UDACITY

Artificial Intelligence for Robotics: A Brief Summary

This document provides a summary of the course, Artificial Intelligence for Robotics, and highlights main concepts.

Lesson 1: Localization (using Histogram Filters)

- **Localization** refers to the back-and-forth iteration between *sensing* and *moving* that allows us to track and maintain the position, orientation, and velocity of a target object.
    - Noise in movement will increase the uncertainty we have about position, while sensory information will tend to decrease our uncertainty.
- **Histogram filters** can be used for localization in *discretely-defined spaces*, featuring the ability to keep *multimodal* estimates of target location with memory requirements that increase *exponentially* with the number of dimensions.
- Represent our belief of the current location of the target as a probability distribution over the environment divided into a discrete grid.
- Measurement updates (sensing) are evocative of products. Here, we use Bayes Rule, which tells us the relationship between the data D we receive and the likelihood of our location X (the $\propto$ symbol below reads as "proportional to"; normalize over all locations to obtain a proper probability distribution):

The posterior probability assigned to a location x_i given observation D is proportional to the product of the likelihood of the observation given the location and the prior probability of being at that location.

- Motion updates (movement) are evocative of convolution. Here, we use the Theorem of Total Probability, which tells us the relationship between our previous position X^t at time step t and our new position X^{t+1} at the next time-step $t+1$:

Lesson 2: Kalman Filters

- **Kalman filters** can be used for localization in *continuously-defined spaces*, retaining a *unimodal* estimate of target position, orientation, and velocity with memory requirements that increase *quadratically* with the number of dimensions.

- Represent our knowledge of the current state (e.g. position, orientation, velocity) as vector $\mathbf{x}$ and the uncertainty in our state as covariance matrix $\mathbf{P}$. Assume errors are gaussian in nature.

- During measurement (update step), we record a vector $\mathbf{z}$, assumed to be the expected measurement $\mathbf{Hx}$ (where $\mathbf{H}$ is a vector mapping the truth $\mathbf{x}$ to observation $\mathbf{z}$) with measurement noise $\mathbf{R}$ (covariance matrix). We update our estimates of $\mathbf{x}$ and $\mathbf{P}$ thusly:

$\mathbf{y} = \mathbf{z} - \mathbf{Hx}$	$\mathbf{y}$: measurement residual, difference between truth and expectation
$\mathbf{S} = \mathbf{HPH}^T + \mathbf{R}$	$\mathbf{S}$: residual covariance, increased by measurement noise
$\mathbf{K} = \mathbf{PH}^T\mathbf{S}^{-1}$	$\mathbf{K}$: optimal Kalman gain, gives least squared errors in updates
$\mathbf{x}' = \mathbf{x} + \mathbf{Ky}$	
$\mathbf{P}' = (\mathbf{I} - \mathbf{KH})\mathbf{P}$	where $\mathbf{I}$ is an identity matrix.

- During movement (prediction step), state transition matrix $\mathbf{F}$ depicts the base change to our state variables $\mathbf{x}$ and $\mathbf{P}$. Changes to the system (such as changes in velocity) are enacted through control vector $\mathbf{u}$ (or $\mathbf{Bu}$, where $\mathbf{B}$ is a matrix depicting a translation from controls to effective change) and an overall process noise $\mathbf{Q}$ (covariance matrix). We update our estimates of $\mathbf{x}$ and $\mathbf{P}$ thusly:

$$\mathbf{x}' = \mathbf{F}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\mathbf{P}' = \mathbf{F}\mathbf{P}\mathbf{F}^T + \mathbf{Q}$$

Lesson 3: Particle Filters

- **Particle filters** can be used for localization in *continuously-defined spaces*, featuring the ability to keep *multimodal* estimates of target position, orientation, and velocity with memory requirements variable dependent on applications (can be close to *quadratic* or be obviously *exponential* depending on environment demands).

- Represent the belief in the state of the target as a population of particles, where each particle contains a single hypothesis regarding the target's true state.

- Measurement updates are enacted by resampling particles from the population with replacement, obtaining a new population of particles (of the same size) that is propagated to the next time step. The probability of a particle to be resampled is proportional to its importance weight, which in turn is proportional to the likelihood of making the measurement observation under the particle's held state.

- Motion updates are enacted by pushing each particle ahead one time step according to its own hypothesis. If there are changes to the system made during motion, they too are made during this step on each particle.

- It is important that there be noise in particle state, lest the population become dominated by clones of the same particle, none of which describes the system well. Noise allows for diversity in hypothesis, and the particles with the best-fitting hypotheses will be carried to the next generation and the worst-fitting particles will die out.

Lesson 4: Search

- Suppose we are posed with a motion planning problem, where we wish to navigate from an origin location to a goal location, given a map and cost function for actions and terrain. Here, we model the map as a grid or connected graph with costs for traveling between cells or nodes.

- **Dijkstra's Algorithm** guarantees a least-cost path through tracking of $g(x)$, the minimum cost from the origin to a location x. The algorithm recursively selects the cell in the "open" (not yet visited) set with the smallest $g(x)$ and assigns a tentative $g(x)$ to its neighbors. The searched cell is added to a "closed" (visited) set and the algorithm continues selecting least-cost "open" cells until the goal is reached. If we track the actions taken to move into each node, we can reverse the actions to get an optimal path. If the weights on actions are flat, then the algorithm becomes a case of **Breadth-first search**.

- The **A* Algorithm** builds on Dijkstra's algorithm by supplementing $g(x)$ with a heuristic function $h(x)$ that estimates the distance from location x to the goal. The algorithm uses the sum of these functions $f(x) = g(x) + h(x)$ to decide which "open" cell to expand next in sequence. If $h(x)$ is always less than or equal to the actual cost to reach the goal from x, then the A* Algorithm will always find the optimal solution. (If the heuristic function is $h(x) = 0$, then we simply have Dijkstra's algorithm.)

- If we wish to create an optimal policy for all points to move to the goal, rather than just a single origin point, we can use **dynamic programming**. Here, we start at the goal and move outwards, recursively computing a value function for each cell until all cells have been filled. As noted before, by tracking actions and reversing the process, we can obtain the optimal policy for each cell.

Lesson 5: PID Control

- To translate a path generated in a discrete domain into a continuous domain, we use smoothing. From an original set of points X we want to obtain a new set of points Y such that we minimize (for a smoothing parameter α):

- To solve this using gradient descent (for a non-cyclic path and no constraints except for endpoint positions), we iterate over the following update equations until the overall change in Y positions has converged below a specified tolerance or for a specified number of iterations (using weights α and β):

;

- A **P-controller** allows us to adjust for the difference between our actual line of movement from the desired line (crosstrack error) by providing compensatory steering proportional to the size of the error. The P-controller is, at best, marginally stable, creating movement that oscillates about the target line.

- The **PD-controller** adds a component to the P-controller that takes into account the temporal derivative (approximated by the difference in errors taken at consecutive timesteps) and attempts to stabilize the system as it gets close to the target line.

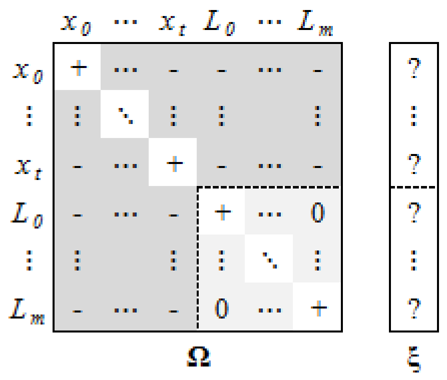
- The **PID-controller** further refines our control by adding an integral component that takes into account the integral of all previous errors (approximated by the sum) and helps compensate for systematic bias in the system. A generic formula for the complete controller is as follows (with gain parameters τ_p, τ_i, τ_d):

- The twiddle algorithm (coordinate ascent) is used to search for good gain parameter values. Twiddle is a variation on hill-climbing which cycles through the parameter values, selecting locally beneficial values until they converge to a local minimum or for a specific number of iterations. On each step of the algorithm, for parameter value p_i and change parameter Δp_i:

take p_i' as the smallest of:	$p_i - \Delta p_i$	p_i	$p_i + \Delta p_i$
then take Δp_i' (for $a<1<b$):	$b * \Delta p_i$	$a * \Delta p_i$	$b * \Delta p_i$

Lesson 6: SLAM (Simultaneous Localization And Mapping)

- **GraphSLAM** is based on a matrix $\mathbf{\Omega}$ and vector ξ that depict constraints between expected locations $\mathbf{x}$ and environmental landmarks $\mathbf{L}$:



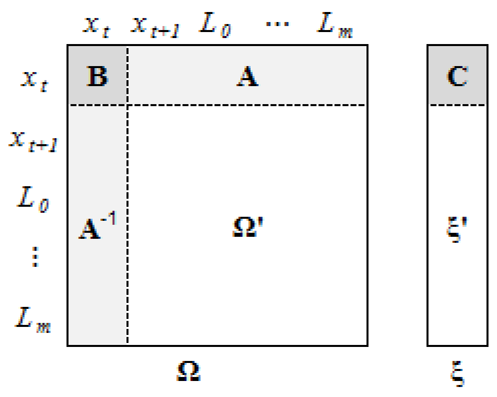
- In a movement, the movement from time t-1 to t will modify the cell values in the intersections of rows corresponding to locations $x_{t\text{-}1}$ and x_t (including cells on the diagonal) and the values in the vector associated with the same locations. Similarly, sensing of landmark L_i at time x_t modifies the matrix values and vector values associated with those entities according to the constraints.

- Constraints are local and additive, where confidence in a constraint is depicted with a weight, e.g. 1/σ where σ is movement or sensory noise.

- Our best estimate of $\mathbf{x}$ and $\mathbf{L}$ positions is $\boldsymbol{\mu} = \boldsymbol{\Omega}^{-1}\xi$. With each new movement and sense measurement, we update $\boldsymbol{\Omega}$ and ξ, then update our plan of action, map of environment, etc.

- Memory costs in GraphSLAM may become prohibitively expensive, even with sparse matrices, as time grows long. If our map does not grow much, **Online SLAM** can save on space by only retaining the most recent location. For an $\boldsymbol{\Omega}$ and ξ whose rows and columns are ordered by $\{x_t, x_{t+1}, L_0, \ldots, L_m\}$, update $\boldsymbol{\Omega}$ and ξ by:

$$\boldsymbol{\Omega} \leftarrow \boldsymbol{\Omega}' - \mathbf{A}^{\mathsf{T}}\mathbf{B}^{-1}\mathbf{A};\ \xi \leftarrow \xi' - \mathbf{A}^{\mathsf{T}}\mathbf{B}^{-1}\mathbf{C}, \text{ where:}$$


x_t
x_{t+1}
L_0
$\cdots$
L_m
x_t
x_{t+1}
L_0
$\vdots$
L_m
$\mathbf{B}$
$\mathbf{A}$
$\mathbf{A}^{-1}$
$\mathbf{\Omega}'$
$\mathbf{C}$
ξ'
$\mathbf{\Omega}$
ξ