Car tracking

omar maaref
https://github.com/omarmaaref/Car-Tracking

A study by the World Health Organization found that road accidents kill a shocking 1.2 million people a year worldwide. In response, there has been great interest in developing

autonomous driving technology that can drive with calculated precision and reduce this death toll. Building an autonomous driving system is an incredibly complex endeavor.

Objectif :

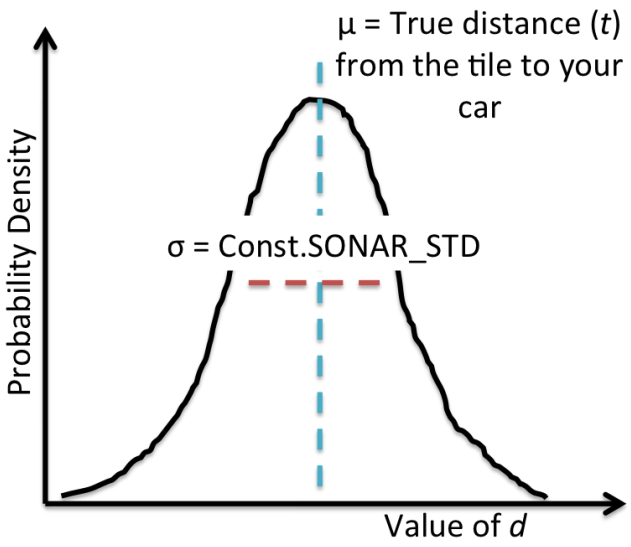
In this project, **I will focus on the sensing system**, which allows us to track other cars based on noisy sensor readings.

Modeling :

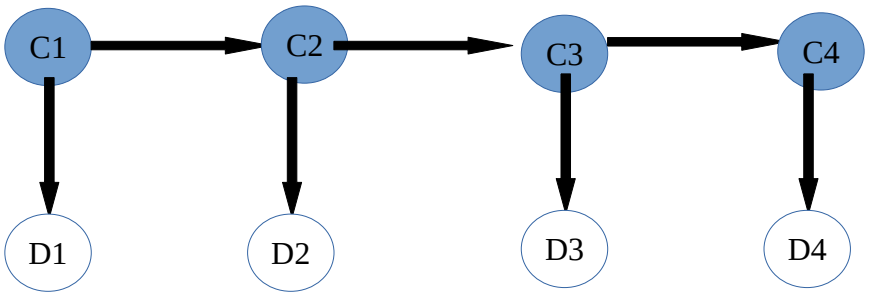
We assume that the world is a two-dimensional rectangular grid on which your car and K other cars reside. At each time step t, your car gets a noisy estimate of the distance to each of the cars. As a simplifying assumption, we assume that each of the K other cars moves independently and that the noise in sensor readings for each car is also independent.

At each time step t, let $C_t \in R^2$ be a pair of coordinates representing the actual location of the single other car (which is unobserved). We assume there is a local conditional distribution $p(c_t | c_{t-1})$ which governs the car's movement. Let $a_t \in R^2$ be your car's position, which you observe and also control. To minimize costs, we use a simple sensing system based on a ***microphone*** . The microphone provides us with D_t, which is a Gaussian random variable with mean equal to the true distance between your car and the other car and variance σ^2 (in the code, σ is Const.SONAR_STD, which is about two-thirds the length of a car). In symbols,

$$D_t \sim N(\| a_t - C_t \|_2, \sigma^2).$$



our model’s representation (Bayesian network)



$P(C_t \mid D_1=d_1,\ldots,D_t=d_t) \propto P(C_t \mid D_1=d_1,\ldots,D_{t-1}=d_{t-1})p(d_t \mid c_t)$

$p(C_{t+1}=c_{t+1} \mid D_1=d_1,\ldots,D_t=d_t) \propto \sum p(C_t=c_t \mid D_1=d_1,\ldots,D_t=d_t)p(c_{t+1} \mid c_t).$

Making inference using particle filter (1):

Though exact inference works well for the small maps, it wastes a lot of effort computing probabilities for *every available tile*, even for tiles that are unlikely to have a car on them. We can solve this problem using a particle filter. Updates to the particle filter have complexity that's linear in the number of particles, rather than linear in the number of tiles.

This algorithm takes two steps:

Step 1 : Proposal .

Proposal based on the particle distribution at current time t.

Concept: We have a particle distribution at current time t, and we want to propose the particle distribution at time t+1. We would like to sample again to see where each particle would end up using the transition model.

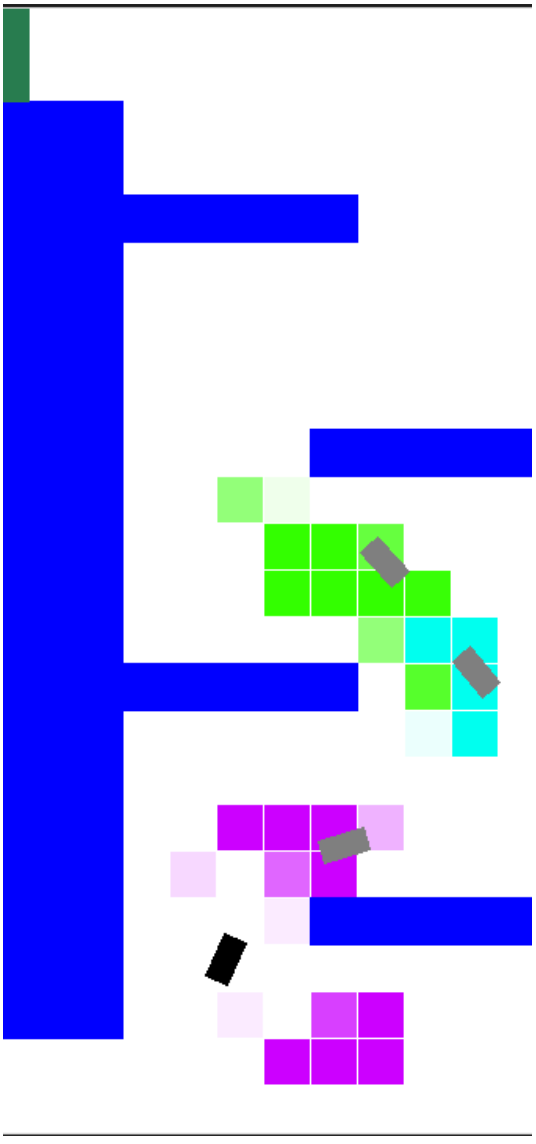
Step 2 : Re-weight.

Re-weight the particles based on the observation. Concept: We had an old distribution of particles, and now we want to update this particle distribution with the emission probability associated with the observed distance. Think of the particle distribution as the unnormalized posterior probability where many tiles would have 0 probability. Tiles with 0 probabilities (i.e. those with no particles) do not need to be updated. This makes particle filtering runtime to be $O(|\text{particles}|)$. By comparison, the exact inference method.

#(1):Particle filters or Sequential Monte Carlo methods are a set of Monte Carlo algorithms used to solve filtering problems arising in signal processing and Bayesian statistical inference.

Step 3 : Re-sample.

Re-sample the particles. Concept: Now we have the reweighted (unnormalized) distribution, we can now re-sample the particles from this distribution, choosing a new grid location for each of the NUM_PARTICLES new particles. To be extra clear: these new NUM_PARTICLES should be sampled from the new re-weighted distribution, not the old belief distribution, with replacement so that more than one particle can be at a tile



[] https://stanford-cs221.github.io/spring2021/
[] https://en.wikipedia.org/wiki/Particle_filter
[] https://en.wikipedia.org/wiki/Bayesian_network