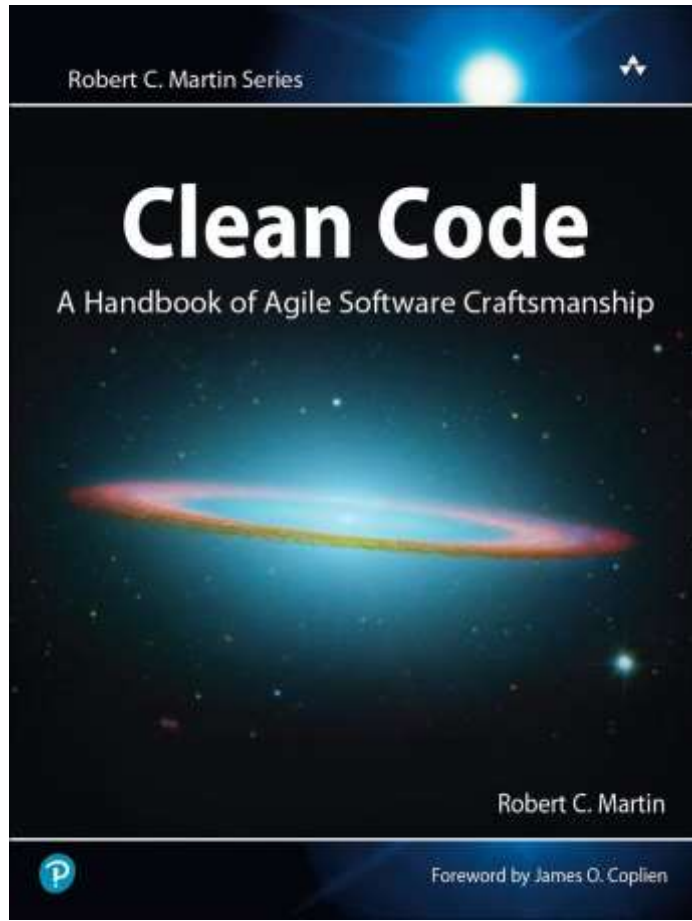


# **Gyroscope & Accelerometer Data Fusion**



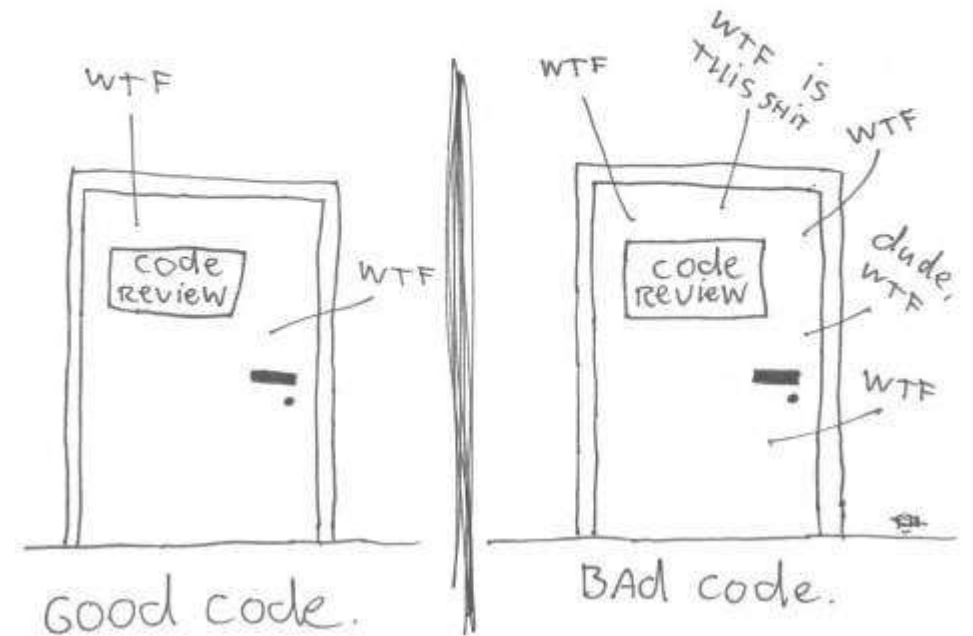
**Vincent van Hees – 7th of April 2021**  
**Webinar series OS Software in Physical Behaviour Research Field**

# Clean Code by Uncle Bob



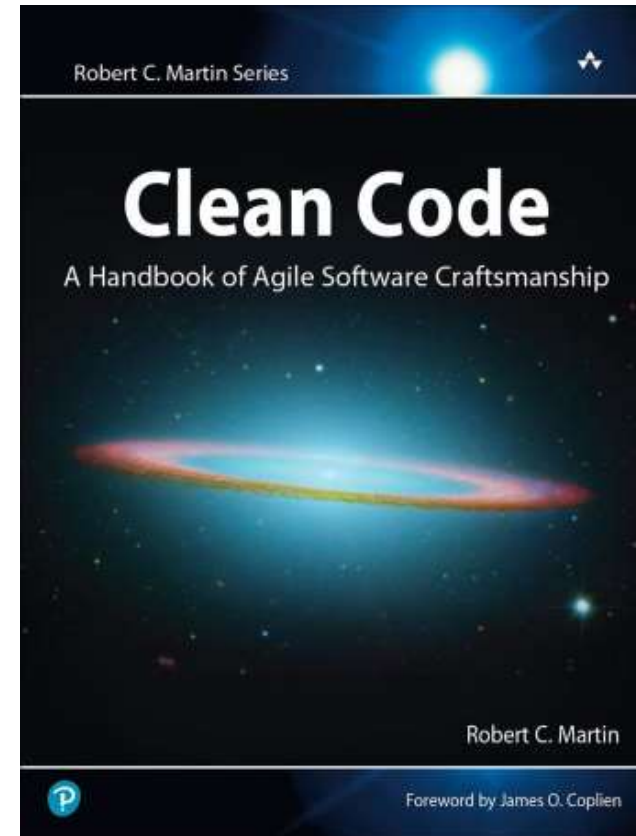
Training videos: <https://cleancoders.com>  
Also 'free' recordings from conference talks on YouTube

The ONLY VALID MEASUREMENT  
OF CODE QUALITY: WTFs/MINUTE



# Clean Code by Uncle Bob

- Object names
- Function names
- Comment
- Formatting
- Object and data structures
- Error Handling
- Unit-tests
- ....

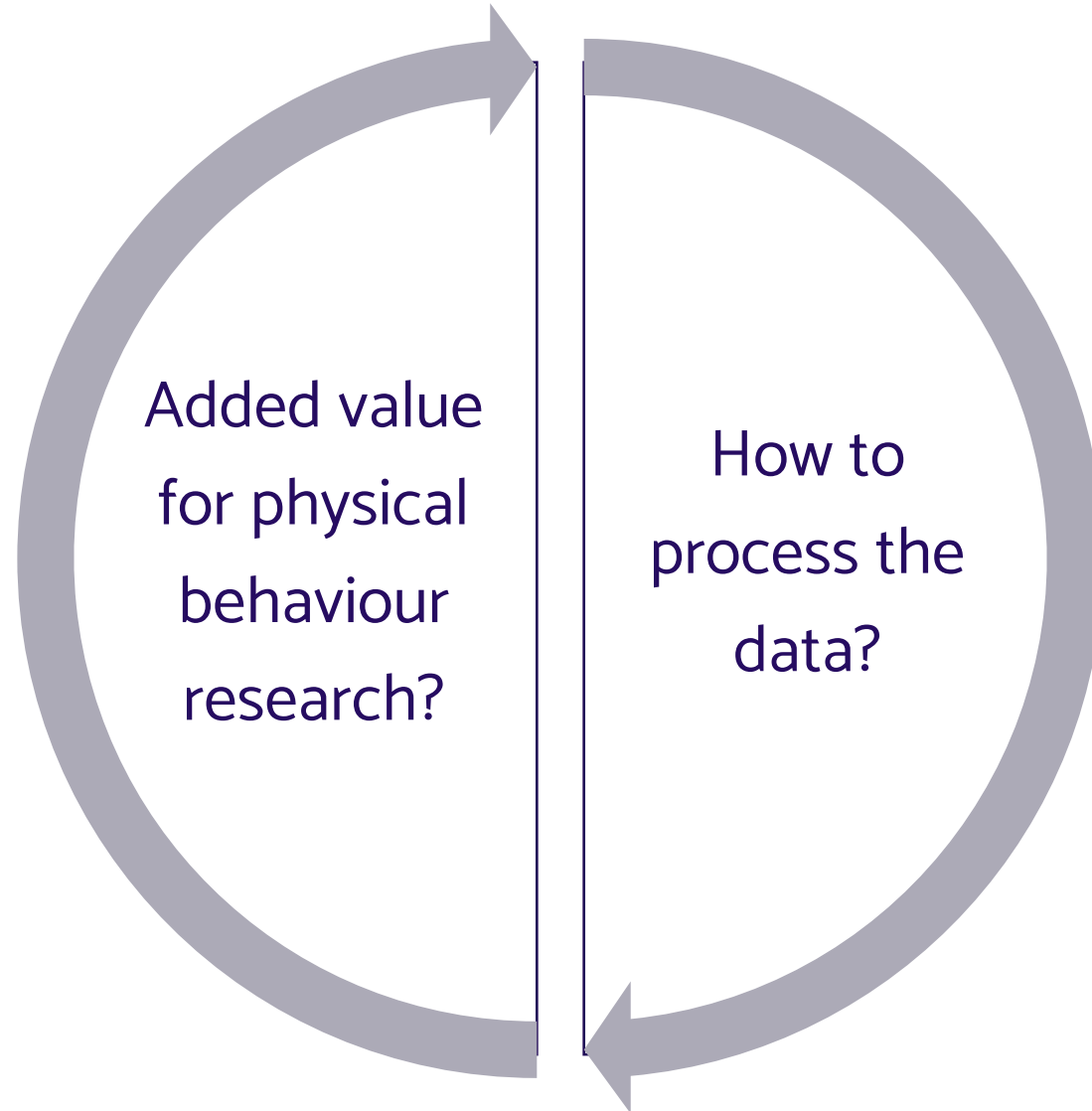


# Gyroscopes

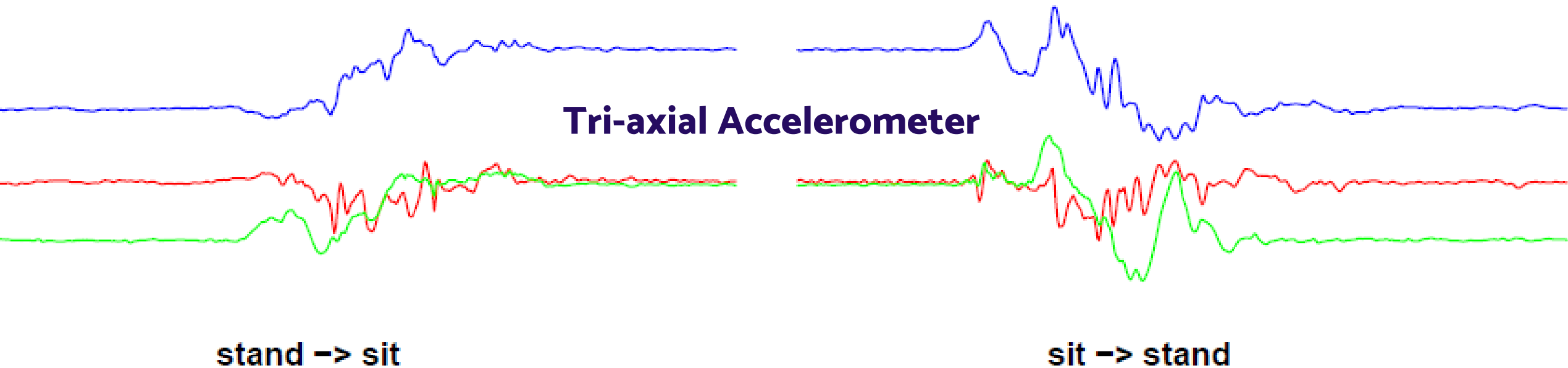
- Measures angular velocity around each axis
- Works based on resonating mass, see also [1, 2].
- Feasibility has improved in recent years
  - 7 days @ 100 Hertz with 3 accelerometer and 3 gyroscope
  - Based on small battery
- Potential challenges:
  - More data storage
  - More data processing
  - Higher price than accelerometer-only solutions

1. N. Yazdi, F. Ayazi, and K. Najafi, "Micromachined inertial sensors," *Proceedings of the IEEE*, vol. 86, no. 8, pp. 1640–1658, 1998.
2. Motion tracking in field sports using GPS and IMU - MSc Thesis - Matthijs Roobeek

# Questions to be answered



# Example data collected from wrist



# Advantage of using a gyroscope?

	Orientation tracking during Statistic conditions	Orientation tracking during Dynamic conditions
Accelerometer	Good	Poor
Gyroscope	Poor	Good*
'Fusion' of Accelerometer and Gyroscope	Good*	Good*

\* Orientation relative to poles (north/south) remains difficult without additional magnetometer

Applications: Drones, Robots, Gait analysis, Animation industry, ...

# Fusion at what level?

- Classification
- Feature
- Raw data





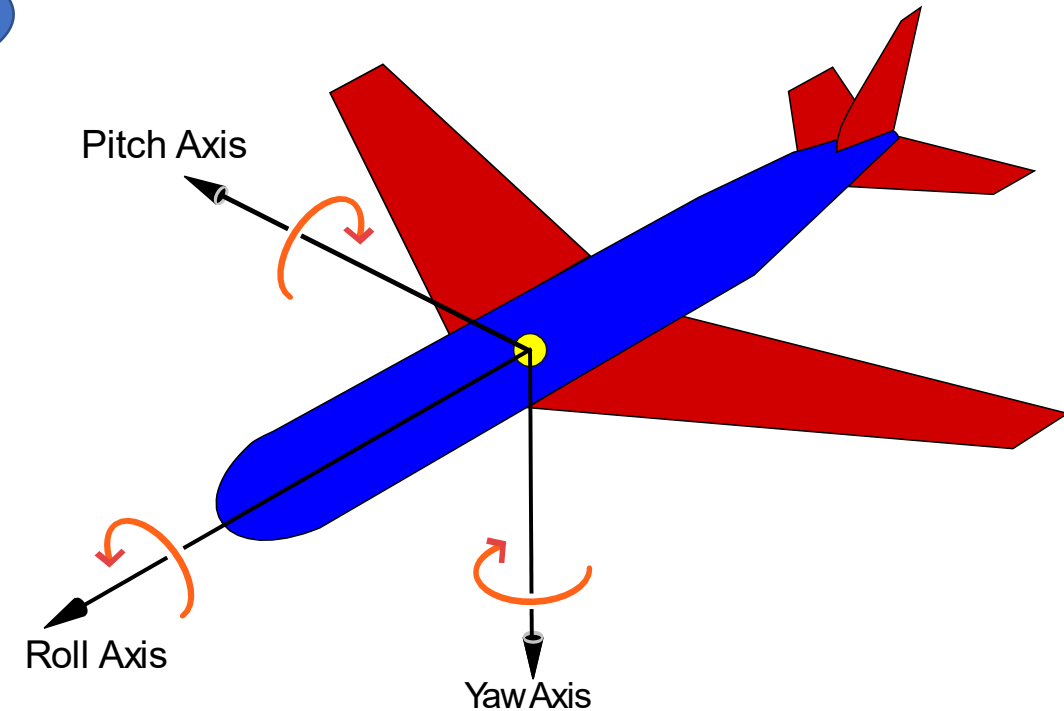
# Exploration of Fusion algorithms

Starting point:

- Pitch / Roll / Yaw angle from Accelerometer
- Pitch / Roll / Yaw angle change from Gyroscope

Rotation axis  
parallel to earth

Rotation axis  
equals sensor axis



Source Wikipedia, licence:  
<https://creativecommons.org/licenses/by-sa/3.0/deed.en>

# Exploration of Fusion algorithms

- Luinge and Veltink 2005 <https://link.springer.com/article/10.1007/BF02345966>
  - No code with original paper
  - External implementations:
    - Matlab: <https://github.com/tytell/accelmat>
- Madgwick *et al.* 2009 <https://ieeexplore.ieee.org/document/5975346>
  - C and Matlab code with original paper <https://x-io.co.uk/open-source-imu-and-ahrs-algorithms>
  - External implementations:
    - Python: <https://github.com/Mayitzin/ahrs/>, [https://github.com/morgil/madgwick\\_py](https://github.com/morgil/madgwick_py), and more
    - R: <https://github.com/cran/RAHRS> (not on CRAN anymore)
    - C++: <https://github.com/arduino-libraries/MadgwickAHRS>
- ... and more

**... but without a full understanding this  
may be risky**

# Own attempt to write a fusion algorithm

- Input:
  - 3 x gyro and 3 x acc
  - Function *g.cwaread* from the GGIR R-package to read Axivity's AX6 .cwa data
- Output:
  - Orientation of gravity relative to sensor coordinate system
  - Local acceleration without gravity
- Accuracy: Not well tested, only visual checks
- Speed: 24 hours @ 100 Hertz in 30 seconds

<https://cran.r-project.org/web/packages/GGIR/vignettes/SensorFusionWithGGIR.html>

The screenshot displays the '4 Algorithm description' section of the GGIR vignette. On the left, a table of contents lists sections from 1 to 6, with '4 Algorithm description' highlighted. The main content area shows the title '4 Algorithm description' followed by a paragraph: 'In this chapter I describe the algorithm as implemented in GGIR function `separategnecity`.' Below this is section '4.1 Input', which states 'The input arguments of the function are:' and lists three items: `acc` (a three-column matrix with accelerometer values in g-units), `gyr` (a three-column matrix with gyroscope values in radians per second), and `sf` (the sample frequency in Hertz). Section '4.2 Coordinate system' explains that calculations are done within the local Cartesian coordinate system of the sensor. Section '4.3 Angular velocity vector' describes how gyroscope signals are split into an angular velocity vector. A code block shows R code for calculating the magnitude of the angular velocity vector (`theta`) and creating a zero vector (`nzzero`). The final line of code shows `theta` being divided by the sample frequency (`sf`) to get radians per time step.

4 Algorithm description

In this chapter I describe the algorithm as implemented in GGIR function `separategnecity`.

4.1 Input

The input arguments of the function are:

- `acc`: a three-column matrix with the accelerometer values in g-units
- `gyr`: a three-column matrix with the gyroscope values in radians per second
- `sf`: the sample frequency in Hertz. The algorithm assumes a constant sample frequency throughout the recording. Please note that GGIR has function `resample` to aid in converting irregularly sampled data to a regular sample frequency

4.2 Coordinate system

All calculations will be done within the local Cartesian coordinate system of the sensor. In contrast to other algorithms that I encountered I am not attempting to calculate the roll, pitch or yaw of the sensor, because calculating those would require transitioning between multiple coordinate systems. For example, pitch derived from a gyroscope is expressed in the local coordinate system of the sensor, while pitch derived from an accelerometer is relative to the orientation of gravity. Instead, I will derive the orientation of gravitational acceleration relative to the local coordinate system of the sensor and use the angular velocity vector relative to the local coordinate system to rotate the orientation of the gravity during time segments involving movement. By tracking the orientation of gravity, which I will refer to as the gravity vector, within the local coordinate system of the sensor over time I will be able to subtract gravity from the original acceleration signals.

4.3 Angular velocity vector

The gyroscope signals represent angular velocity in radians per second per sensor axis. As I want to rotate the gravity vector, I need to split the gyroscope signals up in an angular velocity vector with a magnitude (`theta` in the code) per time step and 3 coordinates for the vector orientation (object `nv` in the code) relative to the sensors coordinate system.

```
theta = sqrt(rowSums(gyr ^ 2)) # theta: magnitude of angular velocity around orientation vector (nv)
NV = matrix(0, N, 3) # N: number of samples
nzzero = which(theta > 0)
NV[nzzero,] = as.numeric(gyr[nzzero,] / theta[nzzero])
```

Next, `theta` is divided by the sample frequency to get radians per time step

```
theta = theta / sf
```

# Lessons learnt so far

1. Algorithm speed is critical
2. Gold standard benchmark needed
3. Do not give up on existing algorithms yet
4. Unclear role of hardware specifications
5. Output of interest varies between algorithms

# Next steps

- Find / create benchmarks:
  - Simulated data
  - Optical system to track orientation
  - Energy expenditure or Activity type classification
- Get standard algorithms to work, e.g. Madgwick
- Look into gyroscope calibration:
  - Check for bias based on non-movement episodes?
  - Check for scaling error based on ...?
- Look into role of hardware specifications
- Work out how to go from pitch/roll/yaw to  $g$ -orientation
- Collaboration

# Questions?

