MAY mobility

Building an AV the May Way



Edwin Olson | October 2024
CHIEF EXECUTIVE OFFICER



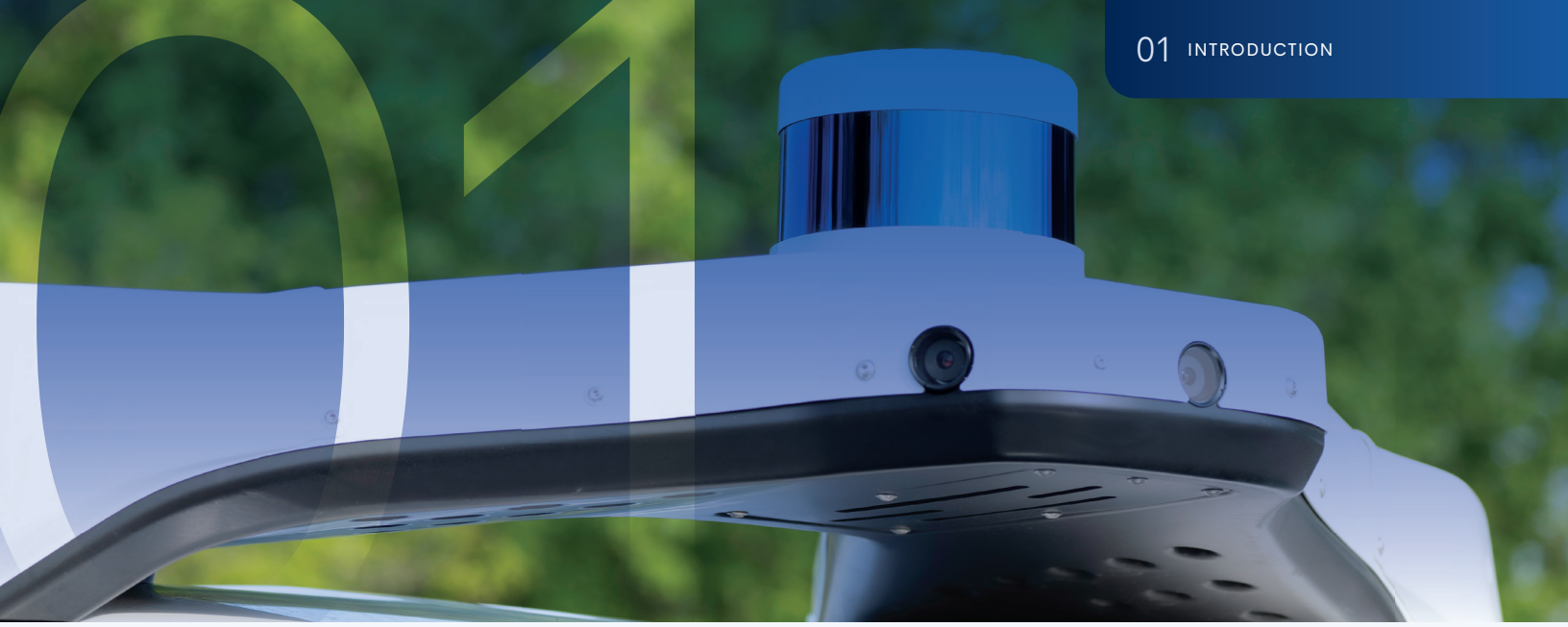
Table of Contents

3 Introduction

4 The Thought Experiment

6 Online Reinforcement Learning

9 Discussion



Introduction

It has been exciting to watch data-driven learning methods massively improve speech recognition, large language models, weather forecasting and so many other areas. The fundamental idea behind these tools (including convolutional nets, transformers, end-to-end systems and so on) is the same: collect a bunch of data, learn a model in the data center, then deploy that model to the field.

And so of course engineers tried the same approach with autonomous vehicles: collect/generate a bunch of driving scenarios, learn a model and deploy that model to cars driving in the real world.



But it hasn't worked well. No matter how much data we collect or generate ahead of time, vehicles continue to find themselves in unfamiliar situations. And when a vehicle is outside of its training set, it's much more likely to make a mistake.

Because of the need to collect, generate, label and curate datasets, AV companies have become the most capital intensive businesses in history. Companies like Waymo and Cruise have spent tens of billions of dollars, and will spend much more before they become profitable. Worse, it's not a one-time cost: each new city expands the need for data because people behave differently in different places.



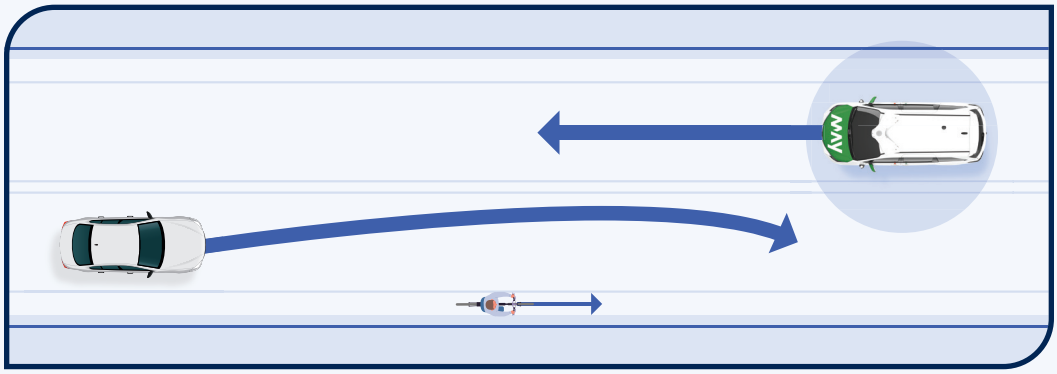
The Thought Experiment

In order for an autonomous car to handle a situation well, it needs to have been trained on a similar situation. We wouldn't expect a handwriting-recognition system trained on the digits 0–9 to recognize letters or smiley faces!

This is not true of human drivers. The first time we encounter an odd situation—perhaps a group of kids playing hockey in the street—we can reason through what to do.

How do we do it? We imagine. We make predictions about what other road users are going to do, and we imagine what the future might look like.

Consider this example, where the autonomous vehicle is on the right. A bicyclist and a car are approaching. The oncoming car will veer around the bicyclist—and we probably want our car (on the right) to be somewhere else when that happens. We could either slow down so the white car passes the bicycle first, or speed up so that we're out of the way of both of them.





I think a human driver would be unlikely to make the right call by just looking at the situation. Instead, I think the human would "play out" the situation step by step, forming a mental "movie" of what is likely to happen. That movie would incorporate their predictions of what agents would do, and like any movie, considers the passage of time as a critical factor.

A human might even imagine multiple movies, potentially imagining variants of the situation. Maybe some are benign ("the bicyclist turns right into their driveway, leaving plenty of room for both cars"), and maybe some are scary ("both cars pass the bicycle at the same time, leaving too little room for the bicyclist.")

Wouldn't it be great if autonomous vehicles could do the same thing—imagine how the current scenario might unfold, identify the outcomes that could be dangerous and adapt accordingly?



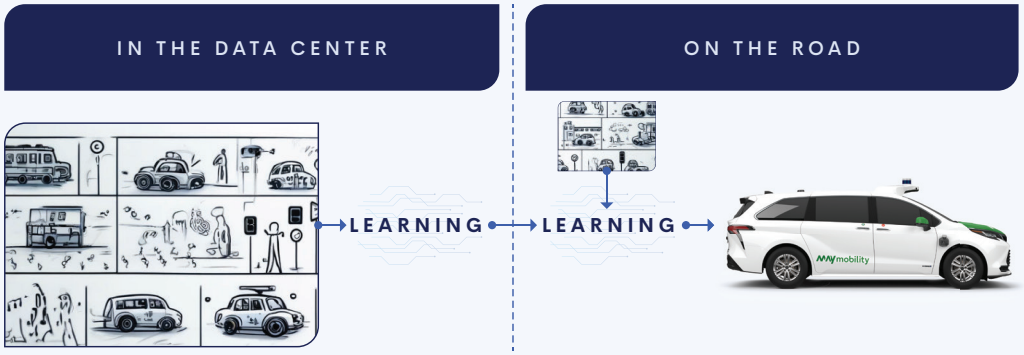


Online Reinforcement Learning

At May Mobility, our vehicles do exactly this. We generate new training data on the fly—examples that are incredibly relevant to the situation that the vehicle is in. And we use this training data to learn an updated driving strategy for the autonomous car.

We begin by training in the data center, like many AV companies. Today's off-line learning methods are incredibly powerful, and it would be silly to "throw the baby out with the bath water".

But autonomous cars need to go further—we need to be able to learn and adapt in the moment. That's illustrated on the right side of the figure: we generate additional training data and do additional learning onboard the car. These additional steps are exactly analogous to a human imagining what might go right or wrong.





When it can take weeks to train a new model in the data center, it may sound preposterous that we can learn anything fast enough in the car to matter. To be relevant, any in-situ learning has to be done within about **200 milliseconds**!

This is hard to do, especially using only the computational resources that are available on a car. But it can be done with a method we developed called Multi-Policy Decision Making (MPDM).

There are a few key levers that we exploit to learn really fast:

- When learning offline (in the data center), we have to learn policies that perform well in as many situations as possible. In contrast, when learning in situ, you only need to worry about the situation you're in right now. That reduces the problem from "all situations" to "one situation".
- We make the in-situ problem even easier by changing what we learn in the data center. We learn exactly the ingredients we need to do online learning very efficiently.

What are these ingredients?

Instead of a single driving model, we get a set of driving strategies from the data-center. This set is diverse—representing a range of driving styles and strategies. The goal is for us to have a set of policies such that at least one is a good choice for virtually any situation. Because most driving is relatively boring, most of our driving is done with just a few policies that you might characterize as calm and patient. They provide smooth and comfortable rides. The rest of our policies are for unusual driving situations. For example, other policies might be better when greater precision is needed (narrow lanes), or where being more aggressive is better (merging into traffic). (The curation of these base policies is itself a fascinating problem!)

But critically, we don't prescribe when each policy should be used. It would be hopelessly brittle for us to say "*use policy #37 when turning right on red and policy #22 when passing a bicyclist.*" Instead, we evaluate the performance of all of the policies all the time, and the best performing policy "*wins*" control of the vehicle.

With this background in place, the basic MPDM algorithm is actually pretty easy to describe:

1. Pick a policy for every agent in the scene. Our perception system outputs a probability distribution over policies for each agent, so this is just a probabilistic sampling operation.
2. For each of the policies available for our autonomous car, simulate what's likely to happen over the next ~10 seconds. Since we have policies assigned to each agent at this point, this is a Monte Carlo simulation where we repeatedly ask each agent's policy "what would you do next?" Every outcome is scored according to a reward function, which evaluates safety and comfort—not just for us, but for everyone on the road.
3. We pick the policy for our autonomous vehicle that had the best performance overall.

It took years of development to go from a mathematical formulation to a system that is fast and inexpensive enough to build into hundreds (and soon thousands) of autonomous vehicles. Along the way, we developed a patent portfolio around both the core technology as well as many enhancements beyond the basic approach described here. You can read about them in our filings.

But here's the punchline: our policy-centric MPDM approach gives our vehicles a better ability to imagine what's going to happen next. It allows us to operate safely in situations that aren't in our training data. And since building massive training sets and retraining models in the data center is incredibly expensive, we're able to build autonomous vehicles for a fraction of the cost.

That's exciting to us, because it means that we can accelerate the deployment of autonomous driving into the world. And that will mean safer streets for all road users, including pedestrians and cyclists, while helping to reduce the need for acres of lanes and parking.



Discussion

What makes Autonomous Driving hard?

We can view all AI systems as functions, *y=f(x)*. Given some input *x*, we want to learn to produce some output *y*.

- The *size* of the input space *x*. Chess is a harder problem than tic-tac-toe because there are more possible board positions—there's more values of *x* to consider.
- The *rate* at which the output changes. If small changes to the input result in large changes to the output, it takes more data to learn the function.

Both of these factors are working against autonomous driving. The input space is large: we must consider the positions and past actions of all other actors in the environment; the shape of the road; the goals of the autonomous vehicle; the state of traffic control devices; the weather and driving conditions, etc. And the correct decision for a vehicle is very sensitive to the input: teleporting a pedestrian by half a meter can change an autonomous car's best decision from "*drive on by*" to "*slam on the brakes*".

This sensitivity to inputs means that it is very difficult to directly learn the correct action "*y*" from an initial state "*x*" without explicitly considering how the road users will interact over the next several seconds. It becomes easier to do this reasoning if you think in terms of time explicitly—each agent making relatively simple decisions, but doing so repeatedly over time and in response to the actions of other agents. *(see "direct learning" section for more on this.)*

What is the impact of GenerativeAI on autonomous vehicles?

Generative AI's main application to autonomous driving is the generation of training data, including scenario generation and simulated perception data. Even though it's infeasible to generate enough offline training data to ensure that your vehicle is always "within its training set", there are certainly benefits to having larger training sets.

So of course, virtually all AV companies (including May) generate additional simulation data to augment the real-world data they have. But it simply can't compete with on-vehicle learning like MPDM in terms of ensuring that a vehicle always has relevant training data.



What is the impact of end-to-end learning on autonomous vehicles?

The idea of end-to-end learning is to learn more of the relationship between the inputs and the outputs, with engineers imposing less structure on intermediate reasoning steps. The alternative to end-to-end is a modular system, in which engineers divide a large problem into pieces. Each piece can be then learned separately.

The motivation behind end-to-end learning is that if engineers impose the wrong structure, it can limit the performance of the system. But the risk in end-to-end learning is that they need to learn more, which means they need more data.

Bigger and more complex models need more data. If the model is too big in comparison to the amount of data, you end up with overfitting. When a system overfits, it may make decisions based on a criteria other than what is intended (or reasonable). For example, an autonomous car might drive through a red light despite having seen thousands (or millions) of training examples with red lights because it doesn't understand that this situation is similar to the previous examples.

In other words, end-to-end learning methods make the hardest problem in autonomous driving—having the right training data—worse.

There are two other challenges with end-to-end learning. First, the best performing architectures (e.g. convolutional nets versus transformers) vary by task. LIDAR data might flow in through one part of the system, camera data through another (differently-architected) part, and these are then combined together. There isn't some "general purpose goop" that you throw all of your multi-modal data into willy-nilly; engineers build structure (some might even say "modularity") into the architecture itself. This is not a criticism—it is smart to exploit structure. But tmany people misunderstand end-to-end to mean that "everything is learned", which misses the massive role that architecture plays.

The second challenge with an end-to-end system is introspection and validation. Deep learning systems opaquely encode information within them: it is very difficult for a human to interpret the values being passed between layers. This is a challenge if you want to make strong claims about safety.

Returning to our classic $y=f(x)$ view of learning, if you feed images (x values) in and get steering commands (y values) out, you can obviously show that specific values of x do (or don't) lead to collisions. But it's hard to reason about other values of x, and if you do get a failure, it's hard to figure out why that mistake occurred. A modular system might provide human-interpretable intermediate results, for example a list of detected cars. This can help make it clear why the system performed the way it did.

What about camera-only systems like Tesla?

Tesla is focusing on personally-owned vehicles, and consumers are very sensitive to price. In turn, this means that Tesla must keep their costs very low. With LIDAR costs relatively high today, Tesla relies on cameras instead. Unfortunately for Tesla, camera-only systems just aren't good enough yet, perhaps best demonstrated by the many delays to Tesla's fully-self-driving product.

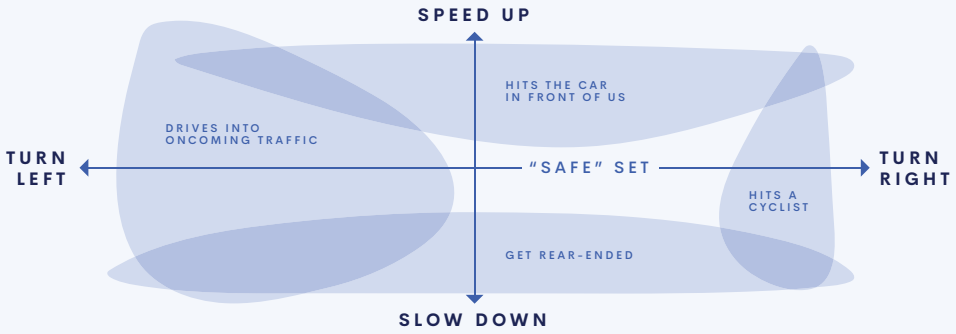
Today, virtually all high-performance autonomous vehicles use LIDAR in addition to cameras. But the role of cameras in the sensor suite is rapidly expanding—computer vision technology is improving. For example, May Mobility's vehicles have progressed from just a couple cameras to nine over just the last two years.

Camera-only systems will likely be technologically feasible by the end of the decade. But it remains an open question of whether camera-only systems will be the best approach—that will depend on the performance and cost of sensors that cameras are competing with.

It's important to note that, at least today, the relatively low cost of camera sensors is offset by the comparatively high cost of the computers (usually GPUs) to process the images. Not only are these computers expensive, but they also consume large amounts of power and generate heat—both of which are significant practical issues in scaling production volumes.

What about safe-set approaches like Mobileye?

The basic idea behind Mobileye's approach is to start with the set of all actions that a vehicle can take and to subtract away those that could lead to a collision. To do this, you make worst-case assumptions about the behavior of other agents around you. For example, the car in front of you could slam on the brakes, which limits how fast you should be accelerating.



Safe-set approaches have been quite successful in several control domains, and elegantly lead to provable performance guarantees when you can truly make worst-case assumptions about other agents. But they are a poor fit for autonomous driving for a simple reason: if you make worst-case assumptions about what others can do, you'll quickly find that there's no safe action for you at all. Accelerate too fast and you'll hit the car ahead; accelerate too slowly and you'll hit the car behind you.

A natural impulse is to relax the idea of "worst-case" to "reasonable worst-case". But there's no rigorous or scalable way to define what "reasonable" means, and you inevitably re-introduce all of the edge cases that rule-based systems suffer from.

These kinds of approaches are quite sensible for driver-assist features, where a human can step in when the system runs out of options. But there's little evidence that this kind of approach can be applied successfully to level 4+ autonomous driving.

Trajectory vs. Policy-based vehicles

Most AV systems reason in terms of trajectories: "*this pedestrian will be at these locations at these times.*" You can think about the trajectories of other agents as obstacles which exist in both time and space. For many AVs (including systems I've built in the past), this makes motion planning a geometry problem: find a time-space trajectory for the autonomous vehicle that does not collide with the time-space trajectories of the other agents.

The problem with trajectory-based systems is that they do not capture the conditional dependence of the behavior of other agents.

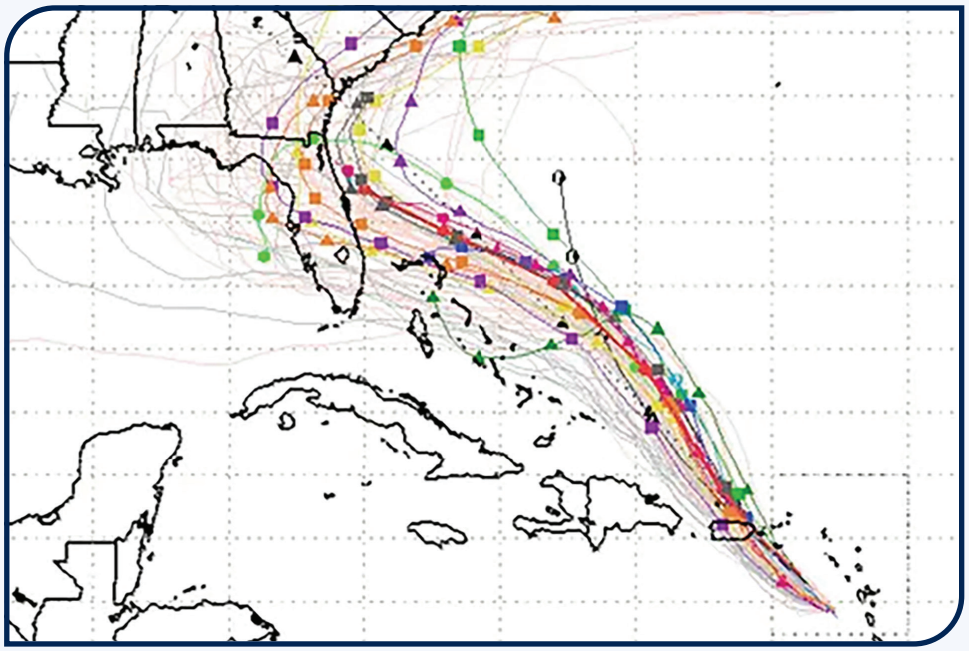
There are two key factors:

1. We cannot predict precisely what other agents will do, due to both uncertainty in our own perception and reasoning capabilities, as well as (pesky) free will.
2. The behavior of other agents depends on the behavior of other agents, often in very brittle ways. At a four-way stop, if one car goes, the other is likely to stop, and vice versa. This dependency is impossible to capture with fixed trajectories.

Of course, a set of trajectories can capture a single variation of interactions, but can't capture the range of possible interactions. And driving is a highly dynamical system: changes to one agent's behavior can cascade into massive changes to the behavior of all the other agents. In short, this means that a trajectory planned by an autonomous vehicle may not actually be collision-free.

A far better approach, unique to May's vehicles, is to represent the behavior of all agents in terms of "policies". A policy answers the question "*given what the world looks like now, what would you do next?*" Roughly speaking, a policy is the combination of an intention ("*I want to turn right*") and a behavior profile ("*I favor calm, comfortable accelerations*").

MPDM is based on policies rather than trajectories, and this is the key to making high-quality predictions about the likely interactions between agents. Critically, we're not always sure about what other agents will do—will the pedestrian step off the curb or not? Our approach allows us to pick policies that are safe under multiple possible outcomes. In fact, we can plan over outcomes that are multi-modal, not just minor variations on a single scenario.



This leads to an interesting property of May's vehicles. Most AVs display the path that the vehicle will take for the riders to look at—perhaps a little blue line drawn on a screen in the back seat. But because we plan in policy space rather than trajectory space, we do not know what path the car will take. Instead, we have several hundred possible paths, each based on simulations of what might happen next. (These are analogous to the "spaghetti plots" used in forecasting hurricanes.) We pick a safe policy by considering all of these outcomes. We still display a path ahead, but for us, it is merely the most probable prediction, but what the vehicle will actually do depends on what other road users do.

What are the power/cost considerations of MPDM technology?

With the additional training example generation and learning, one might reasonably assume that the power consumption requirements of May vehicles would be higher than other kinds of AVs. Counter-intuitively, this is incorrect!

May Mobility vehicles do spend more power learning online than other systems (since other systems don't learn online). But the major source of power consumption in AVs is in the perception system.

If you are building an AV that thinks in trajectory space, your system will be very sensitive to errors in the trajectory predictions of other agents. If you're wrong about where a pedestrian is going to be, then you're wrong about where the autonomous vehicle can safely drive!

Because of this sensitivity to failures in the perception system, trajectory-based systems often try to squeeze out the absolute best performance possible from their sensors—going far past the optimal power/performance point.

May's system is perfectly comfortable dealing with the possibility that road users might do multiple things, and it makes plans that are safe overall—not just with respect to a single trajectory. Because a policy-based approach isn't fixated on the accuracy of a single prediction from the perception system, we can operate our perception at the "sweet spot" in the power/performance curve. This saves a massive amount of power, and also reduces the cost of the computers in our bill of materials.

In short, May's AV technology operates at a fraction of the power of many other AV companies: the additional power used by our in-situ learning system is massively offset by savings in the perception system.

The industry will ultimately need to develop autonomous driving systems in the 100–200W range, and May's stack is far closer to this target than most AV companies.



Direct learning vs. Time-based reasoning

Step-by-step simulation is an incredibly powerful computational framework that can greatly reduce the complexity of models required, and thus the amount of training data required to solve a problem.

Suppose you are given an initial position and speed of a set of planets, stars, and galaxies, and your job is to compute their position after a million years. You may be familiar with the idea of chaotical systems, and N-body problems like this are highly chaotic: small changes to the initial position of a small moon can have massive long-term impacts on the output.

You could pose this problem as a learning problem $y=f(x)$—predict/learn the state after a million years (y) given the state at the beginning (x).

A conventional neural network architecture will attempt to learn this in a "single shot"—a direct mapping between the initial and final positions, without considering the sequence of events that would occur in between. But since small changes to initial conditions can dramatically change the final configuration, it makes the learning problem extremely difficult. *(see "What makes a learning problem difficult?")*

For an N-body problem like this, however, you could make nearly perfect predictions simply by applying Newton's laws, with no learning whatsoever. After all, we understand how planets and stars move due to gravity. By repeatedly computing the gravitational forces between the planets, we can make accurate predictions. The key idea is that the repeated application of a simple model can capture very complex dynamics that are very hard to learn in a single shot.

I believe this is one of key reasons that humans find driving to be easy: we can imagine the future—one step at a time—by applying relatively simple understandings of how pedestrians, bicycles, and other cars will behave. Humans are not as predictable as planets, but even a relatively simple model can become quite powerful when applied in a simulation framework.

And this is part of how MPDM's performance is so surprisingly good—repeated application of our behavioral models and policies can easily match the performance of a much more complex single-shot learner.

How does sampling outcomes deal with the combinatorial explosion of many coupled agents?

This one is a bit esoteric, but if you're still reading, hi!

When May vehicles sample future outcomes, we are limited in the number of scenarios we can consider. If the scene is (nearly) empty, we can easily consider every possible permutation of the actions of other agents. Consider two agents with two potential actions each: a pedestrian might wait or go straight; a car might pass the bicyclist or not. This gives a total of 4 possibilities. We sample about 1000 scenarios per second, so this is easily achievable (many times over).

But what happens if there are 20 or 100 agents? The number of possible permutations could easily exceed 1000. Are we in trouble?

If every agent could affect every other agent, then yes — this would be very challenging! But this is not how the driving domain works: not all agents are tightly coupled. For example, two consecutive cars in a lane probably are tightly coupled, but two cars driving away from each other probably aren't.

We can think of these agent couplings as a graph in which nodes represent agents in the world and edges represent dependencies in their behavior. If every agents' behavior depends on every other agent, then the graph is fully connected. But in practice, agents that are far apart, or are in non-conflicting traffic lanes, will not be dependent and thus there would not be an edge in the graph between those two agents.

For example, suppose you have four agents (A, B, C, and D), each with 10 possible policies. Naively, this gives us a complexity of 10^4. Let's suppose that A and B are tightly coupled, and C and D are tightly coupled, but there are no strong couplings between those two cliques. The complexity of any clique is exponential in the size of the clique, but the cliques are half the size. And so the complexity here is much lower, only $2*10^2$. Sparsity in this graph gives rise to massive reductions in complexity by essentially dividing the problem into independent sub-problems.

One could try to explicitly construct this graph to find cliques/clusters in the graph. But this is unnecessary: if you sample enough for the actual complexity of the problem, you don't actually need to build the graph in the first place. In real-world situations, cliques are relatively small and so a relatively small number of samples is sufficient.



Is MPDM related to other AI approaches?

MPDM is a form of online reinforcement learning, and our patents describe how to use this in a Monte Carlo Tree Search (MCTS) framework. These same basic ideas have been deployed with great success by DeepMind to solve games (Atari, Go, Chess) and to fold proteins. MPDM in particular frames the problem in a way that supports in-situ learning, whereas DeepMind's learning happens in the data center.

Reaction Time and Emergencies

The mark of a good driving system is one that avoids close calls in the first place. If you can predict safety-critical situations well in advance, you can arrange for your autonomous car to be far away from those "pinch" points . Often, only tiny modulations of speed or heading are needed to turn a potential collision into an unremarkable drive down the street. This is where MPDM excels.

Of course, emergency situations can arise without warning, like a child running out into the street from behind a parked car. In such a case, the safety of your system comes down to reaction time: how quickly can your car begin deceleration once an emergency situation is detected?

You might initially think that MPDM, because it runs at about a 200ms update rate, might delay a vehicle's reaction significantly. But this is not correct. MPDM plans at the level of driving "strategy", but every policy has the ability to react to emergency situations: we don't need to elect a new policy to start braking for the emerging child.

In fact, in many autonomous vehicle systems, the vehicle's motion planner can be a limiting factor in reaction time. Many motion planners operate at about 10Hz, or about 100ms. That's the delay that occurs after the child is detected and before braking can begin. In contrast, the polices we build do not compute full trajectories and can often begin an emergency response faster than a traditional system.

Where did MPDM come from?

MPDM was originally invented at the University of Michigan by the founder of May Mobility, with key patents exclusively licensed to May Mobility. May has extended and improved these methods over the last 7 years, building a significant patent portfolio.

As a reinforcement learning (RL) algorithm, MPDM draws inspiration from the vast RL literature, particularly methods based on Monte Carlo search.

But two less obvious solutions include:

- RODNEY BROOKS' SUBSUMPTION ARCHITECTURE
  This was a formative idea in robotics about having multiple control strategies "compete" for control. Brooks was able to build some of the first robust walking robots by having leg joints dynamically switch between multiple control laws. The arbitration mechanism between these policies was much simpler than the simulation-based approach used by MPDM, owing to the computational constraints of systems at the time.
- BEN KUIPERS' SPATIAL SEMANTIC HIERARCHY (SSH.)
  A robot might build a map of an environment and know that applying controller A is an effective way to move from position 1 to position 2; applying controller B might get you from position 2 to 3. These controllers could be quite simple, such as wall-following behaviors. By building a graph of how different controllers can connect distant places, you can build up a lot of capability from very simple building blocks. One appealing aspect is that there are very few constraints on the structure of those building blocks—they can even be opaque black boxes. That is incredibly useful when modeling interactions between road users—those interactions may not lend themselves to simple analytic models.



The best way to see MPDM's advantages is to join us for a ride.

Please contact our investor relations team to learn more.

