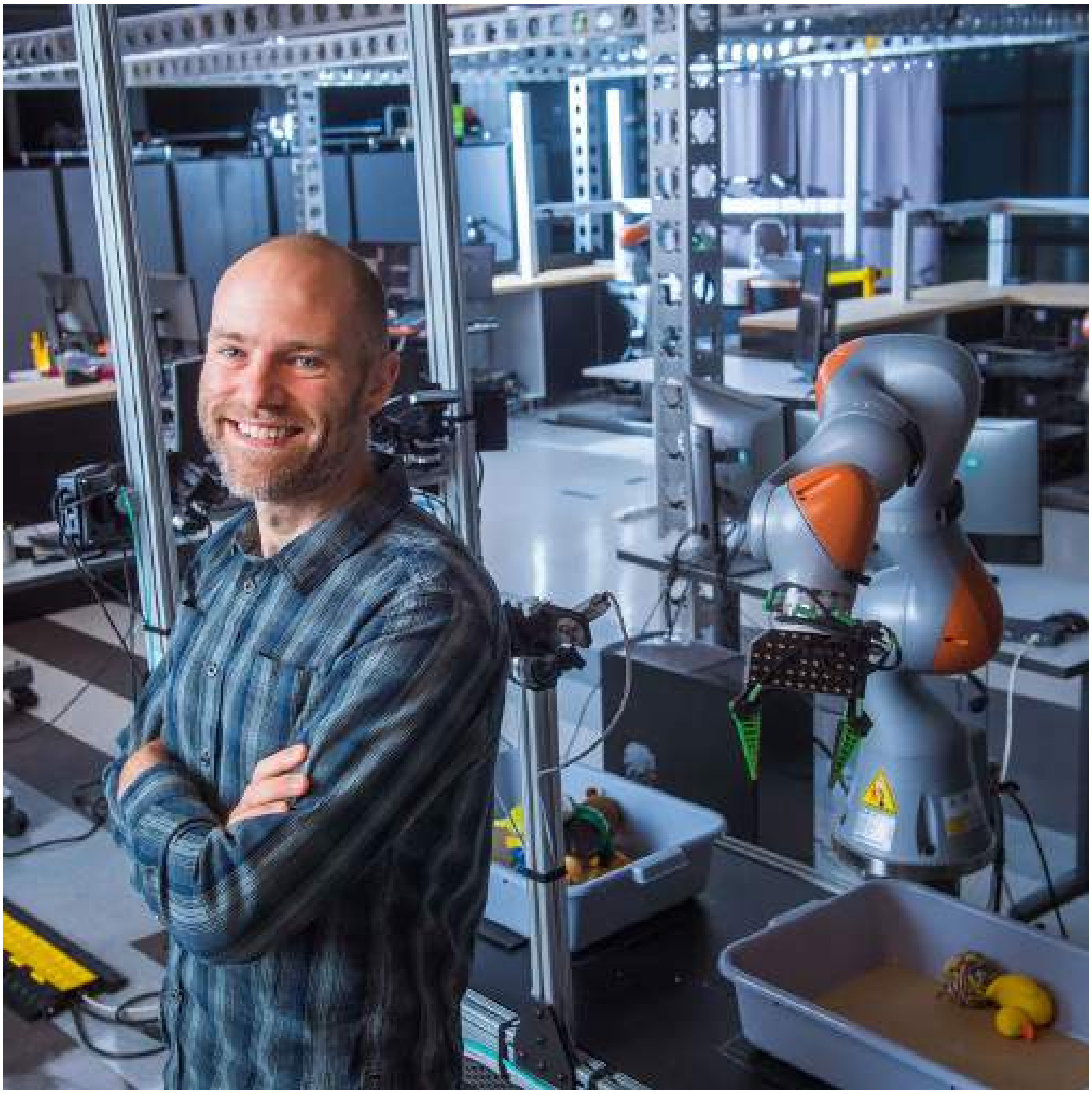
# Drake: Model-based design in the age of robotics and machine learning

Toyota Research Institute · Follow
Published in Toyota Research Institute
9 min read · May 3, 2021

By: Russ Tedrake, V.P. Robotics Research

A rendered simulation in Drake showcasing our advanced contact modeling.

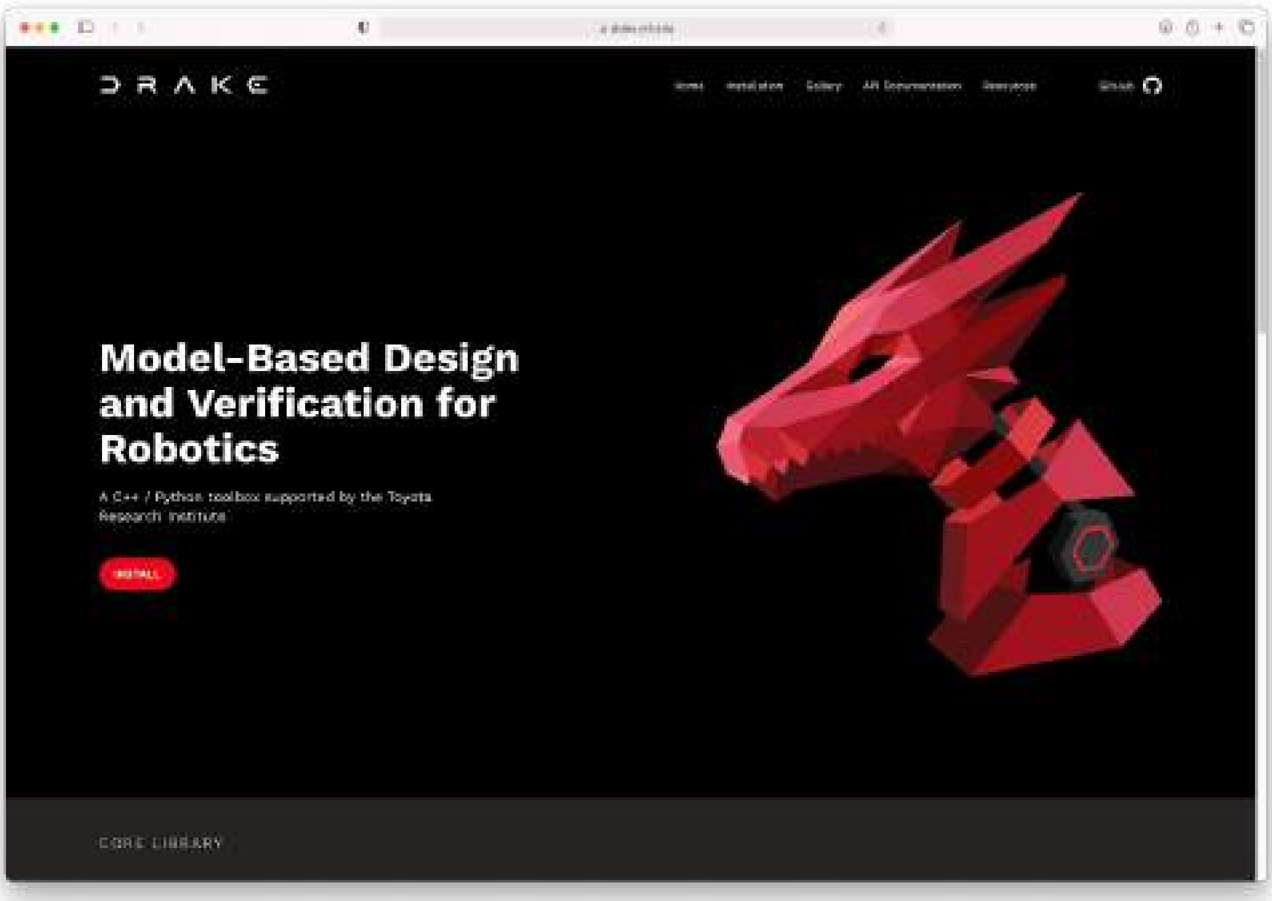
When I joined Toyota Research Institute (TRI) more than five years ago, I believed that an industrial research lab like TRI could make fundamental contributions to robotics that would be hard to make in an academic lab or a startup. And I joined with a commitment that we would share our best tools and results with the world through open-source software.



Just before joining TRI, I competed in the DARPA Robotics Challenge to program a humanoid robot for nearly-autonomous operation in a disaster response scenario. This experience gave me a deep appreciation for the value of software engineering and helped me realize the world has never seen truly mature implementations of the best ideas from control theory, machine learning, mathematical optimization, and verification applied to robots at scale.

The introduction of scale and real-world testing poses a host of basic research challenges that simply aren't visible in simpler prototypes. For example, to study rare failures, one needs a system that is successful 99.9% of the time when performing a relevant task (like loading the dishwasher for in-home robots). That's very hard to achieve in academia. Another challenge is fleet learning. Right now, many of us are thinking about programming a single robot and making it learn efficiently from its experiences. But that isn't actually the problem we need to solve for robotics. How does the paradigm change when we're working with an entire fleet of robots that are able to pool their experiences?

Roboticists are ready to tackle these challenges, and industrial research labs like TRI have an important role to play. The rise of deep learning has been fueled not only by a few great conceptual ideas, but also by a major investment in software and hardware tools by the top industry labs and by their decision to make these tools open source. Deep networks are now an incredibly important part of a roboticist's toolkit, but they still represent only a fraction of that toolkit.


DRAKE
Home
Installation
Gallery
API Documentation
Resources
GitHub
Model-Based Design and Verification for Robotics
A C++ / Python toolbox supported by the Toyota Research Institute
INSTALL
CORE LIBRARY

*Drake* is TRI's attempt to provide mature, open-source tools for another large portion of the roboticist's toolkit, complementary to the deep learning toolboxes, to further buttress advanced research in the field. It is a library (in C++ with Python bindings) with three major components: 1) the **multibody dynamics** engine, 2) the "**systems framework**" for organizing and combining system models from a library into a block diagram, and the 3) **optimization framework** for mathematical programming. Each of these components is important for robotics research; the vision for Drake is to facilitate making powerful combinations of all three components within a single application.

## Multibody dynamics

One of the most prominent and most important components of Drake is the physics engine. After working on legged robots for many years, I came to appreciate that simulating robots *manipulating* diverse objects (with their hands) is more difficult than simulating a legged robot. Both systems make and break contact with the environment, but the diversity of contact geometries and configurations in manipulation is much greater and the numerics are much worse. Expectations are high, too. If a simulated humanoid robot's foot slides a little when coming into contact with the ground, nobody will notice. But if a coffee mug slowly slides out of the robot's hand, that's bad news. Achieving accuracy and efficiency requires a nuanced relationship between the multibody equations and the numerical integration codes.

Drake has an incredible dynamics team working on the problem, led by Michael Sherman who has been a lead architect/developer on a number of mainstream simulation packages like SD/FAST and Simbody. The Drake team has committed to closing the "Sim2Real gap" for dexterous robotic manipulation. Today, you'll find a state-of-the art implementation of physics for both rigid and compliant bodies, with a particular emphasis on robust numerics for contact mechanics to fight the numerical instabilities that plague most physics engines. You can read more about this in our blog post about **[Rethinking Contact Simulation]**. We are now working on large-deformation models for soft-body simulation, and aim to support thin elements (like cloth) soon after.

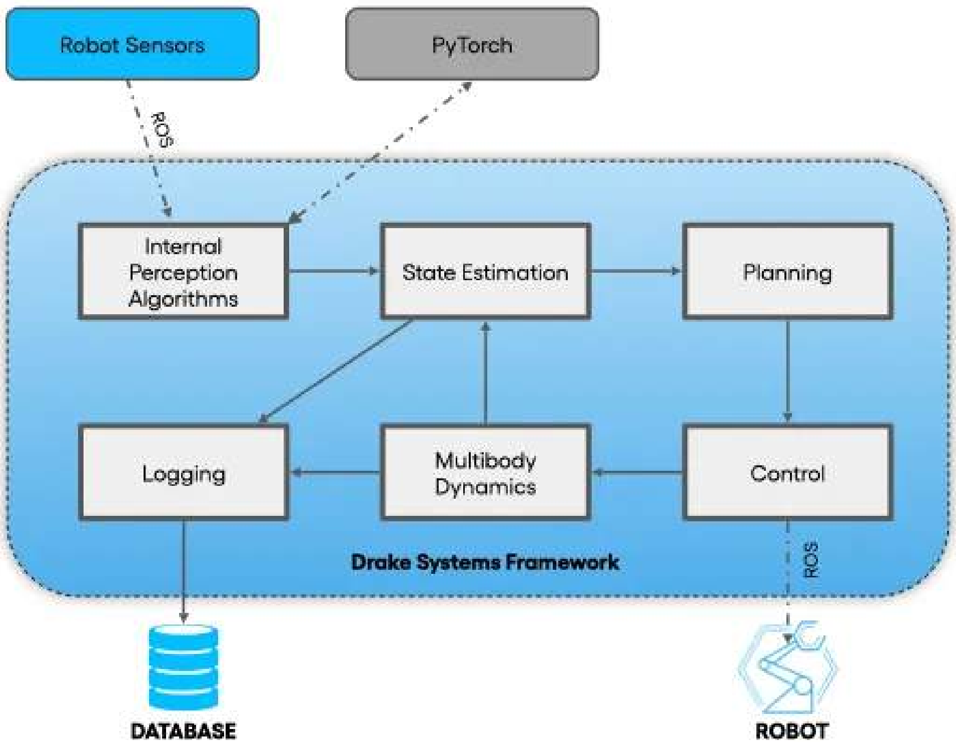
00:09

Most simulators use point contacts, but we take integrals over the contact surfaces (visualized on the right) for more robust computation. We'll give more details on this “hydroelastic” contact model in a future blog post.

The Drake developers have a philosophy of rigorous test-driven development. The governing equations for multibody physics are well known, but there are often bugs in a complex engine like this. If you scan the codebase, you will find unit tests that contain comparisons with closed-form solutions for nontrivial mechanics problems like a tumbling satellite, countless checks on energy conservation, and many other checks that help the rest of the team focus on manipulation with the confidence that the multibody models are implemented correctly.

Importantly, this dynamics engine is not only for simulation. It is also built for optimization and for control. The exact same equations used for simulation can be used to compute forward or inverse kinematics and Jacobians. They can also be used for more complex queries like the gradient of an object's center of mass. We provide smooth gradients for optimization whenever they are available (even through contact). Drake also supports symbolic computation, which is very useful for structured optimization and for use cases like automatically extracting the famous “lumped parameters” for parameter estimation directly from the physics engine. This would be very hard to do in most physics engines!

## Systems Framework



Algorithms for perception, estimation, planning, and control can be combined with the physics engine and robot hardware using block diagrams.

Although the physics engine is important, modeling robots requires much more than just realistic and performant physics. Drake also provides a library of sensor models, actuator models, low-level controllers, and low-level perception algorithms. For example, we have models for relatively simple sensors, like a rotary encoder or inertial measurement unit. But we also have complex sensor models like simulated cameras which require a full rendering pipeline. We even support a variety of camera models, including high-speed lightweight shader-based renderers that are ideal for many simulation workflows, and full physics-based renderers that can be used for training and testing perception algorithms.

Building on the far-reaching success of model-based design frameworks like Simulink and Modelica, we encapsulate each of these models as a “system” that can be easily composed into more complicated systems in a block diagram. To be a fully compliant element in the Drake systems framework, each model must declare all of its state variables and pull any randomness through an input port.

Compared to authoring a ROS node, this asks a little more of the system author up front. But the downstream benefits are immense. The extra structure enables fully deterministic replay and more advanced design and analysis techniques. Once you have a system, whether it’s a simple system or a complex diagram, you can easily use that system in a multiprocess network passing framework (like ROS), or run it all in a single process for deterministic execution and debugging. Drake's collection of carefully vetted models is growing continuously, and we welcome your contributions!

The systems framework provides the abstraction and encapsulation that allows a mature software project to scale. There are numerous robotics companies, from startups to large industry players, that are using Drake now in production. They often tell me that they started using Drake because of the physics engine, but that it's the systems framework that has really enabled them to grow and scale.

The systems framework provides a software engineering abstraction, but it also provides a powerful mathematical abstraction that enables more advanced algorithms. When I think of a simple, discrete-time dynamical system in state-space form, I think of equations of the form:
$x[n+1] = f(n, x[n], u[n], p, w[n]),$
$y[n] = g(n, x[n], u[n], p, w[n]),$
where $x$ is the state, $u$ is an input, $y$ are the outputs, $p$ are the parameters, and $w$ represents (potentially random) disturbance inputs. Drake supports much more complex systems than this, with mixtures of continuous and discrete dynamics, event handling, and abstract (structured) state types. But for systems that do admit this form, including the composition of a time-stepping model of multibody dynamics with numerous sensors, and even a feedback controller, Drake goes to some lengths to make the simple structured form of the equations available.

Given a clean mathematical model, it becomes clear how we formulate solutions to some fairly sophisticated questions about the model:

- Simulation is solving for $x[\cdot]$ given $x[0]$, $u[\cdot]$, and p.
- Planning or trajectory optimization is searching for $\{u[\cdot],x[\cdot]\}$ (with $w[\cdot]=0$). Robust planning includes taking an expectation or worst case over w.
- State estimation is searching for $\{x[\cdot],w[\cdot]\}$
- System identification is searching for $\{p,\ w[\cdot],$ and often $x[\cdot]\}$
- Stability analysis is (for $w=0$), for instance, finding a set of initial conditions $x[0]$ for which $lim\ n\rightarrow \infty,\ x[n] \rightarrow 0$. Stochastic stability analysis asks, e.g. the probability of leaving a region over some finite-time horizon.

- Verification / falsification asks if there exists a *w[·]* such that $\exists$ *n s.t. x[n]* $\in$ failure set.

Some people would say that the problems we are tackling now in robotics are too complex to be treated with clean mathematics. I strongly disagree. I think it's precisely *because* the systems are so complex that we must think more clearly about our formulations. Drake was designed to help bridge the gap between the clean mathematics and the incredibly complex problems in robotics.

## Optimization Framework

The last major component of Drake is the optimization framework, which provides a front-end to easily write mathematical programs and then dispatch them to open-source or commercial solvers. In this sense, the optimization framework plays a role like CVX or Yalmip in MATLAB, and JuMP in Julia. We support a hierarchy of convex optimization problems, mixed-integer optimization, and general nonlinear optimization.

```
prog = MathematicalProgram()
x = prog.NewContinuousVariables(2)
prog.AddConstraint(x[0] + x[1] == 1)
prog.AddConstraint(x[0] <= x[1])
prog.AddCost(x[0] ** 2 + x[1] ** 2)
result = Solve(prog)
```

**mathematical_program_sample.py** hosted with ❤ by **GitHub** **view raw**

Formulating a simple mathematical program in Drake. We also make it easy to add costs and constraints from the physics engine, and to solve common optimization problems from control.

The systems framework and the optimization framework share a common core of templatized scalar types to support automatic differentiation and symbolic computation. Perhaps the best way to illustrate the power of this is through an example: the direct trajectory optimization code. We can easily set up a trajectory optimization problem by handing it as a system (which can be an entire diagram), and then adding costs and constraints. If the system happens to have linear state dynamics, and the costs and constraints are all convex, then the trajectory optimizer will automatically dispatch the problem to a high-performance convex optimization library. If not, the optimizer will dispatch to a more generic nonlinear solver automatically.

Some people today believe that stochastic gradient descent (and its variants) are the only algorithms required in robotics. Drake is wired to provide analytical gradients, even in complex systems. I personally believe there are also rich untapped connections between mechanical systems (both smooth and non-smooth) and the fields of combinatorial optimization and algebraic geometry. Drake is a great playground for exploring these ideas, and applying them to complex systems.

### Algorithms for perception, planning, and control

These three core components fit together to enable cutting edge research in advanced algorithms for robotics. For some of the more mature algorithms, we provide implementations in Drake. Some examples include: solutions to Lyapunov and Riccati equations for linear systems, various transcriptions of trajectory optimization, value iteration, and sums-of-squares optimization for reachability and region of attraction analysis.

But not all of these can nor should live in Drake. Some of them have library requirements that would be too heavy to include directly in Drake. For instance, we don't demand *PyTorch* nor *ROS* as a dependency, but provide examples of how to use them together. A great example of this is TRI's work on **[Large Behavior Models]** which builds a powerful imitation learning framework in PyTorch enabled by a very solid robot control foundation built in Drake. We are now starting to build the ecosystem of shared tools and repositories using Drake as a library. We have a number of tutorials on the Drake website and are writing more. I teach two advanced robotics classes at MIT, one called Underactuated Robotics and the other called Robotic Manipulation; both have extensive course notes with Jupyter notebook examples running Drake on Google Colab.

### Drake is ready

I have been conservative about advertising Drake. I wanted it to be excellent, and I worried about the cost of supporting a large user base before we were ready. Inevitably, we still have a long list of ideas and improvements, but I am truly proud of where Drake stands right now and the science and engineering approaches it can enable. We've developed a mature software library, battle-hardened against a multitude of real-world experiments and extensive Monte-Carlo (and adversarial) testing. We provide regular minor releases and have committed to deprecation timelines that support use by serious companies in development and production.

00:36

"Dexai uses Drake to solve multibody dynamics problems involving contact. We use robots to manipulate foodstuffs which have non-linear dynamics and often require torque control for effective manipulation strategies. Drake allows us to perform offline optimization of our trajectories subject to a range of constraints, as well as run real-time control loops on our robots." — David Johnson, Founder and CEO of Dexai.

If you have needs for an industrial strength robotics toolbox that can enable your research and engineering applications, then take a serious look at Drake. You can connect with us on Github and Stack Overflow. We welcome your open-source contributions and your feedback can influence the features/algorithms we prioritize.

I think the future is bright for our best algorithms enabling incredible robots.

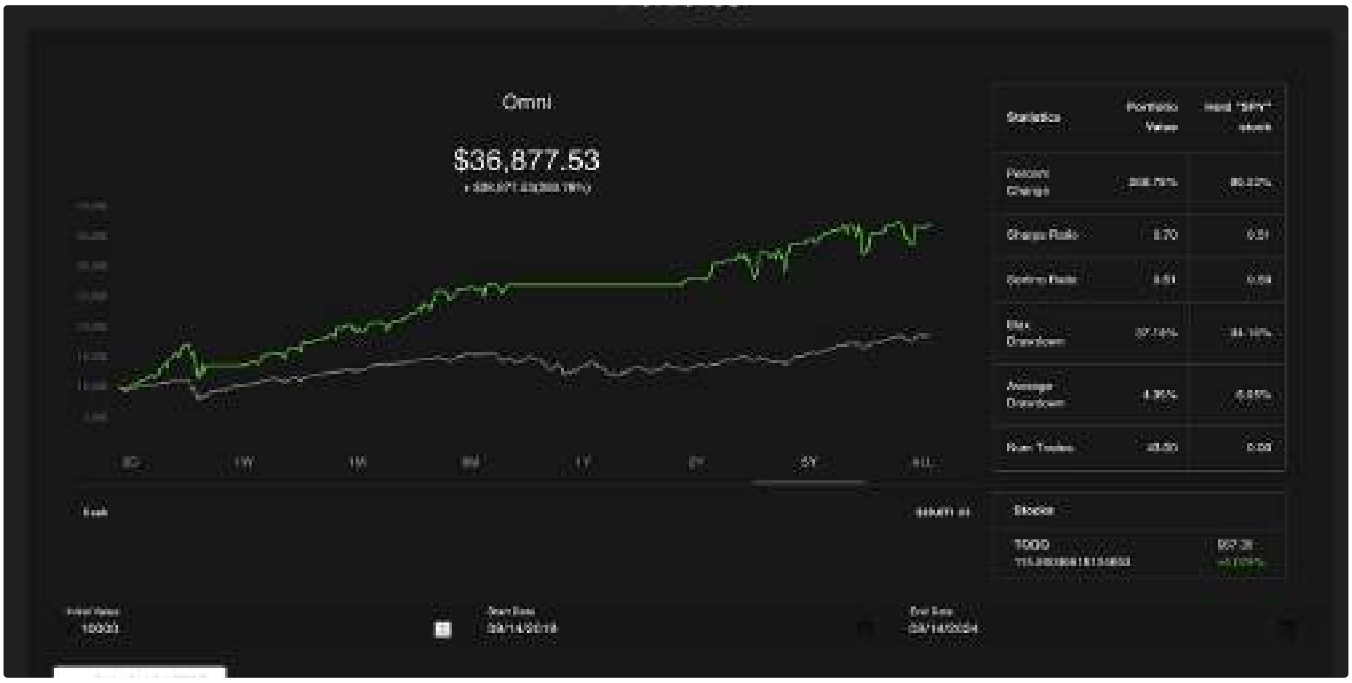
Robotics · Optimization · Control · Model Based Design · Machine Learning

Feb 28 229 1

See all from Toyota Research Institute

See all from Toyota Research Institute

## Recommended from Medium



In DataDrivenInvestor by Austin Starks

### I used OpenAI's o1 model to develop a trading strategy. It is DESTROYING the market

It literally took one try. I was shocked.

Sep 16 7.8K 189