The Journal of Open Source Software

strapdown-rs: A Simple Strapdown INS Implementation in Rust

James Brodovsky [1*¶]

1 Temple University, United States ¶ Corresponding author * These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 05 June 2025
Published: unpublished

Summary

Inertial navigation systems (INS) are critical for many applications in robotics, aerospace, and autonomous systems. They provide real-time estimates of position, velocity, and orientation using data from an inertial measurement unit (IMU) as well as other aiding sensors like GPS. Strapdown implmentations of INS are becoming increasingly common due to the proliferation of micro electro-mechanical system (MEMS) IMUs. These MEMS IMUs are popular for their low size, weight, and power (low SWaP) characteristics, making them suitable for a wide range of applications including drones, robotics, and mobile devices.

`strapdown-rs` is a Rust-based software library for implementing strapdown inertial navigation systems (INS). It provides core functionality for processing inertial measurement unit (IMU) data to estimate position, velocity, and orientation using a strapdown mechanization model that is typical of modern systems particularly in the low size, weight, and power (low SWaP) domain (cell phones, drones, robotics, UAVs, UUVs, etc.). Additionally, it provides some basic simulation capabilities for simulating INS scenarios (e.g. dead reckoning, closed-loop INS, intermitent GPS, GPS degradation, etc.).

`strapdown-rs` prioritizes correctness, numerical stability, and performance. It is built with extensibility in mind, allowing researchers and engineers to implement additional filtering, sensor fusion, or aiding algorithms on top of the base INS framework. This library is not intended to be a full-featured INS solution, notably it does not have code for processing raw IMU or GPS signals and only implements a loosely-couple INS.

The toolbox is designed for research, teaching, and development purposes and aims to serve the broader robotics, aerospace, and autonomous systems communities. The intent is to provide a high-performance, memory-safe, and cross-platform implementation of strapdown INS algorithms that can be easily integrated into existing systems. The simulation is intended to be used for testing and verifying the correctness of the INS algorithms, by providing a simple simulation that allows users to generate a "ground truth" trajectory.

Statement of Need

Strapdown INS implementations are commonly written in MATLAB, Python, or C++, and are typically *proprietary* or are heavily integrated into an existing architecture or framework. This project provides a high-performance, memory-safe, and cross-platform implementation in Rust — a modern systems language well-suited for embedded and real-time applications. Why Rust and why another INS library? Several reasons that are pertinent critiques of each language:

MATLAB

Many proprietary research INS implementation exist in the maritime, aerospace, and defense engineering sectors which have robust experience with developing and researching INS algorithms. While MATLAB is great for prototyping, it is not suitable for production systems due to performance and deployment issues, nor is it a systems programming language. This makes for the common refrain of "prototype in MATLAB, implement in C/C++". This generates additional workload and complexity for industry researchers and engineers as it introduces additional complexity and potential for bugs via the translation process. MATLAB is also antithetical to open science being a proprietary language with a closed-source ecosystem.

C/C++

While different, C and C++ are often used interchangeably in the context of INS implementations. C is a low-level language that provides direct access to hardware and memory, making it suitable for performance-critical applications. However, C lacks modern features such as memory safety and high-level abstractions, which can lead to complex and error-prone code. C++ offers some of these features but is often criticized for its complexity and steep learning curve. Both languages also have a steep learning curve for those who are not familiar with systems programming. Simply put, these languages do not have the same level of safety and ease of use as higher-level languages like Python or MATLAB. This can make it difficult for researchers and engineers to implement and maintain complex algorithms, especially in the context where performance is still needed. Furthermore, these languages lack modern tooling making managing your dependencies, building, and testing more difficult.

Python

Python is a great language for rapid prototyping and development, but it is not suitable for performance-critical applications. It also has issues with memory management and real-time constraints due to it's garbage collected nature. While Python is widely used and has many high-performance libraries (namely NumPy) for numerical computing, some applications simply cannot be vectorized appropriately to take advantage of the underlying C libraries or through additional tools like Numba. When running simulations, there is sometimes no avoiding a loop, something Python is notoriously slow at executing.

Specifically, one algorithm that is frequently used in navigation is a Particle Filter (PF). Particle filters are often used for state estimation in non-linear systems, and they require a large number of particles to be effective, particularly when used in systems with large state vectors. This introduces the primary problem that motivated the development of `strapdown-rs`. It makes sense to re-use the same code for typical local-level frame forward mechanization. This is a standard set of equations that can be used in multiple different INS architectures. For a Kalman Filter based INS, this is relatively simple and you can typically use whatever language's linear algebra library you prefer to store the data. You can also do the same for the particle filter, and have list of vectors that represent the particles. However, this forces you into the trap of Python: iterating through the the list.

Alternatively, you could vectorize the operations, but this introduces additional complexity and requires you to test and verify that the vectorized operations match the original forward mechanization equations. This makes it difficult to swap out the filtering algorithm or the forward mechanization equations without rewriting large portions of the code.

Thus what is needed is a modern, compiled, systems programming language with a useful linear algebra library.

Rust

Rust is a modern systems programming language that combines the performance of C/C++ with the safety and concurrency features of higher-level garbage-collected languages. It is designed for performance-critical applications and has a strong focus on memory safety, making it an ideal choice for implementing strapdown INS algorithms. From a scientific development perspective, Rust puts guardrails on scientist-developers who's primary skill set isn't in writing production-grade memory safe code. By following basic good practices in Rust, you get the benefits of modern tooling and language syntax that you get with Python, Java, and Go with the performance of C or C++, with the additional guarantee of if it compiles the only bugs are *logic* bugs.

Open Source

The `strapdown-rs` library is open source, which means that it is freely available for anyone to use, modify, and distribute. This is important for scientific research and development, as it allows researchers to share their work and collaborate with others in the field. Many such comprehensive INS implementations are often developed in-house, are proprietary, and closed source. Open source software also promotes transparency and reproducibility, which are essential for scientific research. By releasing `strapdown-rs` as an open-source library, it provides a reusable foundation for anyone building INS pipelines, sensor fusion stacks, or GNSS-denied navigation systems — particularly for research involving robotics, aerospace vehicles, or embedded autonomy platforms.

Overview of Functionality

`strapdown-rs` contains three primary library modules: `strapdown`, `strapdown::earth` and `strapdown::filter`. It also has a reference implementation of a loosely-coupled INS in the `strapdown::sim` module that can be accessed via the installable binary.

Library Modules

The `earth` module contains constants and functions related to the Earth's shape and other geophysical features (gravity and magnetic field). The Earth is modeled as an ellipsoid with a semi-major axis and a semi-minor axis(National Geospatial-Intelligence Agency, 2014). The Earth's gravity is modeled as a function of the latitude and altitude using the Somigliana method. The Earth's magnetic field is modeled using a diapole model (NOAA NCEI Geomagnetic Modeling Team and British Geological Survey, 2024). This module relies on the nav-types crate (Nordmoen & Feuerstein, 2022) for the coordinate types and conversions, but provides additional functionality for calculating rotations for the strapdown navigation filters. This permits the transformation of additional quantities (velocity, acceleration, etc.) between the Earth-centered Earth-fixed (ECEF) frame and the local-level frame.

The `strapdown` module provides some helper functions as well as the forward mechanicization equations for strapdown inertial navigation systems. It provides a set of structs for modeling both IMU data and the base nine element strapdown state (latitude, longitude, and altitude; velocities north, east, and down; and attitude). It includes and implementation for the local-level frame forward mechanization, which is a common approach for strapdown INS and follows the equations from Chapter 5.4 of (Groves, n.d.).

The `filter` module contains the core functionality for implementing strapdown INS algorithms, primarily of which is a loosely-couple integration architecture according to Chapter 14.1.2 of (Groves, n.d.). This module contains implementations of various inertial navigation filters, including Kalman filters and particle filters. These filters are used to estimate the state of a strapdown inertial navigation system based on IMU measurements and other sensor data. The filters use the strapdown equations (provided by the StrapdownState) to propagate the state in the local level frame.

Executable Modules

the `sim` module provides a reference implementation of a loosely-coupled INS using the `strapdown` and `filter` modules. It implements a basic full-state inertial navigation system that uses an unscented Kalman filter (UKF) to estimate the state of the system. It also contains structs for the handling and modeling of test data and navigation solution data.

References

Groves, P. (n.d.). *Principles of GNSS, inertial, and multisensor integrated navigation systems* (2nd ed.). Artech House. ISBN: 978-1-60807-005-3

National Geospatial-Intelligence Agency. (2014). *Department of defense world geodetic system 1984: Its definition and relationships with local geodetic systems* (No. NGA.STND.0036_1.0.0_WGS84). National Geospatial-Intelligence Agency.

NOAA NCEI Geomagnetic Modeling Team and British Geological Survey. (2024). *World magnetic model 2025*. NOAA National Centers for Environmental Information. https://doi.org/10.25921/aqfd-sd83

Nordmoen, J., & Feuerstein, M. (2022). *Nav-types*. https://github.com/nordmoen/nav-types