

[Open in app](#) ↗

Search



★ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)

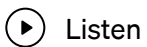


Particle Filter Part 1 — Introduction



[Mathias Mantelli](#) · Following

7 min read · Aug 5, 2023



Listen



Share

... More

This part of the series has been written in collaboration with [Sharad Maheshwari](#) and [Rachit Gandhi](#).

Hellooo guys!

After our previous series on [Kalman Filter](#), let's talk about the Particle Filter — yet another state estimation technique, but stronger and more in tune with the real world.

This series has the following parts —

- Part 1 — Introduction to the Particle Filter — Why and What (we're here);
- Part 2 — Building intuition and introducing the equations;
- Part 3 — Formal algorithm and a practical example;
- Article 4 — Particle Filter in action with code

And as usual, we focus on building an example-based intuitive understanding before

going formal. And we of course end with building a Particle Filter from scratch in Python.

No drab jargon-y definitions to bore you right off the bat. That's how we do it 😊

Why the Particle Filter?

As we said, we made a series on the Kalman Filter.

Then why the Particle Filter now, you ask? Isn't our beloved Kalman Filter perfect for estimating states?

Ummm, well, not always.

The Kalman Filter is based on certain assumptions which might not always hold true in the real world. That's where the Particle Filter comes in. But let's not get ahead of ourselves and first jog our memory about concepts like the Bayes Filter and the Kalman Filter. The discussion will set a great springboard for us to dive (pun intended) into the Particle Filter.

Let's do this!

Quick Recap — Bayes Filter, Kalman Filter

Before getting into the “Why” of the Particle Filter, it is imperative to quickly remember the Bayes filter (henceforth referred to as BF) and the Kalman Filter (henceforth referred to as KF).

Note — If you are new to BF and KF, we've covered them [in this long series](#) on the Mighty “Kalman Filter”.

Bayes Filter — BF is a powerful state estimation tool that combines a state propagation model and sensor measurements to estimate the state of the system. It forms the basis of the entire Kalman Filter family!

For instance, in the example we used in our detailed [article on BF](#), we use it to estimate the robot's position, given that we know the motion commands and the sensor measurements. After the formal derivation, we ended up with two steps in the filter — the prediction step and the update step.

- Prediction Step — takes into account the previous state belief and the control data to estimate the predicted belief using a mathematical state propagation model.
- Update Step — combined prediction output and sensor measurement to estimate the current state

The two steps are repeated in a recursive fashion to provide an efficient method for tackling state estimation problems in dynamic environments. BF has been used in many different fields, and it is quite popular in the field of robotics.

In conclusion, the BF serves as the foundation for state estimation, allowing us to integrate prior knowledge with new observations in a probabilistic framework. The KF and PF are specific algorithms that leverage the principles of the BF to estimate the state of linear and nonlinear systems, respectively.

Okay, so that was BF in a nutshell. As we said, the KF builds on top of the BF. How so?

Kalman Filter — Like BF, KF is a recursive mathematical algorithm to estimate the dynamic system state from a series of noisy measurements. At its core, the KF is equivalent to BF + Gaussian distribution assumptions (all distributions are Gaussian) + Linear model assumptions (both motion and observation models are linear).

Note — Please check our post on [KF Intuition](#) and [KF Equations](#) if you wish to go deeper.

In scenarios where these two assumptions are strongly valid, the KF provides the optimal estimation of the system's state. Therefore, if the motion and observation models are linear, we only need to sequentially estimate the new states as new measurements arrive, making the KF well-suited for real-time applications.

Do you see the problem here?

The KF is based on two assumptions — gaussian distributions and linearity. But the real world doesn't follow these assumptions in most cases. And that's where the KF falls short and underperforms.

But how do we estimate the state without such assumptions? — The Particle Filter.

Particle Filter — Another state estimation approach

With a primer on the BF and the KF, we've set the stage for our Particle Filter — the main topic of this series. And that's some fresh territory. Once you're through, we expect you to have a deeper intuitive and formal understanding of the Particle Filter (henceforth referred to as PF) 😊

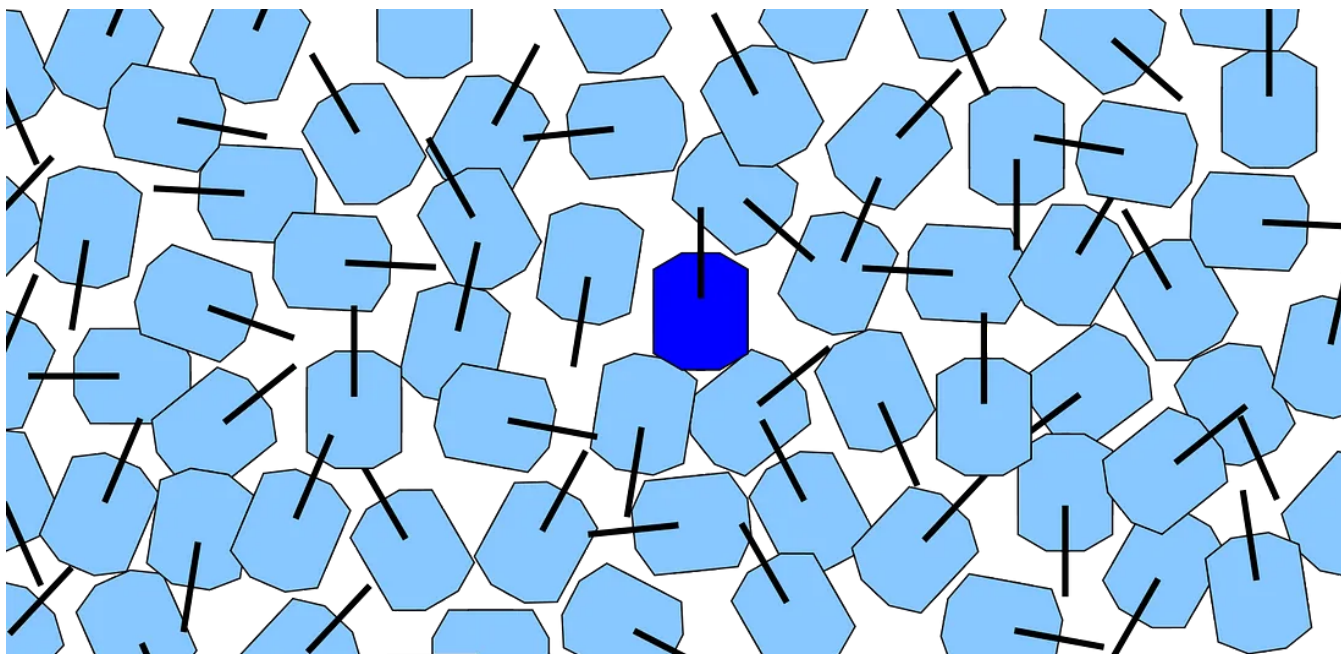
The PF is considered an “alternative” to the KF primarily because it offers a different and more generic approach to state estimation. While the KF is well-suited for linear and Gaussian systems, it can struggle with nonlinearities and non-Gaussian noise. In contrast, the PF thrives in these complex scenarios and, therefore, serves as a viable alternative where the KF is not applicable.

The PF, also known as Sequential Monte Carlo (SMC) or Monte Carlo Localization (MCL), is a powerful state estimation technique used to estimate the state of a system in the presence of non-linearities (non-linear state evolution) and non-Gaussian noise. It can handle a broader range of complex and dynamic systems.

While we usually avoid jargons in the first part of a series, we couldn't help but have a few in the next paragraph. But we promise to build intuition in our next part. So don't worry if it doesn't make complete sense just yet 😊

At its core, the PF represents the state of a system using a set of particles, each particle representing a hypothesis of the true state (as we show in the figure below). These particles are propagated through time using the system's motion model, and their weights are updated based on the likelihood of the measurements given their corresponding hypotheses. Through resampling and importance weighting, the PF adapts the particle set to focus on regions of higher probability, providing an effective and flexible state estimation approach. The PF's particle-based

representation and Monte Carlo sampling approach make it more versatile in handling various types of nonlinearities and non-Gaussian noise, making it a powerful alternative for state estimation tasks in modern applications.



The PF holds significant importance for state estimation due to several key reasons:

1. **Nonlinear System Handling:** Many real-world systems exhibit nonlinear behaviors, making the PF a valuable tool for state estimation in such scenarios. It allows for an accurate and robust estimate of nonlinear system states, which is beyond the capabilities of the KF.
2. **Non-Gaussian Noise Tolerance:** The PF can effectively handle non-Gaussian noise distributions, which are common in real-world applications. This enables the filter to capture more realistic noise characteristics, leading to improved state estimation accuracy.
3. **Multimodal Distributions:** In situations where the true state distribution is multimodal, the PF's ability to represent multiple hypotheses using particles becomes advantageous. It excels at capturing and tracking multiple modes in the state space, providing a comprehensive estimation approach.

Hence, the PF offers several advantages over traditional estimation methods:

1. **Nonlinearity Handling** — The PF can effectively estimate states in nonlinear systems, making it applicable to a broader range of real-world scenarios.
2. **Non-Gaussian Noise Tolerance** — It can accommodate non-Gaussian noise distributions, allowing for accurate state estimation in situations where Gaussian assumptions fail.
3. **Multimodal Distributions** — The PF's ability to represent multiple hypotheses using particles allows it to track multiple modes in the state space, making it well-suited for estimating complex and multimodal distributions.
4. **Flexibility and Generalization** — The PF's adaptability to various types of systems and noise distributions makes it a versatile state estimation tool for diverse domains.
5. **Implementation Simplicity** — In some cases, implementing the PF can be easier than the KF, particularly for complex nonlinear systems, where deriving linearised models for the KF may be cumbersome.
6. **Real-time Performance**: While the computational cost of the PF increases with the number of particles, it can still provide real-time estimation for many practical applications.

In conclusion, the PF is a crucial and versatile state estimation technique, offering a powerful alternative to the KF in scenarios involving nonlinear dynamics and non-Gaussian noise. Its ability to handle complex systems, multimodal distributions, and diverse noise characteristics makes it a valuable tool in a wide range of applications, from robotics and autonomous vehicles to finance and medical diagnostics.

Summary

With a naive introduction and some context using BF and KF, we've just taken the first step here in our series on the Particle Filter (and state estimation in general).

This post focuses on building an intuitive understanding of PF, which will be further developed in the next article. But for now, the takeaways here are:

- BF is the backbone of both KF and PF and understanding this is a great starting point;
- KF is important for the state estimation problem, but its assumptions make it imperfect for many real-world estimation scenarios;
- PF shows up as a robust alternative for such cases where KF does not work;
- As PF derives from BF, it also works based on the two-step loop — the prediction step with the motion model, and the update step with the measurement model.

Now that we have a good understanding of these concepts, we move on to building our intuition for the Particle Filter and unveiling the equations.

But don't worry, stay with us. This is just the beginning and there is a lot more to come.

See you guys!

Ciao!

[Particle Filter](#)[Robotics](#)[Localization](#)[Monte Carlo Method](#)[Monte Carlo](#)[Following](#)

Written by Mathias Mantelli

132 Followers

mathiasmantelli.com

More from Mathias Mantelli






 Mathias Mantelli

Kalman Filter Series — Introduction

This article has been written in collaboration with Sharad Maheshwari.

16 min read · May 19, 2023

 51  3



 Mathias Mantelli

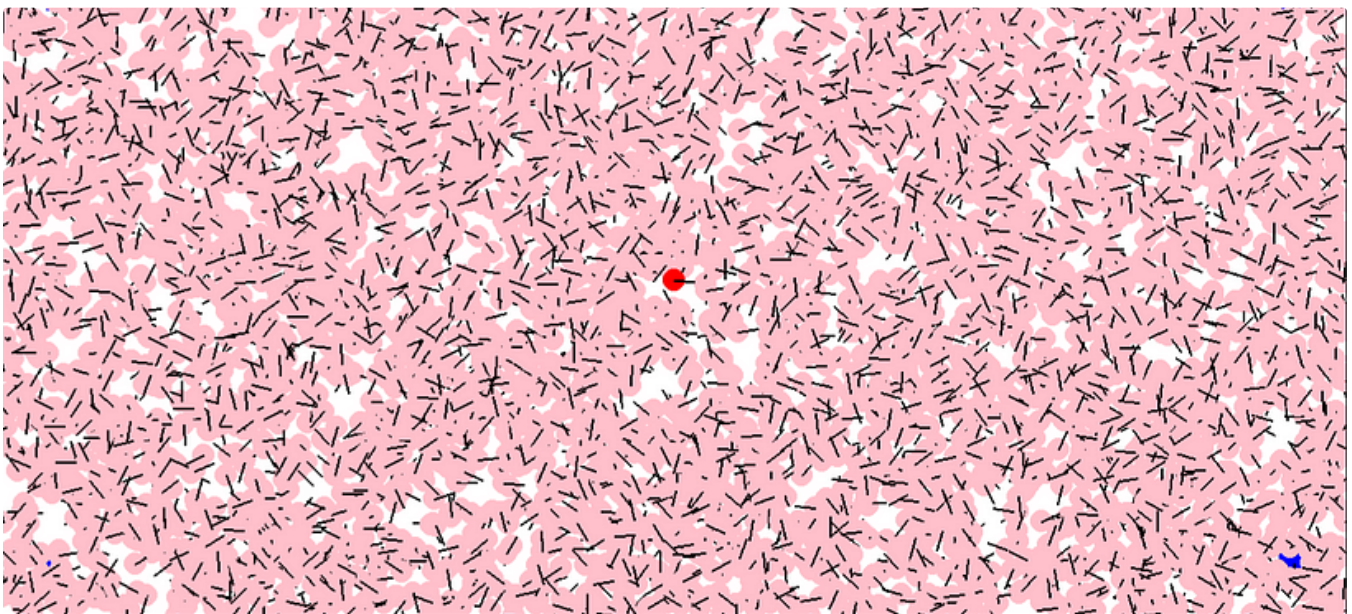
Kalman Filter Part 2 — Bayes Filter

This article has been written in collaboration with Sharad Maheshwari. [Go to his YouTube channel](#) to get more about this series.

13 min read · Jun 26, 2023



5



Caption



Mathias Mantelli

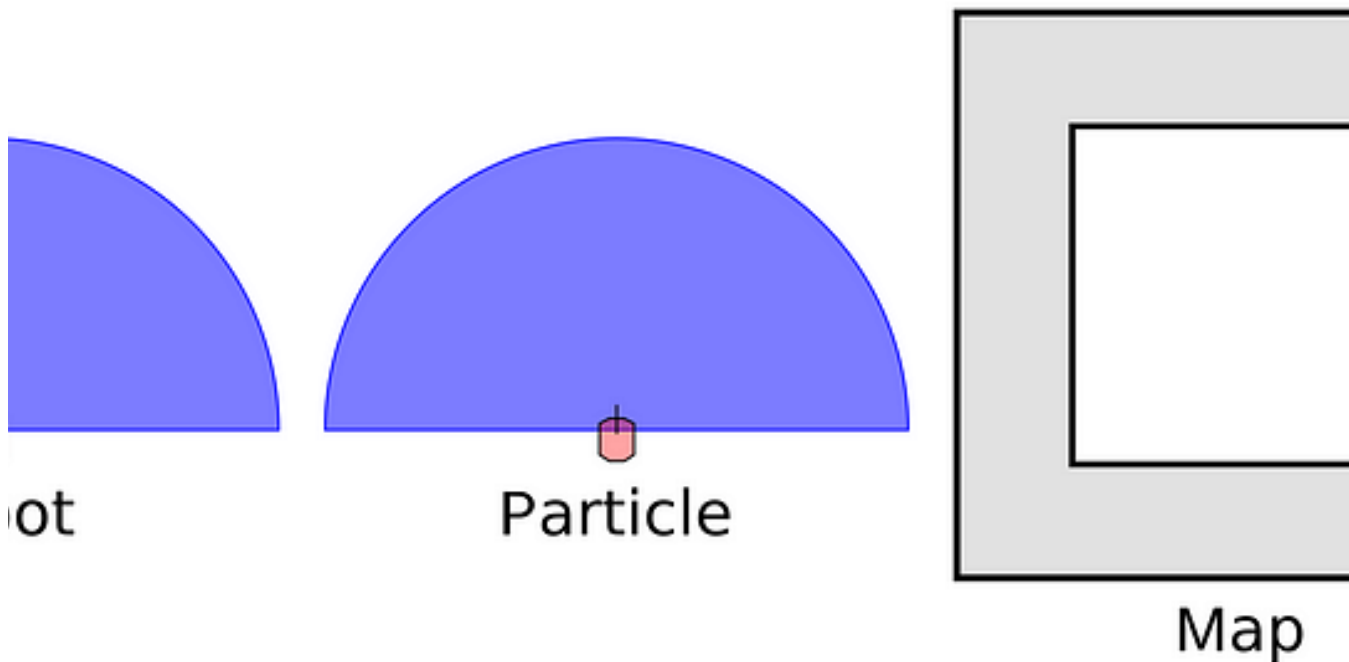
Particle Filter Part 4—Pseudocode (and Python code)

This is the fourth part of our Particle Filter (PF) series, where I will go through the algorithm of the PF based on the example presented...

6 min read · Mar 7, 2024



44



Mathias Mantelli

Particle Filter Part 2—Intuitive example and equations

This article has been written in collaboration with Sharad Maheshwari. Go to his YouTube channel to learn more about robotics.

9 min read · Oct 31, 2023



86

[See all from Mathias Mantelli](#)

Recommended from Medium



Jake Hession

Liquid Neural Nets (LNNs)

A deep dive into Liquid Neural Networks, one of the most exciting recent developments in time series forecasting

7 min read · Feb 6, 2024

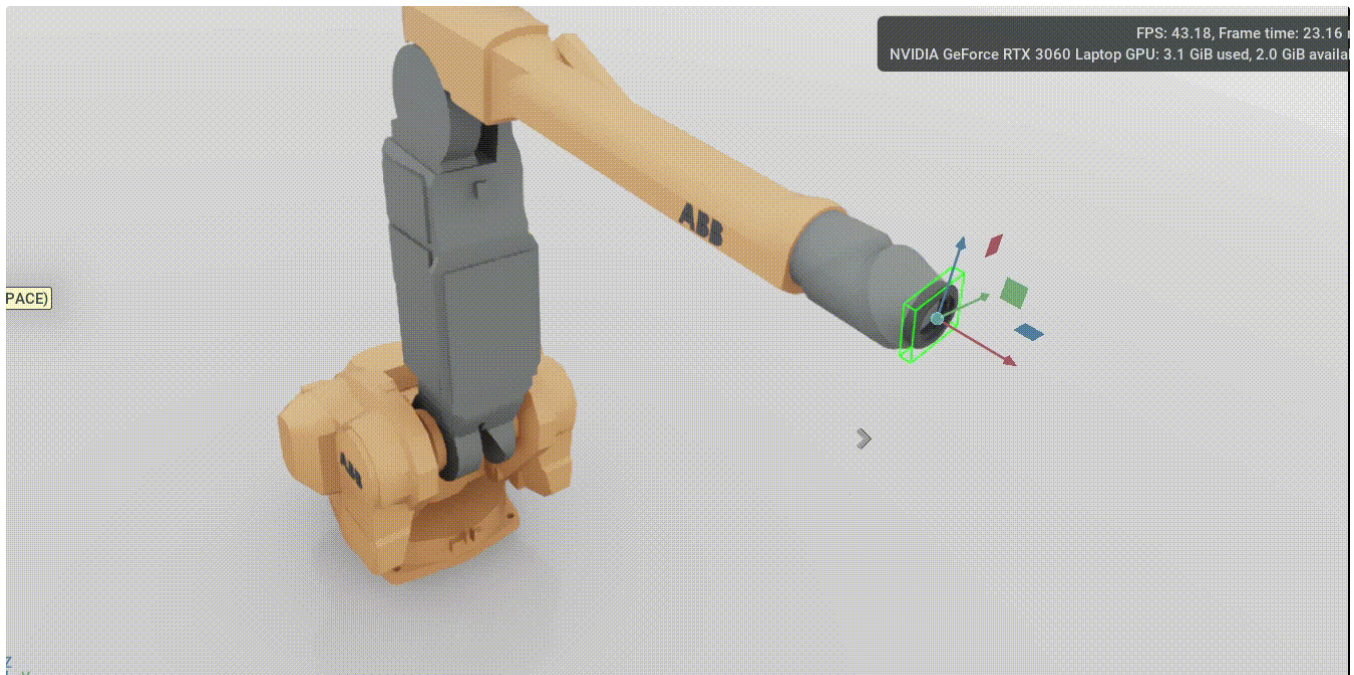


482



3





Ake Zavala JM

A Practical Approach to the NVIDIA Omniverse Robotics Simulation Toolkit, Part 1:

Creating a robotic arm via the Python OpenUSD API

8 min read · Nov 29, 2023



50



1

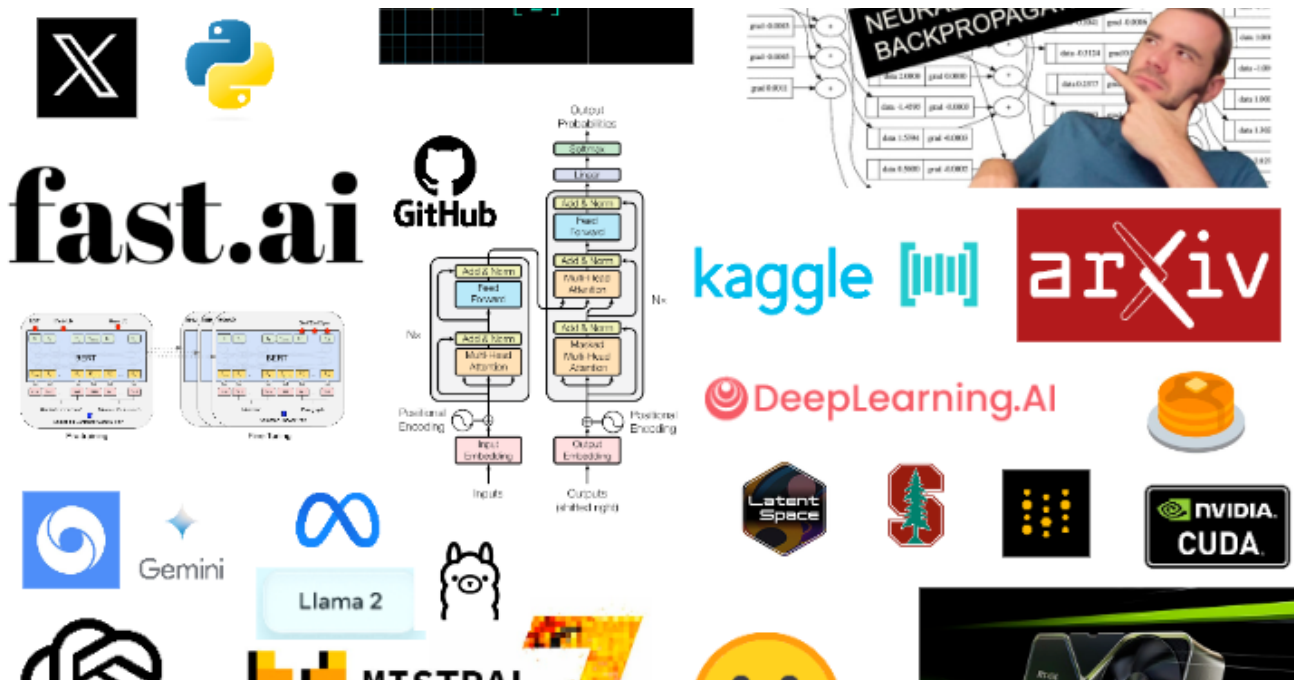


Lists



Staff Picks

624 stories · 905 saves



 Benedict Neo in bitgrit Data Science Publication

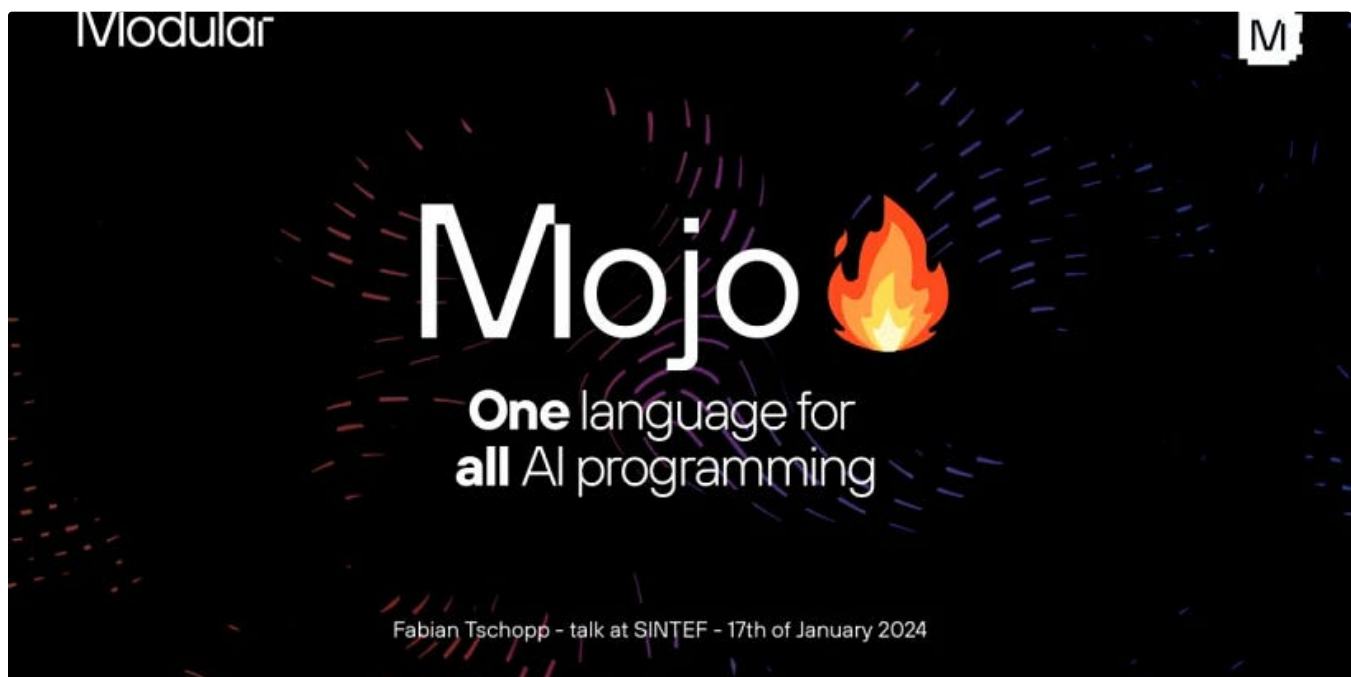
Roadmap to Learn AI in 2024

A free curriculum for hackers and programmers to learn AI

11 min read · Mar 11, 2024

 10.1K  110



 Dylan Cooper in Stackademic

Mojo, 90,000 Times Faster Than Python, Finally Open Sourced!

On March 29, 2024, Modular Inc. announced the open sourcing of the core components of Mojo.

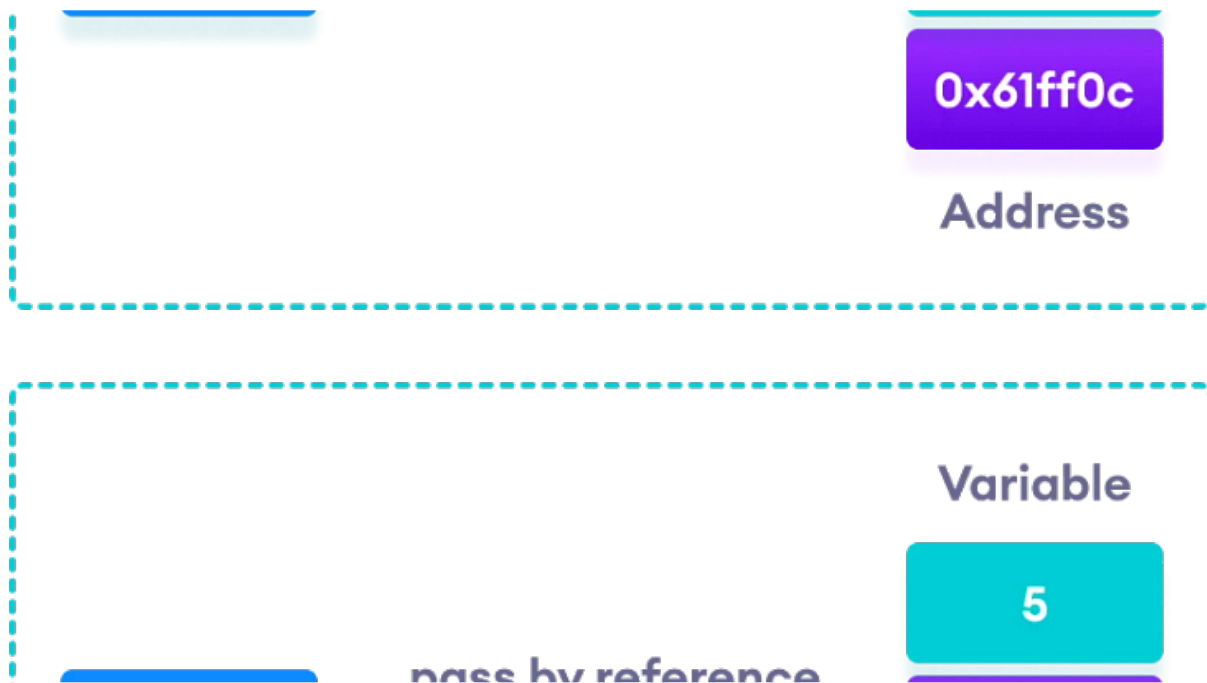
🌟 · 10 min read · Apr 8, 2024



2.9K



21



 Nitish Singh

Passing By Pointer vs Passing By Reference in C++

Let's take a closer look at the `smallest_element` function and analyze how the arguments are being passed to it.

7 min read · Nov 26, 2023

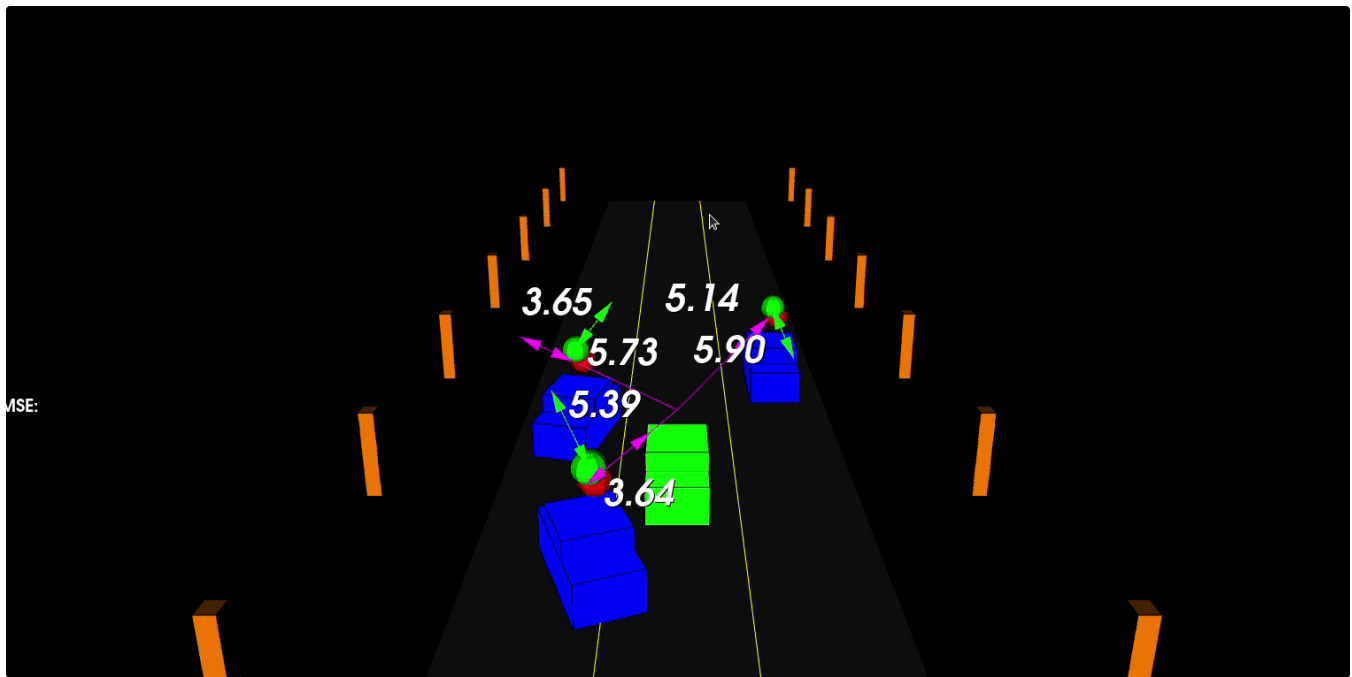


179



1





Nikhil Nair

LiDAR and Radar Sensor Fusion using Unscented Kalman Filter

Sensor fusion is the process of combining data from multiple sensors to obtain a more accurate and reliable estimate of the state of a...

13 min read · Jan 21, 2024



172



2



[See more recommendations](#)