T = R^{-1} \]
  \[ \det(R) = 1 \]

• The set of all \( n \times n \) matrices that have these properties are called the Special Orthogonal group of order \( n \)
  \[ R \in SO(n) \]

Rotational transformations

• Now assume \( p \) is a fixed point on the rigid object with fixed coordinate frame \( O_1 \)
  ➢ The point \( p \) can be represented in the frame \( O_0 \) (\( p^0 \)) again by the projection onto the base frame

3D rotations

• General 3D rotation:
  \[ R^0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix} \]

• Special cases
  ➢ Basic rotation matrices
  \[ R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix} \]
  \[ R_y = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix} \]
  \[ R_z = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

Rotating a vector

• Another interpretation of a rotation matrix:
  ➢ Rotating a vector about an axis in a fixed frame
  ➢ Ex: rotate \( v^0 \) about \( y_0 \) by \( \pi/2 \)
  \[ v^0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \]
  \[ v^1 = R_{y_{10}} v^0 \]
  \[ = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \]
Composition of Rotation Matrices

- Since \((R_1 R_2)R_3 = R_1(R_2 R_3)\) \(\text{Associativity}\)

- And \(R_1, R_2 \in SO(3) \Rightarrow R_1 R_2 \in SO(3)\)

- Then, w/ respect to the current frame
  \[ p^0 = R_1^0 p^1 \]
  \[ p^1 = R_1^1 p^2 \]
  \[ p^2 = R_1^2 p^3 \]

- BUT REMBER: In general, members of \(SO(3)\) do not commute \(R_2 R_1 \neq R_1 R_2\)