Computation of Rotation Radii and Angles using Accelerometer Data and Optimization Algorithms

Jérôme Chabbert

John Abbott College

360-RES-AB Winter 2020

Course Coordinator: Roberta Silerova
Research Supervisor: Chris I. Larnder

21 May 2020

Abstract

The goal of the research outlined in this paper is to verify the possibility of providing the functions of a gyroscope using a lower cost alternative accelerometer, while still providing sufficient accuracy. In order to achieve this goal, a prototype library was developed capable of finding the radius and angle of a rotating accelerometer using its acceleration data without any prior conversions. Using optimization algorithms such as Adam and Stochastic Gradient Descent, or SGD, and Python, a programming language, the library was able to accurately compute both of these values separately for a simulated data set with ideal conditions. An important objective of the project was to provide a clear and extensible software framework which could be easily built upon and adapted for more specific needs. The goal was achieved through comprehensive documentation and tutorials that are available to anyone with access to our library. At the moment, our library is comprised of a multitude of modules such as Plotter.py for graphing, Load.py and SpikeTracker.py for data processing, and Simulate.py for synthetic data generation which are compatible with both TensorFlow, Google's machine learning library, and our own optimization module along with a plethora of test scripts that demonstrate its usage. Further research and development is required to enable our implementation of SGD with two variables to find the radius and angle in tandem and to confidently assess the effectiveness of our method.

1 Introduction

Embedded or IoT (Internet of Things) devices all have one goal: cost efficiency. A small device needs to be as energy and space efficient, and affordable as possible. The industry is always seeking alternatives to expensive, older technology through the creation of new data analysis software suites that use cheaper devices. In line with this train of thought, the goal of this research project is to verify the possibility of providing the functions of a gyroscope using a lower cost alternative accelerometer, while still providing sufficient accuracy. Optimization algorithms that are usually used for machine learning are implemented with the goal of finding the radius of rotation and angle between the true radial rotation and the measured value if they aren't already aligned.

These two values can give us

$$\alpha = \frac{a_t}{r}$$

and

$$\omega = \sqrt{\frac{a_r}{r}}.$$

Both of which could normally only be given by gyroscopes. The cost function that is used in the project to optimize for the radius of rotation is

$$cost_{(n)} = \dot{a}_{r(n)} - \frac{a_{t(n)}^2 \Delta t}{r} - 2a_{t(n)}\sqrt{\frac{a_{r(n)}}{r}} \tag{1}$$

where $a_{r(n)}$ is the radial acceleration at index n, $a_{t(n)}$ is the tangential acceleration at index n, Δt is the time between 2 measures, $\dot{a}_{r(n)} = \frac{a_{r(n+1)} - a_{r(n)}}{\Delta t}$ is the derivative of a_r with respect to time and r is the radius of rotation. To optimize for the angle between the true radial acceleration and the measured radial acceleration θ, the following transformation is used:

$$a_{measured(n)} \cdot \begin{bmatrix} cos(-\theta) & -sin(-\theta) & 0 \\ sin(-\theta) & cos(-\theta) & 0 \\ 0 & 0 & 0 \end{bmatrix} = a_{rotated(n)} \tag{2}$$

where

$$a_{measured(n)} = \begin{bmatrix} a_{x(n)} \\ a_{y(n)} \\ a_{z(n)} \end{bmatrix}$$

is the measured acceleration data and

$$a_{rotated(n)} = \begin{bmatrix} a_{r(n)} \\ a_{t(n)} \\ a_{g(n)} \end{bmatrix}$$

is the "actual" acceleration data that is rotated so that the measured radial rotation and the actual radial rotation should be aligned if $cost = 0$. Solving Equation 2 gives the values for each component of $a_{rotated(n)}$:

$$a_{r(n)} = a_{x(n)}cos(-\theta) - a_{y(n)}sin(-\theta)$$
$$a_{t(n)} = a_{x(n)}sin(-\theta) + a_{y(n)}cos(-\theta)$$
$$a_{g(n)} = 0$$

The algorithm that is used to optimize the cost function is called Stochastic Gradient Descent or SGD. Its only parameters are α, the learning rate, and $\vec{x}_0$, the starting point. The learning rate dictates how much of the gradient is taken to approximate the next step. SGD works by looping the following until $\vec{x}$ has converged:

For a cost function $C(\vec{x})$ and step n,

$$\vec{x}_{n+1} = \vec{x}_n - \alpha \cdot \nabla C(\vec{x}_n) \tag{3}$$

but, since the true $\nabla C(\vec{x})$ is not known, it is estimated using

$$\nabla C(\vec{x}) \approx \frac{1}{2\gamma} \begin{bmatrix} C\left(\begin{bmatrix} x_1+\gamma \\ x_2 \\ \vdots \end{bmatrix}\right) - C\left(\begin{bmatrix} x_1-\gamma \\ x_2 \\ \vdots \end{bmatrix}\right) \\ C\left(\begin{bmatrix} x_1 \\ x_2+\gamma \\ \vdots \end{bmatrix}\right) - C\left(\begin{bmatrix} x_1 \\ x_2-\gamma \\ \vdots \end{bmatrix}\right) \\ \vdots \end{bmatrix} \tag{4}$$

where γ is the precision of the approximation. In this project, it is set, by default, at 0.00001.

The other algorithm that is used in this project is called Adam[1]. Its parameters are α, the learning rate, β_1 and β_2, the exponential rates of decay which are in the interval $[0, 1)$, and $\vec{x}_0$, the starting point. This algorithm loops the following operations until $\vec{x}$ has converged:

For a cost function $C(\vec{x})$ and step n,

$$\vec{m}_n = \beta_1 \cdot \vec{m}_{n-1} + (1 - \beta_1) \cdot \nabla C(\vec{x}_n)$$

$$\vec{v}_n = \beta_2 \cdot \vec{v}_{n-1} + (1 - \beta_2) \cdot (\nabla C(\vec{x}_n))^2$$

$$\hat{m}_n = \frac{\vec{m}_n}{1 - \beta_1^n}$$

$$\hat{v}_n = \frac{\vec{v}_n}{1 - \beta_2^n}$$

$$\vec{x}_{n+1} = \vec{x}_n - \alpha \frac{\hat{m}_n}{\sqrt{\hat{v}_n} + \epsilon}$$

where $\vec{m}$ is the first moment vector estimate, $\vec{v}$ is the second moment vector estimate, $\hat{m}$ is the bias-corrected first moment vector estimate, $\hat{v}$ is the bias-corrected second moment vector estimate, and ϵ is the error. The initial parameters of this algorithm are $\vec{m}_0 = 0$ and $\vec{v}_0 = 0$. The same approximation for $\nabla C(\vec{x})$ that is used for SGD is used here.

2 Methods and Implementation

To achieve this goal, a scenario was established where an accelerometer would be strapped to a rotating object like a turntable.

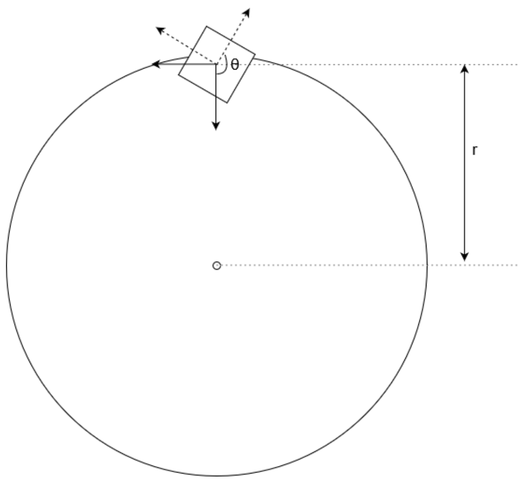

Figure 1: Experimental Setup

The accelerometer as seen in Figure 1 would be fixed in such a way that the angle θ would be constant throughout the data collection. The object would then be rotated about its central axis as the radius would also stay constant. This was done to minimize the divergence between the points so that they could be used to compare their respective radius and angle values with the other points.

A data analysis library was developed to find these parameters. It uses the 3.7 version of Python with Tensorflow[1] which is Google's machine learning module for Python that comes with a multitude of implementations of various optimization algorithms such as the Stochastic Gradient Descent (SGD), Adam and Momentum algorithms, Numpy[2] which is an advanced mathematics library that is used for its vector and matrix implementations, and Matplotlib[3] which is used to output graphs of the data. The library is made to be modular meaning that its different parts can be swapped to fit a specific need.

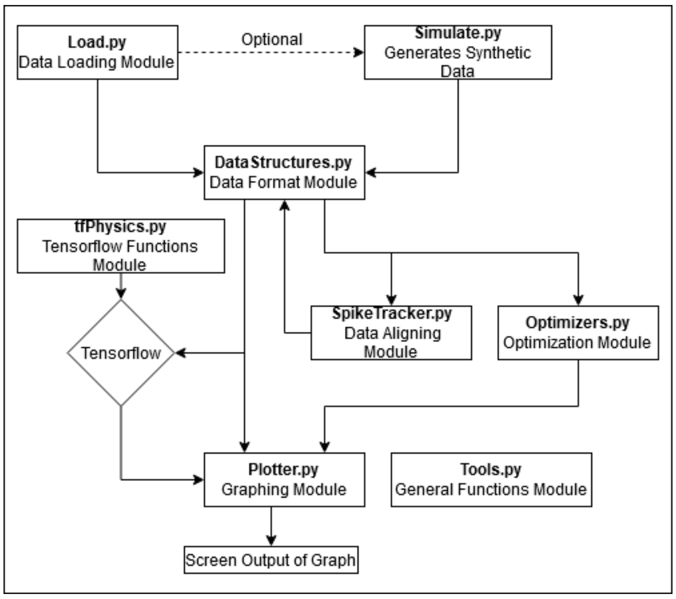

Figure 2: General Flow of the Library

- Load.py is the data loading module of the library which supports only .csv files with angular velocity in rad/s, or acceleration data in g's, m/s^2 or in "counts" which are a specific unit for the GCDC X2-2 and X16 accelerometers.
- Simulate.py is the synthetic data generation module of the library. It can create new angular velocity data given $\alpha(t)$, an angular acceleration function, N, the number of data points, Δt, the difference in time between the data points in seconds, and ω_0, the initial angular velocity in rad/s. It can also convert this new data or angular velocity data from Load.py into acceleration data given r, a radius. It can also rotate any acceleration data by an angle θ.
- DataStructures.py is the module that dictates how the program reads and uses data. It defines two main classes: *RotaryData* and *AccelData*. *RotaryData* stores angular velocity data and *AccelData* stores acceleration data.
- SpikeTracker.py aligns multiple data classes so that all their time frames are synced with each other. It takes *AccelData*s and *RotaryData*s and returns the aligned version of them.

[1]https://www.tensorflow.org/
[2]https://numpy.org/
[3]https://matplotlib.org/

- Optimizers.py is the optimization module that was written for this project. It consists of 4 classes at the moment:
  - *Optimizer* which defines all the basic functions an optimizer should have if it is implemented. The classes that inherit its properties are:
    - *AdamAlgorithm_1D* implements the optimization algorithm, Adam [1]. This version of Adam only works to optimize 1 variable.
    - *SGD_1D* implements the Stochastic Gradient Descent algorithm to optimize 1 variable.
    - *SGD_2D* which is an extension of *SGD_1D* that optimizes 2 variables.
- Tools.py houses general purpose functions such as vector rotation and a function to find the smallest value in an array called *min_lambda*
- tfPhysics.py is the Tensorflow-enabled version of Tools.py that has the added function of having the cost functions for the Tensorflow tests.
- Plotter.py is the graphing module of the library. It implements the functions of the Matplotlib plotting library for Python in a way that is easier to use for our purposes such as plotting the acceleration and time data of an *AccelData* instance.

For example, it is possible to take drop the *SpikeTracker.py*, *Optimizers.py* and *Tools.py* modules to make a Tensorflow-only test and it would still work.

3 Results and Analysis

The library can find the angle if it's given the radius and vice-versa.

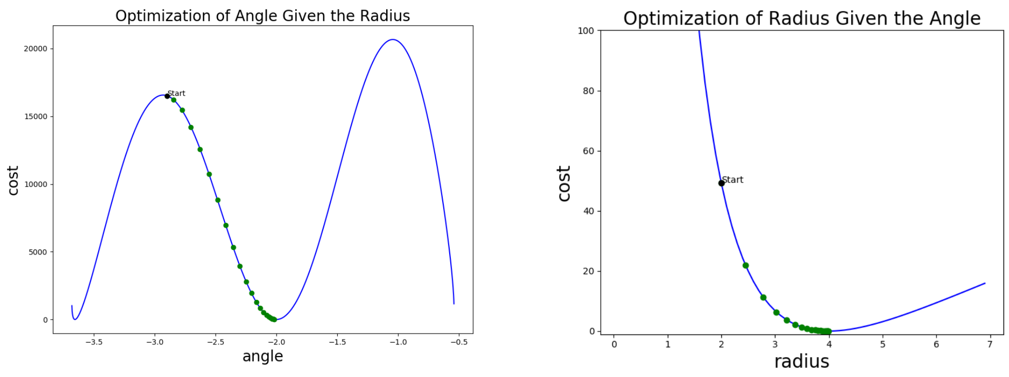

Figure 3: Results of 1 Variable Optimization

Currently, both the Tensorflow library and the *Optimizer.py* module have trouble finding both of them at the same time due to the large regions of NaN results (See Figure 4)

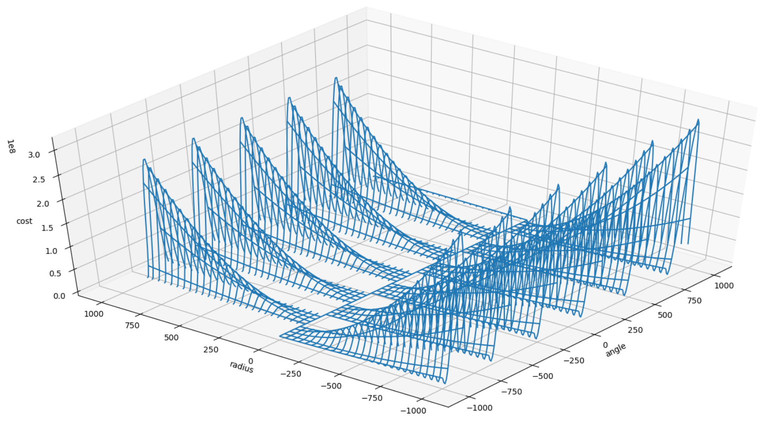

Figure 4: Plot of the cost function

The periodic nature of cosine and sine functions is reflected on the angle axis shown by the cost function as the blue wireframe.

4 Discussion

The library currently supports both 1 variable and 2 variable optimization using the project's own optimization module with the program flow

$$\textit{Simulate.py} \longrightarrow \textit{DataStructures.py} \longrightarrow \textit{Optimizers.py} \longrightarrow \textit{Plotter.py} \longrightarrow \textit{Screen Output}$$

or a similar program flow using Tensorflow and tfPhysics.py instead of Optimizers.py but it cannot optimize the cost function that is being used when presented with a 2 variable scenario. The problem does not seem to be the *Optimizer.py* module as, if it is given a function which is continuous everywhere, it successfully finds the closest minimum. (See Figure 5)

Due to there being large regions of the cost function with a value of NaN, whether the optimization finds the right values depends on the starting point and, since they are unknown at the start, the starting point cannot be placed in such a way that the optimization would avoid those NaN regions. This happens because the third term of the cost function is $\sqrt{\frac{a_r}{r}}$ which gives NaN when we rotate the measured acceleration vector in such a way that the resulting a_r value is negative. This could be fixed by extending the graph into the imaginary number range but it would bring the possibility of finding imaginary values for the angle and the radius which would not be very useful due to the real nature of this project.

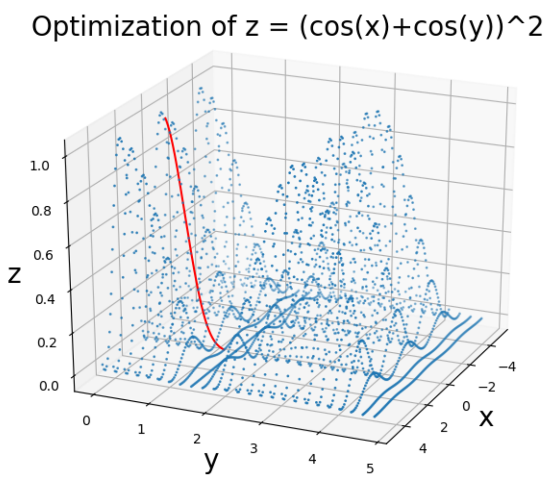

Figure 5: 2 Variable Optimization

During this project, it was learned that documentation should be done during the development and not after, that building a project plan in advance is a lot more efficient than not having one, and that making sure that the whole team has the same version of whatever module is being used prevents future incompatibility problems.

5 Conclusion

The intention of this project was to make a proof-of-concept Python library which could retrieve angular motion parameters from acceleration data and be expanded upon by future users to fit a more specific need. The current version of the library supports both 1 variable and 2 variable optimization. It can find θ, the angle between the true radial acceleration and the measured radial acceleration, when given r, the radius of rotation. It can also find r given θ. The only thing that is missing, as mentioned before, is to either change the cost function or find a way to limit the search range of the optimization module. In conclusion, the project is very close to meeting the goal of verifying the possibility of replacing a gyroscope by an accelerometer.

6 Acknowledgements

This project was made possible under the supervision of Prof. Larnder (research supervisor) and Prof. Silerova (course coordinator), and with the collaboration of Hoang Anh, Justin and Adamo. I would like to express my gratitude to everyone who worked on this project with me.

References

[1] Kingma D. P., Ba J. 2015. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representation 2015.*