Sarvesh Patil

Mechanical Engineer

ABOUT ME

I'm a mechanical engineer who likes figuring things out by building them. Most of my projects start as quick CAD sketches, cardboard mockups or breadboard experiments and eventually turn into something that actually works. I enjoy mixing design, mechanics, electronics and code to bring ideas to life. I focus on understanding the details of what I build and enjoy taking an idea from a rough concept to a working prototype, refining and documenting everything clearly along the way.

sarveshpatil73.sp@gmail.com

Houston, TX

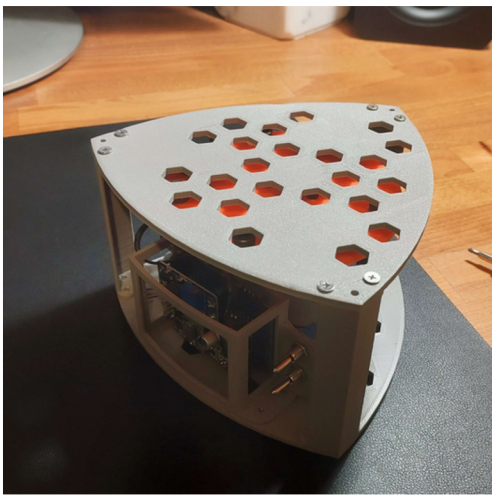
+1 (217) 904-0560

https://github.com/SarveshPatil7

http://www.linkedin.com/in/sarveshpatil07

Self Balancing Prism

A mobile reaction wheel based inverted pendulum setup that balances on an edge



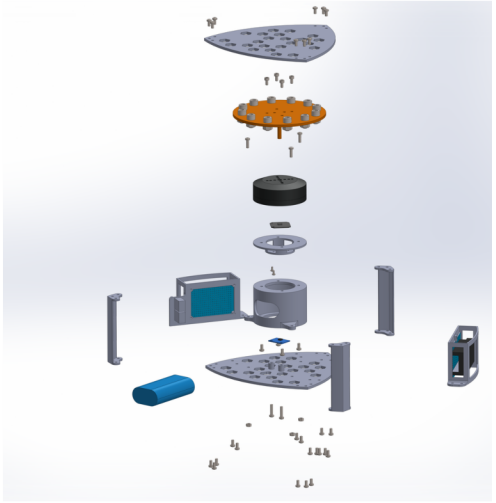
Completed prototype

OVERVIEW

I saw some inverted pendulum projects and was further inspired by ‘shapes of constant width’ and thought it would be fun to build my own version that balances on an edge. This was a personal project with no set goal other than learning. I wanted to design the hardware, write the firmware, and get the whole system working by myself

WHAT I DID

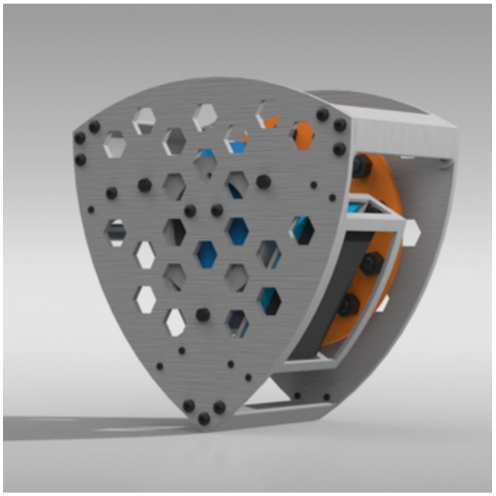
- Designed the assembly in SolidWorks and 3D-printed it with PLA
- Selected and integrated all components (ESP32-S3, GBM5208 BLDC motor, DRV8313 motor driver, MT6701 encoder, MPU6050 IMU)
- Wrote the full firmware : angle estimation, task scheduling, PID + state observer, and FOC motor control
- Performed experimental tuning, validation, drift suppression, and stability testing



Exploded assembly

HOW IT WORKS

- Dual-core ESP32-S3 running FreeRTOS
- One core handles the IMU sampling
- The other runs the angle estimation, PID and FOC loop
- Angle estimation merges gyro + accelerometer data using complimentary filters and drift correction for stable orientation tracking
- The GBM5208 reaction wheel provides the counter-torque needed to keep the prism upright
- The PID controller is complimented by a state observer for the motor to avoid saturation of the motor speed



Prototype rendering in Solidworks Visualize

Thermal FEA Solver

OVERVIEW

This was one of the projects I made for the class ME 471 - Finite Element Analysis. I was given a basic Python framework to parse Abaqus .inp files and plot results, but the entire solver had to be written from scratch.
I used the assignment as a chance to understand the full workflow behind 2D thermal FEA and to implement the complete finite element formulation myself.

WHAT I DID

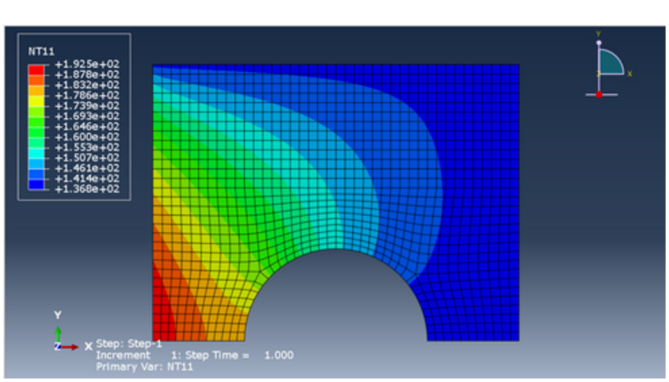
- Wrote the entire solver logic including stiffness matrix formation, element routines, assembly, and boundary condition application
- Implemented conduction, convection, and flux contributions at the element level using the Q4 formulation
- Extended the steady state solver to transient solver using adjustable time stepping schemes
- Developed the time integration workflow including lumped capacitance matrix formation, critical time step calculation, and both Forward Euler and Crank Nicolson updates

HOW IT WORKS

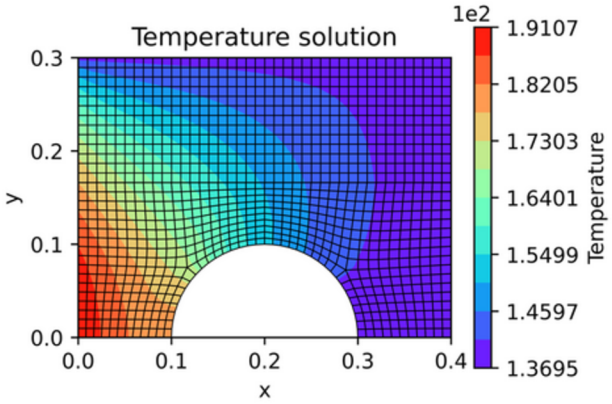
- The provided input reader builds the mesh and boundary data
- My routines compute the element matrices for each Q4 element and assemble them into the global system
- Boundary conditions are applied through system partitioning into free and prescribed nodes
- For transient analysis, the system uses the heat capacity matrix and integrates the temperature field forward in time
- Results are returned to the instructor's visualization script to generate contour plots

RESULTS

- My solver matched Abaqus temperature fields within well under one percent on all test meshes
- Both steady state and transient solutions followed the expected trends across different mesh sizes
- Time history plots from my code and Abaqus were almost identical, which confirmed that the element formulations and boundary conditions were implemented correctly



A. Results from Abaqus



B. Results from the python solver

Comparison of results from Abaqus vs custom coded solver results

Immersion Cooling Electric Vehicle Battery

OVERVIEW

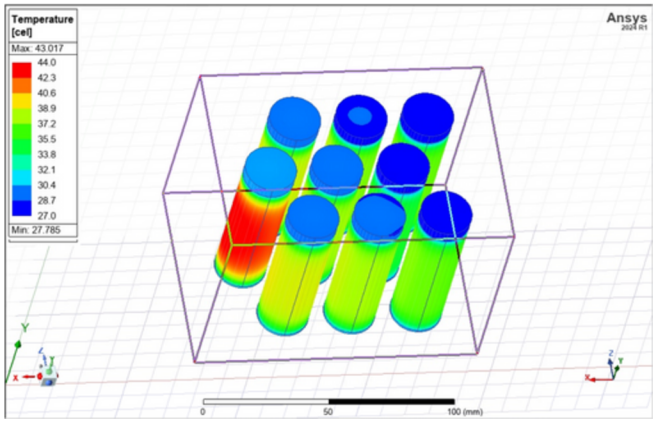
This project was part of my electronics cooling class. Our team studied the thermal limits of the coldplate cooled battery pack used in the 'Lucid Air'. The 21700 cells in the pack struggle during high discharge because the coldplates cannot keep the cell temperatures uniform, which leads to hotspots and a higher risk of thermal runaway. We explored immersion cooling as an alternative and evaluated how much it could improve temperature levels and uniformity.

WHAT I DID

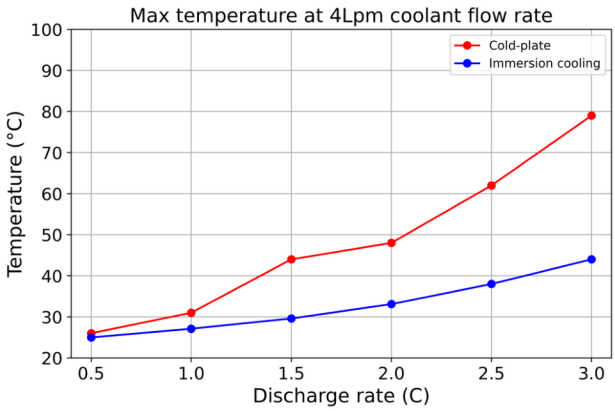
- Recreated the Lucid cell arrangement by reverse engineering the spacing from reference photos using image analysis
- Built the full CAD for the module and the 21700 cells including the internal jellyroll structure
- Modeled both the coldplate design and the immersion cooled version
- Verified coolant properties and gathered the full material set for mineral oil
- Ran a portion of the simulations for 1C to 3C discharge conditions
- Wrote the SOP for the partial immersion setup and helped with the final report and plots

HOW IT WORKS

- Each cell generates heat based on its discharge rate and internal resistance
- In the coldplate design, heat moves into an aluminum plate that carries coolant in a channel, effective mostly in regions in directly in contact with the coldplate
- In the immersion design, mineral oil removes the heat from the entire surface of the cell for a more uniform effect
- The simulations track maximum temperature, temperature spread, and how flow rate affects both
- The geometry and heat models were built by our team and simulated in Icepak



A. Ansys Icepak simulation



B. Comparing the performances

High-Dimensional Reliability Analysis

Dimension Reduction using Deep Learning for High-Dimensional Reliability Analysis

OVERVIEW

In my Data Driven Design Methods coursework, we studied the classic methods like Monte Carlo, FORM, SORM and Kriging. These work well for simple problems, but real design work often involves dozens of uncertain parameters, and the cost of analysis goes up rapidly with the number of variables. This made me curious about how it can be made practical for design decisions. I found a paper by Li and Wang that used an autoencoder, a feedforward network and a Gaussian Process surrogate to reduce the dimensionality of a limit state function. This makes it possible to run reliability calculations in a much smaller space. I decided to implement the full workflow myself in MATLAB to understand how the method works and how accurate it can be.

WHAT I DID

- Recreated the complete dimensional reduction and surrogate modeling method described in the paper
- Built separate modules for data generation, autoencoder training, DFN training, Gaussian Process modeling, and model verification
- Ran 16 case studies on the Griewank function for 10 to 200 dimensions
- Evaluated accuracy, computational cost and convergence across the sample set size of 10 to 4000

HOW IT WORKS

- The autoencoder is trained to represent the data to and from the reduced dimensional space, aka latent space
- A feedforward network is trained to predict this latent representation directly from the input variables
- A Gaussian Process surrogate is built in the latent space to approximate the outputs
- Finally, the feedforward network and surrogate model are used together, to estimate the outputs for a given testing inputs
- The estimated outputs are then compared against the calculated outputs to verify the accuracy

RESULTS

- The predicted data was within 8% of the calculated results for a 10 dimensional problem
- Acheived the computation of a 200 dimensional problem within 0.03 seconds