Introduction to Computational Traffic Models

Ramona Stefanescu

Samsung Semiconductors, Inc.

October 18, 2018

Traffic Models

There are 4 main compenents in modeling the road traffic:

- Route calculation
- Prediction
- Behavior planning
- Trajectory generation

Outline

Route calculation

In calculating the route from a given location to another location we can use one the following methods:

- Graph-based Planning Methods
- Configuration Space Methods
- Probabilistic Road Maps Methods
- Artificial Potential Field Methods

Graph-based Plan Methods

- A graph, G, consists of a set of vertices, V, and a set of Edges, E, that link pairs of vertices.
- The goal is to construct a path through the grid/graph from the start to goal.
- Find path tha minimizes the cost function.

Grassfire

- Identify the "STOP" position and mark it with 0
- Starting from 0 add 1 for each cell away from the starting position
- The distance values produced by the grassfire algorithm indicate the smallest number of steps needed to move from each node to the goal

Grassfire algorith - pseudo code

- For each node n in the graph
  - n.distance = infinity
- Create an empty list
  - n.goal = 0, add goal to list
- while list not empty
  - Let current = first node in list, remove current from list
  - For each node, n that is adjacent to current
    - if n.distance=infinity
    - n.distance = current.distance + 1
    - add n to the back of the list

Grassfire Algorithm

- It will find the shortest path between the start and the goal if one exists
- If no path exists that will be reported
- The computational effort required to run the grassfire algorithm on a gris increases linearly with the number of nodes
- $\mathcal{O}(|\mathcal{V}|)$, where $|V|$ is the no of nodes

Dijkstra's Algorithm

- For each node n in the graph: n.distance = infinity
- Create an empty list
- Start.distance = 0, add start to list
- While list not empty
  - Let curent = node in the list with the smallest distance, remove current from list
  - For each node, n that is adjacent to current
    - if n.distance >current.distance + length of edge from n to current then:
    - n.distance = current.distance + length of edge from n to current then
    - n.parent = current
    - add n to list if it isn't there already

Computational Complexity of Dijkstra's algorithm

1. A naive version of Dijkstra's can be implemented with a computational complexity that grows quadratically with the number of nodes:

$$\mathcal{O}(|V|^2) \tag{1}$$

2. Using clever structures known as **priority queue** the computational complexity can be reduced to something that grows more slowly:

$$\mathcal{O}((|E| + |V|)log(|V|)) \tag{2}$$

Grassfire and Dijkstra's Alg

- When applied on a grid graph where all of the edges have the same length, Dijkstra's algorithm and the grassfire procedure have similar behaviors.
- They both explore nodes in order based on their distance from the starting node until they encounter the goal.
- Both explore all or almost all nodes and this can be a problem when you have large number of nodes.

A^*

- A^* is an example of broader class of procedure *best first search* algorithms.
- They explore a set of possibilities by using an approximating heuristic cost function to sorth the varies alternatives
- Similar to grassfire and Dijkstra. These two explore evenly in all directions until they find the goal node.
- A^* introduces the idea of heuristic distance, which is an estimate between a given node and the goal node. We can use this information to gide the search in to direction that can get us to goal faster.

Heuristic functions

- For **shortest path problems**, we're interested in heuristic function H, which given a node n return a non-negative value that is indicative of the distance from that node to the goal.
- For **path finding problems**, we often use heuristic functions that are monotonic, which means that they satisfy the following properties:
  1. H(goal) $= 0$
  2. For any 2 adjacent nodes x and y: $H(x) <= H(y) + d(x, y)$ and $d(x, y)$ denotes the length of the edge between those two nodes x and y.

Example Heuristic Functions

- Euclidean Distance

$$H(x_n, y_n) = \sqrt{((x_n - x_g)^2 + (y_n - y_g)^2)} \tag{3}$$

- Manhattan distance

$$H(x_n, y_n) = |x_n - x_g| + |y_n - y_g| \tag{4}$$

A^* pseudo code

- For each node n in the graph: n.f = infinity, n.g = infinity
- Create an empty list
- start.g = 0, start.f = H(start) add start to list
- While list not empty:
  - Let current = node in the list with the smallest f value, remove current from the list
  - If (current == goal node) report succes
  - For each node, n that is adjacent to current
    - if (n.g > current.g + cost of edge from n to current))
    - n.g = current.g + cost of edge from n to current
    - n.f = f.g + H(n)
    - n.parent = current
    - add n to list of it isn't there already

Configuration Space

- The motion planning problems we've considered so far, we've reduced the problem to planning on a graph, where the robot can take on various discrete positions.
- The configuration space of a robot is the set of all configurations and/or positions that the robot can attain.
- The region of configuration space that the robot can attain is referred to as the free space of the robot.

Path Planning in Configuration Space

- Start the motion planning problem framed on a continuous configuration space and then use various approached to reformulate this problem in terms of a graph.

Collision Detection Function

- Let x denote the coordinates of a point in configuration space
- CollisionCheck(x) should return 0 if x is in freespace and 1 if x results in a collision with the obstacles.

Probabilistic Road Map

- The approach of discreetizing the configuration space evenly on the grid, can work well when the space is small, say two or three. But the number of samples required can grow to be freigheningly large as we increase the dimension of the space to five, six, ten etc.
- **Pseudocode**
  - Repeat n time
  - $\rightarrow$ Generate a random point in configuration space, x
  - $\rightarrow$ If x is free then:
    - Find the k closest points in the roadmap to x according to the **Dist** function
    - Try to connect the new random sample to each of the k neighbors using the **LocalPlanner** procedure. Each succesful connection forms a new edge in the graph.

The Dist function

- The PRM procedure relies upon a distance function, Dist, that can be used to gauge the distance between two points in configuration space.
- Common choices for distance function include:
  - The L1 distance: $Dist_1 = \sum_i |x_i - y_i|$
  - The L2 distance: $Dist_2 = \sqrt{\sum_i (x_i - y_i)^2}$
- A complete path planning algorithm would fina path if one existed, and report failure if it didn't. i
- With the PRM procedure, it is possible to have a situation where the algorithm would fail to find a path even when one exist.

PRM

- In practice, the number of samples that one chooses to generate for the road map is an important parameter of this procedure.
- One idea is to try to sample more points closer to the bounderies of configuration space obstacles.
- To date however there is no single sampling strategy that is guaranteed to work well in all cases.
- Since samples are choosen randomly the resulting trajectory can sometimes seem very jerky and unnatural.
- Use a path smoother
- By relaxing the notion of completeness a bit and embracing the power of randomization, PRM provide effective methods for planning routes that can be applied to a wide range of robotic systems.

Rapidly Exploring Random Tree (RRT)

- In the probabilistic roadmap procedure, the basic idea was to construct a roadmap of the free space consisting of random samples and edges between them.
- Once that has been constructed, you could connect the desired start and end points to this graph and plan a path from one end to the other.
- The advantage of this approach is that you can reuse the roadmap over and over again to answer multiple planning problems.
- There are times when you're interested in answering one specific planning problem.
- In those situations, it can be wasteful to construct roadmaps that span the entire free space.

Rapidly Exploring Random Tree (RRT)

- Like the probabilistic road map technique, this approach works by generating random samples and connecting them together to form a graph, but the sampling scheme is a bit different.
- The RRT procedure proceeds by constructing a special kind of graph called a tree, where every node is connected to a single parent and the tree is rooted at a given starting location.
-

RRT procedure

- Add start node to tree
- Repeat n times
- $\rightarrow$ Generate a random configuration, x
- $\rightarrow$ If x is in freespace using the **CollisionCheck** function
  - Find y, the closest node in the tree to the random configuration x
  - If $(Dist(x, y) > delta)$ – check if x is too far from y
  - $\rightarrow$ find a configuration, z, that is along the path from x to y such that $Dist(z, y) <= delta$
  - $\rightarrow$ x=z
  - If $(LocalPlanner(x, y))$ –check if you can get from x to y
  - $\rightarrow$ Add x to the tree with y as its parent

RRT 2 tree procedure

- While not done
  - Extend Tree A by adding a new node, x
  - Find the closest node in Tree B to x,y
  - If (LocalPlanner(x,y)) –check if you can bridge the 2 trees
  - $\rightarrow$ add edge between x and y
  - $\rightarrow$ this completes a route between a root of Tree A and the root of Tree B. Return this route
  - c$\rightarrow$ Else: Swap Tree A and Tree B

Artificial Potential Field Methods

- This is another approach to guiding robots into obstacle filled environments based on artificial potential fields.
- The basic idea here is to try to construct a smooth function over the extent of the configuration space, which has high values when the robot is near to an obstacle and lower values when it's further away.
- We also want this function to have its lowest value at the desired goal location
- And it's value should increase as we move to configurations that are further away.
- If we can construct such a function, we can use it's gradient to guide the robot to the desired configuration.

Potential function

- There are many ways to build *potential function* but one of the most popular is to construct a simple quadratic function, that is zero at the goal and increases as you move away from it.
- $f_a(x)$ - an attractive potential function, $x = (x_1, x_2)^T$ the current position of the robot and $x_g = (x_1^g, x_2^g)^T$ the desired goal can be defined as follows:

$$f_a(x) = \xi(||x - x_g||^2) \tag{5}$$

where ξ is simply a constant scaling parameter

Repulsive Potential Field

- In addition to getting to the goal we also want the robot to avoid the obstacles in the environment
- Use a second funtion to repel the robot from these configuration space obstacles
- Is constructed based on a function, $\rho(x)$, that returns the distance to the closest obstacle from a given point in configuration space, x.

$$f_r(x) = \begin{cases} \eta(\frac{1}{\rho(x)} - \frac{1}{d_0})^2 & \text{if } \rho(x) \leq d_0 \\ 0 & \text{if } \rho(x) > d_0 \end{cases} \tag{6}$$

- Where ρ is simply a constant scaling parameter and d_0 is a parameter that controls the influence of the repulsive potential
- This function has been used sucessfully in image processing - called **distance transformation**

- We have two components: **an attractive potential**, which pulls the robot towards the desired configuration, and **a repulsive potential**, which steers the robot away from obstacles
- A really attractive feature of this procedure is that it only requires a robot to be able to evaluate the gradient of this artificial potential function.
- This is sometimes reffered to as sensor based planning. The reading from the robot's position sensor are used to pull it towards the goal. While the measurement from it's range sensors like laser scaners or stereo system are used to push it away from the obstacles in the environment.

Outline

Prediction

- A prediction module uses a map and data from sensor fusion to generate predictions for what all other dynamic objects in view are likely to do.
- Approaches:
  1. Model Based Approach: knowledge about physics, constrains impose by the road, traffic etc
  2. Data-Driven Approach: used data to extract model

Model-Based Approach

For each **dynamic object** nearby

- Identify common driving behaviors (change lane, turn left, cross street, etc)
- Define process model for each behavior (is a mathematical model that computes the state of the object at time t+1)
- The process model will incorporate uncertain states
- Update beliefs by comparing the observation with the output of the process model
- Trajectory generation

Frenet Coordinates

- Frenet Coordinates are ways of representing position on a road in a more intuitive way than traditional (x, y) Cartesian Coordinates.
- With Frenet coordinates, variables s and d to describe a vehicle's position on the road.
- The s coordinate represent distance along the road (also known as **longitudinal displacement**) and the d coordinate represents side-to-side position on the road (also known as **lateral displacement**)

Process Models for lane following

There is a trade-off between simplicity and accuracy when choosing a process model. One very simple approach is to treat the car as a point particle with holonomic properties (the point can move in any direction at any time).

- **Linear point model - constant velocity** (Frenet coordinates: assumes to keep a constant distance to the lane center)

$$\begin{bmatrix} \dot{s} \\ \dot{d} \end{bmatrix} = \begin{bmatrix} \dot{s_0} \\ 0 \end{bmatrix} + W \tag{7}$$

- **Non-linear point model** (constant acceleration with curvature in Cartesian coordinates)

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\omega} \\ \dot{a} \\ \dot{c_0} \\ \dot{c_1} \end{bmatrix} = \begin{bmatrix} (v + at)cos(\theta) \\ (v + at)sin(\theta) \\ w \\ a \\ 0 \\ 0 \\ vc_1 \\ 0 \end{bmatrix} + W \tag{8}$$

Process Models for lane following

- **Kinematic Bicycle Model with controller** (PID controller on distance and angle)

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \frac{v}{L}\tan\delta \\ a \end{bmatrix} + W \tag{9}$$

- The vehicle is a non-holonomic system
- Take two inputs: a steering angle and acceleration
- For the steering angle we could use a PID controller with the targer lane centerline as the refrence line:

$$\delta_t = -J_P CTE - J_D C\dot{T}E - J_I \sum_{i=0}^{t} CTE_i \tag{10}$$

  where CTE is the *Cross Track Error*
- For acceleration we can use a constant velocity model or a constant acceleration model. If we want a more complex accleration behavior we could use a PID controller with the speed limit as the target.

Process Models - cont

- Dynamic bicycle model with controller (PID controller on distance and angle)

$$\begin{bmatrix} \ddot{s} \\ \ddot{d} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{\theta}\dot{d} + a_s \\ \dot{\theta}_{\dot{s}} + \frac{2}{m}(F_{c,f}cos\delta + F_{c,r}) \\ \frac{2}{I_z}(l_f F_{c,f} - l_r F_{c,r}) \end{bmatrix} + W \tag{11}$$

- $F_{c,f}$ represents the lateral force on the tires at the front of the vehicle, while $F_{c,r}$ represents the lateral force on the rear tire. We can add more complexity by modeling the four wheels of the car.
- While this model is a more accurate representation of the physics, in practice using it doesn't usually make sense for prediction since there is so much uncertainty inherent to predicting the behaviors of all drivers, that minor accuracy improvement to process models just aren't worth the computational overhead that they come with.

Autonomous Multiple Model Algorithm - variables

A simple approach to multimodal estimation is called **Autonomous Multiple Model Estimation or AMM**.

- Consider some set of M process models/behaviors
- Probabilities for process models $\mu_1, \mu_2, \ldots, \mu_M$
- Nultiple model algorithms serve a similar purpose for model based approaches: they are responsible for maintaining beliefs for the probability of each maneuver.
- AMM cam can ve summarized with the following equation:

$$\mu_k^{(i)} = \frac{\mu_{k-1}^{(i)} L_k^{(i)}}{\sum_{j=1}^{M} \mu_{k-1}^{(j)} L_k^{(j)}} \tag{12}$$

AMM -cont

- If we ignore the denomitor in the previous eq (since it just serves to normalize the probabilities), we can capture the essence of the algorithm with:

$$\mu_k^{(i)} \propto \mu_{k-1}^{(i)} L_k^{(i)} \tag{13}$$

where the $mu_k^{(i)}$ is the probability that model number i is the correct model at time k and $L_k^{(i)}$ is the **likelihood** for that model (as computed by comparison to process model).

Trajectory generation

- Trajectory generation is straightforward once we have a process model.
- We iterate our model over and over until we generated a prediction that spans whatever time horizon we are supposed tp cover.
- Note that each iteration of the process model will necessarily add uncertainty to our prediction.

Data-Driven Approach - Trajectory

There are 2 types of learing: offline and online

- The off-line learning consistents of the following steps:
  1. Get a LOT of trajectories
  2. Clean the data
  3. Define some measure of similarity
  4. Perform unsupervised clustering
  5. Define **prototype trajectories** for each cluster

Online Trajectory Prediction

For every update cycle:

1. Observe vehicle's partial trajectory
2. Compare to **prototype trajectories**
3. Predict a trajectory

Outline

Behavior Planning

- Is part of the path planning module, along with Prediction and Trajectory
- The Behavior Planning takes as input a map of the world, route to destination and predictions about what other static and dynamic are likely to do
- It produces as output an adjusted maneuver for the vehicle which the trajectory planner is responsible for reaching collision-free, smooth and safe
- The responsability of the Behavior Planner is to suggest maneuvers which are: feasible, as safe as possible, legal and efficient.
- The Behavior Planner is not responsible for: execution details and collision avoidance.

Outline

Trajectory generation

- A trajectory is not just a curve that the car can follow but also a time sequence in which we say how fast the car should go.
- In finding a trajectory we need to account for both non-colission as well passenger comfort
- Continous trajectories $->$ drivable trajectories

The motion planning problem

Configuration space:

- Defines all possible configurations of our robot in a given world
- Configuration space can be 2D [x,y] or 3D [x,y,θ]
- Given: an initial configuration with a start and a goal, as well constrains describing how the vehicle is allowed to move, its dynamics and a description of the environment

Usually, the start configuration is the current configuration given to us by the localization module and sensors(car location, speed, acceleration etc). The behavior layer gives the desired end configuration and maybe some constraints regarding where to go and at which speed. Finally, prediction completes this problem by giving us information about how the obstacle region will evolve in time. This way, the sequence of actions that we generate takes into account other vehicles and pedestrian actions.

Types of Motion Planning Alg

1. **Combinatorial methods**
   - Consist in diving the free space into small pieces and solve the motion planning problem by connecting these atomic elements
   - They are very intuitive way to find initial approximate solution but they usually do not scale well for large environments
2. **Potential fields**
   - Are reacting methods - each obstical is going to create an anti-gravity field which makes it harder for the vehicle to come close to it
   - The main problem with most potential fields methods is that they sometimes push us into local minima which can prevent us from finding a solution

Types of Motion Planning Alg - cont

3. **Optimal Control**
   - Consists in trying to solve the motion planning problem and the control inputgeneration in one algorithm
   - Using a dynamic model of a vehicle, start configuration and end configuration, we want to generate a sequence of inputs, for example, steering angle and throttle inputs, that would lead us from start to end configuration while optimizing a cost function relative to the control inputs such as minimizing gas consumption. And, relative to the configuration of the car, optimizing the distance from other vehicles.
   - Most of them are based on numerical optimization methods
   - However, it is hard to incorporate all of the constraints related to the other vehicles in a way for these algorithms to work fasr
4. **Sampling Based Methods**
   - Require a somewhat easier to compute definition of the free space

Sampling based methods

- Use a collision detection module that probes the free space to see if a configuration is in collision or not
- Unlike, combinatorial or optimal control methods, which analyze the whole environment, not all parts of the free space need to be explored in order to find a solution
- Explored parts are stored in a graph structure that can be searched with a graph search algorithm like Dijkstra or A star
- Two main classes of methods can be identified as sampling based: **discrete methods**, which rely on a finite set of configurations and/or inputs (like a grid superimposed over our configuration space). Deterministic graph search algorithms: A^*, D^*, Dijkstra
- **probabilistic methods:** - performs a random exploration of the configuration and input space. The set of possible configurations or states that will be explored is potentially infinite. Gives some of these methods the nice property that they are probabilistic complete and sometimes probabilistic optimal (meaning that they will always find a solution if you allow them enough computation time). Algorithms: RRT, PRM etc

Hybrid A^*

- The world is continuous, compared to A^* where the world was discrete
- It uses an optimistic heuristic function guide grid cell expansion
- It does not always finds a solution, if when one exists
- The solution it finds is guaranteed to be driveable
- The solutions it finds are not always optimal

With hybrid A^*, we have lost completeness and optimality guarantees in favor of drivability
In practice, however this algorithm is efficient at finding good path almost all the time