

# What do drones think about?



Shallow dive into embedded Rust and sensors.

# \$whoami

Dawid Królak

- General purpose software engineer
- Rust enthusiast
- Amateur volleyball player
- Aerorust community core member
- Passionate about European space exploration and engineering programs.

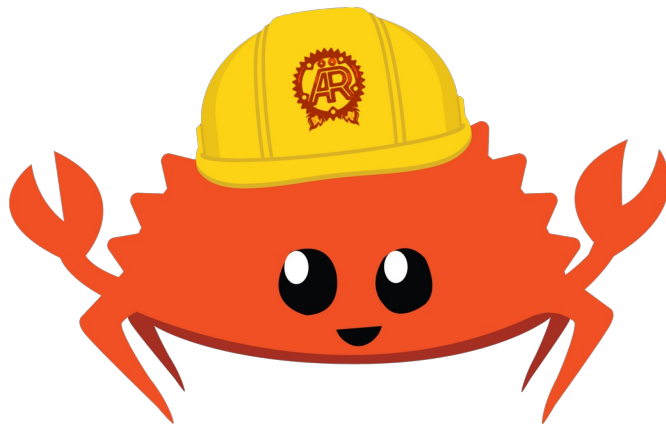
<https://twitter.com/@Taaviczus>

<https://github.com/taavit>

# Aerorust

Our community aims to grow the Rust programming language ecosystem for aerospace applications because it "empowers everyone to build reliable and efficient software".

- Nanosat workshops
- NMEA crate
- Free-flight-stabilization
- Casual meetups



<https://aerorust.org/>

What to choose...



Inertial Measurement Unit

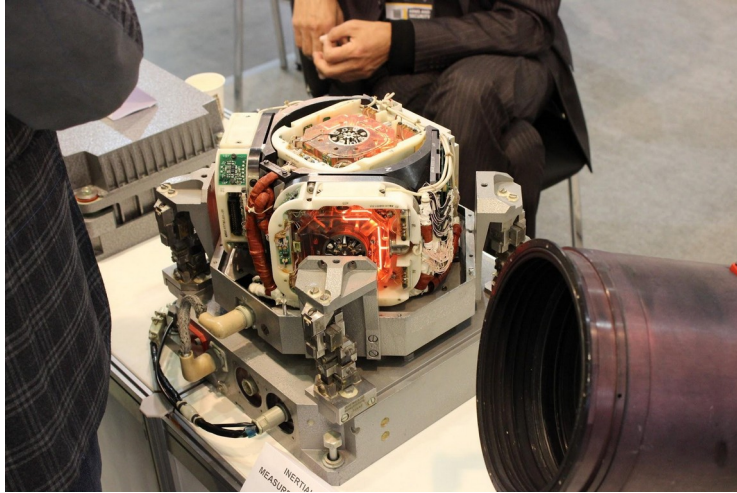
# Basics

# What is IMU?

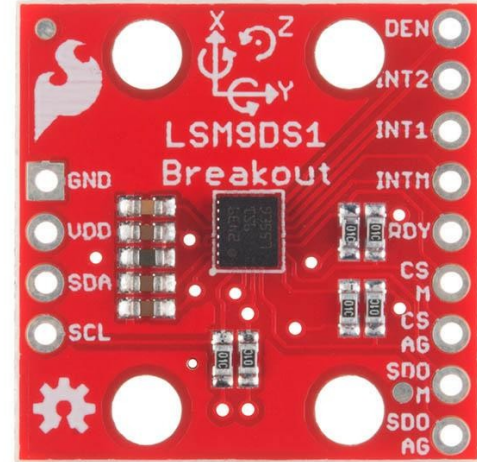
Device that measures and report forces acting on a given object that combined can provide orientation in a given geometric space (reference frame).

It's about attitude aka orientation in space, not position. (For position we have Inertial Navigation System)

# IMU in real life



VictorAnyakin, CC BY-SA 4.0  
<<https://creativecommons.org/licenses/by-sa/4.0/>>,  
via Wikimedia Commons



# What IMU reports

## Quaternions

- Special number system (Complex numbers on steroids)
- Easy to calculate (matrices) by computer
- For some people new concept needed to learn

$$a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$$

## Euler angles

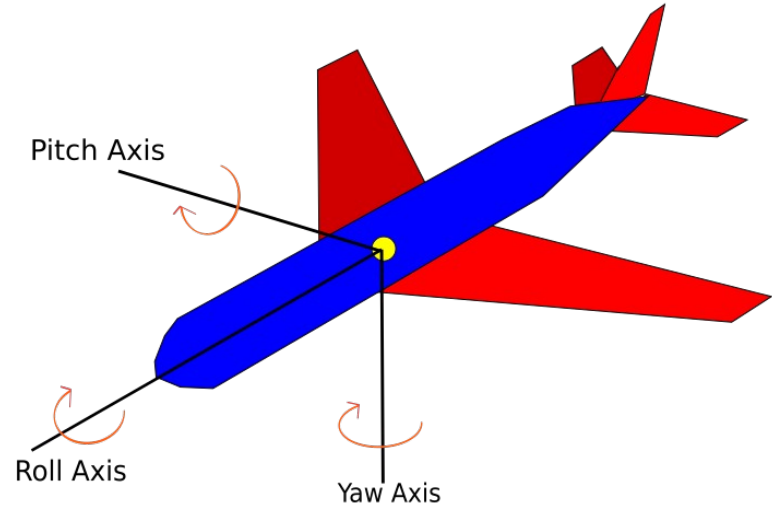
- Easy to understand
- Easy to read
- Not the best for machine calculation
- Good enough for now



# Euler angles

To recreate orientation we have to determine:

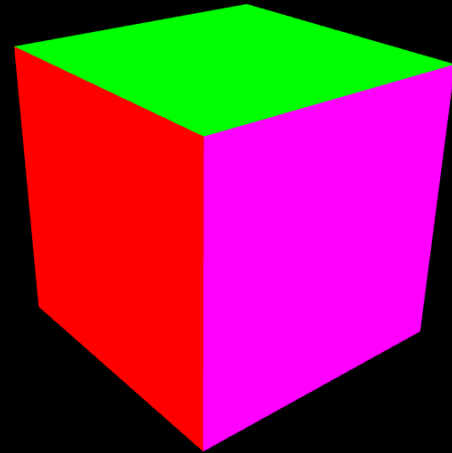
- yaw
- pitch
- roll



**Use the same order of angles!**

# Project!

Display cube with real time  
orientation



---

## Step by step:

- Find MCU, sensors, breadboards
- Connect everything
- Write firmware
- ~~Write frontend~~
- Plug and play!

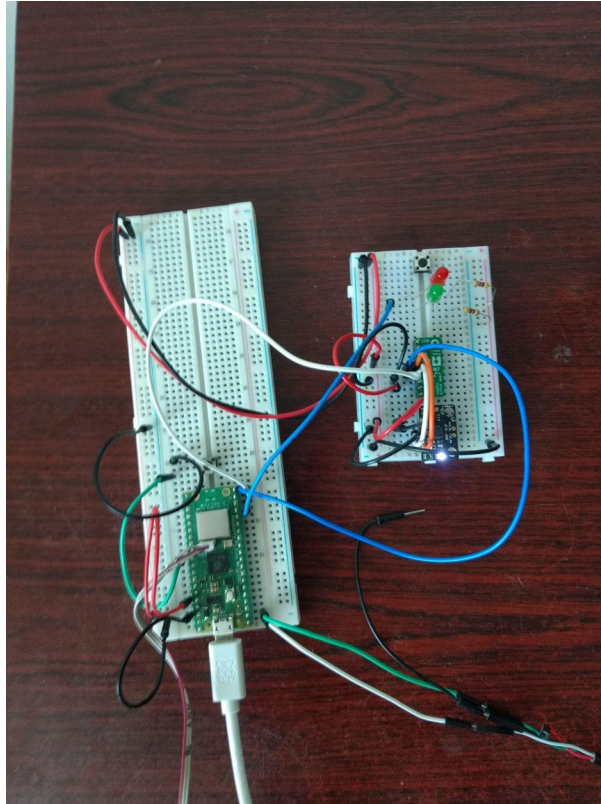
# Setup hardware and software

# Finding stuff

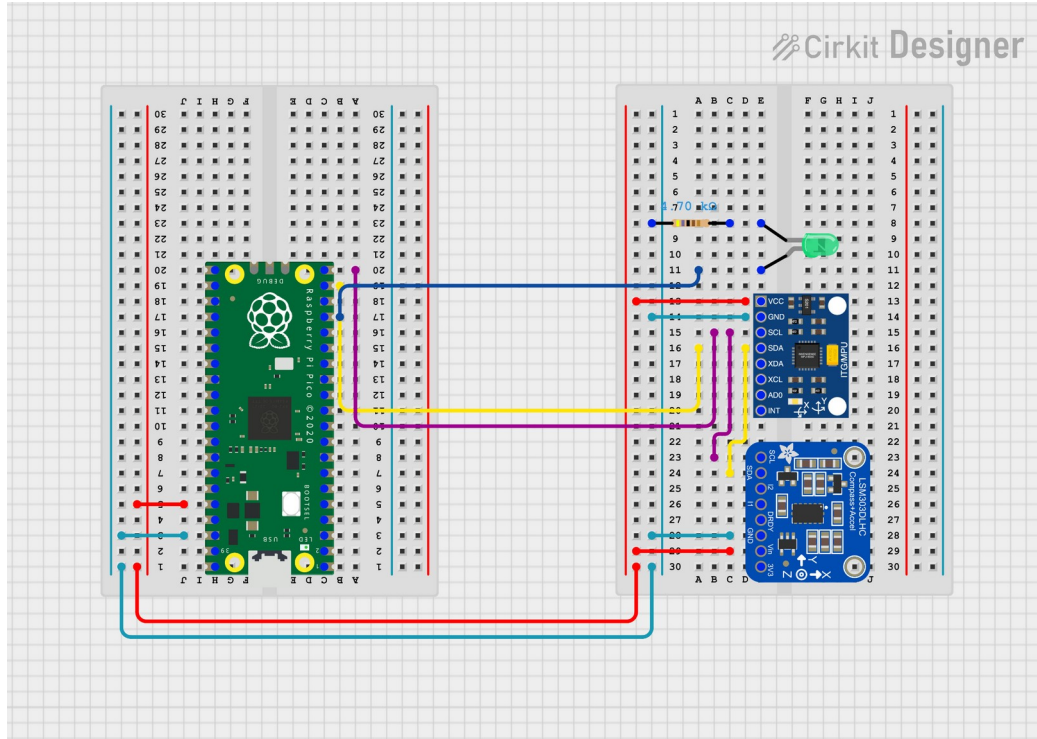
- 2x Breadboard
- Raspberry Pico
- LSM303D (acc+mag)
- MPU6050 (gyro+acc)
- 1 x LED
- 1 x 4.7k resistor



# Connecting



# Connecting



# Firmware



# Embassy

- Multiple tasks
- Well documented
- De-facto standard for embedded rust framework

## Embassy

About Documentation API Reference Blog Chat GitHub

### The next-generation framework for embedded applications

Write safe, correct and energy-efficient embedded code faster, using the Rust programming language, its async facilities, and the Embassy libraries.

Get started

#### Rust + async ❤️ embedded

The [Rust programming language](#) is blazingly fast and memory-efficient, with no runtime, garbage collector or OS. It catches a wide variety of bugs at compile time, thanks to its full memory- and thread-safety, and expressive type system.

Rust's [async/await](#) allows for unprecedentedly easy and efficient multitasking in embedded systems. Tasks get transformed at compile time into state machines that get run cooperatively. It requires no dynamic memory allocation, and runs on a single stack, so no per-task stack size tuning is required. It obsoletes the need for a traditional RTOS with kernel

```
use defmt::Info;
use embassy_executor::Spawner;
use embassy_nrf::gpio::{AnyPin, Input, Level, Output, OutputDrive, Pin, Pull};
use embassy_nrf::Peripherals;
use embassy_time::{Duration, Timer};

// Declare async tasks
#[embassy_executor::task]
async fn blink(pin: AnyPin) {
    let mut led = Output::new(pin, Level::Low, OutputDrive::Standard);

    loop {
        // Timingout is globally available, no need to mess with hardware timers.
        led.set_high();
        Timer::after_millis(150).await;
        led.set_low();
        Timer::after_millis(150).await;
    }
}

// Main is itself an async task as well.
#[embassy_executor::main]
async fn main(spawner: Spawner) {
    // Initialize the embassy runtime.
    let p = embassy_nrf::init(Default::default());

    // Spawned tasks run in the background, concurrently.
    spawner.spawn(blink(p.P0_13.degrade())).unwrap();

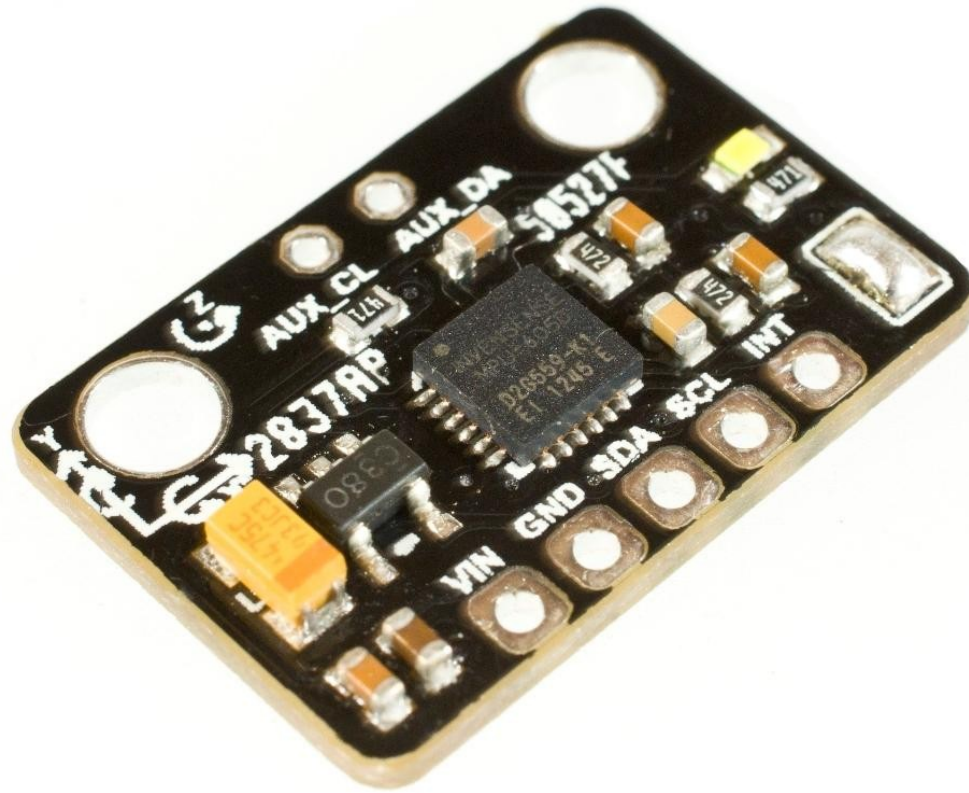
    let mut button = Input::new(p.P0_11, Pull::Up);
    loop {
        // Asynchronously wait for GPIO events, allowing other tasks
```

## Code setup (embassy)

```
let p = embassy_rp::init(Default::default());
let i2c = embassy_rp::i2c::I2c::new_blocking(p.I2C1, scl, sda, Config::default());

let config = uart::Config::default();

let mut uart = uart::Uart::new(
    p.UART0, p.PIN_0, p.PIN_1, Irqs, p.DMA_CH0, p.DMA_CH1, config,
);
```



MPU6050

# MPU6050 - Gyro + Accelerometer

- Ready to go synchronous driver for sensor
- Contains DMP (Digital Motion Processor)
  - Provides ready to go attitude data
  - Needs upload firmware to the sensor
  - Check licences!

# Gyroscope data

- Read angular velocity (deg/s)
- Read time between measurements
- Integrate
- Voila!

# Reading data from sensor

```
let mut now = Instant::now();
let mut pitch = 0.0;
let mut roll = 0.0;
let mut yaw = 0.0;

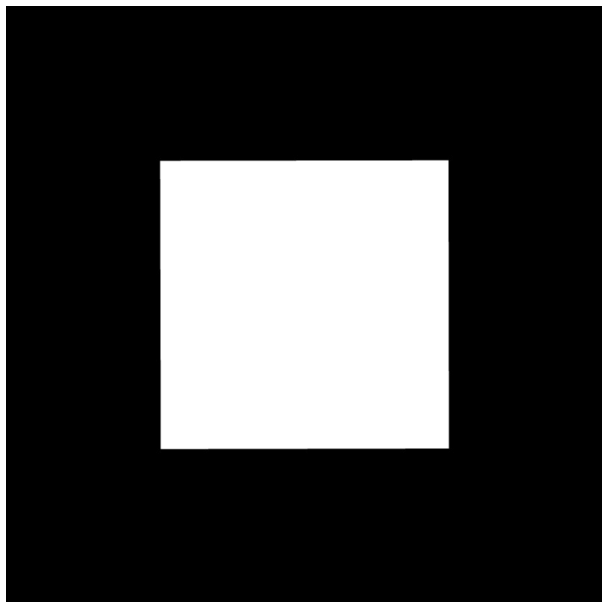
loop {
    let mut buffer = String::<64>::new();
    let gyro = sensor.gyro().unwrap().scaled(GyroFullScale::Deg2000);
    let cur_time = Instant::now();
    let timestep = ((cur_time - now).as_micros() as f32) / 1_000_000.0;
    now = cur_time;
    let step_x = (gyro.x() * timestep).to_radians();
    let step_y = (gyro.y() * timestep).to_radians();
    let step_z = (gyro.z() * timestep).to_radians();

    pitch += step_x;
    roll += step_y;
    yaw += step_z;

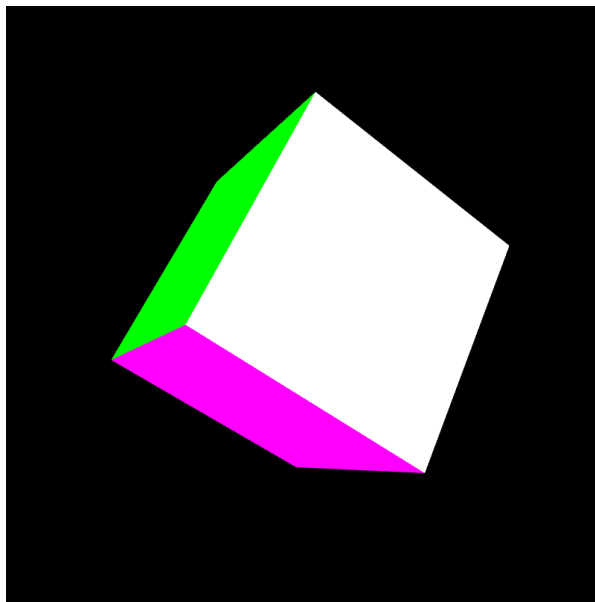
    write(&mut buffer, format_args!("{}",{:0.5},{:0.5},{:0.5}*/r\n", f32::from(yaw), f32::from(pitch), f32::from(roll))).unwrap();
    uart.write(buffer.as_bytes()).await.unwrap();
}
```

# Results

How it's started



How it's going (after few minutes)



# Gyro is (not) enough

- Detect current angular velocity, not attitude (need integrate and sum)
- Noise (as every sensor)
- Drift
- Not perfect

Timestep	x	y	z
0.009456	-0.111343330	0.10919662	-0.0060349824
0.009466	-0.111383624	0.10924699	-0.0060349824
0.009452	-0.111393680	0.10921682	-0.0060349824
0.009487	-0.111413874	0.10922691	-0.0060450790
0.009451	-0.111413874	0.10923697	-0.0060249628
0.009463	-0.111413874	0.10923697	-0.0060148920
0.009551	-0.111444370	0.10927763	-0.0060047274
0.009459	-0.111484630	0.10928769	-0.0059443284
0.009445	-0.111484630	0.10931785	-0.0059141736
0.009388	-0.111504614	0.10934782	-0.0059041830
0.009388	-0.111514606	0.10934782	-0.0059041830
0.009450	-0.111524664	0.10935788	-0.0058639552
0.009454	-0.111524664	0.10938806	-0.0058538940
0.009449	-0.111554830	0.10937800	-0.0058840616
0.009484	-0.111564930	0.10938810	-0.0058840616
0.009462	-0.111585066	0.10938810	-0.0058739916
0.009467	-0.111605210	0.10939817	-0.0058739916
0.009451	-0.111625330	0.10938811	-0.0058639340

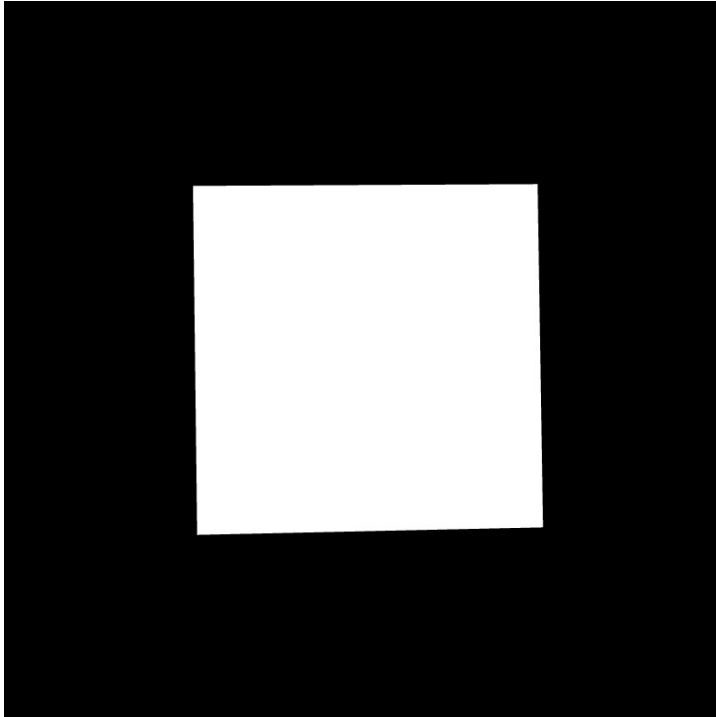


## Let's add accelerometer

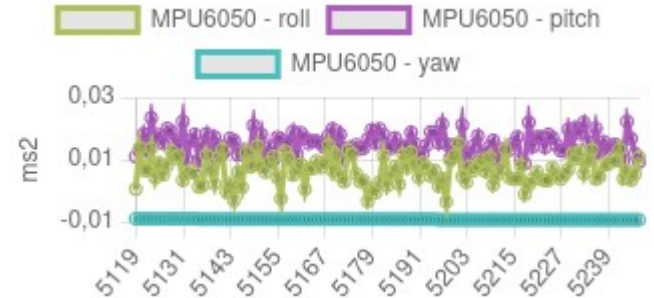
```
let accel_yaw = 0.0;  
let accel_pitch = -libm::atan2f(accel.y(), accel.z());  
let accel_roll = -libm::atan2f(  
    -accel.x(),  
    libm::sqrtf(accel.y() * accel.y() + accel.z() * accel.z()),  
);
```

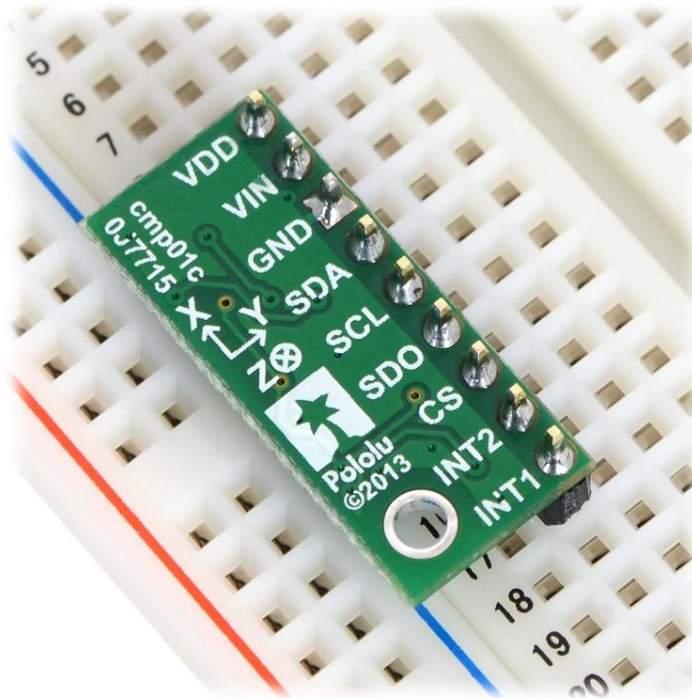
**Cannot calculate yaw**  
**This works thanks to gravity in Z axis**

# Results



Problem: **Noise!**





<https://www.pololu.com/product/2127>

## LSM303D - Accelerometer + Magnetometer

# LSM303D - Driver

### Register description

## 8 Register description

The device contains a set of registers for configuration, status, and data. This section describes the registers used for acceleration and magnetic data.

**Output register mapping**

The table below provides a listing of the 8-bit registers embedded in the device and the corresponding addresses.

Name	Type	Register address	Default	Comment
		Hex	Binary	
Reserved		00-04	000 0101	Reserved
TEMP_OUT_L	r	05	000 0110	Output
STATUS_OUT_M	r	06	000 0111	Output
OUT_X_L_M	r	07	000 1000	Output
OUT_Y_L_M	r	08	000 1001	Output
OUT_Z_L_M	r	09	000 1010	Output
OUT_X_H_M	r	0A	000 1011	Output
OUT_Y_H_M	r	0B	000 1100	Output
OUT_Z_H_M	r	0C	000 1101	Output
Reserved		0D	000 1110	Reserved
WHO_AM_I	r	0E	000 1111	Output
Reserved		0F	000 1111	Reserved
INT_CTRL_M	r	10-11	01001001	Reserved
INT_SRC_M	r	12	000 1000	Reserved
INT_THS_L_M	rw	13	001 0010	Reserved
INT_THS_H_M	rw	14	001 0011	Reserved
OFFSET_X_L_M	rw	15	001 0100	Output
OFFSET_X_H_M	rw	16	001 0101	Output
OFFSET_Y_L_M	rw	17	001 0110	Output
OFFSET_Y_H_M	rw	18	001 0111	Output
REFERENCE_X	rw	19	001 0100	Output
REFERENCE_Y	rw	1A	001 0101	Output
REFERENCE_Z	rw	1B	001 0110	Output
CTRL0	rw	1C	001 0111	Output
CTRL1	rw	1D	001 1000	Output
CTRL2	rw	1E	001 1001	Output
	rw	20	010 0000	Output
	rw	21	010 0001	Output

### Register description

## 8.15 REFERENCE\_Z (1Eh)

Reference value for high-pass filter for Z-axis acceleration data.

## 8.16 CTRL0 (1Fh)

BOOT	FIFO_EN	FTH_EN	0 <sup>(1)</sup>	HP_Click	HPIS1	HPIS2
0 <sup>(1)</sup>	0 <sup>(1)</sup>	0 <sup>(1)</sup>	0 <sup>(1)</sup>	0 <sup>(1)</sup>	0 <sup>(1)</sup>	0 <sup>(1)</sup>

1. These bits must be set to '0' for correct operation of the device.

### Table 32. CTRL0 register description

BOOT	Reboot memory content. Default value: 0 (0: normal mode; 1: reboot memory content)
FIFO_EN	FIFO enable. Default value: 0 (0: FIFO disable; 1: FIFO enable)
FTH_EN	FIFO programmable threshold enable. Default value: 0 (0: disable; 1: enable)
HP_Click	High-pass filter enabled for click function. Default value: 0 (0: filter bypassed; 1: filter enabled)
HPIS1	High-pass filter enabled for interrupt generator 1. Default value: 0 (0: filter bypassed; 1: filter enabled)
HPIS2	High-pass filter enabled for interrupt generator 2. Default value: 0 (0: filter bypassed; 1: filter enabled)

### Table 33. CTRL1 register description

BOOT	Reboot memory content. Default value: 0 (0: normal mode; 1: reboot memory content)
FIFO_EN	FIFO enable. Default value: 0 (0: FIFO disable; 1: FIFO enable)
FTH_EN	FIFO programmable threshold enable. Default value: 0 (0: disable; 1: enable)
HP_Click	High-pass filter enabled for click function. Default value: 0 (0: filter bypassed; 1: filter enabled)
HPIS1	High-pass filter enabled for interrupt generator 1. Default value: 0 (0: filter bypassed; 1: filter enabled)
HPIS2	High-pass filter enabled for interrupt generator 2. Default value: 0 (0: filter bypassed; 1: filter enabled)

### Table 34. CTRL1 register

ADDR3	ADDR2	ADDR1	BDU	AZEN	AYEN	AXEN
0000	0000	0000	0000	0000	0000	0000

### Table 35. CTRL1 register description

ADDR [3:0]	Acceleration data rate selection. Default value: 0000 (0000: Power-down mode; Others: Refer to Table 36)
BDU	Block data update for acceleration and magnetic data. Default value: 0 (0: continuous update; 1: output registers not updated until MSB and LSB have been read)
AZEN	Acceleration Z-axis enable. Default value: 1 (0: Z-axis disabled; 1: Z-axis enabled)
AYEN	Acceleration Y-axis enable. Default value: 1 (0: Y-axis disabled; 1: Y-axis enabled)
AXEN	Acceleration X-axis enable. Default value: 1 (0: X-axis disabled; 1: X-axis enabled)

28/52

DocID023312 Rev 2

ST

# LSM303 - Driver

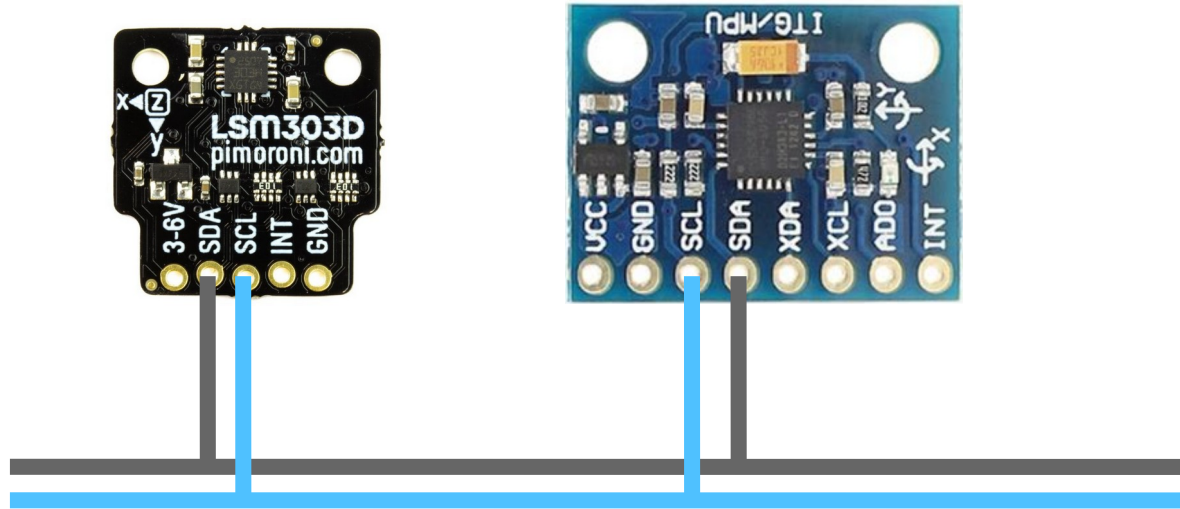
```
use embedded_hal_async::i2c::I2c;

pub struct LSM303D<I2C: I2c> {
    i2c: I2C,
    acc_multiplier: f32,
    mag_divider: f32,
    address: u8,
}

impl<I2C: I2c> LSM303D<I2C> {
    pub fn new(i2c: I2C) -> Self {
        Self {
            i2c,
            mag_divider: 1.0,
            acc_multiplier: 1.0,
            address: ADDRESS,
        }
    }

    pub async fn configure_int1(&mut self, configuration: Int1Configuration) -> Result<(), ()> {
        self.i2c
            .write(self.address, &[Register::Ctrl3 as u8, configuration.into()])
            .await
            .map_err(|_| ())
    }
}
```

# I2C Bus



Who owns I<sup>2</sup>C?

# Sharing sync I<sup>2</sup>C bus

```
static I2C_BUS: StaticCell<
    Mutex<CriticalSectionRawMutex, RefCell<I2c<I2C1, embassy_rp::i2c::Blocking>>>,
> = StaticCell::new();

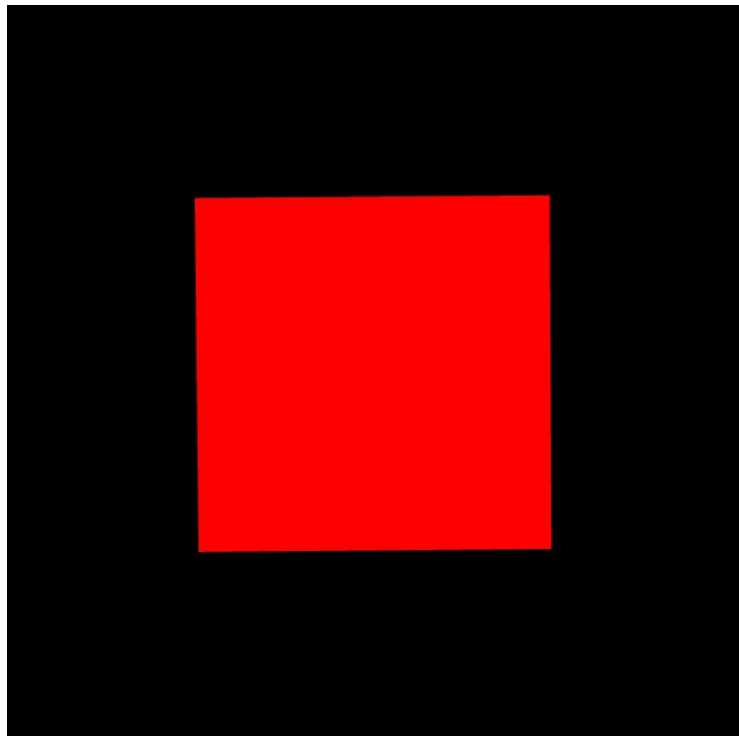
#[embassy_executor::main]
async fn main(_spawner: Spawner) {
    ///... Code
    let i2c = embassy_rp::i2c::I2c::new_blocking(p.I2C1, scl, sda, i2c_config);
    let i2c_bus = Mutex::new(RefCell::new(i2c));
    let i2c_bus = I2C_BUS.init(i2c_bus);
    let i2c_dev1 = I2cDevice::new(i2c_bus);
    let i2c_dev2 = I2cDevice::new(i2c_bus);
    ///... Code
}
```

# Adding magnetometer

```
let mut lsm303 = LSM303D::new(i2c_dev2);
let lsm_config = Configuration::default().configure_magnetometer(
    MagnetometerDataRate::Hz50,
    MagneticSensorMode::ContinuousConversion,
    MagnetometerFullScale::Mag2,
    MagnetometerResolution::High,
);
lsm303.configure(lsm_config).unwrap();
loop {
    ///... Code
    let mag_result = lsm303.read_measurements().unwrap().magnetometer;
    let mag_yaw = libm::atan2f(mag_result.x, mag_result.y);
    ///... Code
}
```



# Results

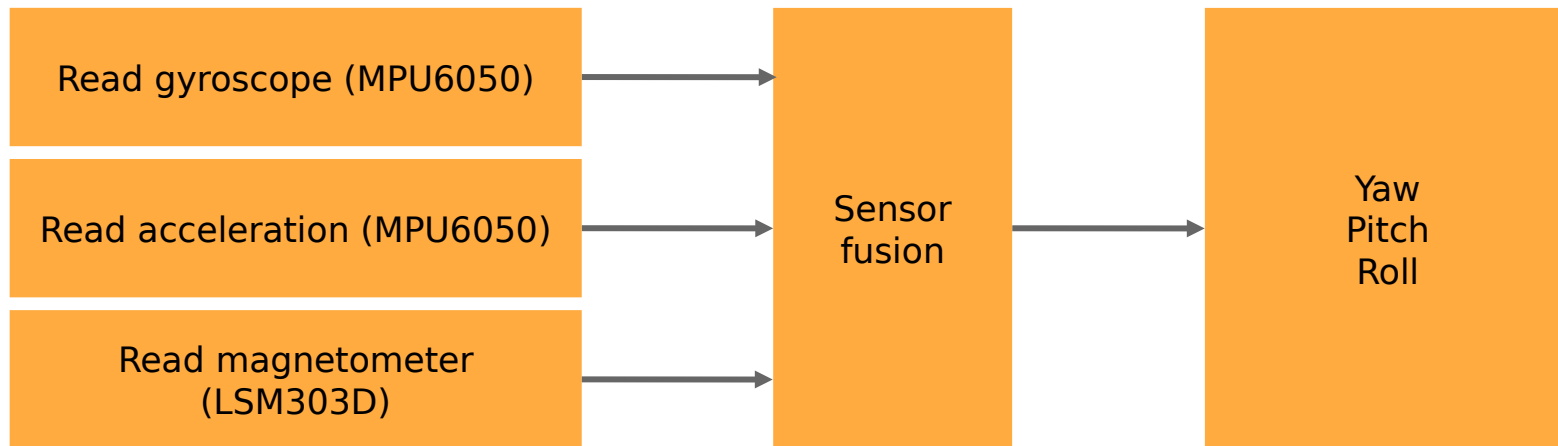


Problems:  
**Electric devices can  
affect measurement!**

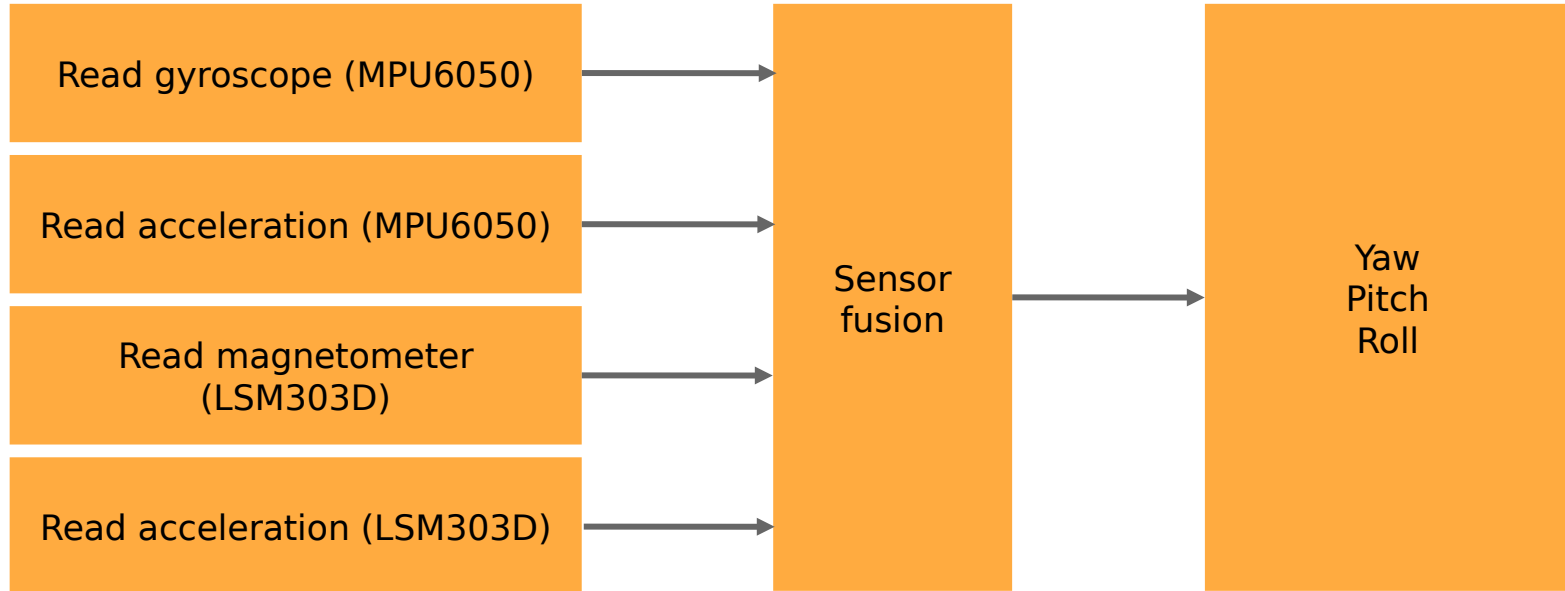
**Mostly used for yaw**

# Sensor fusion

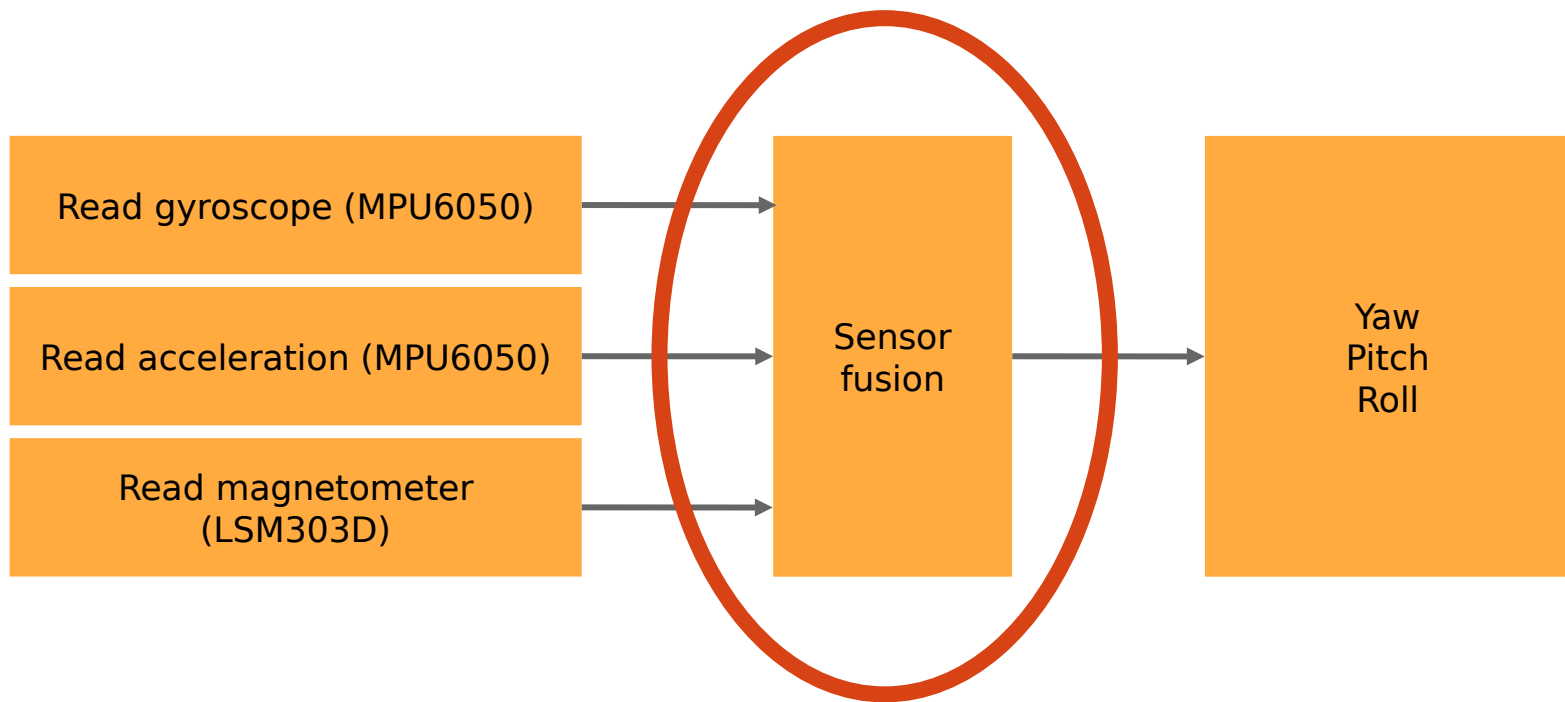
# Sensor fusion



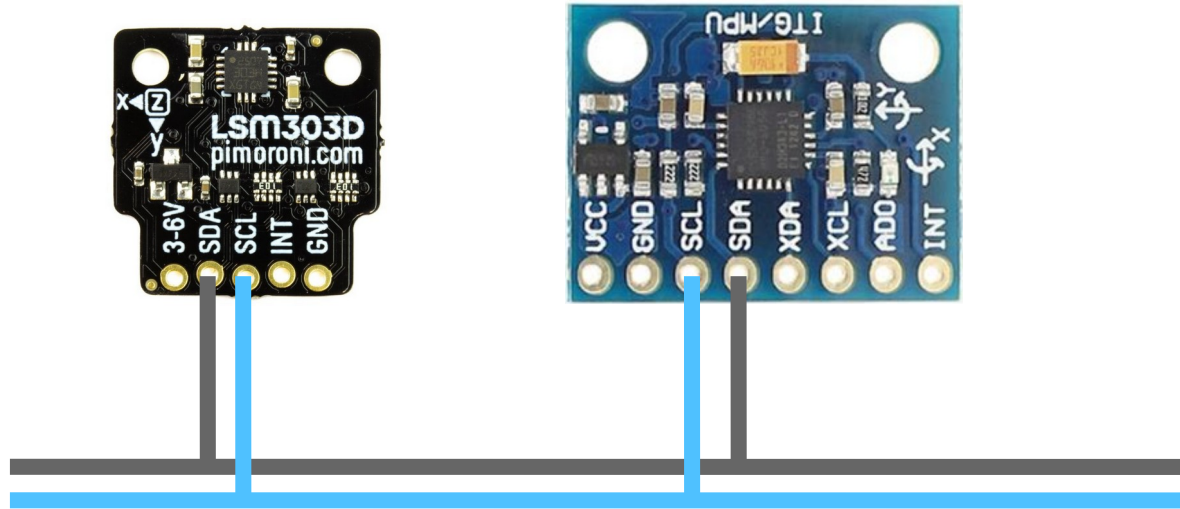
# Sensor fusion



# Sensor fusion

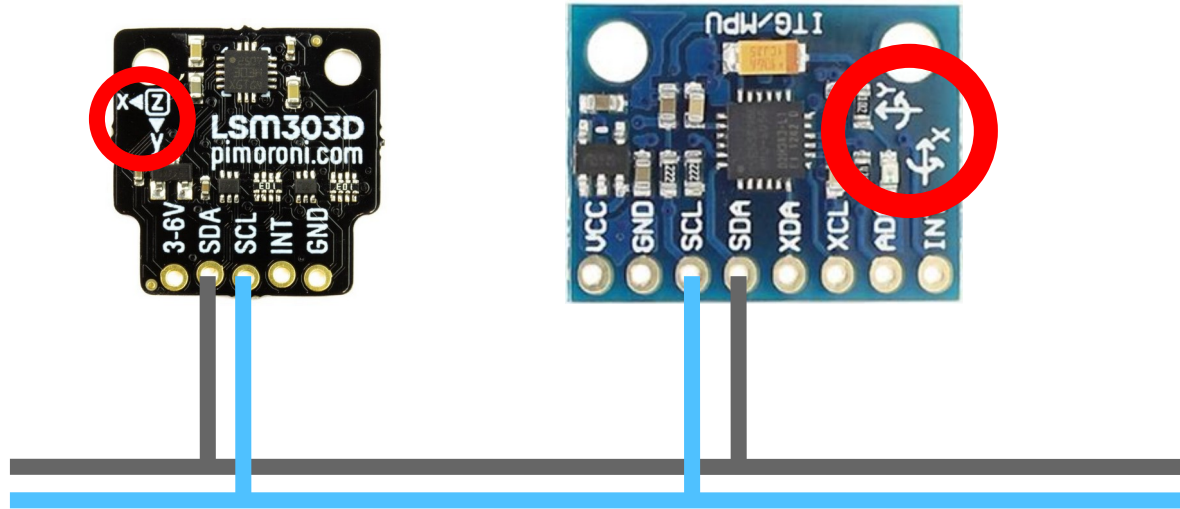


# I2C Bus



Who owns I<sup>2</sup>C?

Check axis!



Double check this, trust me ;)

# Data Fusion





# Complementary filter

1. Get calculation from gyro
2. Get calculation from accelerometer + magnetometer
3. Combine!

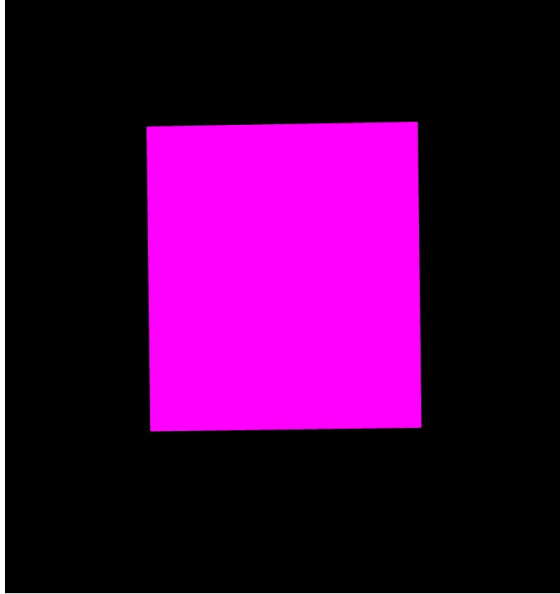
# Code example

```
let timestep = ((cur_time - now).as_micros() as f32) / 1_000_000.0;
now = cur_time;

let time_constant = 0.75;
let a = time_constant / (time_constant + timestep);

yaw = a * (yaw + gyro.z().to_radians() * timestep) + (1.0 - a) * mag_yaw;
pitch = a * (pitch + gyro.x().to_radians() * timestep) + (1.0 - a) * accel_pitch;
roll = a * (roll + gyro.y().to_radians() * timestep) + (1.0 - a) * accel_roll;
```

Something isn't right...

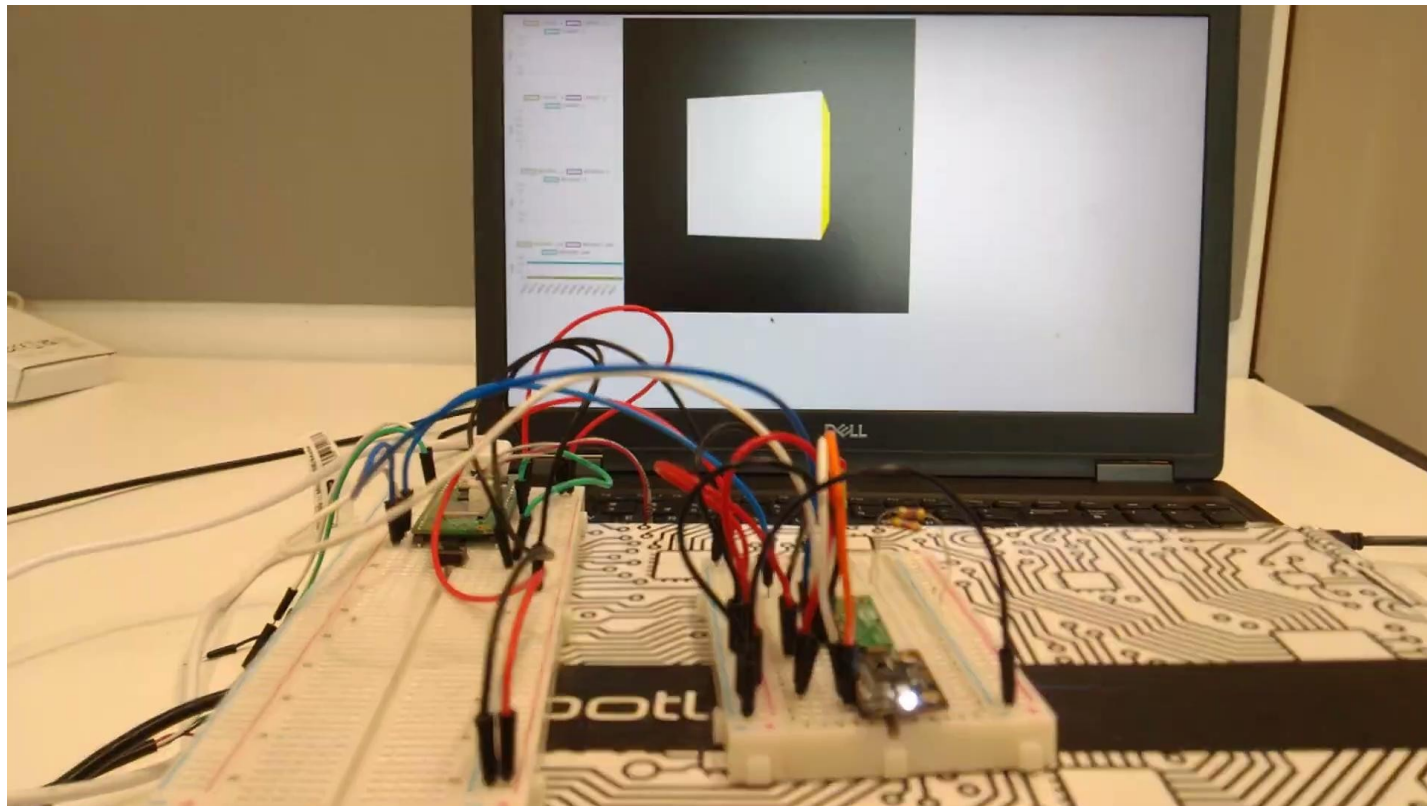


# Tilt compensation

```
fn magnetic_yaw(mag_result: MagnetometerMeasurements, accel_pitch: f32, accel_roll: f32) -> f32 {  
    let cx = mag_result.x * libm::cosf(accel_pitch)  
        + mag_result.y * libm::sinf(accel_roll) * libm::sinf(accel_pitch)  
        + mag_result.z * libm::cosf(accel_roll) * libm::sinf(accel_pitch);  
    let cy = mag_result.y * libm::cosf(accel_roll) + mag_result.z * libm::sinf(accel_roll);  
    libm::atan2f(-cy, cx)  
}
```

# Demo!

# Demo



Is it end of a  
ride?

Or just a beginning?

Let's explore more!

---

How far we are from real world application?

Very far...



## What to learn next?

- Handle negatives angles
- Complementary filter, while it's working OKish, I might want to check other filters (Kalman e.g.)
- Math - Could use some matrices instead of derived equations. Maybe go with nalgebra crate?
- Send binary data? Send stream in chunks (lower resolution in presenting data)?
- Replace RP2040 with something with FPU?
- Improve connections between components (design own hat for pico?)

# Questions

Ok(()))