Trajectory Generation Using Cubic Curvature Polynomials

Yuliang Li

2017-4

Clemson University

Index

- Trajectory Generation Problem
- Method for trajectory generation
- Experiments & Examples
- Future work
- References

Before we start

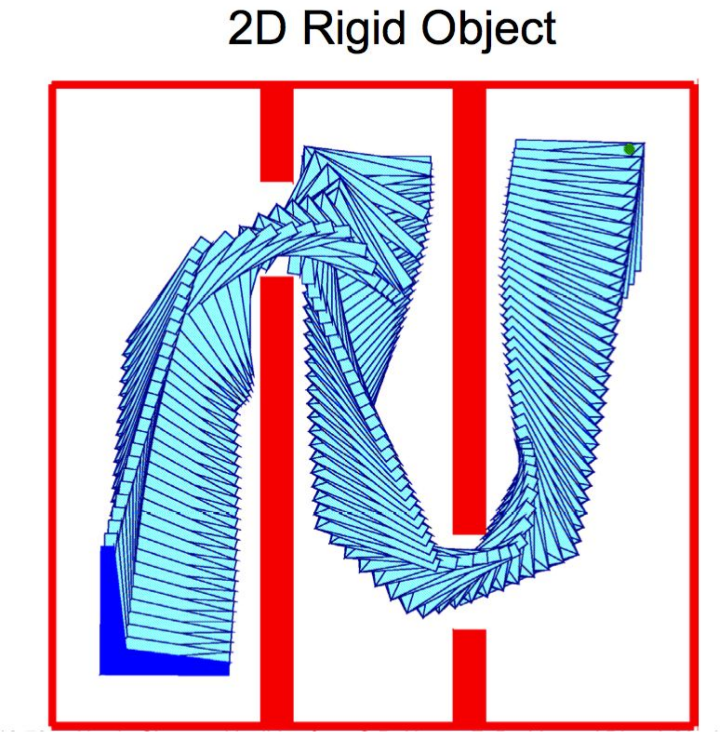
History of this algorithm

- 2001, Introduced by Nagy, Bryan, and Alonzo Kelly.
- 2007, Boss the autonomous vehicle, CMU (winner of DARPA ’ 07)
- 2011, McNaughton’s PhD Thesis, CMU
- 2015, Open-Source project CPFL-Autoware

Trajectory Generation Problem

- **Configuration (C) Space** of A is the space of all possible configurations of A.

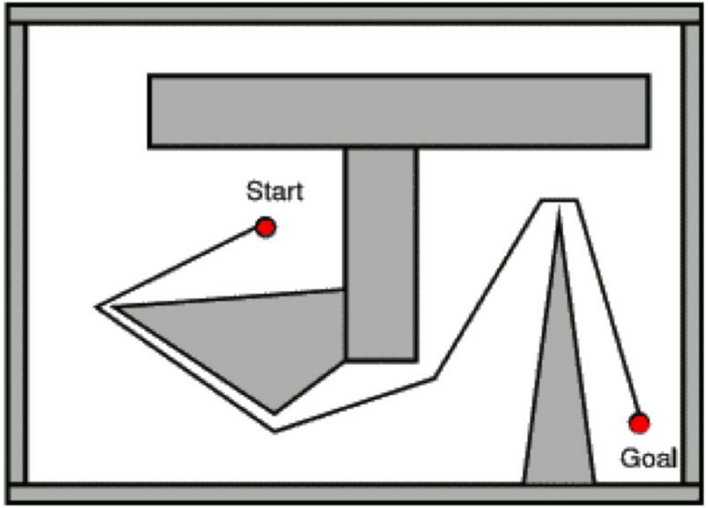
- **Motion Planning**: How to move a robot from an initial configuration to a goal configuration in its C Space.



Pic: [5] Choset

Trajectory Generation Problem

- Inverse Kinematics Problem:
  - determining the control input which causes the vehicle to achieve a goal posture (x,y,θ,κ).
- Why Curvature κ:
  - steering mechanism has mass and is subject to restoring forces that depend on vehicle state and terrain.
- The Task:
  - Solve an underdetermined differential equation for a vector trajectory which achieves some unknown state trajectory that ends at the goal posture.
- Difficults:
  - two input: Linear Velocity & Steer Angle Velocity V.S 4 outputs (x,y,θ,κ).
  - steering actuator moves continuously V.S controllers which ask it to move discontinuously

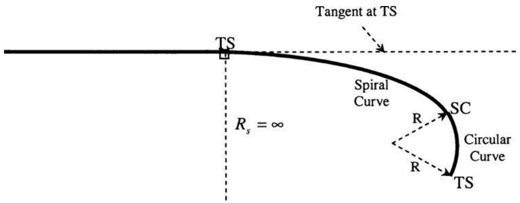


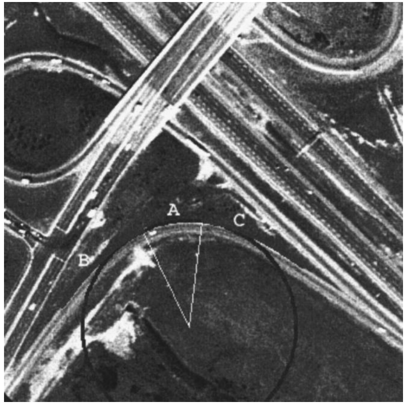
[4] Nagy & Kelly

Trajectory Generation Problem

- Polynomial Spirals:
  - The curve is a generalization of the Euler's spiral, for which the curvature is linearly related to the arc length s.
- Cubic Solution:
  - Consider the solution to the state equations for a curvature input which is a third order polynomial in arc length

$$\kappa(s) = \kappa_0 + as + bs^2 + cs^3$$





Pic: Google Image Search

Trajectory Generation Problem

- Benefits:
  - Variability: 4 params $[a,b,c,s]^T$ to determine 4 state (x,y,θ,κ), so it has sufficient degrees of freedom
  - Feasibility:Such inputs are continuous in the third derivative of steering angle and hence are smooth in the torque applied to the steering actuator.
  - Feasible Controls: Cubics cover the set of all curve. It therefore approximate the optimal control signal very well.

κ: curvature
s: arc length

Recall the vehicle posture (x,y,θ,κ).

$$\kappa(s) = \kappa_0 + as + bs^2 + cs^3 \tag{2}$$

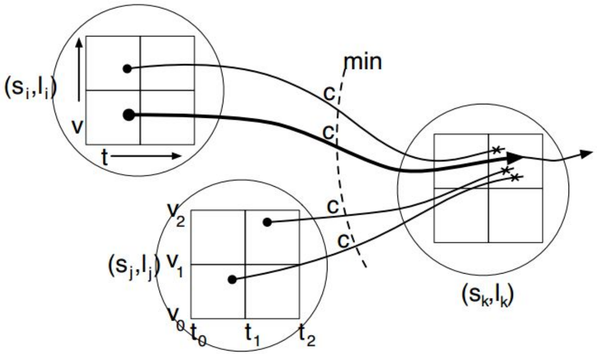
$$\theta(s) = \theta_0 + \int_0^s \kappa(s) = \theta_0 + \kappa_0 s + \frac{as^2}{2} + \frac{bs^3}{3} + \frac{cs^4}{4}$$

$$x(s) = x_0 + \int_0^s \cos(\theta(s)) \qquad y(s) = y_0 + \int_0^s \sin(\theta(s))$$

[4] Nagy & Kelly

Trajectory Generation Problem

- Solution:
  - Using cubic curvature to connect the initial vehicle config and goal config.



Pic: [3] McNaughton

What’s Next

- Steps:
  1. Define models for vehicle, path and trajectory
  2. Solve the optimization problem for path planning
  3. Generate trajectories based on paths
  4. Apply cost functions to candidate trajectories
  5. Output final control signals

Models

- Vehicle:
  - config of posture (x,y,θ,κ)
- Path:
  - A path is a continuous function ρ mapping the interval [0,1] into a C Space.
  - $\rho:[0,1] \rightarrow C$
  - $\kappa(s) = \kappa_0 + \kappa_1 s + \kappa_2 s^2 + \kappa_3 s^3$
- Trajectory:
  - A new space $M = \{(x, y, \theta, \kappa, t, v)\}$.
  - Add time the velocity to the path and map the interval into the new space
  - $\tau:[0,1] \rightarrow M$
- Curvature:
  - Assume the vehicle is moving parallel to the road
  - Calculated from waypoints

[3] McNaughton

What's Next

- Steps:
  1. Define models for vehicle, path and trajectory
  2. Solve the optimization problem for path planning
  3. Generate trajectories based on paths
  4. Apply cost functions to candidate trajectories
  5. Output final control signals

Solve the optimization problem

- Revise the curvature equation
  - original: $\kappa(s) = \kappa_0 + \kappa_1 s + \kappa_2 s^2 + \kappa_3 s^3$
  - new: $\kappa(s) = a(p) + b(p)s + c(p)s^2 + d(p)s^3$, in which $p = \{p_0, p_1, p_2, p_3, s_g\}$
- Assume params are equal to the path curvature at equally spaced points along the path.

$$\begin{aligned}
\kappa(0) &= p_0 \\
\kappa\left(\frac{s_G}{3}\right) &= p_1 \\
\kappa\left(\frac{2s_G}{3}\right) &= p_2 \\
\kappa(s_G) &= p_3,
\end{aligned}$$

$$\begin{aligned}
a(\mathbf{p}) &= p_0 \\
b(\mathbf{p}) &= -\frac{11p_0 - 18p_1 + 9p_2 - 2p_3}{2s_G} \\
c(\mathbf{p}) &= \frac{9(2p_0 - 5p_1 + 4p_2 - p_3)}{2(s_G)^2} \\
d(\mathbf{p}) &= -\frac{9(p_0 - 3p_1 + 3p_2 - p_3)}{2(s_G)^3}.
\end{aligned}$$

$p_0 = k_0$, $p_3 = k_{sg}$

- Leave us only three unknown params $p=\{p_1, p_2, s_g\}$

[3] McNaughton

Solve the optimization problem

- What we have now:
  - Equation: $\kappa(s) = a(p) + b(p)s + c(p)s^2 + d(p)s^3$, in which $p = \{p_0, p_1, p_2, p_3, s_g\}$
  - init state: κ_{init} , goal state: κ_{goal}
  - unknown params $p=\{p_1, p_2, s_g\}$
- Solve the inverse kinematics problem
- Solution: Jacobian inverse technique and Newton’s Method

Solve the optimization problem

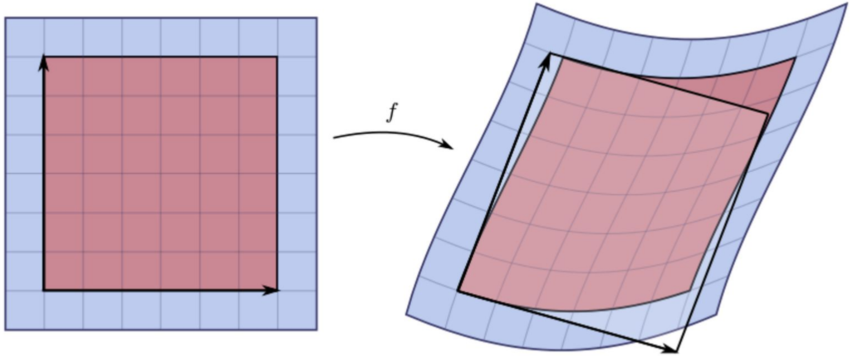
What is Jacobian:

Jacobian matrix is the matrix of all first-order partial derivatives of a vector-valued function.

The linear map $J_f(p)$ is the best linear approximation of f near point p for x close to p.

Taylor Series: $f(x) = f(p) + f'(p)(x - p) + o(x - p).$

Jacobian Matrix: $\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{p}) + \mathbf{J_f}(\mathbf{p}) \cdot (\mathbf{x} - \mathbf{p}) + o(\|\mathbf{x} - \mathbf{p}\|)$



A nonlinear map f : R2 → R2 sends a small square to a distorted parallelogram close to the image of the square under the best linear approximation of f near the point.

Pic: Wikipedia

Solve the optimization problem

- Newton’s Method(Gradient Descent):
  - 0. initial guess the params
  - 1. calculate Jacobian for estimation state
  - 2. calculate step change for new params
  - 3. update the params
  - Loop 1-3 until converge or reach max steps

$q_{init} = [x_i, y_i, \theta_i, \kappa_i]$
$q_{goal} = [x_g, y_g, \theta_g, \kappa_g]$

Target: find p that makes $q^{\mathbf{p}}_{init}(s_G) = q_{goal}$

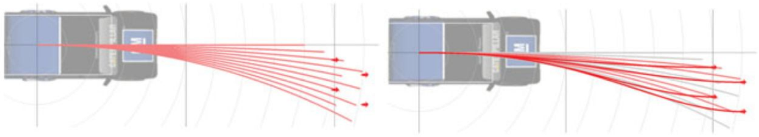
$$
\begin{aligned}
\mathbf{J} &\leftarrow \mathbf{J}_{\hat{\mathbf{p}}}(q^{\mathbf{p}}_{init}(s_G)) \\
\Delta\mathbf{q} &\leftarrow q_{goal} - q^{\mathbf{p}}_{init}(s_G) \\
\Delta\hat{\mathbf{p}} &\leftarrow \mathbf{J}^{-1}\Delta\mathbf{q} \\
\hat{\mathbf{p}}' &\leftarrow \hat{\mathbf{p}} + \Delta\hat{\mathbf{p}}.
\end{aligned}
$$

[3] McNaughton, Pic: Wikipedia

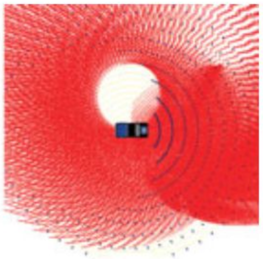
Solve the optimization problem

Initial Guess:

Offline Data Training



(a)



(c)

Nagy & Kelly's initial heuristics:

$$d = \sqrt{x_f^2 + y_f^2} \qquad \Delta\theta = |\theta_f|$$

$$s = d\left(\frac{\Delta\theta^2}{5} + 1\right) + \frac{2}{5}\Delta\theta \qquad c = 0$$

$$a = \frac{6\theta_f}{s^2} - \frac{2\kappa_0}{s} + \frac{4\kappa_f}{s}$$

$$b = \frac{3}{s^2}(\kappa_0 + \kappa_f) + \frac{6\theta_f}{s^3}$$

[1] Ferguson, [4] Nagy & Kelly

Solve the optimization problem

- Termination condition
  - Converged
  - Reach the maximum loop count

Nagy & Kelly's method:

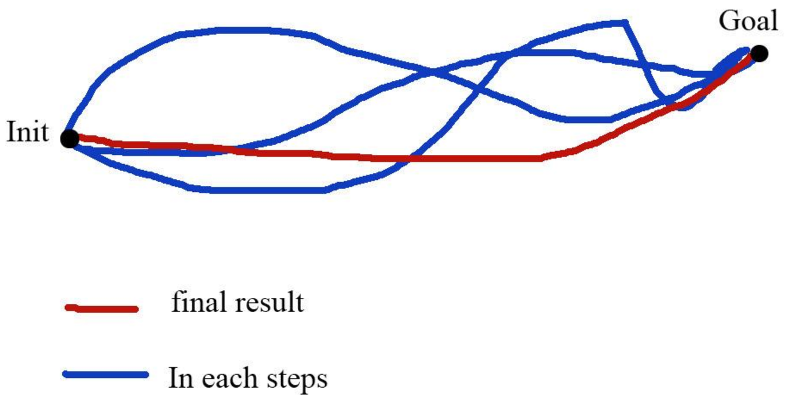
Table 1: Termination Conditions

Condition	Value
Allowable cross-track error	0.001 m
Allowable in-line error	0.001 m
Allowable heading error	0.1 rad
Allowable curvature error	0.005 1/m

[4] Nagy & Kelly

Path Results

- Results:
  - Best params $p = \{p_0, p_1, p_2, p_3, s_g\}$
  - Curvature: $\kappa(s) = a(p) + b(p)s + c(p)s^2 + d(p)s^3$
  - Depend on the init and goal state, there might be multi paths solutions.

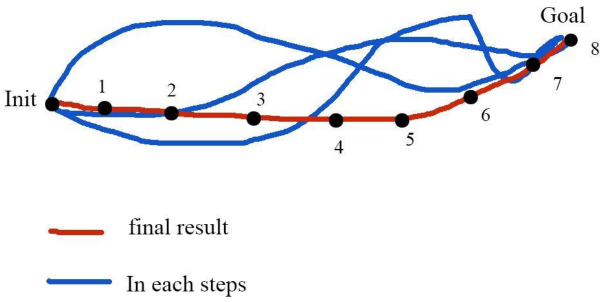


Pic: [3] McNaughton

What’s Next

- Steps:
  1. Define models for vehicle, path and trajectory.
  2. Solve the optimization problem for path planning
  3. Generate trajectories based on paths
  4. Apply cost functions to candidate trajectories
  5. Output final control signals

Trajectories Generation



$$\kappa(s) = a(p) + b(p)s + c(p)s^2 + d(p)s^3$$

- Recall the curvature equation is a continuous function, thus it can be divided into any number of parts we need
- Assign trajectory info to each parts

Trajectories Generation

- Recall trajectory model:
  - A new space M= {(x, y, θ, κ, t, v)}
  - τ:[0,1] -> M
- Recall path model:
  - C Space C== {(x, y, θ, κ)}
  - ρ:[0,1] -> C
- From paths to trajectories:
  - Use a starting time and velocity [t_0 v_0] and apply a constant acceleration a over the course of the path
  - $\tau_r(s) = [\ x_p(s)\ y_p(s)\ {}_{\theta p}(s)\ \kappa_p(s)$
    $t0 + t(s, v0, a)\ v(s, v0, a)\ ]$

[3] McNaughton

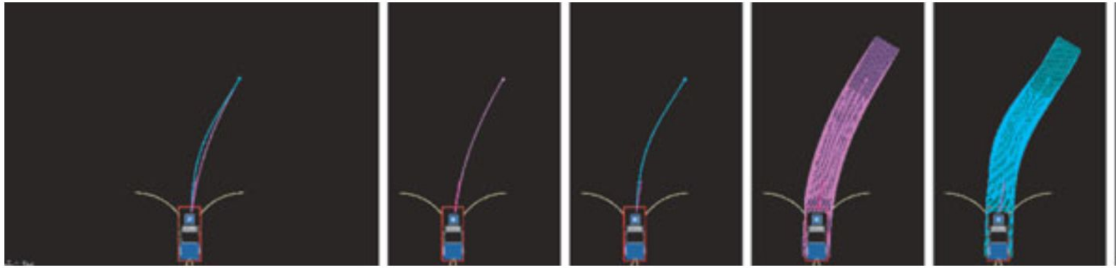
Trajectories Generation

- Equestions

$$v(s, v_0, a) = \begin{cases} \sqrt{v_0^2 + 2as} & \text{if } v_0^2 + 2as \geq 0 \\ \text{undefined} & \text{otherwise,} \end{cases}$$

$$t(s, v_0, a) = \begin{cases} {}^{s}/{}_{v_0} & \text{if } a = 0 \\ \dfrac{v(s, v_0, a) - v_0}{a} & \text{if } a \neq 0, v(s, v_0, a) \in \mathbb{R} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

- Results



[3] McNaughton Pic: [1] Ferguson

What's Next

- Steps:
  1. Define models for vehicle, path and trajectory.
  2. Solve the optimization problem for path planning
  3. Generate trajectories based on paths
  4. Apply cost functions to candidate trajectories
  5. Output final control signals

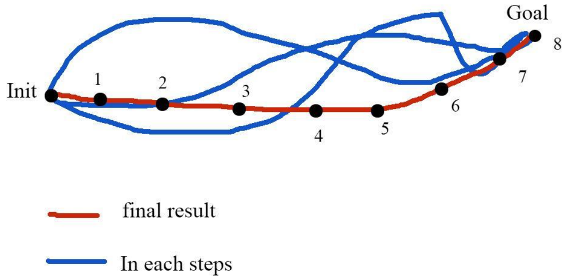
Cost Functions

- Static Cost
  1. Static obstacle(not used in DO8)
  2. Distance to way point area
  3. Curvature V.S Steering limit
- Dynamic Cost
  1. Dynamic obstacle (not used in DO8)
  2. Acceleration & deceleration
  3. Velocity
  4. Lateral acceleration
  5. Steering rate

[2] McNaughton

Cost Function

- How to evaluate
  - Divide trajectory into multi parts
  - calculate cost for each part
  - sum all the costs together
  - $C_T = \sum_{1}^{n} Cost(i)$

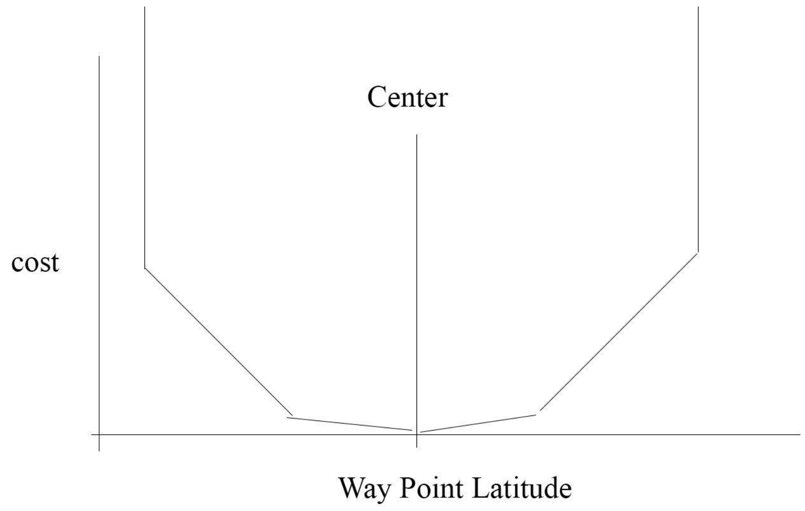


$C_T = C_1+C_2+C_3+C_4+C_5+C_6+C_7+C_8$

[2] McNaughton

Cost Functions

- Distance to lane center
  - close to center, the penalty will be very small, near 0.
  - in tolerance area, small penalty will be added
  - away from the tolerance area, high cost
  - away from the lane, the cost of this trajectory will be infinite
  - C_{lane} = Cost (dist)



[2] McNaughton

Cost Functions

- Velocity Limit
  - A cost will be applied if at any parts the vehicle exceeds speed limit
  - $C_v = k$ if $v > v_{limit}$
    $= 0$ otherwise
- Longitudinal Acceleration
  - Assume constant acceleration
  - A cost will be applied if if at any parts the vehicle exceeds acceleration limit
  - $C_a = k$ if $a > a_{limit\text{-}max}$ or $a < a_{limit\text{-}min}$
    $= 0$ otherwise

[2] McNaughton

Cost Functions

- Lateral Acceleration
  - Equation for calculate lateral acceleration

$$
\begin{aligned}
a_{\perp} = \ddot{y} &= \frac{d}{dt}\dot{y} = \frac{d}{dt} v \sin\theta \\
&= a \sin\theta + \dot{\theta} v \cos\theta = 0 + (v\kappa) v \\
&= \kappa v^2,
\end{aligned}
$$

  - Again, apply a binary cost
  - C_{al} = k if al > $al_{limit\text{-}max}$ or al < $al_{limit\text{-}min}$
    = 0 otherwise
- Curvature Rate
  - Justify the curvature according to steering limitations
  - $C_{cv} = \infty$ if $\kappa > \kappa_{limit\text{-}max}$ or $\kappa < \kappa_{limit\text{-}min}$
    = 0 otherwise

[2] McNaughton

Cost Functions

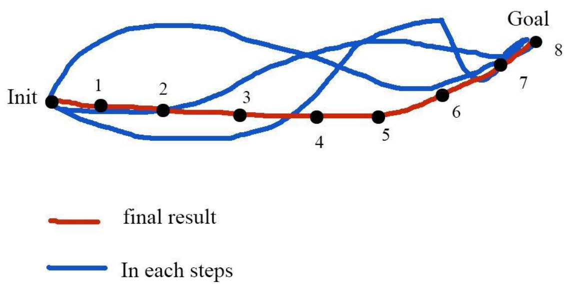
- Final Cost Function

$C_{T\text{-}l} = C_{lane} + C_v + C_a + C_{al} + C_{cv}$

$C_T = \sum C_{T\text{-}i}$

// normalized by number of samples and the length of the path

$C_T = C_T * n / s$



[2] McNaughton

What’s Next

- Steps:
  1. Define models for vehicle, path and trajectory.
  2. Solve the optimization problem for path planning
  3. Generate trajectories based on paths
  4. Apply cost functions to candidate trajectories
  5. Output final control signals

Trajectories to Control Signals

- Deal with control delays
  - Control Delay: Latency in the vehicle's physical response to the actuator commands.
  - Planning Latency: From the generation timestamp to the final vehicle movement.

- Example: In 2011, CMU's BOSS
  - Control delay: 80ms
  - Planning Latency: 200ms

[1] Ferguson

Trajectories to Control Signals

- Trajectories Queue
  - Vector based function [x, y, θ, t,v, a_1,a_2] over time interval [t_0 t_f]
  - Recall our results are continuous functions, here we divide the result into discrete parts
  - Update cycle: 10 HZ (100ms)
  - Send cmd every cycle
  - Actual control signal: earlier than “control delay” and no later than” planning latency”

Trajectories to Control Signals

- More to research
  - Fixed Frame V.S Rolling Frame
  - Deviation between planner and vehicle motion

Pseudo Code

```
# state: (x,y,θ,v) , waypoints, closet, next, next next
function generateTrajectory( currentState, nextWayPoints):
        nextState = estimateStatefromWaypoints();
        P = initialGuessP()
        # Newton's method
        loop ( converge or maximum reach):
                tempState = estimateStateP(P)
                Δs = nextState – tempState
                J_p  = calculateJacobion(tempState)
                ΔP = Δs * inverse(J_p )
                tempP = tempP+ΔP
                # run limitation check to see if the P is good enough
                checkConverge()
        P = tempP
        # return paths based on p = {p0,p1, p2,p3,sg }
        paths = getPaths(P)
        # apply vehicle model on the paths
        trajctories = getTraj(paths)
        # select the best trajectory
        finaltraj = runCostFunction(trajctories)
        return finaltraj
```

Experimental Results

$$
\begin{aligned}
\mathbf{J} &\leftarrow \mathbf{J}_{\hat{\mathbf{p}}}(q^{\mathbf{p}}_{init}(s_G)) \\
\Delta \mathbf{q} &\leftarrow q_{goal} - q^{\mathbf{p}}_{init}(s_G) \\
\Delta \hat{\mathbf{p}} &\leftarrow \mathbf{J}^{-1}\Delta \mathbf{q} \\
\hat{\mathbf{p}}' &\leftarrow \hat{\mathbf{p}} + \Delta \hat{\mathbf{p}}.
\end{aligned}
$$

Nagy & Kelly's method for initial guess, 2001

Table 2: Runtimes

Initial Parameter Source	Avg. Runtime (sec.)
Default Heuristics	0.0134
Previous Cubic Parameters	0.00557

[3] McNaughton

Experimental Results

CMU’s Boss

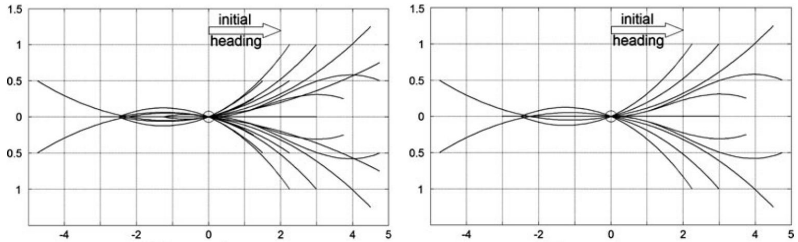
- In real traffic scenario, 2010
- GPU: Nvidia GeForce GTX 260, CPU: Intel Core 2 Quad processor
- The planner runs on a 10 Hz (100ms) update cycle
- Speed limit: 30mph

Search Phase	GPU Time	CPU Time	Speedup
Plan trajectories from source pose onto lattice	2 ms	12 ms	6
Update all paths in lattice	0.1 ms	4 ms	40
Plan all trajectories coming out of a single station	2 ms	42 ms	21
Whole planning cycle	45 ms	650 ms	15

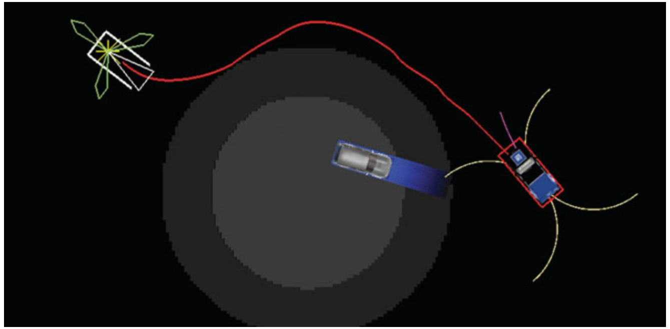
[3] McNaughton

Beyond this

- Apply to real traffic scenario
  - Use SLAM to create environment map
  - Create a occupancy grid map (or lattice map)
  - Use A*-similar algorithm to find the best path
  - Connect vertices along the path
  - Generate trajectories between vertices



(a) Dense action space
(b) Coarse action space



Pic: [1] Ferguson

References

- [1] Ferguson, Dave, Thomas M. Howard, and Maxim Likhachev. "Motion planning in urban environments." *Journal of Field Robotics* 25.11-12 (2008): 939-960.
- [2] McNaughton, Matthew. "Parallel algorithms for real-time motion planning." (2011).
- [3] McNaughton, Matthew, et al. "Motion planning for autonomous driving with a conformal spatiotemporal lattice." *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011.
- [4] Nagy, Bryan, and Alonzo Kelly. "Trajectory generation for car-like robots using cubic curvature polynomials." *Field and Service Robots* 11 (2001).
- [5] Choset, Robotic Motion Planning: Configuration Space
- [6] Baker, Christopher R., David I. Ferguson, and John M. Dolan. "Robust mission execution for autonomous urban driving." *Robotics Institute* (2008): 178.

About

- I wrote this document in order to study the trajectory planning method used in CMU’s Boss the autonomous driving car.
- Matthew O’Kelly, who wrote the Autoware’s trajectory planning module, helped me a lot during my study on this. Thank you!
- And the next might be something based on this algorithm using C++ or Python. In summer break 2017.