Algorithmic Robotics Guide

Material taken from EECS 498

Author
Anthony Liang

December 16, 2020

Contents

Chapter 1

Introduction

This is a guide I put together during my time as a TA for the Algorithmic Robotics class taught at the University of Michigan by Professor Dmitry Berenson. Most of the content is directly taken from his slides, but I also appended some information from my own past knowledge and online resources. This guide will cover topics ranging from a review of mathematical concepts in linear algebra and statistics to foundations of robot planning, state estimation, and control.

Chapter 2

Linear Algebra Review

Matrices are the fundamental representation for data in robotic applications.

2.1 Vectors and vector spaces

Scalar: a single number (e.g. 1.234)

Vector: an ordered list of n scalars where n is the dimensionality (e.g. $\begin{bmatrix}1\\2\\3\end{bmatrix}$)

Vectors can be interpreted as arrows in an n-dimensional vector space. [Insert diagram]

Vector operations
Vectors must have the same dimensionality.

- Addition: $v + w = \begin{bmatrix}v_1 + w_1\\v_2 + w_2\\\vdots\end{bmatrix}$
- Subtraction: $v - w = \begin{bmatrix}v_1 - w_1\\v_2 - w_2\\\vdots\end{bmatrix}$
- Scalar multiplication: $\alpha v = \begin{bmatrix}\alpha v_1\\\alpha v_2\\\vdots\end{bmatrix}$
- Norm: "intuitively" represents the length of a vector, is a scalar value
  p-norm - $\|v\|_p = (\sum_{i=1}^{n} |v_i|^p)^{\frac{1}{p}}$
  1-norm - $\|v\|_1 = (\sum_{i=1}^{n} |v_i|)$
  2-norm - $\|v\|_2 = (\sum_{i=1}^{n} |v_i|^2)^{\frac{1}{2}}$
- Unit vector: a vector with Euclidean norm of 1 ($\|v\| = 1$)
- unit vectors are used to describe directions in coordinate frames and transforms

Basis Vectors

A set of vectors is said to be linearly independent if no vector is a linear combination of another other vectors in the set.

e.g. $\{ \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix} \}$ is linearly independent and commonly called the *standard basis* for $\mathbb{R}^3$

$\{ \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}2\\1\\0\end{bmatrix} \}$ is not linearly indepdent because the last vector a linear combination of the first two, $2v_1 + v2 = v_3$

A set of vectors $\mathcal{B} = \{b_1, b_2, ...\}$ spans a vector space if any vector in the space can be written as a linear combination of other vectors in the space. Formally, for any $v \in \mathbb{R}^n, v = \alpha_1 b_1 + \alpha_2 b_2 + \dots$

A basis of vector space $\mathbb{R}^n$ is a set of $\underline{\text{linearly independent}}$ vectors that $\underline{\text{span}}$ the entire space.

Note: Basis vectors are not unique. (show example)

Vector dot product (inner product)

- $v \cdot w = \langle v, w \rangle = \sum_{i=1}^{n} v_i w_i$
- The angle between two vectors is: $\theta = arccos(\frac{v \cdot w}{\|v\|\|w\|})$
- v and w are $\underline{\text{orthogonal}}$ if $v \cdot w = 0$. The angle between two orthgonal vectors is $\frac{\pi}{2}$
- Dot product and scalar: $\alpha(v \cdot w) = (\alpha v) \cdot w = v \cdot (\alpha w)$
- Distribution over addition: $v \cdot (w + p) = v \cdot w + v \cdot p$

Vector projection

[TODO]

Vector cross product

The cross product of two vectors results in a third vector that is orthogonal to both vectors. Apply right hand rule to determine the direction of the resulting vector.

$$c = a \times b$$

$$c = \begin{bmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{bmatrix}$$

2.2 Matrices

A matrix is a rectangular array of values, two-dimensional vector. A vector is a matrix with 1 column. Like vectors, we can add matrices with the same dimensions together and multiply a matrix by a scalar value.

Matrix operations
For matrices A and B with dimensions $m \times n_a$ and $n_b \times k$ respectively.

- Multiplication: $(AB)_{ij} = \sum_{k=1}^{n_a} a_{ik} b_{kj}$
  - Multiplication is not commutative! You can only multiple two matrices if the number of columns of matrix A is equal to the number of rows in matrix B (i.e. $n_a = n_b$)
- Transpose: the transpose of a matrix is done by flipping a matrix over its diagonal such that $[A^T]_{ij} = A_{ji}$
  - e.g. [todo]
  - $(AB)^T = B^T A^T$
  - $(A^T)^T = A$
- Identity: the identity matrix (I_n) is an $n \times n$ matrix with 1's along its diagonal and 0's everywhere else
  - e.g. $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
- Inversion: A^{-1} is the inverse of A if $A^{-1}A = AA^{-1} = I$
  - To have an inverse, A must be a square matrix and invertible.
  - A square matrix is $\underline{\text{singular}}$ if it is not invertible.
  - A square matrix is invertible, if either it has rank n or if its determinant is 0. There are many other ways to check for invertability.
  - Matrix inversion is commonly used in linear algebra to solve a system of linear equations. $Ax = b \to x = A^{-1}b$

Pseudo-inverse

- The Moore-Penrose Pseudo-inverse is defined as $A^+ = (A^T A)^{-1} A^T$ (left pseudo-inverse)
- The Moore-Pensore Pseudo-inverse works even when A is not a square matrix. If A is square and invertible, then $A^+ = A^{-1}$.

TODO write about underdetermined systems

Chapter 3

Transformations

3.1 Homogenous Transforms

3.1.1 Conventions

- Objects are abstracted by a set of axes fixed to the body, called coordinate frames.
- Points possess position but not orientation. Rigid bodies possess both position and orientation.
- Mechanics is about relation between two objects.
  - a is "r-related" to b is: r_a^b.
  - velocity (v) of a robot (r) relative to (e): v_r^e
  - "r" is not a property of a. "r" is a property of a *relative* to b
- Relationship is directional and asymmetric: $r_a^b \neq r_b^a$
- Vectors of physics are coordinate system independent.
- Vectors of linear algebra are coordinate system dependent.
- Subscripts denote the object frame possessing the quantity: v_{wheel} - velocity of the wheels
- Superscripts denote the coordinate system within which the quantity is expressed: v_{wheel}^{world} - velocity of the wheel w.r.t the world

3.1.2 Definitions

- Affine Transforms: most general linear transform
  - $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} t_2 \\ t_2 \end{bmatrix}$
  - Can be used for translation, rotation, scale, reflections, and shears
  - Preserves linearity but not distance (hence not areas or angles)
- Homogeneous Transforms: $t_1 = t_2 = 0$

  - $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$
  - Can be used for rotation, scale, reflections, and shears (**not translation**)
  - Preserves linearity but not distance (hence not areas or angles)
- Orthogonal Transforms: Same as homogeneous transforms, buts
  - $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$
  - $$\begin{aligned} r_{11}r_{12} + r_{21}r_{22} &= 0 \\ r_{11}r_{11} + r_{21}r_{21} &= 1 \\ r_{12}r_{12} + r_{22}r_{22} &= 1 \end{aligned}$$
  - Pairwise dot product of columns in R must equal 0. Norm of each column must be 1.
  - Can be used for rotation and reflections.
  - Preserves linearity <u>AND</u> distance.
- Rotation Matrix: Same as orthogonal transforms, but
  - $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$
  - $$\begin{aligned} r_{11}r_{12} + r_{21}r_{22} &= 0 \\ r_{11}r_{11} + r_{21}r_{21} &= 1 \\ r_{12}r_{12} + r_{22}r_{22} &= 1 \\ det(R) &= 1 \end{aligned}$$
  - Can be used for rotations.
  - Preserves linearity <u>AND</u> distance.
- Orientation: altitude and (heading or yaw)
- Pose: position and orientation
  - 2D: $\begin{bmatrix} x & y & \psi \end{bmatrix}^T$
  - 3D: $\begin{bmatrix} x & y & z & \theta & \phi & \psi \end{bmatrix}^T$
- : Posture: pose plus some configuration

3.1.3 Homogeneous Transforms

- Pure Direction

- Points in 3D can be rotated, reflected, scaled and shared with 3x3 matrices but not translated.
- Trick: Move to 4D
- $$p_2 = p_1 + p_k = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} + \begin{bmatrix} x_k \\ y_k \\ z_k \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_k \\ 0 & 1 & 0 & y_k \\ 0 & 0 & 1 & z_k \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = trans(p_k)p_1$$
- Operators
  - $trans(u, v, w) = \begin{bmatrix} 1 & 0 & 0 & u \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & w \\ 0 & 0 & 0 & 1 \end{bmatrix}$
  - $rot_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) & 0 \\ 0 & sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
  - Operating on a point v.s. operating on a direction
- The columns of the identity HT can be considered to represent the coordinate frame itself.
- Homogeneous Transforms are both operators and frames. They can be both the things that operate on other things and things operated upon.

3.1.4 Euler Angles

- Can define rotation relative to axes of a frame
- Euler angles have trouble rotating about two or more axes
- Many euler angles map to one rotation (gimbal lock)

3.1.5 Quaternions

Chapter 4

Convexity and Optimization

4.1 Convex Optimization

- Mature field with deep mathematical foundations
- Scales well with dimensionality, solves problems with 1000s of variables
- Convex optimizers are usually really fast
- Functions are defined as $f : A \to B$, "f maps elements in the set A to elements in set B"
- Derivatives: a linear approximation to a function at a certain point
  - $Df(x) = \underset{h \to 0}{lim} \frac{f(x+h)-f(x)}{h}$ for f from $\mathbb{R} \to \mathbb{R}$
  - Suppose $f : \mathbb{R}^n \to \mathbb{R}^m$
    - f is differentiable at x if there exists a matrix $Df(x) \in \mathbb{R}^{mxn}$
    - $Df(x)$ is called the derivative (or **Jacobian**) of the function
    - $Df(x)_{ij} = \partial \frac{f_i(x)}{\partial x_j}$ for $i = 1...m, j = 1, ..., n$

4.2 Search for Optimization

- Consider non-convex continuous problems and problems where variables are discrete
- A graph is a set of vertices V and edges E
- Graphs capture the idea of adjacency and we can use adjacent relationships to search the graph for a certain node or path between nodes
- In optimization, adjacency between nodes can be used to determine what solutions to explore next
- e.g. n-queens
- Local search algorithms: used to search graph for best solution
  - To solve optimization problems using search:

    - Define set of possible solutions (nodes)
    - Define adjacency between solutions (edges)
    - Define cost/value/fitness function for nodes
  - Descent methods are a form of local search algorithms
  - Hill climbing: consider next possible moves and pick one that improves the cost function the most
    - Drawbacks: depending on initial state, can get stuck in local optima
    - Can try running algorithm some number of times with random start state. If you run enough times, you will get the answer (in the limit). Takes a lot of time, no guarantees on when to terminate.
  - Simulated Annealing
    - Explicitly inject variability into search process
    - More variability at beginning of search and decrease this over time (don't want to move away from good solution)
    - Using a temperature schedule
- Evolutionary Algorithms
  - Genetic algorithms: inspired by process of evolution in nature
  - Operators:
    - Crossover: new state generated from two parent states
    - Mutations: Randomly change component of state

Algorithm 1: Genetic algorithms

```
1. Initialize population (k random states);
2. Select a set of parents from population for mating (based on fitness);
3. Generate children via crossover of parents;
4. Mutation (add randomness to children);
5. Evaluate fitness of children;
6. Replace worst parent with children;
7. Repeat from step 2
```

4.3 Search for a Path (used in motion planning)

- Formulating a path search problem
  - State space
  - Successor Function:
  - Actions
  - Action Cost
  - Goal Test

- Tree Search Algorithms
  - Completeness: does it always find a solution if one exists?
  - Optimality: does it always find the least-cost solution?
  - two types of complexity:
    - Time complexity
    - Space complexity
  - Measured in terms of
    - b: maximum branching factor of search tree
    - d: depth of least-cost solution
    - m: maximum depth of state space
  - Breadth-first search
  - Depth-first search
  - Best-first search
  - A* Search
  - Variants of A*
    - Dynamic A* (D*), Lifelong Planning A*, Anytime Repairing A*

Chapter 5

Motion Planning

- Motion Planning is the automatic generation of motion / path for a robot that does not collide with obstacles.
- Path planning is global search for path to goal whereas obstacle avoidance ("local navigation") is a reactive method
  - Exact algorithms
    - Either find a solution or prove one doesn't exist
    - Computationally expensive
    - Unsuitable for high-dimensional spaces
  - Discrete search
    - Divide space into grid, use A* search
    - Unsuitable for high-dimensional spaces
  - Sampling-based planning
    - Sample the C-space, construct path from samples
    - Good for high-dimensional spaces
    - Weak completeness and optimality guarantees

5.1 Methods

- Visibility graph
  - Continuous representation (configuration space formulation)
  - Discretization (random sampling)
  - Graph searching (BFS, DFS, A*)
- A visibility graph is a graph such that
  - nodes: q_{init}, q_{goal} or obstacle vertex

  - edges: edge exists between nodes u and v if the line segment between u and v is an obstacle edge or does not intersect the obstacles

- Algorithm
- Computational Efficiency
- Cell decomposition: decompose free space into simple cells and represent connectivity of free space by adjacency graph of these cells
  -
- Potential field: define potential function over free space that has global minimum at goal and follow the steepest descent of the potential function

5.2 Configuration Space

- Open set - a set with no boundary. Every point in the set has an open neighborhood which is also in the set.
- Closed set - a set with a boundary. A closed set is a complement of some open set and vice versa.
- A set X is called a topological space if there is a collection of open subsets of X such as
  - the union of any number of open sets is an open set
  - the intersection of a finite number of open sets is an open sets
  - both X and $\emptyset$ are open sets
- Two topological spaces X and Y are **homeomorphic** if there is a bijective function $f : X \to Y$ and both f and f^{-1} are continuous.
- Homeomorphisms can not add or remove holes.
- Common Topological Spaces: the real numbers ($\mathbb{R}^1$), the unit circle ($\mathbb{S}^1$)
- Can make more complex spaces using the Cartesian product.
- $\mathbb{R}^1 \times \mathbb{S}^1$ = a hollow cylinder
- The **configuration** of a moving object is a specification of the position of every point on the object
  - A configuration q is usually expressed as a vector of the DOF of the robot
- The **configuration space** C is the set of all possible configurations. Usually this is a topological space. A configuration q is a point in C.
- The **dimension of a configuration space** is the minimum number of DOF needed to specify the configuration of the object completely.
- An **articulated** object is a set of rigid bodies connected by joints.
- A **path** in C is a continuous curve connecting two configurations q_{start} and q_{goal}.
- A **trajectory** is a path parameterized by time.

  - A configuration q is collision-free if the robot placed at q does not intersect any obstacles in the workspace.
  - The free space C_{free} is a subset of C containing all free configurations.
  - A configuration space obstacle C_{obs} is a subset of C that contains all configurations where the robot collides with workspace obstacles or with itself.
- Minkowski sum [insert image]

5.3 Sampling-based Planning

- How do we plan in **high-dimensional** C-spaces?
- Exact methods either find a solution of prove none exists
- They require computing C-space obstacles which are very computationally expensive!
- Discrete search run-time and memory requirements are sensitive to branching factors (# of successors)
- Number of sessions depends on dimension, n-dimensional 8-connected space has $3^n - 1$ successors
- In sampling-based planning, instead of systematically-discretizing the C-space, take samples in the C-space and use them to construct a graph.
- Advantages
  - Don't need to discretize
  - Don't need to explicitly represent C-space
  - Easy to sample high-dimensional spaces
- Disadvantages
  - Probability of sampling an area depends on the area's size, hard to sample narrow passages
  - No guarantees on completeness / optimality
- Prrobabilistic Roadmap (PRM) - multi-query algorithms because roadmap can be reused if environment and robot haven't changed between queries
  - Build a roadmap of the space from sampled points and search roadmap to find a path
  1. "Learning" Phase
     (a) Construction step
        i. Build roadmap by sampling random free configurations and connect them using a fast local planner
        ii. Store configurations as nodes in a graph
        iii. Edges of graph are paths between nodes found by local planner

  - Need a distance metric to define "nearest": $D(q_1, q_2)$, use Euclidean distance
  - Naive NN can be slow with 1000s of nodes, so use **kd-tree** to store nodes and do NN queries
  - Kd-tree is a data structure that recursively divides the space into bins that contains points (like Oct-tree) and nearest neighbor searches through bins to find nearest point
  - Local planner can be anything, but must be fast because it is called many times by the algorithm
  - Easiest way is just to connect points using a straight line and check whether the line is collision free

(b) Expansion step

  - You can have disconnected components that should be connected
  - Expansion uses heuristics to sample more nodes to connect disconnected components
  - No "right" way, this step is environment dependent

2. Query Phase
   - Given start q_s and goal q_g. Connect them to the roadmap using a local planner.
   - Then search the graph G to find the shortest path between q_s and q_g using A*, Dijkstra's, etc.

Algorithm 2: Path shortening / smoothing

```
for $i = 0$ to maxiterations do
    pick two points $q_1$ and $q_2$ on the path randomly;
    try to connect them with a line segment;
    if successful, replace path between $q_1$ and $q_2$ with the line segment;
end
```

3. PRM Failure Modes
   - Cannot connect q_s and q_g to any nodes in the graph
   - Cannot find a path in the graph but path is possible
4. PRM issues
   - Uniform random sampling misses narrow passages
   - Exploring whole space, but all we want is a path
5. Sampling strategies
   - Gaussian sample
   - Bridge sample

        * Sample a q_1 in collision
        * Sample a q_2 in neighborhood of q_1 with some prob distribution
        * If q_2 is in collision, get the midpoint of (q_1, q_2)
        * Check if midpoint is in collision and if not add it as a node

- Rapidly-exploring Random Trees (RRTs) : single-query method
    - Build a **tree** instead of a graph**
    - The tree grows in C_{free}
    - Like PRM captures some connectivity, but unlike PRM it only explores what is connected to q_{start}
    - RRT Goal Biasing
        * Bias RRTs towards goal to produce a path
        * When generating a random sample, with some probability pick the goal instead of random node
    - RRT Extension Types
        * RRT-Extend: Take one step towards a random direction
        * RRT-Connect: Step towards random sample until it is either reached or you hit an obstacle
        * BiDirectional RRTs: grow tree from both start and goal
        * RRT produces bad paths, must perform path smoothing (ALWAYS)

Algorithm 3: Naive Tree algorithm

```
1 $q_{node} = q_{start}$;
2 for $i$ = 1 to num_samples do
3     $q_{rand}$ = sample near $q_{node}$;
4     Add edge $e = (q_{rand}, q)$ if collision-free;
5     $q_{node}$ = pick random node of tree;
6 end
```

Algorithm 4: Build RRT

```
1 T.init($q_{init}$);
2 for $k$ = 1 to $K$ do
3     $q_{rand}$ = random_config();
4     extend(T, $q_{rand}$);
5 end
```

5.4 Nonholonomic Planning

- **Holonomic constraints** depend only on configuration
- $F(q, t) = 0$

- These have to be bilateral constraints (no inequalities)
- Example: kinematics of a unicycle
  - Can move forward and backward
  - Can rotate about the wheel center
  - Can't move sideways
- **Non-holonomic** constraints are non-integrable. Thus they must contain derivatives of configuration. Sometimes called differential constraints.
- State space (configuration + velocity) vs control space (speed or acceleration, steering)
- Simple Car
  - Non-holonomic constraint: $-\dot{x}sin(\theta) + \dot{y}cos(\theta) = 0$
  - Essentially means you can't move sideways
- Two-point Boundary Value Problem (BVP): find a control sequence to take system from state x_i and x_g while obeying kinematic constraints
- Discrete Planning Option 1: Sequencing primitives
  - Discretize control space into primitives (pick steering angles, accelerations, velocities)
  - Disadvantage: losing full continuous completeness and discontinuous curvature
  - Choice of primitives affects completeness, optimality, and speed
- Discrete Planning Option 2: State Lattice
  - Pre-compute state lattice
  - Two methods to get lattice:
    * Forward: using motion primitives
    * Inverse: Use BVP solvers to find trajectories between states
  - Impose continuity constraints at graph vertices
  - Search state lattice like any graph
- Sampling-based Planning
  - Building state lattice is impractical in high dimensions
  - We are now sampling state space, not C-space!
  - Challenges
    * Dimension of space is doubled
    * Moving between points is harder
    * Distance metric is unclear (worst problem)
  - RRT Non-holonomic Planning

*

Chapter 6

Kinematics

Chapter 7

Grasping

- Grasping studies how to stably make contact with objects and move them
- Definitions

7.1 Definitions

- A point contact is sometimes called a finger
- A **wrench** is a combination of force and torque applied to an object
- **Wrench space** is the space of wrenches applied to an object
  - 2D object: 3 dimensional wrench space (2 force, 1 torque)
  - 3D object: 6 dimensional wrench space (3 force, 3 torque)
- A grasp **immobilizes** an object if it can counter any wrench applied to the object. This guarantees the stability of the grasp.
- A **friction cone** is the set of forces that can be applied at a contact force without sliding on the object. Assume Coulomb friction.
  - Depends on the coefficient of friction between hand and object (μ)
  - Bigger μ implies a wider friction cone.

7.2 Form Closure

- A form closure grasp is when the object cannot move **regardless of surface friction**
- You need at least N+1 contacts to achieve first-order form closure, where N is the number of DOF of the object

7.3 Force Closure

- Frictional properties of the object can be used to immobilize it

- If a grasp achieves form closure, it also achieves force closure
- Intuition, need a contact force to cancel out external disturbance force. Convex hull must contain origin for there to exist such a contact force.
- For a 3D object, you only need 3 contacts to achieve force closure (as opposed to 7 for form closure)
- Force Closure Metrics
  - Popular metric: radius of largest hyper-sphere you can fit in convex hull
  - Task specific metric: using an ellipsoid instead of a hyper-sphere

Algorithm 5: Testing for force closure

```
1 Input: Contact locations;
2 Output: Is the grasp in force-closure?;
3 1. Approximate friction cone at each contact with a set of wrenches.;
4 2. Combine wrenches from all cones to a set of points $S$ in wrench space.;
5 3. Compute the convex hull of S (smallest convex set that contains all points);
6 4. If the origin is inside the convex hull, return YES. Else return NO.;
```

7.4 Searching for Force Closure Grasps

- Peter Allen et al. 2000s
  - Sample pose of hand relative to object with fingers in a pre-determined shape
  - Approach object until contact and close fingers
  - Get contact points between hand and object
  - Test these contact points for force closure
- Pre-compute grasp sets: searching for grasps is slow!
- Columbia Grasp Database

7.5 Integrating Grasping and Motion Planning

- Pre-compute grasp set offline, get force-closure scores
- Online: compute 2 scores for each grasp: Environment Clearance Score and Reachability Score
- Test grasps in order of ranking
- Recent work in grasping uses deep learning methods
- General idea
  - Generate many grasp candidates
  - Learn a quality metric that uses the point cloud data directly
  - Output highest quality grasp

Chapter 8

SVD and PCA

How do we transform the data to get rid of "unimportant" dimensions/rotations?

8.1 Definitions

- **Variance** is the measure of deviation from the mean for points in one dimension
- **Covariance** is the measure of how much each dimensions vary from the mean with respect to each other
- Covariance is measured between pairs of dimensions to see their correlation
- Magnitude of covariance is not as important as sign
- $+$ covariance means both dimensions increase or decrease together
- - covariance means one increases while the other decreases
- covariance $= 0$ means the dimensions are independent of one another
- Estimate covariance matrix:
  1. Subtract the mean of the datapoints from every column of X
  2. $Q = \frac{XX^T}{n-1}$
- Eigenvalue problem
  - A: $n \times n$ matrix
  - v: $n \times 1$ non-zero vector
  - λ: scalar
  - $Av = \lambda v$
- A value of λ for which this equation has a solution is called an **eigenvalue** of A
- A v corresponding to this value of λ is called an **eigenvector** of A
- All eigenvectors of a matrix are *orthgonal* to each other
- Eigenvectors of a covariance matrix

  - Eigenvectors of Q with the largest eigenvalues correspond to dimensions that have the strongest correlation in the dataset
  - Eigenvectors of Q are **principle components**

8.2 Singular Value Decomposition (SVD)

- SVD decomposes any matrix M into $M = U\Sigma V^T$
- The columns of U are the eigenvectors of MM^T
- Σ is a diagonal matrix where the elements of the diagonal are the $\sqrt{eigenvalues}$ of M^TM and MM^T in decreasing order of magnitude
- Columns of V are the eigenvectors of M^TM
- Columns of U and V are **orthonormal** meaning each column vector has unit magnitude and orthogonal to all other column vectors.

8.3 Principle Component Analysis (PCA)

- We care about the variance of the data
- High variance = high importance
- PCA is a technique used
  - Remove rotation in a dataset
  - Reduce dimensionality of a dataset
- PCA computes linear transformation that chooses a new coordinate system for the data set such that the greatest variance by any projection of the data set comes to lie on the first axis (called the **first principal component**)

Algorithm 6: PCA

```
1. Given dataset X;
2. Compute mean of X;
3. X = X - $\mu$ (subtract mean from every point in X);
4. Compute covariance Q of X;
5. Take the SVD of $Q = U\Sigma V^T$;
6. $X_{new} = V^T X$;
```

8.3.1 Limitations of PCA

- PCA is sensitive to the scaling of the variables
- ** need to review this part

8.3.2 Applications of PCA

- Eigenfaces: analysis of database of face images

- Instead of using all the pixel values, we can represent a face as a weighted combination of eigenfaces
- SVD takes a long time to run for high-dimensional data
- Can be used for video compression
- Many high-dimensional datasets have hidden low-dimensional structure

Chapter 9

Probabilistic Models

9.1 Probability Basics

9.1.1 Discrete Random Variables

- X denotes a random variable and it can take on a countable number of values in $\{x_1, x_2, ..., x_n\}$
- $P(X = x_i)$ is the probability that the random variable X takes on value x_i
- $P(...)$ is called the **probability mass function**
- e.g. $P(Room) = \langle 0.7, 0.2, ..., 0.02 \rangle$

9.1.2 Continuous Random Variables

- X takes on a value in the continuum
- $P(X = x)$ is the $probability density function$
- $P(x \in (a, b)) = \int_a^b P(x)dx$

9.1.3 Axioms of Probability Theory

- $0 \leq P(a) \leq 1$
- $P(true) = 1$ and $P(false) = 0$
- $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$

9.1.4 Joint and Conditional Probability

- $P(X = x \wedge Y = y) = P(x, y)$
- If X and Y are **independent** then $P(x, y) = P(x)P(y)$
- $P(x|y) = \frac{P(x,y)}{P(y)}$
- $P(x, y) = \frac{P(x|y)}{P(y)}$

- If X and Y are **independent** then $P(x|y) = P(x)$
- $P(x, y|z) = P(x|z)P(y|z)$ means that x and y are **conditionally independent**
- If I know z, I don't need to know x to compute the probability of y.

9.1.5 Law of Total Probability (Discrete)

- $\sum_x P(x) = 1$
- $P(x) = \sum_y P(x, y) = \sum_y P(x|y)P(y)$

9.2 Bayes Rule

- $P(x|y) = \frac{P(y|x)P(x)}{P(y)} = \frac{likelihood*prior}{evidence}$
- Usually $P(y)$ is difficult to compute, so use normalization trick
- $P(x|y) = \eta P(y|x)P(x)$ where $\eta = \frac{1}{\sum_{x \in X} P(y|x)P(x)}$

9.2.1 Casual and Diagnostic Reasoning

- Suppose a robot wants to determine probability of a door being open
- It obtains measurement z. What is the $P(open|z)$
- $P(open|z)$ is **diagnostic** and $P(z|open)$ is **causal**

9.2.2 Conditional Independence Example

- Consider three variables: $RobotLocation, GPSEstimate, LandmarkEstimate$
- $GPSEstimate$ and $LandmarkEstimate$ are NOT independent, $P(GPSEstimate|LandmarkEsti$
  $P(GPSEstimate)$
- $GPSEstimate$ and $LandmarkEstimate$ are conditionally independent given $RobotLocation$
- If I know the robot's location, then I can compute the landmark estimate without knowing the GPS estimate

9.3 Bayes Net

- Encode conditional independence relationships in a Bayes Net. Used to describe cause-effect relationships
- Directed and acyclic graph
  - Nodes represent random variables
  - Edges represent conditional dependencies
  - Nodes that are not connected are conditionally independent of each other
  - Node is associated with a probability function $P(X_i|Parents(X_i))$, this is defined by a conditional probability table (CPT)

- Inference ....

9.3.1 Markov Random Fields

- Graph is undirected and may be cyclic

9.3.2 Conditional Random Fields

- Undirected graphical model whose nodes can be divided into exactly two disjoint sets:
  - X: the input variables
  - Y: the observed and output variables
- Used to model the conditional distribution: $P(Y|X)$
- CRFs can be used for object recognition and image segmentation

9.4 Learning a probabilistic model

-

Chapter 10

Filters

10.1 Bayes Filter

- Bayes Filter accounts for both robot state and perception data
- To use Bayes filter, the state must be discrete, usually represented by a grid
- Each grid cell, contains the **belief** (probability that the true state of the system is x_t)

Algorithm 7: Discrete Bayes Filter Algorithm

```
Inputs: Bel(x), d;
$\eta = 0$;
if d is a perceptual data item z then
    for all x do
        $Bel'(x) = P(z|x)Bel(x)$;
        $\eta = \eta + Bel'(x)$;
    end
    for all x do
        $Bel'(x) = \eta^{-1}Bel'(x)$;
    end
end
else if d is an action data item u then then
    for all x do
        $Bel'(x) = \sum_{x'} P(x|u, x')Bel(x')$;
    end
end
Return $Bel'(x)$
```

10.2 Kalman Filter

- Kalman filter used when state space is continuous variables
- They key idea is to represent everything with *gaussians*
- Univariate and multivariate gaussians

- We stay in "Gaussian world" as long as we start with Gaussians and perform only linear transformations
- Estimate state x of a *discrete-time* controlled process governed by linear stochastic difference equation:

  $$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

  and sensor measurement

  $$z_t = C_t x_t + \delta_t$$

  where:
  x_t = current state
  A_t = matrix describing how state changes from $t-1$ to t without controls
  B_t = matrix that describes how control u_t changes the state from $t-1$ to t
  C_t = matrix that describes how to map state x_t to an observation z_t
  ϵ_t = process noise normally distributed with covariance R_t
  δ_t = measurement noise normally distributed with covariance Q_t

Algorithm 8: Kalman Filter

```
1 Prediction: use dynamics to predict what will happen;
2 $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$;
3 $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$;
4 Correction: use sensor measurement to correct prediction;
5 $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$;
6 $\mu_t = \bar{\mu}_t + K_t(z_t - C_t \bar{\mu}_t$;
7 $\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$;
8 return $\mu_t, \Sigma_t$
```

- Comments:
  - Highly efficient: only need to compute matrix multiplication
  - Optimal for linear Gaussian systems, but most robotics systems are nonlinear

10.3 Extended Kalman Filter (EKF)

- Most robotics problem deal wth nonlinear dynamics and sensors

  $$x_t = g(u_t, x_{t-1})$$

  $$z_t = h(x_t)$$

- EKF trick: use a *local linear approximation* by computing the Jacobians of g and h

  $$x_t = g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1})$$

  $$z_t = h(x_t) \approx h(\bar{\mu}_t) + H_t(x_t - \bar{\mu}_t)$$

  where
  $G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}}$
  $H_t = \frac{\partial h(\bar{\mu}_t)}{\partial x_t}$

- Comments
    - Highly efficient
    - Not optimal, because it is an approximation of the nonlinear function
    - Can diverge if nonlinearities are large (e.g. close to beacon)
    - Everything must be a Gaussian
    - Cannot be used if transition is non-linear, e.g. multimodal distributions

10.4 Unscented Kalman Filter

- Key idea:
    1. Sample a set of sigma points from Gaussian distribution
    2. Pass sigma points through function
    3. Re-estimate Gaussian

Algorithm 9: Unscented Kalman Filter

```
1 $\mathcal{X}_{t-1}$ = Compute sigma points using $\mu_{t-1}$ and $\Sigma_{t-1}$;
2 $\bar{\mathcal{X}}_t^* = g(u_t, \mathcal{X}_{t-1})$ // Apply dynamics ;
3 $\bar{\mu}_t = \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{X}}_t^{*[i]}$ // Estimate new mean from sigma points;
4 $\bar{\Sigma}_t = \sum_{i=0}^{2n} w_c^{[i]} (\bar{\mathcal{X}}_t^{*[i]} - \bar{\mu}_t)(\bar{\mathcal{X}}_t^{*[i]} - \bar{\mu}_t)^T + R_t$;
5 $\bar{\mathcal{X}}_t$ = Compute sigma points using $\bar{\mu}_t$ and $\bar{\Sigma}_t$;
6 $\bar{\mathcal{Z}}_t = h(\bar{\mathcal{X}}_t)$ // Apply sensor model;
7 TODO;
```

- Comments
    - Highly efficient
    - Better approximation than EKF: accurate in first two terms of Taylor expansion
    - Derivative-free, does not require any Jacobians

10.5 Particle Filter

- No need to make assumptions of distributions unlike Kalman Filter which assumes that all error is Gaussian
- Sample from the implicit distribution of the state at any given time
- At time t, distribution is represented by the M samples of the robot's states

$$X_t = x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]}$$

$$W_t = w_t^{[1]}, w_t^{[2]}, ..., w_t^{[M]}$$

- If starting location is know, can initialize particles around start, else scatter particles through the environment

Algorithm 10: Particle Filter

```
1  $X_0$ = Sample M particles from $P(X_0)$;
2  $t = 0$;
3  while True do
4      $t++$;
5      $u_t$ = action();
6      $z_t$ = sensor();
7      $S_t = X_t = \{\}$;
8      for $m = 1$ to $M$ do
9          sample $x_t^{[m]} \sim p(x_t|u_t, x_{t-1}^{[m]}$ from $X_{t-1})$;
10         $w_t^{[m]} = p(z_t|x_t^{[m]})$;
11         $S_t = S_t \cup (x_t^{[m]}, w_t^{[m]})$;
12     end
13     for $m = 1$ to $M$ do
14         draw $i$ with probability $\sim w_t^{[i]}$;
15         add $x_t^{[i]}$ from $S_t$ to $X_T$;
16     end
17 end
```

- Sampling strategies
  - Need to preserve diversity of particles, inject random particles
  - Low-variance sampling
  - Stratified sampling
- Comments
  - Represent distribution with a set of particles
  - Approximate arbitrary probability distributions
  - More particles = better approximation but more computation cost

Chapter 11

MDP and POMDPs