POLITECNICO DI TORINO

Master of Science Degree in
Mechatronic Engineering

Real-Time State Estimation of a Two-Wheeled Inverted Pendulum Robot for Motion and Navigation Control

Master's thesis

Author:	Federico Casali
Examiner:	Prof. Alessandro Rizzo
Supervisor:	Ing. Dustin Lehmann

Academic Year 2020/2021

Abstract

Robotic systems appear in a wide varieaty of fields including manufactury, space exploration, laboratory research and surgery. The spread of robotic technology in this wide range of applications has been motivated by their capability to perform jobs more accurate, more reliable and, last but not least, the possibility of eliminating harmful task for humans. The complex environments where robotic systems are employed require them to be able to adapt an learn. To develop different motion learning strategies and evaluate them in real-world applications, a robotic testbed has been developed capable of performing dynamic maneuvers. An autonomous sytem relies on its perception of its own status and the environment to make decisions and compute new actions. Therefore, a key aspect in making a robot really autonomous is the state estimation problem.

The investigated robotic system is a Two-wheeled inverted pendulum robot (TWIPR), with instable and complex dynamics. In order to develop control strategies to stabilize the system and enable motion control of the orientation, velocity and position one needs a reliable, accurate and real-time capable estimation of the system's motion states. In order to estimate all dynamic states of the system, the robotic testbed is equipped with different sensors, such as inertial measurement units (IMUs), odometry sensors and an optical motion capture system.

The estimation problem can be separated into the rotational and the translational states. Tracking of the robot's orientation (more specifically it's inclination) is a key aspect for the state-feedback controller that stabilizes the system. We therefore implement a fast, accurate and magnetometer-free estimation algorithm based on an extended Kalman filter (EKF) to track the robot's inclination to enable a high-frequency attitude controller. The translational states such as the robot's velocity and position are used for motion and navigational control in upcoming learning experiments. The multitude of different sensors providing information on the motion states and the inherent challenges of localization tasks and non-linearities in such require the use of a particle filter-based approach. This type of filter makes no assumptions on the shape of uncertainties of the different information sources and can track different hypothesis of the motion states simultaneously. This makes such a filter ideal to deal with competing information, high uncertainties, non-linearities and temporal loss of information sources. This last scenario is often encountered if the main source of localization information is prone to occlusion such at is the case with the optical motion capture system.

The estimation algorithms are tested in simulations. The main objective of the simulational studies is to identify optimal filter parameters for different scenarios and to identify the method's sensitivity to different sources of error and uncertainties. For this, simulational models for the sensors and the robot are identified and implemented.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, the topic of the thesis is introduced briefly. The problem, the main objectives and a short description of the general structure of the thesis are given, to serve as an overview.

1.1 Motivation

The span of robotic system applications can be very broad, including but not limited to communication and entertainment, manufactury and assembly, lifestyle support robots, rescue, transportation, medicine, etc. Robots are seen as machines that are able to interact and modify the environment in which they operate. This means that they must deal with technological fields and issues of various complexities, so it is important for them to be able to adapt and learn. Robots operate by carrying out actions conditioned by the data that they acquire through sensors on its own status and the surroundings, as well as by intrinsic rules of behaviour programmed in them.

The task of connecting perception to action is entrusted to the control system. This provides the robot with a certain degree of autonomy to make decisions in order to accomplish pre-established goals. This is particularly important in the case of mobile robots since the environments in which they operate are likely to be unknown and of complex nature. When a robot is moving to a certain location, it can encounter obstacles in its path, whether they are known in advance or not. Thus, it needs to be able to detect those obstacles and take actions to avoid collisions that may cause damage to the robot, the obstacle and, more important, injuries to humans.

A key aspect to make a robot autonomous comes from the capability of perception. This is entrusted to the sensory system which can aquire information on certain inner and outer variables. However, some important states that define the robot's own status may be hidden or not directly measurable. Furthermore, sensory data is usually corrupted by noise and disturbances and could lead to wrong conclusions. This is why a reliable, accurate and real-time capable estimation of the system's motion states is necessary.

To develop different motion learning strategies and evaluate them in real-world applications, a robotic testbed has been developed capable of performing dynamic maneuvers. The investigated robotic system is a Two-wheeled inverted pendulum robot (TWIPR), shown in the figure below, with instable and complex dynamics. In order to develop control strategies to stabilize the system and enable motion control of the orientation, velocity and position one needs a reliable, accurate and real-time capable estimation of the system's motion states. In order to estimate all dynamic states of the system, the robotic testbed is equipped with different sensors, such as inertial measurement units (IMUs), odometry sensors and an optical motion capture system.

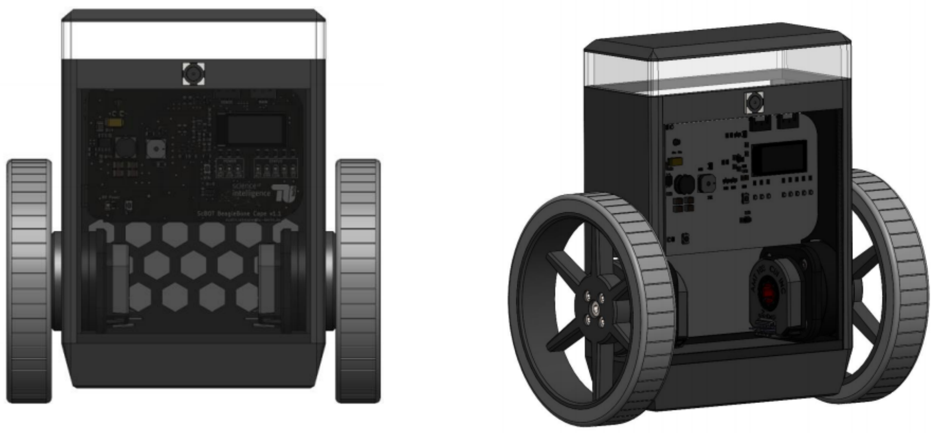


Figure 1.1: A TWIPR

Inertial Measurement Units (IMUs) have the advantage of low-cost, wide field of application and they require no direct interaction with the object of interest. Therefore, they are used in a wide variety of robotic applications. The principle of orientation estimation with inertial sensors relies on the measurement of the angular velocity, the acceleration and the magnetic field strength and a fusion algorithm.

Due to the large number of application areas in which the use of inertial sensors for orientation and position estimation is involved, there is a large amount of literature about it. Usually, the estimation problems are nonlinear and there are many different parametrizations of orientation that can be considered depending on their properties. Therefore, it is important to carefully select the most suitable model and algorithms to improve the accuracy of the estimates.

Odometry is also commonly employed in mobile robots for position estimation. They provide an accurate measurement of the translational velocity and the heading of the robot. Although wheel odometry is the simplest technique available for position estimation, it is quite sensible to systematic and non-systematic errors affecting the entire drivetrain system, specially to wheel slippage, which is very likely in differential-drive robots like the TWIPR.

1.2 Objective of the thesis

The objective of this thesis is to study the dynamics of the TWIPR and how they affect the sensorial data regarding state estimation. This will be used to derive robust methods for the inertial orientation and position estimation of this particular differential-drive mobile robot. The algorithms derived are aimed to minimize the overall tracking error between the true and estimated states by exploiting the dynamic properties of the robot. The algorithms will be studied on simulations aimed to real-world application. Here, different assumptions made during the derivation of the system dynamics or the filtering algorithms will be dropped to see how the estimation problem is affected.

1.3 Structure

To obtain accurate position and orientation estimates using inertial sensors in combination with additional measurements or models, a number of important things need to be considered. First, an introduction to the system under study is given, where both the robot and the sensors used will be presented. In addition, the quantities measured by the inertial sensor and the information provided by odometry need to be accurately described and the sources of error need to be characterized. This is the topic of Chapter 2.

In Chapter 3 we will discuss different parametrizations of orientation. This will highlight the challenges in parametrizing and estimating orientations and show that the orientation estimation problem is inherently nonlinear. Then, some basic principles of orientation estimation with inertial sensors will be introduced, together with different algorithms.

The position estimation problem will be addressed in Chapter 4. Here, an overview on relative and absolute localization techniques is given. Then, we analyse odometry and inertial navigation to identify their strenghts and weaknesess. Finally, the problem of nonlinear filtering is introduced where the particle filter is presented and all its steps explained in detail.

Simulations are pefromed in Chapter 5 to analyze the proposed methods. The entire chapter is divided into the orientation estimation methods and the position estimation methods.

Chapter 2

System Characteristics

In this chapter the main aspects of the two-wheeled inverted-pendulum robot (TWIPR) are introduced, as well as its dynamic model. The theory given here only serves as a short introduction since it is not the main focus of this thesis. In the second section, the utilized sensors are introduced and the error characteristics are studied.

2.1 The TWIPR

A two-wheeled inverted pendulum robot belongs to the class of nonlinear systems, and it consists of an inverted-pendulum body on two wheels. This kind of robotic systems are challenging, have complex dynamics and are very interesting for research related to systems with underactuated dynamycs, so it is widley used as testbed for learning problems. Furthermore, a TWIP robot is an under-actuated system, which makes it a prospective robot as an educational device for teaching or a research platform for testing advanced control methods.

The TWIPR has drawn much attention for the past decade because of its compact design, its zero radius turning ability, and its agility in narrow spaces and crowded conditions. Different devices with this kind are beeing used as personal transporters, such as the Segway, the self-balancing scooter and the self-balancing unicycle.

2.1.1 Notations and Coordinate Systems

The coordinate systems are denoted by manuscrit letters. The coordinate system axes of frame $\mathcal{A}$ are denoted as ${}_{\mathcal{A}}\mathbf{x}$, ${}_{\mathcal{A}}\mathbf{y}$ and ${}_{\mathcal{A}}\mathbf{z}$. Left subscripts denote the coordinates system in which a vector is described, left superscripts denote the coordinate system a point is belonging to. In order to discuss the system dynamics and the quantities measured by the accelerometer and gyroscope, a number of coordinate frames need to be introduced

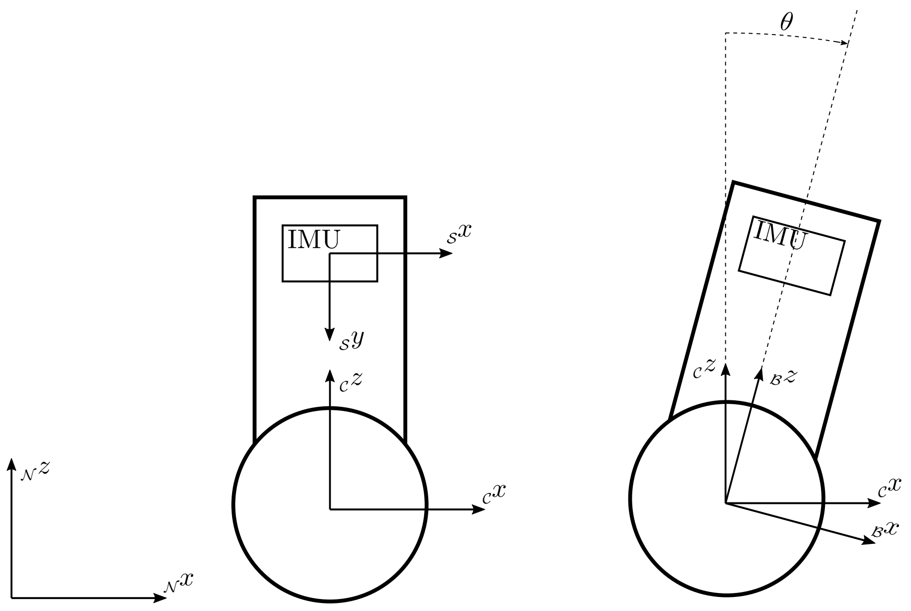


Figure 2.1: Reference frames

The Earth's frame $\mathcal{E}$ has vertical z-axis and a north-bound x-axis.

The Newtonian reference frame $\mathcal{N}$ is a fixed frame in which we want to navigate. We are interested in the position and orientation of the robot with respect to this frame.

The chassis frame $\mathcal{C}$ is attached to the body, but it's vertical axis remains parallel to the global vertical axis; in other words, this frame rotates with the body when its heading changes, but not when its elevation does.

The body frame $\mathcal{B}$ is the coordinate frame attached to the body. Depending on how the IMU is mounted on the body, the orientation of this frame can coincide with the sensor frame or not, and their origins can also differ.

The sensor frame $\mathcal{S}$ is the coordinate frame of the IMU. All the inertial measurements acquired are given in this frame.

2.1.2 System Dynamics

From a modelling point of view, the robot can be seen as two wheels W and a body B forming the inverted pendulum, as shown in Figure 2.2. The whole body of the robot moves together by straight motion and yaw rotation and the inverted pendulum has additional pitch motion. The pitch angle of the pendulum is denoted by $\theta \in [-90°, 90°]$ and the yaw angle of the robot is denoted by $\psi \in [0°, 360°]$. Assuming that there is no

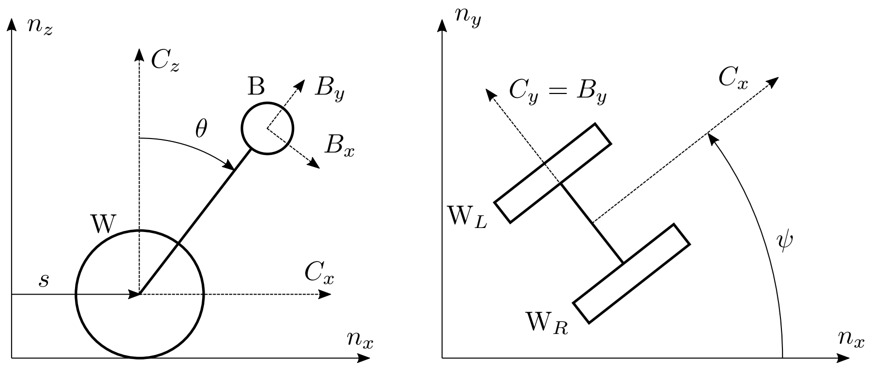


Figure 2.2: Mechanical symplification o a TWIPR and its degrees of freedom.

slip on the ground, the resulting equations of motion are given by

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + D\dot{q} + G(q) = B\tau \tag{2.1}$$

Where inertia matrix $M \in \mathbb{R}^{3\times3}$, centrifugal and Coriolis matrix $C \in \mathbb{R}^{3\times3}$, damping matrix $D \in \mathbb{R}^{3\times3}$, gravity vector $G \in \mathbb{R}^3$, input matrix $B \in \mathbb{R}^{3\times2}$, input vector $\tau \in \mathbb{R}^2$, and generalized coordinates $q \in \mathbb{R}^3$ have the following forms

$$M = \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ 0 & 0 & a_{33} \end{bmatrix}, \qquad C = \begin{bmatrix} 0 & c_{12} & c_{13} \\ 0 & 0 & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}, \qquad q = \begin{bmatrix} s \\ \theta \\ \psi \end{bmatrix},$$

$$D = \begin{bmatrix} d_{11} & d_{12} & 0 \\ d_{21} & d_{22} & 0 \\ 0 & 0 & d_{33} \end{bmatrix}, \qquad B = \begin{bmatrix} 1/r & 1/r \\ -1 & -1 \\ -d/2r & d/2r \end{bmatrix}, \qquad \tau = \begin{bmatrix} \tau_L \\ \tau_R \end{bmatrix},$$

$$G = \begin{bmatrix} 0 & -m_B l g \sin(\theta) & 0 \end{bmatrix}^T$$

These equations of motion are derived in [1]. The elements of the matrices above are listed below, and they depend on the parameters in Table 2.1.

$$\begin{aligned}
a_{11} &= m_B + 2m_W + 2J/r^2 \\
a_{12} &= a_{21} = m_B l \cos\theta \\
a_{22} &= I_2 + m_B l^2 \\
a_{33} &= I_3 + 2K + (m_W + J/r^2)d^2/2 - (I_3 - I_1 - m_B l^2)s^2\theta \\
c_{12} &= -m_B l\dot{\theta}\sin\theta \\
c_{13} &= -m_B l\dot{\psi}\sin\theta \\
c_{23} &= (I_3 - I_1 - m_B l^2)\dot{\psi}\sin\theta\cos\theta \\
c_{31} &= m_B l\dot{\psi}\sin\theta
\end{aligned}$$

$$
\begin{aligned}
c_{32} &= -(I_3 - I_1 - m_B l^2)\dot{\psi}\sin\theta\cos\theta \\
c_{33} &= -(I_3 - I_1 - m_B l^2)\dot{\theta}\sin\theta\cos\theta \\
d_{11} &= 2c_\alpha / r^2 \\
d_{12} &= d_{21} = -2c - \alpha / r \\
d_{22} &= 2c_\alpha \\
d_{33} &= (d^2/2r^2)c_\alpha
\end{aligned}
$$

Parameter	**Value**	**Description**
m_B	–	mass of the pendulum body
m_W	–	mass of a wheel
l	–	distance between the wheel axis and the pendulum's center of gravity
r	–	wheel radius
d	–	distance between the two wheels
J	–	moment of inertia of a wheel with respect to the reference frame C in the direction of c_2
K	–	moment of inertia of a wheel with respect to the reference frame C in the direction of c_3
I_1	–	moment of inertia of the pendulum's body with respect to the reference frame B in the direction of b_1
I_2	–	moment of inertia of the pendulum's body with respect to the reference frame B in the direction of b_2
I_3	–	moment of inertia of the pendulum's body with respect to the reference frame B in the direction of b_3
c_α	–	viscous friction coefficient

Table 2.1: Parameters of the mechanical system.

2.1.3 Feedback Controller

A lot of research has been done on control of TWIPR in past few decades. As in [3] a method based on adaptive fuzzy control has been proposed. The simulation results showed that the method was able to control the system for larger initial angles and also improves the stability. In [4] a method using reinforcement learning and fuzzy neural networks has been presented. The method not only controls the task of continuous states and actions but also maintain the balance of the robot in short time. In [5] authors designed a pole placement feedback controller and fuzzy logic controller for stabilizing a two-wheeled self-balancing robot. The system structure model was built using the kinetic equations based on Newton dynamics mechanics theory. The simulation results indicated better dynamics performance of fuzzy controller as compared to other controller. In [6] a balance control of two-wheeled robot by fuzzy and PID control has been studied. The study implemented the control circuit to a real robot model. In [7] a fuzzy gaussian neural network (FGNN) controller for controlling the speed and azimuth of mobile robot driven by two independent wheels has been proposed. The learning controller consisted of two FGNNs based on independent reasoning and connection weights. In [8] three different controllers i.e. fuzzy logic controller, linear quadratic controller and PID controller were implemented and compared on a real time TWIPR.

The robotic testbed in which the methods proposed here will be evaluated makes use of a state feedback with eigenvector assignment like shown in Figure 2.3 that allows the robot to reach a region of attraction of 90° for the pitch angle, and high maneuverability. The input to the TWIPR and the state are defined as

$$\boldsymbol{u} = \begin{bmatrix} M_R & M_L \end{bmatrix}^T \tag{2.2}$$

$$\boldsymbol{x} = \begin{bmatrix} s & \dot{s} & \theta & \dot{\theta} & \psi & \dot{\psi} \end{bmatrix}^T \tag{2.3}$$

where M_R, M_L are the motors torque.

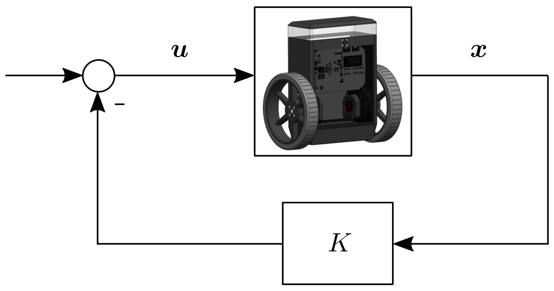


Figure 2.3: Inner-loop control with state feedback.

The design of K allows to stabilize the system, decouple ψ and θ, and ensure performance requirements established before.

In addition to the previous inner-loop control, the robot also counts with an outer-loop to perform translational and steering velocity control using PID controllers (see Figure 2.4).

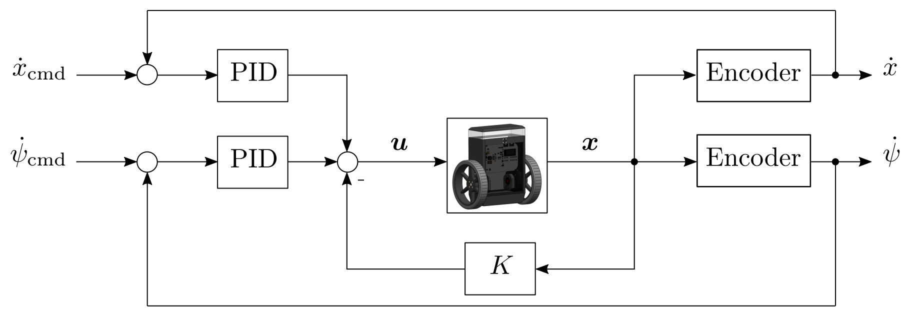


Figure 2.4: Outer-loop control with PID controllers.

2.2 Sensors

In this section the different sensors equipped by the robot are introduced, as well as other external sensors that will be used for state estimation purposes. Furthermore, their measurements error characteristics are presented.

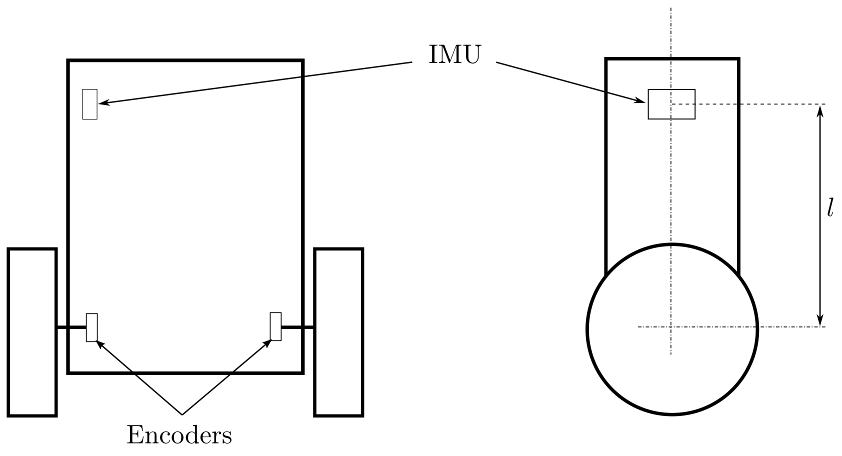


Figure 2.5: IMUs and encoders mounted on the robot.

2.2.1 Inertial Measurements Unit

Inertial measurement units are electronic devices that combine different sensors (usually accelerometers, gyroscopes and magnetometers) to provide useful information that we can use for orientation and position tracking purposes. Thanks to their small size, low prize, and the ability to provide measurements without line of sight or mechanical interaction, they are beeing used in a widley variety of application, ranging from automotive to medical applications.

Only accelerometers and gyroscopes will be covered here because magnetometers will not be used for estimation purposes due to the disturbance in the magnetic field present on close electronics (like the motors) and indoor environments [9].

Accelerometer

Accelerometers manufactured using MEMS techniques measure the proper acceleration, typically, by measuring the displacement of a supported mass that bends when the device is accelerated. Proper acceleration refers to the one the device experiences relative to free fall.

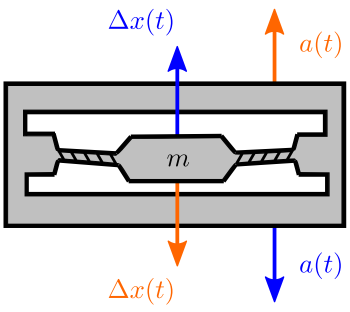


Figure 2.6: A MEMS accelerometer [10].

The specic force $\mathbf{f} \in \mathbb{R}^3$ is measured in the sensor frame $\mathcal{S}$ at equidistance time instants t_k with sample time t_s.This specific force can be expressed as

$$ {}_{\mathcal{S}}\mathbf{f}(t) = {}^{\mathcal{N}}_{\mathcal{S}}R\left({}_{\mathcal{N}}\boldsymbol{a}_{ii}(t) - {}_{\mathcal{N}}\boldsymbol{g}(t)\right) \tag{2.4}$$

where $\boldsymbol{g} \in \mathbb{R}^3$ denotes the gravity vector and ${}_{\mathcal{N}}\boldsymbol{a}_{ii} \in \mathbb{R}^3$ denotes the linear acceleration of the sensor expressed in the navigation frame. The subscripts ii indicate that the differentiation is performed in the inertial frame. For navigation purposes, we are interested in the position of the sensor in the navigation frame ${}_{\mathcal{N}}\mathbf{p} \in \mathbb{R}^3$ and its derivatives as performed in the navigation frame. Assuming again that the navigation frame is fixed to the earth frame, it is possible to express ${}_{\mathcal{N}}\boldsymbol{a}_{ii}$ in terms of ${}_{\mathcal{N}}\boldsymbol{a}_{nn}$ as

$$ {}_{\mathcal{N}}\boldsymbol{a}_{ii}(t) = {}_{\mathcal{N}}\boldsymbol{a}_{nn}(t) + 2\,{}_{\mathcal{N}}\boldsymbol{\omega}_e(t) \times {}_{\mathcal{N}}\mathbf{v}_n(t) + {}_{\mathcal{N}}\boldsymbol{\omega}_e(t) \times {}_{\mathcal{N}}\boldsymbol{\omega}_e(t) \times {}_{\mathcal{N}}\mathbf{p}(t) \tag{2.5}$$

where ${}_{\mathcal{N}}\boldsymbol{\omega}_e \in \mathbb{R}^3$ is the angular velocity of the earth with respect to a stationary frame, expressed in the navigation frame; and ${}_{\mathcal{N}}\boldsymbol{a}_{nn}$ is the acceleration required for navigation purposes. The term ${}_{\mathcal{N}}\boldsymbol{\omega}_e \times {}_{\mathcal{N}}\boldsymbol{\omega}_e \times {}_{\mathcal{N}}\mathbf{p}$ is known as the *centrifugal acceleration* and $2\,{}_{\mathcal{N}}\boldsymbol{\omega}_e \times {}_{\mathcal{N}}\mathbf{v}_n$ is known as the *Coriolis acceleration*. The full derivation of this relation is derived in [2]. The centrifugal acceleration is typically absorbed in the gravity vector, while the magnitude of the Coriolis acceleration can be neglected for beeing to small compared to the measured accelerations.

Gyroscope

MEMS gyroscopes make use of the Coriolis effect, which states that in a reference frame rotating at angular velocity $\boldsymbol{\omega}(t) \in \mathbb{R}^3$, a mass m moving with velocity $\mathbf{v}(t) \in \mathbb{R}^3$ experiences a force $\mathbf{F}_c(t) = 2m(\mathbf{v}(t) \times \boldsymbol{\omega}(t))$ These gyroscopes use a vibrating structure to measure the Coriolis effect by exploiting the fact that vibrating structures tend to continue vibrating in their own plane even if their support rotates. Thus, when the gyroscope is rotated, a secondary vibration is induced along the perpendicular sense axis due to the Coriolis force. The angular velocity can be determined by measuring this secondary rotation; the absolute value will be given by the oscillation amplitude, while the direction of rotation can be obtained from the phase difference between both oscillations.

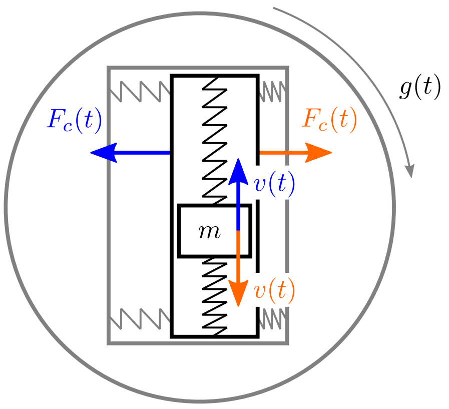


Figure 2.7: A MEMS gyroscope [10].

The gyroscope measurement ${}_{\mathcal{S}}\boldsymbol{\omega}_{\text{total}}$ expresses the angular velocity of the sensor frame with respect to the inertial frame, expressed in the former one. This velocity is composed of:

$$ {}_{\mathcal{S}}\boldsymbol{\omega}_{\text{total}}(t) = {}^{\mathcal{N}}_{\mathcal{S}}R\left({}_{\mathcal{N}}\boldsymbol{\omega}_{\text{e}}(t) + {}_{\mathcal{N}}\boldsymbol{\omega}_{\text{tr}}(t)\right) + {}_{\mathcal{N}}\boldsymbol{\omega}_{\text{nv}}(t) \tag{2.6}$$

where ${}^{\mathcal{N}}_{\mathcal{S}}R$ is the rotation matrix from the navigation frame to the sensor frame. The angular velocity of the earth frame with respect to the inertial frame is denoted by ${}_{\mathcal{N}}\boldsymbol{\omega}_{\text{e}}$. This *earth rate* is approximately $7.29 \cdot 10^{-5}$ rad/s and the rotation is about the earth's own z-axis. The term ${}_{\mathcal{N}}\boldsymbol{\omega}_{\text{tr}}$ represents the *transport rate*, which is non-zero whenever the navigation frame is not defined stationary with respect to the earth. The angular velocity required for navigation purposes, i.e. the *navigation rate*, is denoted by ${}_{\mathcal{N}}\boldsymbol{\omega}_{\text{nv}}$ and is the one needed to determine the orientation of the body with respect to the navigation frame.

Since the robot (and thus the sensor) will not travel over significant distances, the navigation frame $\mathcal{N}$ can be assumed stationary and with it, the transport rate is zero. Also, the magnitud of the earth rate is small compared to the actual rotations that the robot will experience; therefore, it can be neglected.

2.2.2 Encoders

Incremental encoders measure changes in position and the direction of movement of a rotary device to which is attached. It has two quadrature-encoded outputs, A and B, which generate pulses when the device is rotated. The direction of motion is indicated by

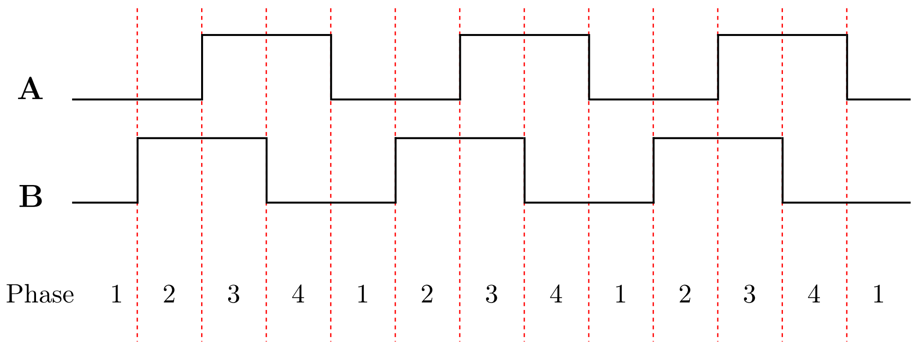


Figure 2.8: Encoder channels

the phase difference between both channels, being positive for one direction and negative for the other; and the angular velocity of the rotatory device is directly proportional to the frequency of the pulses emitted. An absolute position of the encoder can be obtained by using an up/down counter to count incremental position changes. The cumulative count gives the distance traveled since the tracking began, so the initial position of the device must be known prior to the beginning of the tracking. Every signal edge on channel A or B is an indication of a position change. Since each square-wave cycle on A or B contains four signal edges, the resolution of the encoder is one-fourth of the displacement represented by a full cycle. In the case of our robot, each wheel is equipped with incremental encoders that produce 1024 pulses per revolution. Multiplying this by the gear ratio $\tau_G = \frac{5175}{247}$ gives the number of pulses per wheel revolution. The encoder will then have a cycle resolution of $2\pi/(1024\tau_G)$, so the encoder resolution is

$$Res = \frac{2\pi}{1024\tau_G}\frac{1}{4} = \frac{\pi}{2048\tau_G} = 0.000073rad \tag{2.7}$$

2.2.3 Motion Capture System

Motion capture systems utilize data captured from camera sensors to triangulate the 3D position of an object between two or more cameras calibrated to provide overlapping projections. Data acquisition is traditionally implemented using special markers attached to the object. Tracking a large number of markers or expanding the capture area is accomplished by the addition of more cameras. These systems produce data with three degrees of freedom for each marker, and rotational information must be inferred from the relative orientation of three or more markers. The software available in the lab gives the possibility to group markers to form a rigid body (minimum 3 markers), which allows to track also the body's' orientation.

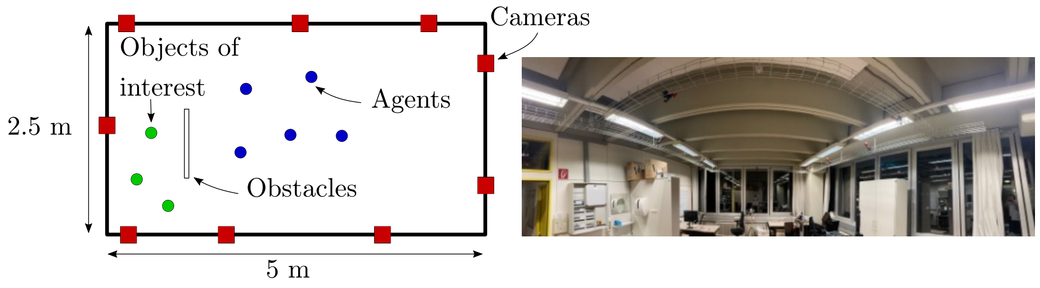


Figure 2.9: Motion capture system layout and workspace.

The optical motion capture system used for validation is an OPTITRACKFLEX13 system [11]. It uses 10 infrared cameras, distributed at the ceiling of the measuring space. The cameras have a resolution of 1280×1024 px, recording at a frame rate of 120 FPS. The software used for capturing and recording the marker positions is Motive 2.0 by OptiTrack. The cameras are calibrated at the beginning of each set of measurements. After thorough calibration, a mean position error (RMS) of 0.3 mm for each marker can be obtained. The coordinate system is calibrated by a calibration square, which can be adjusted in order for the vertical axis to coincide with the global vertical axis. The TWIPR uses active markers mounted on the top, which are special LEDs manufactured by OptiTrack, designed specifically to work with their cameras.

2.3 Error Characteristics

In this section the errors affecting the measurements of the sensors defined in the previous section are studied.

Typical sources of error affecting the measured signal in IMU sensors can be classified into calibration errors and non-calibration errors. The first ones can be eliminated with a proper calibration process, but the second ones will always be present and must be dealt with.

Regarding the encoders, since they are part of the entire drivetrain system, we are interested in the system as a whole, and not just the sensors.

2.3.1 Accelerometers and Gyroscopes

Scaling Errors

This type of errors is given by the ratio between the true value and the measured one. They can be removed by introducing multiplicative gains for the measured signal, which can be determined by measuring known values. A scaling error in the accelerometer of 3% can cause a 1.72° error in the inclination angle estimation.

Linearity is a further consideration for scale factor. In case that the scaling values depend on the real signal, then a nonlinear relation exists. With these two errors, the difference between the true signal and the measured signal can be described by

$$y_{\text{measured}} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} y_{\text{true}} \tag{2.8}$$

where s_x, s_y and s_z are constant if there is linearity, or a function of the true signal if there is a non linear relation.

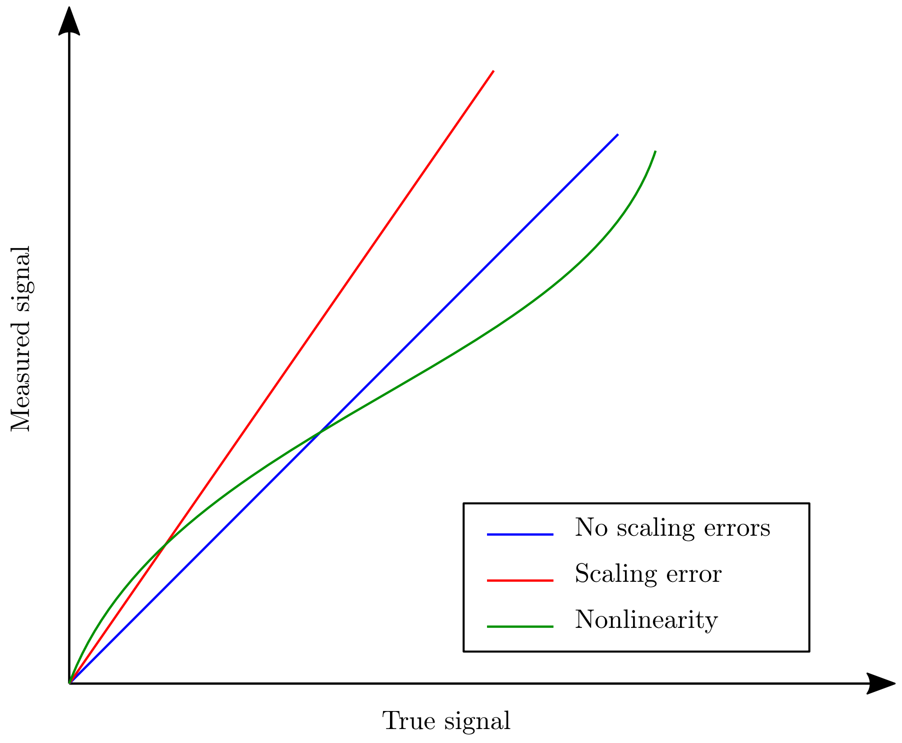


Figure 2.10: Scaling errors: measured signal over the true signal.

These kind of errors are more prone to appear in accelerometers, in gyroscope we seldomly have scaling errors.

Non-orthogonality and Misalignment

non-orthogonality $\mathbf{C}$ refers to the measurement axes not being pairwise orthogonal to each other, which leads to a correlation between sensors. Misalignment m happens when the axes of the gyroscope do not coincide with the ones of the accelerometer. Introducing these errors into (2.8) leads to

$$y_{\text{measured}} = \mathbf{C} \begin{bmatrix} s_x & m_{xy} & m_{xz} \\ m_{yx} & s_y & m_{yz} \\ m_{zx} & m_{zy} & s_z \end{bmatrix} y_{\text{true}} \tag{2.9}$$

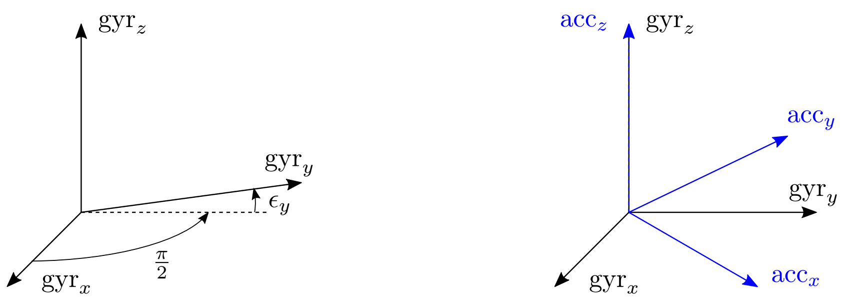


Figure 2.11: Example of non-orthogonality (left) and of misalignment between the gyroscope and the accelerometer.

To determine the orthogonality error of accelerometer axis pairs, the static response of each axis to gravity is measured as the accelerometer is rotated through the space of all possible 90° orientations. This can be done using either a precision gimbal mount or on a known orthogonal surface.

Bias

The bias of a sensor is the offset of its output signal from the true value (in m/s^2 for the accelerometer and in $^\circ/s$ for the gyroscope). A constant bias error of ϵ, when integrated, causes an error which grows linearly with time $\theta(t) = \epsilon\, t$; and when double integrated, causes an error which grows quadratically with time $\theta(t) = \epsilon\, \frac{t^2}{2}$ The bias error can be estimated by taking a long-term average of the sensor output while it is measuring a true value of zero.

However, the complete bias is composed of what can be considered as a constant part and a variable one, denoted as turn-on bias. The latter one is much smaller than the former one, so a calibration based on a long measurement can be used to estimate it and apply corrections. But every time the sensor is powered-up, the turn-on bias changes, so if an even more precise calibration is wanted, it would be necessary to estimate this bias every time the sensor is powered-up. Typical order of magnitude: $1\,°/s$ for MEMS gyroscopes and $0.1\,m/s^2$ for MEMS accelerometers.

Adding this error to (2.9) gives

$$y_{\text{measured}} = \mathbf{C}\begin{bmatrix} s_x & m_{xy} & m_{xz} \\ m_{yx} & s_y & m_{yz} \\ m_{zx} & m_{zy} & s_z \end{bmatrix} y_{\text{true}} + \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} \tag{2.10}$$

This equation contains all deterministic errors that can be eliminated by calibration measurements with a reference measurement system (e.g. actuated gimbals). With the given dataset, all calibration parameters can be obtained by solving the optimization problem

$$\underset{\mathbf{C},s,m,b}{\text{argmin}} \sum_{i}^{N \gg 1} \left(y_{\text{measured}}(t_i) - \mathbf{C}\begin{bmatrix} s_x & m_{xy} & m_{xz} \\ m_{yx} & s_y & m_{yz} \\ m_{zx} & m_{zy} & s_z \end{bmatrix} y_{\text{true}} - \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} \right)^2 \tag{2.11}$$

Thermo-Mechanical White Noise

The output of an IMU will be perturbed by some thermo-mechanical noise which fluctuates at a rate much greater than the sampling rate of the sensor. As a result, the samples obtained from the sensor are perturbed by a white noise sequence, which is simply a sequence of zero-mean uncorrelated random variables. In this case each random variable is identically distributed and has a finite variance σ^2. To see what effect this noise has on the integrated signal, let N_i be the i^{th} random variable in the white noise sequence. Each N_i is identically distributed with mean $E(N_i) = E(N) = 0$ and finite variance $Var(N_i) = Var(N) = \sigma^2$. By the definition of a white sequence $Cov(N_i, N_j) = 0$ for all $i \neq j$. The result of integrating the white noise signal $\epsilon(t)$ over a timespan $t = nT_s$ is:

$$\int_0^t \epsilon(\tau)d\tau = T_s \sum_{i=1}^{n} N_i \tag{2.12}$$

where n is the number of samples received from the device during the period and T_s is the time between successive samples. Using the standard formulae $E(aX + bY) =$

$aE(X) + bE(Y)$ and $Var(aX + bY) = a^2Var(X) + b^2Var(Y) + 2abCov(X, Y)$ (where a and b are constants and X and Y are random variables) it follows that:

$$E(\int_0^t \epsilon(\tau)d\tau) = T_s\, n\, E(N) = 0 \tag{2.13}$$

$$Var(\int_0^t \epsilon(\tau)d\tau) = {T_s}^2\, n\, Var(N) = T_s\, n\, \sigma^2 \tag{2.14}$$

Hence the noise introduces a zero-mean random walk error into the integrated signal, whose standard deviation

$$\sigma_s(t) = \sigma\sqrt{T_s\, t} \tag{2.15}$$

grows proportionally to the square root of time.

The result of double integrating the signal $\epsilon(t)$ is:

$$\int_0^t \int_0^t \epsilon(\tau)d\tau d\tau = \delta t \sum_{i=1}^{n} T_s \sum_{j=1}^{i} N_j = {T_s}^2 \sum_{i=1}^{n} (n - i + 1)N_i \tag{2.16}$$

The expected error of the double integrated signal and its variance are:

$$E(\int_0^t \int_0^t \epsilon(\tau)d\tau d\tau) = {T_s}^2 \sum_{i=1}^{n} (n - i + 1)E(N_i) = 0 \tag{2.17}$$

$$Var(\int_0^t \int_0^t \epsilon(\tau)d\tau d\tau) = {T_s}^4 \sum_{i=1}^{n} (n - i + 1)^2 Var(N_i) \approx \frac{1}{3} T_s\, t^3\, \sigma^2 \tag{2.18}$$

Thus, this introduces a second order zero-mean random walk with standard deviation that grows proportionally to $t^{3/2}$:

$$\sigma_s(t) = \sigma\, t^{3/2} \sqrt{\frac{T_s}{3}} \tag{2.19}$$

Bias Stability

Flicker noise present in MEMS cause the bias to change over time. A bias stability measurement describes how the bias of a device may change over a specified period of time, typically around 100 seconds, in fixed conditions (usually including constant temperature). Under the random walk model bias stability can be interpreted as follows; If B_t is the known bias at time t, then a 1σ bias stability of $0.01°/h$ over 100 seconds means that the bias at time $(t + 100)$ seconds is a random variable with expected value B_t and standard deviation $0.01°/h$.

As we are interested in how this error affects the orientation obtained from integrating the rate gyro signal, or the position and velocity obtained from integrating the accelerometer signal. If we assume the bias random walk model, then for the gyroscope the result of integrating the bias fluctuations is a second-order random walk in angle whose uncertainty grows proportionally to the square root of time; and for the accelerometer, a second order random walk in velocity whose uncertainty grows proportionally to $t^{3/2}$, and a third order random walk in position which grows proportionally to $t^{5/2}$. In reality bias fluctuations do not really behave as a random walk. If they did then the uncertainty in the bias of a device would grow without bound as the timespan increased. In practice the bias is constrained to be within some range, and therefore the random walk model is only a good approximation to the true process for short periods of time.

Parameter	**ISM6DSOX**	**MPU9250**
Linear acceleration zero-g level offset accuracy	$\pm 20\,\mathrm{m}g$	$\pm 80\,\mathrm{m}g$
Linear acceleration zero-g level change vs. temperature	$\pm 0.1\,\mathrm{m}g/°\mathrm{C}$	$\pm 1.5\,\mathrm{m}g/°\mathrm{C}$
Acceleration noise density	$110\,\mu g/\sqrt{\mathrm{Hz}}$	$300\,\mu g/\sqrt{\mathrm{Hz}}$
Angular rate zero-rate level	± 1 dps	± 5 dps
Angular rate typical zero-rate level change vs. temperature	$\pm 0.01\,\mathrm{dps}/°\mathrm{C}$	$\pm 0.24\,\mathrm{dps}/°\mathrm{C}$
Rate noise density	$3.8\,\mathrm{mdps}/\sqrt{\mathrm{Hz}}$	$10\,\mathrm{mdps}/\sqrt{\mathrm{Hz}}$

Table 2.2: Specifications of different commercial IMUs.

2.3.2 Odometry

Since encoder readings are used together with information on the robot characteristics for odometry or dead-reckoning purposes, instead of referring to the actual errors afecting an encoder measurement, a description of all types of error that generate inaccuracies in the translation of wheel encoder readings into linear motion will be given.

Odometry is based on the assumption that wheel revolutions can be translated into linear displacement relative to the floor. This assumption looses validity in cases of wheel skiddage or slippage. But apart from this extreme cases, there are other sources of error than can be classified into *systematic* and *non-systematic* errors:

Systematic Errors:

- Unequal wheel diameters.
- Average of actual wheel diameters differs from nominal wheel diameter.
- Actual wheelbase differs from nominal wheelbase.
- Misalignment of wheels.
- Finite encoder resolution.
- Finite encoder sampling rate.

Non-Systematic Errors:

- Travel over uneven floors.
- Travel over unexpected objects on the floor.
- Wheel-slippage.

Systematic errors are particularly grave because they accumulate constantly, but it is possible to perform some tests to measure them and apply corrections [12]. The problem with non-systematic errors is that they are unpredictable and can cause large position errors.

To transform the encoder measurements from angular velocities to a translational velocity v and a steering velocity $\dot{\psi}$, it is necessary to use some information on the dimensions of the robot; that is, the wheel radius r and the wheelbase b

$$v(t) = \frac{r(\omega_R(t) + \omega_L(t))}{2} \tag{2.20}$$

$$\dot{\psi}(t) = \frac{r(\omega_R(t) - \omega_L(t))}{d} \tag{2.21}$$

For modelling purposes, the contact point between the wheels and the ground is located at the middle of the wheels width, because it is assumed that the wheels axis are coaxial. However, in real-applications the drivetrain may suffer some elastic deformation under load, like shown in Figure 2.12. This causes a change in the actual wheelbase of the robot, reducing it from the nominal value.

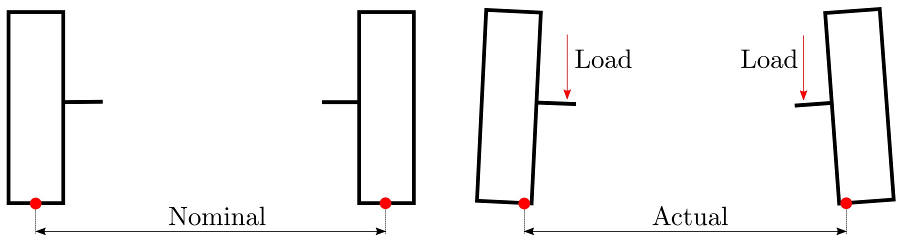


Figure 2.12: Actual and nominal wheelbase of the TWIPR.

Out of equations (2.20) and (2.21), only the latter makes use of the wheelbase, and so the estimated velocity is not affected by this error, only the heading rate. If we define the wheelbase ratio as

$$E_d = \frac{d_{\text{actual}}}{d_{\text{nominal}}} \tag{2.22}$$

and introduce it into (2.21) we get

$$\dot{\psi}(t) = \frac{r(\omega_R(t) - \omega_L(t))}{E_d d_{\text{nominal}}} \tag{2.23}$$

This shows that the result of having a wheelbase error is a scaling factor, which increases or decreases the amount of heading rate depending on whether E_d is larger or smaller than 1.

To study the effects of unequal wheel diameter, let's consider the case where both wheels are given the same rotational speed, what would be the case for example if we want the robot to traverse a straight path. Starting from equations (2.20) and (2.21) but introducing different radius for each wheel we get

$$v(t) = \frac{r_R\omega_R(t) + r_L\omega_L(t)}{2} \tag{2.24}$$

$$\dot{\psi}(t) = \frac{r_R\omega_R(t) - r_L\omega_L(t)}{d} \tag{2.25}$$

Now, rearranging both equations to write each wheel angular velocity as a function of the rest of the variables yields

$$\omega_R(t) = \frac{v}{r_R} + \frac{\dot{\psi}d}{2r_R} \tag{2.26}$$

$$\omega_L(t) = \frac{v}{r_L} - \frac{\dot{\psi}d}{2r_L} \tag{2.27}$$

If both wheels are rotating at the same speed, we can say that the right-side of equation (2.26) is equal to the right-side of equation (2.27), so

$$\frac{v}{r_R} + \frac{\dot{\psi}d}{2r_R} = \frac{v}{r_L} - \frac{\dot{\psi}d}{2r_L} \tag{2.28}$$

rearranging things to put all radius variables on one side of the equation

$$\frac{r_R}{r_L} = \frac{2v + \dot{\psi}d}{2v - \dot{\psi}d} \tag{2.29}$$

The left-side of the above equation is defined as the wheel radius ratio E_r. Using this definition, we can know solve the equation for $\dot{\psi}$

$$\dot{\psi}_{E_r} = \frac{2v(E_r - 1)}{d(E_r + 1)} \tag{2.30}$$

Note that this result is only valid when both wheels are rotating at the same speed. Nevertheless, this is just to show the effect of unequal wheel diameter in a common situation as it is driving in a straght line.

Equation (2.30) shows that if both wheels have the same radius, that is $E_r = 1$, then there is no steering rate when both wheels rotate at the same speed. But as soon as they are different, there is a resulting change in the heading. If the right wheel is larger, then the robot will start steering to the left, and if the left wheel is larger, it will steer to the right. Therefore, instead of following a straight path, the robot with unequal wheel diameters will follow a curved path.

Chapter 3

Orientation Estimation

Real-time tracking of rigid bodies orientation plays an important role in robotics applications. Extensive research has been conducted to investigate full 3-degree-of-freedom (3-DOF) orientation tracking using IMU data and combining it in different ways to enhance the advantages of each individual sensor. One common approach is to integrate the gyroscope measurements to obtain information about the orientation of the sensor; however, this alone is prone to drift because of the noise and bias affecting the measurements. Another way of estimating orientation is through vector observation, which provides an orientation estimate relative to an Earth-fixed frame by measuring at least two vectors of local frame and comparing these vectors with the known positions of vectors in an Earth-fixed frame. The most spread vector observation algorithms are FGA, QUEST and TRIAD. However, these approaches use the magnetic field measurement to estimate the orientation with respect to a vertical axis (yaw o heading). Unfortunately, indoor environments are usually affected by a variety of magnetic fields coming from different appliances, and even if this was not the case, also the robot itself generates magnetic noise due to its motors. For these reasons, the magnetometer cannot be used to determine the heading.

To overcome the problems that both approaches have, sensor fusion algorithms must be applied to combine the advantages of each approach and thus, improve the estimate.

3.1 Probabilistic Models

In this section it is introduced the concept of probabilistic modeling, which constitute the core of estimation algorithms. Also, notations used in this work will be presented. Models are useful descriptions of a system dynamics, whether it is a mobile robot or a sensor. These models will allow to develop a simulation environment to test the algorithms before being validated on the true system, and will be used in combination with the measurements to estimate a state.

The main goal here is to infer information about the states $\mathbf{x}_t \in \mathbb{R}^n$ for $t = 1, \dots, N$, using the probabilistic models and the available measurements $\mathbf{y}_t \in \mathbb{R}^q$ for $t = 1, \dots, N$. This can be expressed in terms of a conditional probability distribution

$$p(\mathbf{x}_t|\mathbf{y}_t)$$

In the estimation problem, the idea is to obtain point estimates $\hat{\mathbf{x}}_t$ and to know how certain we are about them. When the distribution is Gaussian, it is entirely described by its mean and covariance.

when all measurements $\mathbf{y}_{1:N}$ are used to obtain the posterior distribution of $\mathbf{x}_{1:N}$, we have a smoothening problem. This is generally possible to perform offline because it is necessary to wait until all the measurements are collected before computing the conditional probability distribution. In our case, we are interested in using all measurements up to and including time t to estimate the state $\mathbf{x}_t$, which is referred as a filtering problem.

Now we will make a fundamental assumption to our application, and that is that our models are Markovian and thus, all information up to the time t is contained in the state $\mathbf{x}_t$. Using Bayes' rule and the Markov property, the conditional distribution can be decomposed as

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) \propto p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) \tag{3.1}$$

where

$$p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})d\mathbf{x}_{t-1} \tag{3.2}$$

The dynamics of the system state can be modeled by means of the possibly nonlinear function $f(\cdot)$ as

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t) \tag{3.3}$$

Where $\mathbf{u}_t \in \mathbb{R}^p$ is an input to the system and $\mathbf{w}_t$ reflects the uncertainty of the dynamic model, and it is referred to as process noise.

The information provided by the measurements can be modeled by the possible nonlinear function $h(\cdot)$ as

$$\mathbf{y}_t = h(\mathbf{x}_t, \mathbf{e}_t) \tag{3.4}$$

Where $\mathbf{e}_t$ is the equivalent to $\mathbf{w}_t$, but is referred to as measurement noise. The combination of both equations forms a state space model and it is the basis of many estimation algorithms, such as the Kalman Filter.

3.2 Mathematical Representation of Orientations and Rotations

First of all, it is necessary to properly define and distinguish two commonly confused terms: orientations and rotations. The former is a state of some rigid body with respect to a reference frame; while the latter is a process that changes this state.

Euler's rotation theorem states that every sequence of rotations can be described by a rotation axis $\mathbf{j}$ and a rotation angle α. This is known as the axis-angle representation. Other more commonly used representations include rotation matrices, Euler angles and Quaternions.

3.2.1 Rotation Matrices

Consider two coordinate systems $\mathcal{A}$ and $\mathcal{B}$. The rotation matrix that transforms an arbitrary vector ${}_{\mathcal{A}}\mathbf{p}$ from system $\mathcal{A}$ to system $\mathcal{B}$ is the unique matrix ${}^{\mathcal{A}}_{\mathcal{B}}R$ that satisfies

$${}_{\mathcal{B}}\mathbf{p} = {}^{\mathcal{A}}_{\mathcal{B}}R\,{}_{\mathcal{A}}\mathbf{p} \qquad\qquad \forall_{\mathcal{A}}\mathbf{p} \in \Re^3 \tag{3.5}$$

Every rotation matrix corresponds to exactly one orientation and vice versa. Consequently, every rotation matrix corresponds to two different rotations (both of which lead to the same orientation).

3.2.2 Euler Angles

Every orientation can be produced by a concatenation of three rotations around major coordinate axes and can hence be described by the corresponding three angles. The three elemental rotations may be extrinsic (rotations about the axes of the original coordinate system), or intrinsic (rotations about the axes of the rotating coordinate system, attached to the moving body, which changes its orientation after each elemental rotation). The most commonly employed convention is z-y-x, which first rotates an angle ψ around the z-axis, subsequently an angle θ around the y-axis and finally an angle ϕ around the x-axis.

Using Euler angles to describe orientations can present some singularities. Considering the previous intrinsic convention z-y-x, if the second angle is set to $\theta = 90°$, then a rotation around x-axis is indifferent from a rotation around z-axis. Thus, the orientation can be represented by an infinite number of Euler angles. This singularity is also known as gimbal lock.

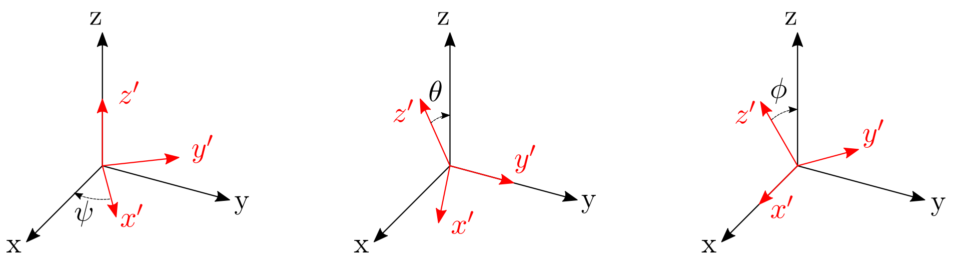


Figure 3.1: Euler angles.

3.2.3 Unit Quaternions

A commonly used parametrization of orientation is that of unit quaternions. This representation has the advantage of avoiding gimbal lock problems that arise when using Euler angles. A unit quaternion is defined as a complex number with 3 imaginary parts, according to

$$\mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} \qquad q_0, q_1, q_2, q_3 \in \mathbb{R} \qquad \|\mathbf{q}\|_2 = 1 \tag{3.6}$$

The imaginary part is called the vector part, while q_0 is the real part. This does not represent a unique description of an orientation since $-\mathbf{q}$ represents the same orientation as $\mathbf{q}$. Because any arbitrary rotation can be described by only three independent parameters, the four parameters of the unit quaternion are constrained by the relation

$${q_0}^2 + {q_1}^2 + {q_2}^2 + {q_3}^2 = 1 \tag{3.7}$$

According to Euler's rotation theorem, any rotation or sequence of rotations of a rigid body or coordinate system about a fixed point is equivalent to a single rotation by a given angle θ about a fixed axis that runs through the fixed point. Then, a rotation of θ around an axis defined by a unit vector $\mathbf{u} = u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k}$ can be represented by a quaternion using Euler's formula

$$\mathbf{q} = \cos\frac{\theta}{2} + (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})\sin\frac{\theta}{2} \tag{3.8}$$

Applying a rotation to a vector $\mathbf{p}$ can be achieved by evaluating the conjugation of $\mathbf{p}$ by $\mathbf{q}$, considering $\mathbf{p}$ as a quaternion with zero real part

$$\mathbf{p}' = \mathbf{q}\mathbf{p}\mathbf{q}^{-1} \tag{3.9}$$

where $\mathbf{q}^{-1} = \cos\frac{\theta}{2} - (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})\sin\frac{\theta}{2}$ is the inverse of $\mathbf{q}$.

3.2.4 Heading and Inclination

Orientations can also be decomposed into a heading and inclination. The heading angle Ψ describes the orientation of the body in the horizontal plane; and the inclination angle Θ is the angle between the local z-axis and the vertical z-axis of the reference frame.

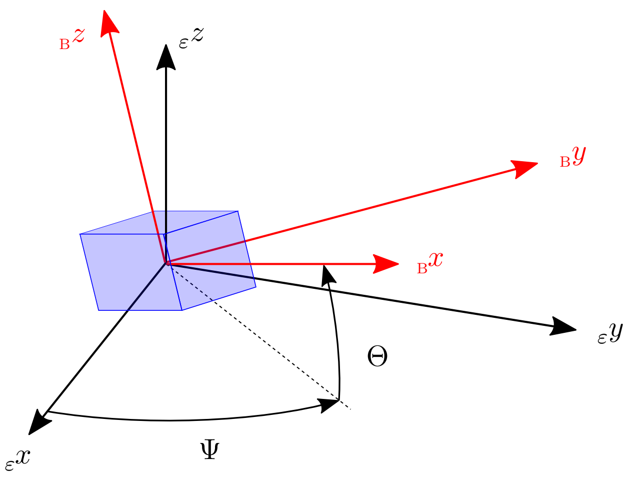


Figure 3.2: Heading and inclination angles.

With this representations, almos every quaternion can be described by

$$\mathbf{q} = \left(\Psi\,@\begin{bmatrix}0\\0\\0\end{bmatrix}\right) \otimes (\Theta\,@\,\mathbf{k}),\ \mathbf{k} \perp \begin{bmatrix}0\\0\\1\end{bmatrix} \tag{3.10}$$

This angles can be obtained from a given quaternion $\mathbf{q} = \begin{bmatrix}w & x & y & z\end{bmatrix}^T$ by first computing the heading as

$$\Psi = 2\,\mathrm{atan2}(z, w) \tag{3.11}$$

and the inclination is computed using the residual quaternion defined as

$$\mathbf{q}_{\mathrm{res}} = \left(\Psi\,@\begin{bmatrix}0\\0\\1\end{bmatrix}\right)^{-1} \otimes \mathbf{q} \tag{3.12}$$

with $\mathbf{q}_{\mathrm{res}} = \begin{bmatrix}w_{\mathrm{res}} & x_{\mathrm{res}} & y_{\mathrm{res}} & z_{\mathrm{res}}\end{bmatrix}^T$. Then, the inclination is computed as

$$\Theta = 2\ \arccos(w_{\mathrm{res}}) \tag{3.13}$$

3.3 Basics of Orientation Estimation

A simple way to estimate the orientation of a body is to integrate the angular rate from an initial known orientation. Thus, we will use as dynamic model of the system to predict the current orientation the strapdown integration of the gyroscope measurement given by

$$ {}^{S}_{N}\mathbf{q}_k = {}^{S}_{N}\mathbf{q}_{k-1} \otimes \begin{bmatrix} \cos(\frac{1}{2}\|\mathbf{g}_k\|T_s) \\ \sin(\frac{1}{2}\|\mathbf{g}_k\|T_s)\frac{\mathbf{g}_k}{\|\mathbf{g}_k\|} \end{bmatrix} \tag{3.14} $$

This gives good results even when the body experiences rapid moves. However, due to the noise and bias affecting the gyroscope measurements, their integration will lead to drift which will increase over time.

Another way to estimate orientation is by vector observation, which provides an orientation estimate relative to a fixed frame by measuring a vector on a local frame and comparing it with the known position of the vector in a fixed frame. This can be applied to gravity and Earth's magnetic field. Assuming that the body is not experiencing other accelerations, the measured acceleration should be equal to the gravitational acceleration transformed to the sensor frame:

$$ \boldsymbol{a}_k = \mathbf{q}_k{}^{-1} \otimes \begin{bmatrix} 0 & 0 & g \end{bmatrix}^T \otimes \mathbf{q}_k + \mathbf{w}_k \tag{3.15} $$

This is useful to determine the inclination of the body, but it does not affect the heading of the estimate. As said before, this approach relies on the assumption that the body is not experiencing any other acceleration than the gravitational one. This assumption will not hold whenever its velocity changes and will lead to wrong estimates.

3.4 Sensor Fusion

Sensor fusion is about combining information from different sensor sources. It has become a synonym for state estimation, which can be interpreted as a special case of parameter estimation where the parameters are the state of the system under consideration.

To produce a more accurate orientation estimate, the orientations estimated by angular rate integration and vector observation should be fused to compensate disadvantages of each other.

3.4.1 Extended Kalman Filter

Since we are working with quaternions, and their multiplication is highly nonlinear, to fuse the information from the different sensors, we propose an Extended Kalman Filter. Assuming that the measurement noise is additive and that both the process and the measurement noise are zero-mean Gaussian with constant covariance, the state space model is given by

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k \tag{3.16}$$
$$\mathbf{y}_k = h(\mathbf{x}_k) + \mathbf{e}_k \tag{3.17}$$

where k is the sample index, $\mathbf{x}_k \in \mathbb{R}^4$ is the state of the system, $\mathbf{u}_k \in \mathbb{R}^3$ the input and $\mathbf{y}_k \in \mathbb{R}^3$ the measurements. $\mathbf{w}_k \sim N(0, W)\,\forall k$ represents the system noise, and $\mathbf{e}_k \sim N(0, E)\,\forall k$ the measurement noise. In this application, we define the state, the input, and the measurements as

$$\mathbf{x}_k = {}^S_N\mathbf{q}_k = \begin{bmatrix} q_{0,k} & q_{1,k} & q_{2,k} & q_{3,k} \end{bmatrix}^T \tag{3.18}$$
$$\mathbf{u}_k = \mathbf{g}_k = \begin{bmatrix} \mathrm{g}_{x,k} & \mathrm{g}_{y,k} & \mathrm{g}_{z,k} \end{bmatrix}^T \tag{3.19}$$
$$\mathbf{y}_k = \boldsymbol{a}_k = \begin{bmatrix} a_{x,k} & a_{y,k} & a_{z,k} \end{bmatrix}^T \tag{3.20}$$

With these definitions, the nonlinear system for the orientation dynamics is

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k = \mathbf{x}_{k-1} \otimes \begin{bmatrix} \cos(\frac{1}{2}\|\mathbf{u}_k\|T_s) \\ \sin(\frac{1}{2}\|\mathbf{u}_k\|T_s)\frac{\mathbf{u}_k}{\|\mathbf{u}_k\|} \end{bmatrix} + \mathbf{w}_k \tag{3.21}$$
$$\mathbf{y}_k = h(\mathbf{x}_k) + \mathbf{e}_k = {\mathbf{x}_k}^{-1} \otimes \begin{bmatrix} 0 & 0 & g \end{bmatrix}^T \otimes \mathbf{x}_k + \mathbf{e}_k \tag{3.22}$$

To ensure faster convergence it is wise to choose an initial orientation $\mathbf{x}_0$ that is close to the true orientation; otherwise it can be randomly chosen or it can be set to the identity quaternion. If the orientation is known, then it can be used along with its corresponding covariance matrix; if not, a zero state (identity quaternion) is assumed with a high covariance matrix.

Then, for each time step, the following operations have to be performed:

1. Compute the partial derivative matrix:

$$\mathbf{F}_{k-1} = \frac{\partial f(\mathbf{x}_{k-1}, \mathbf{u}_k)}{\mathbf{x}_{k-1}} \tag{3.23}$$

2. Perform the time update of the state estimate and estimation-error covariance:

$$\hat{x}_k^- = f(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k) \tag{3.24}$$
$$\mathbf{P}_k^- = \mathbf{F}_{k-1}\mathbf{P}_{k-1}^+\mathbf{F}_{k-1}^T + \mathbf{W} \tag{3.25}$$

3. Compute the partial derivative matrix:

$$\mathbf{H}_k = \frac{\partial h(\mathbf{x}_k)}{\mathbf{x}_k} \tag{3.26}$$

4. Perform the measurement update of the state estimate and estimation-error covariance:

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{E})^{-1} \tag{3.27}$$

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k (\mathbf{y}_k - h(\hat{\mathbf{x}}_k^-)) \tag{3.28}$$

$$\mathbf{P}_k^+ = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_k^- (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)^T + \mathbf{K}_k \mathbf{E} \mathbf{K}_k^T \tag{3.29}$$

This approach only uses the accelerometer readings to correct the state estimate, which can only provide information about the inclination of the body, but not about its heading.

The Kalman Filter requires the statistics of the process and the measurement noise to be known in advance. This are the "tuning" parameters and they have to be carefully chosen in order to obtain appropriate results. These parameters are the initial state covariance $\mathbf{P}_0$ (and the initial state $\mathbf{x}_0$), the process noise covariance $\mathbf{W}$ and the measurement noise covariance $\mathbf{E}$.

Initial State Covariance The estimation error-covariance $\mathbf{P}_0$ represents the amount of uncertainty of the initial state chosen, and thus it controls the handover from the initial transient to the state process noise covariance for steady state filter behavior. If $\mathbf{P}_0$ is set equal to zero (for a very confident choice for the initial estimates) then the filter ignores and does not make use of the measurements to correct the prediction. If $\mathbf{P}_0$ is extremely large (a pessimistic choice for the initial values), then the filter weights the measurements much more and provides very little weightage or ignores the state model values leading to large fluctuations in the state and parameter estimates along with large final uncertainty.

System Noise Covariance In principle, $\mathbf{W}$ should reflect the uncertainty in the assumed state model or any unmodelled feature of the state or even unknown random state input. The $\mathbf{W}$ along with the initial $\mathbf{P}_0$ plays a very important role in the filter operation, since choosing proper values for them can reduce significantly the chances of divergence. The value of $\mathbf{W}$ should be small enough to retain the information from the measurement but not large to increase the uncertainty so that the filter estimates become useless. A large value of $\mathbf{W}$ will lead to a short transient with large steady state uncertainty of the estimates and vice versa for small $\mathbf{W}$

Measurement Noise Covariance Usually, a good initial estimate for $\mathbf{E}$ can be obtained from the calibration of the measuring instrument and generally it is assumed to be constant.

The Extended Kalman Filter presented above can be improved by making the measurement noise covariance non-constant. Since the measurement update step makes use of vector observation to correct the system update estimate, it is based on the assumption that the accelerometer is measuring, on average over a small timespan, only gravity to work properly. Therefore, when the object of interest is subject to additional linear acceleration, e.g. perfoming some maneouver, the measurement update may yield wrong results, so a possible way to work around this is to increase the diagonal values of the measurement covariance matrix E_{ii} whenever the norm of the acceleration is different from the magnitude of gravity (with some tolerance)

$$E_{ii}(\boldsymbol{a}) = \begin{cases} 10^3, & \text{for } g_{lb} \leq \|\boldsymbol{a}\|_2 \leq g_{ub} \\ 10^6, & \text{for } \|\boldsymbol{a}\|_2 \leq g_{lb} \text{ or } g_{ub} \leq \|\boldsymbol{a}\|_2 \end{cases}, \text{ for } i = 1, 2, 3 \tag{3.30}$$

where g_{lb} and g_{ub} denote the lower and upper bounds for the magnitude of the gravity. Note that the values chosen for the elements on the diagonal are just an example to show that different values are used depending on the norm of the acceleration. The actual values used will be evaluated in simulations, as well as the bounds mentioned before.

In practice, this approach can be problematic in certain cases because during acceleration the norm of the acceleration could be around 9.8 m/s^2 but in a different direction than gravity due to different accelerations overlapping. Therefore, it would be better to adapt the measurement covariance matrix by keeping track of the norm of the acceleration for some small timespan.

Another possible solution that can be applied to our case, that is, to the orientation estimation problem of the TWIPR, is to vary the measurement covariance matrix according to the magnitude of the pitch angular velocity. Exploiting the dynamic properties of the robot, we know that whenever it needs to attain some translational acceleration, it must also perform some pitch motion for stability purposes (pitch forward to move forward, pitch bacwards to move backwards). This pitch motion also generates rotational accelerations that are measured by the IMU because it is not located along the wheel's axis. So it can be safely assumed that whenever the IMU experiences accelerations other than gravity, it also measures pitch angular velocity. And due to the location of the IMU on the robot, one of the gyroscope's axis will only measure pitch motion, while the other two will handle heading and roll rate. With that said, the proposed method to vary the measurement covariance matrix is similar to the previous one, but with dependence on the mentioned gyroscope measurement

$$E_{ii}(\mathbf{g}) = \begin{cases} 10^3, & \text{for } |\mathrm{g}_y| \leq \mathrm{g}_{\mathrm{th}} \\ 10^6, & \text{for } |\mathrm{g}_y| \geq \mathrm{g}_{\mathrm{th}} \end{cases}, \text{ for } i = 1, 2, 3 \tag{3.31}$$

where g_{th} is a threshold that will be tuned on the simulations. It is also possible to make it a continuous function, where the values of the measurement covariance matrix increasing proportionally to the mentioned gyroscope measurement.

Chapter 4

Position Estimation

Existing techniques for estimating the position of a mobile robot can be divided into two classes: relative tracking techniques and global localizaiton techiques. When using the former, the error tends to grow with time so it cannot be used for long distances. The kinematic model may have some inaccuracies, encoders have limited precision, IMUs are corrupted by noise and there are external sources affecting the motion that are not observable by the sensors (e.g. wheel-slippage, uneven floor).

In case of global localization, integration of noisy data is not required and thus there is no accumulation of error with time or distance traveled. The problem comes with the type of sensor used. For outdoor navigation, GPS is the most widley used, but this suffers from signal blockage, interference and insufficient accuracy for stand-alone navigation systems. In the case of indoor applications, it is possible to get very accurate systems with high sampling rates, but they can be costly and the workplace is limited.

Nonlinear object tracking from noisy measurements is a challenging task of mobile robotics, especially under dynamic environments. This can be tackled by different filtering methods such as Kalman filters and Bayesian filtering. Most object tracking problems are nonlinear systems with non-Gaussian noise, so in this project the particle filter is proposed to handle the position estimation problem. This type of Sequential Monte Carlo (SMC) methods has shown remarkable versatility and enhanced performance with respect to other choices, such as Kalman filter like methods.

4.1 Relative Tracking

In this section, an overview on techniques used in mobile robot relative positioning (also called *dead-reckoning*) is given. These methods make use of on-board sensors and a kinematic model to move from a given state in time to the next one.

In the case of differentially driven robots, like the TWIPR, the kinematic model that links the transition from the pose at a given instant in time, and the pose after a given time-step T_s is given by

$$\begin{aligned} p_x(t+T_s) &= p_x(t) + vT_s \cos\psi(t) \\ p_y(t+T_s) &= p_y(t) + vT_s \sin\psi(t) \\ \psi(t+T_s) &= \psi(t) + T_s\dot{\psi} \end{aligned}$$

where T_s is the sampling time, v represents the translational velocity of the robot and $\dot{\psi}$ is the steering rate, both of which are inputs to the system. Both inputs can be computed from the mentioned on-board sensors, leading to odometry (if encoders are used) and inertial navigation (if IMUs are used).

Odometry Odometry is based on simple equations to compute the translational velocity and heading rate from encoder measuremetns. However, these equations are based on the assumption of pure rolling motion of the wheels, which hold true when the relative velocity between the contact point of the wheel and the floor is zero. Given the angular velocity of the left and right wheels, the velocity and heading rate can be calculated by

$$v(t) = \frac{r_R\omega_R(t) + r_L\omega_L(t)}{2} \qquad \dot{\psi}(t) = \frac{r_R\omega_R(t) - r_L\omega_L(t)}{d} \tag{4.1}$$

where the three major odometry parameters are the radius of its right and left wheels, r_R and r_L respectively and the distance d between its wheels. If some of these parameters are not known accurately or there is some uncertainty, then determining v and $\dot{\psi}$ with these formulas can be erroneous.

Inertial State Estimation Inertial estimation uses gyroscopes and accelerometers to measure rate of rotation and acceleration, which then are integrated once to obtain orientation and velocity, respectively

$$\mathbf{v}_{\text{imu}}(t) = \mathbf{v}(t-1) + \boldsymbol{a}_T(t)T_s \qquad \dot{\boldsymbol{\psi}}(t) = \mathbf{g}(t)T_s \tag{4.2}$$

where $\boldsymbol{a}_T$ denotes the translational part of the acceleration, $\mathbf{g}$ is the gyroscope measurement and T_s is the sampling time. Inertial navigation systems have the advantage that they are self-contained. On the downside, inertial sensor data drifts with time because of the need to integrate rate data to yield position; any small constant error increases without bound after integration. Inertial sensors are thus unsuitable for accurate positioning over an extended period of time.The errors affecting the IMUs measurements were already described in Section 2.3.

4.2 Absolute Localization

There is no universal classification of absolute localization methods, but the most common ones used in robotic applications are active beacons, GPS, landmark recognition and motion capture systems. A detailed description of different indoor localization techniques can be found in [13].

Active Beacons This method computes the absolute position of a body from measuring the direction of incidence of three or more actively transmitted beacons. this approach allows high sampling rates and yields high reliability, but it does also incur high costs in installation and maintenance. The transmitters, usually using light or radio frequencies, must be located at known sites in the environment, and the accuracy of the positioning estimate depends on the accuracy on the beacons location.

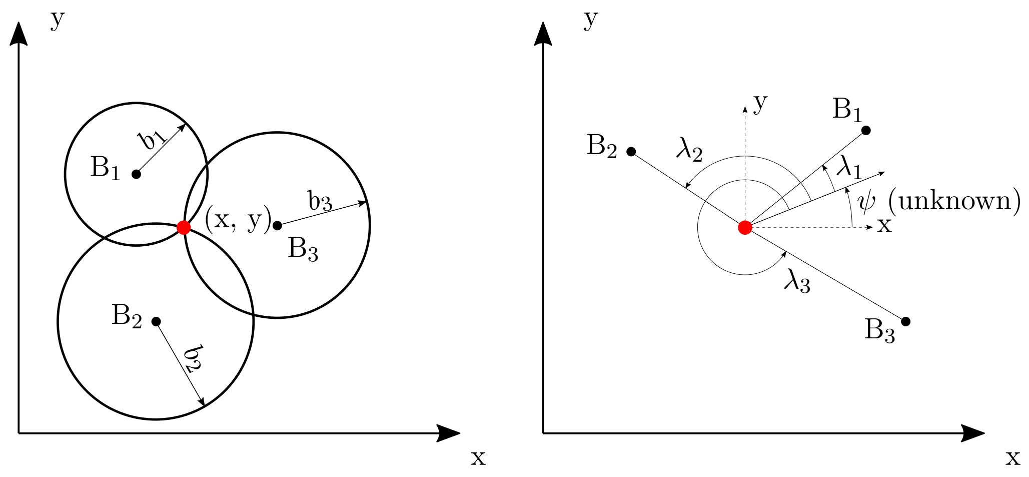


Figure 4.1: Trilateration problem (left) vs triangulation problem (right).

One can distinguish between two different types of active beacon systems: trilateration and triangulation. The former one involves the determination of a vehicle's position based on distance measurements to known beacon sources; while in the latter a rotating sensor on board the vehicle registers the angles at which it "sees" the transmitter beacons relative to the vehicle's longitudinal axis. From these three measurements the unknown coordinates and the unknown vehicle orientation can be computed.

Global Positioning Systems Satellite Navigation is based on a global network of satellites that transmit radio signals in medium earth orbit. These satellites emit signals to receivers that, using trilateration methods, can compute their position by measuring the travel time of the satellites signals. The GPS receiver calculates its own position and time based on data received from multiple GPS satellites. Each satellite carries an accurate record of its position and time, and transmits that data to the receiver.

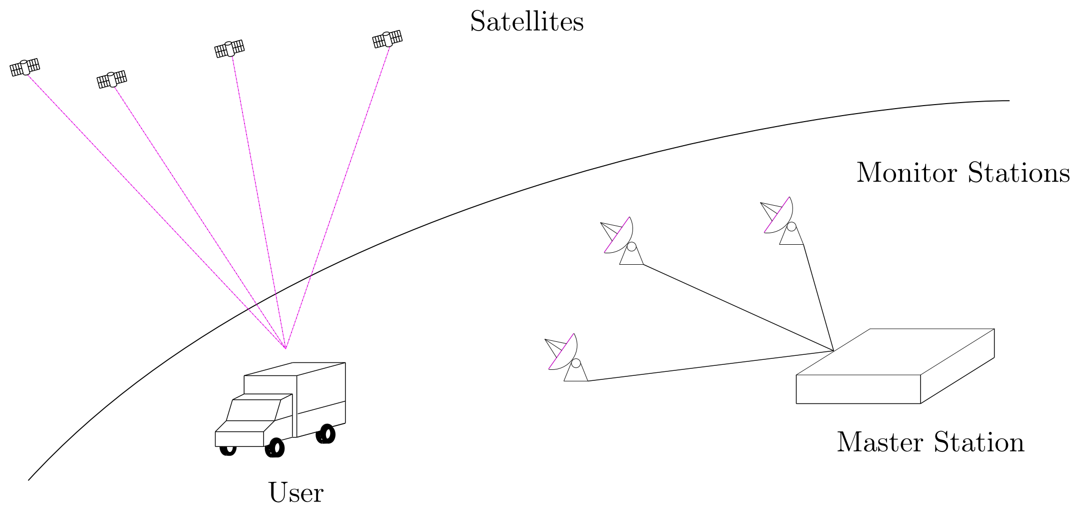


Figure 4.2: A Global Positioning System and its fundamental agents [14].

Although conceptually very simple, this methodology introduces at some technical challenges like time synchronization between individual satellites and GPS receivers; precise real-time location of satellite position; and accurate measurement of signal propagation time. The first of these problems is addressed through the use of atomic clocks. The precise real-time location of satellite position is determined by a number of widely distributed tracking and telemetry stations at surveyed locations around the world.

GPS accuracy in commercial use is typically between 15 m and 40 m. This varies depending on surroundings, devices used, weather and many other factors. The problems associated with using GPS for mobile robot navigation include blockage of signals due to path interference, insufficient position accuacy to be used alone.

Landmark Recognition In this method distinctive landmarks that a robot can recognize from its sensory input, are placed at known locations in the environment. The main task in localization is then to recognize the landmarks reliably and to calculate the robot's position. This means that more processing is necessary than with active beacon systems. In many cases onboard computers cannot process landmark algorithms quickly enough for real-time motion. Positioning accuracy depends on the geometry of the robot and the landmarks but is typically within a few centimeters.

Landmarks can be *natural* if they are objects or features that are already in the environment, or *artificial* if they are specially designed to be placed in the environment with the sole purpose of enabling robot navigation. The most obvious sensor to be used for landmark recognition is computer vision, but range sensors can also be used to detect distinct signatures, such as those of a corner or an edge, or of long straight walls.

The basic components of a landmark positioning system include a sensor to detect the landmarks, a method to match the features observed with the known landmarks, and a method to compute the localization from the matches.

Model Matching In this method information acquired from the robot's onboard sensors is used to create a map of its local environment. Then, it is compared to a map or world model of the environment. If features from the sensor-based map and the world model map match, then the vehicle's absolute location can be estimated. Usually, model matching is used in combination with other relative and absolute tracking methods, where GPS and dead-reckoning are the most commonly used ones for map relative localization of a vehicle.

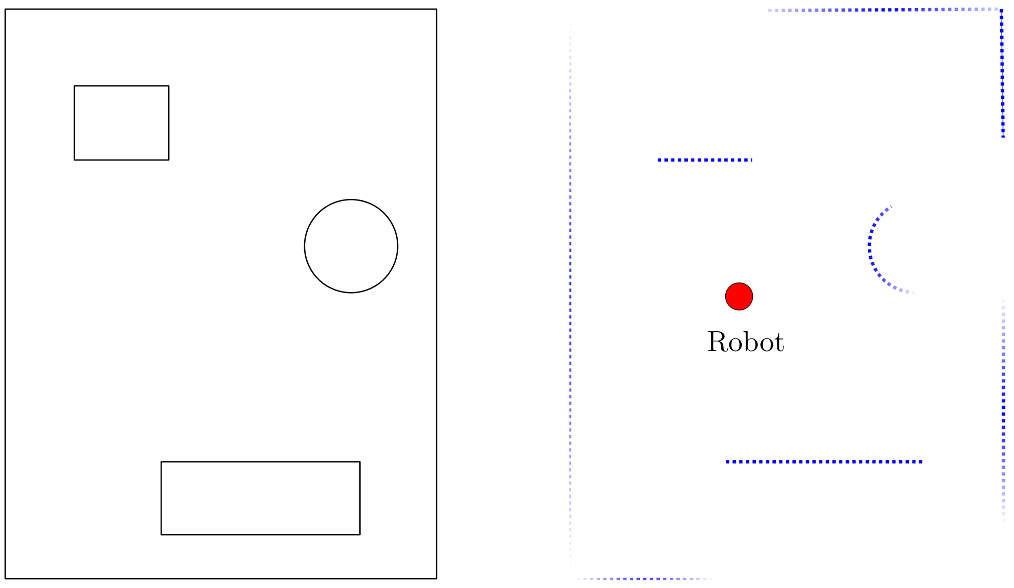


Figure 4.3: Known map of the environment (left) and sensed objects by the robot (right).

Model matching positioning methods can be used to generate an updated map of the environment, and so allows a robot to learn it and react accordingly. The problem with these methods is that they require a large processing power, and so their application is usually limited to relatively simple environments.

Optical Systems This kind of systems utilize data captured from image sensors to triangulate the position of an object, between two or more cameras calibrated to provide overlapping projections. Special markers (passive or active) are usually attached to the object to make it easier for the sensors to locate the object. Passive markers are typically rubber balls covered with a retroreflective material to reflect light that is generated near the cameras lens; while active markers are powered to emit their own light, instead of reflecting the one generated by the cameras.

Markerless configurations are also possible thanks to the development of computer vision techniques. High resolution images of the target being tracked can provide more information than just motion data, but they require more computational power to do so.

The motion capture system described in Section 2.2.3 is an optical system.

4.3 Nonlinear Filtering

Nonlinear filtering can be a difficult and complex subject. It is certainly not as well understood as linear filtering. There is still a lot of room for advances and improvement in nonlinear estimation techniques. However, some nonlinear estimation methods have become widespread. These techniques include nonlinear extensions of the Kalman filter, unscented filtering, and particle filtering.

4.3.1 State Space Model

The most common way of describing a dynamic system is by its state space-representation model, which relates a state $\mathbf{x}_k \in \mathbb{R}^3$ to the observations $\mathbf{y}_k \in \mathbb{R}^2$:

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k) \tag{4.3}$$

$$\mathbf{y}_k = h(\mathbf{x}_k, \mathbf{e}_k) \tag{4.4}$$

The first equation is known as the state equation, where k denotes the time index, $f(\cdot)$ is a known, possibly nonlinear function, $\mathbf{u}_k \in \mathbb{R}^2$ is the input of the system and $\mathbf{w}_k$ is a stochastic process noise. The second equation is referred to as the output equation, where $h(\cdot)$ is a known, possibly nonlinear function, and $\mathbf{e}_k$ is a stochastic measurement noise. Both process and measurement noise are assumed with known pdf p and mutually independent.

$$\mathbf{w}_k \sim p_{wk} \qquad\qquad \mathbf{e}_k \sim p_{ek}$$

State Equation

The TWIPR is a non-holonomic differential drive mobile robot. The robot origin is centered between the two differentially driven wheels. The state of interest for the position estimation algorithm is:

$$\mathbf{x} = \begin{bmatrix} p_x & p_y & \Psi \end{bmatrix}^T \tag{4.5}$$

Where p_x, p_y refer to the origin coordinates, and Ψ refers to the heading. The following system model describes the transition for each state:

$$p_{x,k} = p_{x,k-1} + vT_s \cos \Psi_{k-1} \tag{4.6}$$

$$p_{y,k} = p_{y,k-1} + vT_s \sin \Psi_{k-1} \tag{4.7}$$

$$\Psi_k = \Psi_{k-1} + \dot{\Psi} \tag{4.8}$$

where $v \in \mathbb{R}$ and $\dot{\Psi} \in \mathbb{R}$ are inputs to the system.

To estimate the velocity of the TWIPR, odometry and inertial data will be fused using a complementary filter as

$$v(t) = \alpha v_{\text{imu},x}(t) + (1-\alpha) v_{\text{enc}}(t) \tag{4.9}$$

where v_{enc} denotes the velocity computed by odometry and $v_{\text{imu},x}$ denotes the x component of the velocity obtained from inertial measurements as defined in equations (4.1) and (4.2), respectively.

Just as done with the velocity, the heading rate will also be estimated using the same complementary filter

$$\dot{\Psi}(t) = \alpha \dot{\Psi}_{\text{imu},z}(t) + (1-\alpha)\dot{\Psi}_{\text{enc}}(t) \tag{4.10}$$

where Ψ_{enc} denotes the heading computed from odometry and $\Psi_{\text{imu},z}$ denotes the z component of the heading rate obtained from inertial measurements, using equations (4.1) and (4.2), respectively.

Update Equation

Each sensor used on the robot has a corresponding measurement model $h(x)$ which describes the measurement's sensitivity to the robot state. Measurement updates will be applied sequentially Because the TWIPR uses multiple asynchronous sensors, so we want to update the state as soon as new data arrives.

Motion Capture Measurements

The OptiTrack system gives an absolute measurement of the position of the robot with high accuracy. These measurements can be compared directly with the state of the robot, thus, the measurement update equations are

$$\begin{bmatrix} p_x \\ p_y \end{bmatrix}_{OptiTrack} = \begin{bmatrix} p_x \\ p_y \end{bmatrix}_{state}$$

But we are interested in computing the relative likelihood q_i of each particle conditioned on the measurements. To this end, we will evaluate the distance d between the particles and the position given by these measurements and assign weights to each particle according to it.

$$d = \left\| \begin{bmatrix} p_x \\ p_y \end{bmatrix}_{OptiTrack} - \begin{bmatrix} p_x \\ p_y \end{bmatrix}_{state} \right\|$$

If the distance is small, a big weight will be assigned to the particle, but if it is large it will get a small weight.

4.3.2 Bayesian Filtering

The Bayesian approach to nonlinear filtering is to compute or approximate the conditional pdf of the state given the observations, which is given by the general Bayesian update recursion:

$$p(\mathbf{x}_{k+1}|\mathbf{y}_{1:k}) = \int p(\mathbf{x}_{k+1}|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{y}_{1:k})d\mathbf{x}_k \tag{4.11}$$

$$p(\mathbf{x}_k|\mathbf{y}_{1:k}) = \frac{p(\mathbf{y}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{y}_{1:k-1})}{p(\mathbf{y}_k|\mathbf{y}_{1:k-1})} \tag{4.12}$$

These two equations correspond to a time update and a measurement update equation, respectively. The primary output of this filtering algorithm is the posterior distribution, from which statistical measures can be extracted. Analytical solutions to these equations are available only for a few special cases. In particular, if $f(\cdot)$ and $h(\cdot)$ are linear, and $\mathbf{x}_0$, $\mathbf{w}_k$, and $\mathbf{e}_k$ are additive, independent, and Gaussian, then the solution is the Kalman filter. For nonlinear or non-Gaussian models there is in general no finite- dimensional representation of the posterior distributions. That is why numerical approximations are needed.

4.3.3 The Particle Filter

The particle filter estimates a posterior probability density over the state space conditioned on the data collected so far. It is a way to numerically implement the Bayesian estimator which uses a set of particles S_k with associated weight $q^i \in \mathbb{R}$ to represent or approximate the posterior distribution.

$$S = \left\{ \mathbf{x}_k^i, q_k^i \right\}_{i=1}^{N_s} \tag{4.13}$$

The aim is to evaluate the posterior probability function $p(\mathbf{x}_k|\mathbf{y}_{1:k})$ and then obtain an estimation of $\mathbf{x}_k$. Usually, this process involves the following steps:

1. Perform the time propagation using the known system dynamics $f(\mathbf{x}_{k-1}, \mathbf{u}_k)$ to propagate each particle from $\mathbf{x}_{k-1}^n$ to $\mathbf{x}_k^n$. In addition, random noise is stochastically added to the states when calculating the updated state estimates $\mathbf{x}_k^n$. This random sampling represents the system noise and increases the overall distribution of the particles.

2. Compute the relative likelihood q_i of each particle $\mathbf{x}_k^n$ conditioned on the measurement $\mathbf{y}_k$. This is done by evaluating the pdf $p(\mathbf{y}_k|\mathbf{x}_k^n)$ on the basis of the nonlinear measurement equation and the pdf of the measurement noise.

3. Scale the relative likelihoods obtained in the previous step so that the sum of all the likelihoods is equal to one.

4. This is called the resampling step, and it will be discussed further because it is a critical step but, as we will see, it is not necessary to perform it every time.

5. Compute any desired statistical measure of the resulting pdf $p(\mathbf{x}_k|\mathbf{y}_k)$. Generally, we are interested in the mean and the variance.

Particle Propagation

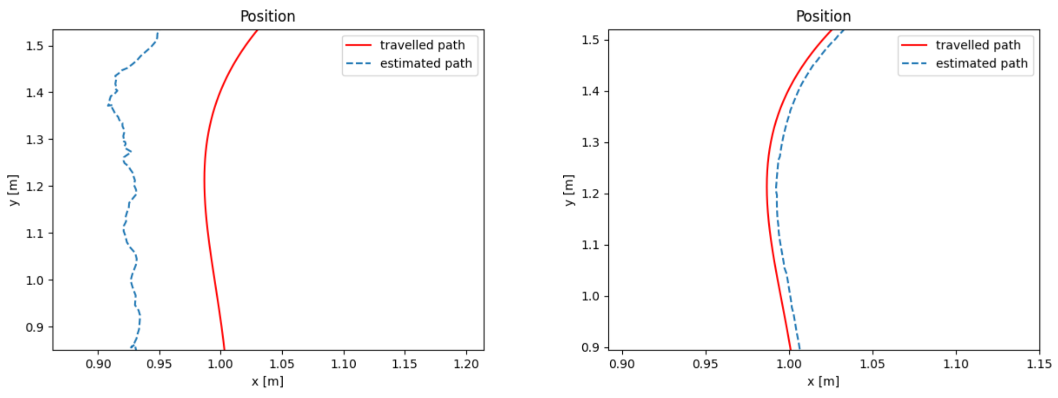
As stated in the first step of the particle filter algorithm, noise should be added to the states to improve the overall distribution of the particles. However, adding random noise to each state independently will not yield the desired result. Given that the particles represent a possible state of the robot, they should be "realistic". What is meant by this is that the particle propagation in space must be consistent with its own velocity and heading. The system update equations perfectly describes the motion of the robot, the only sources of uncertainty in it come from the velocity and the heading, so noise should be added to represent this uncertainties.

If we just add random noise to each state, the estimate we get is like the one on Figure 4.4a. As we can see, the estimated path is not smooth and thus, not very accurate.

With this taken into account, the idea is to add random noise to the actual sources of uncertainty in our model, which are the velocity and the heading, and then propagate

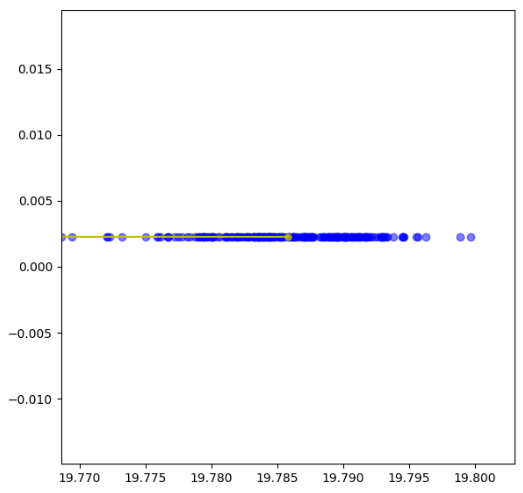
Algorithm 4.3.1 Standard Particle Filter Algorithm

```
1. Initialization
    Draw $N_s$ particles $\mathbf{x}_0^i \sim p_{x_0}$, $i = 1 : N_s$
    Set all weights $q_i = 1/N_s$, $i = 1 : N_s$
    Time index $t = 1$
2. System Update
    Evolve particles based on the system model
    $\mathbf{x}_t^i = f(\mathbf{x}_{t-1}^i, \mathbf{u}_t)$, $i = 1 : N_s$
3. Measurement Update
    Obtain measurements $\mathbf{y}_k$ and update the weights by the likelihood:
    $q_i = p(\mathbf{y}_k|\mathbf{x}_k^i)$, $i = 1 : N_s$
    Normalize weights $q_i = \frac{q_i}{\sum_{j=1}^{N_s}(q_j)}$
4. Resample
    Compute effective number of Samples $N_{eff} = \frac{1}{\sum_{i=1}^{N_s}(q_i)^2}$
    if $N_{eff} < N_{th}$ then
        Resample particles $[\{\widetilde{\mathbf{x}}_t^n\}_{n=1}^{N_s}] = Resample[\{\mathbf{x}_t^i, q_t^i\}_{i=1}^{M}, N_s]$
    end if
5. Loop
    Increment time index $t = t + 1$
    Go to Step 2
```



(a) Random noise added to the position (b) Noise added to the velocity and the heading

Figure 4.4: Position estimation with added random noise


(a) Noise added to the velocity

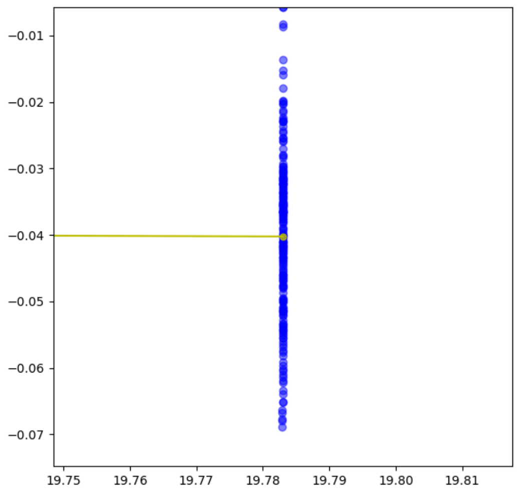

(b) Noise added to the heading

Figure 4.5: System update with additive noise

the particles.

$$
\begin{aligned}
p_{x,k} &= p_{x,k-1} + (v + \mathbf{w_{1k}})T_s \cos \Psi_{k-1} \\
p_{y,k} &= p_{y,k-1} + (v + \mathbf{w_{1k}})T_s \sin \Psi_{k-1} \\
\Psi_k &= \Psi_{k-1} + \dot{\Psi} T_s + \mathbf{w_{2k}}
\end{aligned}
\tag{4.14}
$$

The uncertainty in the velocity generates a particle distribution along the direction of movement, while the uncertainty in the heading generates that distribution along the direction normal to the previous one, like shown in Figure 4.5. The combination of these two results in an ellipse-shaped set of particles (Figure 4.6).

The shape of the ellipse will depend on the amount of uncertainty on each variable. The uncertainty on the velocity will depend on mostly on two factors: the velocity of the robot and the presence of slipping. The first one means that while the robot is driving at low speed, the encoder measurements are more reliable and accurate, but when the robot increases its speed, the accuracy reduces a little. The second one has to do with the fact that when slipping is detected, the velocity used to propagate the particles is obtained from the accelerometer, and this is not so accurate due to all the disturbances, thus there is an increased uncertainty. Besides these two factors, time also affects the distribution. The longer we propagate the particles without any measurement to compute their conditional probability and perform resampling, the greater the uncertainty will get.

The uncertainty on the heading will mostly depend on time and the possibility to update the heading of the robot with some given measurement

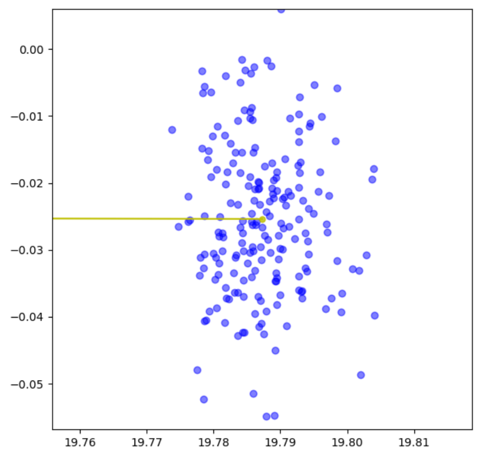


Figure 4.6: System update with additive noise

Relative Likelihood

In this step, we use the information of the available sensors to determine how likely is each propagated particle to be the unknown real state that we are aiming to estimate. This is important because it helps us (together with the resampling step) to remove the particles that are less likely and concentrate them into the more likely areas.

Since we know the measurement equation, the simplest way of doing this is to compute the relative likelihood q_i that the measurement is equal to a specific measurement $\mathbf{y}^*$, given the premise that $\mathbf{x}_k$ is equal to the particle $\mathbf{x}_k^i$:

$$q_i = P[(\mathbf{y}_k = \mathbf{y}^*)|(\mathbf{x}_k = \mathbf{x}_k^i)] \tag{4.15}$$

$$q_i = P[\mathbf{e}_k = \mathbf{y}^* - h(\mathbf{x}_k^i)] \tag{4.16}$$

Where $\mathbf{e}_k$ is the measurement noise, which is assumed to be a zero-mean random gaussian noise with standard deviation that can be approximated. First of all, given that we are aiming to estimate the robots position in the environment, the most useful measurement to do so is the one coming from the motion capture system, since it provides absolute position measurements.

Each particle has a position and a weight which estimates how well it matches the measurement. Normalizing the weights so they sum to one turns them into a probability distribution. The particles that are closest to the robot will generally have a higher

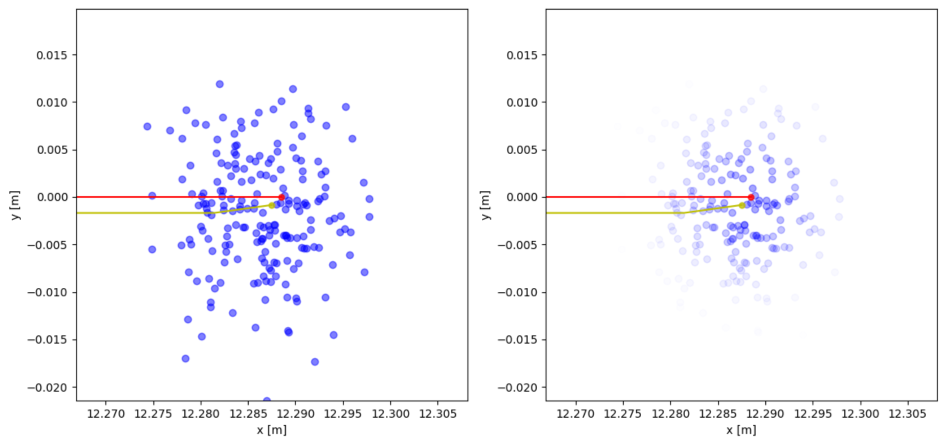


Figure 4.7: Relative likelihood of each particle

weight than the ones far from the robot. The weight of the particle is computed as the probability that it matches the Gaussian of the sensor error model. The further the particle from the measured distance the less likely it is to be a good representation, like In Figure 4.7.

Another useful measurement to update the particles comes from the distance sensors, like the ones located at each side of the robot. Consider for example the case when the robot is driving on a straight line. At some moment, the distance sensor located at it´s left side detects an object nearby. If we know the location of that object, we can use this information to update the particles and generate knew weights that take into account this information (see Figure 4.6).

Resampling

The resampling step is an important part of the filter that is used to deal with the problem of degeneracy of the algorithm, that is, to avoid all relative weights from tending to zero except for one that tends to one. A relative weight of one is not an indicator of how close a state is to the true one since this is only a relative weight. It merely says that one sequence in the whole particles set is much more likely than all of the other ones. Resampling solves this problem but creates another because the random sampling increases uncertainty and is computationally expensive. It is therefore of interest to start the resampling process only when it is really needed. An indicator of the degree of degeneracy is the effective number of samples, defined in terms of the coefficient of the variance of the weights as

$$N_{eff} = \frac{N_s}{1 + N_s^2 Var(q_i)} \tag{4.17}$$

The effective number of particles is thus equal to the total number of particles when all weights are equal, and it is equal to 1 when $q_i = 1$ with probability $1/N_s$ and $q_i = 0$ with probability $(N_s - 1)/N_s$. A computable approximation of the previous equation is given by

$$N_{eff} = \frac{1}{\sum_{i=1}^{N_s} (q_i)^2} \tag{4.18}$$

If the effective number of particles is less than a given threshold $N_{eff} < N_{th}$, it means that resampling should take place. We will choose this threshold as a percentage of the total number of particles used.

Another trick that can be used to prevent the problem of degeneracy (and thus, avoid resampling too often), is called "roughening", which is based on increasing the random noise to each particle in the time propagation step, which is similar to adding artificial noise to the Kalman filter.

The computational effort of the particle filter is often a bottleneck to its implementation. The estimation error in a particle filter converges to zero as the number of particles approaches infinity, but also does the computational effort, so a tradeoff between them has to be selected in order to achieve good performance. Consequently, resampling has been extensively researched, and, as a result, various resampling schemes have been proposed in [15]–[18]. A detailed classification of resampling algorithms with qualitative comparison of them can be found in [19].

The most commonly used methods are described below and pseudocodes for selected algorithms are provided. The pseudocodes are presented in a simple and unified way but not in forms that optimize the implementation of the algorithms.

Multinomial Resampling The idea behind this algorithm is simple. First, compute the cumulative sum of the normalized weights. Then, generate random numbers from the uniform distribution on $(0, 1]$ and use binary search to find its position inside the cumulative sum array. Since the sampling of each particle is random, the upper and lower limits of the number of times a given particle is resampled are zero (not sampled) and N_s(sampled N_s times), respectively. This yields the maximum variance of the resampled particles.

Residual Resampling Residual resampling consists of two stages. In the first one, the normalized weights are multiplied by the number of samples, and then the integer value of each weight (N_t^i) is used to define how many samples of that particle will be taken. This ensures that all higher weight particles are chosen at least once. The total number of replicated particles in this stage is $N_t = \sum_{i=1}^{N_s} N_s q_i$. Since this number may not be equal to the number of samples, to select the remaining samples we use the residual of the weights:

$$r_t^i = q_t^i - \frac{N_t^i}{N_s}$$

Then the particles are drawn according to the residual weights and by using multinomial resampling (or another random sampling method).

Stratified Resampling This scheme aims to make selections relatively uniformly across the particles. It works by dividing the cumulative sum into N_s equal sections, and then selects one particle randomly from each section. This guarantees that each sample is between 0 and $2/N_s$ apart. The upper and lower limits of the times the i-th particle is resampled are $\mathbf{max}(\mathbf{Floor}[N_s q_i]-1, 0)$ and $\mathbf{Floor}[N_s q_i]+2$ respectively; where $\mathbf{Floor}[\cdot]$ returns the integer part of the number passed to the function.

Systematic Resampling As with stratified resampling the space is divided into N_s divisions. We then choose a random offset to use for all of the divisions, ensuring that each sample is exactly $1/N_s$ apart. The upper and lower limits of the times the i-th particle is resampled are $\mathbf{Floor}[N_s q_i]$ and $\mathbf{Floor}[N_s q_i] + 1$.

In order to compare all these methods, lets consider a simple example. Assuming a number of samples $N_s = 7$ and their corresponding weights beeing $(0.1, 0.2, 0.3, 0.4, 0.2, 0.3, 0.1)$. If we perform resampling to this set using the four previous methods, we would obtain something like Figure 4.8.

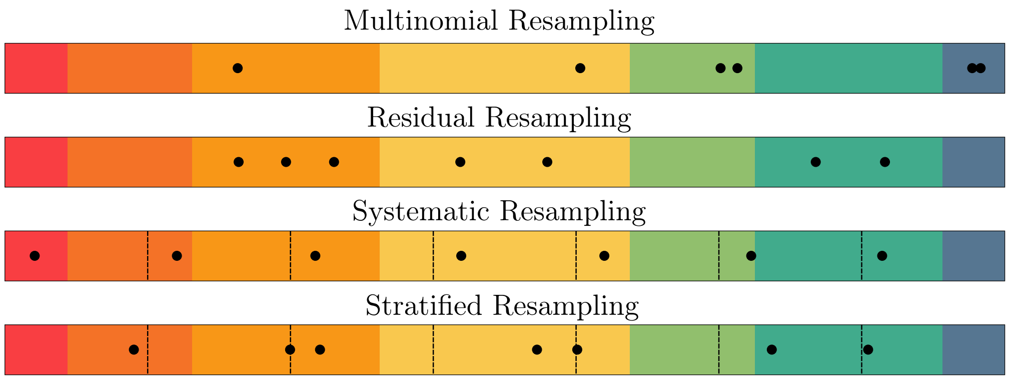


Figure 4.8: Comparison of different resampling algorithms.

The performance of the multinomial resampling is quite bad. There is a very large weight that was not sampled at all. The largest weight only got one resample, yet the smallest weight was sample was sampled twice.

The residual resampling algorithm does excellently at what it tries to do: ensure all the largest weights are resampled multiple times. Nevertheless, It does not distribute evenly the samples across the particles and many reasonably large weights are not resampled at all.

Systematic sampling does an excellent job of ensuring we sample from all parts of the particle space while ensuring larger weights are proportionality resampled more often. Stratified resampling is not quite as uniform as systematic resampling, but it is a bit better at ensuring the higher weights get resampled more.

Algorithm 4.3.2 Multinomial Resampling

```
$[\{\widetilde{x}_t^n\}_{n=1}^{N_s}] = Multinomial[\{x_t^i, q_t^i\}_{i=1}^{M}, N_s]$

$Q_t^i = CumulativeSum(\{q_t^i\}_{i=1}^{N_s})$
$n = 0$
while $n <= N_s$ do
    $u \sim U(0, N]$
    $m = 1$
    while $Q_t^i <= u$ do
        $m = m + 1$
    end while
    $n = n + 1$
    $\widetilde{x}_t^n = x_t^m$
end while
```

Algorithm 4.3.3 Systematic Resampling

```
$[\{\widetilde{x}_t^n\}_{n=1}^{N_s}] = Systematic[\{x_t^i, q_t^i\}_{i=1}^{M}, N_s]$

$Q_t^i = CumulativeSum(\{q_t^i\}_{i=1}^{N_s})$
$n = 0$
$m = 1$
$u_0 \sim U(0, 1/N]$
while $n <= N_s$ do
    $u = u_0 + n/N$
    while $Q_t^i <= u$ do
        $m = m + 1$
    end while
    $n = n + 1$
    $\widetilde{x}_t^n = x_t^m$
end while
```

Algorithm 4.3.4 Stratified Resampling

```
$[\{\widetilde{x}_t^n\}_{n=1}^{N_s}] = Stratified[\{x_t^i, q_t^i\}_{i=1}^{M}, N_s]$

$Q_t^i = CumulativeSum(\{q_t^i\}_{i=1}^{N_s})$
$n = 0$
$m = 1$
while $n <= N_s$ do
    $u_0 \sim U(0, 1/N]$
    $u = u_0 + n/N$
    while $Q_t^i <= u$ do
        $m = m + 1$
    end while
    $n = n + 1$
    $\widetilde{x}_t^n = x_t^m$
end while
```

Algorithm 4.3.5 Residual Resampling

```
$[\{\widetilde{x}_t^n\}_{n=1}^{N_s}] = Residual[\{x_t^i, q_t^i\}_{i=1}^{M}, N_s]$

$j = 0$
for $i = 1 : N_s$ do
    $N_t^i = Floor(N_s \times q_t^i)$
    $r_t^i = q_t^i - N_t^i/N_s$
    for $k = 1 : N_t^i$ do
        $j = j + 1$
        $\widetilde{x}_t^j = x_t^i$
    end for
end for
$N_r = N - j$
for $i = 1 : N_s$ do
    $r_t^i = r_t^i \times N_s/(N_s - N_r)$
end for
$[\{\widetilde{x}_t^n\}_{n=N_r+1}^{N_s}] = Multinomial[\{x_t^i, r_t^i\}_{i=1}^{N_s}, N_s - N_r]$
```

4.4 Slip Detection

Wheel encoders provide a precise and inexpensive measurement of wheel rotation. When used on a mobile robot, encoders measure rotational distance which is directly proportional to each wheel's angular velocity and (assuming no odometry errors or wheel slip) translational velocity.

Odometry errors are referred to in literature as either systematic or non-systematic errors. A systematic error typically results from incorrectly calibrated odometry parameters: the encoder ticks-per-meter conversion, wheel diameter, or the distance between the wheels. A non-systematic error, on the other hand, represents error from the robot environment: wheel slip, wheel skid, external forces, etc.

It is important to avoid feeding the filter with incorrect information, otherwise the estimated state will not be accurate. To this end, we need to derive a way to detect non-systematic errors, and thus neglect the encoder readings when they are not reliable. In particular, we are interested in detecting slipping and skidding as they are the main source of non-systematic errors in our application.

There are two possible situations to be identified: slipping (or skidding) occurring in just one of the two wheels, or in both wheels at the same time. One way to identify slipping is to compare the encoder readings with those of the gyroscope. In the case of the first scenario, de difference between both encoder readings is related with the heading angular velocity of the robot by

$$\dot{\psi} = \frac{r(\omega_R - \omega_L)}{d}$$

By transforming the gyroscope measurement from the sensor frame to the navigation frame, we can then compare the z-axis measurement with the information coming from the encoders. If no slippage occurred, then one can expect good correspondence between the rate-of-turn derived from the encoders and the gyroscope. Poor correspondence suggests wheel slippage.

Unfortunately, this method is not useful when both wheels are slipping, especially when they are spinning at almost the same speed. This second scenario can be dealt with by comparing the linear velocity of the robot computed by updating the robots' velocity with inertial measurements, with the one obtained from the encoder measurements, which is given by

$$v = \frac{r(\omega_R + \omega_L)}{2}$$

When both wheels are slipping, the velocity computed from the encoder measurements will be much greater than the one obtained from the system update using the accelerometer measurements.

In both cases, a threshold must be selected to specify when slipping is happening or not.

Chapter 5

Simulation-based Evaluation

5.1 Introduction

In this chapter, the methods developed in the previous sections will be tested in simulations. The parameters of these methods will be studied in order to come up with the best combination for real applications. The simulations will also allow to study the influence of the disturbances on the accuracy of the estimation.

5.2 Error Metrics

In order to verify the methods and to quantify the accuracy, error metrics have to be defined which quantify the performance of the methods.

The method used to calculate point distance largely determines the overall properties of the performance metric [20]. Most commonly used methods are based on subtraction and include: the *error* E, the *absolute error* AE, and *squared error* SE:

$$E(t) = A(t) - P(t) \tag{5.1}$$

$$AE(t) = |A(t) - P(t)| \tag{5.2}$$

$$SE(t) = (A(t) - P(t))^2 \tag{5.3}$$

with A and P denoting the actual and estimated values respectively.

The standard error can be used to determine whether the estimation method tends to overestimate or underestimate actual values, i.e. biased. The problem with this is that the positive and negative errors will be cancelling each other, meaning that the result of calculating the performance metric may yield zero demonstrating a falsely high accuracy. For this reason, the absolute error will be also used to provide a measure of average error.

Finally, squared error has the property penalizing extreme errors or being susceptible to outliers.

Since we do not have a way to correct the heading, the complete orientation will be divided into heading Ψ and inclination Θ. For each one of them, the mean error ME and mean absolute error MAE defined before are

$$ME_{\Theta} = \frac{1}{N}\sum_{t=1}^{N}\Theta(t) - \hat{\Theta}(t) \qquad ME_{\Psi} = \frac{1}{N}\sum_{t=1}^{N}\Psi(t) - \hat{\Psi}(t) \tag{5.4}$$

$$MAE_{\Theta} = \frac{1}{N}\sum_{t=1}^{N}|\Theta(t) - \hat{\Theta}(t)| \qquad MAE_{\Psi} = \frac{1}{N}\sum_{t=1}^{N}|\Psi(t) - \hat{\Psi}(t)| \tag{5.5}$$

Regarding the position estimation problem, given that the robot is driving in a plane with two coordinates, measuring the error in each of them individually does not provide a good estimate on how far the estimated position is from the real one. Thus, both of them will be combined to give the distance between the two positions:

$$E_p(t) = \sqrt{(x(t) - \hat{x}(t))^2 + (y(t) - \hat{y}(t))^2} \tag{5.6}$$

and just as done with the orientation, the mean value of the position error during the time of ineterest will be computed

$$ME_p = \frac{1}{N}\sum_{t=1}^{N}E_p(t) \tag{5.7}$$

To evaluate the uncertainty of the estimate, i.e. the particles distribution, a standar deviation will be calculated. While the standard deviation of the x and y coordinates provides some information about the dispersion of the particles, there is a problem with it: it does not provide a single summary statistic of the dispersion, is actually two separate statistics. A measure which overcomes this problem is the *standard distance deviation* d, which is the standard deviation of the distance of each particle from the weighted estimate. First, the distance is computed as

$$d^i(t) = \sqrt{(x^i(t) - \hat{x}(t))^2 + (y^i(t) - \hat{y}(t))^2} \qquad \text{for } i = 1, ..., N_s \tag{5.8}$$

Then, the standar deviation σ_p is computed as follows

$$\sigma_p(t) = \sqrt{\frac{1}{N_s - 2}\sum_{i=1}^{N_s}(d^i(t))^2} \tag{5.9}$$

Note that 2 is subtracted from the number of points to produce an unbiased estimate since there are two constants from which the distance along each axis is measured ($\hat{x}$ and $\hat{y}$).

Also a standard deviation of the particles heading σ_Ψ is be defined as

$$\sigma_\Psi(t) = \sqrt{\frac{1}{N_s - 1}\sum_{i=1}^{N_s}(\Psi^i(t) - \hat{\Psi})^2} \tag{5.10}$$

5.3 Simulation Setup

The purpose of this section is to give an overview of the simulation environment, its limitations and how real-application disturbances are taken into account.

The robotic motion is simulated in Python using the dynamic equations and control scheme from Chapter 2. As a remainder, this model was derived considering that both wheel-radius are equal, the distance between the wheels, i.e. the distance between the wheels contact points on the ground is perfectly known; both wheels are perfectly aligned along one common axis and that the robot is unnable to experience roll rotations or wheel-slip. It is also assumed that both motors actuating each wheel are exactly equal, and thus it is possible to make each wheel rotate at the same speed and provide the same torque.

The simulated robot takes linear and steering velocity commands as inputs, and returns the state defined in Section 2.1.2 and its time derivatives

$$\mathbf{x} = \begin{bmatrix} s & \dot{s} & \theta & \dot{\theta} & \psi & \dot{\psi} \end{bmatrix}^T \tag{5.11}$$

This is then used to compute the sensors measurements using the models described in Appendix A. The parameters of the error characteristics used to simulate the sensors are the ones listed on Table 2.2 for the MPU9250. Finally, these sensorial data is feeded to the different estimation methods described in Chapter 3 and Chapter 4 to track the robots position and orientation.

5.4 Methods for Orientation Estimation

In chapter Chapter 3 the Extended Kalman Filter was presented to be used in our application, and some information about the initial state and covariance, system noise and measurement noise covariances was given. But before we start giving values to these parameters, first a closer look to the problem at hand is given.

The set of orientations that the TWIPR can experience include pitch angles from $-90°$ to $90°$, and the full $360°$ for heading. In normal conditions, we consider that the robot is unable to experience roll rotations because that would mean that one of the wheels looses contact with the ground and the whole stability of the robot is compromised. Furthermore, the dynamic equations used to build the simulation environment do not consider this posibility, and thus we can safely assume that in every simulation, the roll angle will be $\phi = 0°$.

5.4.1 Initial Parameters

A common practice is to use the identiy quaternion as initial orientation, and let the estimate converge to its true value, which takes just a few seconds. But in our application, we know the actual inclination of the robot when it is initialized, because it only has two stable orientations when powered down: laying down or standing perfectly straight. Therefore, to define its initial orientations the first accelerometer measurement (after proper calibration) will be used to compute the robot inclination. Regarding the heading, since there is no absolute measurement to obtain it from, it will just be assumed to be zero or its true value will be set using the motion capture system.

Once the initial orientation is known, the initial state covariance is given low values, denoting that we are certain about the initial state

$$\mathbf{P}_0 = \begin{bmatrix} 10^{-2} & 0 & 0 & 0 \\ 0 & 10^{-2} & 0 & 0 \\ 0 & 0 & 10^{-2} & 0 \\ 0 & 0 & 0 & 10^{-2} \end{bmatrix} \tag{5.12}$$

For the system and measurement noise covariances, we know that the accelerometer measurements are more noisy than the gyroscope measurements. Furthermore, during motion, the accelerometer does not provide accurate information on the attitude of the sensor.

Using the simulations it is possible to test how the estimate is affected by changing the values in the covariance matrices. Figure 5.1 shows the pitch angle estimation with different values for the measurement covariance, where the robot was given only a constant forward velocity command. Pitch motion is mainly experienced when the TWIPR changes its velocity. The purpose of giving it a constant velocity command is to check the algorithm performance in presence of rapid movements (where the estimate should rely more on the gyroscope measurements) and how well does the measurement update correct the drift caused by the gyroscope bias using the accelerometer data.

For the value of $W = 10^5$ the estimate performs poorly when the pitch angle changes rapidly. On the contrary, for $W = 10^8$ the estimate is good at the begining but then the effects of the bias corrupts the estimate and it is not corrected by the measurement

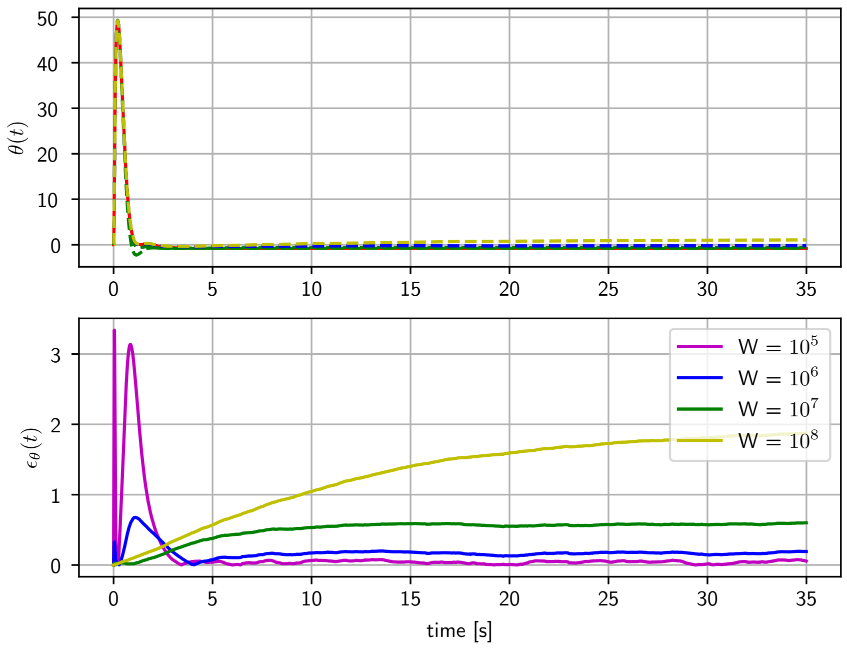


Figure 5.1: Effect of the measurement noise covariance in the estimate

update. Between the remaining two options, it is preferred the one with $W = 10^6$ since it tracks better the pitch angle during rapid movements and that is the moment where it is more critical to know the angle for stability control purposes. Therefore we choose for the covariances **W** and **E**

$$\mathbf{W} = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix} \qquad \mathbf{E} = \begin{bmatrix} 10^6 & 0 & 0 \\ 0 & 10^6 & 0 \\ 0 & 0 & 10^6 \end{bmatrix} \tag{5.13}$$

5.4.2 Object moving freely in space

To perform a more exhaustive test on the Extended Kalman Filter, testing will be made on IMU measurements simulated from an object moving freely in space, without any restrictions on movement. Differents scenarios will be simulated, all of them listed on Table 5.1.

ID	Time	Description
S1	60 s	slow rotation movements without translation
S2	60 s	medium-speed rotation movements without translation
S3	60 s	fast rotation movements without translation
S4	60 s	medium translation without rotation
S5	60 s	slow arbitrary motions
S6	60 s	medium-speed arbitrary motions

Table 5.1: Simulations of an object moving freely in space.

In Figure 5.2 is presented the inclination and heading estimate errors from one simulation S6. The heading estimate drift over time because there is no method to correct it. This can be seen in the mean error for both estimates

$$ME_{\Theta} = 0.0089^{\circ} \qquad ME_{\Psi} = 1.15^{\circ}$$

The mean error for inclination is close to zero, which indicates that the estimate is unbiased. In the case of the heading, the error is much larger, indicating that the estimate is, in this case, underestimated, i.e. biased. Usually magnetometer measurements are used to correct heading, but the problem of employing them were already discussed. The heading estimate uses only gyroscope measurements, therefore this problem will appear in every simulation. Thus, only the inclination error will be analyzed here, and the heading estimate will be adressed again in the position estimate section because it is mostly relevant for navigation purposes only.

Figure 5.3 shows the gyroscope and accelerometer measurements from the simulation S2 and simulation S4; and in Figure 5.4 it is represented the inclination estimate of the same simulations. It can be seen that when there is only rotations taking place, the estimate is much more accurate than the one with translational movement. The reason for this is that the measurements from the acceleration can only correct the inclination estimate when they are measuring only gravity. When other sources of acceleraion appear, then the method cannot get the direction of gravity and thus, the estimate is affected negatively.

A possible way to deal with this problem is to make the measurement covariance matrix **E** be variable over time, depending on the norm of the acceleration. This means that whenever the norm of the acceleration is different from gravity (with a given tolerance), matrix **E** should be increased to reflect that the measurement is less reliable so that the method can trust more on the gyroscope for that moment.

Figure 5.5 shows a comparison between the inclination estimate with the standard EKF with a constant measurement covariance matrix and with the same matrix varying depending on the norm of the acceleration measurement, as proposed in Section 3.4.1.

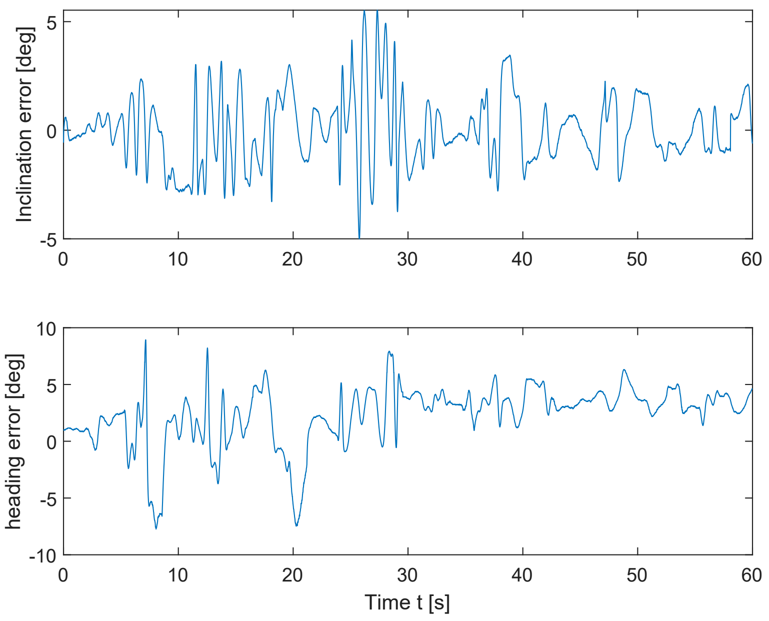


Figure 5.2: Inclination and heading errors for a simulation from scenario S6.

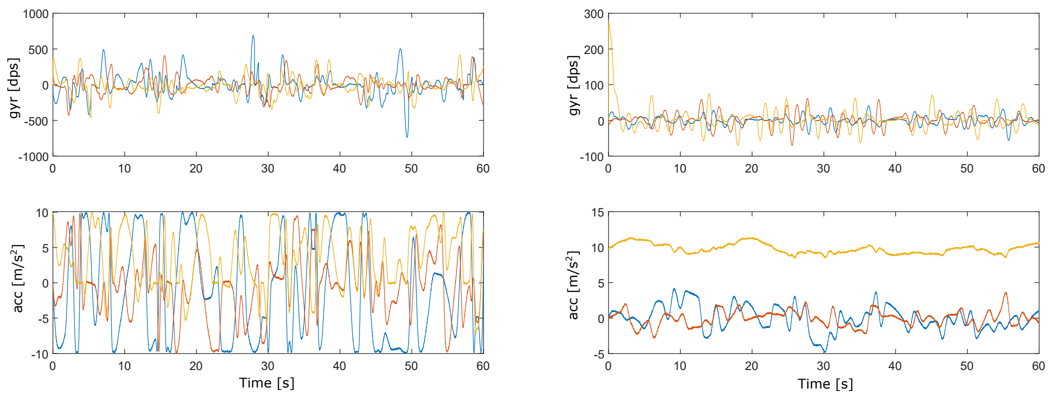


Figure 5.3: Sensor readings for simulation S2 (left) and simulation S4 (right).

To evaluate the accuracy of the estimation, 100 trials were performed for every simulation scenario, and the statistical values of the inclination mean absolute error are presented in Table 5.2.

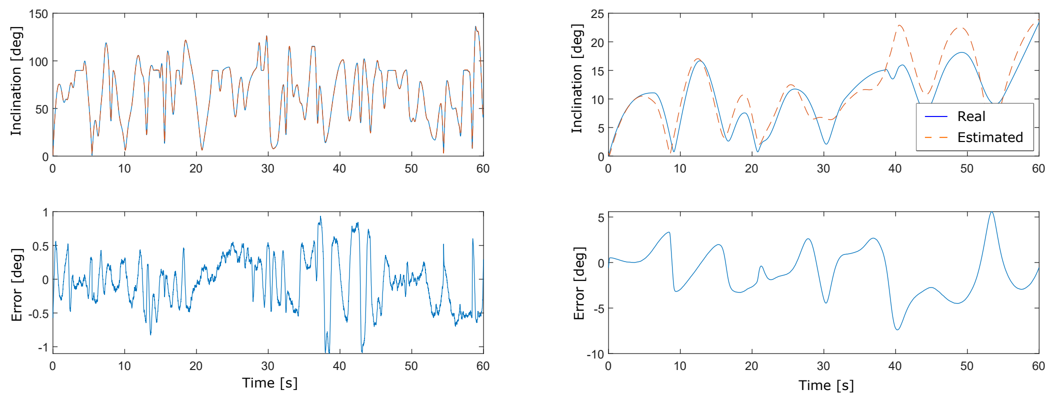


Figure 5.4: Inclination estimate and error for simulations S2 and S4.

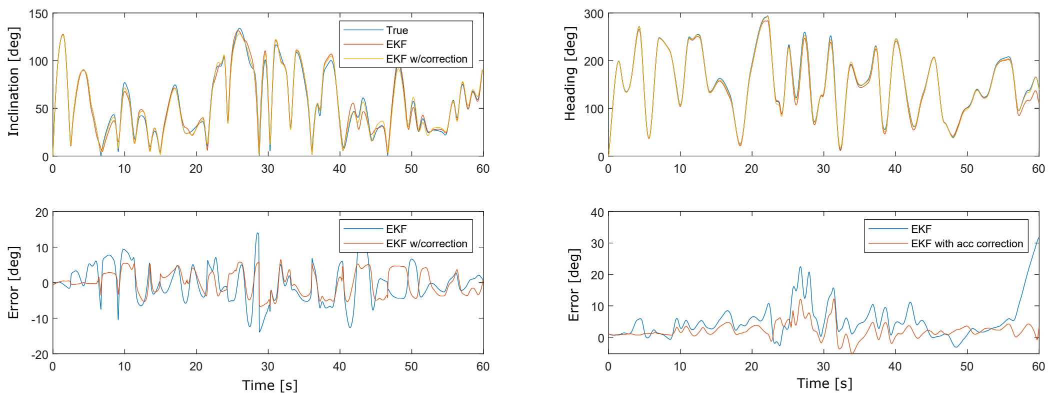


Figure 5.5: Comparison between the standard EKF and the EKF with variable measurement covariance matrix.

	EKF		**EKF 2**	
ID	**MAE$_\Theta$**	**Max**	**MAE$_\Theta$**	**Max**
S1	$0.9° \pm 0.6°$	$2.7°$	$0.8° \pm 0.6°$	$2.9°$
S2	$0.8° \pm 0.6°$	$3.1°$	$0.8° \pm 0.6°$	$3°$
S3	$0.9° \pm 0.6°$	$2.8°$	$0° \pm 0°$	$0°$
S4	$2.4° \pm 1.8°$	$10.3°$	$1.9° \pm 1.5°$	$7.7°$
S5	$1.2° \pm 0.9°$	$4.6°$	$1° \pm 0.8°$	$4.2°$
S6	$2.5° \pm 1.9°$	$8.3°$	$2° \pm 1.6°$	$7.6°$

Table 5.2: Simulations for orientation method

The EKF is able to track the inclination of the body well within 4.2°, with a mean error less than 2° for slow rotations and translations. However, the EKF does not perform well for faster motions, showing errors in inclination up to 7.6°. In all cases, the method with the variable measurement covariance matrix shows better results, specially when the translational movement is increased.

In the case of rotation-only simulations, the inclination error is not very different between the slow, medium and fast rotations. This is due to the gyroscope being able to track this angular velocities quite well, and also because the simulations considers that there is actually no translation happening. In reality this cannot be done easily. Rotation are likely to produce accelerations other than gravity because it is not possible to perfectly make an object rotate around the accelerometer axes.

5.4.3 Robotic motion

The robot does not have the possibility to move with the freedom that the object under study in the previous section did. For instace, it cannot experience rotations around the axis of movement (roll rotations) and the pitch angle range of motion goes from roughly $-90°$ to $90°$ because it is limited by the ground. The only completely free degree of freedom is the heading.

In reality, the robot could experience some roll rotation depending on how well it is balanced and the aggresiveness of the maneuver that may experience. This of course is not desired since it leads to instability and possible overturn. In the simulation environment, this is not considered in the dynamic equations and so it can be assumed that the inclination angle es exactly equal to the pitch angle.

In order to move forward, the TWIPR tilts its body forward, and backward in order to drive backwards. Figure 5.6 shows the estimated pitch angle with the EKF in blue and the given ground truth in red for the robot moving back and forth. The method was able to track the pitch angle with a MAE of 2.9° and a maximum error of 4.8°, with the highest errors taking place on the peaks. Making the measurement covariance matrix variable depending on the norm of the accelerations improves the estimate, but only a little, so it is still a large error, as shown in Table 5.3. This also shows the resulting metrics using the proposed method that uses the gyroscope measurement to modify the measurement covariance matrix and it clearly improves much more the estimate. The filtering algorithm is able to track the inclination with a mean absolute error less than 2°, considering fast dynamic movements.

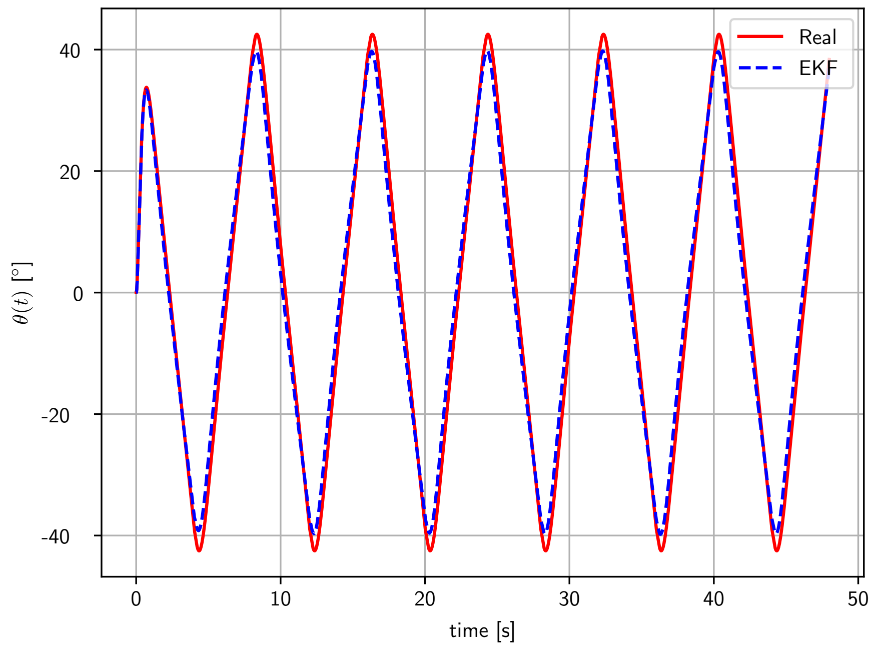


Figure 5.6: Pitch angle estimation of the TWIPR when moving back and forth.

	EKF		**EKF w/ $E(\|a\|)$**		**EKF w/ $E(\|g_\theta\|)$**	
Motion	**MAE$_\Theta$**	**Max**	**MAE$_\Theta$**	**Max**	**MAE$_\Theta$**	**Max**
Back and forth	2.9°	4.8°	2.6°	4.3°	1.7°	2.5°
Back and forth and steering	3°	5.3°	2.7°	4.5°	2°	3.6°

Table 5.3: Inclination estimate errors using the proposed orientation estimation methods on simulations of the robot moving.

When steering velocity is added to the previous motion, the results are almost the same, with an increase on the maximum error.

5.5 Methods for Position Estimation

The particle filter to be implemented in this simulation needs some variables to be selected. Given that the position estimation algorithm is required to run in real-time, the number of particles will be set to $N_s = 200$. The threshold for the resampling step will be set to 0.5.

First, navigation by odometry is investigated. Here, the systematic-errors will be evaluated on how they affect the estimate, and some insight on wheel-slip detection is given. Then, inertial navigation is compared to odometry regarding velocity estimation. Finally, both methods are combined using the complementary filter described in Section 4.3.1 and the particle filter is evaluated.

5.5.1 Odometry

Odometry can be used to keep track of the robot translational velocity and steering velocity. Depending on the severity of the systematic and non-systematic errors discussed in Section 2.3.2, the information provided by odometry can be helpful or not. In the following, some of these sources of error will be studied individually in simulations to see how they affect the estimate. Depending on the magnitud of the errors, the encoder data could be useful for estimation purposes or not. However, it is not possible to know in advance these errors, so a calibration phase must be performed on the robot to try to reduce them as much as possible, and then it can be decided if it is worth using odometry data or not.

Borenstein and Feng introduced the bidirectional square-path experiment [12] to evidenciate systematic errors in differentially driven robots. Basically, it consists in making the robot follow a square path in both clockwise and counterclockwise direction, and measuring the error in the position at the end. Once this errors are noticed, they can be properly calibrated into software to reduce them.

Two new error characteristics are defined to represent odometry errors in orientation: Type A and Type B. A Type A is defined as an orientation error that reduces (or increases) the total amount of rotation of the robot during the square-path experiment in both cw and ccw direction. By contrast, Type B is defined as an orientation error that reduces (or increases) the total amount of rotation of the robot during the square-path experiment in one direction, but increases (or reduces) the amount of rotation when going in the other direction.

Actual wheelbase different from nominal

Figure 5.7 shows a simulation where the robot turned four times for a nominal amount of 90° per turn. However, because the actual wheelbase of the vehicle was smaller than the nominal value used in the odometry computations, the estimation actually turned less degrees in each corner of the square path. One can thus observe that in both the clockwise and the counterclockwise simulation, the estimation ends up turning less than the actual amount. Therefore, the orientation error is of Type A.

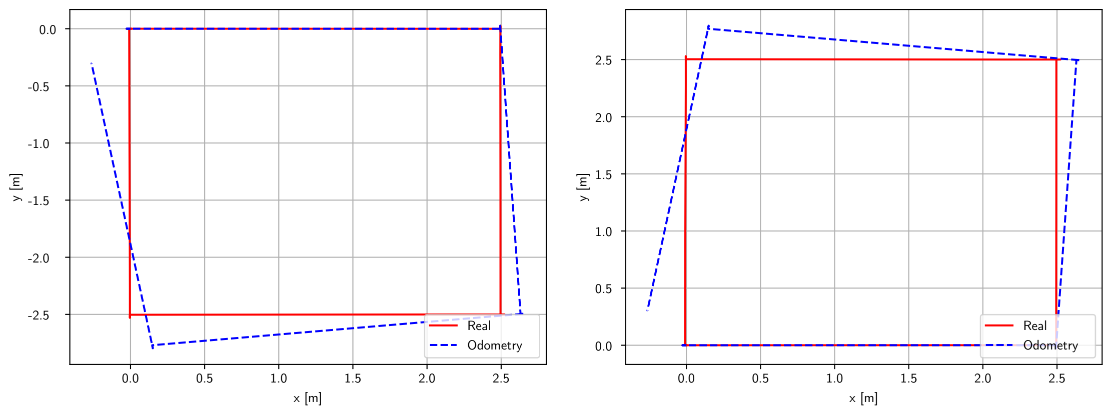


Figure 5.7: Position error due to different wheelbase than assumed. In this case, the difference between actual and nominal wheelbase is 1 cm. Nominal square path in red, actual travelled path in blue.

This error in wheelbase only affects how much the robots turns, but it has no effect on the translational velocity. This means encoders can be used to estimate velocity even though the wheelbase has not been properly calibrated.

Unequal wheel diameters

The same simulation as before is used in this case too evaluate the effect of unequal wheel diameters in the overall position of the robot.

In Figure 5.8 the trajectory of the robot with unequal wheel diameters is shown. This error expresses itself in a curved path that adds to the overall orientation at the end of the run in counterclockwise direction, but it reduces the overall rotation in the clockwise direction. Therefore, the orientation error is of Type B. Even though the difference between the diameters is around half a milimeter, the error in the position at the end becomes quite large.

This difference in wheel radius also affects the heading estimate coming from odometry. A 0.1 mm error in radius causes a heading rate of 0.36 $^{\circ}/s$ when travelling at 1 m/s.

The error in velocity is less than 0.1%, so it is possible to use encoder data to accurately estimate velocity even though the error in radius is large.

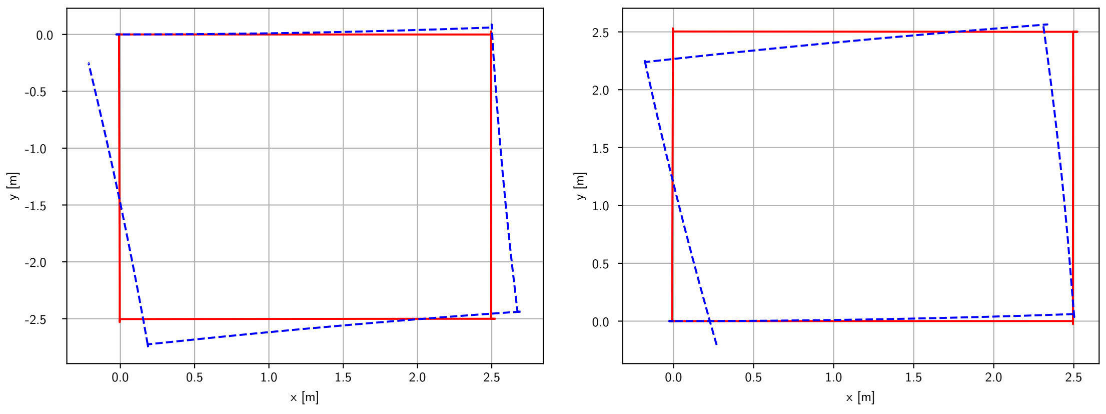


Figure 5.8: Position error due to unequal wheel radius. In this case, the difference in radius is 0.3 mm. Nominal square path in red, actual travelled path in blue.

Wheel-slippage

Wheel-slip is common when using differential drive mobile robots, specially in inverted-pendulum ones, like the TWIPR, because rapid accelerations may be needed to maintain stability and prevent the robot from falling. This non-systematic error is one of the most difficult to deal with. When slipping occurs, the encoder attached to the wheel in question will measure a really large angular velocity, and if this information is used in the estimation algorithms, the results will be erroneous.

Simulating wheel-slippage in the TWIPR is extremely complicated and beyond the scope of this thesis.

5.5.2 Inertial Navigation

An inertial navigation system directly measures linear acceleration and angular velocity and uses them to calculate the pose of the robot. The acceleration measured in the IMU reference frame can be converted to the navigation frame, and using the estimated inclination and heading, the gravitational and rotational acceleration part can be substracted to obtain only the translational part, which is the one responsible for the change in velocity and position of the robot. Unfortunately, this comes with errors since the measurements are noisy and the estimated orientation is not perfectly accurate.

The advantage of using an inertial sensor is that it is self-contained, no external motion information is needed for positioning, and its ability to provide fast, low-latency dynamic measurements. Even though very small errors in the rate information can cause an unbounded growth in the error of integrated measurements, it is a good option for moments when odometry cannot be trusted.

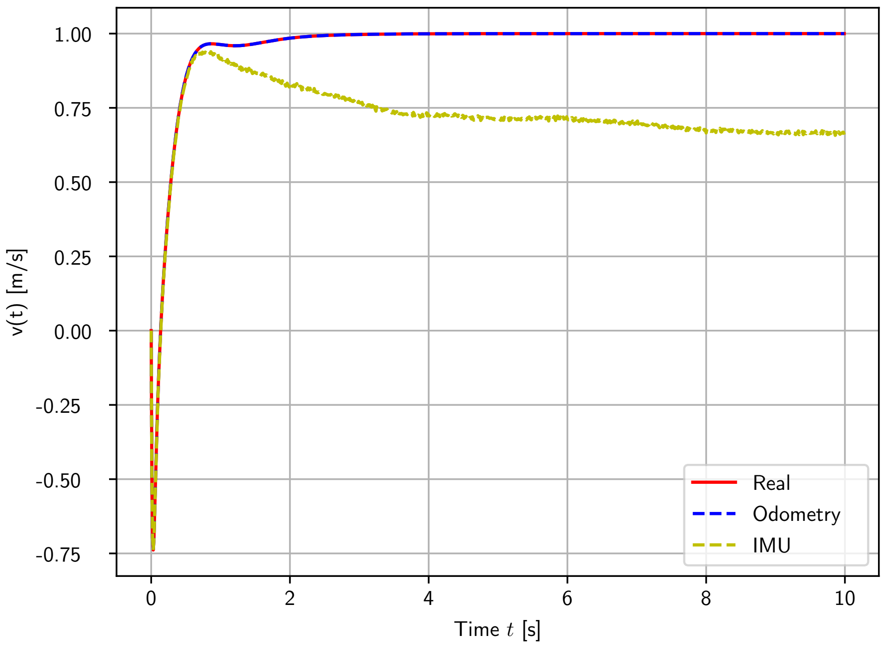


Figure 5.9: Comparison between the translational velocity computed from odometry and from inertial navigation.

In Figure 5.9 is represented the velocity estimation computed from integrating the accelerometer measurement as explained before. The estimate is not very accurate in the long-term compared with odometry. However, the important advantage of inertial navitagion is that the information provided by the IMUs is always available and the error characteristics corrupting are known. They can provide a good short-term estimate when odometry is not reliable, like in cases when wheel-slip is very likely to take place.

5.5.3 Complementary Filters

As discussed in the previous sections, both odometry and inertial navigation have their advantages and disadvantages, and the position estimates obtained are subjected to them. That is why it is important to combine them to get a better estimate of the translational velocity and the heading rate.

In the case of translational velocity, the systematic-errors on odometry do not cause much error in the estimate, so the complementary filter should rely more on odometry than inertial sensors, whenever possible (that is, when there is no risk of wheel-slip). On the contrary, the heading rate computed from inertial sensors suffers from bias and noise, while the one computed from odometry is sensitive to the systematic and nonsystematic-errors. This is a case when combining both corrupted pieces of information can yield better results than relying on only one of them.

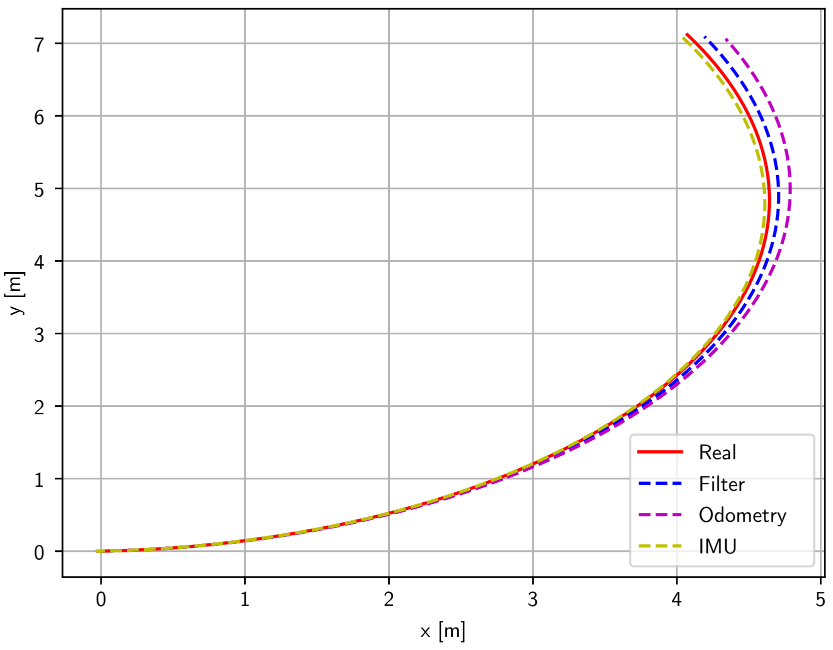


Figure 5.10: Comparison between the position estimate using different heading estimates.

Figure 5.10 shows the position of the robot after some motion computed using different heading estimates. This simulation was performed considering the right wheel to have a radius 0.1 mm larger than the left one, and a wheelbase 3 mm shorter than the assumed one. The coefficient of the filters in (4.9) and (4.10) were set to 0.5. In this case, the inertial navigation information helps to correct the heading drift caused by the systematic-errors in odometry.

The value of the coefficient must be tuned in the experimental evaluation. It is difficult to predict how well the systematic-errors of odometry can be calibrated, and there are also other sources of error than were not considered in simulations. Depending on the results obtained from experiments, the complementary filter coefficient α will be tuned to rely more on odometry or in inertial navigation.

5.5.4 The Particle Filter

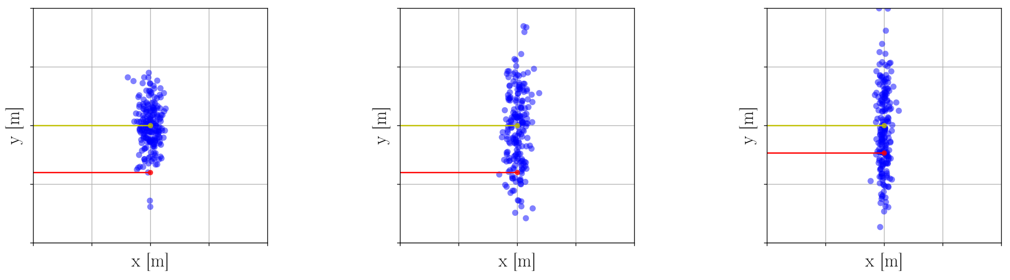
Now that the advantages and disadvantages of using odometry and inertial navigation to keep track of the robot position are known, sensor fusion can be applied to enhance each method's strenghts and improve the estimate.

As defined in Section 4.3.3, the sources of uncertainty inside the state equation are the velocity and the heading. Thus, to keep track of this uncertainty it necessary to feed adequate noise to the particle propagation step so that it is not under or overestimated.

Particles Distribution

For the purpose of this analysis, the simulations will be made with the robot moving in a straight line with constant velocity. This way, the standar deviation of the particles in the direction of movement can be assumed to be directly proportional only to the velocity uncertainty; and the deviation across the direction normal to the previous can be assumed as directly proportional to the heading uncertainty.

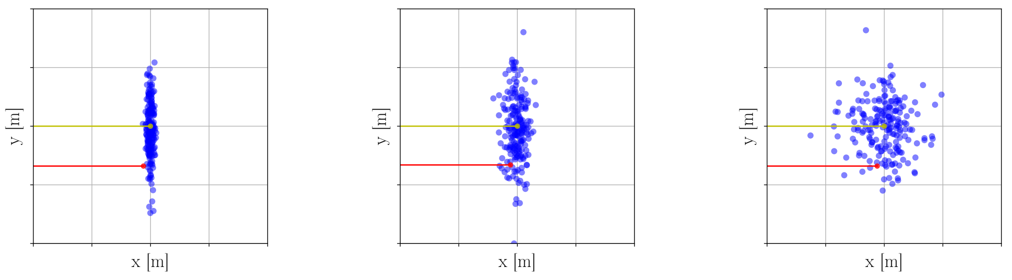
First, the heading noise is evaluated. Figure 5.11 shows the particles distribution obtained by considering different amounts of heading noise. In Figure 5.11a, the real position is at the border of the particles cloud, so there are not many samples around to account for that position. If the real uncertainty was a little larger then the point will definitely not be considered as a possible position and, without particles nearby, the estimate may take longer to converge once the measurements are available, or it might even diverge. In the case of Figure 5.11c, the uncertainty is highly overestimated and thus, the algorithm will assume that the uncertainty of the estimate is much worse that it actually is. Finally, the result of Figure 5.11b seems to represent more accurately the actual situation.



(a) $\sigma_\psi = 0.0005$ (b) $\sigma_\psi = 0.001$ (c) $\sigma_\psi = 0.005$

Figure 5.11: Particles distribution with different amounts of heading noise added. The red line represents the true position of the robot, and the yellow one represents the estimated position.

In Figure 5.12 the same procedure as before is applied but to the velocity uncertainty instead of the heading.



(a) $\sigma_v = 0.01$ (b) $\sigma_v = 0.03$ (c) $\sigma_v = 0.05$

Figure 5.12: Particles distribution with different amounts of velocity noise added. The red line represents the true position of the robot, and the yellow one represents the estimated position.

There is a special case when the robot is not moving, or doing it very slowly. If the encoders read a null velocity, then for sure the robot is not moving. There is no reason to add noise to the velocity to propagate the particles. If we do so, the uncertainty will continue to increse even though it is known that the robot is not actually moving. Nevertheless, even when the robot is not moving, it could be turning, so the system update must continue to be executed at every time step. To adress this issue, boolean logic is added to set the velocity noise to zero whenever the absolute value of the velocity is lower than a threshold.

Figure 5.13 shows the standard deviation of the particles when the robot is moving forward and stopping for some seconds to turn, and then continue moving forward. It can be seen that there are some intervals where the particles standard deviation σ_p remains constant, which coincides with the intervals when the robot is not moving forward, but at that same time the heading uncertainty continues to grow because the robot is turning. The same goes for the case when the robot is neither moving forward nor turning, but the heading uncertainty also ceases to increase.

Relative Likelihood

One way to compute the relative likelihood of the particles once a measurement is available is to use the normal probability density function (pdf).

The motion capture system can give absolute position measurements with a certain accuracy. Assuming that the error affecting that measurement is Gaussian, it is possible to compute the particles weights using this normal pdf. First, the distance between

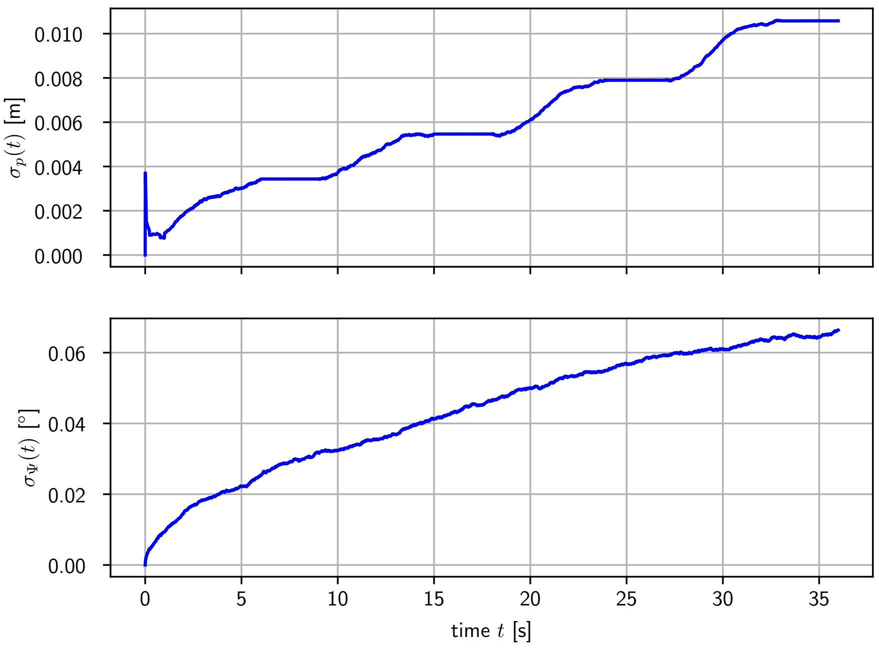


Figure 5.13: Standard deviation of the particles defined in (5.9) and in (5.10)

a particle and the measurement $\mathbf{y}$ is computed. Then using the pdf each particle is evaluated and given a weight according to how far is from the measurement, like depicted in Figure 5.14. Those particles closer to the measurement will get high weights (larger circles) than those far from it (smaller circles). The standard deviation of this pdf depends on the accuracy of the measurements.

In the case that the amount of noise added to the system update step discussed in the previous section is not enough, and the real position of the robot gets away of the cloud of particles, once the measurement are available the particles will slowly converge to the real position; at least theoretically. In practice, the weights assigned to the particles far away from the measurement are limited by numerical problems. Each weight is represented by a certain amount of bits, so when the measurement update is computed, the computer may not distinguish from a weight around 10^{-10} and another around 10^{-11}. Both particles will then get the same weight even though one of them is slightly more likely than the other.

To avoid this problem, it is important to properly calibrate the amount of noise feeded to the particles propagation step in (4.14) so that to ensure that the real position of the robot is contained within the cloud of particles.

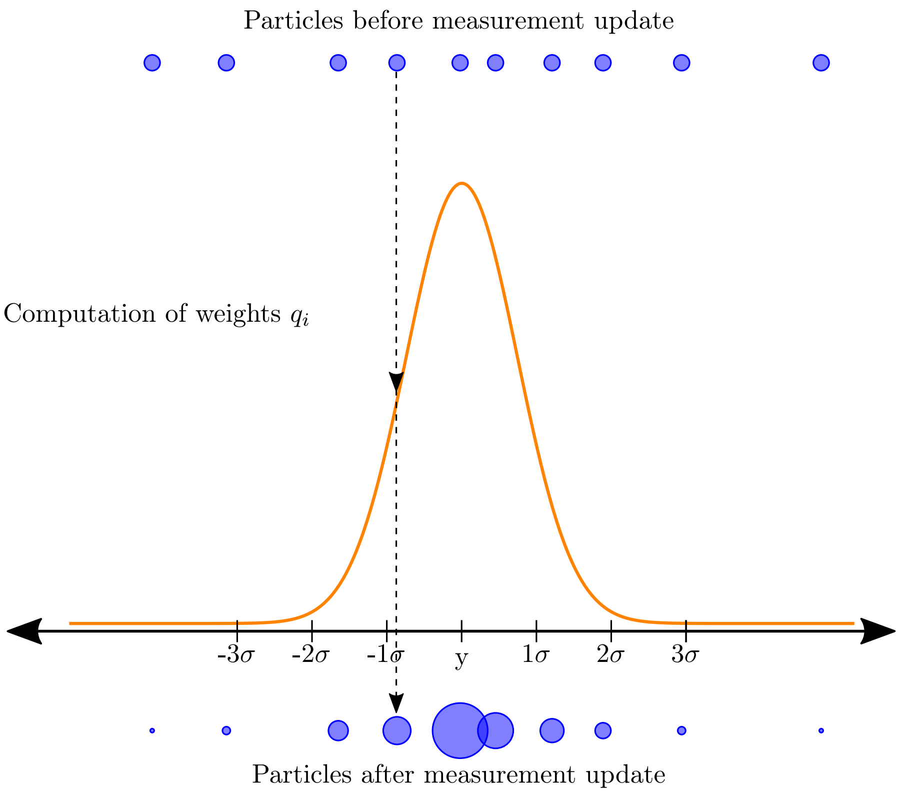

Particles before measurement update
Computation of weights q_i
-3σ
-2σ
-1σ
y
1σ
2σ
3σ
Particles after measurement update

Figure 5.14: Computation of the particles weight using the probability density function of the motion capture system.

Heading Correction

It was already mentioned that each particle represents a possible state of the robot. That means that each of them show the position of the robot with different headings. The interesting part in this comes when absolute position measurements become available. By using this measurements, the particles closer to the measurement will get more weight than the others, and since those same particles are closer to the actual position because their heading is more accurate than the others, then it is possible to correct the heading this way.

Figure 5.15 shows the estimated heading angle of the TWIPR moving on a straight line. The EKF estimate drift over time as explained in Section 5.4, but the particle filter estimate shows some improvements. First of all, the estimate drifts slower because the algorithm makes use also of the encoders to estimate the heading (just in the case where the systematic-errors are calibrated accurately, and there is no large issued related with non-systematic ones), and thus the negative effects of gyroscope intregration can be reduced. Second, when the position measurements from the motion capture system begin to be feeded to the filter at $t = 20$ sec, it can be seen how the estimate corrects itself,

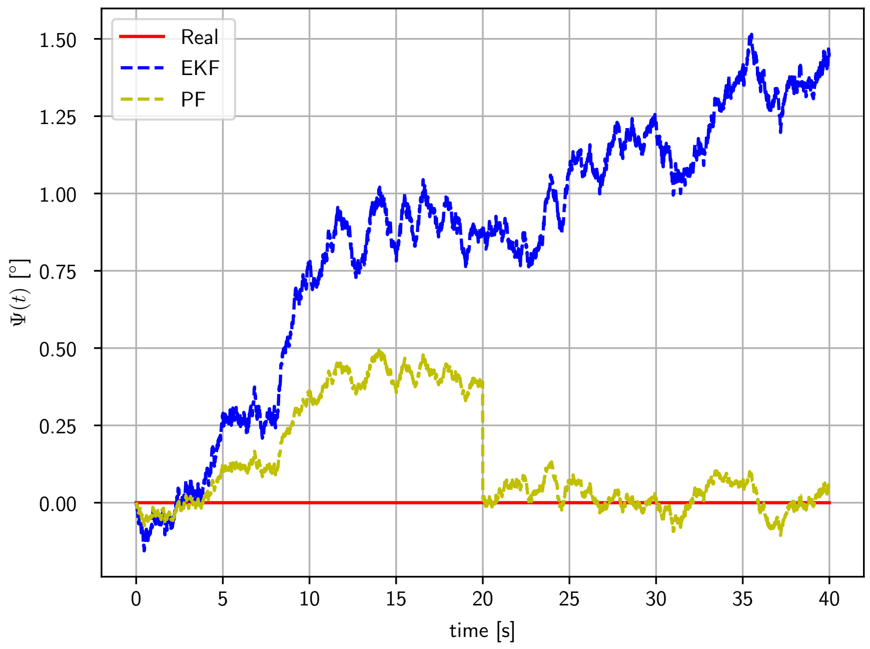


Figure 5.15: Heading angle estimated from the EKF and from the particle filter, with absolute position measurements becoming available at $t = 20$ sec.

for the reasons explained before. Simulations showed that with these measurements, the heading angle error can be kept below 1 degree.

This result also depends on what was stated in the previous section regarding the particles distribution. If the real position of the robot is not near the cloud of particles, then it means that none of the particles is considering a heading close to the real one. The position can be corrected by shifting the particles near the measurement, but the heading cannot. So in order to be able to correct the heading, the real state of the robot must be contained within the ranges that the cloud of particles is considering, that way at least some of the state hypothesis represented by the particles will be close to the real one.

5.5.5 Straight Path

One of the proposed learning experiments that the TWIPR will be put through is driving through a straight tunnel (see Figure 5.16). This means that the robot will not count with absolute position measurements and must rely on its relative tracking capabilities to estimate its position and avoid collisions with the tunnel walls. This is a scenario where the systematic-errors of the drivetrain play a major role, so it is necessary to study the relationship between the amount of uncertainty in the drivetrain dimensions and the deviation from the straight path.

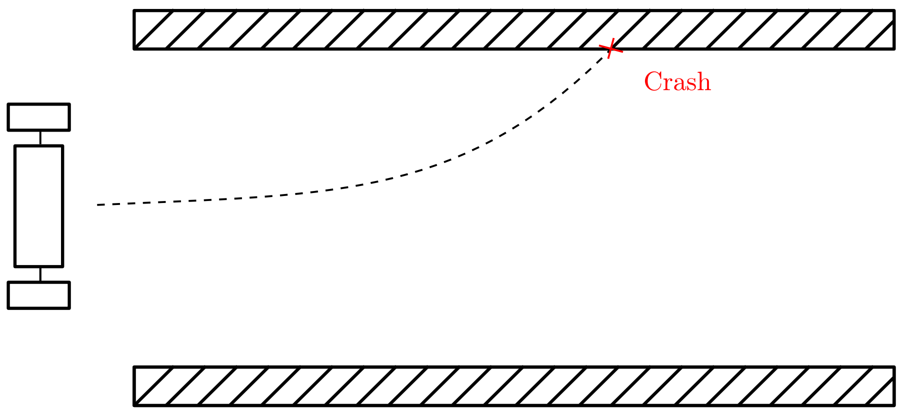


Figure 5.16: Simulation scenario: driving through a straight tunnel.

The robot is controlled by translational and rotational velocity commands. In this scenario, only translational velocity is given to make it move forward and across the tunnel. But even if both motors make the wheels rotate at the exact same angular velocity, the difference in wheel diameter will cause the robot to drift from the nominal path. In Section 5.5.1 it was established that the wheelbase error does not affect the position deviation from the desired path. It is the difference between the wheels radius that causes the robot to move through a curved path instead of the desired straight one.

To further analyze this, we will consider for this straight-tunnel simulation scenario a tunnel width of 50 cm. The robot has a wheelbase of 0.28 cm and a total width of 0.325 m, so that gives a 8.7 cm space between the robot and a wall, considering that the robot starts exactly at the center of the tunnel.

Figure 5.17 shows the relation between the distance that the robot travels through the tunnel before making contact with one of the walls, and the difference between both wheels radius. This relation has been obtained by giving the robot only a translational velocity command, and considering that as a result, each motor actuating the wheels is able to generate exactly the same rotational speed on its axis, at each time instant. As expected, the relation is inversely proportional. With both wheels rotating at the same speed, due to one of them beeing larger than the other in diameter, the resulting trajectory traversed by the robot is curved. The interesting part is that even what may seem as a small error of 1 mm can cause the robot to only be able to travel 1.65 m before crashing into a wall.

Nevertheless, this is where the position estimation algorithm and sensor fusion can help avoid this outcome, or at least extend the distance travelled. Even though odometry will indicate that the robot is moving on a straight line, the heading rate caused by the

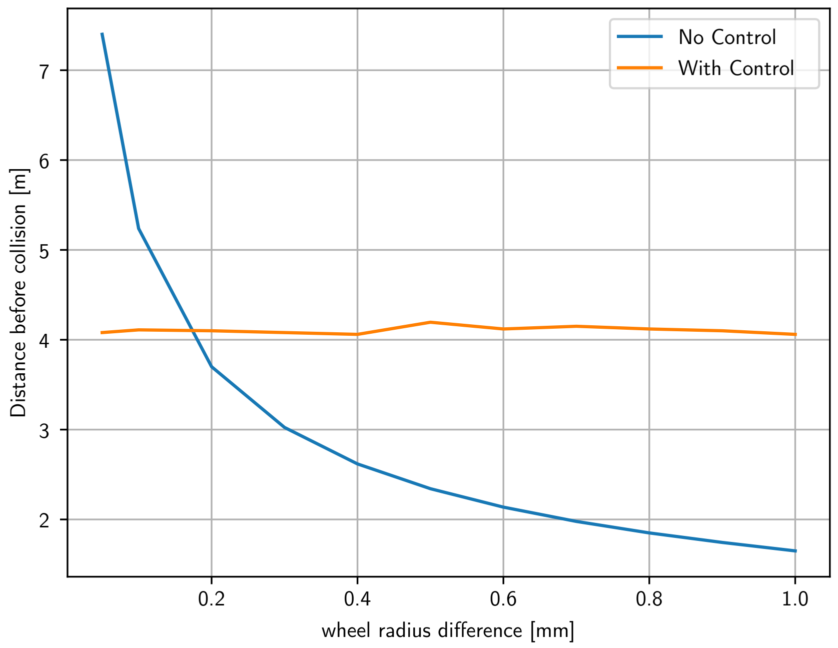


Figure 5.17: Relation between the unequal wheel radius and the distance traveled through a tunnel of 50 cm width before collision with a wall, considering that the robot is given only translational velocity command (labeled as "No Control"), and when a heading proportional controller is used to correct the effects of the unequal wheel diameter based on inertial heading estimation (labeled as "With Control").

unequal wheel radius will be detected by the gyroscope, so the complementary filter used to compute the heading rate of the robot will come up with a combination between those two sources of information.

Figure 5.18 shows the particle filter estimation, which uses the heading rate computed by the complementary filter. It can be seen that it can detect the deviation from the desired path. If the coefficient of the filter is tuned to trust more on the IMU data, then the particle filter estimate will be closer to the real position. This is just to show qualitatively the advantages of sensor fusion algorithms. In a real application, several experiments must be performed to compensate the systematic-errors and then to determine which sensorial data is more reliable and adjust the complementary filter coefficients.

This position estimate can be used to correct the trajectory of the robot and thus, extend the distance travelled before collision or, if possible, avoid it at all. Figure 5.17 also shows the resulting tunnel length travelled before collision, if a simple proportional controller

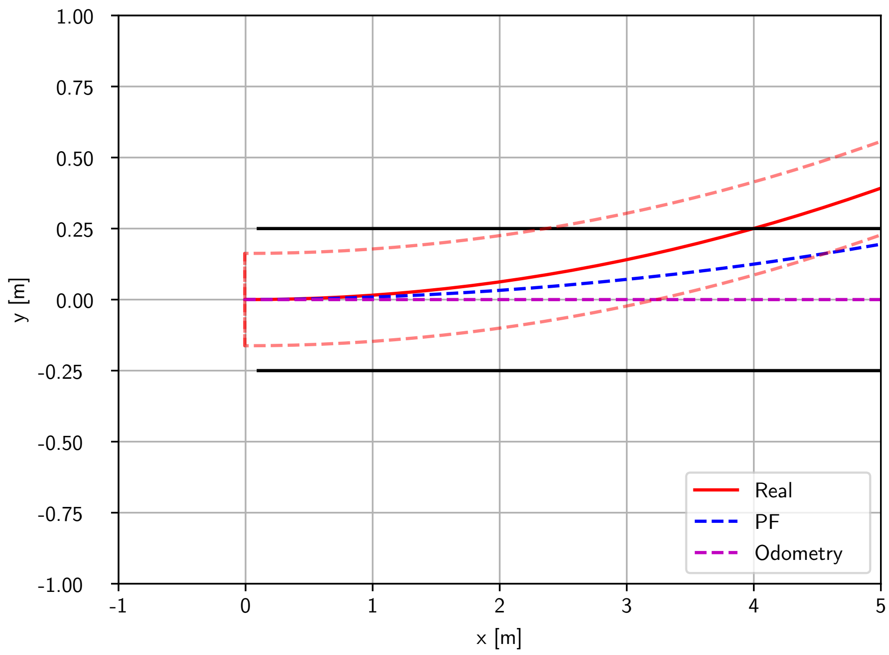


Figure 5.18: Position estimation comparison using the complementary filter to compute the heading rate with a coefficient of 0.5. The red dashed lines represent the width of the robot.

to correct the heading of the robot is used. As the curve shows, in this case it does not matter the amount of unequal wheel radius because the control strategy relies on the heading estimated by IMU data. The change in heading caused by this radius error will be measured by the gyroscope, but what limits the distance travelled is the drift in the heading estimate caused by the strap-down integration of the noisy measurements. Eventually, the estimated state will drift from the real one, casuing the robot to crash. However, simulations indicate that the robot would be able to traverse 4 meters before crashing.

As a final remark on this experiment, it is important to note that while the error in radius is large, IMU data can help the robot travel longer straight distances without absolute position measurements; but if the error in radius is small, it is actually odometry the one that could improve this aspect. Therefore, no definitive conclusion can be derived regarding which method will yield the best result. Only after proper calibration and experiments can we begin to start answering this question.

Even though it was said at the beginning of this section that the wheelbase error does not affect the position of the robot when moving on a straight path, it does affect the odometry computations, specially the heading rate. Since this is used by the complementary filter, reducing the wheelbase error will improve the particle filter estimate.

5.5.6 Curved Path

Now a more challenging scenario will be studied in which the robot will follow an eight-shape path (see Figure 5.19), where in some parts of it, the motion capture system measurements will not be available due to oclusion. The idea is to simulate a situation where the robot looses the absolute position measurements, which can happen when it leaves the range of the cameras, or when it enters a tunnel, or simply when something is blocking the line of sight of the cammeras. When this happens, the robot must rely on the other sensors to keep track of its position and continue to follow the desired path and avoid collisions.

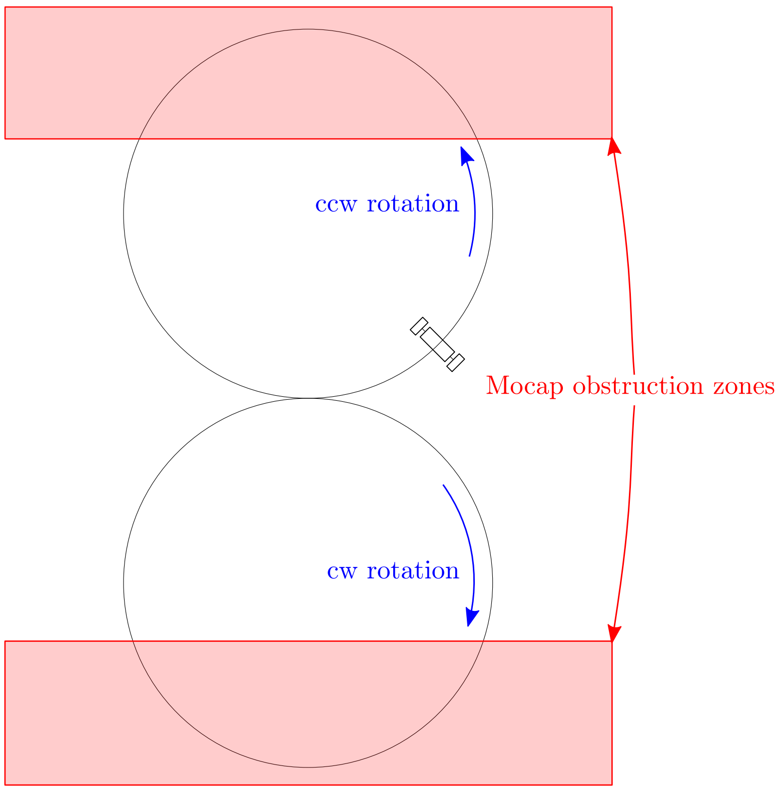


Figure 5.19: Simulation scenario: driving through an eight-shaped path.

The eight-shaped path was chosen to make the robot drive through a curved path both in clockwise and counterclockwise direction to avoid the problems mentioned in Section 5.5.1. In that section, it was shown that depending on the direction of rotation, Type A and Type B errors can add up or compensate each other.

For this simulation, the right wheel will be considered larger in diameter than the left wheel, and the real wheelbase will be smaller than assumed. This means that during the ccw rotation, the systematic-errors in odometry will add up, but they will compensate each other during cw rotation. Therefore, the larger deviation from the real path should occur during the ccw rotation. The robot will be given constant velocity commands for both parts of the path (the ccw and the cw), where the translational velocity will be set to 1 m/s.

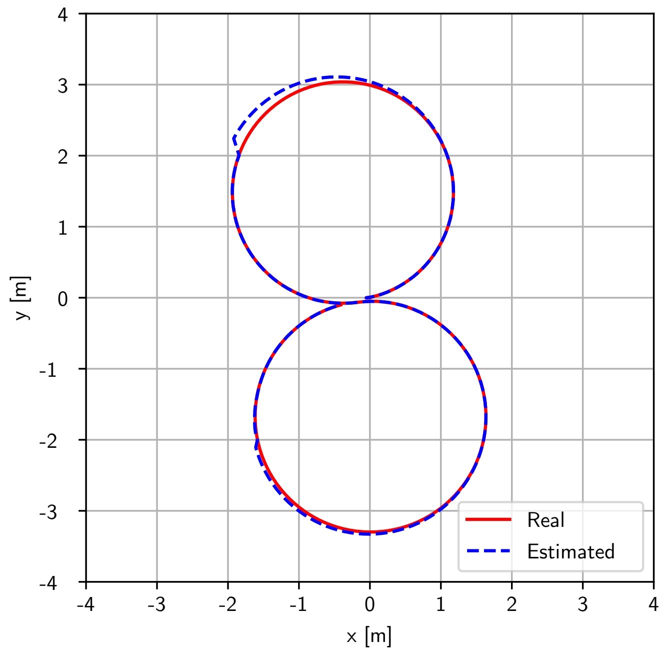


Figure 5.20: Real and estimated position of the robot traversing a curved path, using odometry to estimate the heading rate in the particle filter.

Figure 5.20 shows the results of the mentioned simulation. It is important to note that the real path travelled by the robot is not exactly the eight-shaped one described before. This is because the dynamic equations used to simulate the motion were derived considering both wheels exactly equal; thus, the effect of unequal radius is added afterwards to modify the resulting heading of the robot.

Figure 5.21 shows the tracking error E_p over time, and indeed the deviation from the real position is larger during the ccw rotation, for the reasons mentioned before. This is another case where the advantages of sensor fusion can be exploided to reduce the impact of these systematic errors in the estimation performance. The same figure shows also the tracking error if IMU data is used to estimate heading instead of odometry, and clearly the tracking error is reduced significantly.

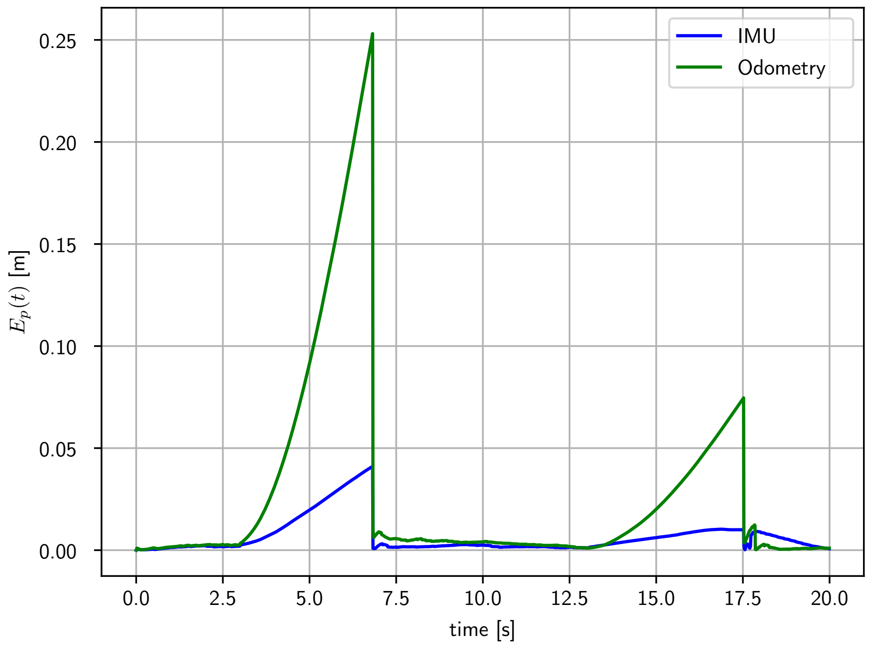


Figure 5.21: Tracking error over time during the curved path simulation, using odometry and IMU data to estimate heading.

This leads to wonder if odometry data is really useful for heading computation. Simulations show that even very small dimensional errors in the drivetrain can cause large deviations in the estimate in a relative short time. IMU data also causes drifts but at a much slower rate. These results tend to indicate that using odometry will only make the estimate worst. So far only unequal wheel radius and wheelbase difference has been addressed, and there are many other sources of uncertainty related to the drivetrain such as misalignment between both wheels axis, misalignment between the motor axis and the wheel axis, frictions, wheel-slip and uneven floor.

In the end, it will depend on how well can the systematic-errors be calibrated; on the magnitude and frequency of nonsystematic-errors, and on how noisy the IMU data is. All of these must be studyied in experiments to decide the most suitable coefficient for the complementary filters.

Nevertheless, it is the belief of the author of this thesis that using only IMU data to compute the heading rate during motion will yield the best results. Odometry should be used for this also, but only to detect when the robot is not moving at all, so that the drift resulting from the strapdown integration of the gyroscope can be reduced or eliminated. On the contrary, to estimate the translational velocity, odometry is more accurate than the IMU (assuming no wheel-slip), but including information from the latter in some degree will not cause significant errors in the estimate. In conclusion, the best possible results can be obtained by making use of adaptive complementary filters, which can modify their coefficients when certain conditions are met.

To continue with the analysis, now we are going to address the particles distribution. Figure 5.22 shows the particles distribution after some time without any mocap data to perform measurement updates. The time instant corresponds to a moment during the ccw rotation. The main interest here is to see if the distribution is representative of the real phenomena. Each particle represents a state hypothesis, that is, a certain position and heading obtained after considering different heading rates and translational velocity, as proposed in Section 4.3.3.

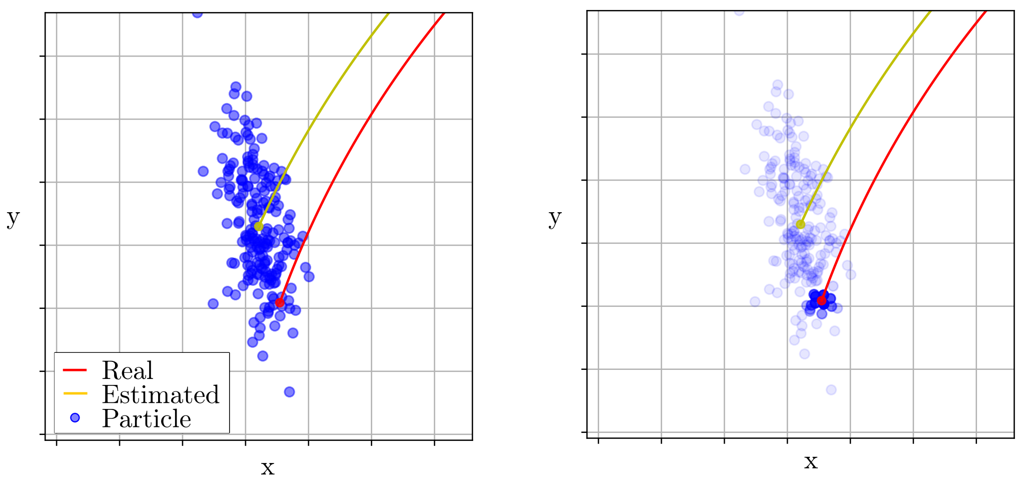


Figure 5.22: Particles distribution and weights before (left) and after(right) the motion capture system registers a position measurement. The weights are represented by the particles transparency.

The robot is moving through a curved path. An hypothesis considering a larger heading rate than the real one would lead to a curved path with higher curvature; while a smaller heading rate would lead to a curved path with smaller curvature. What is shown in Figure 5.22 represents exactly this point. If the estimated velocity is more accurate than the heading rate, as it was shown in previous sections that it is, then all particles can be considered travelling at almost the same speed; therefore, those particles considering a larger heading rate, and so traversing a curved path with higher curvature will be "ahead" than those with smaller heading rate. It was said that due to odometry errors, the estimated heading rate was smaller than the real one, and that is why the estimated path follows a curved path with a higher radius of curvature than the real position. If the amount of noise feeded to the particle propagation step is adecuate, the particles distribution will include the real position of the robot inside it. This is important because when a position measurement arrives, it can quickly update the estimates and correct the heading as shown in the right figure.

Chapter 6

Conclusion

6.1 Summary

The methods derived in this thesis can be divided in two parts. The first part focuses on the the orientation estimation of the mobile robot; while the second part focuses on the navigation problem, which includes the position and velocity.

Estimating the inclination of the robot is a key aspect for the state-feedback controller that stabilizes the system. The proposed method to do so needs to be fast, accurate and magnetometer-free, so an Extended Kalman Filter based estimation algorithm is implemented. The basic algorithm showed that it was able to track the inclination with errors less than $5°$; and the proposed method that exploits the dynamics of the system to make the measurement covariance matrix variable over time improved the estimation, reaching maximum errors of $3.6°$ and a mean-absolute-error below $2°$.

The advantage of this method comes also by its easines to be executed that allows to execute it at a high rate. Given the importance of the pitch angle estimate for the stability control of the robot, it is necesarry to update it rapidly, and with this algorithm it is possible to do it as fast as new measurements are available, up to 1000 Hz, depending on the hardware.

Regarding the navigation problem, the simulation analysis performed revealed interesting results. Usually, odometry has been the main source of information to keep track of a mobile robots' trajectory. Here it has been shown how the systematic-errors that are inherent to any drivetrain system can rapidly drift from the real position, even for what may seem to be very small errors in dimensional variables. Therefore, this thesis showed how the use of IMUs could improve the position tracking problem, specially when it comes to heading estimation.

Nevertheless, both approaches (odometry and inertial estimation) have their own advantages and disadvantages. Thus, applying sensor fusion methods to combine both sources of information has proved to yield the best results. It is difficult to know in advance the severity of all the sources of error, and if they can be calibrated or not. This means that a method that yieds good results for one application may prove not useful for another. So instead of choosing between odometry and inertial navigation, given the tecnological advances in MEMS IMUs which have made them more accurate and low-cost, it is now possible to use both of them together using sensor fusion algorithms.

The particle filter can provide a good estimate of the possible position and heading of the robot. The particles distribution is consistent with the soruces of uncertainty affecting the systems' equations of motion, namely the velocity and the heading. The algorithm allows to consider several hypothesis which then can be evaluated when a measurement is available and thus, correct the estimate in the process.

The next step would be to perform an experimental evaluation to assess if indeed IMU data could yield better results than odometry when it comes to heading estimation.

6.2 Future Work

All the results obtained in this thesis come with some limitations. Simulations are useful to get a first overlook to the real problem, but they are often based on many assumptions and this thesis is not the exception.

To begin with, the simulation of the robot comes from a model which does not consider the possibility of wheel-slip or a wheel loosing contact with the floor. This kind of problems are really difficult to simulate, and given the dynamics of the robot, it is actually likely to happen in a real application. This is usually addressed by limiting the torque of each wheel or adjusting the type of floor to increase the friction between the wheels and the floor, but it cannot be guaranteed that wheel-slip will not take place. This issue must be taken into account because all information computed from encoder measurements becomes useless when wheel-slip happens, and it could quickly lead to high estimate errors.

IMU data was simulated including bias, white noise and bias instability, but no misalignment or non-orthogonality errors were considered. Furthermore, it was assumed that the exact position of the inertial sensors' axis origin was known, which in a real application it will not be that accurately known. Also the axis of the IMU were considered parallel to the axis of the body frame, but in the real robot there may be some inaccuracies when mounting the IMU and thus, both reference frame could have some unknown deviation. Nevertheless, misalignment and non-orthogonality are well calibrated during the manufacturing process, and the errors that this can introduce are constant so they can be calibrated easily. The IMU model derived for the simulations is quite sofisticated, so the main source for differences between simulations and experiments will not come frome the mentioned model.

Regarding the drivetrain, many assumptions are made. The two motors actuating each wheel are considered equal, but in reality they will most likely not be able to provide the exact same torque or angular velocity under the same input. Both wheels may not be colinear and there is also some play between the motor axis and the wheel axis to which it is attached. The wheels, as all objects, are subjected to elastic deformations under different loads, both static and dynamic. They also do not have a constant radius across their circumference, so even if properly calibrated, the resulting wheel radius will only be an average value. All of these uncertainties are inherent of every drivetrain system, so no matter how accurate the encoders used are, these errors will play their part on the estimation accuracy.

Finally, the particle filter presented here has one issue that needs to be adressed. In a certain situation when the position measurements are not available to update the predictions, the uncertainty starts to grow, and so does the distribution of the particles according to the amount of noise feeded to the system update step. But if this noise is underestimated, the real position of the robot could get far away from the cloud of particles representing the belief on the position. Theoretically, when the measurements are reintroduced after this moment of oclusion, the particles will converge to the real position given by the measurements. Unfortunatelly, due to numerical limitations of the operating system this is not the case. When the particles are far away from the real position, the weights given to them by the current measurement update step will assign really small values, and since this weight is represented by a finite amount of bits, the computer may not distinguish between such small numbers, it will asign them the same lowest number that it can recognize. Therefore, all particles could end up with the same weight, even though some of them are a little more likely than the others.

The approach proposed here is simply to try to ensure that enough noise is beeing introduced in the system update step so that the real position of the robot is more likely to be accounted for. This solution also helps in correcting the heading estimate when the measurements are used. However, it is difficult to determine how much noise is adecuate, because overestimating it could be as unfavourable as underestimating it.

A possible solution could be to just shift the particles to the measured position whenever it is detected that after the measurement update, all particles remained with the same weights. The problem with this approach is that while the estimated position will get corrected, the heading will not. Furthermore, it will actually estimate that the real heading is the one of the particle that was closer to the measurement after the position shift, which most likely will not be the right heading.

Another solution could be to increase the standard deviation of the probability density function used to compute the weights in Section 5.5.4 so that it can assign different values to each particle and bring them to an eventual convergence. Afterwards, this standard deviation should be reduced accordingly when the estimate gets closer to the real position.

Appendices

Appendix A

Sensor measurement models

In this annex, measurement models for the sensors used in the TWIPR are be presented. This models were used to perform simulation anaylisis.

A.1 Accelerometer

With all the considerations and assumptions discussed in Section 2.2.1 and the error characteristics presented in Section 2.3, the accelerometer measurement model simplifies to

$$\mathbf{y}_a(t) = {}^{\mathcal{N}}_{S}R\left({}_{\mathcal{N}}\boldsymbol{a}_{nn}(t) - {}_{\mathcal{N}}\boldsymbol{g}\right) + \mathbf{b}_a(t) + \boldsymbol{\eta}_a(t) \tag{A.1}$$

For simplification, the terms ${}_{\mathcal{N}}\boldsymbol{a}_{nn}(t) - {}_{\mathcal{N}}\boldsymbol{g}$ will be unified in one single acceleration ${}_{\mathcal{N}}\boldsymbol{a}$.

The acceleration ${}_{S}\boldsymbol{a}$ represents the true value that the sensor is measuring. In this case, the acceleration that the sensor experiences during motion of the robot is given by the sum of a translational acceleration and an acceleration due to rotations. All of these components can be obtained from the true state of the robot in the global reference frame and must be transformed to the IMU reference frame.

The gravity vector is constant in the global coordinates and in the opposite direction to the z axis. In a steady situation, without any movement of the object, the influence of gravity can be expressed as:

$${}_{S}\boldsymbol{g} = {}^{\mathcal{N}}_{S}R\,{}_{\mathcal{N}}\boldsymbol{g} = {}^{\mathcal{N}}_{S}R\begin{bmatrix}0\\0\\g\end{bmatrix} \tag{A.2}$$

Measurements are affected also by linear acceleration. Considering that the translational acceleration obtained by the motion of the wheels is given only in the x direction in the

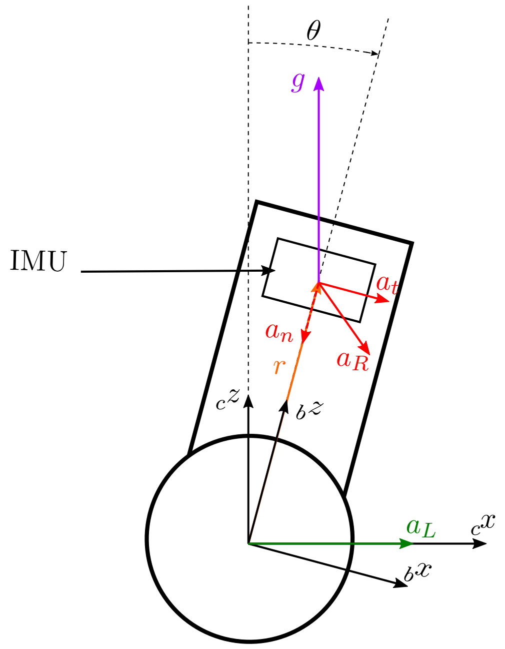


Figure A.1: Accelerations experienced by the robot

navigation reference frame:

$$ {}_s\boldsymbol{a}_L = {}^c_sR\,{}_c\boldsymbol{a}_L = {}^c_sR \begin{bmatrix} a_{x,L} \\ 0 \\ 0 \end{bmatrix} \tag{A.3}$$

However, to compute the acceleration during the movement of the object, we should also analyze rotation dynamics. The rotation axis of the object is translocated as compared to the rotation axis of the sensor by a vector $\boldsymbol{r}$ (having r_x, r_y and r_z as components). As the result of rotation dynamics, the centrifugal and tangential accelerations should be taken into consideration:

$$ {}_s\boldsymbol{a}_R = {}^c_sR\,{}_c\boldsymbol{a}_R = {}^c_sR({}_c\boldsymbol{a}_t + {}_c\boldsymbol{a}_n) \tag{A.4}$$

Where the tangential acceleration is given by:

$$ {}_c\boldsymbol{a}_t = {}_c\dot{\boldsymbol{\omega}} \times {}_c\boldsymbol{r} \tag{A.5}$$

And the normal accelerations are given by:

$$ {}_c\boldsymbol{a}_n = {}_c\boldsymbol{\omega} \times {}_c\boldsymbol{v} = {}_c\boldsymbol{\omega} \times {}_c\boldsymbol{\omega} \times {}_c\boldsymbol{r} \tag{A.6}$$

Summing up all these contributions, we get the following measured accelerations:

$$ {}_s\boldsymbol{a} = -{}_s\boldsymbol{g} + {}_s\boldsymbol{a}_L + {}_s\boldsymbol{a}_R \tag{A.7}$$

A.2 Gyroscope

With all the considerations discussed in Section 2.2.1 and the error characteristics from Section 2.3, the gyroscope measurement model simplifies to

$$\mathbf{y}_\omega(t) = \boldsymbol{\omega}_{\text{nv}}(t) + \boldsymbol{b}_\omega(t) + \boldsymbol{\eta}_\omega(t) \tag{A.8}$$

and the left subindex denoting the sensor frame s has been omitted for clarification.

When simulating the robot, the navigation rate can be obtained from the true state of the robot by transforming the angular velocities $\dot{\phi}$,$\dot{\theta}$ and $\dot{\psi}$ from the Euler intrinsic "zyx" convention in which are given, to the sensor reference frame $\mathcal{S}$.

First, we compute the orientation of the body by following the convention, first a rotation ψ about the z-axis and then a rotation θ about the new y-axis.

$$^{\mathcal{B}}_{\mathcal{N}}\boldsymbol{q}_k(t) = {}^{\mathcal{B}}_{\mathcal{N}}\boldsymbol{q}_\psi(t) \otimes {}^{\mathcal{B}}_{\mathcal{N}}\boldsymbol{q}_\theta(t) \tag{A.9}$$

With this orientation, and the orientation of the previous time step, we can get the relative orientation $d\boldsymbol{q}(t)$ as

$$^{\mathcal{B}}_{\mathcal{N}}d\boldsymbol{q}(t) = {}^{\mathcal{B}}_{\mathcal{N}}\boldsymbol{q}^*_{k-1}(t) \otimes {}^{\mathcal{B}}_{\mathcal{N}}\boldsymbol{q}_k(t) \tag{A.10}$$

Finally, the angular velocity, in the global reference frame, comes from this relative orientation with the following formula:

$$_{\mathcal{N}}\boldsymbol{\omega}_{\text{nv}}(t) = \frac{A(d\boldsymbol{q}(t))V(d\boldsymbol{q}(t))}{T_s} \tag{A.11}$$

Now it is only necessary to transform it to the sensor reference frame.

$$_{\mathcal{S}}\boldsymbol{\omega}(t) = {}^{\mathcal{N}}_{\mathcal{S}}\boldsymbol{q}(t) \otimes {}_{\mathcal{N}}\boldsymbol{\omega}_{\text{nv}}(t) \otimes {}^{\mathcal{N}}_{\mathcal{S}}\boldsymbol{q}^*(t) \tag{A.12}$$

A.3 Encoders

The encoder measurements can be derived from the relationship between the rotational speed of each wheel, the velocity of the robot and the steering velocity

$$\dot{x} = \frac{r(\omega_R + \omega_L)}{2} \qquad \dot{\psi} = \frac{r(\omega_R - \omega_L)}{d}$$

Where r is the wheel radius and d is the distance between the wheels. By combining these two equations we can obtain an expression for each wheel:

$$\omega_R = \frac{\dot{x}}{r} + \frac{\dot{\psi}d}{2r} \qquad \omega_L = \frac{\dot{x}}{r} - \frac{\dot{\psi}d}{2r}$$

Appendix B

UMBmark procedure for systematic calibration

Borenstein and Feng [12] have developed a procedure for the measurement and correction of systematic odometry errors. This method is called the University of Michigan Benchmark test (UMBmark), and it requires the robot to traverse a square path. The robot does not need to be programmed to track the path and reach the exact initial position at the end because this test aims at determining the odometry errors and not control errors. Under ideal conditions, the robot would return to the starting position, but due to errors it will end its path in a different location, near the initial one. In the end, the absolute measurements of position and orientation must be compared to the ones obtained from odometry.

Since performing the test in only one direction may conceal two mutually compensating odometry errors, the experiment must be performed both in clockwise and counterclockwise direction, at least 5 times each.

The result of the experiment could look similar to the one shown in Figure B.1. From this experiment, two things can be noted: the stopping positions after cw and ccw runs are clustered in two distinct areas; and the distribution within the cw and ccw clusters are the result of nonsystematic errors.

The coordinates of the centers of gravity of the clusters are computed as

$$x_{cg,cw/ccw} = \frac{1}{N}\sum_{i=1}^{N} x^i_{abs} - x^i_{od} \tag{B.1}$$

$$y_{cg,cw/ccw} = \frac{1}{N}\sum_{i=1}^{N} y^i_{abs} - y^i_{od} \tag{B.2}$$

where n is the number of times the experiment was performed in each direction.

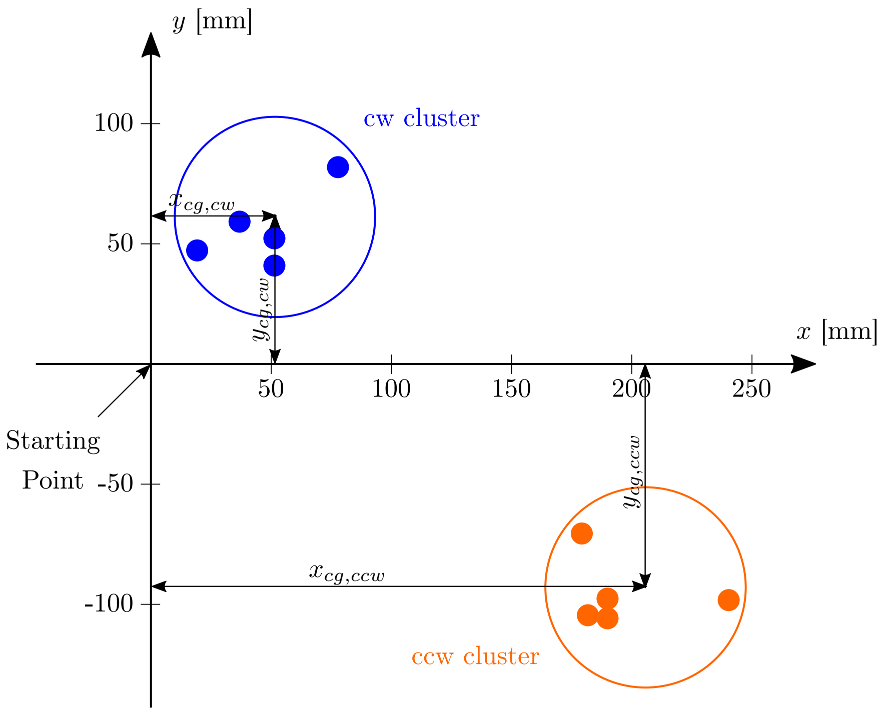


Figure B.1: Typical results from running UMBmark with an uncalibrated vehicle.

In this case, Type A errors are caused mainly by an error in the nominal wheelbase. This causes more or less turning at each corner of the square path. The amount of erroneous rotation in each nominal 90-degree turn is denoted as α. Type B errors are mostly caused by difference in wheel radius. As a result, the robot experiences a curved path instead of a straight one. The incremental orientation error at the end of each leg of the square path is denoted as β.

These two orientation errors can be computed from the results of the experiment by

$$\alpha = \frac{x_{cg,cw} + x_{cg,ccw}}{-4L} \tag{B.3}$$

$$\beta = \frac{x_{cg,cw} - x_{cg,ccw}}{-4L} \tag{B.4}$$

the radius of curvature R of the curved path can be found as

$$R = \frac{L/2}{\sin \beta/2} \tag{B.5}$$

this radius can be used to determine the ratio between the two wheel diameters E_d that caused the robot to travel on a curved, instead of a straight path

$$E_d = \frac{r_R}{r_L} = \frac{R + b/2}{R - b/2} \tag{B.6}$$

The wheelbase error E_b is directly proportional to the actual amount of rotation, so it can be computed from the proportion

$$\frac{b_{actual}}{90^\circ} = \frac{b_{nominal}}{90^\circ - \alpha} \tag{B.7}$$

so that

$$b_{actual} = \frac{90^\circ}{90^\circ - \alpha} b_{nominal} \tag{B.8}$$

Once E_b and E_d are computed, it is straightforward to use their values as compensation factors in the controller software.

References

[1] S. Kim and S. Kwon, *Dynamic Modelling of a Two-wheeled Inverted Pendulum Balancing Mobile Robot.* International Journal of Control, Automation and Systems, vol. 13, August 2015.

[2] J. D. Hol., *Sensor Fusion and Calibration of Inertial Sensors, Vision, Ultra-Wideband and GPS.* Dissertation no. 1368, Linköping University, Linköping, Sweden, June 2011.

[3] X. Ruan, J. Chen, J. Cai and L. Dai, *Balancing Control of the Two-Wheeled Upstanding Robot Using Adaptive Fuzzy Control Method.* Proc. of IEEE International Conference on Intelligent Computing and Intelligent Systems, Shanghai, November 2009.

[4] X. Ruan, J. Cai and J. Chan, *Learning to control two-wheeled self-balancing robot using reinforcement learning rules and fuzzy neural networks.* Proc. of IEEE 4th International Conference on Natural Computation, Jinan, October 2008.

[5] J. Wu and W. Zhang, *Design of fuzzy logic controller for two-wheeled selfbalancing robot.* Proc. of 6th International Forum on Strategic Technology (IFOST), Harbin, August 2011.

[6] Y. H. Wen, Y. S. Lin and Y. G. Leu, *Design and implementation of the balance of two-wheeled robots.* Proc. of IEEE International Conference on Advanced Robotics and Intelligent Systems (ARIS), Tainan, June 2013.

[7] K. Watanabe, J. Tang, M. Nakamura, S. Koga and T. Fukuda, *Mobile robot control using fuzzy gaussian neural networks.* Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Yokohama, July 1993.

[8] A. A. Bature, S. Buyamin, M. N. Ahmad and M. Muhammad, *A comparison of controllers for balancing two wheeled inverted pendulum robot.* International Journal of Mechanical and Mechatronics Engineering (IJMME-IJENS), vol. 14, 2014.

[9] Y. Shu, C. Bo, G. Shen, C. Zhao, L. Li and F. Zhao, *Magicol: Indoor Localization Using Pervasive Magnetic Field and Opportunistic WiFi Sensing.* IEEE Journal on Selected Areas in Communications, vol. 33, no. 7, pp. 1443-1457, July 2015

[10] T. Seel, *Learning control and inertial realtime gait analysis in biomedical applications: improving diagnosis and treatment by automatic adaption and feedback control* Doctoral Thesis. Berlin: Technische Universität Berlin, 2016. doi: 10.14279/depositonce- 4973. url: http://dx.doi.org/10.14279/depositonce-4973 (cit. a p. 33)

[11] Optitrack, Flex13, 2019. `https://optitrack.com/products/flex-13/`.

[12] J. Borenstein and L. Feng, *UMBmark: A Benchmark Test for Measuring Dead-reckoning Errors in Mobile Robots.* 1995 SPIE Conference on Mobile Robots, Philadelphia, PA, October 1995.

[13] F. Zafari, A. Gkelias, and Kin K. Leung *A Survey of Indoor Localization Systems and Technologies.* IEEE Communications Surveys & Tutorials, vol. 21, no. 3, pp. 2568-2599, thirdquarter 2019

[14] J. Borenstein, H. R. Everett, and L. Feng *Where am I? Sensors and Methods for Mobile Robot Positioning.* University of Michigan, April 1996.

[15] G. Kitagawa, *Monte Carlo filter and smoother and non-Gaussian nonlinear state space models.* Journal of Computational and Graphical Statistics, vol. 5, March 1996.

[16] J. Carpenter, P. Clifford, and P. Fearnhead, *An improved particle filter for nonlinear problems.* IEE Proceedings, Radar, Sonar and Navigation, vol. 146, April 1999.

[17] J. Liu and R. Chen, *Sequential Monte-Carlo methods for dynamic systems.* Journal of the American Statistical Association, vol. 93, September 1998.

[18] E. R. Beadle and P. M. Djurić, *A fast-weighted Bayesian bootstrap filter for nonlinear model state estimation.* IEEE Transactions on Aerospace and Electronic Systems, vol. 33, January 1997.

[19] Tiancheng Li, Miodrag Bolić, and Petar M. Djurić, *Resampling Methods for Particle Filtering: Classification, Implementation and Strategies.* IEEE Signal Processing Magazine, April 2015.

[20] A. Botchkarev, *Performance Metrics (Error Measures) in Machine Learning Regression, Forecasting and Prognostics: Properties and Typology* Interdisciplinary Journal of Information, Knowledge, and Management, 2019.