Master's thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Inertial measurement unit modeling

David Česenek

Supervisor: Ing. Jan Chudoba
May 2019

CTU CZECH TECHNICAL UNIVERSITY IN PRAGUE

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Česenek David**	Personal ID number:	**420253**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Control Engineering**		
Study program:	**Cybernetics and Robotics**		
Branch of study:	**Cybernetics and Robotics**		

II. Master’s thesis details

Master’s thesis title in English:

Inertial measurement unit modelling

Master’s thesis title in Czech:

Modelování inerciální měřicí jednotky

Guidelines:

Topic of the thesis is a design and an implementation of a simulation environment for the modelling of the real inertial measurement unit (IMU) designated for the service mobile robot navigation support.
- research state-of-the-art simulation environments and assess their features and usability for the specified task
- choose an optimal environment and implement a model of the inertial unit according to parameters provided by the thesis supervisor
- after consultation with the supervisor develop a method of the designed model testing
- validate the implemented model against the real device

Bibliography / sources:

[1] Thrun, S., Burgard, W. and Fox, D., Probabilistic Robotics. Cambridge, Mass: MIT Press. 2005.
[2] Roland Siegwart, Illah Reza Nourbakhsh and Davide Scaramuzza, Introduction to Autonomous Mobile Robots, MIT Press, 2011.

Name and workplace of master’s thesis supervisor:

Ing. Jan Chudoba, Intelligent and Mobile Robotics, CIIRC

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **09.01.2019** Deadline for master's thesis submission: **24.05.2019**

Assignment valid until:
by the end of summer semester 2019/2020

Ing. Jan Chudoba	prof. Ing. Michael Šebek, DrSc.	prof. Ing. Pavel Ripka, CSc.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt	Student’s signature

Acknowledgements

At the first place, I would like to thank my supervisor, Ing. Jan Chudoba, for many useful pieces of advice and unrelenting support during the work on this thesis.

I am grateful for help and support to all my colleagues from the team connected to the research project to which this thesis contributed. It was my honor to cooperate with you. The consultations with experts Ing. Martin Šipoš Ph.D., and doc. Ing. Radislav Šmíd Ph.D. helped me a lot. I thank you for your time and helpfulness.

I appreciate the support of my whole family and friends. My work on this thesis would be much harder without your support and kind words of encouragement. Jesus Christ, thank you for your sacrifice on the cross and being alive right now. In your power, I was able to finish this thesis, and you have literally saved my life.

And the last but not least, I thank the CTU in Prague for being a very good alma mater.

Declaration

Abstract

The inertial measurement unit (IMU) sensors are massively used in mobile service robots to provide orientation estimation. This thesis is concerned with modeling and simulation of IMU sensor in the robotics simulator Gazebo. The main goal of this thesis is to simulate the heading angle output of a real IMU sensor Bosch BNO055 with high fidelity. To enable the IMU model evaluation I designed and implmented a custom IMU simulation framework as a ROS package. This framework approximates the given trajectory with the help of a Gazebo simulation of a robot model with attached IMU sensor model, captures the simulated IMU output and generates data for the comparison concerning provided dataset measured by real BNO055. I used the best currently available IMU plugins to implement two different URDF/SDF IMU models. The simulations demonstrated the functionality of implemented IMU models, but also revealed the fidelity limitations of current IMU plugins in Gazebo, and led to a discussion about possible future improvements.

Keywords: inertial measurement unit, IMU, noise analysis, noise modeling, simulation, Gazebo, ROS, BNO055

Supervisor: Ing. Jan Chudoba

Abstrakt

Inerciální měřící jednotka (IMU) patří mezi základní senzorické vybavení současných mobilních robotů, kde se používá především pro odhadování ohamžité orientace robotu v prostoru. V této práci se komplexně zabývám simulací IMU v robotickém simulátoru Gazebo za účelem co nejvěrnějšího modelování odhadovaného úhlu natočení robotu, který je jedním z přímých výstupů IMU senzoru Bosch BNO055. Pro podporu vyhodnocení kvality IMU modelu vzhledem k reálným datům z BNO055 jsem navrhl a implementoval simulační prostřední v rámci Robotického Operačního Systému (ROS), které aproximuje zadanou trajektorii, uloží data ze simulovaného IMU a vygeneruje podklady pro vyhodnocení kvality IMU modelu vzhledem k reálným datům z BNO055. Na základě nejlepších dostupných IMU pluginů v Gazebu jsem implementoval dva URDF/SDF modely IMU sensoru, jejichž funkčnost byla následně ověřena řadou experimentů v simulátoru. Provedené simulace potvrdily funkčnost modelů a zároveň poukázaly na limity realističnosti současných pluginů v Gazebu a nastínily možnosti dalšího vývoje pro zvýšení věrnosti simulací IMU.

Klíčová slova: inerciální měřící jednotka, IMU, analýza šumů, modelování šumů, simulace, Gazebo, ROS, BNO055

Překlad názvu: Modelování inerciální měřící jednotky

Contents

Introduction

In the last decade, the importance of mobile service robots increased rapidly. Formerly expensive products on the edge of state-of-the-art inventions and sci-fi became widely available to the general public. Nowadays, various types of mobile robots such as drones, robotic lawn mowers or vacuum cleaners start to be involved in our lives on an everyday basis.

An example of a contemporary mobile service robot is depicted in the figure 1. This robotic vacuum cleaner implements smart navigation algorithm, a few cleaning modes, many safety features and can be easily controlled from a smartphone.

Figure 1: Visualization of the operation of robotic vacuum cleaner DUORO Xcontrol profi, [1]

One of the essential skills every autonomous mobile service robot must have is the ability to localize and navigate itself. Without enough precise localization, the robot cannot act autonomously and fulfill tasks successfully. To accomplish the localization robots perceive their surroundings using various types of sensors. In the last years, especially satellite navigation systems are being used for that purpose. However, in many situations, the satellite navigation signal is not available, and at this point, the deployment of other sensors becomes essential.

Among them, the family of so-called "inertial sensors" plays a vital role. An inertial sensor utilizes the measurement of the forces acting on its body. Based on the type of the sensor it can convert the forces to accelerations or angular velocities, and as a consequence, the instantaneous position, or orientation can be estimated. The *inertial measurement unit* (IMU) is a sensor from this category (see chapter 1 for details). It can estimate both the movements and orientation in 3D space. Modern technologies enabled the production of very low-cost IMU sensors, and thus they are commonly found not only in the mobile robots but literally in almost every modern "smart" device starting from smart watches and ending with a common car.

The increasing demand for better and better mobile service robots places high requirements not only on navigation-related sensors but also on the speed of the robot development process itself.

In this area, the mobile robotics industry hugely benefits from the advent of computer simulations and specialized robotics simulators. The rapid growth of computing power in the last two decades enabled the performing of computer simulations with high physical fidelity, which is used massively in the development process.

In this thesis, I connect the worlds of computer simulation and real hardware. In particular, I cover in detail the whole process of IMU modeling and subsequent simulation in the Gazebo robotic simulator.

The primary aim of this thesis is to create a simulation environment for IMU testing together with an accurate model of a one particular low-cost IMU sensor Bosch BNO055 with a strong emphasis to heading angle estimation. The IMU model in robotic simulator should then serve for the rapid development purposes of the future generations of mobile service robots, where the BNO055 IMU is being considered as a primary source of heading angle estimation.

Apart from being a master's thesis, at the same time, this work is also a part of a real research-oriented international project in close cooperation with a high-tech industrial company.

Thesis outline

In the first chapter 1, I describe theoretically the IMU sensors. Their working principles, types, technical parameters together with error causes and related issues are mentioned. The second chapter 2 is devoted to the simulation. It provides an overview of state-of-the-art robotic simulators and argues for the selection of the Gazebo robotic simulator. Chapter 3 is the heart of this thesis. I describe there the properties of the mobile robot, and the nature of the provided datasets measured in experiments with the real robot. I also discuss the parameters of BNO055 and perform the Allan variance noise analysis. Subsequently, I introduce a design and implementation of a custom simulation framework to enable the IMU simulation in Gazebo/ROS environment. Finally, in the last part of this chapter, the modeling of the IMU sensor itself is described.

The experimental results from the simulations are summarized in the chapter 4. Finally, the conclusion is to be found in the last chapter 5.

Chapter 1
Inertial measurement unit

The *inertial measurement unit* (throughout this thesis referred to as 'IMU') is an electronic device, a sensor with own control unit and I/O interface, that usually comprises of several gyroscopes, accelerometers and sometimes also other additional sensors to measure its rotational and translational movements. Flowchart 1.1 depicts the general structure of the basic IMU as presented in [2]. To enable measurements of any motion in the three-dimensional space usually three accelerometers and three gyroscopes with mutually perpendicular axes are used. Thus, the output of a typical IMU sensor comprises accelerations and angular velocities. By integrating these values over time the current position and orientation of the moving IMU can be computed, and its trajectory estimation acquired.

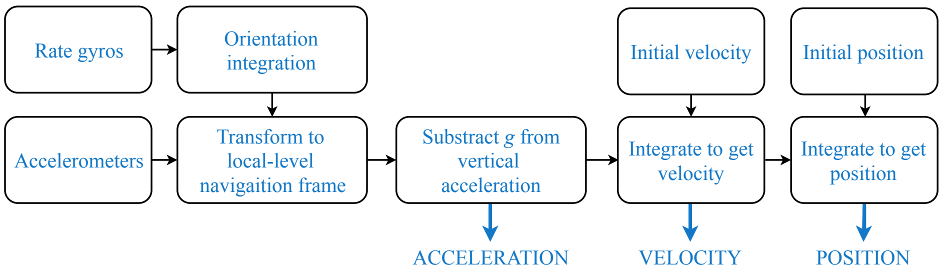

Rate gyros
Orientation integration
Initial velocity
Initial position
Accelerometers
Transform to local-level navigaition frame
Substract g from vertical acceleration
Integrate to get velocity
Integrate to get position
ACCELERATION
VELOCITY
POSITION

Figure 1.1: A simplified IMU block diagram

This chapter describes IMU in the context of mobile robotics from different viewpoints. It was taken into consideration that the main scope of this thesis lays in the simulation of an existing and consumer-oriented IMU attached to a service mobile robot. Thus, the following sections offer an overview of existing solutions and approaches rather than deeper look into theory under IMU's surface.

In the first section 1.1, I define the term *inertial navigation*. Subsequently, in sections 1.2, 1.3, and 1.4, I describe the fundamental physical principles of IMU component sensors. A bit more attention is paid to MEMS accelerometers and gyroscopes that constitute the cornerstone of researched types of IMUs.

Section 1.5 mentions the basic ideas behind the multisensor data fusion used to get the IMU output. Possible error causes are described closely together with most important technical parameters and Allan variance noise analysis in the sections 1.6 and 1.7.

Finally, the calibration of IMU sensor is mentioned in 1.8.

1.1 Inertial navigation

The origin, the evolution, and also the application of IMU sensors are closely related to concepts of dead-reckoning and inertial navigation.

Dead (deduced) reckoning stands for the process, which estimates the current position based on the knowledge of the previous pose, instantaneous velocity, course, and time duration.

In the *inertial navigation*, the dead reckoning approach is used in conjunction with the IMU sensor. The inertial navigation system (INS) uses IMU output to continuously update an estimation of a current position without any external reference. Running INS is subject to ever-increasing error since any arbitrarily small measurement error integrates over time which results in wrong position estimation. Despite this disadvantage, INSs are widely used in aviation, marine, space, automotive, robotics or military applications. INS either works as an additional sensor enhancing the behavior when external navigation systems such as GNSS[1] cannot provide reliable data, or as the primary navigation sensor when INS accuracy is sufficient for a specific application (e.g., some guided missiles, undersea applications). Inertial navigation is of importance for mobile robotics because mobile service robots often work in GNSS-denied environments[2], or the robot is not equipped with an external navigation system by intention due to production price constraints.

There are basically two types of INS [3].

Gyrostabilized INSs are historically older. In this case, the inertial sensors are mounted on the platform that is mechanically isolated and stabilized from moving object itself. This solution requires heavy mechanical construction, but modern devices of this type offer high accuracy.

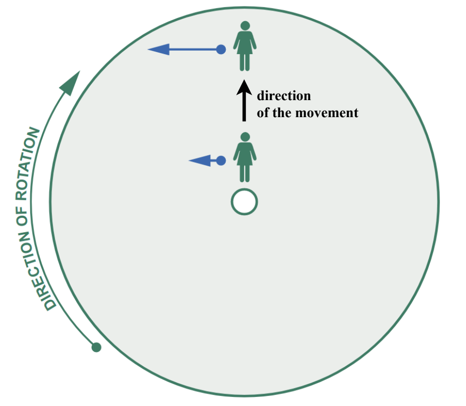
The second type is called *strapdown INS*. The IMU sensor is firmly attached directly to the sensed object and undergoes the same movements as the object. This approach leads to smaller, lighter and more durable INS, which in contrary requires higher computation power to compute the desired physical quantities. In this thesis, I am dealing with this type INS.

1.2 Gyroscope

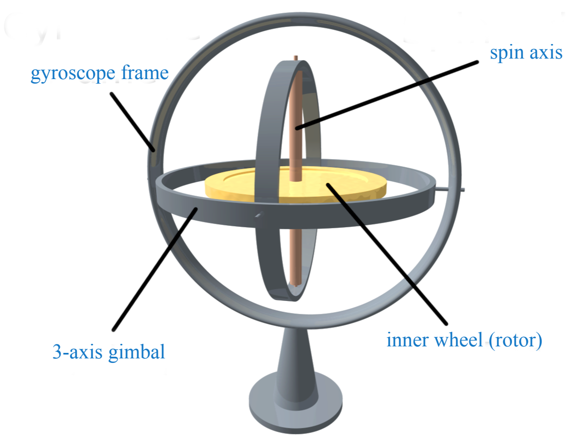
A *gyroscope* is a sensor measuring orientation and angular velocities.
In general, any gyroscope exploits the fact that the rotating frame is a non-inertial reference frame [2]. When an object rotates, there are observable deviations from the expected behavior that would satisfy all Newton's laws. Based on the measured differences and the knowledge of the transformation

[1]Global navigation Satellite Sytem, such as well known GPS

[2]Typically indoor environment, where the GNSS navigation is not usable



(a) : Coriolis effect, blue arrows represent acceleration $\mathbf{a}_c$,[4]



(b) : Classic mechanical gyroscope, [5]

Figure 1.2: Physical principles used in gyroscopic sensors

between inertial and non-inertial reference frames the rotation of the object itself can be computed.

1.2.1 Mechanical gyroscope

Mechanical gyroscopes are historically oldest. Their working principle and structure shows figure 1.2b. A rapidly spinning inner wheel (rotor) mounted within a three-axis gimbal tends to maintain its orientation regardless of the change in the outer frame orientation due to angular moment conservation.

This principle demonstrated Walter R. Johnson already in 1832 when he constructed so-called rotascope [6], but mechanical gyroscope was named and popularized by Léon Foucault in 1852 who used it to prove earth's rotation. The advent of electrical engines enabled permanent rotor rotations and opened the way for practical applications of mechanical gyroscopes, like, e.g. gyrocompass[3]. However, multiple gyroscopes in an arrangement similar to present IMUs were developed for the first time probably later in 1942 in Germany during the WWII as a part of the navigation device for V-2 missiles. They used a pair of mechanical gyroscopes, each with two degrees of freedom, along with one mechanical accelerometer. After the war, a massive development of mechanical gyroscopes for inertial navigation systems continued in conjunction with guided missiles [7].

Although mechanical gyroscopes are not being used in the field of mobile robotics because of their heaviness, complexity, high costs, and energy consumption, they are worth to mention because they prepared the ground for more sophisticated gyroscopic sensors and for expansion of inertial navigation itself.

[3] A gyrocompass is an application of the mechanical gyroscope used to determine the north direction. It was developed in the beginning of 20th century for nautical navigation [7].

1.2.2 Optical gyroscope

Optical gyroscopes avail themself of the Sagnac effect[4]. Assume a ring made of two parallel optical fibers with two light sources, as depicted in figure 1.3. If the ring is not rotating, it takes to light the same time to travel through the optical fibers. When the ring rotates with rotation rate $\Omega \quad [rad \cdot s^{-1}]$, a measurable difference $\Delta l_v \quad [m]$ occurs between trajectories the light takes in opposite directions.

There are two types of optical gyroscopes. A *fiber optic gyroscope* (FOG) uses a coil from optical fiber and laser as a source of light. A phase difference $\Delta\Phi_r \quad [rad]$ is measured, which is proportional to the rotation rate Ω. Using the special theory of relativity equation 1.1 can be derived [8].

$$\Delta\Phi_r = \frac{2\pi \Delta l_v D}{c\lambda}\Omega \tag{1.1}$$

where $c \quad [m \cdot s^{-1}]$ stands for the speed of light in vacuum, $\lambda \quad [m]$ is a wave-length of used light and $D \quad [m]$ is a diameter of the fiber loop.

Instead of fiber optic loop, *a ring laser gyroscope* (RLG) utilizes a set of mirrors or prisms which forms a loop from laser beams. RLG sensor maintains two laser beams in opposite directions in such a way that standing waves have the same number of nodes. When the RLG sensor rotates, the resonant frequencies start to differ between each other, which is measured.

Compared to the mechanical gyroscope, both FOG and RLG gyroscope evince better properties and precision. Because of the absence of moving parts optical gyroscopes are also lighter and more durable which predetermines them for critical aircraft, space, and military applications. However, there are almost no applications of optical gyroscopes in consumer-oriented mobile service robots because of very high costs and a big chassis compare to widely used MEMS gyroscopes.

1.2.3 MEMS gyroscopes

In the field of mobile robotics and consumer products, the most used gyroscopic sensors are *MEMS gyroscopes.*

The abbreviation MEMS stands for the term "Micro-Electro-Mechanical System," which refers to the advanced technology used for manufacturing microscopic devices containing moving parts. Sensors made with MEMS technology can be very tiny (typically only a few millimeters big) although they usually contain auxiliary circuits on the same chip too. Since silicon is the primary material MEMS sensors are made from and MEMS technology is already well established, the MEMS sensors price is usually very affordable in comparison to other sensor types [10]. The details of used fabrication technologies are commonly not available for end customers, and thus they lie

[4]"*Counterpropagating light waves take slightly different times to traverse a loop rotating in inertial space.*"[8]

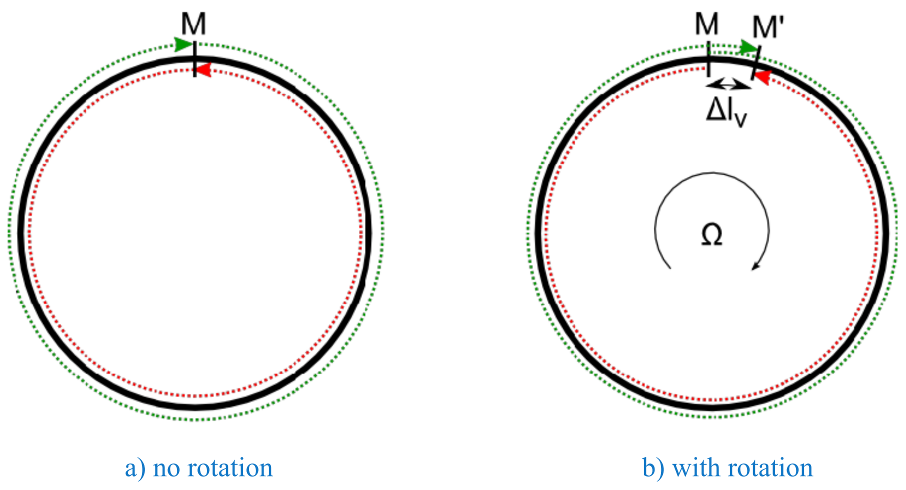


Figure 1.3: Ilustration of Sagnac effect, [9]

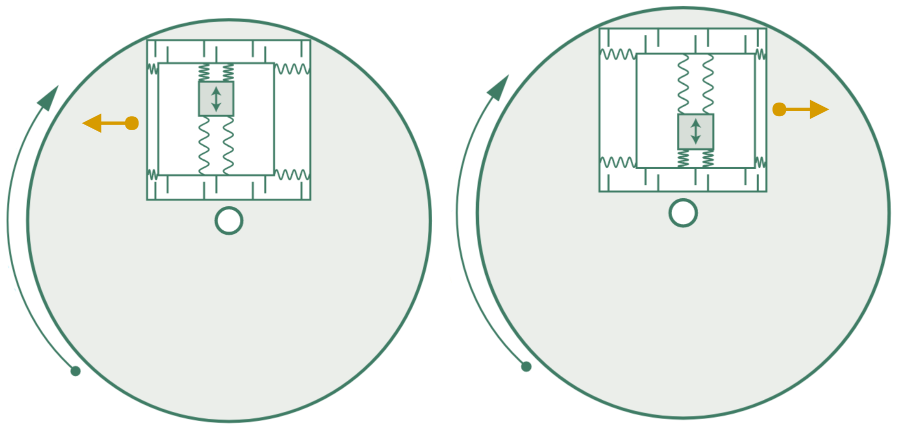


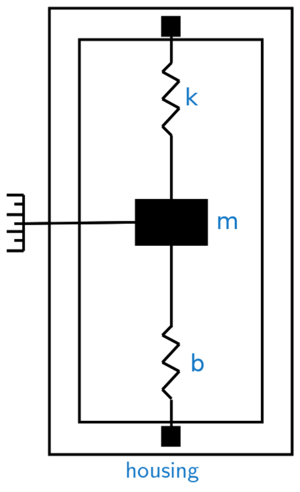
Figure 1.4: Application of Coriolis effect in MEMS gyroscopes. The orange arrows indicate the force applied to the structure due to resonating mass, [4]

not in the scope of this thesis. A brief overview of the manufacturing procedures provides author [11], and further details can be found for example in [12].

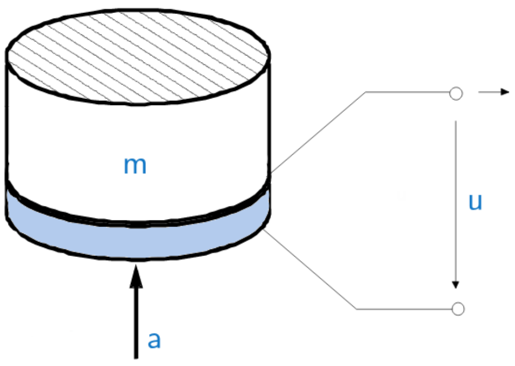
All MEMS gyroscopes exploit somehow the Coriolis effect. Figure 1.2a shows its principle in a simplified way. The person in the picture wants to move from the center to the edge of the disc. When she moves farther from the center her tangential speed relative to the ground, depicted with grater blue arrow, increases. Coriolis acceleration describes the increasing of this tangential speed during her movement, which is caused by a rotating disk - her reference frame. Any moving object within a rotating reference frame undergoes Coriolis effect. The Coriolis effect can be formaly expressed as the equation 1.2

$$\mathbf{F}_c = m\mathbf{a_c} = 2m\omega \times \mathbf{v}, \tag{1.2}$$

where m $[kg]$ is a mass of moving object, ω $[rad \cdot s^{-1}]$ is an angular rate of a rotating refence frame, and $\mathbf{v} = \omega \times \mathbf{r}$ is the instantaneous speed of moving



(a) : mechanical



(b) : piezoeletrical

Figure 1.5: Accelerometers working principles, based on pictures from [14],[15]

object in the distance $\mathbf{r} \quad [m]$ from the rotation center, and $\mathbf{a}_c \quad [m \cdot s^2]$ is so-called Coriolis acceleration.

The MEMS gyroscopes use several techniques to achieve nonstop vibrations of some tangible structure (inertial mass) and to measure angular rate. Among the frequently used structures belongs[2]:

- *tuning fork* - a fork-like structure comprises two inertial masses that are electrostatically induced to vibrate in one axis. When sensor rotates, the fork vibrates out of its plane, which is sensed. Affordable MEMS gyros usually use this working principle or its variants (e.g., "butterfly" structure [11]).
- *vibrating ring* - instead of a fork-like structure, a ring resonates within an elastic mounting. Under the rotation of the sensor, the stationary wave changes which can be measured.
- *hemispherical resonator* - a very precise resonator in the shape of wine glass enables manufacturing of small gyroscopes with superior properties, but the high costs preclude usage in the field of customer-oriented mobile robotics[13].

1.3 Accelerometer

An *accelerometer* is a sensor measuring the acceleration, or more precisely, it estimates the acceleration by measuring the impacts of the inertial forces. Therefore an accelerometer can also be used to measure vibrations or an orientation. There are several main physical principles employed in accelerometers.

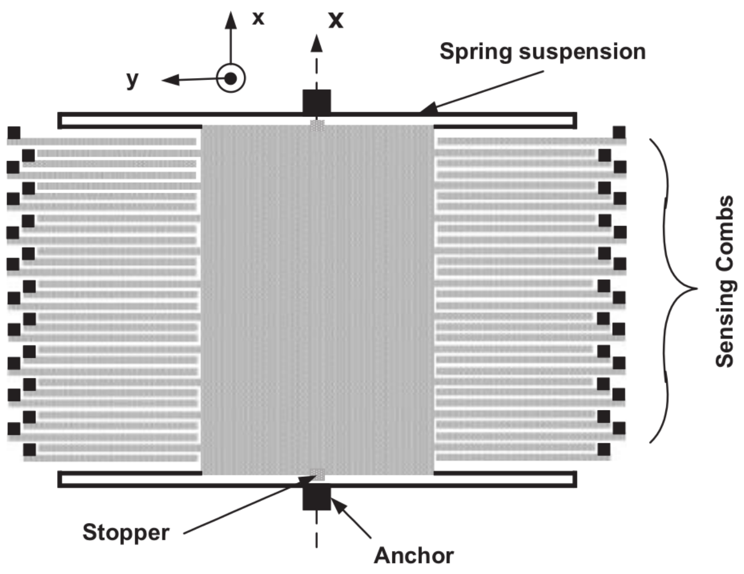


Figure 1.6: The layout of the mechanical MEMS accelerometer[12].

1.3.1 Mechanical accelerometer

The basic principle of the *mechanical accelerometer* depicts the picture 1.5a. The proof mass is elastically attached to the sensor housing. According to second Newton's law, a non-zero acceleration **a** occurs when a force **F** is applied to the sensor's body. If the acceleration **a** occurs in the sensitive axis, the sensor measures the deflection x of the proof mass m and estimates the value of **a**.

Picture 1.6 shows the layout of mechanical accelerometer realized in MEMS technology. Mostly a capacitive transducer converts the mass deflection to the change of capacity. Among its advantages belong the insensitivity to the temperature and a very low drift. The structure of a "comb capacitor" is used to increase the total measured capacity change.

Let us suppose that the proof mass m can move only in the direction of axis x. For the corresponding value of accelaration $|\mathrm{a}_{\boldsymbol{x}}| \equiv \ddot{x}$ then holds the equation:

$$m\ddot{x} + b\dot{x} + kx = ma_x + N_B \tag{1.3}$$

where b $[n \cdot s \cdot m^{-1}]$ is a damping coefficient, k $[N \cdot m^{-1}]$ is a spring constant and N_B stands for the Brownian noise[5] [12]. The mechanical MEMS accelerometers belong to the most frequently used accelerometers in IMU. To their main advantages belong the cheap manufacturing, good low-frequency sensitivity, and the fact, that they also measure the gravity which can be used to determine the orientation. However, their big issue is the high sensitivity to vibrations, especially to hard mechanical shock that can cause a loss of calibration [16].

[5]Stochastic part of the damping force, caused by particle collisions of the ambient medium with the moving mass m.

1.3.2 Piezoelectric accelerometer

Piezoelectric accelerometer exploits the piezoelectric effect present in some materials such as quartz or some types of ceramics. When a mechanical strain applies to these materials, they generate a charge proportional to the applied force. Therefore, the piezoelectric effect can be employed to fabricate accelerometers without moving parts as depicted in the picture 1.5b. Piezoelectric accelerometers have many advantages, such as no wear, extremely wide dynamic range as well as a low output noise or the ability to work without a power source. On the other hand, this type of accelerometers cannot measure static acceleration [15].

1.3.3 Piezoresistive accelerometer

Similarly to the previous accelerometer type, the piezo properties of some materials are used, but in this case the electrical resistance of the materials changes when stress applied. However, the lower sensitivity predetermines them for the shock measurements rather than for typical IMU applications [17].

1.3.4 Thermal accelerometer

The *thermal accelerometers* are a relatively new category of MEMS accelerometers that use the MEMSIC [6]. Instead of proof mass, they use a heated gas and sense its position with temperature sensors. This approach leads to very reliable sensors with no wear and high resistance to vibrations and excellent shock survivability, but at the cost of higher power consumption needed the heat the gas[16].

1.4 Supplementary sensors

In theory, the triplets of accelerometers and gyros suitably oriented in three perpendicular axes should assure the full 6 degrees of freedom (DOF) position and orientation information.

Even so, sometimes one can also find additional sensors such as an air pressure sensor, temperature sensor or magnetometer implemented aside from accelerometers and gyros. Multiple sensors leading into estimations of the same physical quantity provide desired redundancy, which increases the total value of the sensor. The details of these sensors are described in the book [18].

[6]Advanced technology that integrates the MEMS sensor with other circuits on a one single CMOS wafer.

1.5 Multisensor data fusion

When IMU is used in mobile robotics, the primary aim is to get the best possible estimation of instantaneous position and orientation. Outputs from multiple component sensors can be processed and combined to provide better-resulting output.

There are various techniques used to fuse the sensor measurements. Most of them are somehow related to Bayes rule, which is a handy tool to combine sensor outputs with prior beliefs about the state of the world and which regards the pros and cons of the component sensors. Popular approaches include Kalman filter, or Monte Carlo methods such as Particle filter [19].

In the case of IMU, the accelerometer, gyroscope (and magnetometer) are fused together in order to get the precise orientation. The generic data fusion algorithms can be also used to incorporate external sensors such as GPS in the total position information.

The data fusion can be performed as a part of the post-processing in the computer, but a better IMU typically contains a digital motion processor (DMP) with built-in data fusion algorithms to get improved orientation information while the position relies only on the accelerometer.

The detailed description of the multisensor data fusion of the 9 DOF IMU can be found in [20].

1.6 Noises, error causes and technical parameters

As every sensor in the real world, also every IMU sensor is a subject to various types of errors leading to imperfect, noisy sensor's output. The IMU errors can be roughly broken down into two categories: stochastical and deterministic errors.

Stochastical errors are commonly called *noises* and can be formally described as random processes - signals of stochastic nature. Since there is no deterministic formula defining them, they are described statistically. In the time-domain, the noise can be characterized for example by its mean value, autocorrelation, autocovariance or probability distribution function. However, in engineering practice, the frequency domain description is mostly used for its good mathematical properties. The power *spectral density function* (PSD) describes how the distribution of the signal's power is depending on the frequency. It is used as one of the main tools to describe the random process. Assuming that the noises presented in the output from IMU sensors are weak-sense stationarity random processes (roughly speaking, their statistical properties are not changing over time), and the IMU's output is a discrete signal, according to Wiener-Khinchin theorem the power spectral density function is equal to the Fourier transform of the corresponding autocorrelation function [21]. PSD function $S(\omega)$ can be then defined as

$$S(\omega) = \sum_{k=-\infty}^{\infty} r_{xx}(k) \cdot e^{-j\omega k} \tag{1.4}$$

where ω is the angular frequency, and

$$r_{xx}(k) = \mathrm{E}\left[x(n)\overline{x(n-k)}\right] \tag{1.5}$$

where r_{xx} is the autocorrelation function of the discrete random process x and $E\,[..]$ stands for expected value.

Some of these noises can be directly addressed by different statistical techniques like signal filtering or noise modeling (see section 1.7).

Deterministic errors can be caused by systematic errors like manufacturing imperfections and are generally easier to describe and suppress. They can be typically compensated by a precise sensor calibration (see section 1.8).

The overall sensor's noise resitance truly determines the resulting quality and performance of the sensor. To describe it, manufacturers have introduced many technical parameters. Since the IMU is more or less always a combination of accelerometers and gyroscopes, the IMU parameters are rather a combination of parameters of each partial sensor. Unfortunately, the terminology in this field is very ambiguous. Especially the low-cost IMU sensors, mainly used for customer oriented service mobile robots, have often poor documentation and important parameters are missing.

Here I introduce the essential error causes concurrently with the most often used IMU parameters describing them. More detailed description of IMU errors is to be found in [14], [22].

1.6.1 Bias

Ideally, a sensor at rest (with no input at all) would always have zero output. But in reality, there is often a non-zero offset called bias in signal output. The bias can be roughly split into two components.

$$\textit{bias}=\textit{initial offset} + \textit{bias drift} \tag{1.6}$$

While rather fixed initial offset (described by *bias repeatability* parameters such as *Turn-on to Turn-on Bias*) is caused more by manufacturing issues and can be removed by initial precise calibration, the *bias instability* is a big issue. These bias changes can be caused for example by temperature drift during the operation time, power supply voltage drift or by mechanical stress on the system.

The parameter *In-run bias stability* (or *bias instability*) is the theoretical accuracy of bias estimation when running continuously under fixed operating conditions. Formally, it is defined as a minimum of Allan Variance graph (see section 1.7). It is one of the best indicators of the gyroscope or accelerometer

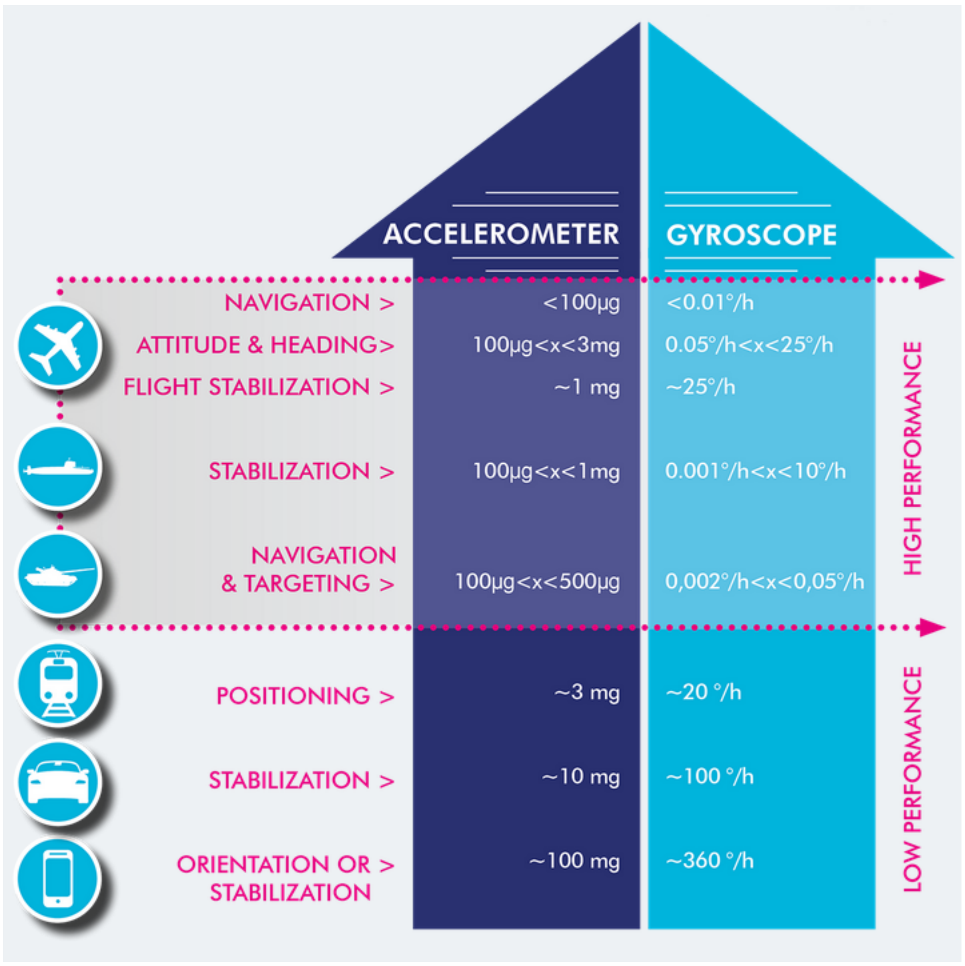


Figure 1.7: IMU performence requirements (in-run bias stability) per application type, source:[24]

accuracy and performance. The IMU performance requirements from the viewpoint of this parameter are nicely summarized in the picture 1.7. However, for the cheaper sensors, this parameter is not usually defined by the manufacturer and has to be found out experimentally.

Rate random walk parameter describes the low-frequency bias fluctuations in long-term scale, sometimes it is also called as a flicker noise. It can be caused by temperature effects or by other unclear reasons [23]. Typical units are $[m/s^2/s/\sqrt{Hz}]$ and $[rad/s/s/\sqrt{Hz}]$ for accelerometers and gyroscopes respectively. Low rate random walk coefficients are important for the long-term applicability of IMU and low bias drift.

Bias drift and its fluctuations are hard to compensate and usually, they are modeled as a random walk.

1.6.2 White noise

Commonly as white noise is considered a random signal with zero mean and equal intensity at all frequencies (i.e., with a constant PSD function). In the context of inertial sensors, it stands rather for an added random signal with zero mean value over a long period, with a correlation time shorter than the sampling rate. Thermo-mechanical events on the microscopic level mainly cause these signal fluctuations [25]. It is usually defined as *noise density* under specific conditions.

When the angular velocities or accelerations are integrated to estimate the orientation angles and the velocity, the white noise starts to manifest itself as a random walk process. Then it make sense to describe the white noise density as parameters *angle random walk* (ARW)$[rad/s/\sqrt{Hz}]$ in case of gyroscopes or analogically *velocity random walk* (VRW)$[m/s^2/\sqrt{Hz}]$ in case of accelerometers.

Especially for the measurement of signals with low amplitude, the white noise can contribute to the total measurement error. Thanks to its zero mean, signal filtering can well mitigate the unwanted effects of white noise in short time horizon, however, in a long-term application, the resulting angular (velocity) random walk together with other errors cause ever-increasing error preventing the IMU sensors from being used separately for a long time.

1.6.3 Sensitivity

In general, the sensor's sensitivity stands for the rate of conversion between the output and input. The imperfection of scale factor is usually called *scale factor error* or *scale factor nonlinearity* and is expressed as ppm[7].

1.6.4 Range

Input range parameter stands for the maximal change in the input signal (expressed as change of measured physical quantity) the sensor can capture, whereas the *full range* parameter is the characterizes which motion can be measured. It is worth to note that there is usually a tradeoff between the full range and a resolution of the output signal. A high value of full range parameters leads to lesser resolution and vice versa.

1.6.5 Bandwidth

Bandwidth describes the behavior in a frequency domain. It shows how the amplitude of the sensor's output depends on the frequency of its input signal. In the datasheet, there is a bandwidth usually depicted as a frequency response chart, or as a tolerance band regarding a reference frequency (given as a percentage, or in dB). For the mobile robot's localization, especially the low frequencies are of interest and the ability to measure the static acceleration for accelerometers is desired. Conversely, it is important to

[7]parts per milion

avoid excessively high frequencies that can cause mechanical resonance of the inertial sensor and consequently even harm the inertial sensor. For that reason, high-frequency vibrations pose one of the threats to MEMS inertial sensors and can negatively influence the measurement accuracy.

1.6.6 Systematic errors

To get meaningful results, the *alignment* of all axis of the internal accelerometers and gyroscopes is important. The measure of unwanted sensitivity in other axis is usually listed in the datasheet in % as a parameter *Cross-axis sensitivity* and can be well suppressed by calibration.

The user should also pay attention to the attachement of the sensor itself to the board, and subseqently, to the alignment of the board containing the sensor to the robot's reference frame. Every physical connection introduces a coordinate transformation and misalignment on any level can result in erroneous output.

1.6.7 Other parameters

When choosing the right IMU sensor, one may also pay attention to implementation details. There is plenty of different package sizes, from tiny "one-chip" solutions to the pre-prepared development boards or shields with separately placed components. Also, the IMU power consumption may vary among different devices, even though the power consumption of a MEMS IMU sensor is typically very low and negligible compared to the power consumption robot's actuators.

IMU sensors provide several ways to communicate with other devices. There are IMUs with analog and digital outputs available. The standard solution comprises serial buses such as SPI or I2C with various output sampling rates and different precisions of internal AD converters. Sometimes there are also additional pins for direct synchronization of external sensors available too.

Last but not least, the IMU sensors also differ in included special features. Better IMU sensors are well equipped smart sensors including programmable microprocessor, buffers, temperature compensation, self-checks, auto-calibration or an intern DSP chip as well.

1.7 Allan variance noise analysis

Probably the most frequently used IMU noise identification approach is called *Allan variance analysis*. This method was developed originally by David W. Allan in 1966 to analyze frequency stability of atomic clocks and oscillators [26], but over the years it became virtually the standard method also for IMU sensor noise analysis, which is also defined in standard IEEE Std 952-1997 [27].

In general, Allan variance is a time-domain analysis that can indetify basic IMU noise terms such as quantization noise, white noise or bias instability by

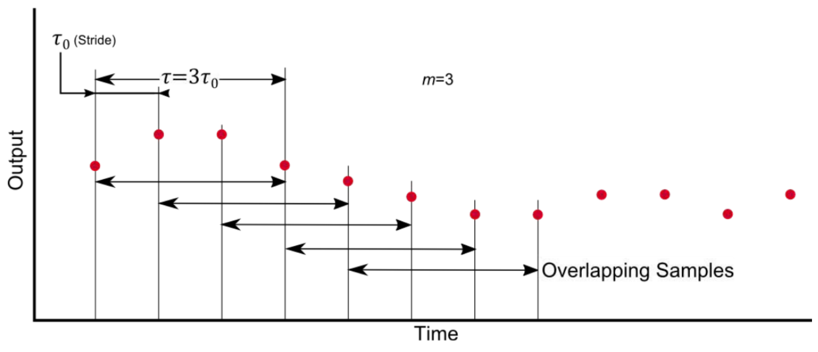


Figure 1.8: Example of cluster forming in Allan variance "overlapping" method, [28]

analysing the Allan deviation plotted as a function of a cluster size in the log scale.

For the Allan variance analysis description, let us assume an output of gyroscope, the procedure for the accelerometer is analogical. Following theoretical description is based mainly on the paper [29], and the IEEE standard Std 952-1997 [28]. A more hands-on approach is given for example in the Freescale tutorial [28].

Let $\Omega(t)$ be the discrete angular velocity signal measured by stationary gyroscope, that has N samples, sampled with a period τ_0. Let a *cluster* be a group of m consecutive samples with time duration

$$\tau = m\tau_0, \tag{1.7}$$

where m can be arbitrary integer satisfying the condition

$$m < \frac{N-1}{2}. \tag{1.8}$$

For the k-th cluster's average rate (average angular velocity) can be written

$$\overline{\Omega_k}(\tau) = \frac{1}{\tau} \int\limits_{k\tau_0}^{k\tau_0+\tau} \Omega(t)\, dt \tag{1.9}$$

For difference of two consecutive clusters ξ_k then holds

$$\xi_k = \overline{\Omega_{k+1}}(t) - \overline{\Omega_k}(t) \tag{1.10}$$

The set of all consecutive differences ξ_i over all clusters with the same length can be seen as a set of random variables, whose variance can be computed by the formula

$$\sigma^2(\tau) = \frac{1}{2(N-2m)} \sum_{k=1}^{N-2m} [\xi_k]^2 \tag{1.11}$$

$$\sigma^2(\tau) = \frac{1}{2(N-2m)} \sum_{k=1}^{N-2m} \left[\overline{\Omega_{k+1}}(\tau) - \overline{\Omega_k}(\tau)\right]^2 \tag{1.12}$$

which is called Allan variance for a clusters with size $\tau = m\tau_0$ [29]. Thus, the Allan deviation is defined as

$$\sigma(\tau) = \sqrt{\sigma^2(\tau)} = \sqrt{\frac{1}{2(N-2m)} \sum_{k=1}^{N-2m} \left[\overline{\Omega_{k+1}}(\tau) - \overline{\Omega_k}(\tau)\right]^2} \tag{1.13}$$

Alternatively, Allan variance (deviation) can be computed also from the output angle θ.

The Allan variance analysis relys on the fact that there exists a unique relationship between Allan variance $\sigma^2(\tau)$, described by equation 1.12, and the PSD function of the intrinsic random processes (noises) that are present in the analysed signal [28]. Assuming that the rate $\Omega(\tau)$ is a stationary random process, it can be proved that holds the equation

$$\sigma^2(\tau) = 4\int_0^\infty df \cdot S_\Omega(f) \cdot \frac{sin^4\left(\pi f \tau\right)}{(\pi f \tau)^2}, \tag{1.14}$$

where $S_\Omega(f)$ stands for PSD function of random process $\Omega(\tau)$ and f is frequency. If the PSD function $S_\Omega(f)$ of a particular noise is identified, it can be substituted into the equation 1.14 and the corresponding Allan varinace can be computed. The Allan variance analysis exploits this relationship in the opposite direction. Applying equation 1.13 to the measured sampled data yields Allan deviation $\sigma(\tau)$, which can be plotted in log scale against the cluster size τ. Because the relationship 1.14 can also be understood as passing the total power output of the random process through a filter with transfer function in the form

$$H(s) = \frac{sin^4(s)}{(s)^2}, \tag{1.15}$$

there exists conjunction between the type of underlying random process and the cluster size τ. In other words, the filter's bandwidth is changig with the varying cluster size τ. So if the PSD function of a particular noise in the signal is known, the noise parameters can by estimated from the log scale Allan deviation graph (further called only "graph").

In the following list, the noise-related formulas for Allan variance computation for the five basic IMU noises are provided. The corresponding PSD functions and details are to be found in [27].

- *Quantization noise*:

$$\sigma_q^2(\tau) = \frac{3Q_z^2}{\tau^2} \tag{1.16}$$

  where Q_z is the quantization-noise coefficient. The magnitude of this noise corresponds to the graph with the slope -1 at $\tau = \sqrt{3}\ s$.

- *White noise*:
$$\sigma_w^2(\tau) = \frac{Q^2}{\tau} \tag{1.17}$$
where Q is the angle (velocity) random-walk coefficient which can be found as the value where the graph's curve with the slope $-\frac{1}{2}$ intersects the value $\tau = 1\ \ s$.

- *Bias instability*:
$$\sigma_b^2(\tau) = \frac{2B^2}{\pi} \times \left[\ln 2 - \frac{\sin^3 x}{2x^2}(\sin x + 4x \cos x) + C_i(2x) - C_i(4x) \right] \tag{1.18}$$
where B is the bias instability coefficient; $x = \pi f_0 \tau$, where f_0 is the cut-off frequency of the underlying PSD function and C_i is cosine-integral fuction. In the graph, the value of Allan deviation of the plateau for $\tau >> f_0$ can be used to estimate the bias instability coefficient according to approximative formula
$$B \approx \sqrt{\frac{2\ln(2)}{\pi}} \cdot ADEV_B \approx 0.664 \cdot ADEV_B \tag{1.19}$$
where $ADEV_B$ is the correspondig value of plateau in the graph.

- *Rate random walk*:
$$\sigma_r^2(\tau) = \frac{K^2}{3} \tag{1.20}$$
where K is rate-random walk coefficient, whose value can by read off as the the value of the graph with slope $\frac{1}{2}$ at $\tau = 3\ \ s$

- *Rate ramp*:
$$\sigma_{rr}^2(\tau) = \frac{R^2\tau^2}{2} \tag{1.21}$$
where R is the rate ramp coefficient. Rate ramp noise is a systematic error, a very slow deterministic drift over time. The magnitude of rate ramp R coeffiecient can be found in graph with slope $+1$ at $\tau = \sqrt{2}\ \ s$

- Other noises: other noises such as sinusoidal noise or exponentially correlated noises can also be described by the Allan variance although they are not so important for IMU sensors. Details are provided in [27].

If holds the assumption that the random processes present in the data are statistically independent, for the Allan variance can be written [27]:
$$\sigma^2(\tau) = \sigma_q^2(\tau) + \sigma_w^2(\tau) + \sigma_b^2(\tau) + ... \tag{1.22}$$
The accuracy of noise estimated parameters depends on a few factors. Aside from the errors caused by imperfections and distortions in the input data, also the Allan variance analysis method itself has theoretical accuracy limitation. It depends on the ratio of the total number of samples N to the number of

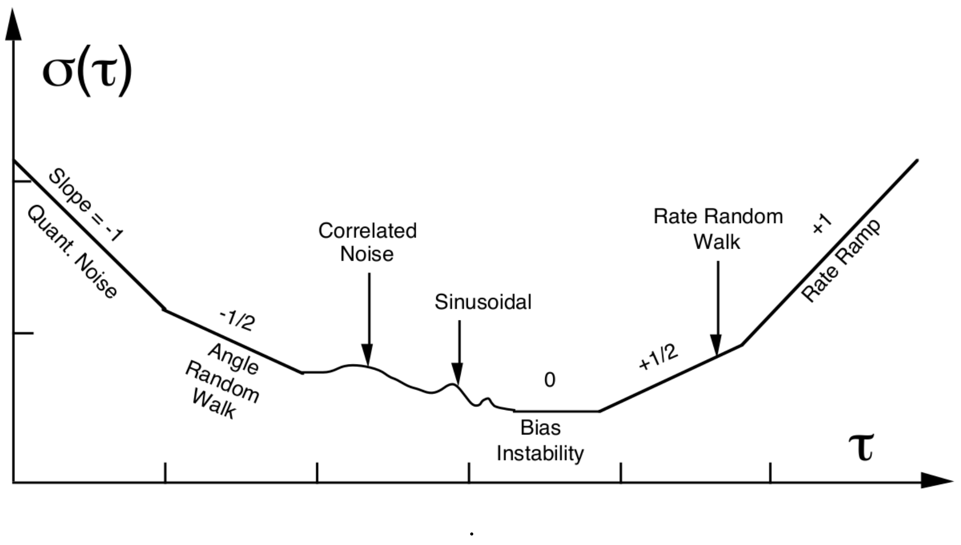


Figure 1.9: The characteristic parts of Allan deviation plot in log scale, [27]

samples M used to build the cluster with the size corresponding to the region where the noise parameter was estimated. It practically means that noise parameters such as Angular random walk or Quantization noise will be found with better accuracy than Rate random walk or Ramp noise parameters that are found for greater values of τ. The percentage error can be expressed as

$$\sigma_{\%} = \frac{1}{\sqrt{2(\frac{N}{M} - 1)}} \tag{1.23}$$

This equation can be also used to determine the total number of data samples N needed to achive a desired accuracy of certain parameter estimation.

The Allan variance noise analysis procedure for a gyroscope[8] can be summarized in the following steps:

1. get N samples from stationary measurement sampled with a certain sampling rate τ_0
2. form the clusters with size $\tau = m\tau_0$, overlapping method is frequently used (see figure 1.8)
3. compute the Allan variance and subsequently Allan deviation from either average of the output rate, or the output angles
4. repeate the previous two steps for different cluster sizes, plot the Allan deviation in log scale against the value of cluster size τ and estimate the noise parameters from the graph
5. repeat the analysis separately for each axis of the gyroscope

[8]analogically for the accelerometer

The relations between the particular noises and the Allan deviation log scale graph nicely summarizes the figure 1.9. The noise analysis of particular IMU sensors are described for example in thesis [30] or in paper [29].

1.8 Calibration

"*Calibration is the process of comparing instrument outputs with known reference information and determining the coefficients that force the output to agree with the reference information over a range of output values*" [14].

Nowadays, the IMU sensors come to market usually already calibrated by the manufacturer that also prescribes recommended working conditions such as temperature range. Better IMUs have sometimes even a self-calibration at one's disposal [31]. Nonetheless, available IMUs commonly enable also manual calibration, which could be very useful when high precision is of interest.

The standard method of accelerometer's calibration uses a linear matrix model comprising gains (sensitivity) and offsets for three axes. By placing the accelerometer in several defined orientations, these parameters can be found and used for calibration. Additional parameters such as cross-axis sensitivity can also be taken into account by extending the basic model. Similarly, an angular rate test serves as a calibration procedure for gyroscopes. An extensive description of the calibration methods present authors in [32] or in [33].

Chapter 2
Simulation

"Scientists have simulated ideas since the beginning of time."
Dr.Richard Gran, Appolo Lunar Module Autopilot designer

Since the very beginning of time, people seek answers to hard questions in many fields of science. The progress there is possible only thanks to the unique human ability to think abstractly, to imagine. To decrease the amount of abstraction, people create models describing the complex reality in a simplified manner and use them to estimate the desired information and to come to conclusions.

A simulation can be defined as a creation of such models that can be manipulated logically to decide how the physical world works [34]. However, in a modern understanding, the simulation in the scientific domain is inseparably connected to computers and numerical methods.

While simple problems are easily solvable analytically, and the knowledge of physical laws provides there sufficient intuition only with a pen and paper, in more complex cases, the researched problem may be too hard to get a closed-form solution, or not easy to imagine. At this moment the computer simulation saves the day. High computational power and numerical methods such as Monte Carlo approach or iterative solvers can be applied to hard mathematical problems to obtain an approximate solution. A vast amount of models incorporated into a single simulation enables studying of model interactions on different abstraction levels. By all means, the simulation can be beneficial even in simple cases, where it can provide the validation of computed results and better insight into the simulated issue.

The history of computer simulation can be traced back to 1940s when during the Manhatten project [1] the first digital computers were used for physical simulations. Since that moment, computer simulation becames gradually inherent tool in many fields. In last years, the capability to precisely simulate and predict the behavioral of very complex systems together with the ever-increasing computer performance enables the advent of new disciplines in the technical field such as virtual and augmented reality or virtual product

[1]American military research and developement project in 1942 - 1946 that resulted in the very first nuclear weapons.

development (digital twins). Especially the industry benefits massively from the applied simulations that can significantly decrease the production costs, increase the reliability of products and accelerate whole development.

In this chapter, I review modern robotics-oriented simulation software tools. In the first section 2.1, I state a few requirements the tool must or should have in order to simulate the IMU sensor attached to the mobile service robot. In the next section 2.2, I discuss the properties of a few particular robotic simulators that are meeting the previously stated requirements. Their pros and cons are discussed with respect to the goal of this thesis. Finally, in sections 2.3 and 2.4, I describe more deeply the finalist of the selection - the Gazebo simulator in conjunction with ROS[2].

2.1 IMU simulation requirements

To be able to simulate IMU sensor attached on the mobile service robot, the simulation tool should fulfil following requierements:

- real physical simulation
- in-built support for both mobile robot and IMU sensor simulations
- easy definition of various test scenarios
- open source software if possible
- ROS compatibility

In addition, it would be really nice to have:

- high-quality documentation and tutorials
- high degree of possible customisation
- contemporary, still evolving software with a vibrant user community
- 3D GUI support for visualization
- programmable in widely used programming language, such as Python or C++
- UNIX support
- easy portability of the code between simulation and real robot

[2]Robotic Operating System is a well established robotics middleware. It is described in detail in section2.3.

2.2 Mobile robotic simulators

The phenomenon of simulation brought an entirely new development paradigm to mobile robotics. The algorithms can be now easily and safely debugged in the simulator environment in a much shorter time than before, without the danger of damaging an expensive robot or causing any other kind of loss. Also, new types of kinematics structures can be tested without the need for expensive prototypes or without a particular testing environment. As a result, much time and money can be saved.

Over the last twenty years, various specialized mobile robotic simulators appeared on the market. Among them, there are many free, open source projects too. Contemporary mobile-robotic-oriented simulators provide much more than just the estimation of real robot's behavioral. They typically come with a set of frequently used robots and environment models and offers a wide variety of robotics-specific additional tools such as controller tuners or plotting tools. The user can also benefit from rich visualization in both 2D or 3D.

From the technical point of view, the modern simulators are usually standalone applications that can run in real-time in multiple processes on distributed computers and that utilize a physics engine library [3] to estimate the resulting response of the model.

According to [36], the simulator's activity during a generic simulation can be described by the following equations 2.1, 2.2, and 2.3. The simulator periodically updates the system state (i.e. the state of the simulated model) **x** by recomputing the following equations in a loop.

$$\dot{\mathbf{x}}_k = \mathbf{f}\left(\mathbf{x}_k,\, u\left(\mathbf{x}_k,\, t_k\right),\, t_k\right) \tag{2.1}$$

$$\mathbf{x}_{k+1} = \int\limits_{t_k}^{t_k+\Delta t} \dot{\mathbf{x}}_k dt + \mathbf{x}_k \tag{2.2}$$

$$t_{k+1} = t_k + \Delta t \tag{2.3}$$

The function **f** describes the change of the system states **x** over time, which should express to the real physical properties of the simulated object, whereas $u\left(\mathbf{x}_k,\, t_k\right)$ is the control command, and t_k is a step-size of simulation time. More detailed description of the working principles of mobile robotics simulators is to be found in [37].

2.2.1 State of the art robotics simulators

Although multiple papers comparing the quality and performance of mobile robotic simulators have been already published, for example, [38, 39, 36,

[3]The computer software responsible for simulation of physical behavior, usually developed as external libraries. Among the most frequently used physics engines belong Open Dynamics Engine (ODE), Dynamics Animation and Robotics Toolkit (DART), BULLET or PhysX. Details and their comparison can be found in [35].

Name	Gazebo	V-REP	Webots
Current version	9.0.0	3.6.1	R2019a
License	Apache 2.0	proprietary / GNU GPL	originally proprietary (recently changed to Apache 2.0)
Developer	Open Source Robotics Foundation (OSRF)	Coppelia Robotics GmbH.	Cyberbotics Ltd.
OS support	Linux; on other OS complicated	Linux, MS Windows, macOS	Ubuntu Linux, MS Windows, macOS
Real physics engine	ODE, Bullet, Simbody, DART	ODE, Bullet, Vortex, Newton	extended ODE version
Model formats	SDF/URDF, OBJ, STD, Collada	OBJ, STL, DXF, 3DS, Collada, URDF	VRML, X3D
Model editor	yes, limited capabilities	yes, full-featured editor including mesh manipulation	yes
World modeling	only simple models included, access to a big open source model databases possible	many high-quallity models included	many high-quallity models included
Programming language	C++	C++, Lua, Java, Python	C/C++, Python, Java, MATLAB
Extensibility	plugins, API	API, plugins, add-ons, embedded scripts	API, plugins, add-ons
ROS support	yes, full support	yes, ROS interface	yes, ROS controller
Documentation	tutorials, API reference	tutorials, API reference	tutorials, API reference
Community	big comunity, alive public forum and mailing list	community, public forum	public forum, customer paid support and mentoring
Comparative advantages	**close integration with ROS, active community support, clear development roadmap, XML world description**	**fluid dynamics simulations, huge model database, intuitive model editor, full-featured and stable SW**	**fast custom rendering, big model database, well-documented, tested and stable SW**
Webpage	www.gazebosim.org	www.coppeliarobotics.com	https://cyberbotics.com/

Table 2.1: The comparison of mobile robotics simulators

37, 40], most of them are somewhat outdated because of rapid progress and changes in this field. Often the simulator's drawbacks mentioned in older papers have been already fixed in current versions or the development of the simulator itself has been discontinued recently. To my best knowledge, probably the best-updated list of currently available robotic simulators is to be found on corresponding English Wikipedia webpage [41]. There are also included tables comparing their main properties and pros and cons.

Besides the prevalence of small or old projects, several well-established, mobile-robotics-oriented simulators can be distinguished. I selected three particular simulators from them for deeper comparison: *Gazebo, V-rep*, and *Webots*.

2.2.2 Gazebo, V-rep and Webots comparison

All three simulators are currently actively developed and widely used in practice. As mature software tools, they are all suitable for complex simulations with mobile robots. They offer a high degree of customization, 3D visualizations and have built-in model editors. Based on the official documentation available in the project webpages [42, 43, 44], they all support plenty of sensor models including the inertial sensors.

In Webots and Gazebo, ready-to-use IMU models exist. The mechanical structure of the real IMU sensor itself is not simulated, but angular velocities, accelerations, and orientation are computed directly by the physics engine, which calculates them internally for every simulated body. In V-rep simulator, the physics engine is also used, but the accelerometer and gyroscope are separate models whose output is determined by measuring the force interactions between two simulated masses [45]. In general, in all simulators, IMU models should return noiseless[4]("ground truth") values for accelerations, angular velocities, and orientation, unless artificially noise is applied to the IMU's output [46], [47], [48].

The basic properties of simulators are compared in table 2.1.

Now the question is, which one is the best choice? In a comprehensive study [49] from 2014, authors studied the user feedback of more than one hundred researchers and engineers from both different countries and different areas of mobile robotics research. Survey participants were asked about, what simulator do they use as a main toll, how they are satisfied with it, and what other simulation tools they know or tried. Gazebo came out as the most frequently used (15% of participants), the best-known and the best community having simulator. In comparison, Webots and V-rep were both identically used by 7% of participants. However, in terms of documentation, tutorials, quality of API and support especially V-rep obtained better rating. The authors conclude that both Gazebo and V-rep are a good choice.

In the same year, Lucas Nogueira made a comparative analysis between Gazebo and V-rep simulators [40]. He did a few experiments with the same model in both simulators and compared them from different viewpoints. He concludes that Gazebo is better integrated with ROS and provides more ready-to-use plugins created by the community. On the other hand, V-rep provides more in-built features especially when it comes to model editing capabilities. And last but not least, during the experiments, V-rep also had a slightly smaller demand for computational power.

In the same year, one more paper concerning mobile robotics simulators

[4]physics engine imperfections, or the incorrect setting can cause a noisy IMU output

was written [37]. Authors compared Gazebo only with Microsoft Robotics Developer Studio (whose development has been discontinued in the meanwhile), 2D Carmen simulator and pure ODE physics engine. They were mainly focused on the physical accuracy and conclude that pure ODE physic engine with their ad-hoc implementation of experiment-specific wrapping code provided slightly more accurate results than Gazebo, which was using ODE physics engine too.

Probably the newest study concerning the V-rep and Gazebo has been published in 2018 [50]. Authors compare V-rep, Gazebo and ARGoS simulators. They more or less confirm the results already published in [40]. They argue that V-rep is the most complex, easy-to-use and full-featured simulator, while Gazebo maintains better ROS compatibility and overcomes V-rep's performance when it comes to large scale environment but lacks some features and is sometimes more difficult to use. The ARGoS simulator, mainly designed for robotics swarms, obtained the lowest rating.

To my best knowledge, there is no up-to-date paper comparing all three simulators including Webots. It can be possibly caused by the pricing policy of its developer company Cyberbotics Ltd. While the V-rep is available for free for non-commercial and educational purposes, and Gazebo is completely open source from its beginning, the Webots was subject to a charge even for universities and maybe for that reason was the university's research mainly focused on other simulators[5]. But recently, on the 18th December 2018, the company Cyberbotics Ltd. announced that from this day forward the new Webots' versions become completely open source software under Apache 2.0. licence [51]. This decision together with a great heritage of well-tested and documented proprietary software makes Webots a serious competitor of the other simulators.

2.2.3 The finalist selection

As described in previous subsections, all three introduced simulators represent advanced state-of-the-art tools with generally similar capabilities. All of them are usable from the viewpoints of the technical requirements for IMU simulation. Also, to my best knowledge, there is no paper at all concerning directly the topic of fidelity of IMU simulation in any robotics simulator.

Based on available papers, the V-rep simulator seems to be slightly better than its competitors, especially when it comes to the number of features. According to official project website, also Webots is feature-rich and stable sofware. But since this thesis is strictly speaking also a part of commercial research project, strong integration with ROS, and open source code were desired, the V-rep's proprietary license was a limiting factor. For the same reason, the Webots simulator was finally not considered, because the selection of the simulator was done already a few months before Webots become open source software, even though it overcomes Gazebo in terms of stability and documentation. In addition, the mentioned possible Gazebo drawbacks such

[5]The company itself claims that Webots is widely used at many universities.

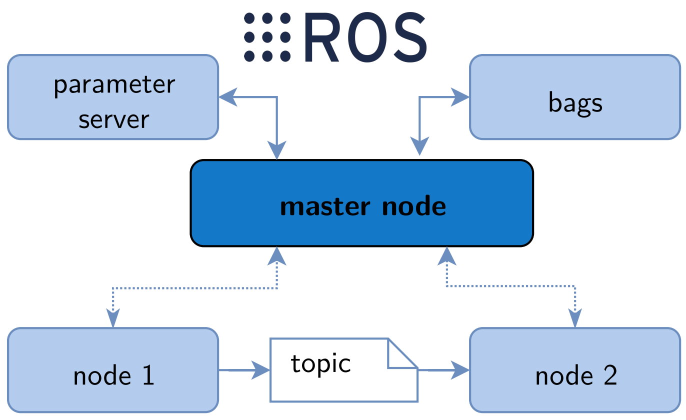


Figure 2.1: The simplified structure of the ROS computational graph

as more difficult model definition or smaller model database does not play a vital role from the point of this thesis.

Thus, I picked up Gazebo simulator in conjunction with ROS as the main simulation tool for this thesis. The following sections describe them more deeply.

2.3 ROS

Robotic Operating System is a robotic middleware rather than an operating system in a standard understanding, although it has some similarities with the classical operating systems. It aims to make development and research in the field of robotics easier by providing a unified framework and guidelines to enable the reuse of the code. In the last years, ROS has become more or less a standard in both industry and university research concerning mobile robotics.

Because the robotics is an extensive field with quite diverse types of hardware with often conflicting requirements, the heart of the ROS consists of a hardware-independent infrastructure that provides unified inter-process communication. Besides, ROS provides also supportive visualization and simulation tools, a hardware abstraction layer, and hardware-specific libraries implementing algorithms for particular applications such as control, navigation or mapping.

Running ROS system can be seen as a distributed system, a so-called *computational graph*, whose separate processes are called ROS nodes. The computational graph has several basic parts:

- *Nodes* are processes that serve a specific task (e.g., motor driver). Usually, multiple nodes are communicating with each other over topics.
- *Master node* is part of the ROS core that registers other nodes and manages the communication among them. Without the master node

administration, the regular nodes cannot communicate. In the running ROS system, there can always be only a single master node.

- *Topics* are communication buses with a specific name and data type that identify the content of the messages. They are used by nodes to send messages on the publish/subscribe basis. ROS defines a family of standard data types; however, custom datatypes' definition is possible too.
- *Messages* are simple data snippets that are sent over topics among nodes.
- *Bags* are special ROS format for logging, storing and replaying the captured messages.

The ROS components are stored in a unified way, sometimes called a *ROS filesystem*. A *package* is the basic organizational unit of source code that can be built separately and usually contains a ROS node, or library implementation. Every package has its own directory and a unique *manifest XML* file. A *repository* is then a collection of packages that share mutual VCS[6].

The project developed under ROS usually contains multiple packages and is build using the automated CMake-based build system called Catkin.

The picture 2.1 depicts an example of the communication between two nodes 1 and 2. Firstly, during the initialization of node 1 master node registers that node 1 wants to publish messages on a certain topic. At the moment when node 2 starts, and when the master figures out that node 2 subscribes the same topic, the master node arranges the peer-to-peer communication from node 1 to node 2. If a service instead of messages were used, the procedure would be similar except for applying principles of bidirectional client-server communication. The master node also arranges the access to ROS parameter server or data logging. The ROS fully supports programming languages Python, C++, and Lisp (for these languages the complete client libraries are available).

A more in-depth description of ROS is to be found in the official online documentation [52], or in the handbook [53].

2.4 Gazebo

The development of Gazebo simulator started in 2002 and was officially introduced in 2004 at the University of Southern California to create high fidelity, powerful and 3D open source simulator [54]. Firstly, it was developed as a part of the Player/Stage[7] project, but in 2012 the Open Source Robotics Foundation responsible for ROS adopted the Gazebo project. Since that time, Gazebo won strong ROS-oriented user community. And it is likely to be also developed in the future thanks to the clear development roadmap [50].

[6]Version Control System

[7]Player is another open source robotics middleware, whereas Stage is a 2D simulator. Both tools are still maintained but in last years often replaced by ROS and Gazebo.

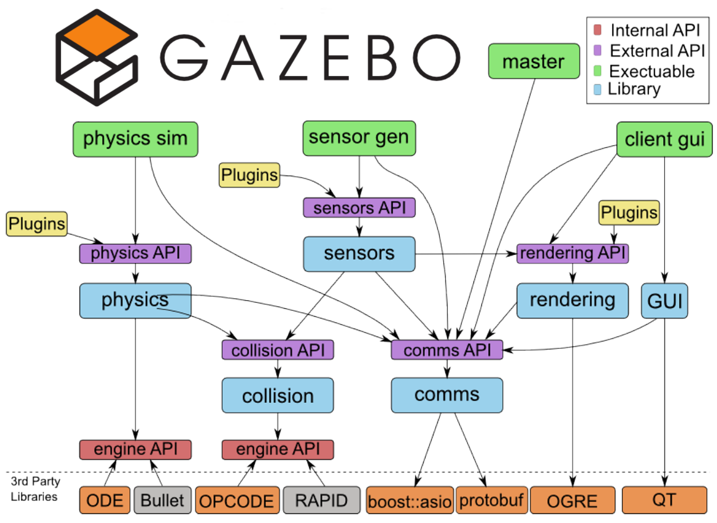


Figure 2.2: Simplified Gazebo dependency graph [49]

Gazebo's main properties and advantages have been already mentioned in the previous section 2.2.2 and in the table 2.1. In this section, I only briefly describe the Gazebo's structure and its connection to ROS.

Gazebo simulator consists of two main parts. **`gzserver`** is the simulator's core running the simulation loop responsible for the computation of physical simulations, using external physics engines such as ODE. **`gzclient`** provides 3D visualization and handy GUI tools to control the underlying gzserver or model world or edit models. There is also **`gzweb`** that can be used instead of gzclient on a web server. The decoupling of the computational and rendering parts enables running of computationally-expensive gzserver on a mainframe and visualizing the output on an ordinary computer for ideal load balance. More detailed structure is depicted in the picture 2.2.

XML-like SDF files define all objects in the simulation including the world model. SDF format has some advantages over the older ROS URDF format, which lacks capabilities like joint loops, or friction definition. However, the robots defined under URDF can be easily converted to SDF thanks to provided parser. There are several SDF models' categories like, e.g., world, model, or actor that can include links, joints, sensors or other parts whose properties can be defined using specific tags. For example, a simple differential drive wheeled robot model, can consist of a rigid box (robot's body), two cylinders (drive wheels) connected by revolute joints, and one ball joint instead of caster wheel.

SDF language has various tags, that allow to set the model properties. For example, **`visual`** tag describes the final appearence in the visualization. Apart of basic shapes, colors, and textures, custom materials and Collada meshes can be add to model for realistic appearence. The complete SDF specification is available online in [56].

To make the models execute the desired behavior, either the user can

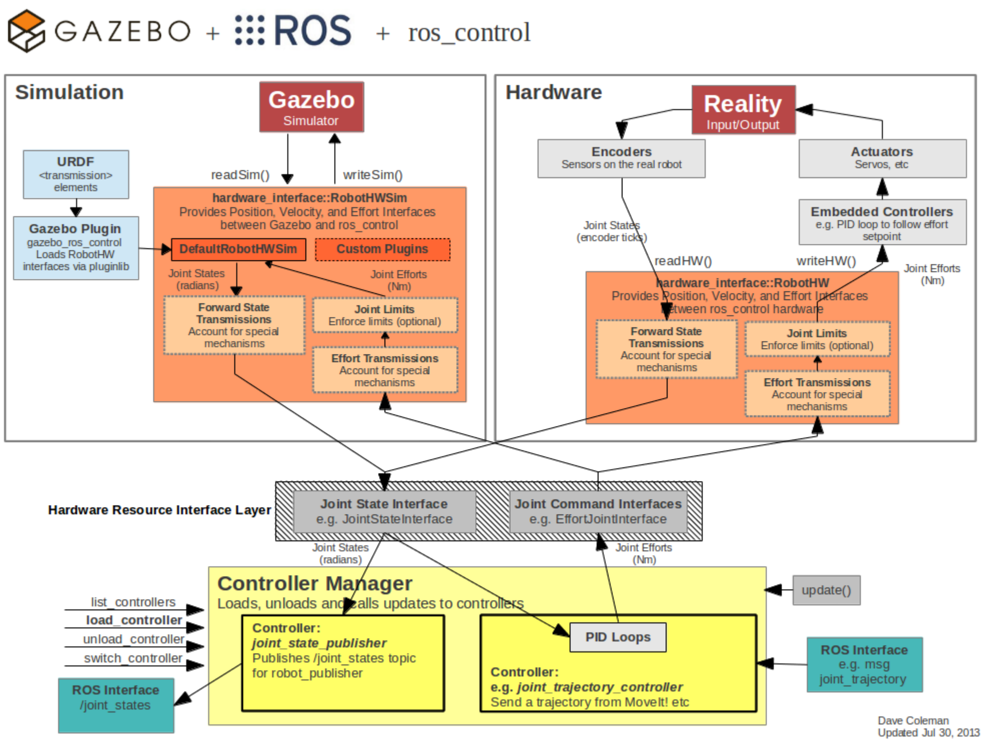

GAZEBO + ROS + ros_control
Simulation
Gazebo
Simulator
URDF
<transmission> elements
Gazebo Plugin
gazebo_ros_control
Loads RobotHW interfaces via pluginlib
readSim()
writeSim()
hardware_interface::RobotHWSim
Provides Position, Velocity, and Effort Interfaces between Gazebo and ros_control
DefaultRobotHWSim
Custom Plugins
Joint States (radians)
Joint Efforts (Nm)
Forward State Transmissions
Account for special mechanisms
Joint Limits
Enforce limits (optional)
Effort Transmissions
Account for special mechanisms
Hardware
Reality
Input/Output
Encoders
Sensors on the real robot
Actuators
Servos, etc
Embedded Controllers
e.g. PID loop to follow effort setpoint
Joint States (encoder ticks)
readHW()
writeHW()
Joint Efforts (Nm)
hardware_interface::RobotHW
Provides Position, Velocity, and Effort Interfaces between ros_control hardware
Forward State Transmissions
Account for special mechanisms
Joint Limits
Enforce limits (optional)
Effort Transmissions
Account for special mechanisms
Hardware Resource Interface Layer
Joint State Interface
e.g. JointStateInterface
Joint Command Interfaces
e.g. EffortJointInterface
Joint States (radians)
Joint Efforts (Nm)
Controller Manager
Loads, unloads and calls updates to controllers
update()
list_controllers
load_controller
unload_controller
switch_controller
ROS Interface
/joint_states
Controller:
joint_state_publisher
Publishes /joint_states topic for robot_publisher
PID Loops
Controller:
e.g. joint_trajectory_controller
Send a trajectory from MoveIt! etc
ROS Interface
e.g. msg joint_trajectory
Dave Coleman
Updated Jul 30, 2013

Figure 2.3: How `ros_control` is connected to Gazebo and real HW in order to control the robot's (model's) movement[55]

implement the actions as a C++ plugin and assign it to the corresponding model, or one can use ROS-compatible plugins, that make the model's control accessible over standard ROS communication interface. In other words, rather than defining the model's behavior directly in Gazebo's plugin, the commands are sent to models as ROS messages or services over specific ROS topics. This approach is very versatile because it splits the simulation and behavior logic. The algorithms developed under ROS for real robots apply without any change also to models in Gazebo and vice versa. For the movement control `ros_control` package can be used. A comprehensive flowchart in the picture 2.3 shows how it communicates with Gazebo simulator and real hardware as well. Similarly, there are also ROS-Gazebo sensor plugins, defining the sensor's behavior. The approach using ROS for the logic definition was used also in this thesis and is described more in the next chapter.

Further details concerning Gazebo simulator are to be found in the project's offical webpage [42].

Chapter 3
IMU modeling

This chapter is devoted to the design and implementation of the IMU model and supportive simulation framework. As stated in the preliminary chapter, the primary goal of this thesis is the creation of an accurate BNO055 IMU model with emphasis to heading angle estimation to increase the fidelity of robot's simulations for future rapid development purposes.

In the first section 3.1, I expand on the nature of the raw data. I discuss the properties of the IMU sensor BNO055, and I describe the Allan noise analysis of the BNO055 here. I also describe the mobile robot on which the real experiments were carried out.

Subsequently, in section 3.2, I propose the design of a custom simulation framework suitable for IMU model development concerning the main goals of this thesis. I also provide some interesting implementation details.

And finally, section 3.3 deals with the ROS/Gazebo IMU model design and implementation.

3.1 Raw data

Virtually the only source of information available for IMU model development in this thesis, except for the official datasheet provided by the IMU's manufacturer, were raw data from real experiments provided by the project owner in the form of ready-to-use datasets in CSV files. Since this thesis is written about a part of a real industrial research and development project, the raw data were not gathered by myself, but measured by different company's department few months before this thesis started. During the work on this thesis, the experimental setup consisting of a mobile robot with the BNO055 IMU sensor was not physically available for any further experiments.

All the experiments were performed in a flat, but the outdoor environment, on a custom mobile robot, which is closer described in the corresponding subsection 3.1.1. The particular testing scenarios and trajectories are then compared in chapter 4.

For each dataset the few different types of raw data were provided.

The reference position and orientation information was provided as a set of waypoints (sampled position and orientation information with precise timestamps) with sampling rate approximately 5 Hz, measured by very precise aNavs GPS sensor [57] which has a theoretical accuracy of 0.01 - 0.03 meters and which provides the orientation in form of quaternions with about 0.25° uncertainty.

The IMU raw data was measured by the Bosch BNO055 IMU sensor, which is closely described in subsection 3.1.2.

The data capture system from ROS was used to store the data in ROS bags format that was afterwards converted to CSV files.

3.1.1 Mobile robot

All the raw data were measured on a differential drive wheeled mobile robot. The robot has two separately controlled rear wheels and two front freely movable caster wheels, as depicted in the picture 3.1. Differential drive mobile robots are frequently used because of their simplicity and easy control.

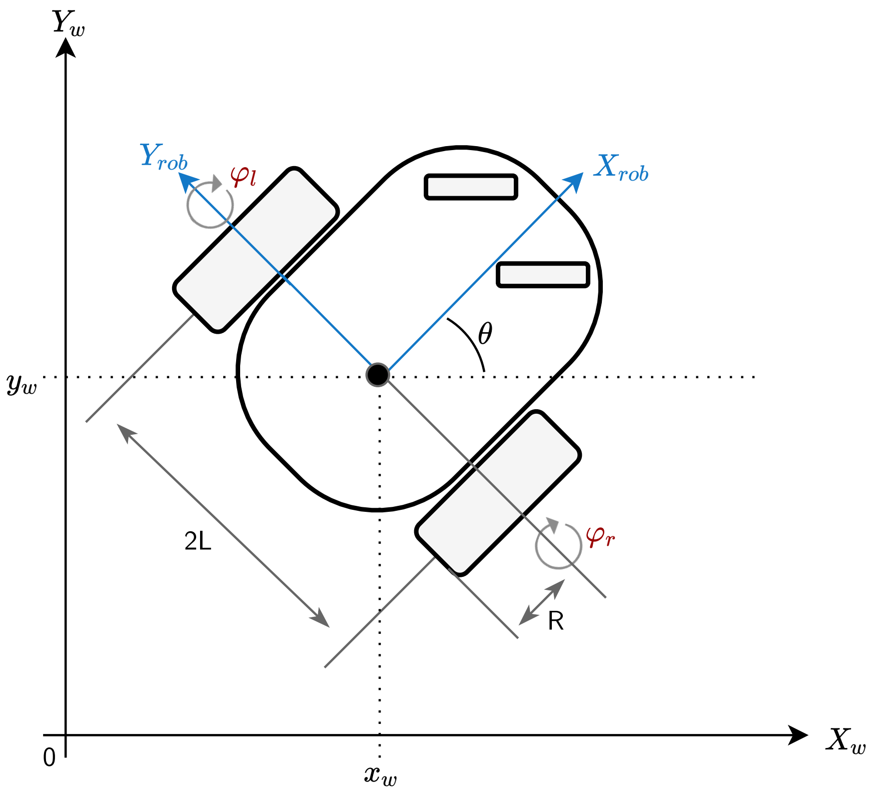


Figure 3.1: The differential drive mobile robot with two front caster wheels

The following equations can describe the kinematic behavior of a differential drive robot.

Let us assume the robot's movements only in the horizontal plane. Three independent coordinates fully determine the robot's state

$$\mathbf{x}_w = [x_w, y_w, \theta]^T \tag{3.1}$$

where θ stands for the robot's heading angle (throughout this thesis also called a yaw angle). The only controllable joints are the rear wheels, which leads to two joint coordinates

$$\mathbf{q} = [\varphi_l,\, \varphi_r]^T \tag{3.2}$$

where φ_l and φ_r are angles of left and right rear wheels respectively.

A homogenous transformation can be defined to transform the coordinates between the world's and robot's frames. The homogenous coordinates are given then

$$\mathbf{x}_w = [x_w,\, y_w,\, 1]^T \tag{3.3}$$

$$\mathbf{x}_{rob} = [x_{rob},\, y_{rob},\, 1]^T \tag{3.4}$$

If we define the transformation matrix in the following way

$$\mathbf{T}_{rob}^{w} = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 & x_w \\ sin(\theta) & cos(\theta) & 0 & y_w \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

for the transformations holds

$$\mathbf{x}_w = \mathbf{T}_{rob}^{w}\mathbf{x}_{rob} \tag{3.6}$$

$$\mathbf{x}_{rob} = (\mathbf{T}_{rob}^{w})^{-1}\mathbf{x}_w \tag{3.7}$$

Note that in general, for the circular motion holds the formula

$$\begin{gathered} \mathbf{v} = \omega \times \mathbf{r} \qquad (3.8) \\ |\omega| \equiv \dot{\varphi} \\ |\mathbf{v}| \equiv v \end{gathered}$$

where $\mathbf{r}$ is the radius vector, ω is the vector of angular velocity.

Now let v_l, v_r are the linear velocities of left and right rear wheels, and let $\dot{\varphi}_l$, $\dot{\varphi}_r$ are angular wheel velocities (or shortly "wheel velocities") of left and right rear wheels respectively.

So if we assume that the movement is not possible in the direction of Y_{rob} axis, it is easy to see that for the velocities of the robot in the robot's frame holds

$$\dot{x}_{rob} = \frac{v_l + v_r}{2} = \frac{R(\dot{\varphi}_l + \dot{\varphi}_r)}{2} \tag{3.9}$$

$$\dot{y}_{rob} = 0 \tag{3.10}$$

$$\dot{\theta} = \frac{v_r - v_l}{2L} = \frac{R(\dot{\varphi}_r - \dot{\varphi}_l)}{2L} \tag{3.11}$$

Pluging in 3.9, 3.10, 3.11 into the derivative of equation 3.7 yields the kinematics model of the differential drive in world's reference frame

$$\dot{x}_w = \frac{R}{2}(\dot{\varphi}_l + \dot{\varphi}_r)cos(\theta) \tag{3.12}$$

$$\dot{y}_w = \frac{R}{2}(\dot{\varphi}_l + \dot{\varphi}_r)sin(\theta) \tag{3.13}$$

$$\dot{\theta} = \frac{R(\dot{\varphi}_r - \dot{\varphi}_l)}{2L} \tag{3.14}$$

where equations 3.12, 3.13 and 3.14 are the solution of the direct kinematic task, which can be also rewritten into to matrix equation

$$\begin{bmatrix} \dot{x}_w \\ \dot{y}_w \\ \dot{\theta} \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{R}{2}cos(\theta) & \frac{R}{2}cos(\theta) \\ \frac{R}{2}sin(\theta) & \frac{R}{2}sin(\theta) \\ -\frac{R}{2L} & \frac{R}{2L} \end{bmatrix}}_{\mathbf{S}} \underbrace{\begin{bmatrix} \dot{\varphi}_l \\ \dot{\varphi}_r \end{bmatrix}}_{\dot{\mathbf{q}}} \tag{3.15}$$

$$\dot{\mathbf{x}}_w = \mathbf{S}\dot{\mathbf{q}} \tag{3.16}$$

For the purpose of robot's simulation and control, solving the inverse kinematics task is useful. Generally, the closed-form solution requires solving the matrix equation 3.16 which leads to the matrix equation

$$\dot{\mathbf{q}} = \mathbf{S}^{-1}\dot{\mathbf{x}}_w \tag{3.17}$$

In case of this robot, the matrix **S** is not easily invertible and the solution can not be foud analytically. However, the closed-form solution still exists and can be found by solving the inverse kinematics task iteratively using the Jacobi matrix approach. In this way the solution was obtained as

$$\dot{\varphi}_l = \frac{1}{R}\left(L\dot{\theta} + cos(\theta)\dot{x}_w + sin(\theta)\dot{y}_w\right) \tag{3.18}$$

$$\dot{\varphi}_r = \frac{1}{R}\left(-L\dot{\theta} + cos(\theta)\dot{x}_w + sin(\theta)\dot{y}_w\right) \tag{3.19}$$

The URDF/SDF model of this mobile robot and the equations 3.18 and 3.19 were provided by a different company's department. Due to the proprietary license, I am not allowed to publish the model's details and pictures.

Nevertheless, many open source models of differential drive mobile robots very similar to this one exist (e.g., well-known TurtleBot). Also, the solution of inverse kinematics for differential drive robot including the full derivation can be found in the literature, for example in [58].

3.1.2 IMU sensor BNO055

Bosch BNO055 is a smart 9-degrees-of-freedom IMU sensor. In addition to the standard 3-axis accelerometer and gyroscope, also a 3-axis magnetometer, 32-bit microcontroller running proprietary BSX3.0 FusionLib software, and temperature sensor are included. Thanks to the embedded microcontroller the IMU sensor can operate in few different operating modes with inbuilt features like sleeping mode, interrupts caused by motion, multisensor data fusion, filtering, and auto-calibration. If the FusionLib software features are enabled, the sensor can directly produce also the absolute orientation in the form of quaternions, or Euler angles computed from filtered linear accelerations, angular velocities and magnetometer's output. Some of the technical parameters available in the official datasheet are listed in table 3.1.

	Bosch BNO055	
	Gyroscope	**Accelerometer**
Range	programmable, 125 – 2000°/s	programmable, ±2 - ±16g
AD convertor	16bit	14bit
Sensitivity	16LSB/°/s	1LSB/mg
- tolerance	±1 (max ±3) %	±1 (max ±4) %
- temperature drift	±0.03 (max ±0.0.7) %/K	±0.03 %/K
Nonlinearity	±0.05 (max ±0.2) %FS	±0.5 (max 2) %FS
Zero rate/g offset	±1 (max ±3)°/s	±80 (max ±150) mg
- temperature drift	±0.015 (max ±0.03) °/s per K	±1 (max ±3.5) mg/K
- voltage drift	0.1 °/s/V	±1.5 (max ±2.5) mg/K
Bandwidth	programmable, 12 – 523 Hz	programmable, 8 – 1000Hz
Output noise density	0.014º/s/√Hz (at 47Hz)	150 (max 190) µg/√Hz
Cross Axis Sensitivity	±1 (max ±3) %	±1 (max ±2) %

Table 3.1: Selection from BNO055 IMU sensor's datasheet [59]

During all the experiments with the mobile robot, a rapid development shield called "Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout - BNO055" was used instead of direct implementation of the sensor chip itself. All the raw data were captured in the so-called "IMU" data fusion mode. In this mode, the magnetometer is not used, and thus only the accelerometers and gyroscopes are fused to get the absolute orientation output with the output rate 100Hz. Also, the otherwise adjustable parameters like bandwidth or range are set automatically in this mode.

For an accurate simulation, the recognition and description of noise sources are important. Unfortunately, the official datasheet [59] is not very talkative when it comes to this point. Some noise-corresponding values are provided by the manufacturer (see table 3.1), but the listed values do not provide the full picture of noises, especially when it comes to long-term operation. Important parameters are either completely missing (e.g., in-run bias stability), or the values are defined in such a way that they are not related a much to the scenarios in which the raw data were gathered. The thing is that raw data were measured when the IMU sensor BNO055 was in "IMU" mode using internal data fusion, auto-setting, and filtering algorithms while datasheet assumes only simpler sensor's modes, usually only for defined temperature 25°C.

Thus, I regard the values listed in datasheet by the manufacturer as rather as additional information, and I decided to perform the experimental noise analysis to get the long-term noise description.

3.1.3 BNO055 noise analysis

For the noise analysis, I used the Allan variance overlapping method. Its theoretical background was provided in section 1.7. Because the original experimental setup with Adrafruit board mounted on the mobile robot was not available for any further experiments. I had to make the static data acquisition necessary for the Allan variance analysis myself.

I done the data acquisition on a different hardware rapid development board, so-called IMU wing-board for "Atmel Xplained Pro" which however implements the same IMU chip BNO055. The arrangement of the testing setup depicts the picture 3.2. I configured BNO055 IMU chip in the same way as the IMU used in the experiments with the mobile robot ("IMU" data fusion mode, magnetometer off, auto-settings and filtering on, 100Hz). An exception is the UART bus which was used instead of I2C bus together with the UART/USB converter (no I2C/USB converter was available), but it should not have any influence on the measured data. The data were gathered into ROS bags by Lenovo X201 in default hardware configuration, OS Linux Mint 19 which was running an open source BNO055 ROS driver [60]. In total, I measured three independent datasets, each of duration approximately six hours. All measurements were performed at standard room temperature (about 23°C). The output units of the IMU were set to rad/s and m/s^2 for angular velocities and linear accelerations respectively.

Over the years, many tools for Allan variance analysis were implemented. Sensor fusion and tracking toolbox for Matlab implements the complete set of tools for Allan variance analysis, but it is an extra paid tool [61]. Alternatively, there are also open source scripts implemented by the Matlab community provided over Matlab File-Exchange platform. In the Python world, there is an open source Python module called AllanTools [62]. For this thesis, I used a tool provided by GAVlab team (The GPS and Vehicle Dynamics Laboratory at the Auburn University) that is already customized to cooperate directly with the ROS bag data format [63].

The initial attempts to get the Allan deviation graphs and to compute noise parameters using the tool [63] led to strange results. Subsequently, when plotting the raw data, I found out that the Allan variance analysis is quite sensitive to outliers[1]. Despite my best effort to ensure a quiet place under constant conditions when measuring the static data, in every dataset, I obtained were some outliers leading to wrong Allan deviation graphs. Thus, I applied the data post-processing to filter out the outliers. For that purpose, I used the *interquartile range* (IQR) statistical approach. This method was successfully exploited by the author in thesis [30] to post-preprocess the raw

[1]In this case, rare samples with extraordinarily high values compared to the rest of the data.

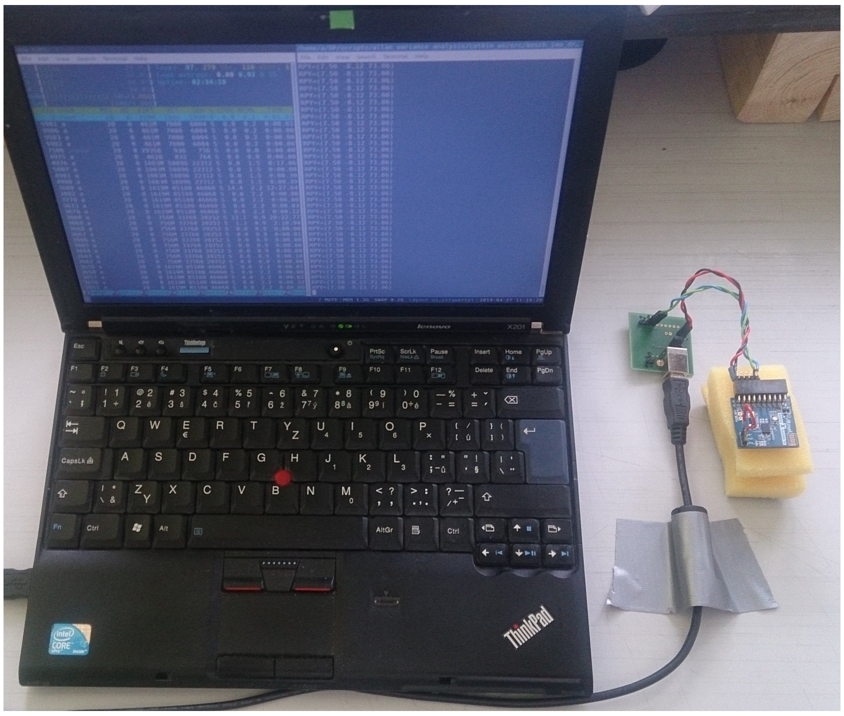


Figure 3.2: The testing setup used for static data acquisition, the sensor itself was placed on a piece of soft foam to mitigate the prospective influence of any vibrations of the surface beneath the sensor.

data before IMU Allan variance noise analysis.

The IQR is a statistical measure of dataset's variability which is defined as

$$IQR = Q_3 - Q_1 \tag{3.20}$$

where Q_1 and Q_3 are first and third quartiles of the dataset respectively. The figure 3.4 explains the meaning of the minimum and maximum thresholds, given by equations [30]:

$$min = Q_1 - IQR \tag{3.21}$$

$$max = Q_2 + IQR \tag{3.22}$$

This filtering was implemented as a post-processing Python script which replaces the detected outliers by the mean value before the analysis.

The pictures 3.5, 3.6, and 3.7 show graphically the resulting Allan deviation graphs plotted based on data computed by the script [63]. Note that the $ADEV$ axes have units rad/s and m/s^2 for the gyroscope and accelerometers respectively. The obtained results were afterward converted to appropriate units. Table 3.2 summarizes the estimated noise parameters. The values in the brackets are theoretical accuracies computed according to equation 1.23 and converted to the uncertainty of parameters. The final estimation of the noise parameters (highlighted by green color) was obtained by averaging individual noise parameters of three independent measurements using the

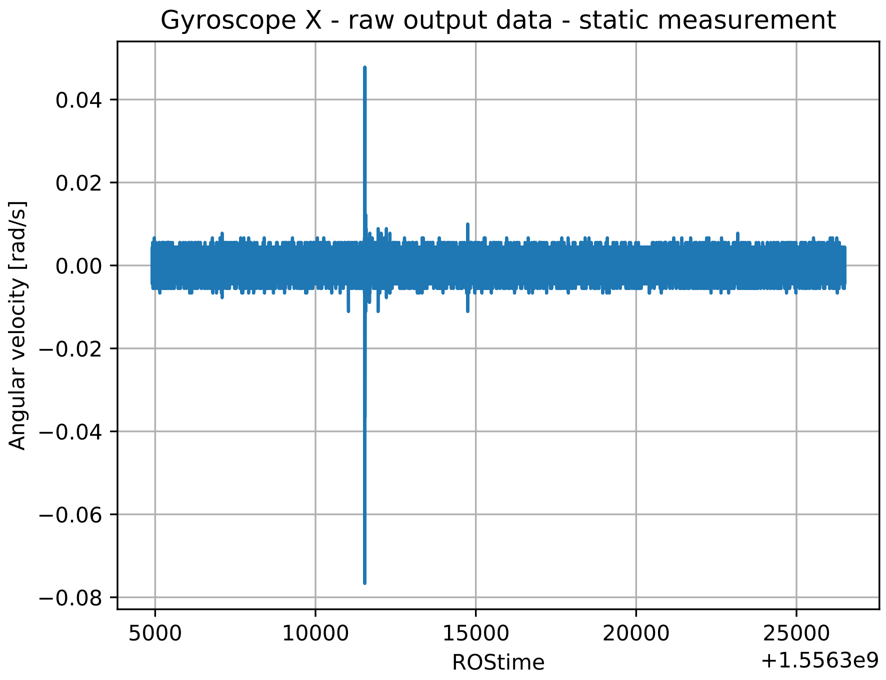


Figure 3.3: Example of raw data with outliers, the ROS time units are nanoseconds

weighted average. The greater the uncertainty of the entry was, the smaller the weight the entry got.

For the accelerometers, the estimated velocity random walk parameter VRW value for all three axes are very similar and they can be also expressed as

$$VRW_x \approx 5.4{\cdot}10^{-3} m/s/\sqrt{3600s} = \frac{5.4 \cdot 10^{-3}}{60} m/s^2/\sqrt{Hz} = 0.00009 \ m/s^2/\sqrt{Hz} \tag{3.23}$$

assuming gravitation constant $g = 9.81\, m/s^2$, VRW units can be converted to noise density format which is also defined in the sensor's datasheet (see selected parameters in table 3.1).

$$VRW_x \approx 9\mu g/\sqrt{Hz} \tag{3.24}$$

$$VRW_y \approx 6.4 \cdot 10^{-3} \ \ m/s/\sqrt{h} \approx 11\mu g/\sqrt{Hz} \tag{3.25}$$

$$VRW_z \approx 7.5 \cdot 10^{-3} \ \ m/s/\sqrt{h} \approx 13\mu g/\sqrt{Hz} \tag{3.26}$$

Also values for gyrosope can also be analogically converted to the white noise densitis. The estimations of the angle random walk parameters ARW are then:

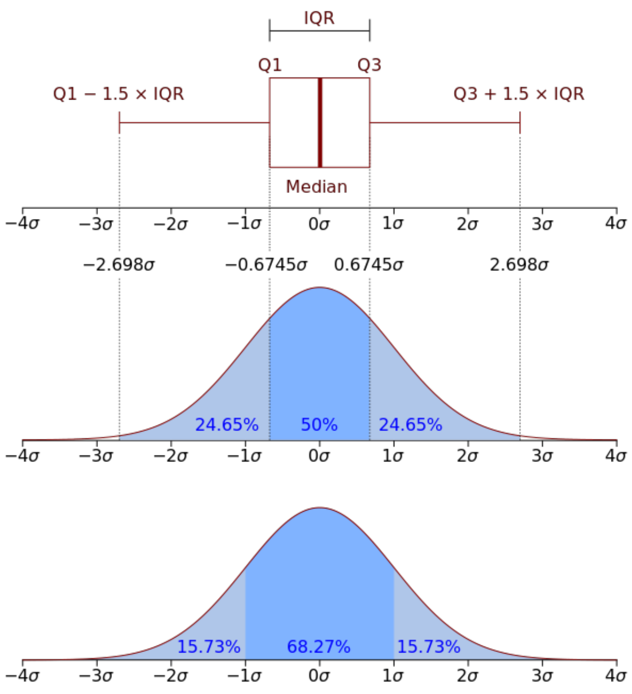


Figure 3.4: The principle of IQR outlier filtering method,[64]

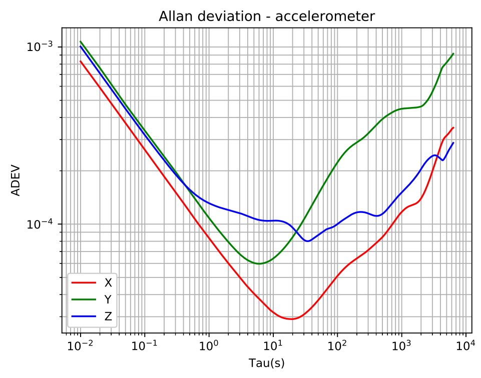
$$ARW_x \approx 0.32°/s/\sqrt{h} = 0.32°/s/\sqrt{3600s} = \frac{0.32}{60}°/\sqrt{s} = 0.00533°/s/\sqrt{Hz} \tag{3.27}$$

$$ARW_y \approx 0.5°/s/\sqrt{h} \approx 0.00833°/s/\sqrt{Hz} \tag{3.28}$$

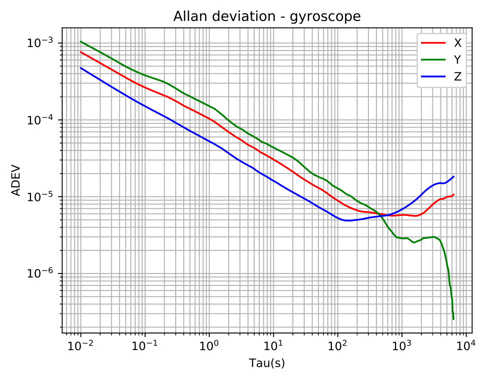
$$ARW_z \approx 0.17°/s/\sqrt{h} \approx 0.002833°/s/\sqrt{Hz} \tag{3.29}$$

The estimated noise parameters are smaller than the values defined in datasheet (see table 3.1). The difference is more distinct for accelerometer, where the found values are roughly 10-times smaller (i.e., better) than the value from the datasheet. For the gyroscope, the difference between estimation and datasheet values is much lower. When taking into account the fact the datasheet values are defined only for the simple sensor's modes and that the datasets used for Allan variance were gathered in the fusion "IMU" mode, which utilizes on-chip noise filtering algorithm, I can conclude that the estimated VRW and ARW parameters are correct and correspond to the reality.

The bias instability of accelerometer expressed in standard industrial format is about $20 - 40mg$. It seems to be a reasonable value for a low-cost IMU. However, when it comes to the gyroscope bias instability estimation, the obtained parameters are extraordinary good for such a low-cost IMU sensor. The estimated value is about $1°/h$ while the usual bias instability for MEMS

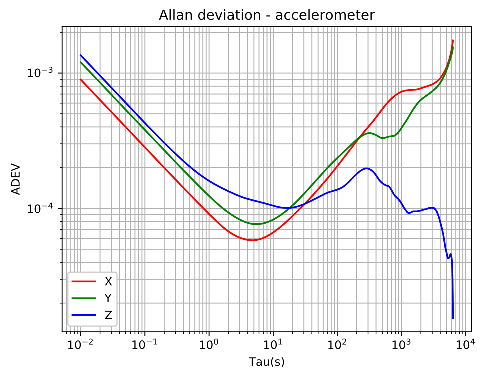


(a) : Accelerometer

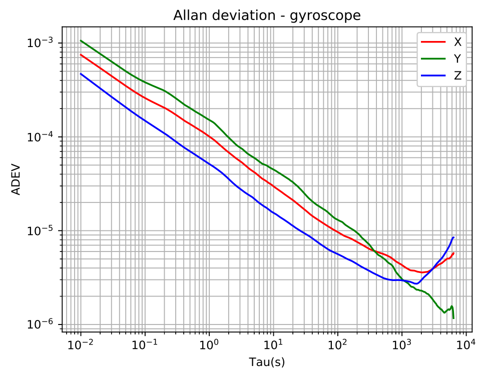


(b) : Gyroscope

Figure 3.5: The Allan deviation graphs - experiment 1



(a) : Accelerometer



(b) : Gyroscope

Figure 3.6: The Allan deviation graphs - experiment 2

IMU sensors in this price category is greater than $5°/h$. Often, this value is equal even to tens of degrees per hour.

That fact that the gyroscope's bias instability parameters estimation is problematic also depict directly the curves in the Allan deviation plots 3.5b, 3.6b, and 3.7b. Instead of having a clear plateau, the deviation often drops unexpectedly which can lead to wrong parameter estimation. To exclude the potential influence of wrong analysis implemented in the script [63], I also doublechecked the parameters by different script [65] in Matlab. Nonetheless, it only confirmed the published results and the curves found during the previous analysis.

Another possible distorting factor may also be the UART bus, which did not work flawlessly. Irregularly, the sensor produced messages with an error flag: `BUS_OVER_RUN_ERROR` instead of sending messages with measured data. This error message presumably indicates that a new message arrives even before the previous message has been read from the UART's receiver buffer [66]. Because of that, I also tried another BNO055 UART ROS driver [67],

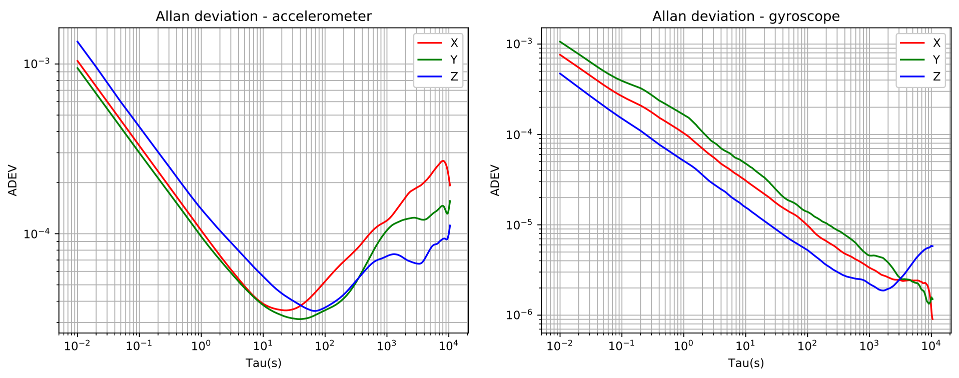
	Velocity random walk [m/s/√h]			
	Measurement	X axis	Y axis	Z axis
Accelerometer	1	0.0049 ± (0.0003)	0.0064 ± (0.0004)	0.0060 ± (0.0004)
	2	0.0053 ± (0.0003)	0.0072 ± (0.0006)	0.0080 ± (0.0006)
	3	0.0063 ± (0.0005)	0.0057 ± (0.0005)	0.0081 ± (0.0005)
	Estimation:	0.0054 ± (0.0003)	0.0064 ± (0.0005)	0.0075 ± (0.0005)
	Bias instability [m/s²/h]			
	Measurement	X axis	Y axis	Z axis
	1	0.1576 ± (0.0043)	0.3237 ± (0.0050)	0.5654 ± (0.0102)
	2	0.3157 ± (0.0043)	0.4152 ± (0.0060)	0.5466 ± (0.0139)
	3	0.1915 ± (0.0045)	0.1701 ± (0.0051)	0.1900 ± (0.0076)
	Estimation:	0.2220 ± (0.0044)	0.2973 ± (0.0053)	0.3971 ± (0.0099)
	Angle random walk [°/√h]			
	Measurement	X axis	Y axis	Z axis
Gyroscope	1	0.3315 ± (0.0020)	0.4846 ± (0.0030)	0.1723 ± (0.0010)
	2	0.3255 ± (0.0020)	0.4826 ± (0.0030)	0.1682 ± (0.0010)
	3	0.3334 ± (0.0160)	0.5162 ± (0.0025)	0.1680 ± (0.0001)
	Estimation:	0.3287 ± (0.0028)	0.4958 ± (0.0028)	0.1683 ± (0.0003)
	Bias instability [°/h]			
	Measurement	X axis	Y axis	Z axis
	1	1.7588 ± (0.3038)	0.8924 ± (0.1848)	1.5149 ± (0.1179)
	2	1.1141 ± (0.3460)	0.7823 ± (0.1978)	0.9223 ± (0.1626)
	3	0.7697 ± (0.2293)	1.4535 ± (0.2448)	0.5834 ± (0.1307)
	Estimation:	1.1729 ± (0.2845)	1.0117 ± (0,2061)	1.0315 ± (0.1346)

Table 3.2: IMU noise parameters estimated based on overlapping Allan variance analysis as estimated with tool [63] and converted to proper units

but the problem persisted. It may be an issue of the used UART/USB converter, but also many other BNO055's users reported a similar issue in various online discussion forums. It seems that the problem is bound solely to UART communication bus and the BNO055 itself. Presumably, this issue is not present on the Adafruit board used in experiments with a mobile robot since it prefers the I2C communication bus. Thus, I assess the gyroscope's bias-stability parameters as untrustworthy, rather best-case, estimation.

It turned out that the tool [63] supports only the detection of ARW/VRW and bias instability noise terms. It is however not a big issue in the view of the weird behavior in the end at the end of Allan deviation graphs. Even so, I tried to estimate at least roughly the rate random walk parameters (RRW) using the Matlab script [65]. The results are summarized in the table 3.3.

These values can be then converted to more common units. For the accelerometer, the resulting rate random walk more or less corresponds to



(a) : Accelerometer **(b) :** Gyroscope

Figure 3.7: The Allan deviation graphs - experiment 3

	Rate random walk [m/s/h/√h]			
Accelerometer	Measurement	X axis	Y axis	Z axis
	1	1.3959 ± (0.1643)	7.0561 ± (0.1380)	1.6995 ± (0.3366)
	2	7.2666 ± (0.0600)	8.7305 ± (0.2047)	x
	3	1.3998 ± (0.1815)	8.1712 ± (0.6371)	0.4332 ± (0.1105)
	Estimation:	4.8584 ± (0.1061)	7.8522 ± (0,2392)	0.7164 ± (0.2396)
	Rate random walk [°/h/√h]			
Gyroscope	Measurement	X axis	Y axis	Z axis
	1	2.8305 ± (0.3333)	x	4.6363 ± (0.2196)
	2	1.5235 ± (0.4467)	x	1.4006 ± (0.2803)
	3	x	x	0.8891 ± (0.1657)
	Estimation:	2.1364 ± (0.6381)	x	2.2234 ± (0.2119)

Table 3.3: The estimation of rate random walk parameters, as find out by the tool [65]

expected values for a MEMS accelerometer.

$$RRW_{A_x} \approx 4.86 \;\; m/s/h/\sqrt{h} \approx \frac{4.86}{3600 \cdot 60} \approx 2.25 \cdot 10^{-5} m/s^3/\sqrt{Hz} \tag{3.30}$$

$$RRW_{A_y} \approx 3.5 \cdot 10^{-5} \;\; m/s^3/\sqrt{Hz} \tag{3.31}$$

$$RRW_{A_z} \approx 0.3 \cdot 10^{-5} \;\; m/s^3/\sqrt{Hz} \tag{3.32}$$

For the gyroscope, the estimation was possible only for two axes, the resulting parameter estimations can be then rewritten as:

$$RRW_{G_x} \approx 2.14°/h/\sqrt{h} \approx \frac{2.14}{3600 \cdot 60} \cdot \frac{\pi}{180} \approx 0.17 \cdot 10^{-6} \;\; rad/s^2/\sqrt{Hz} \tag{3.33}$$

$$RRW_{G_z} \approx 0.18 \cdot 10^{-6} \;\; rad/s^2/\sqrt{Hz} \tag{3.34}$$

Similar to the gyroscope bias instability parameters, these values are smaller by orders of magnitudes compared to a ordinary MEMS low-cost IMU sensors.

While for the accelerometer the estimated parameters correspond by order of magnitude to the expected parameter values, for the gyroscope, only the angle random walk parameter seems to be trustworthy. Better insight into gyroscope's behavior could provide a brand new analysis, ideally based on static datasets with the length of tens of hours, measured in truly defined laboratory conditions using I2C communication bus. But such a kind of noise analysis was neither in the primary scope of this thesis nor technically possible.

3.2 Simulation framework

In this thesis, as a *simulation framework* are called all the scripts created to enable meaningful simulations of IMU sensor in Gazebo simulator. To understand the following framework's philosophy, it is worth to remind that all the resources available during this thesis were only:

- a few datasets with raw data measured by real IMU sensor BNO055 on the mobile robot and corresponding datasets consisting of waypoints measured by precise GPS, as described in previous section 3.1
- the basic URDF/SDF model of the real mobile robot with the solution of its inverse kinematics task

Thus, to be able to simulate the output of the IMU sensor and evaluate its quality concerning the real measurement I had to solve a few additional tasks before I could start to work on actual IMU modeling. So, I placed the following demands on the framework's functionality:

- trajectory tracking of the real robot's trajectory by the robot model in Gazebo simulator
- data acquisition of the simulated IMU sensor
- data post-processing for both simulated and real IMU sensors
- semiautomated IMU output comparison, evaluation, and saving of results

In the following subsection 3.2.1 I first describe the overall structure and high-level design of the framework. Subsequently, implemented solutions of above-stated problems are described more in detail in next subsections. And finally, an extensive user documentation of this framework, including the example configuration file is accessible as a `README.md` in the `imu_simulation_framework` directory on the attached CD (see appendinx B).

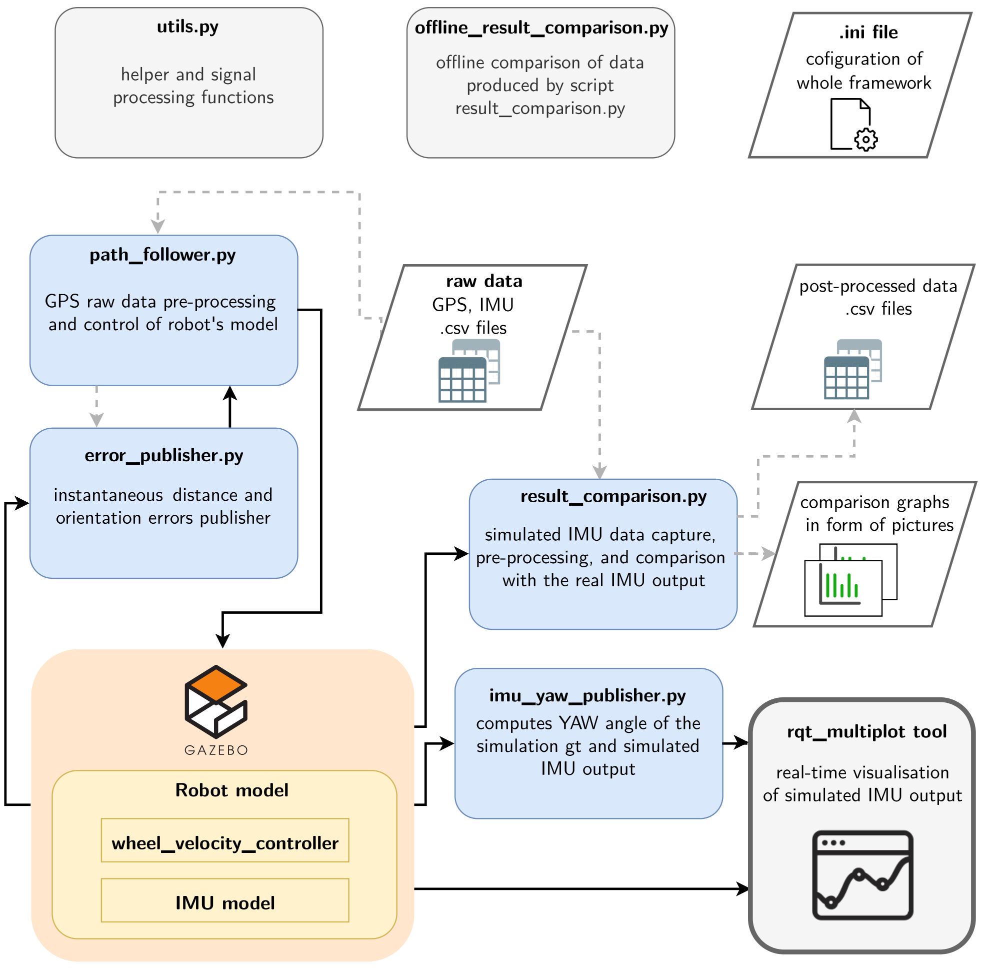

utils.py
helper and signal processing functions
offline_result_comparison.py
offline comparison of data produced by script result_comparison.py
.ini file
cofiguration of whole framework
path_follower.py
GPS raw data pre-processing and control of robot's model
raw data
GPS, IMU
.csv files
post-processed data
.csv files
error_publisher.py
instantaneous distance and orientation errors publisher
result_comparison.py
simulated IMU data capture, pre-processing, and comparison with the real IMU output
comparison graphs in form of pictures
GAZEBO
imu_yaw_publisher.py
computes YAW angle of the simulation gt and simulated IMU output
rqt_multiplot tool
real-time visualisation of simulated IMU output
Robot model
wheel_velocity_controller
IMU model

Figure 3.8: Simplified high-level flowchart of the IMU simulation framework, blue boxes correspond to separate ROS nodes

3.2.1 General framework design

The main idea behind the framework's design was to simulate as good as possible the real experiments with a mobile robot in Gazebo to be able in the end compare the outputs of the simulated and real IMU sensors directly.

For that purpose, I designed the framework as a set of mutually communicating ROS nodes. This way of design ensures good modularity and can be easily extended or modified if needed. It was implemented in the programming language Python 2.7, which is fully supported by ROS and well portable. Also, the Python-based implementation can directly benefit from the rich variety of modules for the data processing, plotting or computations.

Figure 3.8 shows the framework's structure from a high-level point of view. Its operation can be described as follows.

In the beginning, the script `path_follower` starts and loads the raw GPS data from the path defined in the given ini configuration file[2]. According to the user's setting the raw GPS data are pre-processed and converted to the set of waypoints in 2D (i.e., the coordinates $[t_i, x_i, y_i, \theta_i]$, where t_i is the time of reaching the waypoint i and θ_i stands for robot's heading (yaw) angle. Also, the start and stop times stamps of the experiment are identified. The waypoints are then used as an input for the inverse kinematics model, which converts them to the set of wheel velocities, needed to follow the same trajectory in simulation as the real robot did. The waypoints together with computed wheel velocities are stored in temporary file `trajectory_data.csv` which is subsequently loaded by other parts of the simulation framework. Also, a unique file-name prefix including the date and time-related to the start of the simulation is generated py `path_follower` and the start and stop timestamps[3] information are automatically detected. These information are then sent via custom ROS service to `result_comparison` script. Thanks to that all the files generated by `path_follower` or `result_comparison` are stored in a well-arranged way with the same file-name prefix and `result_comparison` can load the corresponding part of IMU raw data from the file path defined in a configuration file. IMU raw data are then automatically truncated to match to the trajectory data and pre-processed. After that `result_comparison` is ready to gather the IMU data.

When the `result_comparison` is ready, `path_follower` sets the robot model in the simulator to the initial robot's position (in agreement with the initial position from the GPS raw data) using ROS service supported by Gazebo simulator and the control loop starts to run.

At this point, according to setting loaded from the configuration file, two different scenarios can happen. If the feedback control is disabled, the `path_follower` simply publishes the computed wheel velocities with correct timing, relying fully on the underlying `wheel_velocity_controller` (see subsection 3.2.2 for details) in Gazebo to control the movement of the robot, which is, in fact, open-loop control. On the other hand, when the feedback control is enabled, the `error_publisher` script comes on the scene. With a certain frequency, it publishes the heading and distance error[4]. Based on that a wheel velocities' "correction" is computed using PID controllers and afterward, the wheel velocities are published. When tuned properly, the feedback control significantly improves the trajectory approximation when the robot in simulator starts to go wrong. The trajectory tracking algorithm and control are discussed in subsection 3.2.2. When the `path_follower` ends, it generates the graph with the history of both angle and distance error, showing the quality of trajectory approximation.

Both `result_comparison` and `error_publisher` subscribe to the ROS

[2]The format of the raw input data required by the simulation framework is described in the user documentation, available on the attached CD.

[3]Start and stop ROS timestamps corresponding to the desired part of GPS raw data do not necessarily correspond the first and last entries of the raw data CSV file. User can set start and stop index in the config file.

[4]Euclidian distance between the reference waypoint' coordinates and the actual robot's coordinates in the simulator.

topic over which the wheel velocities are sent to the `wheel_velocity_controller` in Gazebo. They use this topic for synchronization and trigger their activity (IMU data acquisition of error computation) immediately when `path_follower` starts to publish the wheel velocities.

If no error occurs during the simulation, `path_follower` manages to publish all the precomputed wheel velocities, and immediately after that it stops the robot (send zero wheel velocities) and sends stop request to `result_comparison`. `error_publiser` stops automaticaly when it cannot compute the error anymore (after the time duration of experiment elapses). Also premature shutdown of the experiment is possible by sending manually the `SIGINT` signal[5] to `result_comparison`. In that case, the data captured until this moment are saved and comparison is executed.

According to the settings in the configuration file, several graphs are generated by `result_comparison` in the end of the simulation. For now, the primary output is the comparison of the yaw angles computed from the real and simulated IMU data which lies in the main scope of this thesis. In addition to that also helper functions for trajectory tracking algorithm tuning were implemented, for example, simulated versus desired ground-truth trajectory plotting, or wheel velocities analysis graphs. Following the same general signal processing approach, which was already implemented in general `utils` script, `result_comparison` can be prospectively extended in the future to compare also directly angular velocities or accelerations.

The post-processed simulated IMU output data cropped real IMU raw data, and the actual `wheel_velocities` of the robot model from the simulator, are also saved in the form of CSV files to enable even offline processing. `offline_result_comparison` can load exactly this data and execute the analysis offline, but utterly different tool like Matlab can be used for this purpose too.

For the purpose of real-time visualisation of simulated IMU output the `rqt_multiplot` [68] was used. The ROS node `imu_yaw_publisher` only converts the orientation in `imu` ROS topic from quaternions to yaw angle to directly visualize this angle rather than quaternions.

3.2.2 Trajectory tracking

In a mobile robotics, the *trajectory tracking* describes the problem of following the desired path (typically defined as a set of waypoint) when also fulfilling all the time constraints. The time constraints make it much harder problem to solve compared to the simple path tracking problem where only the following of a given path is of interest regardless of time constraints.

For the IMU model evaluation, the ideal scenarios would be basic defined

[5]This is typically done by pressing `Ctrl+C` in the corresponding console on UNIX operating systems.

paths that could be easy implemented directly in Gazebo. But it was not possible since the real robot was not available for any further experiments. Also, the original control algorithm itself, which would be ideal for mimicking the robot's movement in Gazebo simulator accurately, was not provided. So the trajectory tracking (or at least a good approximation of the original trajectory in the simulator) based on purely GPS raw data was needed to solve.

The provided SDF robot model implements `wheel_velocity_controller`[6]. Because the solution of inverse kinematics was known (see equations 3.18, 3.19) using it to control directly the robot was logical approach.

The computation of wheel velocities was implemented in `path_follower`, in particular in functions:

- `load_reference_trajectory_data` - this function loads the raw data, do the pre-processing (data truncating, interpolation and normalization[7]) it computes the linear velocities as difference between coordinates of consecutive waypoints divided by time difference.
- `compute_wheel_velocities_inverse_kinematic_model` - computes the wheel velocities needed to reach the waypoints in desired time

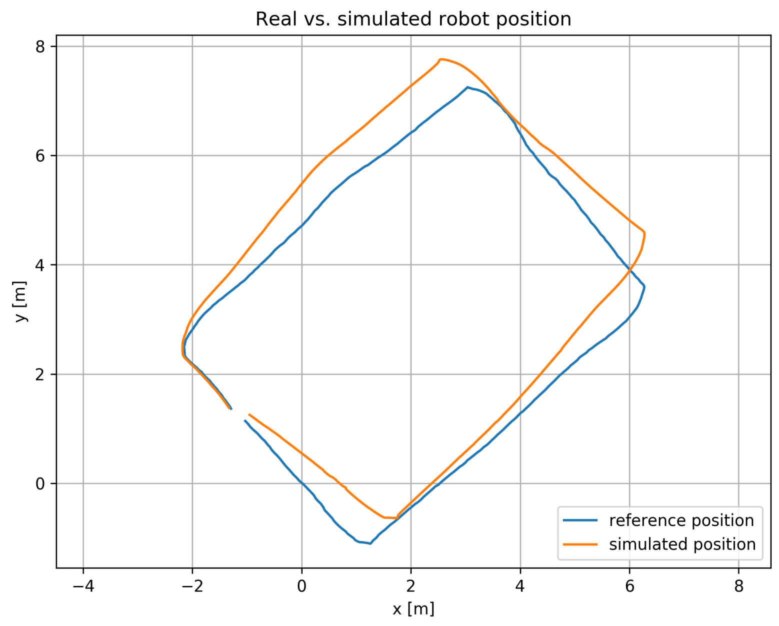


Figure 3.9: Open loop control, path comparison, trajectory A

After first experiments with `wheel_velocity _controller` in Gazebo it turns out though that it does not work perfectly. Even after hours of tuning

[6]ROS PID controllers, separately connected to drive wheels, controlling the wheel velocity and dealing internally with the robot's dynamics.

[7]Make the robot start in the origin no matter what were the original coordinates.

of its underlying PID controllers, it was not working sufficiently. Although it manages to well approximate the speed on straight parts of the trajectory, in the sharp windings the controller failed to make the robot turn in the desired direction fast enough, which was leading to ever-increasing angle and distance error, as depicted in the figure 3.9. In addition to imperfect velocity controller, also the robot model iteself tended steadily to turn a bit in one direction because of small asymmetry of the front wheels. So clearly the naive open-loop control was not sufficient in this case, and different approach had to be found.

Even though there have been some papers published about the differential drive trajectory tracking problem, for example, [69], or [70], therein proposed solutions are based on advanced control techniques such as non-linear control theory. Unfortunately, an implementation from scratch of such a controller assumes a deep understanding of the advanced control theory and does not correspond to the topic of this thesis at all. Because of that, I tried to find an existing ROS/Gazebo trajectory tracking controller that would be easily applicable in my simulation framework. Despite my best effort, I did not find any easy-to-use solution that would directly co-operate with `wheel_velocity_controller`, which has to be kept due to backward compatibility with the rest of the whole project.

So finally I ended up with the assumption that the precise trajectory tracking is not strictly necessary to be able to say something about the quality of simulated IMU outputs (see subsection 3.2.3 for details about data comparison and IMU model evaluation) and I designed my own control solution to improve insufficient behavior of Gazebo velocity controller.

To improve the behavior of `wheel_velocity_controller` I designed and implemented a custom feedback PID control loop that ad hoc slightly increase or decrease the wheel velocities to improve the turning behavior. The following Python code snippet shows the simple yaw angle control, as implemented in function `control_wheel_velocities` in `path_follower`. Note that variable `self.yaw_error` contains the signed difference between the reference and simulated yaw angle and thus can be used to determine the turning direction. Its value is periodically computed by `error_publisher` published via custom ROS topic.

```python
# I part
self.E_yaw = self.E_yaw + self.yaw_error
# D part
ed_yaw = self.yaw_error - self.yaw_error_old
self.yaw_error_old = self.yaw_error

# PID controller
u_yaw = np.abs(self.yaw_error * self.Kp_yaw +
    self.E_yaw * self.Ki_yaw + ed_yaw * self.Kd_yaw )
```

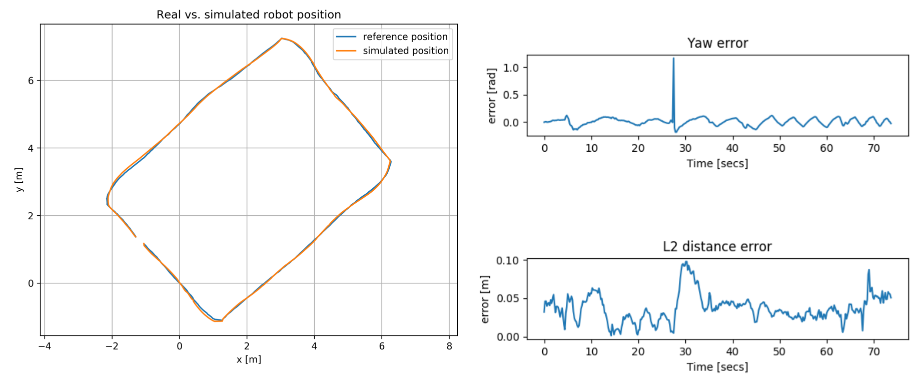
```
if (self.yaw_error < 0): # turn right
    velocity_left = velocity_left + u_yaw
    velocity_right = velocity_right - u_yaw
else:                     # turn left
    velocity_left = velocity_left - u_yaw
    velocity_right = velocity_right + u_yaw
```

In addition to the yaw controller, I designed and also implemented a "distance" PID feedback controller, which converts the Euclidian distance error to the similar correction of wheel velocities. This controller is coupled after the yaw controller, and the main idea is to turn the robot even a bit more in the direction to the reference waypoint. The turning direction is computed as the angle between the vector among coordinates of the robot's current position and the reference waypoint with respect to the current robot's yaw angle.

Experimental results

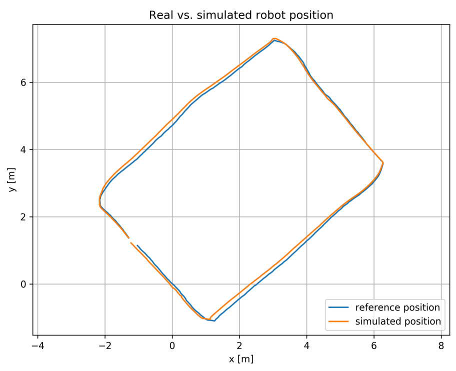
The experiments shows that the feedback control improves the trajectory tracking approximation significantly.

Both distance and yaw angle control provide the best results in the sense of the smaller average distance and yaw angle errors (see figure 3.10), but when it comes to the yaw angle approximation only, the distance PID controller introduces an artificial yaw error - small oscillations that are unwanted with respect to the fact that yaw angle is the primary quantity of comparison (see yaw error in the figure 3.10b). Thus, best results can be achieved by yaw angle PID controller only, when properly tuned (see figure 3.11). Therefore for IMU simulation, this approach was used. The constants of PID controllers can be set in the configuration file.

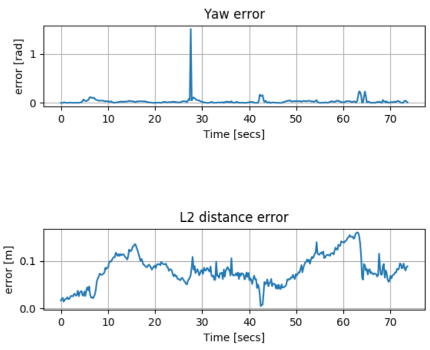


(a) : Path comparison **(b) :** Control errors

Figure 3.10: Feedback PID control - yaw + distance control, trajectory A



(a) : Path comparison



(b) : Control errors

Figure 3.11: Feedback PID control - yaw control only, trajectory A

3.2.3 Data comparison

Some comparison approach is needed to evaluate the quality of the implemented IMU models, with emphasize to the yaw angle estimation.

For the purpose of comparison approach description, let assume two signals describing the robot's yaw angle over time. The ground truth signal $g(t)$ is given by either GPS in the case of provided raw data $g_{real}(t)$, or is provided internally by Gazebo simulator as a ground truth signal $g_{sim}(t)$. The estimated yaw angle signal $f(t)$ is the angle estimation by either real IMU sensor BNO055 $f_{real}(t)$ (where yaw angle is estimated internally on chip by proprietary algorithm) by the IMU model in ROS/Gazebo $f_{sim}(t)$, which is in detail described in later section 3.3.

In general, the comparison of two random signals is always a challenging task. Because of the randomness, there is no simple deterministic metric which could be directly applied to evaluate the similarity of signals, and the comparison should be made on a statistical basis. For that kind of analysis, many repetitions of both random signals must be available, the more the better. Then various types of statistic measures can be applied to both group of signals, and these can be then compared to each other and evaluate the amount of similarity.

In this thesis, I face a much harder situation. Only a ver7 limited number of datasets with raw IMU data was provided, and additional experiments with the sensor and mobile robot were not possible. Besides, most of the datasets were measured exclusively for different trajectories. There is was no bigger set of IMU measurements done for one single trajectory. Because of that, a truly reliable statistical comparison of the real IMU sensor BNO055 with IMU model in Gazebo is not possible. But still, some similarity measure is needed to compare the $g(t)$ with $f(t)$, which would permit at least some kind of comparison of the IMU model with the real IMU sensor, based on the limited datasets available.

One of the most frequently used time-domain signal similarity measures is a cross-correlation function. Unluckily, it does not make much sense in this case. It is a reasonable similarity measure to find some pattern in the noisy signal but does not consider the similarity of the nature of noises, which is desired in this case.

Another simple metrics is a signal difference. It is well known that in short-term operation even a low-cost IMU sensor usually provides accurate angle estimation. The main subject of interest is the long-term behavior. And here can the simple signal difference shed some light on the trend of error-growing process. When plotted graphically it shows directly how the yaw estimation error evolves. It mirrors the sum of all underlying noises and error causes in time domain. Formaly, the yaw estimation errors can be defined as

$$e_{real}(t) = |g_{real}(t) - f_{real}(t)| \tag{3.35}$$

$$e_{sim}(t) = |g_{sim}(t) - f_{sim}(t)| \tag{3.36}$$

for real IMU and simulated IMU respectively.

In this thesis, the desired state is to see similar yaw estimation error in the simulation as for real measurements, in other words

$$|e_{raw}(t) - e_{sim}(t)| \approx 0 \tag{3.37}$$

Since noises are random, the effort to have a precise matching of $e_{raw}(t)$ and $e_{sim}(t)$ does not make much sense. But what is desired is a similar long-term error trend of both simulated and real IMU sensors in a statistical sense.

For trend extracting, the median filtering can be used. The *median filter* is a non-linear filter, which computes for each discrete time step of the signal the value as a median over the window size which can be formally defined as

$$y(n) = med[x(n-k), x(n-k+1), \dots, x(n+k)] \tag{3.38}$$

where $x(n)$ is a discrete input signal, k is a window size, and $y(n)$ is output filtered discrete signal [71].

In this way the comparison was implemented in function `plot_yaw_difference`, the signal processing functions were implemented as general purposes functions in module `utils`. When the signals are differently sampled, both signals are firstly interpolated, and then their difference is computed and plotted. Optionally, the function can try to make a precise time alignment of the yaw estimation and corresponding ground truth signals based on cross-correlation.

To see "under the surface" of the frequency-domain, I also implemented the automated plotting of PSD function. For that purpose, different methods can be used. I used Welch's method, which provides a good trade-off between the frequency resolution and variance of PSD estimation and belongs to the non-parametrical methods (does not need a priori knowledge about the signal). The Welsch's method can in a simplified way described as follows [72]:

1. Split the random signal into partially overlapping segments.
2. For each segment estimate the periodgram[8]
3. Get the final PSD estimation as average over the local estimates.

In the simulation framework, I used Welsch's method algorithm available in the module `scipy.signal` and implemented it in the function `compute_psd_welch` in `utils`, which automatically sets the biggest possible segment size (`nperseg` parameter). The `result_comparison` script then automatically plots the PSD function graph in log-scale for yaw angle output $f_{sim}(t)$, $f_{real}(t)$, and also for the yaw angle errors $e_{sim}(t)$ and $e_{real}(t)$.

3.3 IMU model

The modeling of IMU sensors in ROS/Gazebo world is done on the input-output basis. Rather then simulation of the partial components of the sensor itself, which would be more computation demanding, Gazebo takes the ground truth angular velocities and accelerations computed by the physics engine and "corrupt" them desirably by IMU noise model to simulate the real IMU behavior.

In this thesis, the Gazebo 9.0.0. was used with the default physic engine ODE (Open Dynamics Engine) which is probably the most commonly used physic engine among the robotic simulators [35]. On the lowest level, the accelerations and angular velocities (computed as a rate of change of the orientation in three axes) are part of the state vector of every tangible body simulated in ODE.

In every iteration of the ODE simulation loop, the forces are applied to the simulated bodies, that have a defined weight and consist of several parts connected with joints that introduce mechanical constraints. Based on known physics laws such as Newton mechanic laws the resulting forces are computed and virtually applied to the bodies. Since the mass of the body is know, the acceleration and angular velocity vectors can be updated.

Gazebo simulator supports the majority of conventional sensors including IMU as a part of its core functionality, which is tightly bound with the SDF model-description language. In SDF, there are special tags for all currently supported sensors. So theoretically, it could be sufficient to define the sensor in the robot's SDF file and set its properties there according to SDF documentation [56].

[8]Estimation of the spectral density of the signal, commonly computed by FFT algorithm.

Apart from the core Gazebo sensor support, there is also the possibility of using Gazebo plugins written in C++ to model the sensor's behavior. Gazebo plugins have direct access to almost all Gazebo functionalities via C++ API, and thus they can even completely take over the simulations of sensor's functionality since they have the full access to the physics engine. Compared to SDF sensor models only, plugins offers better modularity and customization options and can be easily connected to ROS.

So in Gazebo simulator, all sensors are ordinarily implemented also as plugins publishing the standard ROS messages, even though the plugins are often more or less only wrappers for the core sensors' support.

I implemented the model of IMU sensor as a small box with a negligible mass in `xacro` file[9]. The following code snippet shows the simplified[10] implementation in such a way that the Gazebo plugin with the logic of IMU sensor itself can be easily changed by replacing the contents of `insert_imu_gazebo_part` macro in file `imu_sensor.gazebo.xacro`. This code snippet assumes that the robot has a link called `world_link` to which is the IMU sensor attached. This link is also then used to compute the appropriate accelerations and angular velocities by the physics engine. Macro `box_inertial` computes the corresponding values of inertia tensor for the IMU model automatically, according to known formulas for box shape [73].

```
<?xml version="1.0"?>
<robot name="imu" xmlns:xacro="http://www.ros.org/wiki/xacro">

 <xacro:include    filename="$(find urdf)/sensors/
 components/imu_sensor.gazebo.xacro"/>

  <link name="imu_link">
     <visual>
        <origin xyz="0 0 0.4" rpy="0 0 0 "/>
        <geometry>
             <box size="0.08 0.04 0.005"/>
        </geometry>
        <material>
          <color rgba="0 0 0.8 0.5"/>
        </material>
    </visual>

    <collision>
       <origin xyz="0 0 0" rpy="0 0 0"/>
       <geometry>
            <box size="0.08 0.04 0.005"/>
```

[9]`xarcro` is an XML-like macro language which helps to keep the URDF/SDF files in a well-arranged way.

[10]The variables were replaced by hard-coded values in this code snippet for better readability.

```
      </geometry>
    </collision>

    <xacro:box_inertial mass="0.002" length="0.08"
      width="0.04" height="0.005"  />
  </link>

  <joint name="joint_imu" type="fixed">
      <axis xyz="0 0.0 0.0"/>
      <origin xyz="0 0 0 " rpy="0 0 0 "/>
      <parent link="world_link" />
      <child link="imu_link" />
  </joint>

  <xacro:insert_imu_gazebo_part/>
```

3.3.1 Noise modeling

To simulate both the white noise and random walk (bias drift), the following simple noise model can be considered [74].

Let $\widetilde{\Omega}[k]$ is the discrete output of the real gyroscope. Then the gyroscope model can be described as

$$\widetilde{\Omega}[k] = \Omega[k] + b[k] + n[k] + \text{initial offset} \tag{3.39}$$

where the $\Omega[k]$ is the ground truth angular velocity, $b[k]$ is current bias drift (random walk term) and $n[k]$ is a additive white noise term.

For the additive white noise model then holds:

$$n[k] = \sigma_{gd} w[k], \quad w[k] \sim \mathcal{N}(0,1) \tag{3.40}$$

$$\sigma_{gd} = \frac{\sigma_g}{\sqrt{\Delta t}} \tag{3.41}$$

where σ_g is the ARW coefficient, and Δt is the sampling time.

A simple random walk model can be defined as:

$$b[k] = b[k-1] + \sigma_{bgd} w[k] \tag{3.42}$$

$$\sigma_{bgd} = \sigma_{bg} \sqrt{\Delta t} \tag{3.43}$$

where σ_{bg} is corresponding to the gyroscope rate random walk parameter.

In this way, the noise can modeled independently for all gyroscopes and acceleroemters (for all three axes). There are also more complex approaches to model the IMU noise, published for example in [75], the stochastic modeling theory is provided in comprehensive handbook [76].

3.3.2 Available IMU plugins

To my best knowledge, until now, four different open source ROS/Gazebo IMU plugins were created and published.

GazeboRosImu

The official Gazebo documentation mentions two ROS/Gazebo IMU plugins [77]. The first one `GazeboRosImu` is probably the oldest IMU plugin in Gazebo/ROS world which was created by the ROS community in 2003 [78]. This plugin access directly the physics engine interface and the IMU measurements are computed directly by ROS and not Gazebo.

The plugin offers only basic features. Gravity is not included in the accelerometer's output, and the only noise model is additive while noise that is described by a single standard deviation parameter that is then applied identically to all output values. It can be considered as a deprecated solution now.

Gazebo generic IMU plugin

The second officially mentioned Gazebo plugin is called `GazeboRosImuSensor` and inherits more logically directly from Gazebo `SensorPlugin` class. Compare to the previous solution, the gravity is included in accelerations, and Gazebo handles the computation. This plugin is a wrapper for the core Gazebo IMU support implemented in `gazebo/sensors/ImuSensor.cc`. The sensor position, reference frame, fixed offsets and level of white noise can be set.

The underlying Gazebo core IMU implementation supports an extended white noise model. For each accelerometer's or gyroscope's axis the additive white noise can be defined independently as a standard deviation and mean value. The white noise can also be set to the bias in the sense that a spcified amount of white noise is added to the set offsets when the IMU model is loaded, which corresponds to the bias repeatability, but the bias random walk is not simulated. Also, for now, no noise is applied to the orientation IMU output at all [47]. However, it is likely that the future of random walk modeling (i.e., low-frequency bias drift) will appear in the future Gazebo release since there is an open pull request to integrate this feature [79].

The IMU model based on this plugin was implemented as `imu.xacro` file. The corresponding Gazebo part is to be found directly in the official documentation [77].

RotorS IMU plugin

A custom IMU plugin was developed by the team from Autonomous Systems Lab at ETH University in Zürich [80] as a part of RotorS project. RotorS is a MAV[11] Gazebo simulator, a complex simulation environment, that also

[11]Micro air vehicle - remotely controlled small air vehicles such as drones

includes various MAV models, worlds definitions etc. Whole RotorS simulator is well documented on its website [81].

According to the source code and documentation, this plugin has clearly the best noise model among all available plugins. It has more complex random walk model than the one described by equations 3.42, 3.43. It models the random walk as a low-frequency drift with drift frequency $f_d = \frac{1}{\tau_d}$ according to the approach described in [76]. In particular, the bias drift is implemented there as:

$$b[k] = \varphi_{gd} b[k-1] + \sigma_{bgd} w[k] \tag{3.44}$$

$$b[k] = e^{\left(-\frac{\Delta t}{\tau_d}\right)} \cdot b[k-1] + \sqrt{-\sigma_{bg} \cdot \sigma_{bg} \cdot \frac{\tau_d}{2} \cdot \left(e^{\left(\frac{-2\Delta t}{\tau_d}\right)} - 1\right)} \cdot w[k] \tag{3.45}$$

which can be rewritten as

$$b[k] = e^{-f_d \Delta t} \cdot b[k-1] + \sigma_{bg} \sqrt{-\frac{1}{2f_g} \cdot (e^{-2f_d \Delta t} - 1)} \cdot w[k], \tag{3.46}$$

where Δt is sampling time, and τ_d is correlation time of the low-frequency drift.

This plugin uses a custom format of produced messages that can be converted to standard ROS topics by a helper plugin `gazebo_ros_interface_plugin`.

Nonetheless, even my best effort, I was not able to make this plugin work. Even though Gazebo simulator built the resulting SDF model successfully, all dependencies were fulfilled, and Gazebo registered both the sensor and helper plugins, no ROS messages were produced at all. In the official project's website, another two Gazebo users announced the same issue with this plugin in a bug reporting system, but the issue thread is still without any reply from the developer team [82]. By all accounts, it seems that although the RotorS sensor Gazebo plugins are available in a stand-alone Ubuntu package `ros-melodic-rotors-gazebo-plugins`, they were never meant to be used outside the RotorS framework.

Hector IMU plugin

Another custom plugin was implemented by Johannes Meyer from Hector team at TU Darmstadt in 2012.

Similarly to the previous RotorS plugin, it also implements the low-frequency drift in addition to the white noise. What makes this plugin unique are custom ROS services that enable online calibration and reconfiguration of plugin's parameters, and also the yaw angle has separate noise model including the low-frequency drift which is an especially exciting feature concerning this thesis primary goal.

On the other hand, beside a brief description at ROS wiki [83], no additional documentation or support is provided and putting the plugin into operation was quite tricky.

Keeping the same notation as in previous plugin, the bias drift is computed by formula:

$$b[k] = e^{-f_d \Delta t} \cdot b[k-1] + \Delta t \cdot w_g[k], \tag{3.47}$$

where

$$w_g[k] \sim \mathcal{N}(0, \;\; \sigma_{drift}\sqrt{2f_d}) \tag{3.48}$$

According to [83], the σ_{drift} is a standard deviation of the drift error. Unfortunately, the reference or derivation of the equation above is provided neither in the documentation nor in the source code. From comparison of equations 3.46 and 3.47 it can be seen that even though the formula is similar, the σ_{drift} parameters does not directly correspond to the rate random walk parameter σ_{bg}.

This plugin was implemented as `imu_hector.xacro`, with the following Gazebo part:

```
<?xml version="1.0"?>
<robot name="imu" xmlns:xacro="http://www.ros.org/wiki/xacro">

<xacro:macro name="insert_gazebo_part_imu">
 <gazebo>
  <plugin filename="libhector_gazebo_ros_imu.so"
           name="hector_imu_plugin" >
    <alwaysOn>true</alwaysOn>
    <bodyName>imu_link</bodyName>
    <topicName>imu</topicName>
    <serviceName>/imu/calibrate</serviceName>
    <gaussianNoise>0.0</gaussianNoise>
    <updateRate>100.0</updateRate>
    <frameName>imu_link</frameName>

    <!-- Accelerometer config, 3 values are corresponding
     to x,y,z axis respectively -->
    <accelOffset>0 0 0 </accelOffset>
    <accelGaussianNoise>0 0 0</accelGaussianNoise>
    <accelDrift>0 0 0 </accelDrift>
    <accelDriftFrequency>0 0 0</accelDriftFrequency>

    <!-- Gyroscope config -->
    <rateOffset>0 0 0</rateOffset>
    <rateGaussianNoise> 0 0 0</rateGaussianNoise>
    <rateDrift>0 0 0 </rateDrift>
    <rateDriftFrequency> 0 0 0</rateDriftFrequency>

    <!-- Separate error model for YAW angle -->
    <yawOffset> 0 </yawOffset>
    <yawGaussianNoise> 0 </yawGaussianNoise>
    <yawDrift> 0 </yawDrift>
    <yawDriftFrequency> 0 </yawDriftFrequency> -->

  </plugin>
 </gazebo>
</xacro:macro>
</robot>
```

To keep this code snippet simple, all the parameters are equal to zero. In the real experiments, the parameters were set online by the `result_comparison` script where the custom setter function was implemented.

Chapter 4
Experiment results

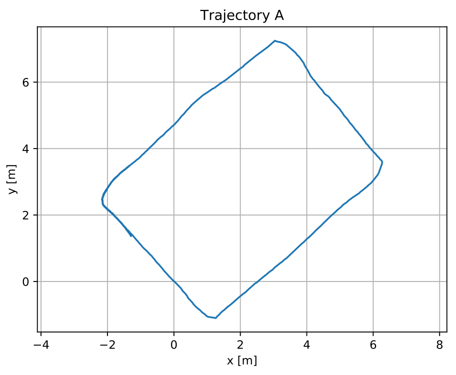
In this chapter, I describe the simulations done to evaluate the implemented Gazebo IMU models described in the previous chapter. For the comparison, I selected four different trajectories with time duration of approximately 10 minutes of robot's driving, which is long enough to see the effects of bias drift. The paths (trajectories[1]) corresponding to the used datasets are depicted in figures 4.1.

In the following text, I tag all data or results corresponding to the experiments with the real mobile robot and real IMU sensor BNO055 as *real*, on the other hand, tag *simulated* corresponds to data or results related to simulations performed in Gazebo and to the IMU model. The trajectories were approximated in Gazebo 9.0.0. and ROS Melodic by the feedback control algorithm described in section 3.2.2 when only the yaw angle control was used. I performed all simulation at laptop Lenovo X201, Intel Core i5 CPU M540, 4GB RAM, Intel HD graphics, with the operating system Linux Mint 18 OS. Because of older hardware setup of the used computer, the simulations were run in a mode without visualization (without `gzclient`).

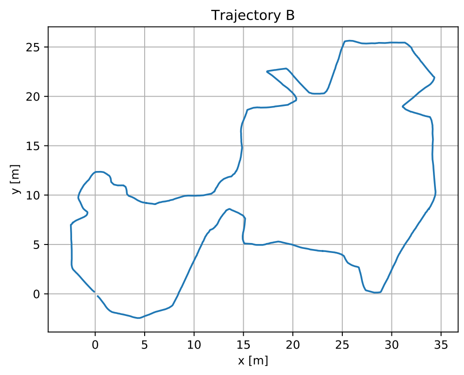
Before the simulations were made, several reductive assumptions about the provided raw data were accepted. Namely, I worked on the following assumptions:

- The terrain imperfections can be considered as insignificant, and the robot's movement can be simulated on an ideally flat surface in the simulator.
- There were no significant mechanical stresses and vibrations caused by external sources that influenced the robot and the IMU output.
- No temperature and voltage drift occurred in the IMU sensor during real experiments.
- The ambient temperature did not vary among different experiments and was similar to common room temperature.

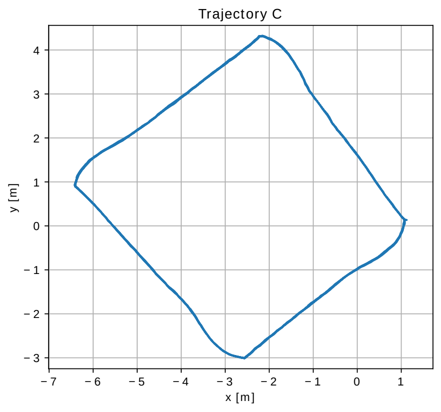
[1]Every dataset includes a trajectory description - set of waypoints from GPS and corresponding raw IMU measurements. In this chapter, I call the datasets simply "trajectories".



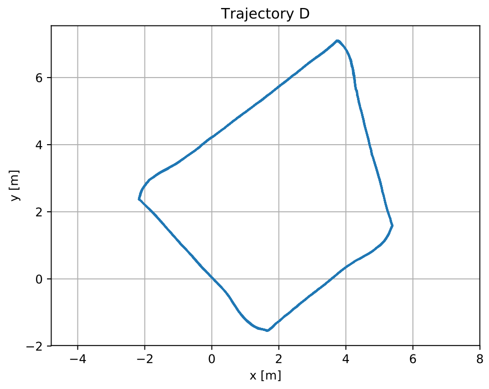
(a) : Trajectory A, followed iteratively



(b) : Trajectory B, followed only once



(c) : Trajectry C, followed iteratively



(d) : Trajectory D, followed iteratively

Figure 4.1: Testing trajectories used in experiments

These assumptions are somewhat optimistic, but necessary to accept due to the lack of information about the provided datasets.

4.1 Gazebo generic IMU plugin

I done the first set of experiments with generic Gazebo IMU plugin (see subsection 3.3.2), which is a standard solution recommended in the official Gazebo tutorial. This plugin implements only the white noise model, which is described by one standard deviation parameter σ_{white}.

The average values of white noise parameters found by Allan variance analysis over all axis are approximately (see subsection 3.1.3):

$$VRW \approx 7 \cdot 10^{-4} \; m/s^2/\sqrt{Hz} \tag{4.1}$$

$$ARW \approx 5 \cdot 10^{-2} \; °/s/\sqrt{Hz} \tag{4.2}$$

Unfortunately, there is no one-to-one matching of these parameters to the parameters of this IMU plugin. The above values correspond in fact to PSD function, which is a function of frequency (bandwidth). If the used sensor's bandwidth is known, the VRW and ARW parameters can be approximately calculated to RMS noise, which can be then considered as a standard deviation of the white noise process [84].

Here the used bandwidth is not known, because the BNO055 IMU sensor set it automatically in the "IMU fusion mode." To get at least a rough estimation, I assumed that bandwidth was 500Hz for accelerometer and 116Hz for the gyroscope. These values are in the middle of possible bandwidth sizes according to datasheet [59].

$$\sigma_{accel} \approx 7 \cdot 10^{-4} \cdot \sqrt{500} \approx 0.017 \;\; m/s^2 \tag{4.3}$$

$$\sigma_{gyro} \approx 0.05 \cdot \sqrt{116} \approx 0.06 \;\; {}^{\circ}/s \approx \frac{\pi}{180} \cdot 0.05 \cdot \approx 0.00105 \;\; rad/s \tag{4.4}$$

Nonetheless, there is only a one single standard deviation parameter that is applied directly also to the orientation output in the form of quaternions, which is of the main interest.

Since the Allan noise analysis did not reveal the white noise directly for the orientation output and the particular scheme of internal data fusion algorithm of BNO055 is unknown, I assumed that mainly the gyroscope contributes to the yaw angle estimation. The angular velocities including the corresponding amount of white noise are integrated and iteratively applied to the resulting position. Thus, I set initially σ_{white} as:

$$\sigma_{white} = 0.00105 \;\; rad/s \tag{4.5}$$

4.1.1 Trajectory A

Experiment 1 - short term experiment

Firstly, I let the model follow only one iteration of the trajectory A, which took about 1 minute long.

The yaw comparison in figure 4.2 shows that for the short-term experiment, the simulated output matches the real signal very well. However, in the closer look to the estimated yaw angle (figures 4.4, 4.5), it can be seen that even after only one minute, a small bias drift of real yaw estimation occurred, while the simulated yaw estimation overlaps the ground truth simulation signal almost entirely, except for small amount of white noise.

The maximal real yaw angle estimation error was about 1.5°, which is an almost negligible error when assuming short-term operation of the mobile robot. The figure 4.4 also shows that in reality, the output from the real BNO055 is sampled on much lower frequency than 100Hz. It may be the attribute of the internal data fusion algorithm, or the data were downsampled as a part of post-processing during real experiments.

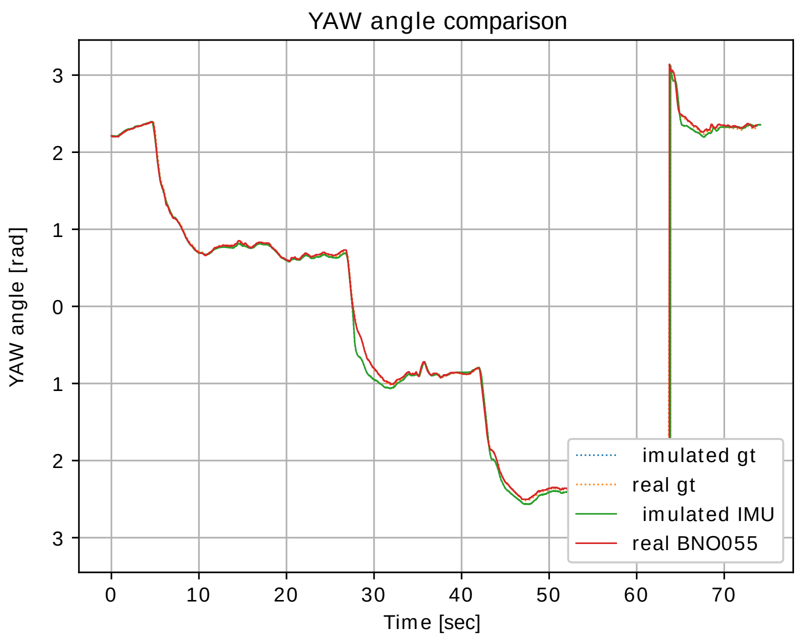


Figure 4.2: Trajectory A, short-term experimet, yaw angle comparison

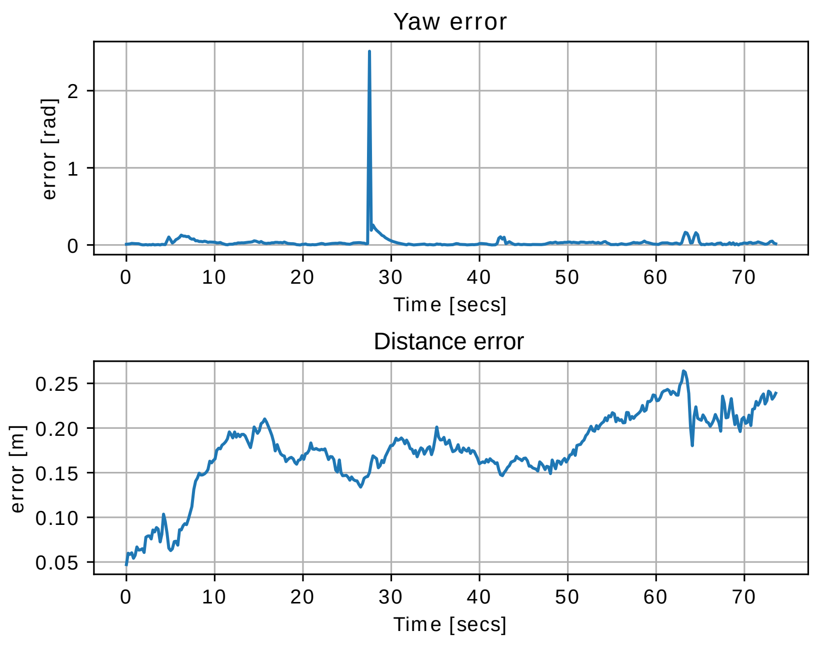


Figure 4.3: Trajectory A, short-term experimet, control (trajectory approximation) error

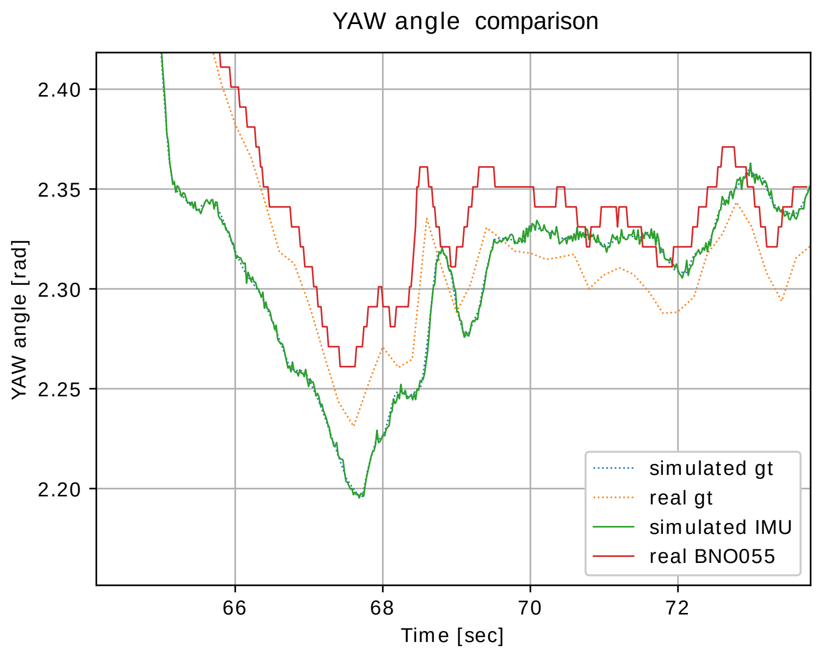


Figure 4.4: Trajectory A, short-term experimet, detail of yaw angle outputs after one minute

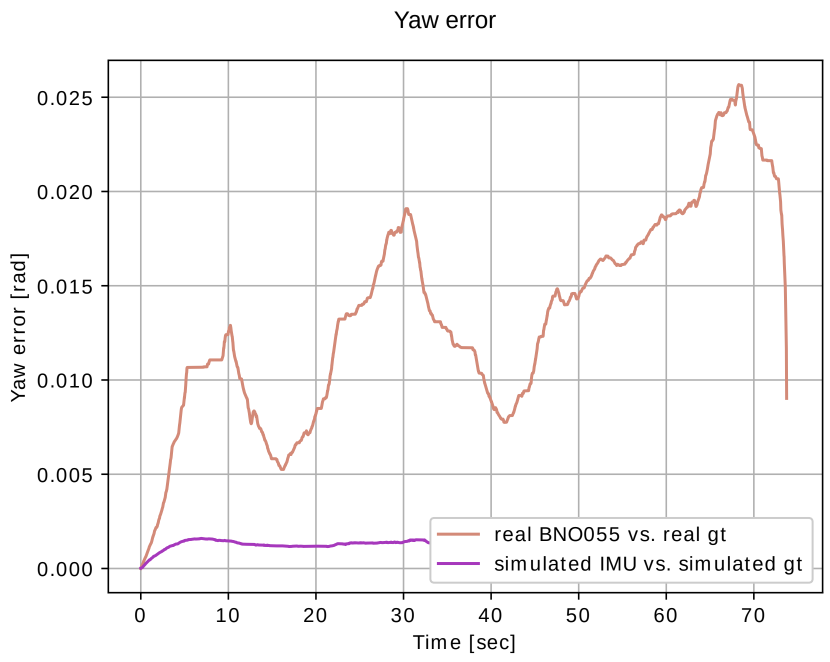


Figure 4.5: Trajectory A, short-term experiment, yaw error, smoothed by median filter with a window size of 101 samples

Experiment 2 - long-term experiment

Based on the previous detailed look at the signal, I doubled the value of σ_{white} for the further experiments:

$$\sigma_{white} = 0.00215 \;\; rad/s \tag{4.6}$$

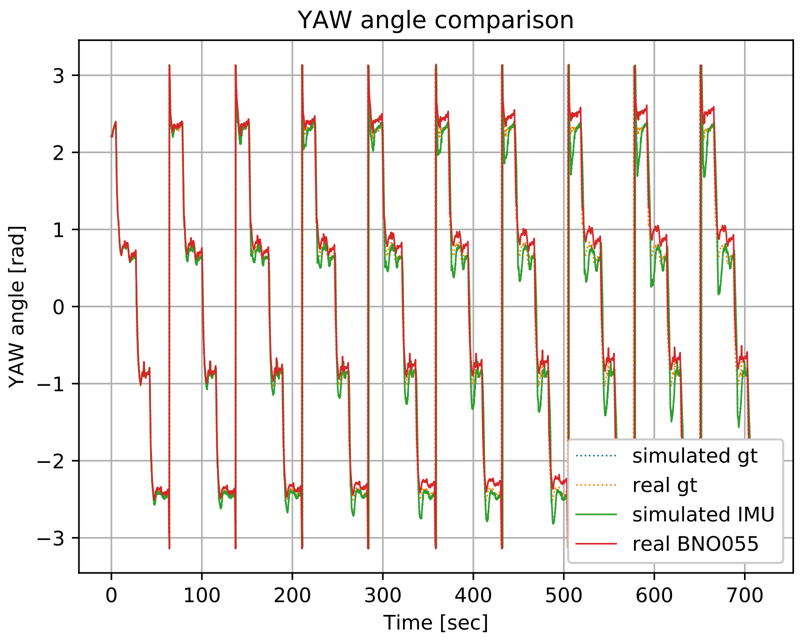


Figure 4.6: Trajectory A, long-term experiment, yaw angle comparison

The figures 4.6, 4.7, 4.9 show that with inceasing time, the real yaw angle signal undergoes steadily growing bias drift leading to growing error of yaw angle estimation. In the end of this experiment, the real yaw angle estimation error is about 0.25 $rad \approx 15°$, which is already significant error for any navigation purposes.

Figure 4.8 reveals a limited functionality of the designed simple trajectory tracking approximation realized by the PID yaw control only in long-term scale. One can see the growing yaw control error with unwanted oscillations in sharp windings. The main reason is an increasing delay, which leads to imperfect timing of the control loop. This delay is also evident in figure 4.7, where the peaks on the simulated signal (i.e., the place where the robot turned) are delayed by a few seconds compared to real signal.

However, the trajectory tracking imperfection does not play a big role in IMU model evaluation, since the simulated IMU output is always compared to the simulated ground truth and afterward the yaw errors are compared to each other. In this experiment, I also plotted the power spectral density functions to see the behavior in the frequency domain. The parameter "nperseg" in the graphs stands for the number of samples per one segment in the Welsch's

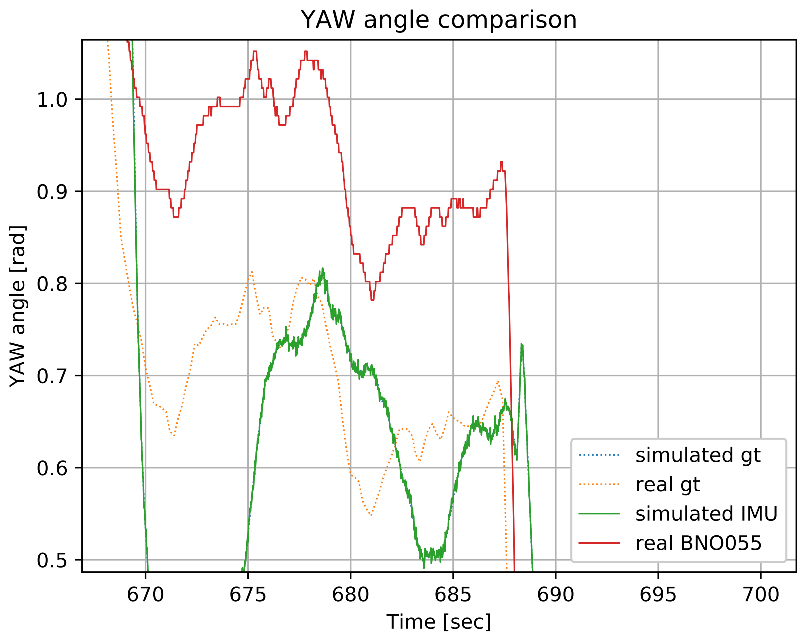


Figure 4.7: Trajectory A, long-term experiment, detail of yaw angle outputs after approximately 11 minutes

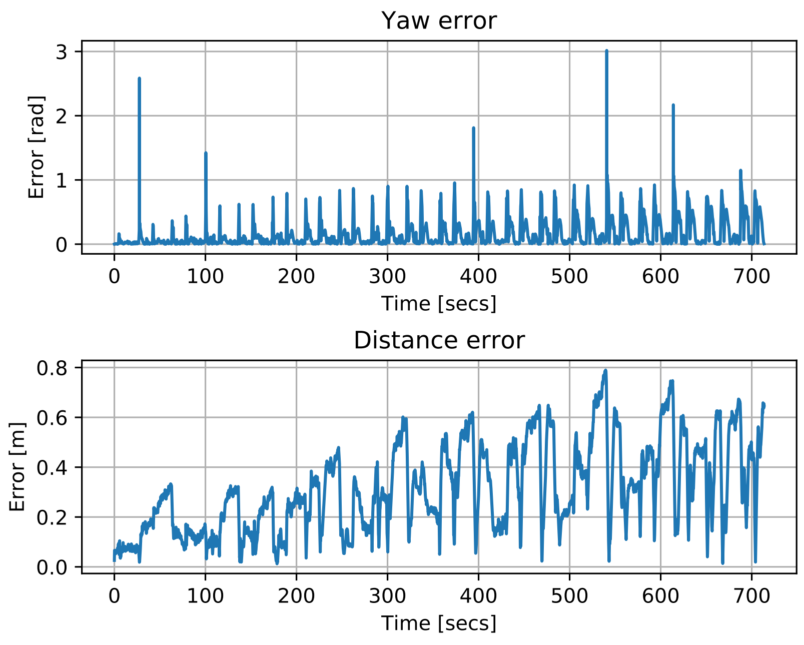


Figure 4.8: Trajectory A, long-term experiment, control errors

PSD estimation algorithm. The PSD functions of both simulated and real yaw signals look very similar (see figure 4.10), even though from the time-

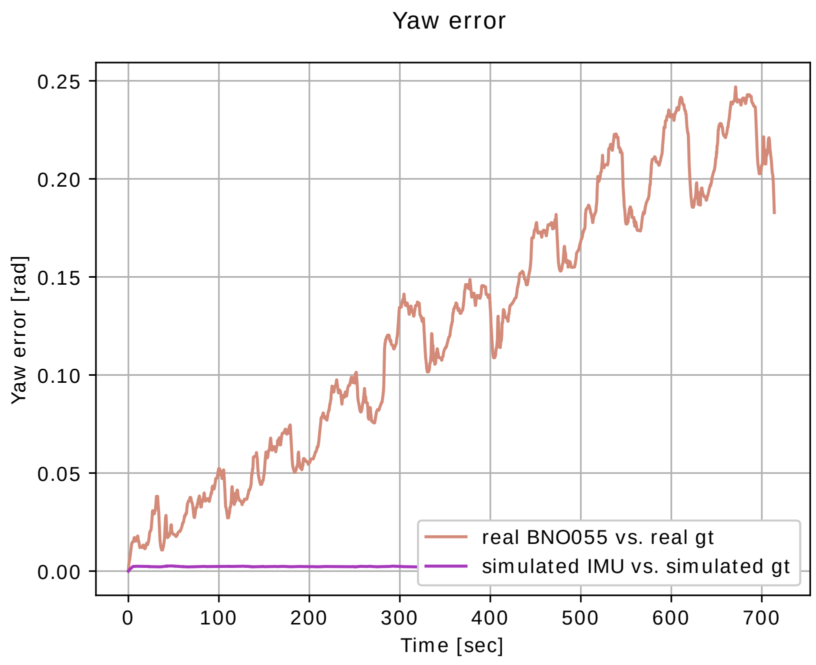


Figure 4.9: Trajectory A, long-term experiment - yaw errors, smoothed by median filter with a window size of 519 samples

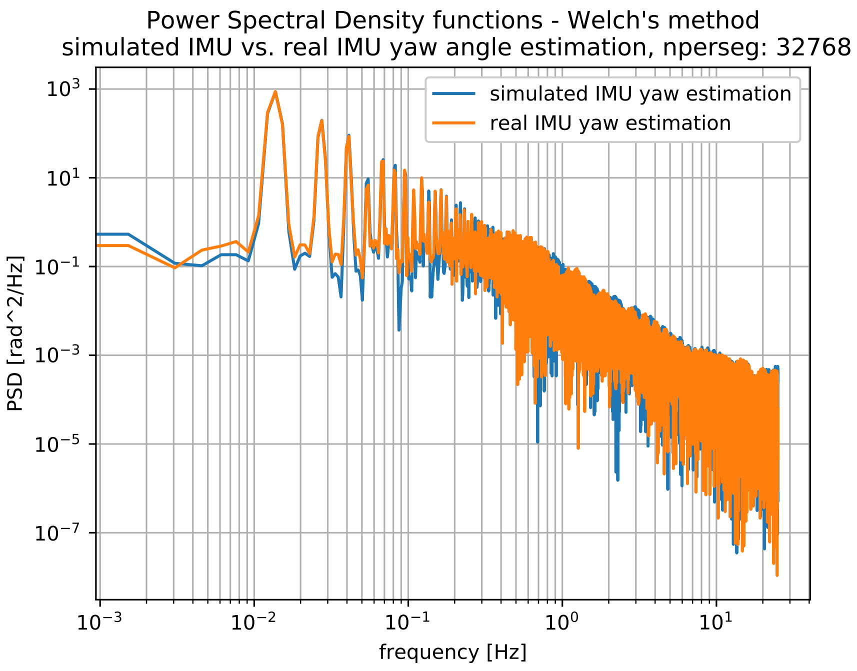


Figure 4.10: Trajectory A, long-term experiment - PSD function of yaw angle outputs

domain comparison is clear that the generic IMU plugin lacks the drift feature. The reason is, that in the area of lower frequencies, several dominant peaks are corresponding to the repetitions of the trajectory A, which manifests themselves as low-frequency signals with much higher power than the bias drift. Therefore in this PSD graph, the bias drift is not observable at all. The presence of low-frequency bias drift in real yaw signal is nevertheless noticeable clearly from the second PSD graph showing the PSD functions of yaw errors (figure 4.11). In this case, the simulated yaw output has significantly smaller power on low frequencies by a few orders of magnitude.

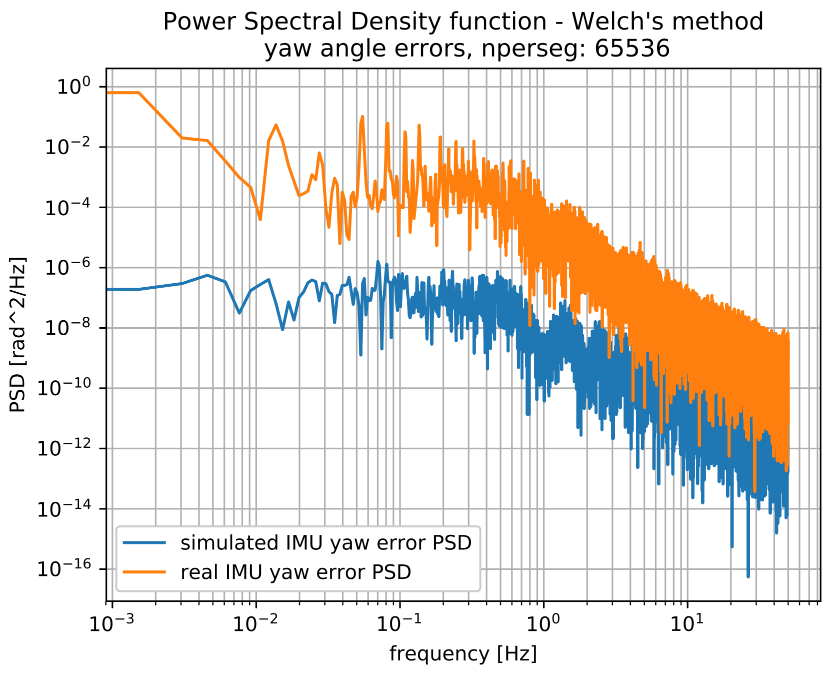


Figure 4.11: Trajectory A, long-term experiment - PSD function of yaw errors smoothed by median filter with window size 101 samples

The performed experiments showed the fundamental limitation of the generic Gazebo IMU plugin. Because it does not simulate the low-frequency bias drift, it is suitable only for short-term experiments and lacks the fidelity in longer experiments.

Thus, I made all other experiments for different trajectories with the Hector IMU plugin only.

4.2 Hector IMU plugin

The main comparable advantage of the Hector IMU plugin is a separate noise model for yaw angle output including the bias drift simulation. Tunable input parameters of its noise model correspond to the variables in the equation 3.47 and 3.48:

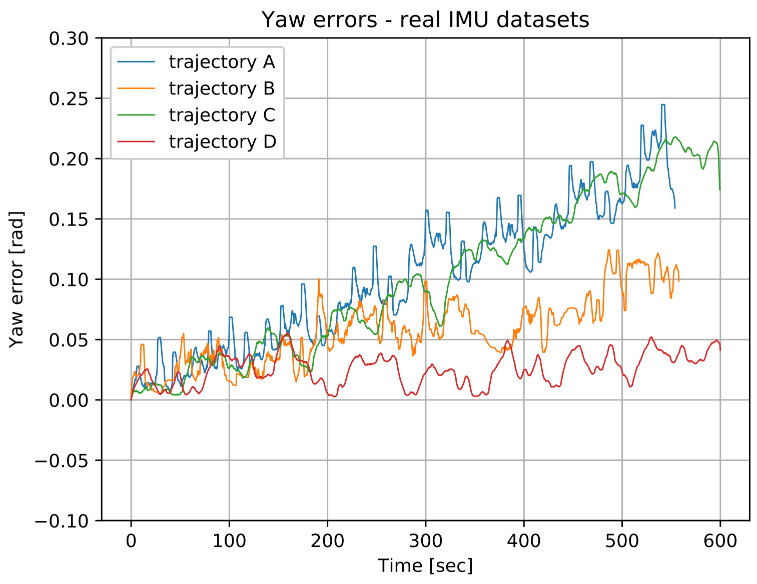


Figure 4.12: Yaw errors for real datasets, signals smoothed by median filter with window size 591 samples

- σ_{white} - standard deviation of the white noise
- f_d - mean frequency of the low-frequency bias drift
- σ_{drift} - standard deviation of the bias drift

While a reasonable estimation of σ_{white} can be used from the experiments with generic IMU plugin (see equation 4.6), neither f_d nor σ_{drif} were found directly by Allan variance noise analysis. Besides, the found rate random walk parameters, corresponding to the gyroscope bias fluctuations seems to be very unreliable. Moreover, the Hector IMU plugin does not compute the orientation from the noisy gyroscope and accelerometer data, but in every discrete timestep, it takes directly the orientation and apply the noise yaw model separately. Thus, I roughly estimated f_d and σ_{drif} based on the available real datasets and then I experimentally verified the correctness of the estimation.

For that purpose, I plotted the yaw errors and corresponding PSD functions of the four used datasets (figures 4.12, 4.13). The detail of PSD functions for lower frequenciesshows in figure 4.13 shows that there is a peak at approximately

$$f \approx 0.015 \;\; Hz \tag{4.7}$$

which is equal to perid (correlation time) approximately

$$\tau \approx 70 \;\; sec \tag{4.8}$$

When comparing this figures with the resulting yaw output PSD function from the long-term experiment with generic IMU plugin (figure 4.10) and

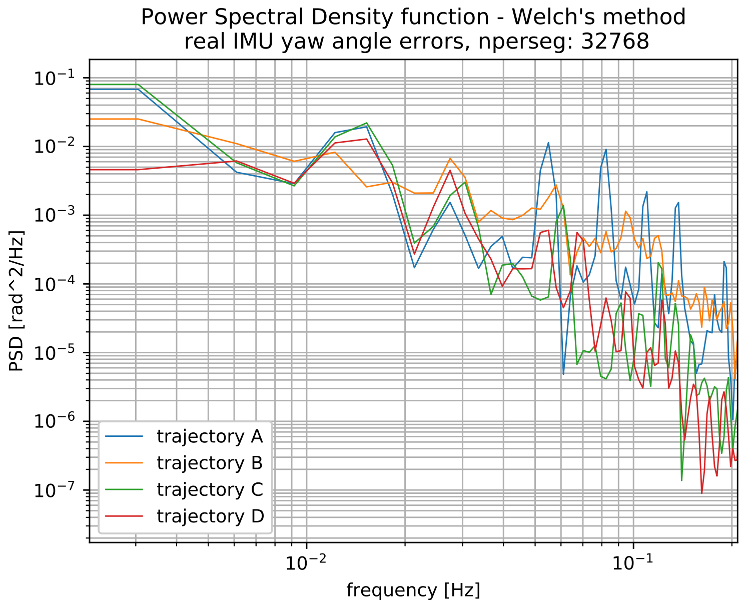


Figure 4.13: Low-frequency part of PSD for real yaw errors; before Welsch's algorithm, signals were smoothed by median filter with window size 591 samples

with the yaw output in the short-term experiment (figure 4.2), it can be seen that f is roughly equal to the frequency of trajectory iteration. The figure 4.13 therefore shows that the repetition movement pattern (in case of trajectories A, C, D) has also some influence to yaw error in real datasets. Thus, I do not consider the frequency peak at f as a correct estimation for f_d parameter.

Because the length and the sampling rate of the datasets limit the frequency resolution of PSD functions, for frequencies lower than 10^{-2} Hz, the PSD functions do not provide much information. Despite this, the trend of PSD for the lowest visible frequencies suggests that holds:

$$f_d < 0.01 \ \ Hz \tag{4.9}$$

Thus, I set the initial estimation of f_d as:

$$f_d = 0.005 \ \ Hz \tag{4.10}$$

From the figure 4.12 I also estimated the maximal magnitude of step changes in the yaw angle estimation error signals as approximately

$$\Delta\Theta \approx 0.05 \ \ rad \tag{4.11}$$

The equation 3.48 shows that σ_{drift} is not directly the standard deviation of the generated white noise[2], because it is damped (multiplied) by $\sqrt{2f_d}$. To achieve similar changes in simulated yaw angle estimation, a meaningful value of σ_{drift} seems to given by formula

[2]σ_{drift} is called a *bias drift standard deviation* by the official plugin description in [83]

$$\sigma_{drift} \approx \frac{\Delta\Theta}{\sqrt{2f_d}} \tag{4.12}$$

Considering the estimated value (equation 4.10) yields:

$$\sigma_{drift} \approx \frac{0.05}{\sqrt{2 \cdot 0.005}} = 0.5 \;\; rad \tag{4.13}$$

4.2.1 Trajectory A

Firstly, I run the simulation with the initial parameters (equations 4.6, 4.10,4.13) three times. To save the space, I do not put all resulting graphs such as control errors or yaw comparison into this text since at first glance, they all look almost identical and do not provide much information. However, all the generated graphs from all experiments are included as files on the attached CD in directory `experiment results` (see appendix B). Figures

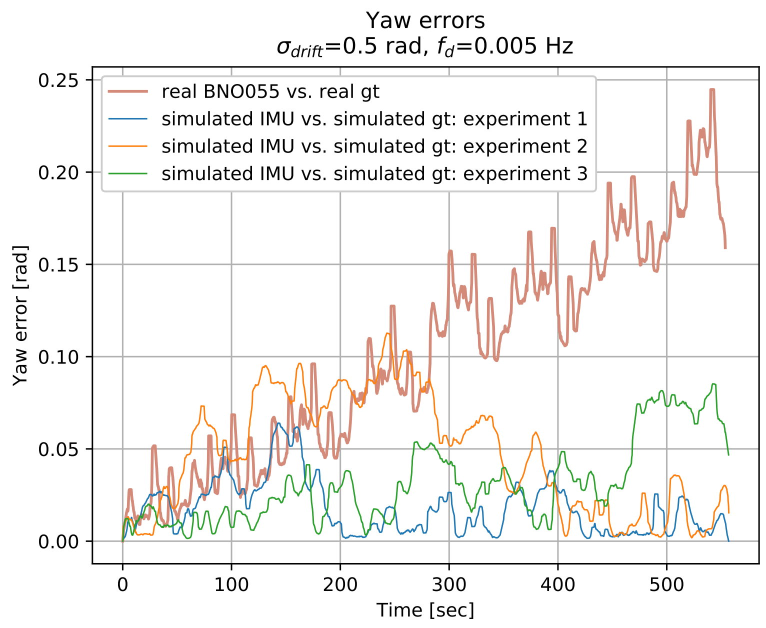


Figure 4.14: Trajectory A, long-term experiments with Hector plugin - yaw error comparison

4.14 and 4.15 depict the resulting yaw errors and corresponding PSD functions respectively. The results show a significant improvement in both time and frequency domain in comparison to the generic Gazebo IMU plugin. The bias drift looks realistically. However, it is hard to evaluate based only on a limited number of datasets and a few repetitions of the same simulation.

It seems that in real data, the steep changes in yaw error signals are caused somehow by the abrupt changes robot's movement itself and that the bias drift itself has a rather slower tendency. Thus, I tried to decrease the f_d. I

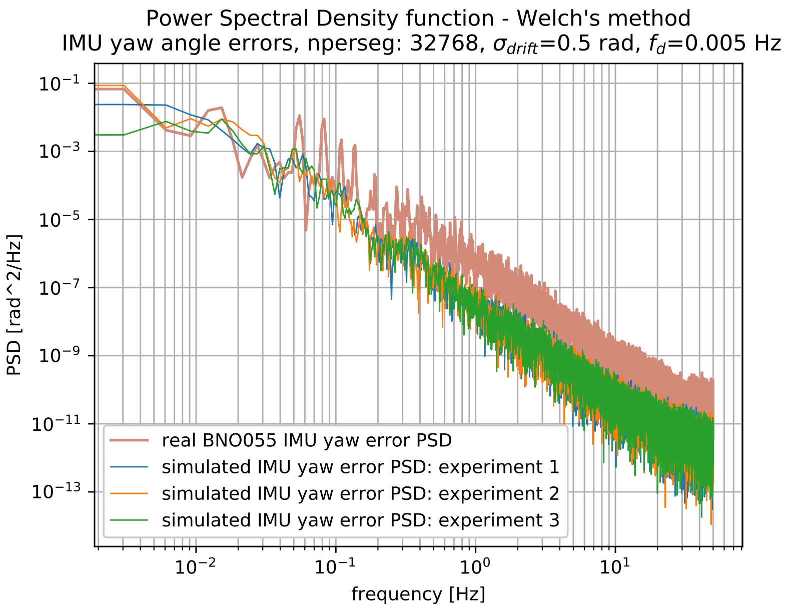


Figure 4.15: Trajectory A, long-term experiments with Hector plugin - PSD functions of yaw errors

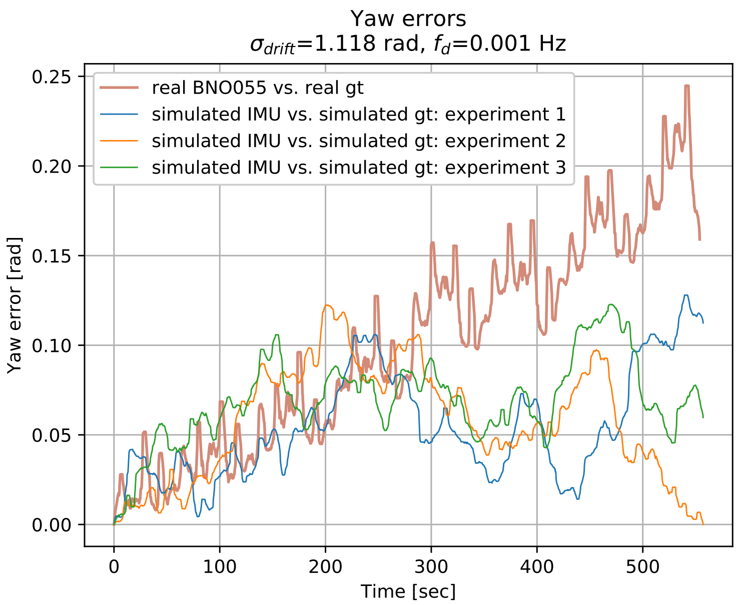


Figure 4.16: Trajectory A, long-term experiments with Hector plugin - yaw error comparison

set the following parameters:

$$f_d = 0.001 \ \ Hz \tag{4.14}$$

$$\sigma_{drift} = 1.118 \ \ rad \tag{4.15}$$

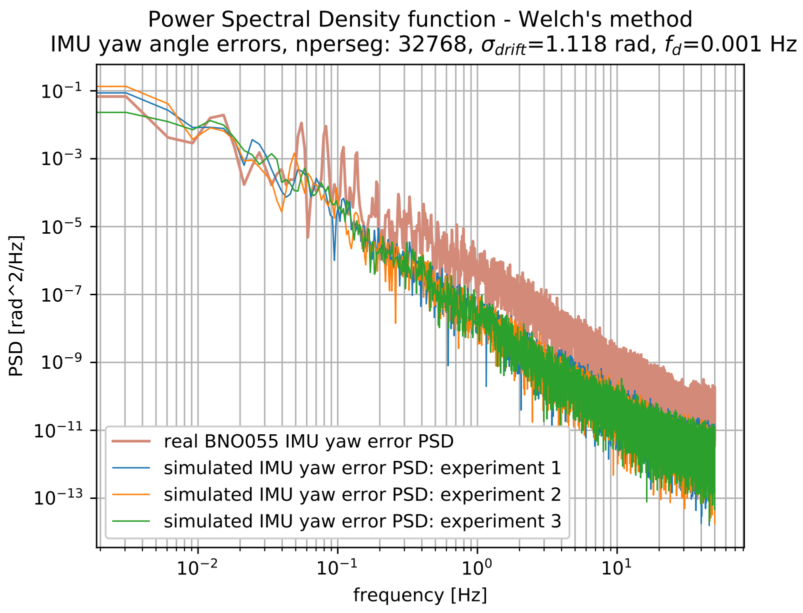


Figure 4.17: Trajectory A, long-term experiments with Hector plugin - PSD functions of yaw errors

With the same configuration I made also simualtions for the other datasets, resulting graphs are in following subsections. Yaw errors were smoothed by median filter with the window size of 519 samples.

4.2.2 Trajectory B

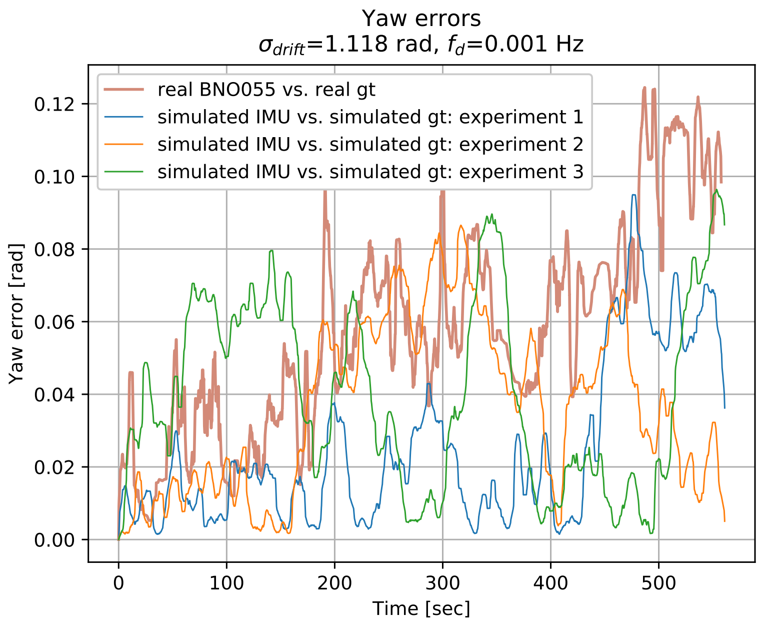


Figure 4.18: Trajectory B, long-term experiments with Hector plugin - yaw error comparison

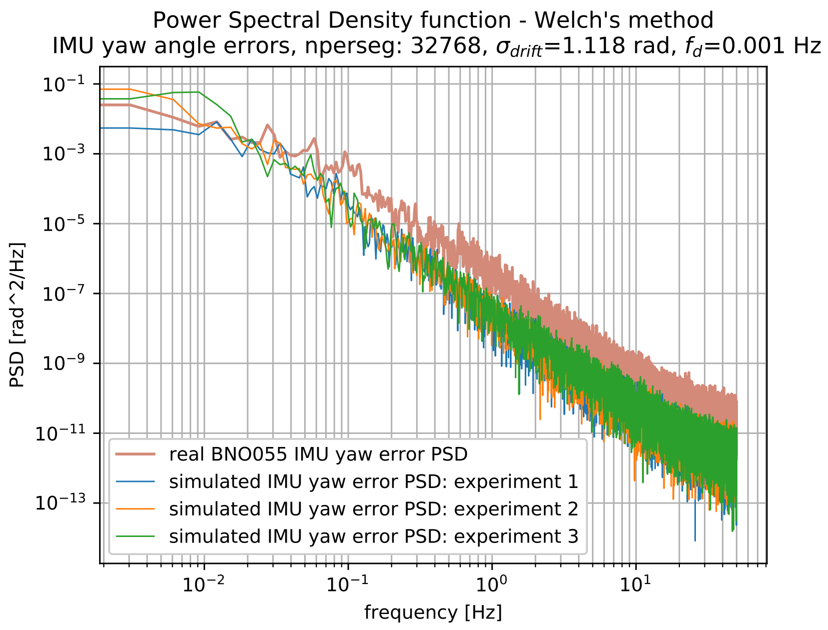


Figure 4.19: Trajectory B, long-term experiments with Hector plugin - PSD functions of yaw errors

For the illustrative purposes, I also include one figure with comparison of yaw outputs for trajectory B, which was followed only once. The detail of signal after approximately 10 minutes can be seen in figure 4.20. This figure clearly shows that both real and simulated signals are influenced by the bias drift.

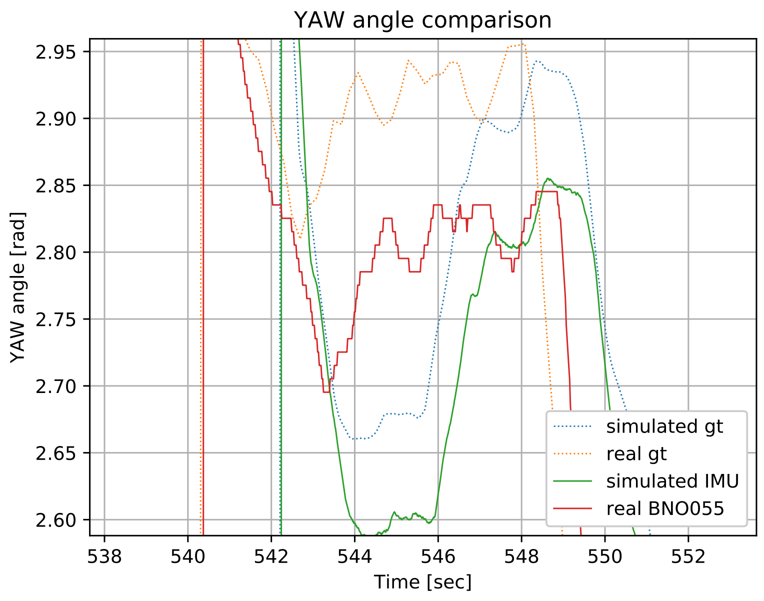


Figure 4.20: Trajectory B, long-term experiments with Hector plugin - detail of yaw angle output after approximately 10 minutes

4.2.3 Trajectory C

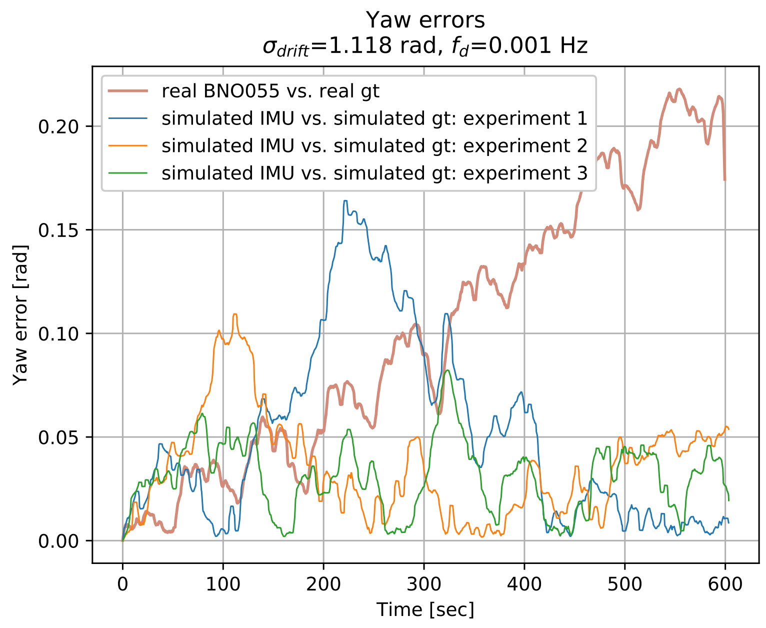


Figure 4.21: Trajectory C, long-term experiments with Hector plugin - yaw error comparison

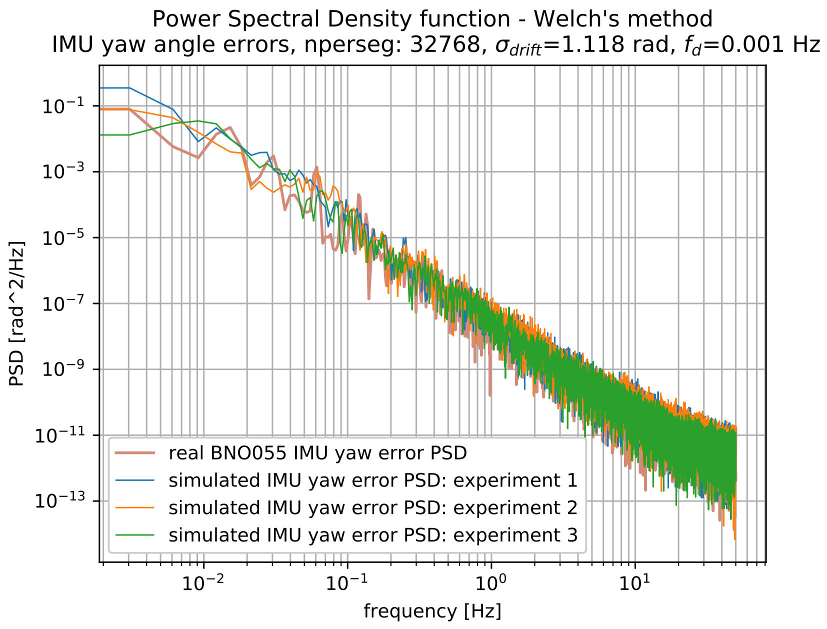


Figure 4.22: Trajectory C, long-term experiments with Hector plugin - PSD functions of yaw errors

4.2.4 Trajectory D

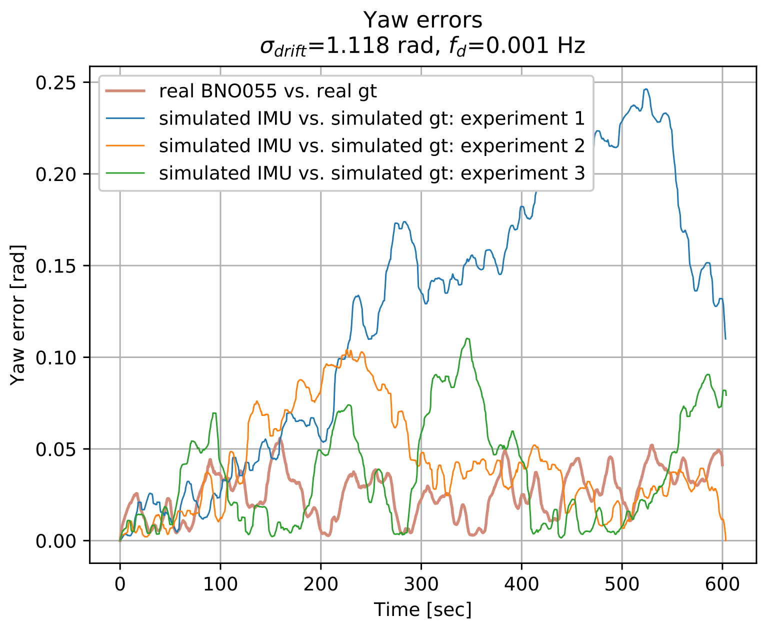


Figure 4.23: Trajectory D, long-term experiments with Hector plugin - yaw error comparison

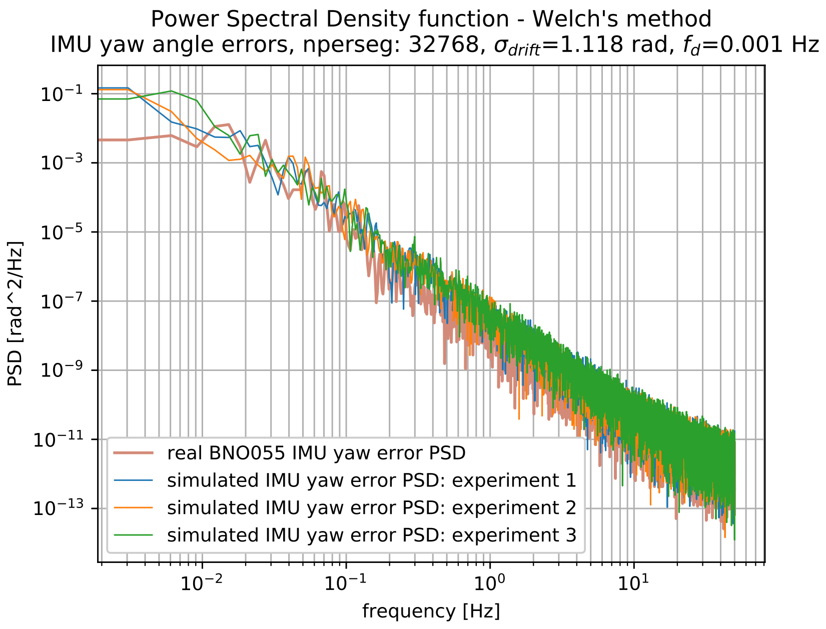


Figure 4.24: Trajectory D, long-term experiments with Hector plugin - PSD functions of yaw errors

4.3 Experiment summary

The performed experiments proved that the generic Gazebo IMU plugin is not sufficient for any long-term experiments when a high fidelity of the simulated IMU output is required. The main reason is the lack of any slow bias drift simulations.

The Hector IMU plugin enables such simulations, but the underlying noise models do not directly correspond to standard parameters used to noise description in IMU sensors, and also the noisy gyroscope and accelerometer data are not fused to produce appropriately noisy orientation output.

As a workaround, I used only the separate yaw angle noise model and estimated its parameters based on reasoning done on the top of provided datasets from real IMU. The set of subsequent experiments proved that in terms of both frequency and time domain behavior, the output signal looks much more realistic compared to the output of a generic IMU plugin.

However, it is important to emphasize here, that a solid statistical comparison and IMU model evaluation was not possible at all, due to the very limited number of datasets[3]. And last but not least also the simulations in Gazebo were quite time demanding. On the used computer, the simulations had to run in real-time.

Based on the inspected datasets, to simulate the yaw output of BNO055, I consider as reasonable f_d valus laying approximately in the interval:

$$f_d \in (0.0008; 0.007) \;\; Hz \tag{4.16}$$

or in terms of corresponding correlation times approximately:

$$\tau_d \in (150; 1250) \;\; sec \tag{4.17}$$

When the magnitude of the yaw error changes is estimated in the time domain, and the f_d is selected, it can be then converted to appropriate σ_{drift} parameter using the equation 4.12.

[3]Apart from four utilized datasets, I had access to few more. But they had often either incomplete or corrupted GPS or IMU raw data. Nonetheless, reliable statistical analysis would require a much higher number of datasets than was available at all.

Chapter 5
Conclusion

In this thesis, I dealt extensively with the modeling and simulation of the inertial measurement unit (IMU) in a robotic simulator for the future rapid development support of service mobile robots. In agreement with the thesis's assignment, I accomplished the following steps:

- I went into the theory of IMU sensors, and described their working principles, properties and main error causes together with the most important technical parameters.
- I researched available state-of-the-art robotic simulators. V-rep, Webots and Gazebo were compared more closely concerning the main goals of this thesis.
- The Gazebo simulator was finally selected as the simulation environment.
- I implemented two different IMU models based on available Gazebo IMU plugins. The models were set to simulate the real behavior of real IMU sensor BNO055 with emphasis to the fidelity of the robot's heading angle estimation.
- I also implemented a custom IMU simulation framework as a ROS package. It communicates with the Gazebo simulator and enables the control of the robot model with respect to provided raw GPS data (trajectory tracking approximation). It captures the simulated IMU output and simulated ground truth position, does the data post-processing, saves the data, and provides semi-automated evaluation of the simulated IMU model with respect to datasets measured by real IMU sensor.
- I performed a set of simulations in Gazebo using the implemented IMU simulation framework. The experiment results clearly showed that the IMU model based on standard Gazebo IMU plugin is not sufficient for long-term simulations due to the lack of low-frequency bias drift modeling. The second IMU model based on Hector plugin can model yaw angle bias drift, but its proper tuning is tricky since its parameters do not directly correspond to ordinary technical parameters of BNO055. However, using this IMU model even with roughly estimated parameters can still significantly enhance the fidelity of BNO055's simulations in

comparison with the standard Gazebo IMU plugin. A solid statistical evaluation of IMU model was not possible yet and should be done before real usage of this IMU model for rapid development purposes in the industry.

In addition to the assignment of this thesis, I also solved the following problems:

- I designed and implemented a custom feedback PID controller to improve the insufficient open-loop control of the differential drives mobile robot model in Gazebo simulator. This controller helps to approximate the original trajectory provided as a set of waypoints.
- I performed a few long-term static measurements with real BNO055 IMU sensor and then I analyzed the noise characteristics of BNO055 IMU sensor by Allan variance method, nonetheless only white noise related parameters were determined reliably.

Working on this thesis was quite a challenging task, because of too many "degrees-of-freedom" and a broad scope of different problems and needed skills.

The original experimental setup with the mobile robot and the original control algorithm were not provided at all. The particular conditions under that the provided datasets were obtained were unknown. An last but not least, the noise parameters listed in the official datasheet and obtained by Allan variance analysis were not much useable. Also, the comprehensive documentation for Hector IMU plugin was missing, which led to problems when putting Hector IMU plugin into work.

I had to use the reverse engineering approach extraordinarily often. Since this thesis also contributed to the real research project in an international company, work on this thesis also comprised a significant amount of communication in foreign language and cooperation across a few countries.

5.1 Future work

In this thesis, I did the modeling of real BNO055 sensor in a simplified way, based on many reduction assumptions. The following ideas can serve as guidelines for the future work concerning this topic.

- The solid statistical analysis and evaluation of the Hector IMU plugin should be done. Ideally, one should perform many independent long-term experiments under well-defined conditions on a few defined trajectories with the real robot (maybe even with a few different pieces of BNO055) and replicate these experiments also with the IMU model in Gazebo. A long duration of experiments together with a more sophisticated method of power spectral density computation could increase the desired insight into low-frequency domain behavior and answer the question whether the Hector IMU plugin is sufficient for a high fidelity BNO055 yaw angle output simulations.

- If the statistical analysis proves that the model based only on Hector IMU plugin and its yaw angle noise model is not sufficient, one should consider the implementation of a brand new custom IMU plugin. A good starting point can be the source code of the RotorS IMU plugin. Its noise model has reasonable physical meaning, and implementation of a known data fusion algorithm would enable its application also for the yaw angle output simulation.
- In some datasets, the real BNO055 seems to tend to prefer one direction during the low-frequency bias drift. Also, a pattern in the robot's movement seems to have an impact on bias random walk behavior. These phenomenons should be analyzed.
- All experiments in Gazebo were performed on an ideally flat surface without any imperfections. Introducing small terrain bulges in the simulated world could better mirror the reality.
- Adding of an optimal trajectory tracking algorithm to the IMU simulation framework could increase its utility value.

Appendix A

Bibliography

[1] RobZone Limited. *DUORO XCONTROL PROFI - RoboZone.cz*. [Online; accessed 08-05-2019]. 2019. URL: `www.robzone.cz/roboticke-vysavace/duoro-x-control-profi`.

[2] Gregory Dudek and Michael Jenkin. "Inertial Sensors, GPS, and Odometry". In: *Springer handbook of robotics*. Berlin: Springer, 2008, pp. 477–490. ISBN: 978-3-540-23957-4.

[3] John L. Weston David H. Titterton. *Strapdown inertial navigation technology - 2nd edition*. The Institution of Electrical Engineer, 2004.

[4] John Geen and David Krakauer. "New iMEMS® angular-rate-sensing gyroscope". In: *Analog Dialogue* 37.3 (2003), pp. 1–4.

[5] Wikimedia Foundation. *3D gyroscope — Wikipedia, The Free Encyclopedia*. [Online; accessed 01-02-2019]. San Francisco (CA), 2019. URL: `https://commons.wikimedia.org/wiki/File:3D_Gyroscope.png`.

[6] Walter R Johnson. "ART. III.–Description of an Apparatus called the Rotascope, for exhibiting several phenomena and illustrating certain laws of rotary motion". In: *American Journal of Science and Arts (1820-1879)* 21.2 (1832), 264B.

[7] Walter Wrigley. "The History of Inertial Navigation". In: *Journal of Navigation* 30 (01 Jan. 1977). DOI: `10.1017/S0373463300043642`.

[8] B. T. Meggitt (eds.) K. T. V. Grattan Y. N. Ning (auth.) *Optical Fiber Sensor Technology: Devices and Technology*. 1st ed. Optoelectronics, Imaging and Sensing 2. Springer US, 1998, pp. 303–310. ISBN: 978-1-4613-7651-4.

[9] Yves Paturel. "Fiber Optic Gyro: Theory & Applications". In: (2014). URL: `http://ainegypt.org/event/papers/Full%20paper%20-%20MELAHA%202014.pdf`.

[10] Wikimedia Foundation. *Microelectromechanical systems — Wikipedia, The Free Encyclopedia*. [Online; accessed 31-01-2019]. San Francisco (CA), 2019. URL: `https://en.wikipedia.org/wiki/Microelectromechanical_systems`.

[11] Martin Vágner. "Návrh a identifikace rozšířeného modelu MEMS gyroskopu". PhD thesis. Vysoké učení technické v Brně, 2015.

[12] Volker Kempe. *Inertial MEMS: principles and practice*. Cambridge University Press, 2011.

[13] David M Rozelle. “The hemispherical resonator gyro: From wineglass to the planets”. In: *Proc. 19th AAS/AIAA Space Flight Mechanics Meeting*. 2009, pp. 1157–1178.

[14] Priyanka Aggarwal. *MEMS-based integrated navigation*. Artech House, 2010.

[15] Manfred Weber. *Piezoelectric Accelerometers - Theory and Application*. [Online; accessed 03-02-2019]. 2012. URL: `https://www.mmf.de/manual/transducermane.pdf`.

[16] James Fennally. *Capacitive Vs Thermal MEMS for High-Vibration Applications*. New York City, 2016. URL: `https://www.automation.com/automation-news/article/capacitive-vs-thermal-mems-for-high-vibration-applications`.

[17] Tarun Kanti Bhattacharyya and Anindya Lal Roy. “MEMS piezoresistive accelerometers”. In: *Micro and smart devices and systems*. Springer, 2014, pp. 19–34.

[18] Jacob Fraden. *Handbook of modern sensors*. 2013.

[19] Hugh Durrant-Whyte and Thomas C Henderson. “Multisensor data fusion”. In: *Springer Handbook of Robotics*. Springer, 2016, pp. 867–896.

[20] Long Dinh Tran. “Data Fusion with 9 Degrees of Freedom Inertial Measurement Unit To Determine Object’s Orientation”. In: (2017).

[21] Chris Chatfield. *The analysis of time series: an introduction*. Chapman and Hall/CRC, 2016.

[22] NovAtel. *IMU Errors and Their Effects*. [Online; accessed 25-04-2019]. 2014. URL: `www.novatel.com/assets/Documents/Bulletins/APN064.pdf`.

[23] Xsens Technologies B.V. *Gyroscopes - Xsens 3D motion tracking*. [Online; accessed 28-01-2019]. URL: `https://www.xsens.com/tags/gyroscopes/`.

[24] *IMU, what for: performance per application infographic*. [Online; accessed 07-03-2019]. 2018. URL: `https://www.thalesgroup.com/en/worldwide/aerospace/topaxyz-inertial-measurement-unit-imu/infographic`.

[25] *Gyroscope - VectorNav Library*. VectorNav Technologies, llc. 2009. URL: `https://www.vectornav.com/support/library/gyroscope`.

[26] David W Allan. “Statistics of atomic frequency standards”. In: *Proceedings of the IEEE* 54.2 (1966), pp. 221–230.

[27] ISS Board. “IEEE standard specification format guide and test procedure for single-axis interferometric fiber optic gyros”. In: *IEEE Std* (1998), pp. 952–1997.

[28] Allan Variance. “Noise Analysis for Gyroscopes”. In: *Freescale Semiconductor Document Number: AN5087 Application Note Rev. 0* 2 (2015).

[29] Naser El-Sheimy, Haiying Hou, and Xiaoji Niu. “Analysis and modeling of inertial sensors using Allan variance”. In: *IEEE Transactions on instrumentation and measurement* 57.1 (2008), pp. 140–149.

[30] Leslie Barreda Pupo. “Characterization of errors and noises in MEMS inertial sensors using Allan variance method”. MA thesis. Universitat Politechnica de Catalunya, 2016.

[31] Myung Hwangbo, JunSik Kim, and Takeo Kanade. “IMU self-calibration using factorization”. In: *IEEE Transactions on Robotics* 29.2 (2013), pp. 493–507.

[32] Mark Pedley. “High precision calibration of a three-axis accelerometer”. In: *Freescale Semiconductor Application Note* 1 (2013).

[33] David Tedaldi, Alberto Pretto, and Emanuele Menegatti. “A robust and easy to implement method for IMU calibration without external equipments”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 3042–3049.

[34] Richard Gran. *What is simulation?* [Online; accessed 03-03-2019]. Youtube, 2012. URL: `https://www.youtube.com/watch?v=0CMafswcNkY`.

[35] Tom Erez, Yuval Tassa, and Emanuel Todorov. “Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx”. In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 4397–4404.

[36] Patricio Castillo-Pizarro, Tomás V Arredondo, and Miguel Torres-Torriti. “Introductory survey to open-source mobile robot simulation software”. In: *2010 Latin American Robotics Symposium and Intelligent Robotics Meeting*. IEEE. 2010, pp. 150–155.

[37] Miguel Torres-Torriti, T Arredondo, and P Castillo-Pizarro. “Survey and comparative study of free simulation software for mobile robots”. In: *Robotica* 34.4 (2016), pp. 791–822.

[38] J. Craighead et al. “A Survey of Commercial amp; Open Source Unmanned Vehicle Simulators”. In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. Apr. 2007, pp. 852–857. DOI: `10.1109/ROBOT.2007.363092`.

[39] Aaron Staranowicz and Gian Luca Mariottini. “A Survey and Comparison of Commercial and Open-source Robotic Simulator Software”. In: *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*. PETRA ’11. Heraklion, Crete, Greece: ACM, 2011, 56:1–56:8. ISBN: 978-1-4503-0772-7. DOI: `10.1145/2141622.2141689`. URL: `http://doi.acm.org/10.1145/2141622.2141689`.

[40] Lucas Nogueira. “Comparative analysis between gazebo and v-rep robotic simulators”. In: *Seminario Interno de Cognicao Artificial-SICA 2014* (2014), p. 5.

[41] Wikimeadia Foundation. *Robotics simulator — Wikipedia, The Free Encyclopedia.* [Online; accessed 25-03-2019]. 2019. URL: `https://en.wikipedia.org/wiki/Robotics_simulator`.

[42] Open Source Robotics Foundation. *Gazebo.* [Online; accessed 25-03-2019]. URL: `http://gazebosim.org`.

[43] Coppelia Robotics Ltd. *Coppelia Robotics V-REP.* [Online; accessed 25-03-2019]. URL: `http://www.coppeliarobotics.com`.

[44] Cyberbotics Ltd. *Webots.* Ed. by Cyberbotics Ltd. [Online; accessed 25-03-2019]. URL: `http://www.cyberbotics.com`.

[45] Coppelia Robotics Ltd. *V-REP forum: Questions/Answers around V-REP: Accelerometer or Gyroscope.* [Online; accessed 18-05-2019]. 2013. URL: `http://www.forum.coppeliarobotics.com/viewtopic.php?t=93`.

[46] Coppelia Robotics Ltd. *V-REP forum: Questions/Answers around V-REP: Sensor noise.* [Online; accessed 18-05-2019]. 2018. URL: `http://forum.coppeliarobotics.com/viewtopic.php?t=7457`.

[47] Open Source Robotics Foundation. *Gazebo: tutorial: Sensor noise model.* [Online; accessed 18-05-2019]. 2014. URL: `gazebosim.org/tutorials?tut=sensor_noise&cat=sensors`.

[48] Cyberbotics Ltd. *Webots documentation: Lookup Table.* [Online; accessed 18-05-2019]. 2019. URL: `https://cyberbotics.com/doc/reference/distancesensor#lookup-table`.

[49] Serena Ivaldi, Vincent Padois, and Francesco Nori. “Tools for dynamics simulation of robots: a survey based on user feedback”. In: *arXiv preprint arXiv:1402.7050* (2014).

[50] Lenka Pitonakova et al. “Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators”. In: *Annual Conference Towards Autonomous Robotic Systems.* Springer. 2018, pp. 357–368.

[51] Tom Norton. *Version R2019a - Webots Goes Open Source.* Ed. by Cyberbotics Ltd. [Online; accessed 25-03-2019]. URL: `https://cyberbotics.com/doc/blog/Webots-2019-a-release`.

[52] Open Source Robotics Foundation. *Documentation - ROS Wiki.* Ed. by Open Source Robotics Foundation. [Online; accessed 29-03-2019]. URL: `https://http://wiki.ros.org/`.

[53] Lentin Joseph and Jonathan Cacace. *Mastering ROS for Robotics Programming: Design, build, and simulate complex robots using the Robot Operating System.* Packt Publishing Ltd, 2018.

[54] Nathan Koenig and Andrew Howard. "Design and use paradigms for gazebo, an open-source multi-robot simulator". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 3. IEEE. 2004, pp. 2149–2154.

[55] Dave Coleman. *Gazebo tutorial: ROS control*. Ed. by Open Source Robotics Foundation. [Online; accessed 06-04-2019]. 2013. URL: `http://gazebosim.org/tutorials/?tut=ros_control`.

[56] Inc. Open Source Robotics Foundation. *SDF*. Ed. by Inc. Open Source Robotics Foundation. [Online; accessed 06-04-2019]. 2019. URL: `http://sdformat.org/spec`.

[57] ANAVS GmbH. *The Multi-Sensor RTK-Module*. Ed. by ANAVS GmbH. [Online; accessed 12-04-2019]. 2019. URL: `https://anavs.de/multi-sensor-rtk-module/`.

[58] E. Maulana, M. A. Muslim, and A. Zainuri. "Inverse kinematics of a two-wheeled differential drive an autonomous mobile robot". In: *2014 Electrical Power, Electronics, Communicatons, Control and Informatics Seminar (EECCIS)*. Aug. 2014, pp. 93–98. DOI: `10.1109/EECCIS.2014.7003726`.

[59] Bosch Sensortec. "BNO055 Intelligent 9-axis absolute orientation sensor". In: *Bosch Sensortec, Baden-Württemberg, Germany* (2016). URL: `https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BNO055-DS000.pdf`.

[60] Michał Drwięga. *bosch imu driver*. [Online; accessed 25-04-2019]. 2017. URL: `https://github.com/mdrwiega/bosch_imu_driver`.

[61] Mathworks. *Sensor fusion and tracking toolbox*. [Online; accessed 25-04-2019]. 2019. URL: `https://nl.mathworks.com/products/sensor-fusion-and-tracking.html`.

[62] Anders E.E. Wallin. *AllanTools PyPI*. [Online; accessed 25-04-2019]. 2019. URL: `https://pypi.org/project/AllanTools`.

[63] Dan Pierce. *GitHub - GAVLab - allan variance - Allan variance approach for characterizing inertial signals*. [Online; accessed 26-04-2019]. 2017. URL: `https://github.com/GAVLab/allan_variance`.

[64] *Interquartile range — Wikipedia, The Free Encyclopedia*. [Online; accessed 30-04-2019]. San Francisco (CA), 2019. URL: `https://en.wikipedia.org/wiki/Interquartile_range#/media/File:Boxplot_vs_PDF.svg`.

[65] Juan Jurado. *Tools for Inertial Allan Variance Analysis and Simulation*. [Online; accessed 30-04-2019]. 2017. URL: `https://nl.mathworks.com/matlabcentral/fileexchange/61777-tools-for-inertial-allan-variance-analysis-and-simulation`.

[66] Cypress developer community. *UART overrun error - Cypress developer community*. [Online; accessed 30-04-2019]. 2011. URL: `https://community.cypress.com/docs/DOC-12028`.

[67] Christian Holl. *GitHub - Hacks4ROS: C++ Node for the Bosch IMU BNO055 (UART)*. [Online; accessed: 28-04-2019]. 2016. URL: `https://github.com/Hacks4ROS/h4r_bosch_bno055_uart`.

[68] Ralf Kaestner. *rqt_multiplot - ROS wiki*. [Online; accessed 03-05-2019]. 2018. URL: `http://wiki.ros.org/rqt_multiplot`.

[69] D. Chwa. "Tracking Control of Differential-Drive Wheeled Mobile Robots Using a Backstepping-Like Feedback Linearization". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 40.6 (2010), pp. 1285–1295. ISSN: 1083-4427. DOI: `10.1109/TSMCA.2010.2052605`.

[70] Khoshnam Shojaei et al. "Adaptive trajectory tracking control of a differential drive wheeled mobile robot". In: *Robotica* 29.3 (2011), pp. 391–402. DOI: `10.1017/S0263574710000202`.

[71] James D. Broesch. *Applications of DSP: Median Filter*. [Online; accessed 23-04-2019]. 2009. URL: `https://www.sciencedirect.com/topics/engineering/median-filters`.

[72] Peter Welch. "The use of fast Fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms". In: *IEEE Transactions on audio and electroacoustics* 15.2 (1967), pp. 70–73.

[73] *List of moments of inertia — Wikipedia, The Free Encyclopedia*. [Online; accessed 09-05-2019]. San Francisco (CA), 2019. URL: `https://en.wikipedia.org/wiki/List_of_moments_of_inertia`.

[74] ETH ASL. *IMU Noise Model*. [Online; accessed 09-05-2019]. 2016. URL: `https://github.com/ethz-asl/kalibr/wiki/IMU-Noise-Model`.

[75] Petko Petkov and Tsonyo Slavov. "Stochastic modeling of MEMS inertial sensors". In: *Cybernetics and information technologies* 10.2 (2010), pp. 31–40.

[76] Peter S Maybeck. *Stochastic models, estimation, and control*. Vol. 3. Academic press, 1982.

[77] Open Source Robotics Foundation. *Gazebo turorial: Using Gazebo plugins with ROS*. [Online; accessed 02-05-2019]. URL: `http://gazebosim.org/tutorials?tut=ros_gzplugins#IMU`.

[78] Andrew Howard Nate Koenig. *gazebo::GazeboRosIMU Class Reference*. [Online; accessed 03-05-2019]. 2008. URL: `http://docs.ros.org/diamondback/api/gazebo_plugins/html/classgazebo_1_1GazeboRosIMU.html`.

[79] James Goppert. *pull request 2673: Add random walk support to GaussianNoise - Bitbucket*. [Online; accessed 07-05-2019]. URL: `https://bitbucket.org/osrf/gazebo/pull-requests/2673/add-random-walk-support-to-gaussiannoise`.

[80] Fadri Furrer et al. “Robot Operating System (ROS): The Complete Reference (Volume 1)”. In: ed. by Anis Koubaa. Cham: Springer International Publishing, 2016. Chap. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. ISBN: 978-3-319-26054-9. DOI: 10.1007/978-3-319-26054-9_23. URL: http://dx.doi.org/10.1007/978-3-319-26054-9_23.

[81] Fadri Furrer. *RotorS simulator wiki - GitHub*. [Online; accessed 23-03-2019]. 2017. URL: https://github.com/ethz-asl/rotors_simulator/wiki.

[82] Wang Liu Liu. *use librotors_gazebo_imu plugin.so in turtlebot simulator - issue 495 - ethz - asl - RotorS - GitHub*. [Online; accessed 23-03-2019]. 2018. URL: https://github.com/ethz-asl/rotors_simulator/issues/494.

[83] Johannes Mayer. *hector_gazebo_plugins - ROS wiki*. [Online; accessed 21-03-2019]. 2016. URL: http://wiki.ros.org/hector_gazebo_plugins.

[84] Matt Duff. *Calculating Spectral Noise Density to RMS Noise*. [Online; accessed 11-05-2019]. 2010. URL: https://www.youtube.com/watch?v=ywChrIRIXWQ.

Appendix B
CD contents

The structure of the attached CD is depicted in the figure B.2. For the simplicity, I do not mention all the files generated from the experiments are depicted separately, but rather introduce how the files are orderd (see the figure B.1).

```
├── trajectory A
│   ├── sigma=0.5, fd=0.005
│   │   ├── 1
│   │   │   ├── 20190516-174258_control_error.png
│   │   │   ├── 20190516-174258_imu_yaw.png
│   │   │   ├── 20190516-174258_trajectory.png
│   │   │   ├── 20190516-174258_yaw_error.png
│   │   │   ├── 20190516-174258_yaw_errors_difference.png
│   │   │   ├── 20190516-174258_yaw_errors_psd.png
│   │   │   └── 20190516-174258_yaw_psd.png
│   │   ├── 2
│   │   │   ├── 20190516-160849_control_error.png
│   │   │   ├── ...
│   │   └── 3
│   └── sigma=1.118, fd=0.001
│       ├── 1
│       ├── 2
│       └── 3
├── trajectory B
│   ├── 1
│   ├── 2
│   └── 3
├── trajectory C
│   ├── 1
│   ├── 2
│   └── 3
└── trajectory D
    ├── 1
    ├── 2
    └── 3
```

Figure B.1: The structure of the directory `experiment results`, all numbered directories include similar files to the content of the directory `trajectory A/sigma=0.5, fd=0.005/1/`

```
├── experiment results
│   ├── trajectory A
│   │   ├── sigma=0.5, fd=0.005
│   │   └── sigma=1.118, fd=0.001
│   ├── trajectory B
│   ├── trajectory C
│   └── trajectory D
├── IMU models
│   ├── components
│   │   ├── imu_sensor.config.xacro
│   │   ├── imu_sensor.gazebo.xacro
│   │   ├── imu_sensor_hector.gazebo.xacro
│   │   ├── imu_sensor_hector.urdf.xacro
│   │   └── imu_sensor.urdf.xacro
│   ├── imu_hector.xacro
│   ├── imu.xacro
│   ├── LICENSE
│   ├── README.html
│   ├── README.md
│   └── README.pdf
└── imu_simulation_framework
    ├── CMakeLists.txt
    ├── config.ini
    ├── flowchart.png
    ├── launch
    │   ├── error_publisher.launch
    │   ├── imu_yaw_publisher.launch
    │   ├── path_follower.launch
    │   └── result_comparison.launch
    ├── LICENSE
    ├── msg
    │   ├── ErrorCoordinates2.msg
    │   └── ImuYaw.msg
    ├── offline processing
    │   ├── compare_yaw.py
    │   ├── offline_result_comparison.py
    │   └── plot_multiple_errors.py
    ├── package.xml
    ├── README.html
    ├── README.md
    ├── README.pdf
    ├── rqt_multiplot_imu_config.xml
    ├── scripts
    │   ├── error_publisher.py
    │   ├── imu_yaw_publisher.py
    │   ├── __init__.py
    │   ├── path_follower.py
    │   ├── result_comparison.py
    │   ├── utils.py
    │   └── utils.pyc
    ├── setup.py
    └── srv
        └── StartStopTime.srv
```

Figure B.2: The structure of the root directory on the attached CD