

Open in app ↗

Sign up

Sign in

Medium

Search



# Forward Kinematics using Orocos KDL

5 min read · Mar 10, 2017



Survy Vaish

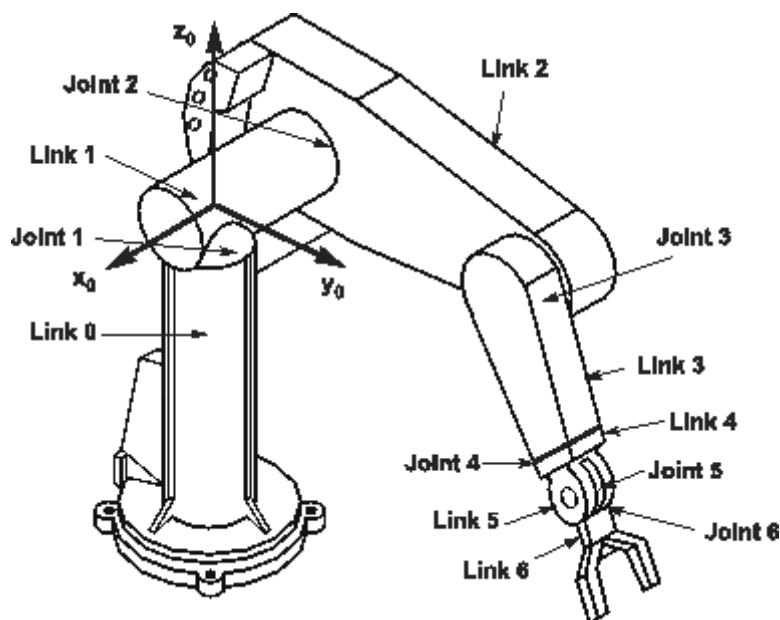
Follow



Listen



Share

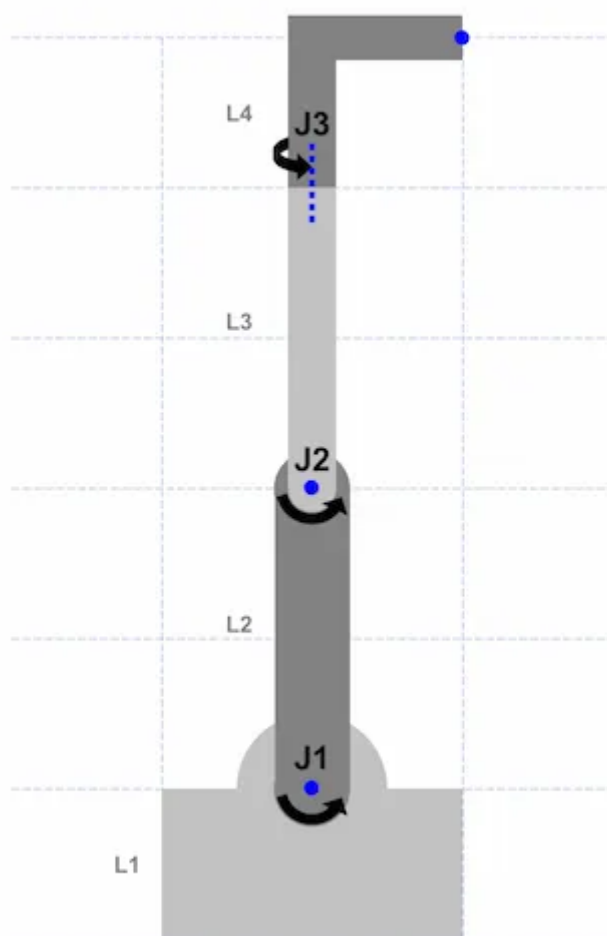


This post briefly describes how to define coordinate systems for robots and then explains how to use [Orocos KDL](#) to perform [forward kinematics](#).

## A Simple Robot

To make things a little easier, we'll be using the simple robot model shown below. It has four links (L1 — L4) and three degrees of freedom. Link 1 is fixed to the ground while Link 2 and Link 3 can rotate in the screen plane. Link 4 can rotate around like a periscope, about the axis shown in the schematic.

Our goal is to find the pose of the last link, the end effector, given the measurement of three joint angles (J1, J2, J3).

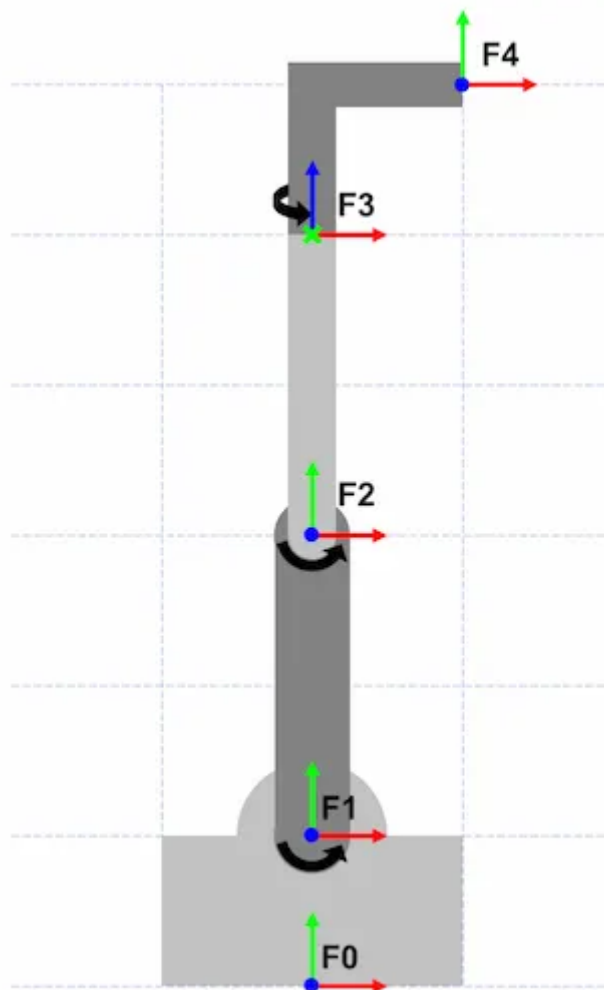


Links and Joints

## Defining Coordinate Frames

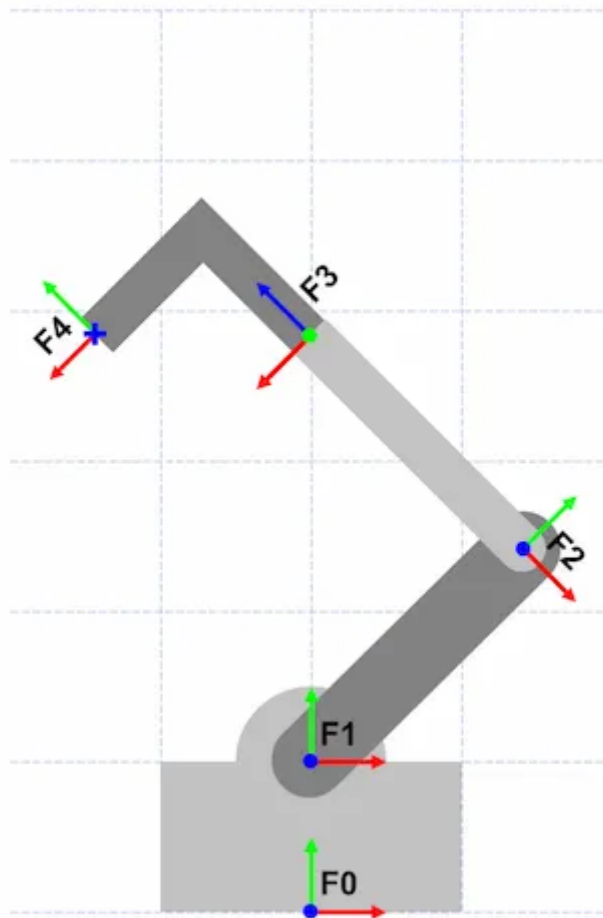
Let's start with defining the coordinate frames for each link.  $F_0$  is our reference coordinate frame, the frame relative to which we would like to find our end effector's pose,  $F_4$ .

I am using the widely used Denavit-Hartenberg convention to define the intermediate links. Specifically, it says that the Z axis should be in the direction of the joint axis.



Coordinate frames attached to links

- $F1$  is attached to Link 1 with the Z axis aligned to Joint 1, coming out of the plane.
- $F2$  is attached to Link 2 with the Z axis aligned to Joint 2, coming out of the plane.
- $F3$  is attached to Link 3 with the Z axis aligned to Joint 3, along the axis of rotation for Link 4.



Robot pose for joint angles  $-45^\circ$ ,  $90^\circ$ ,  $180^\circ$

Here's another image to illustrate how the coordinate frames are attached to the links and joints.

- Joint 1 is rotated  $-45^\circ$
- Joint 2 is rotated  $90^\circ$
- Joint 3 is rotated  $180^\circ$

Assuming that the origin is at F0, the end effector's position, F4, is roughly at  $(-1.5, 3.8, 0)$ . We'll calculate this as well the the full pose using forward kinematics in the next section.

## Forward Kinematics

### What is Orocos KDL?

Orocos (Open Robot Control Software) is a suite of libraries for robot arm control. Orocos KDL (Kinematics and Dynamics Library) provides the ability to create *kinematic chains* to perform forward and inverse kinematics.

### Step 1: Creating Segments

In KDL, kinematic chains consist of *segments*. Segments are essentially the links of the robot. A segment is a combination of a *Joint* and a *Frame*. The joint tells the segment how a frame moves as the joint angle changes. For example, it specifies if the frame rotates about the Z axis.

Frames are specified relative to the preceding frame:

- *F1* is defined relative to *F0*
- *F2* is defined relative to *F1*
- *F3* is defined relative to *F2*
- *F4* is defined relative to *F3*

Let's create an empty chain to which we'll add the segments.

```
Chain kdlChain = Chain();
```

For the first segment, note how *F1* is oriented the same way as *F0*, just located at (0, 1, 0). Hence, we can initialize our frame using a simple vector. Also note that *F1* doesn't move relative to *F0*. In this case we'll use a *Joint::None* while creating the segment.

```
Joint joint1(Joint::None);  
Frame frame1 = Frame(Vector(0.0, 1.0, 0.0));  
kdlChain.addSegment(Segment(joint1, frame1));
```

Similarly, for *F2* there is again no change in orientation of the frame, just a translation to (0, 2, 0). Link 2 however rotates about the Z axis of the previous frame, so we'll use *Joint::RotZ*.

```
Joint joint2(Joint::RotZ);  
Frame frame2 = Frame(Vector(0.0, 2.0, 0.0));  
kdlChain.addSegment(Segment(joint2, frame2));
```

*F3* is rotated about the X axis by  $-90^\circ$  and then translated 2 units about the new Z axis. We can multiply two intermediate frames to construct the final one.

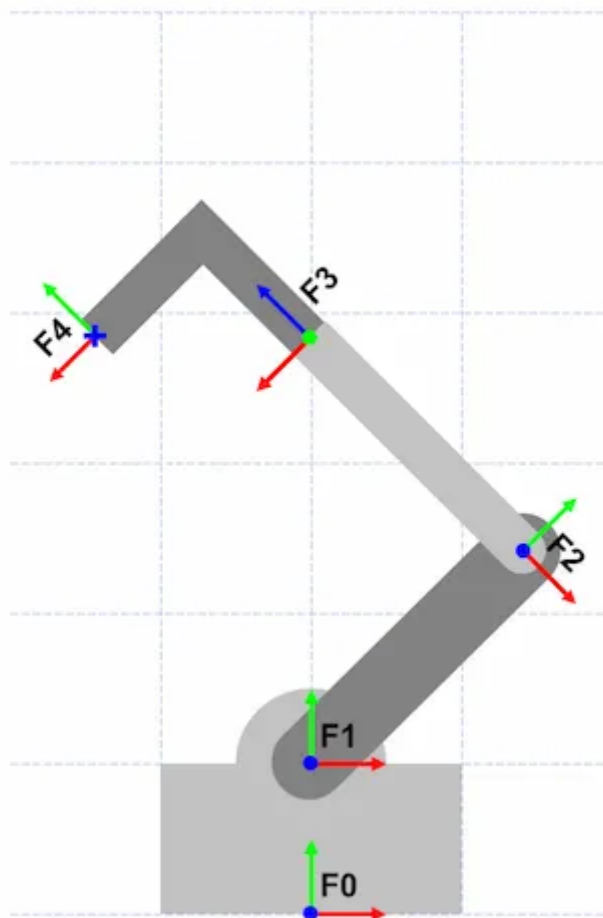
```
Joint joint3(Joint::RotZ);  
Frame frame3 = Frame(Rotation::EulerZYX(0.0, 0.0, -M_PI / 2)) *  
                 Frame(Vector(0.0, 0.0, 2.0));  
kdlChain.addSegment(Segment(joint3, frame3));
```

*F4* first reverses the previous rotation and is then translated 1 unit in the X and Y directions.

```
Joint joint4(Joint::RotZ);  
Frame frame4 = Frame(Rotation::EulerZYX(0.0, 0.0, M_PI / 2)) *  
                 Frame(Vector(1.0, 1.0, 0.0));  
kdlChain.addSegment(Segment(joint4, frame4));
```

## Step 2: Joint Angles

Next we'll construct a joint angles variable that will contain the three angles for which we wish to perform forward kinematics.



Robot pose for joint angles  $-45^\circ$ ,  $90^\circ$ ,  $180^\circ$

```
JntArray jointAngles = JntArray(3);
jointAngles(0) = -M_PI / 4.;           // Joint 1
jointAngles(1) = M_PI / 2.;           // Joint 2
jointAngles(2) = M_PI;                 // Joint 3
```

### Step 3: Forward Kinematics

Finally, we can run forward kinematics for the joint angles in step 2, using the chain we constructed in step 1. This will give is the end effector pose.

```
ChainFkSolverPos_recursive FKSolver =
    ChainFkSolverPos_recursive(kdlChain);
Frame eeFrame;
FKSolver.JntToCart(jointAngles, eeFrame);
```

Forward kinematics solution (contents of *eeFrame*):

-0.7071	-0.7071	0.	-1.414
-0.7071	0.7071	0.	3.828
0.	0.	-1.	0.
0.	0.	0.	1.

Let's verify a couple of things against the diagram.

- the end effector's position is roughly at (-1.5, 3.8, 0); the calculated position is (-1.414, 3.828, 0.)
- the end effector frame's Z axis unit vector is pointing along *F0*'s -Z Axis
- the X axis unit vector is (-0.7071, -0.7071, 0.) which checks out too since in the image it's in the screen plane and pointing down and to the left — negative x, negative y relative to *F0*
- similarly, the Y axis unit vector is (-0.7071, 0.7071, 0.) which means it should be in the screen plane and pointing up and to the left — negative x, positive y relative to *F0*

## The code

Code is available on [Github](#).

Here's the relevant code all at once:



## Acknowledgements

[Will McMahan](#) for technical proofreading.

Robotics

Forward Kinematics

Tutorial



Follow

## Written by Survy Vaish

43 followers · 19 following

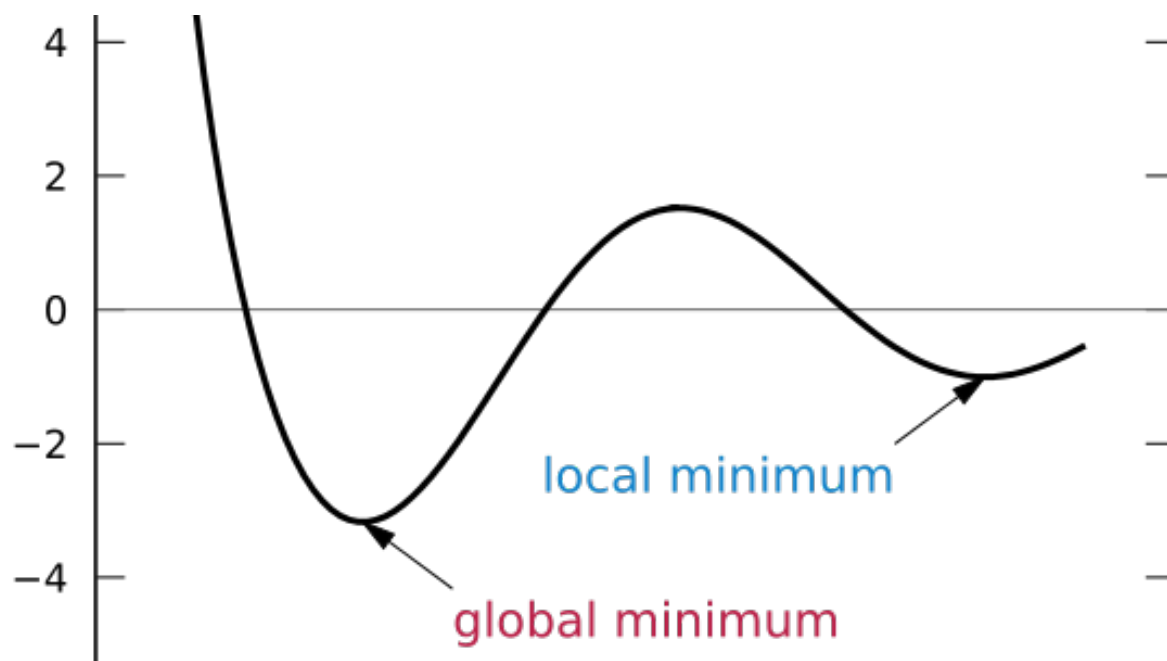
### No responses yet



Write a response

What are your thoughts?

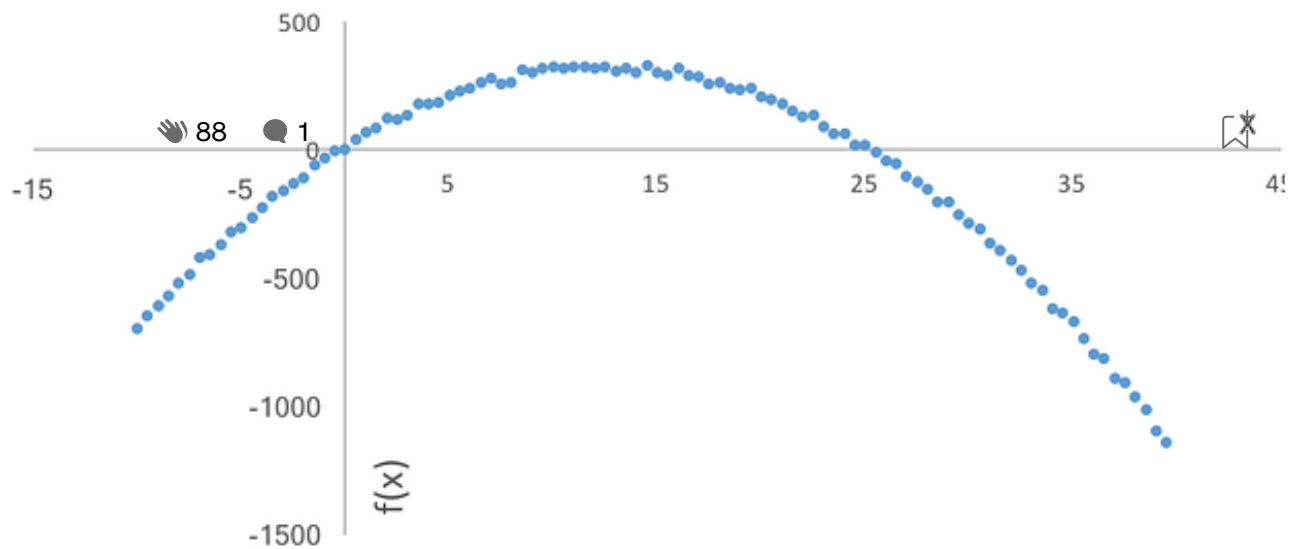
### More from Survy Vaish





Survy Vaish

## Levenberg-Marquardt Optimization (Part 1)



Survy Vaish

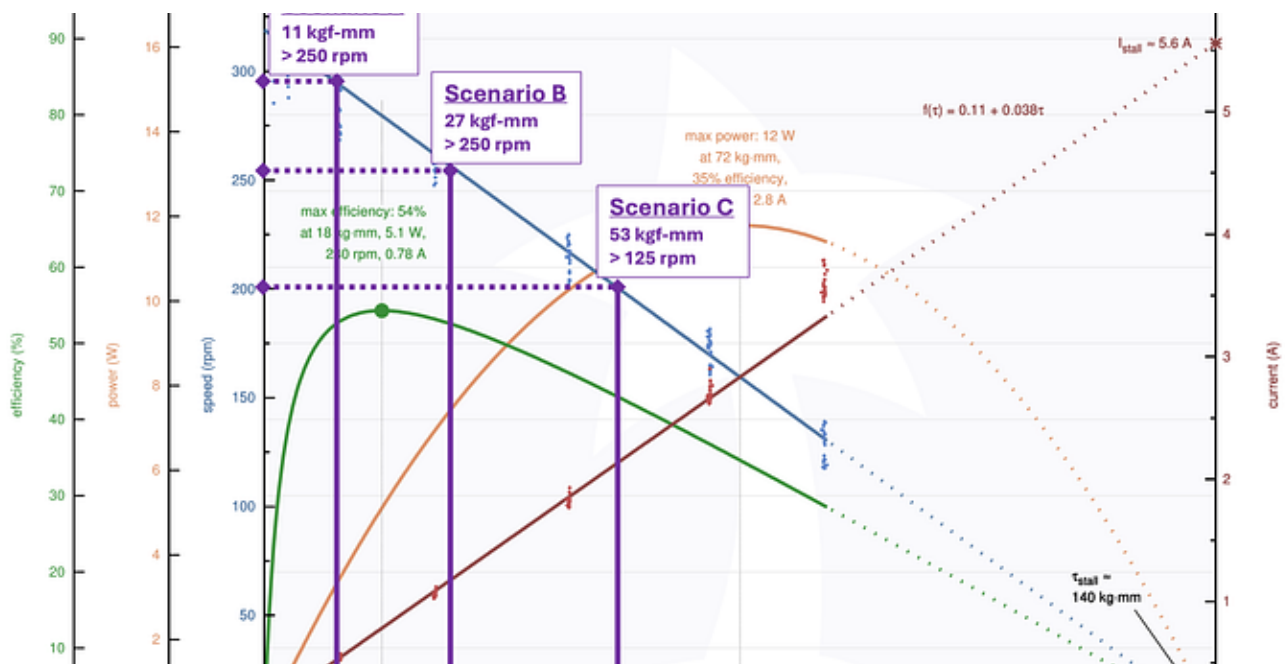
## Levenberg-Marquardt Optimization (Part 2)


A tutorial on how to use Eigen's Levenberg-Marquardt optimization API for non-linear least squares minimization.

Mar 13, 2017

👍 134


💬 2



 Survy Vaish

## Robomagellan | How to size and select motors for an outdoor robot

This post, part of a larger series on building an outdoor robot to compete in the Robomagellan competition, describes how I selected motors...

Jul 8, 2020  4 Survy Vaish

## Shelter in Place Arts and Crafts

Being stuck at home due to the coronavirus has given my family and I an opportunity to pursue tons of different arts and crafts projects...

Apr 1, 2020  1  1[See all from Survy Vaish](#)

## Recommended from Medium

Original Image



VAE Re-construct



Efrat taig

## VAE. The Latent Bottleneck: Why Image Generation Processes Lose Fine Details

Why do AI-generated images lose critical details? If you're frustrated by vanishing textures, blurry text, or delicate features that simply...

Mar 31 9



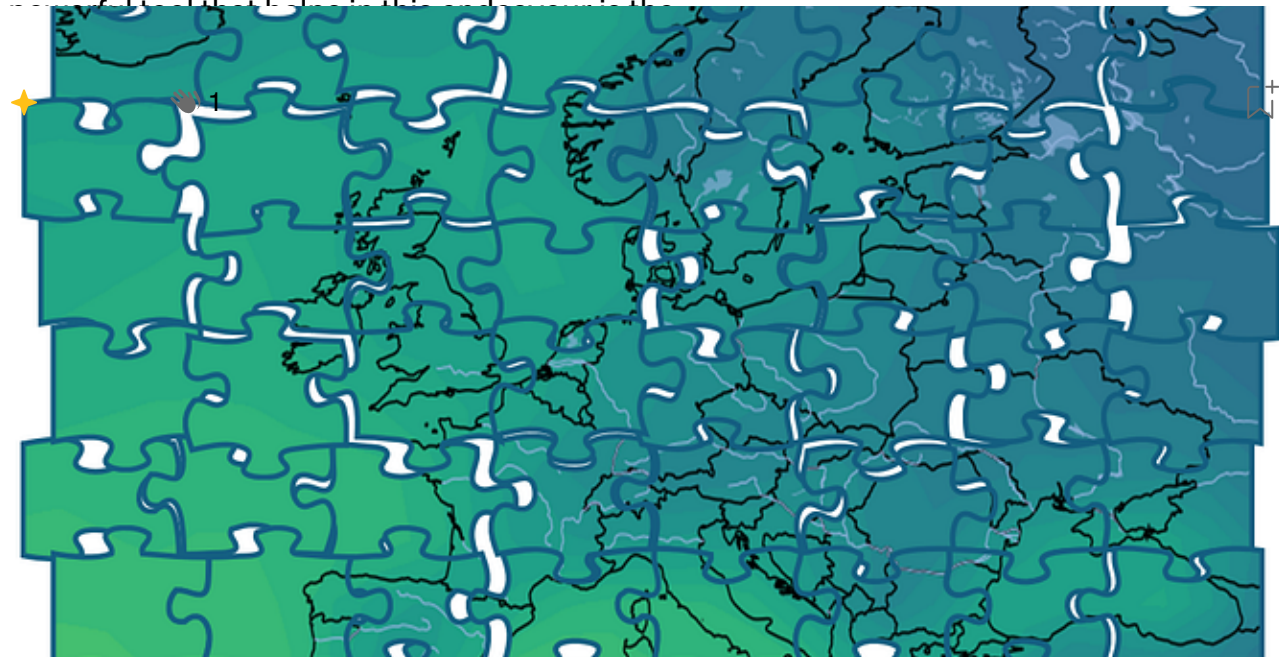




Aakash Chavan Ravindranath, Ph.D

## Hidden Markov Model and the Medilian Fund: A Deep Dive into Market Prediction

Introduction In finance, making informed predictions is key to successful trading. One




Danu Caus

## GPyTorch Gaussian Process for Multidimensional Continuous Signal Reconstruction and Image...

Building models that know what they don't know.

May 4 🖱 21



 In The Pythonworld by Pythonworld

## Why I Stopped Using Classes in Python (And What I Use Instead)

After years of writing class-based Python code, I realized simpler tools often get the job done better. Here's what I switched to—and...

★ Jul 23 🖱 1.5K 💬 39



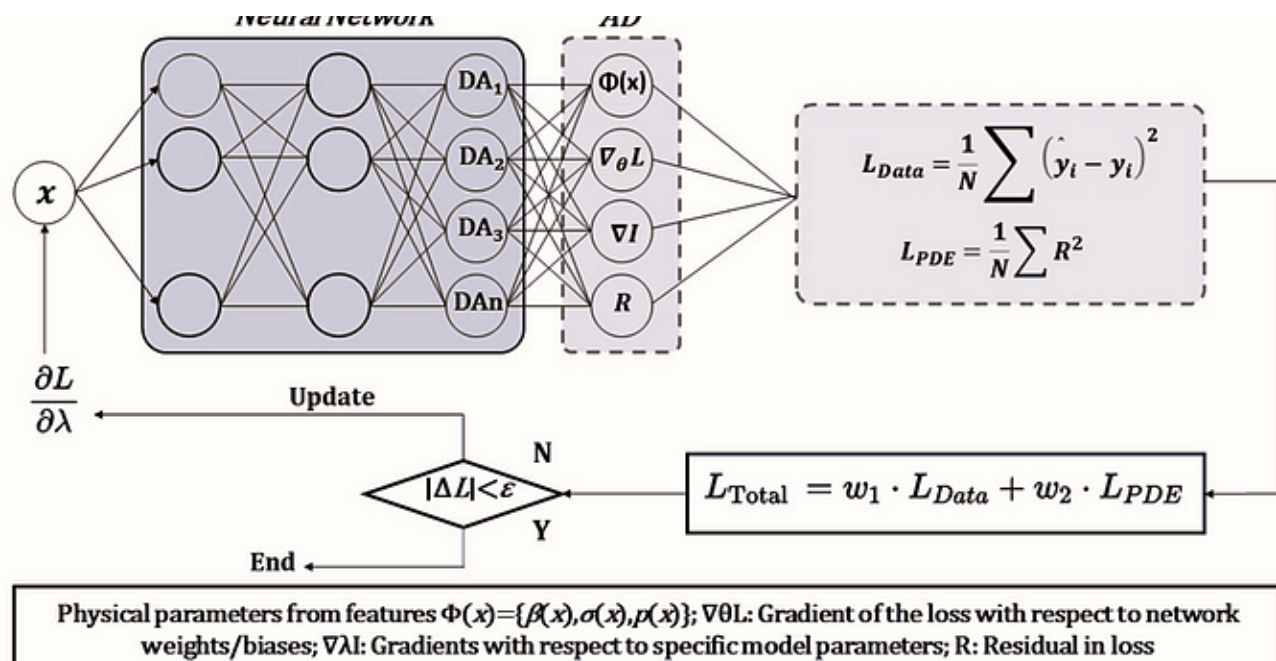
 Yash Batra

## The 47-Line Code That Made One Developer \$2 Million from AI

In late 2024, a solo indie developer pushed 47 lines of Python code to GitHub.

★ Jul 9 🖱 2.5K 💬 81





Rania Labib

## Solving the Diffusion Equation with Physics Informed Neural Networks (PINNs) for Daylighting Design...

Introduction



Feb 26



15



See more recommendations