A Problem Solving Philosophy

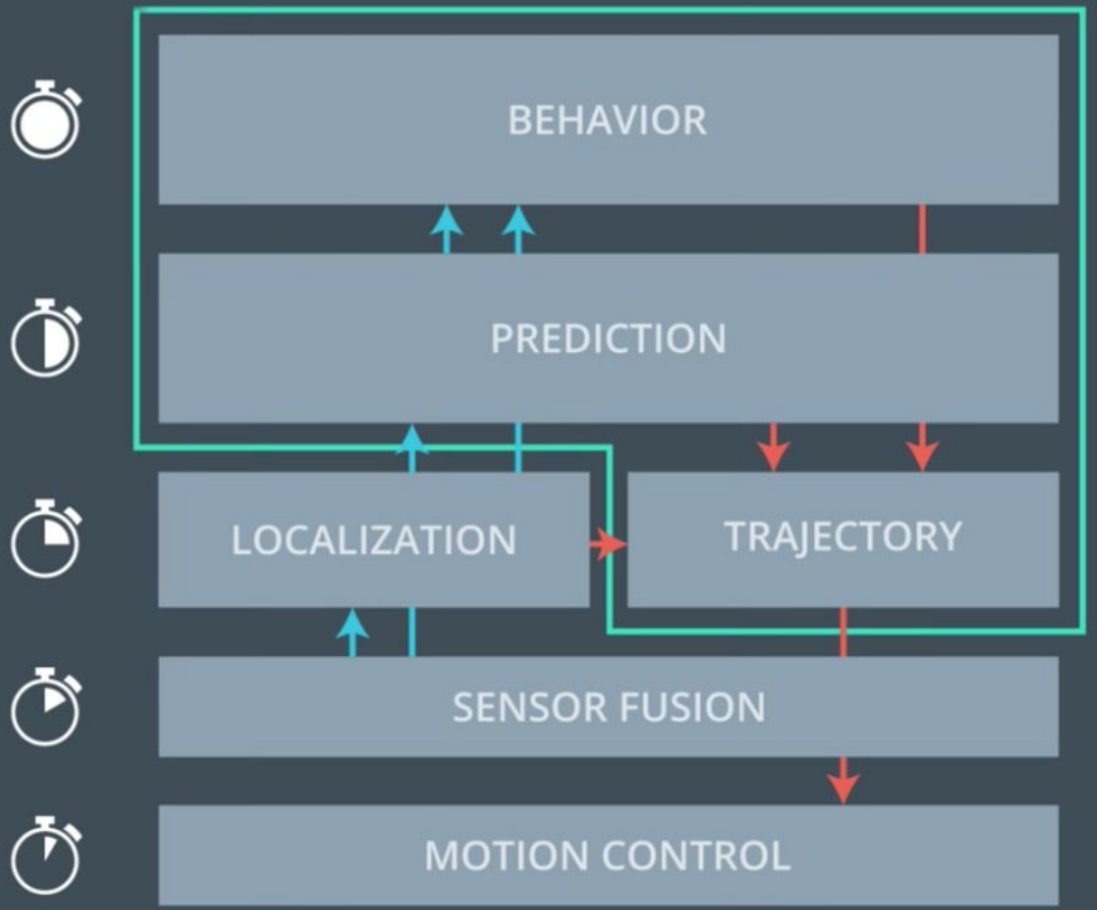
A Problem Solving Philosophy

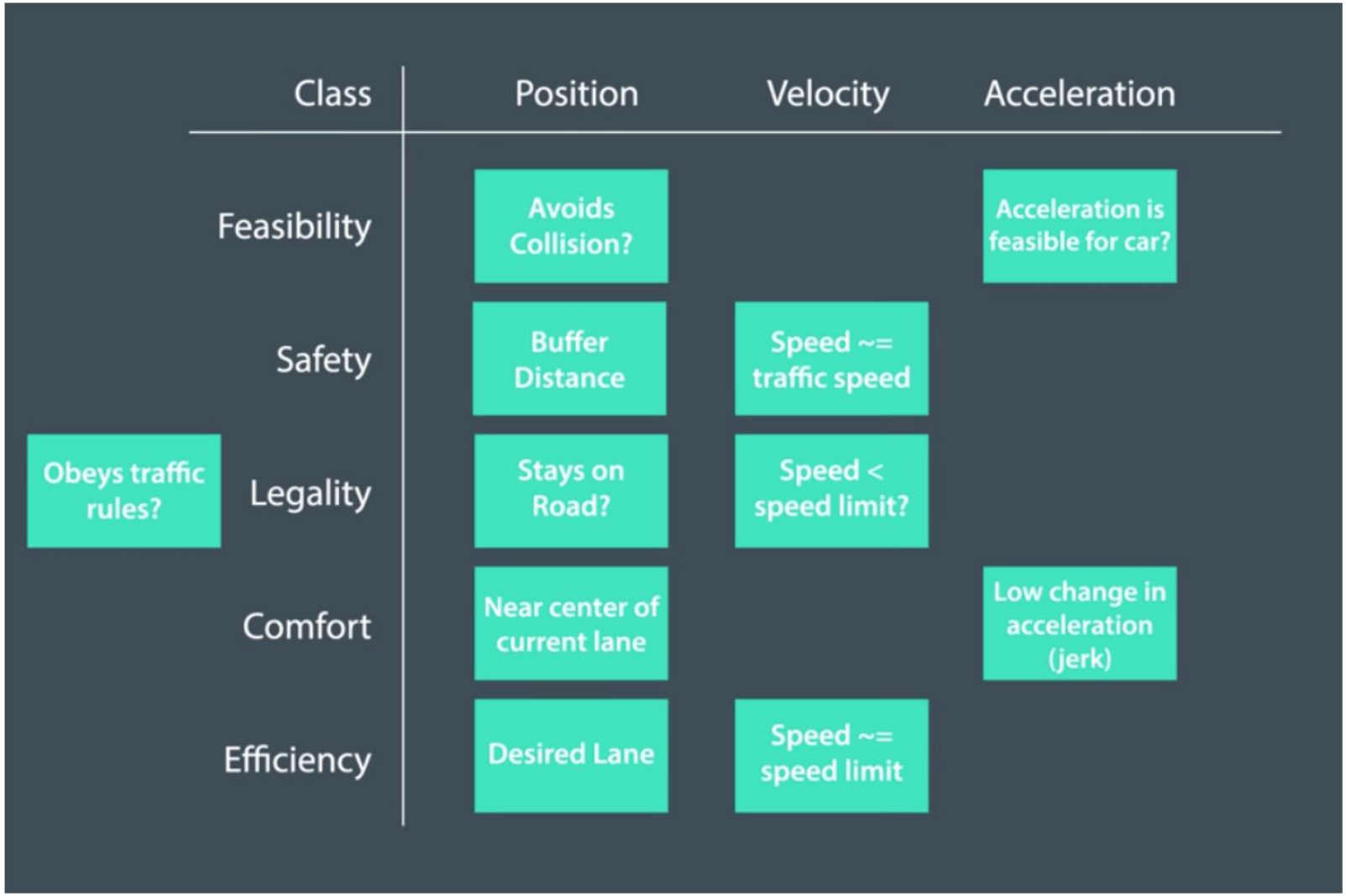
- What I keep in mind when I try to solve problems

A Problem Solving Philosophy

- Disclaimer!
- GUIDELINES NOT RULES

Behavior Control




Class
Position
Velocity
Acceleration
Feasibility
Avoids Collision?
Acceleration is feasible for car?
Safety
Buffer Distance
Speed ~= traffic speed
Obeys traffic rules?
Legality
Stays on Road?
Speed < speed limit?
Comfort
Near center of current lane
Low change in acceleration (jerk)
Efficiency
Desired Lane
Speed ~= speed limit

Motivating Example

Motivating Example

For a long period of time I was in a state of analysis-paralysis and did not even write a single line of code.

Why?

Motivating Example: So many questions!

- Which data is important and which isn't?
- How far into the future should I plan?
- Should I predict what the nearby cars are going to do before deciding what to do?
- What method should I use in prediction?
- How accurate and useful are these predictions?
- How many possible behaviors should I consider *(e.g slow down, go fast, turn right, prepare to turn left)*?
- For any behavior, how many paths should I consider before deciding which is the best one? If I decide to go slow, how slow? If I decide to turn left where exactly in the left? How do I make sure that my path is the best and does not violate any safety or comfort guidelines?


Distance Without Incident
Best: 2.75 Miles
Curr: 2.75 Miles
Timer Seconds: 204.27
PASS: You went 2.75 miles in 204.27 seconds
AccT: 0 m/s^2
AccN: 0 m/s^2
AccTotal: 0 m/s^2
Jerk: -1 m/s^3
48.79
MPH
ESC
MENU

If you want more details:

Code:

github.com/mithi/highway-path-planning

Detailed thought process:

medium.com/@mithi/reflections-on-designing-a-virtual-highway-path-planner-part-1-3-937259164650

A Problem Solving Philosophy

- **All models are wrong**
- YAGNI (DTSTTCPW)
- TIMTOWDI
- Consider “The Academic Way”

ONE: ALL MODELS ARE WRONG

BUT SOME MODELS ARE USEFUL!

- George Box, a famous statistician.

ONE: ALL MODELS ARE WRONG

- “The law PV = nRT relating pressure P, volume V and temperature T of an “ideal” gas via a constant R is not exactly true for any real gas, but it frequently **provides a useful approximation**”

ONE: ALL MODELS ARE WRONG

- “For such a model there is no need to ask the question **“Is the model true?”**. If “truth” is to be the “whole truth” **the answer must be “No”**.

ONE: ALL MODELS ARE WRONG

- The only question of interest is **“Is the model illuminating and useful?”**

A Problem Solving Philosophy

- Start with a simple model with many simple assumptions and work from there.

Four models for lane following

Linear point model
(constant velocity)

$$\begin{bmatrix} \dot{s} \\ \dot{d} \end{bmatrix} = \begin{bmatrix} \dot{s}_0 \\ 0 \end{bmatrix} + \mathbf{W}$$

NON-linear point model (constant acceleration with curvature
c(s) = c_0 + c_1 * s)

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\omega} \\ \dot{a} \\ \dot{c}_0 \\ \dot{c}_1 \end{bmatrix} = \begin{bmatrix} (v+at)cos(\theta) \\ (v+at)sin(\theta) \\ \omega \\ a \\ 0 \\ 0 \\ vc_1 \\ 0 \end{bmatrix} + \mathbf{W}$$

Kinematic bicycle model with controller
(PID controller on distance and angle)

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

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

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

$$\delta_t = -J_P CTE - J_D \dot{CTE} - J_I \sum_{i=0}^{t} CTE_i$$

A Problem Solving Philosophy

- If the assumption does not work, make the model more complex.

A Problem Solving Philosophy:

- All models are wrong
- **YAGNI (DTSTTCPW)**
- TIMTOWDI
- Consider “The Academic Way”

TWO: YAGNI (DTSTTCPW)

- **Y**ou **A**ren’t **G**onna **N**eed **I**t

TWO: YAGNI (DTSTTCPW)

- **D**o **T**he **S**implest **T**hing **T**hat **C**ould **P**ossibly **W**ork

TWO: YAGNI (DTSTTCPW)

- A programmer should not add functionality until deemed necessary.

TWO: YAGNI (DTSTTCPW)

- Always implement things when you actually need them, **never** when you just foresee that you need them.

Jack Diederich - Ex-Googler

Prefer Easy Things

- Don't do hard things in the first place
- Revert complications later

"I hate code and I want as little of it as possible in our product"

-me, always

TWO: YAGNI (DTSTTCPW)

Over-Engineering is when someone decides to build more than what is really necessary based on speculation

TWO: YAGNI (DTSTTCPW)

- Make everything simple if you can get away with it!!!

RELEVANT EXAMPLE: THROTTLE ADJUSTING

- SIMPLE EQUATION
- PID BASICS
- MODEL PREDICTIVE CONTROL

THROTTLE: SIMPLE EQUATION

```
speednorm = speed/MAX_SPEED
anglenorm = angle/MAX_ANGLE

throttle =
   MAX_THROTTLE - A*speednorm² - B*anglenorm²

throttle = max(MIN_THROTTLE, throttle)
```

THROTTLE ADJUSTING: PID CONTROL

PLAY VIDEO HERE

youtube.com/watch?v=4Y7zG48uHRo

Aerospace Controls Laboratory

Massachusetts Institute of Technology

THROTTLE ADJUSTING: PID CONTROL

$$u = \underbrace{K_p e}_{\text{Proportional Term}} + \underbrace{K_i \int_0^t e\,dt}_{\text{Integral Term}} + \underbrace{K_d \frac{d}{dt} e}_{\text{Differential Term}}$$

THROTTLE ADJUSTING: PID CONTROL

```
double SimplePIDController::compute(const double error){

  p_error = error;
  i_error += error;
  d_error = error - previous_error;
  previous_error = error;

  return -1. * (Kp * p_error + Ki * i_error + Kd * d_error);
}
```

- Assumes **dt is 1.0** (constant)

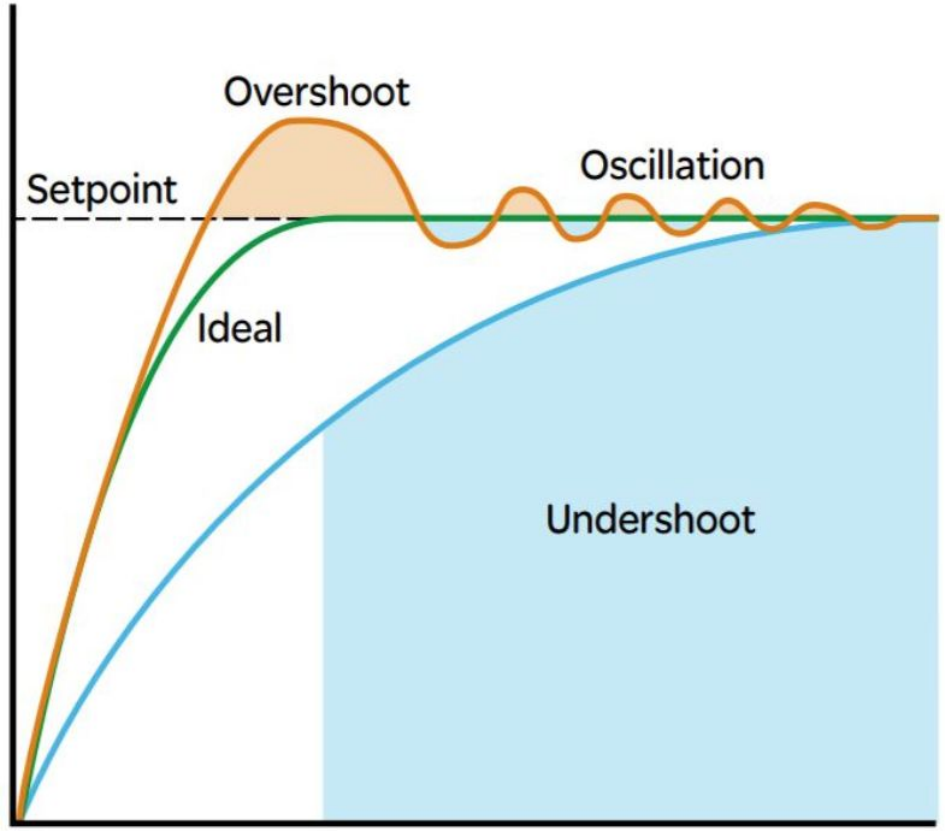
THROTTLE ADJUSTING: PID CONTROL

What is the percentage of the PID algorithm applications in industry?

I read a survey from Honeywell company from the year 2000, that the PID algorithm is used in 97% of industrial applications. My question is, do somebody have newer survey about this? Thanks.

researchgate.net/post/What_is_the_percentage_of_the_PID_algorithm_applications_in_industry

THROTTLE ADJUSTING: PID CONTROL



Tune PID controller coefficients

- Manually
  - Some more complex algorithms are developed to automate this but probably not worth your time

THROTTLE ADJUSTING: PID CONTROL

TWEAKING / TUNING PID

Kp - Coefficient for error

Too low - longer to reach desired speed

Too high - Overshoot, can spiral out of control

Kd - Coefficient - rate of change

Too low - Oscillation, overshoot

Too high - longer to reach desired speed

Ki - Coefficient - cumulative error over time

Too low - takes longer to reach desired speed

Too high - Oscillations, can spiral out of control

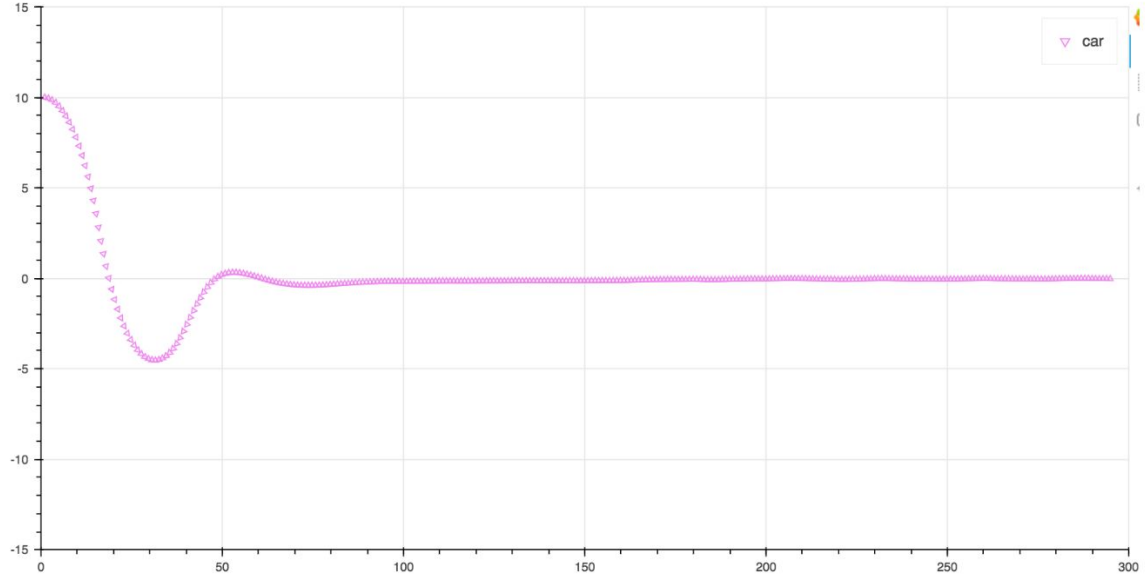
- Used for systematic bias like wind, mechanical defects, drifts
- When not converging to desired setpoint

THROTTLE ADJUSTING: PID

p 0.50
i 0.01
d 3.00

```
tp: 0.5
ti: 0.005
td: 3.0
last error: 0.0301692306072
average error of last 100 steps: 0.0431811833564
average error 0.877674699415
```

BokehJS successfully loaded.



github.com/mithi/simple-pid-toy

THROTTLE ADJUSTING: MPC

- Model predictive control
- Might be an overkill for your purposes
- Probably requires a larger processor than RPI3 (or use C++)

THROTTLE ADJUSTING: MPC

- Still Worthwhile to discuss, why?
- Able to address PID’s limitations
- Widely used in aggressive autonomous driving

THROTTLE ADJUSTING: MPC

PLAY SOME VIDEOS HERE

A little excerpt video on MPC explanation :)

AutoRally - Autonomous Dirt Track Car for Aggressive Driving
Georgia Tech - May 20, 2016 - 1:32
https://youtu.be/T4ZB3RYSbrk

Obstacle Avoidance using Learning Model Predictive Control
Ugo Rosolia - Mar 1, 2018 - 58 seconds
https://www.youtube.com/watch?v=_ESVg0gt9bk

THROTTLE ADJUSTING: MPC

- Basically PID is reactive and
- MPC plans ahead, complex, accounts for latency

MPC

According to wikipedia:

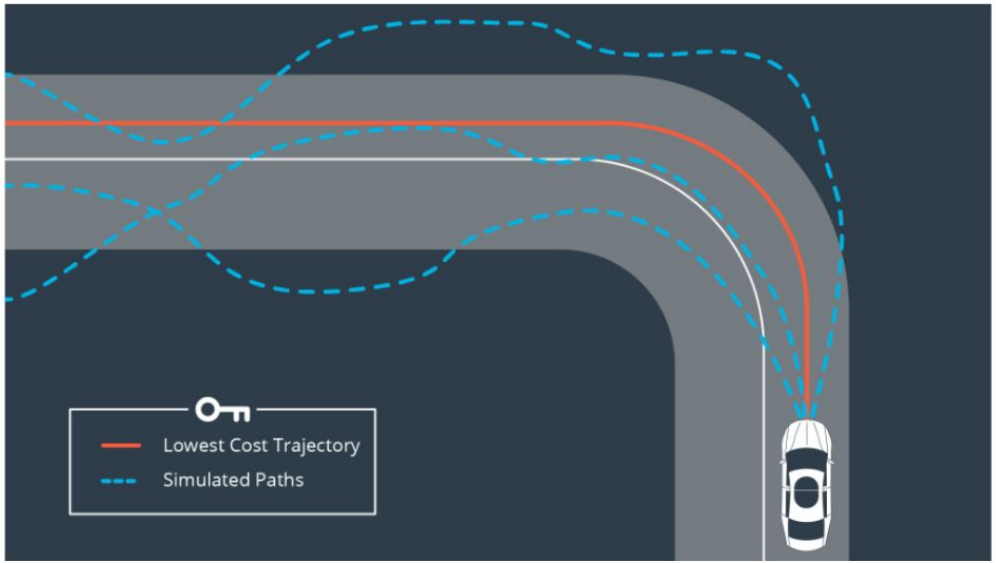
> Model predictive controllers rely on dynamic models of the process. The main advantage of MPC is the fact that it allows the current timeslot to be optimized, while keeping future timeslots in account. This is achieved by optimizing a finite time-horizon, but only implementing the current timeslot. MPC has the ability to anticipate future events and can take control actions accordingly.

More detailed explanation in relation to self-driving cars

github.com/mithi/mpc

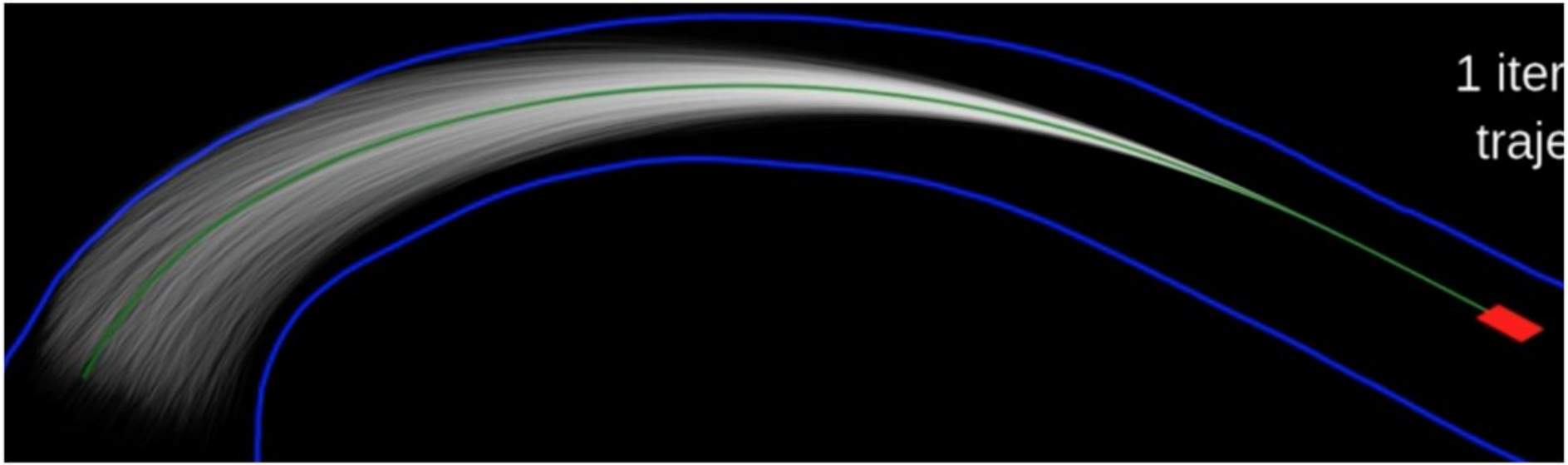
THROTTLE ADJUSTING: MPC

Model Predictive Control (MPC) uses an optimizer to find the control inputs that minimize the cost function.



We actually only execute the very first set of control inputs. This brings the vehicle to a new state and then we repeat the process.

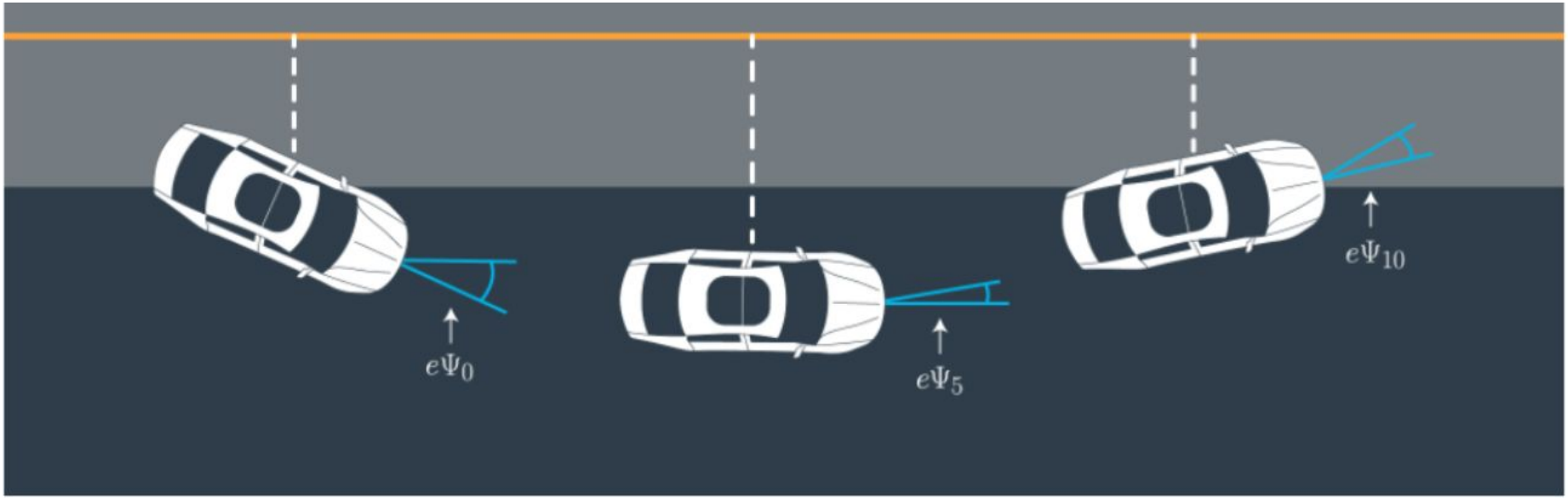
THROTTLE ADJUSTING: MPC



THROTTLE ADJUSTING: MPC

- Simulate a lot of trajectories and pick the one with the least cost
- Trajectories require physics model

REQUIRES PHYSICS MODEL TO PLAN AHEAD



The dashed white line is the cross track error.

REQUIRES PHYSICS MODEL TO PLAN AHEAD

So based on *physics*, here is a simplified version of how the world (with our vehicle in it) works. How the state variables get updated based elapse time `dt`, the current state, and our actuations `delta` and `a`.

```
px` = px + v * cos(psi) * dt
py` = py + v * sin(psi) ( dt)
psi` = psi + v / Lf * (-delta) * dt
v` = v + a * dt

Lf - this is the length from front of vehicle to its Center-of-Gravity
```

We can also predict the next `cte`, and `epsi` based on our actuations.

```
cte` = cte - v * sin(epsi) * dt
epsi` = epsi +  v / Lf * (-delta) * dt
```

MINIMIZE COST (EXAMPLE)

- i. We don't want to steer if we don't really need to
- ii. We don't want to brake if we don't really need to
- iii. We don't want consecutive steering angles to be too different
- iv. We don't want consecutive accelerations to be too different

So mathematically it should be like:

```
cost = A * cte^2 + B * epsi^2 + C * (v - vmax)^2 +
       D * delta^2 + E * a^2 + F * (a` - a)^2 +  G * (delta` - delta)^2

... integrated over all time steps
```

Curated MPCVideos: watch at your own time

Fast Nonlinear Model Predictive Control for Unified Trajectory Optimization
EthZurich - ADRLAbEth - Feb 15, 2016 - 3:48 minutes
https://www.youtube.com/watch?v=Y7-1CBqs4x4
Learning MPC for Autonomous Racing
Ugo Rosolia = Jan 18, 2017 - 1:38 Minutes
https://www.youtube.com/watch?v=4kHDv9senpE
Understanding Model Predictive Control, Part 1: Why Use MPC?
MATLAB - Published on May 15, 2018 - 4:50 Minutes
https://www.youtube.com/watch?v=8U0xiOkDcmw

RELEVANT EXAMPLE: THROTTLE ADJUSTING

- SIMPLE EQUATION
- PID BASICS
- MODEL PREDICTIVE CONTROL

TWO: YAGNI (DTSTTCPW)

- **D**o **T**he **S**implest **T**hing **T**hat **C**ould **P**ossibly **W**ork

THREE: TIMTOWDI

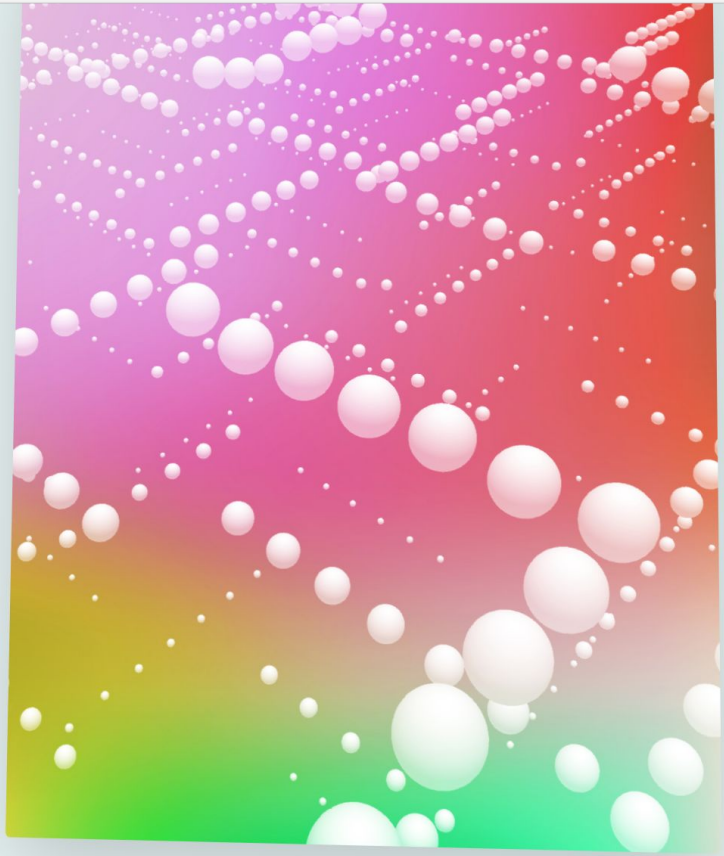
- **T**here **I**s **M**ore **T**han **O**ne **W**ay **T**o **D**o **I**t
- *Pearl-programming motto*

In contrast to Python’s:

- *There should be one - and preferably only one - obvious way to do it*

URBAN LEGEND: THE BAROMETER

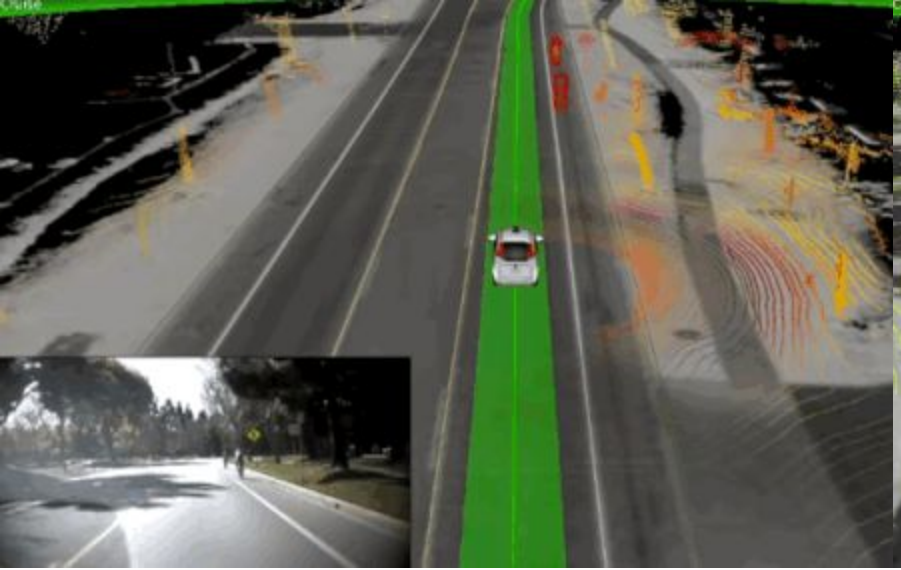
1. Tie a string to the barometer, lower it down the side of the building, and measure the string.
2. You can just drop the barometer over the side of the building and figure out the height by the time it takes to hit the ground. 0.5*a*t^2
3. You can measure the shadow and the length of the barometer, and then measure the shadow of the building, and use the ratio of shadow-to-length to figure out the height of the building.
4. You can make a pendulum from the barometer and measure the gravitational force exerted by measuring the swing, calculate the force of Earth's gravity, which lessens as you get away from the Earth, thus determining the height of the building.
5. You can climb down the fire escape, using the barometer as a ruler, and mark off the building's height in barometer lengths units.
6. Or if you're smart, you can go to the janitor and tell him that you'll give him this beautiful new barometer if he tells you the height of the building.
7. If you're very dull, you can use the barometer to calculate the air pressure on the ground and at the top of the building and use the difference to work out the height of the building.

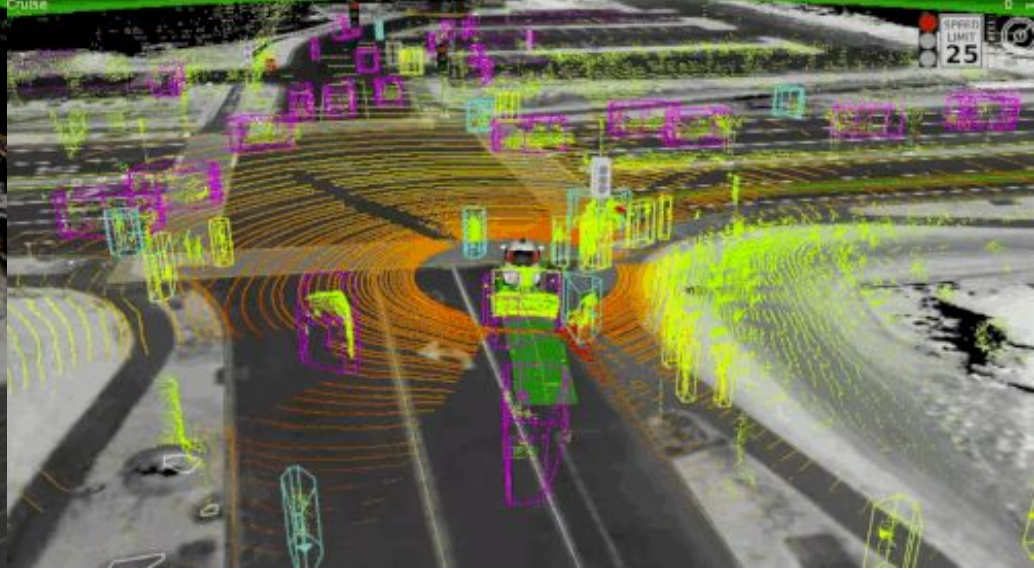


a Scalable Alternative to Reinforcement Learning

We've discovered that **evolution strategies (ES)**, an optimization technique that's been known for decades, rivals the performance of standard **reinforcement learning (RL)** techniques on modern RL benchmarks (e.g. Atari/MuJoCo), while overcoming many of RL's inconveniences.

In particular, ES is simpler to implement (there is no need for backpropagation), it is easier to scale in a distributed setting, it does not suffer in settings with sparse rewards, and has fewer hyperparameters. This outcome is surprising because ES resembles simple hill-climbing in a high-dimensional space based only on finite differences along a few random directions at each step.


Cruise


Cruise
SPEED LIMIT
25

ELON MUSK

- **THE FUTURE OF DRIVERLESS CARS IS CAMERAS**
- *(no lidar)*

NEWS 06 October 2017

GM EXPERT: ELON MUSK IS 'FULL OF CRAP' ON TESLA'S AUTONOMOUS DRIVING CAPABILITY

In a posting on Medium last year, Kyle Vogt, CEO of GM's Cruise Automation, wrote that "Lidar sensors contribute to the redundancy and overlapping capabilities needed to build a car that operates without a driver, even in the most challenging environments."

Elon Musk still doesn't think LIDAR is necessary for fully driverless cars

'In my view, it's a crutch'

By Andrew J. Hawkins | @andyjayhawk | Feb 7, 2018, 6:45pm EST

SHARE


7HPV053
MacRumors
© Courtesy of MacRumors


TESLA
MODEL S

A Problem Solving Philosophy

- All models are wrong
- YAGNI (DTSTTCPW)
- TIMTOWDI
- **Consider “The Academic Way”**

Consider the Academic Way

- Find out how very very intelligent people are trying to solve the problem.

Consider the Academic Way

- Find out how very people who have been studying stuff like this for a long time do it

Consider the Academic Way

- Read research papers
- We will consider a two research papers on the next sections of this talk

A Problem Solving Philosophy

- **All models are wrong** (some models are useful)
- **YAGNI** (Prefer Simple things)
- **TIMTOWDI** (More than one way)
- **Consider “The Academic Way”**