# Assistive Shopping Trolley Controller Design for the Elderly

*A thesis presented for the degree of*
*Master of Science in Mechanical Engineering*
*at the Delft University of Technology*

Vianne Heusdens

February 27, 2021

| | |
|---|---|
| Student number: | 4148304 |
| Supervisors: | Prof.dr.ir. M. Wisse (TU Delft) |
| | Prof.dr.ir. D.A. Abbink (TU Delft) |
| | Ir. R. Berci-Hajnovics (external) |

Faculty of Mechanical, Maritime and Materials Engineering
Delft University of Technology

# Abstract

We are living in an aging society, which is putting an increasingly heavy strain on our healthcare system. As people age many become less mobile, leading to loneliness and a deteriorating health. Subsequently elderly often end up in nursing homes; an experience which is unpleasant as well as expensive. Assisting the elderly with robotic devices can help increase their mobility, therefore reducing healthcare costs and increasing elderly satisfaction.

This thesis follows up on the master thesis of R. Berci-Hajnovics [8], where she sets out to design a novel concept for an assistive, motorized shopping trolley aimed at reducing the physical burden of carrying groceries over uneven terrain. In the current thesis, a control system for the proposed assistive shopping trolley is designed with the goal of attenuating disturbances like road inclination and added mass on the trolley's dynamic behavior. Predictability of its behavior is considered as well in order to gain the confidence of the elderly user.

An analysis of potentially suitable control types for implementation in the assistive shopping trolley has been performed, wherein their characteristics are compared on several different aspects relating to their predictability and disturbance attenuating behavior. Based on this analysis, a controller implementing a so called Disturbance Observer (DOB) is deemed best suited for the job. A DOB uses a dynamic model of the controlled plant to estimate the influence of present disturbances on the plant dynamics. This estimation can consequently be used to produce an opposing force, thereby canceling the effects of the disturbances. Furthermore, a DOB includes a filter which provides additional capabilities, such as robustness to uncertainty within the model and filtering of high-frequency noise in the data to further increase its performance. A DOB is most suitable for the current cause due to its overall adequate performance with respect to the imposed criteria, as well as its relatively high disturbance attenuation accuracy due to its ability to use the plant's dynamics in making an estimation of the disturbances at play.

Following the results of the analysis, this thesis proposes the implementation of a controller including a DOB for application in the assistive shopping trolley. In its design, a model of the trolley dynamics is being used, which provides a basis for the controller's internal dynamic model. This model is furthermore used as a representation of the trolley plant in a stability analysis of the trolley-controller closed-loop behavior. The model is based on the dynamics of an inverted pendulum on wheels, with an additional constraint on the position of the trolley handle to represent the interaction with the human user. By neglecting minor, nonlinear dynamic effects such as variations in the trolley's orientation and air drag, the general model is simplified to a linear equation.

The controller's internal model is constructed to represent the trolley's nominal (desired) behavior, determined to be that of an empty trolley on a flat road, by substituting corresponding nominal values in the parameters of the general model. Subsequently, criteria for robust stability of the trolley-controller's closed loop behavior are determined using a $H_\infty$ approach; In this approach the system's behavior is analyzed given the 'worst-case' uncertainties. Furthermore, the system's performance with respect to disturbance estimation and noise filtering is analyzed using its loop transfer function. The insights gained from these analyses are used in the design of the controller and to make recommendations for future work.

The obtained controller is implemented in the Robotic Operating System (ROS) framework; an open-source robotics platform which makes use of a decentralized system of processes, simplifying the creation and sharing of complicated software structures across a wide variety of applications. For this purpose, the controller is subdivided into multiple elements, each performing a specific function. These elements, called 'nodes', are submitted to a variety of unit tests to verify their proper implementation.

The controller's performance is put to the test in simulation using ROS's accompanying simulation engine, 'Gazebo'. The controlled trolley's acceleration error is compared to that of a regular trolley in several scenario's, where different types of disturbances are applied. The results show that the controller indeed reduces the effects of disturbances on the trolley's behavior; the controller is able to recognize the trolley's changing behavior due to an encountered disturbance and it correctly attenuates its effect by ordering the required compensatory motor torque.

More research is required with respect to the effect of uncertainty within the model dynamics to ensure stability and controllability of the trolley system. Moreover, the closed-loop system response should be tested in the presence of various types of noise to determine how well the current results apply when the controller is used in a real-life scenario.

# Preface

This masters thesis is the final step in my journey towards my masters degree in Mechanical Engineering at the Delft University of Technology. Coming from a background in Industrial Design Engineering, I am passionate about helping people through technology and design; therefore I am grateful to have had the opportunity to work on a project which is both technical and human-oriented. The work presented in this master thesis was made possible by DoBots, a company with a welcoming and inspiring atmosphere which immediately made me feel at home. I would therefore like to thank everyone working there, as well as at sister-companies Crownstone and Almende. Special thanks go to my daily supervisor Reka Berci-Hajnovics for her guidance and support during this project. Furthermore, my thanks go to my university supervisor David Abbink for taking me on as a thesis student, and university supervisor Martijn Wisse for taking over the final part of the project and supporting me across the finish line. Finally, I want to thank my friends and family, who have been a great aid to me whilst doing this research. An additional shout-out to my boyfriend Joost Meulenbeld, who has been of incredible support during both the highs and the lows that the recent period has brought me. Without you, I don't think I would have been where I am now.

*Vianne Heusdens*

# Contents

# Glossaries

## Definitions

**$H_\infty$-stability analysis** Method to determine the stability of a closed-loop system in the presence of uncertainty.

**Critical point** The point -1 on the real axis of the complex plane, indicating the point where a system's feedback signal becomes self-reinforcing.

**Deadband** A range of a parameter value around 0 in which the parameter value will still be assumed 0.

**Disturbance** Undesired changes in system parameters and environmental variables which affect the trolley dynamics.

**Gazebo** An open-source simulator specialized for use in the field of robotics.

**Loop analysis** Method in which a system's stability is analyzed by looking at the propagation of sinusoidal signals in the system's feedback loop.

**Loop dynamics** Part of the plant dynamics within a feedback system which is included in the system's loop transfer function.

**Loop transfer function** Transfer function for a signal going around the feedback loop.

**Model uncertainty** Differences between the expected and actual behavior of a dynamic system.

**Neglected dynamics uncertainty** Model uncertainty caused by dynamics which are deliberately left out of scope during the design of the controller.

**Nominal** Representing the desired instance.

**Parameter uncertainty** Model uncertainty caused by unknown deviations of parameters from their nominal value.

**Passivity** The property of requiring an input in order to operate.

**Perturbation** Deviation of a system's behavior from the average.

**Plant** An arbitrary dynamic system.

**Predictability** The property of how well the behavior of an instance can be estimated beforehand.

**Properness** A system property indicating that its output does not depend on future inputs.

**Q-filter** Internal filter used in the controller.

**Robustness** The sensitivity of a control system to model uncertainty.

**Uncertainty region** A region representing possible variations in a system's behavior in the complex plane.

**Uncertainty set** A set of all possible dynamic plant models, given a certain range of model uncertainty.

**Unmodeled dynamics uncertainty** Model uncertainty caused by unknown dynamics.

## Acronyms

**API** Application Programming Interface

**COM** Center of mass

**DOB** Disturbance Observer

**FBD** Free Body Diagram

**HIE** Human Input Estimator

**IMU** Inertial Measurement Unit

**ROS** Robotic Operating System

**SDF** Simulation Description Format

**SISO** Single Input Single Output

**URDF** Unified Robot Description Format

# Symbols

| **Symbol** | **Unit** | **Description** |
|---|---|---|
| $W$ | $[-]$ | Trolley wheel center of mass |
| $F$ | $[-]$ | Trolley frame center of mass |
| $H$ | $[-]$ | Trolley handle |
| $N$ | $[-]$ | Point of contact between trolley wheel and road surface |
| $x_W$ | [m] | x-position of trolley wheel center of mass $W$ |
| $y_W$ | [m] | y-position of trolley wheel center of mass $W$ |
| $x_F$ | [m] | x-position of trolley frame center of mass $F$ |
| $y_F$ | [m] | y-position of trolley frame center of mass $F$ |
| $\theta$ | [rad] | Trolley wheel angle |
| $\beta$ | [rad] | Trolley frame angle w.r.t. road |
| $\gamma$ | [rad] | Road inclination angle |
| $L_B$ | [m] | Height of trolley bag |
| $W_B$ | [m] | Depth of trolley bag |
| $L_{WH}$ | [m] | Height of trolley frame |
| $W_F$ | [m] | Depth of trolley frame |
| $L_{WF}$ | [m] | Distance between wheel axis and center of mass of combined trolley frame and bag, measured along the trolley frame |
| $W_{WF}$ | [m] | Distance between wheel axis and center of mass of combined trolley frame and bag, measured perpendicular to the trolley frame |
| $L_H$ | [m] | Distance in y-direction between road and trolley handle |
| $L_{H'}$ | [m] | Distance in y-direction between wheel axis and trolley handle |
| $m_F$ | [kg] | Mass of trolley frame |
| $m_B$ | [kg] | Mass of trolley bag |
| $m_W$ | [kg] | Mass of trolley wheel |
| $m_{FT}$ | [kg] | Combined mass of trolley frame and bag |
| $F_H$ | [N] | Total human force on the trolley handle |
| $F_{HX}$ | [N] | x-Component of the human force on the trolley handle |
| $F_{HY}$ | [N] | y-Component of the human force on the trolley handle |
| $F_{WX}$ | [N] | x-Component of the constraint force between trolley frame and wheels |
| $F_{WY}$ | [N] | y-Component of the constraint force between trolley frame and wheels |
| $F_{FG}$ | [N] | Gravitational force on point $F$ |
| $F_{WG}$ | [N] | Gravitational force on point $W$ |
| $F_{STAT}$ | [N] | Static friction acting on trolley wheels |
| $F_N$ | [N] | Normal force acting on trolley wheels |
| $T_M$ | [N m] | Torque provided by trolley wheel motors |
| $T_{ROLL}$ | [N m] | Rolling resistance of trolley wheels |
| $\mu$ | $[-]$ | Rolling friction coefficient |
| $g$ | $[\mathrm{m\,s^{-2}}]$ | Gravitational acceleration |
| $G$ | $[-]$ | Representation of the trolley-plant loop dynamics included in the trolley-controller feedback loop. *see: Loop dynamics* |
| $P$ | $[-]$ | Representation of a general dynamic plant model. |
| $Q$ | $[-]$ | Representation of the Q-filter dynamics. *see: Q-filter* |
| $\Delta$ | $[-]$ | Representation of a perturbation. *see: Perturbation* |

| **Symbol** | **Unit** | **Description** |
|---|---|---|
| $L$ | $[-]$ | Loop transfer function. |
| $P_a$ | $[-]$ | Representation of the average plant model. |
| $P_p$ | $[-]$ | Representation of any dynamic model within a given uncertainty set. *see: Uncertainty set* |
| $T$ | $[-]$ | Complementary sensitivity function. |
| $\Pi$ | $[-]$ | Uncertainty set. *see: Uncertainty set* |
| $r$ | $[-]$ | Radius of a disk-shaped approximation of a lumped perturbation region in the complex plane. |
| $\omega_c$ | $[-]$ | Cutoff frequency. |
| $w$ | $[-]$ | Weighing function. |

# List of Figures

# List of Tables

# List of Code Snippets

# Chapter 1

# Introduction

This chapter is based on the introductory chapter of ‘Assistive Shopping Trolley Control for the Elderly - A Literature Review’ by V. Heusdens [9].

The aging society puts an increasingly puts an increasingly heavy strain on our healthcare system. Many elderly have to move out of their own home and into a nursing home due to age-related problems, such as limited mobility, memory loss, deteriorating health and loneliness [8]. This experience is very unpleasant, as well as expensive. According to the Dutch health-care authorities (Nza) [10], elderly in the age of 65 years and older are responsible for roughly half of the cost of health-care, whilst only taking up around 20% of the population. The percentage that elderly take up of the total population is to rise to 25% in the next 20 years, thus it is to be expected that the cost of healthcare will rise accordingly. Of all the healthcare expenses generated by elderly, over 35% is due to the ‘wet langdurige zorg’ (wlz; long-term care act), which consists mainly of costs related to living in nursing homes. Assisting the elderly with robotic devices can extend the time elderly can spend at their own home, therefore reducing the costs of wlz as well as increase elderly satisfaction.

The main problems elderly face in their homes can be categorized into four subjects: limited mobility, memory loss, deteriorating health and loneliness [8]. The devices that are currently on the market mainly focus on limited mobility and memory loss, therefore there is a need for a solution regarding deteriorating health and loneliness. In her master thesis, R. Berci-Hajnovics proposes to tackle these problems through improving exercise and socialization by stimulating the elderly to run errands [8]. Running errands is a way for elderly to keep in touch with their friends whilst exercising by walking to the shop and back. Currently, many elderly like to take the trip to the store but have to take a rest regularly when carrying heavy groceries. Above all, they are often barely or not at all capable of carrying the groceries over obstacles like slopes, bumps, or up the stairs, which impairs them from running errands by themselves. This results in fewer trips to the grocery store, thus less contact with their friends. Therefore, loneliness can set in. In order to tackle this problem, it is proposed to design an assistive shopping trolley which reduces the physical burden of carrying groceries over uneven terrain. An example of a regular shopping trolley can be seen in fig. 1.1.

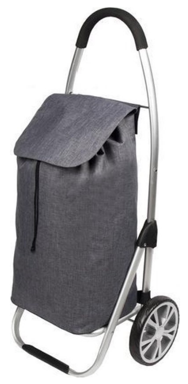


Figure 1.1: Shopping trolley example [11]

## 1.1 Objective

Following up on [8], this thesis will focus on the design of a control system for the assistive shopping trolley for elderly.

As described in the previous section, the main goal of the assistive shopping trolley is to decrease the required effort of pulling the assistive shopping trolley to a comfortable level for elderly users. To accomplish this it is not only important to provide the actual assist, it is also essential that the experience of using the trolley is pleasant for the elderly user; otherwise the trolley won't be used at all. Due to elderly generally having an above-average distrust of technology [8], emphasis will be placed on gaining acceptance.

The objective of gaining acceptance is approached by setting two requirements. Firstly it is required that the assistive trolley control system will be 'passive'; this means that the user needs to apply force on the trolley handle in order for the motor to be activated. Therefore the user is assured that the trolley will not take off on its own. Furthermore, it is desired to *attenuate the influence of changing system parameters and environmental variables*, from now on referred to as 'disturbances', on the trolley dynamics. An example of the attenuation of changing environmental variables is the controller compensating for effects of road inclination, effectively giving the sensation of the trolley being pulled along a flat road. An example of the attenuation of changing system parameters is the controller compensating for the change in its dynamics due to the addition of heavy groceries. By attenuating disturbances it is ensured that the trolley will always react the same to the user input, and therefore the predictability of the trolley will rise; The user will know that he/she will not get stuck during their trip because of changing conditions, as long as the trolley can be pulled comfortably in the 'normal' situation.

Concluding, the objective of this thesis is:

*Develop a control system for an assistive shopping trolley, that makes it feel as if it is empty and on a flat road while behaving predictably for the elderly.*

## 1.2 Thesis outline

Before the control system can be developed, a thorough dynamical model is derived in chapter 2. Next, to select the most appropriate control paradigm, an overview of candidate control methods is presented in chapter 3. Chapter 4 concerns the design of the controller using the chosen paradigm in detail. Subsequently, the implementation of the controller in software is described in chapter 5. In chapter 6, the controller's performance is tested in simulation and the results are analyzed. To conclude, chapter 7 summarizes the key points and insights presented in this thesis.

# Chapter 2

# Trolley Dynamics

In order to design a suitable controller for our trolley it is of vital importance to understand its dynamics and corresponding properties. For this purpose, a model of the trolley dynamics is derived. This model will later be used for several different applications; Firstly its properties are used to analyze possible suitable control types in chapter 3: ‘Control Method Overview’. Furthermore, in chapter 4: ‘Controller Design’ it is used as a basis for the internal model of the controller and to perform a stability analysis of the closed-loop response of trolley and controller in chapter 4: ‘Controller Design’. Finally it is used to verify the proper functioning of the simulation which is used to test the controller performance, described in chapter 6: ‘Simulation’.

This chapter is constructed as follows: In section 2.1 a model representing the relevant trolley dynamics is composed; section 2.1.1 describes the free-body-diagram of the trolley, along with a description of the relevant variables. Subsequently section 2.1.2 describes the accompanying dynamic equation. This equation is simplified in section 2.2 by making some important assumptions and observations regarding the trolley dynamics and the scope of the variables. As a conclusion, the most relevant model limitations are discussed in section 2.3. A summary of the most important elements considered in this chapter is provided in section 2.4.

## 2.1 Model description

The shopping trolley consists of two wheels and a frame with a bag attached at the axis between the two wheels, as can be seen in fig. 1.1. Additionally, two motors are fitted between the wheels and the trolley frame, and the user exerts a force on the trolley handle which is positioned at the top of the frame.

For modelling purposes, the trolley can be represented as an inverted pendulum on wheels. A schematic representation of this representation can be seen in fig. 2.1. We are interested in the forward motion, therefore we only look at a 2D representation of the trolley in its symmetric plane, which stands perpendicular to the wheel axes. Inverted pendulums are by themselves unstable systems and thus require additional forces to be stabilized[1]. In independent systems these stabilizing forces are often produced by applying a torque on the pivot point and/or accelerating the pivot point horizontally, therefore inducing an acceleration force. An example of such stabilization is the balancing of a Segway; Here the user induces an imbalance by leaning forwards or backwards, which is then compensated by an acceleration of the Segway in that direction.

With shopping trolleys this stabilization is done by the user, which applies a balancing force on the trolley handle. The applied force on the handle can be furthermore used to propell the trolley forward, hence the applied human force on the trolley handle serves multiple purposes. To distinguish between the force elements for each purpose the force on the trolley handle is decomposed into two components which align colinear and perpendicular to the road; the colinear component being used for propelling the trolley forward and the perpendicular component being used for balancing the trolley. An schematic representation of this decomposition can be seen in fig. 2.2.

[1]With the exception of the situation where the inverted pendulum is positioned in its equilibrium position.

### 2.1.1 Free Body Diagram

A schematic representation of the shopping trolley along with the forces acting on it can be made in the form of a Free Body Diagram (FBD). This FBD can be found in fig. 2.3. Points $W$ and $F$ indicate the trolley wheel and -frame centers of mass, respectively. Point $H$ indicates the trolley handle and point $N$ indicates the point of contact between the trolley wheel and the road surface.

The coordinate system is chosen such that its origin lies at wheel center $W$ and the x-axis is always aligned with the road at the point of contact between wheel and road $N$. This configuration is chosen to simplify the definition of forces, since many of them are colinear- or perpendicular to the trolley's varying direction of travel.

Additionally the following components are considered, with all forces specified to be in [N] and torques in [N m]: $F_H$ represents the force exerted by the human on the trolley handle, with $F_{HX}$ and $F_{HY}$ being its components along the x- and y axis. $F_{FG}$ and $F_{WG}$ represent the gravitational acceleration forces on points $F$ and $W$, which are the centers of mass of the wheels and frame, respectively. Here $F_{FG} = m_{FT}g$ and $F_{WG} = m_W g$, where $m_{FT}$ and $m_W$ represent the mass of wheel and frame-bag-combination in [kg] and $g$ is the gravitational acceleration constant in [m s$^{-2}$]. $F_{WX}$ and $F_{WY}$ represent the constraint forces keeping the wheels and frame together. $T_M$ represents the torque exerted on the wheels and frame by the motors. $F_{STAT}$ and $F_N$ represent the static friction and normal-force on the wheels, respectively. These can also be seen as constraint forces, keeping the wheel from slipping and moving through the road surface. $T_{ROLL}$ represents the rolling friction torque on the wheels, occurring mainly due to hysteresis caused by deformation of the tires and road surface [12]. Sliding- and rolling friction in the bearings is also taken into account in this parameter. It is defined as

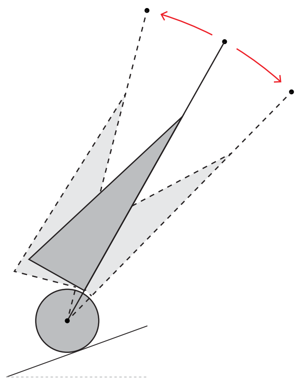


Figure 2.1: The trolley represented as an inverted pendulum.

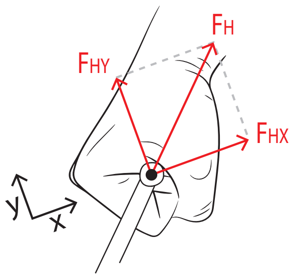


Figure 2.2: Force on the trolley handle split into components used for propulsion (x) and balancing (y).

$$T_{ROLL} = \mu F_N R_W \tag{2.1}$$

where $\mu$ is a dimensionless rolling friction coefficient and $R_W$ is the wheel radius.

Dynamic friction due to tire slip and air drag are neglected due to the low-velocity operating range of the trolley [13][14]. Other forces that aren't mentioned, for example viscous friction in the bearings, have a negligible size with respect to the dimensions of other forces at work and will therefore be ignored as well.

Additionally, distances $L_{WH}$, $L'_H$, $L_B$, $W_B$, $L_{WF}$ and $W_{WF}$ are indicated. $L_{WH}$ represents the distance between the wheel axis and the trolley handle, where $L'_H$ represents this distance measured perpendicular to the road (in y-direction). $L_B$ and $W_B$ represent the length and width of the trolley bag respectively, and $L_{WF}$ and $W_{WF}$ represent the distance between the wheel axis and the center of mass of the frame and bag combination, having respective directions colinear and perpendicular to the length of the frame. Distances $L_{WF}$ and $W_{WF}$ are estimated based on the trolley's configuration. An elaboration on this estimation can be found in appendix A. The frame width $W_F$ is not depicted to avoid cluttering of the image, although it is taken into account in the calculation of the center of mass position.

Coordinates $x_W$, $y_W$, $x_F$ and $y_F$ are introduced to indicate the $x$- and $y$-positions of the wheel- and trolley- centers of mass, respectively. It is desired to be able to indicate the trolley's velocity and acceleration with respect to its direction of movement rather than defined in the stationary world coordinate frame. The trolley's local coordinate frame with its origin in point $W$ rotates with the trolley's orientation and therefore its direction of movement, however since its origin lies in the wheel center all velocities and accelerations defined with respect to this frame will not include the movement of the trolley with respect to the world. To solve this problem, the center of mass velocities $\dot{x}_W$, $\dot{y}_W$, $\dot{x}_F$ and $\dot{y}_F$ and corresponding

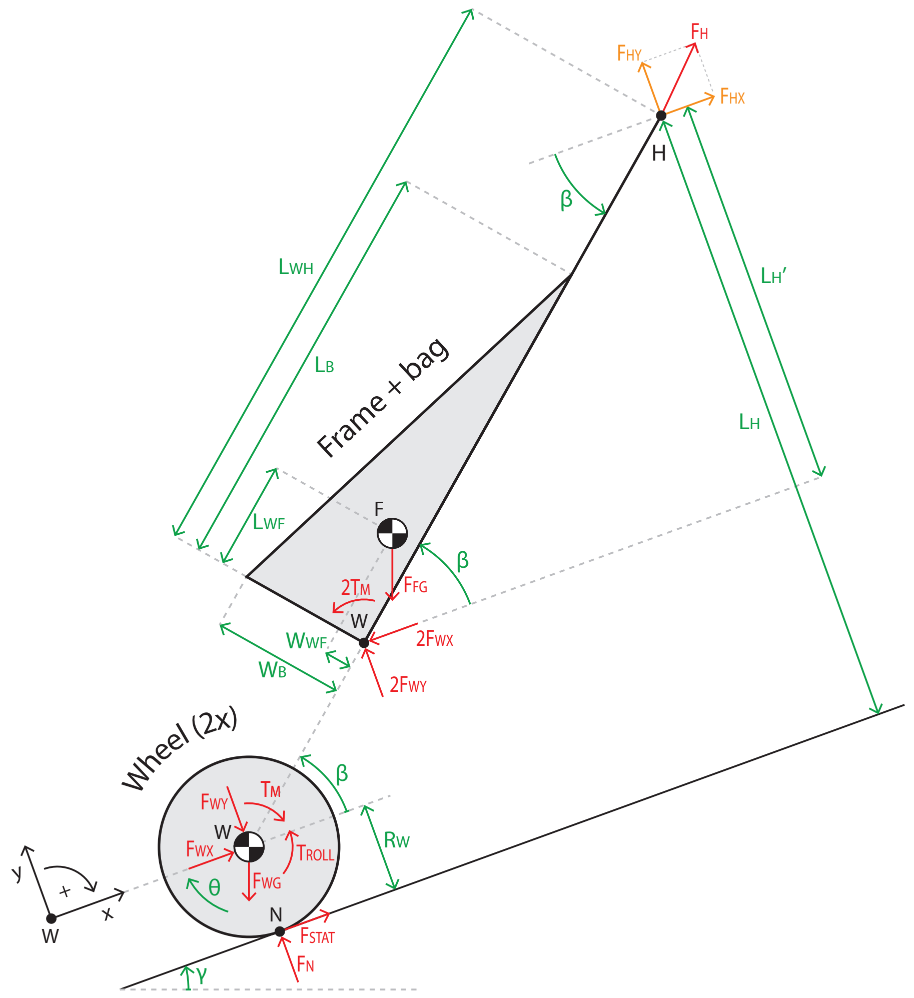


Figure 2.3: Free Body Diagram showing the forces and moments acting on the trolley.

Table 2.1: Overview of model elements and parameters used in fig. 2.3

| **Element** | **Description** |
|---:|:---|
| $W$, $F$ | Wheel and frame center of mass |
| $H$ | Handle |
| $N$ | Point of contact between wheel and road surface |
| $x_W$, $y_W$ | x- and y-position of wheel center of mass $W$ w.r.t. a stationary point $O$ in the global world frame [m] |
| $x_F$, $y_F$ | x- and y-position of frame center of mass $F$ w.r.t. a stationary point $O$ in the global world frame [m] |
| $\theta$ | Wheel angle [rad] |
| $\beta$ | Frame angle w.r.t. road [rad] |
| $\gamma$ | Road inclination angle [rad] |
| $L_B$, $W_B$ | Height and depth of bag [m] |
| $L_{WH}$, $W_F$ | Height and depth of frame ($W_F$ not depicted in fig. 2.3) [m] |
| $L_{WF}$, $W_{WF}$ | Distance between wheel axis and center of mass of combined frame and bag, measured along- and perpendicular to- the frame [m] |
| $L_H$, $L_{H'}$ | Distance in y-direction between road and handle, and between wheel axis and handle [m] |
| $R_W$ | Wheel radius [m] |
| $m_F$, $m_B$, $m_W$ | Mass of frame, bag and wheel [kg] |
| $m_{FT}$ | Combined mass of frame and bag [kg] |
| $F_H$, $F_{HX}$, $F_{HY}$ | Total, x-and y-component of the human force on the handle [N] |
| $F_{WX}$, $F_{WY}$ | Constraint forces between frame and wheels [N] |
| $F_{FG}$, $F_{WG}$ | Gravitational force on points $F$ and $W$ [N] |
| $F_{STAT}$ | Static friction acting on wheels [N] |
| $F_N$ | Normal force acting on wheels [N] |
| $T_M$ | Motor torque [N m] |
| $T_{ROLL}$ | Rolling resistance of wheels [N m] |
| $\mu$ | Rolling friction coefficient [−] |
| $g$ | Gravitational acceleration [m s$^{-2}$] |

accelerations are specified in the orientation of the trolley's local coordinate frame, but with respect to the location of the world frame.

Finally coordinates $\gamma$, $\theta$ and $\beta$ are introduced, which represent the inclination angle of the road, the wheel angle with respect to the initial position and the angle between the frame and the road, respectively. Angles $\gamma$and $\beta$ are defined with respect to the road orientation at the point of contact between the wheel and the road, $N$. All are defined to be in [rad]. An overview of all model parameters can be found in table 2.1.

### 2.1.2 Dynamic model equation

Using the FBD in fig. 2.3 as a reference, the Newton-Euler equations of motion and constraint equations are constructed to obtain a dynamic equation which describes the trolley's dynamic behavior. The coordinate vector is defined to be:

$$\mathbf{x} = \begin{pmatrix} x_W & y_W & \theta & x_F & y_F & \beta \end{pmatrix} \tag{2.2}$$

where $x_W$, $y_W$, $x_F$ and $y_F$ are the x- and y-positions of the wheel and frame center of mass, respectively, and $\theta$ and $\beta$ are the angle of the wheel with respect to its initial orientation and the angle of the frame with respect to the road at point $N$, respectively.

#### 2.1.2.1 Newton-Euler equations of motion

The Newton-Euler equations of motion describe the translational and rotational dynamics of the rigid bodies of the trolley. The trolley is subdivided into three rigid bodies, namely the frame and the two wheel. In 2D the equations of motion for every body consist of three components: A component in x-direction, one in y-direction and a component for rotational elements. Since both wheels are identical, there is no need to separately construct their equations of motion. Therefore the equations of motion consist of a total of 6 components:

*Wheel*

$$\sum F_x = 0: \qquad m_W \ddot{x}_W = F_{WX} + F_{STAT} - \sin(\gamma) F_{WG} \tag{2.3a}$$

$$\sum F_y = 0: \qquad m_W \ddot{y}_W = F_N - F_{WY} - \cos(\gamma) F_{WG} \tag{2.3b}$$

$$\sum M_{(W)} = 0: \qquad I_W \ddot{\theta} = T_M - T_{ROLL} - R_W F_{STAT} \tag{2.3c}$$

*Frame and bag*

$$\sum F_x = 0: \qquad m_{FT} \ddot{x}_F = F_{HX} - 2F_{WX} - \sin(\gamma) F_{FG} \tag{2.3d}$$

$$\sum F_y = 0: \qquad m_{FT} \ddot{y}_F = F_{HY} + 2F_{WY} - \cos(\gamma) F_{FG} \tag{2.3e}$$

$$\begin{aligned} \sum M_{(W)} = 0: \qquad I_F \ddot{\beta} = &-2L'_{WF} F_{WX} - 2W'_{WF} F_{WY} + 2T_M \\ &- \big(\sin(\beta) L_{WH} - L'_{WF}\big) F_{HX} \\ &+ \big(\cos(\beta) L_{WH} - W'_{WF}\big) F_{HY} \end{aligned} \tag{2.3f}$$

where

$$m_{FT} = m_F + m_B; \qquad I_W = \frac{1}{2} m_W R_W^2;$$

$$L'_{WF} = \sin(\beta) L_{WF} + \cos(\beta) W_{WF}; \qquad W'_{WF} = \cos(\beta) L_{WF} - \sin(\beta) W_{WF};$$

$$F_{FG} = m_{FT} g; \qquad F_{WG} = m_W g;$$

$$T_{ROLL} = \mu F_N R_W.$$

The mass moment of inertia of the trolley wheels, $I_W$, is taken to be that of a solid cylinder rotating around its axis. The forces applied by the human on the handle $F_{HX}$ and $F_{HY}$ and motor torque $T_M$ are inputs of the system.

#### 2.1.2.2 Constraint equations

The constraint equations describe the geometric constraints of the trolley. There are two points in the trolley FBD which restrict the motion of the rigid bodies: The rotational joint on point $W$, which connects the wheel to the frame and therefore couples their relative positions, and point $N$, which is the contact point between the wheels and the road and thus puts a restriction on the wheel movement in y-direction. The equivalent constraint equations on the rigid body centers of mass are:

*Wheel center of mass*

$$x_W = R_W \theta \tag{2.4a}$$

$$y_W = 0 \tag{2.4b}$$

*Frame center of mass*

$$x_F = x_W + \cos(\beta)L_{WF} - \sin(\beta)W_{WF} \tag{2.4c}$$

$$y_F = y_W + \sin(\beta)L_{WF} + \cos(\beta)W_{WF} \tag{2.4d}$$

The goal is to combine the equations of motion and constraint equations to obtain a single equation describing the trolley dynamics. Eqs. (2.3) include accelerations, where eqs. (2.4) describe the geometric constraints with respect to position. In order to combine these sets of equations, eqs. (2.4) need to be differentiated twice to obtain the constraint equations w.r.t. the center-of-mass accelerations:

*Wheel center of mass*

$$\ddot{x}_W = R_W \ddot{\theta} \tag{2.5a}$$

$$\ddot{y}_W = 0 \tag{2.5b}$$

*Frame center of mass*

$$\ddot{x}_F = \ddot{x}_W - L_{WF}(\sin(\beta)\ddot{\beta} + \cos(\beta)\dot{\beta}^2) + W_{WF}(\sin(\beta)\dot{\beta}^2 - \cos(\beta)\ddot{\beta}) \tag{2.5c}$$

$$\ddot{y}_F = \ddot{y}_W - L_{WF}(\sin(\beta)\dot{\beta}^2 - \cos(\beta)\ddot{\beta}) - W_{WF}(\sin(\beta)\ddot{\beta} + \cos(\beta)\dot{\beta}^2) \tag{2.5d}$$

#### 2.1.2.3 Combined dynamic equations

The equations of motion (2.3) and constraint equations with respect to the center-of-mass accelerations (2.5) can now be combined into a system of equations describing the trolley coordinate accelerations and joint constraint forces related to the geometric constraints ($F_{WX}$, $F_{WY}$, $F_{STAT}$, $F_N$):

$$\begin{pmatrix} \mathbf{F_{ext}} \\ \mathbf{a(x,\dot{x})} \end{pmatrix} = \begin{pmatrix} \mathbf{M} & \mathbf{A} \\ \mathbf{B} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{\ddot{x}} \\ \mathbf{F_c} \end{pmatrix} \tag{2.6}$$

with the vectors

$$\mathbf{\ddot{x}} = \begin{pmatrix} \ddot{x}_W & \ddot{y}_W & \ddot{\theta} & \ddot{x}_F & \ddot{y}_F & \ddot{\beta} \end{pmatrix}^T, \tag{2.6a}$$

$$\mathbf{F_c} = \begin{pmatrix} F_{WX} & F_{WY} & F_{STAT} & F_N \end{pmatrix}^T, \tag{2.6b}$$

$$\mathbf{F_{ext}} = \begin{pmatrix} -\sin(\gamma)F_{WG} \\ -\cos(\gamma)F_{WG} \\ T_M - T_{ROLL} \\ F_{HX} - \sin(\gamma)F_{FG} \\ F_{HY} - \cos(\gamma)F_{FG} \\ 2T_M - \big(\sin(\beta)L_{WH} - L'_{WF}\big)F_{HX} + \big(\cos(\beta)L_{WH} - W'_{WF}\big)F_{HY} \end{pmatrix}, \tag{2.6c}$$

$$\mathbf{a(x,\dot{x})} = \begin{pmatrix} 0 \\ 0 \\ \big(W_{WF}\sin(\beta) - L_{WF}\cos(\beta)\big)\dot{\beta}^2 \\ -\big(W_{WF}\cos(\beta) + L_{WF}\sin(\beta)\big)\dot{\beta}^2 \end{pmatrix} \tag{2.6d}$$

and matrices

$$\mathbf{M} = diag\big(m_W,\ m_W,\ I_W,\ m_{FT},\ m_{FT},\ I_F\big), \tag{2.6e}$$

$$\mathbf{A} = \begin{pmatrix} -1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & R_W & 0 \\ 2 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 \\ 2L'_{WF} & 2W'_{WF} & 0 & 0 \end{pmatrix}, \tag{2.6f}$$

$$\mathbf{B} = \begin{pmatrix} 1 & 0 & -R_W & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & (L_{WF}\sin(\beta) + W_{WF}\cos(\beta)) \\ 0 & -1 & 0 & 0 & 1 & (-L_{WF}\cos(\beta) + W_{WF}\sin(\beta)) \end{pmatrix} \tag{2.6g}$$

where

$m_{FT} = m_F + m_B;$ $\qquad I_W = \frac{1}{2}m_W R_W^2;$

$L'_{WF} = \sin(\beta)L_{WF} + \cos(\beta)W_{WF};$ $\qquad W'_{WF} = \cos(\beta)L_{WF} - \sin(\beta)W_{WF};$

$F_{FG} = m_{FT}g;$ $\qquad F_{WG} = m_W g;$

$T_{ROLL} = \mu F_N R_W.$

This system of equations is in essence just a compact representation of the combined equations of motion 2.3 and constraint equations 2.5, with the upper row of matrices and vectors containing the former and the bottom row containing the latter. Additionally, $T_{ROLL}$ is replaced by its representation $\mu F_N R_W$.

For the control of the trolley we are interested in the trolley's acceleration in x-direction: $\ddot{x}_W$. Therefore the system of eqs. 2.6 will be rewritten into a single equation with output $\ddot{x}_W$ via substitution of variables. Variables $L'_{WF}$, $W'_{WF}$, $I_W$ and $T_{ROLL}$ will be substituted by the functions they represent, as provided on the facing page. The final dynamic model equation has become:

$$\begin{aligned}
&\Big[I_F m_{FT} + 3I_F m_W + 0.5L_{WF}^2 m_{FT}^2 \mu \sin(2\beta) + 0.5L_{WF}^2 m_{FT}^2 \cos(2\beta) + 0.5L_{WF}^2 m_{FT}^2 \\
&\quad + 3L_{WF}^2 m_{FT} m_W + L_{WF} W_{WF} m_{FT}^2 \mu \cos(2\beta) - L_{WF} W_{WF} m_{FT}^2 \sin(2\beta) \\
&\quad - 0.5W_{WF}^2 m_{FT}^2 \mu \sin(2\beta) - 0.5W_{WF}^2 m_{FT}^2 \cos(2\beta) + 0.5W_{WF}^2 m_{FT}^2 + 3W_{WF}^2 m_{FT} m_W\Big] \ddot{x}_W \\
&= \Big[I_F + L_{WF}^2 m_{FT} + 0.5L_{WF} L_{WH} m_{FT} \mu \sin(2\beta) + 0.5L_{WF} L_{WH} m_{FT} \cos(2\beta) \\
&\quad - 0.5L_{WF} L_{WH} m_{FT} + 0.5L_{WH} W_{WF} m_{FT} \mu \cos(2\beta) - 0.5L_{WH} W_{WF} m_{FT} \mu \\
&\quad - 0.5L_{WH} W_{WF} m_{FT} \sin(2\beta) + W_{WF}^2 m_{FT}\Big] F_{HX} \\
&+ \Big[I_F \mu + L_{WF}^2 m_{FT} \mu - 0.5L_{WF} L_{WH} m_{FT} \mu \cos(2\beta) - 0.5L_{WF} L_{WH} m_{FT} \mu \\
&\quad + 0.5L_{WF} L_{WH} m_{FT} \sin(2\beta) + 0.5L_{WH} W_{WF} m_{FT} \mu \sin(2\beta) \\
&\quad + 0.5L_{WH} W_{WF} m_{FT} \cos(2\beta) + 0.5L_{WH} W_{WF} m_{FT} + W_{WF}^2 m_{FT} \mu\Big] F_{HY} \\
&+ \Big[2I_F R_W^{-1} + 2L_{WF}^2 m_{FT} R_W^{-1} - 2L_{WF} m_{FT} \mu \cos(\beta) + 2L_{WF} m_{FT} \sin(\beta) \\
&\quad + 2W_{WF} m_{FT} \mu \sin(\beta) + 2W_{WF} m_{FT} \cos(\beta) + 2W_{WF}^2 m_{FT} R_W^{-1}\Big] T_M \\
&+ \Big[-I_F \mu \cos(\gamma) - I_F \sin(\gamma) - 0.5L_{WF}^2 m_{FT} \mu \cos(\gamma) + 0.5L_{WF}^2 m_{FT} \mu \cos(2\beta + \gamma) \\
&\quad - 0.5L_{WF}^2 m_{FT} \sin(\gamma) - 0.5L_{WF}^2 m_{FT} \sin(2\beta + \gamma) - L_{WF} W_{WF} m_{FT} \mu \sin(2\beta + \gamma) \\
&\quad - L_{WF} W_{WF} m_{FT} \cos(2\beta + \gamma) - 0.5W_{WF}^2 m_{FT} \mu \cos(\gamma) - 0.5W_{WF}^2 m_{FT} \mu \cos(2\beta + \gamma) \\
&\quad - 0.5W_{WF}^2 m_{FT} \sin(\gamma) + 0.5W_{WF}^2 m_{FT} \sin(2\beta + \gamma)\Big] F_{FG} \\
&+ \Big[-2I_F \mu \cos(\gamma) - 2I_F \sin(\gamma) - 2L_{WF}^2 m_{FT} \mu \cos(\gamma) - 2L_{WF}^2 m_{FT} \sin(\gamma) \\
&\quad - 2W_{WF}^2 m_{FT} \mu \cos(\gamma) - 2W_{WF}^2 m_{FT} \sin(\gamma)\Big] F_{WG} \\
&+ \Big[I_F L_{WF} \mu \sin(\beta) + I_F L_{WF} \cos(\beta) + I_F W_{WF} \mu \cos(\beta) - I_F W_{WF} \sin(\beta) \\
&\quad + L_{WF}^3 m_{FT} \mu \sin(\beta) + L_{WF}^3 m_{FT} \cos(\beta) + L_{WF}^2 W_{WF} m_{FT} \mu \cos(\beta) \\
&\quad - L_{WF}^2 W_{WF} m_{FT} \sin(\beta) + L_{WF} W_{WF}^2 m_{FT} \mu \sin(\beta) + L_{WF} W_{WF}^2 m_{FT} \cos(\beta) \\
&\quad + W_{WF}^3 m_{FT} \mu \cos(\beta) - W_{WF}^3 m_{FT} \sin(\beta)\Big] m_{FT} \dot{\beta}^2
\end{aligned} \tag{2.7}$$

where

$$m_{FT} = m_F + m_B; \qquad F_{FG} = m_{FT} g; \qquad F_{WG} = m_W g.$$

### 2.1.3 Verification

The proper functioning of the model is analyzed by simulating the model response over time for several situations. All parameters are used as defined in fig. 2.3. Normally the trolley handle will be held up by the user, therefore there is no need to implement a constraint to simulate a collision between the trolley handle and the ground. In the current simulations however the force of the user is not taken into account, thus the trolley frame is allowed to rotate 360° around the wheel center without being stopped by the road.

In order to analyze the principal behavior of the inverted pendulum it is assumed that the center of mass of the trolley-frame and -bag are in line with the trolley frame ($W_{WF} = 0$) and there is no rolling friction ($\mu = 0$). All parameter values used for the simulations can be found in appendix B.

**Horizontal frame** Firstly, the models oscillatory behavior is analyzed by placing the trolley on a horizontal surface and specifying an initial condition where the trolley frame is horizontal ($\beta = 0$). Since the model's center of mass is specified to be along the centerline of the frame and there is no friction present, it can be expected that the motion will be undamped and symmetric. As can be seen from fig. 2.4 this is indeed the case.

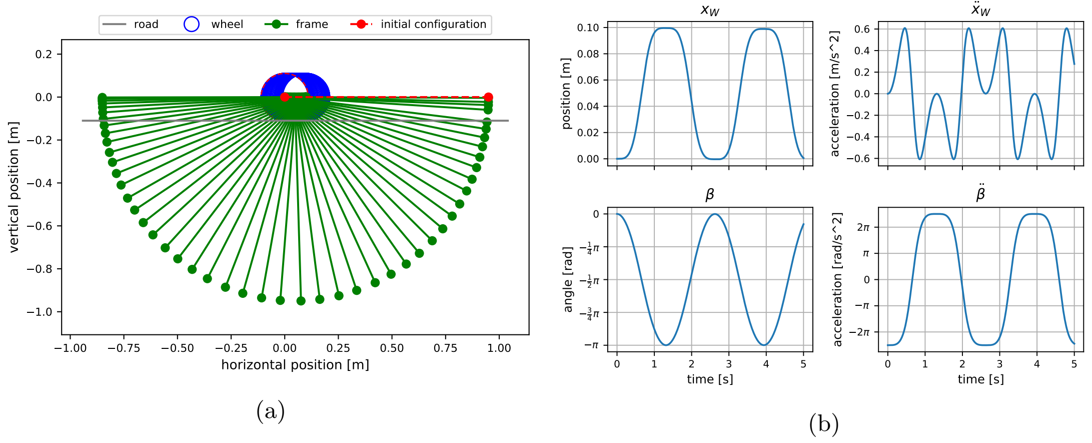


Figure 2.4: Dynamic model behavior for initial condition of $\beta = 0$.

**Inclined road** Secondly the model's behavior on an inclined road, which can be seen in fig. 2.5, is analyzed. It can be seen that the trolley is pulled downwards due to gravity, and the slope guides the trolley to the left as it goes down. Due to the frame's inertia the horizontal movement causes an oscillation in the frame around the wheel axis, which indicates that the coupling between horizontal and rotational dynamics is correctly implemented.

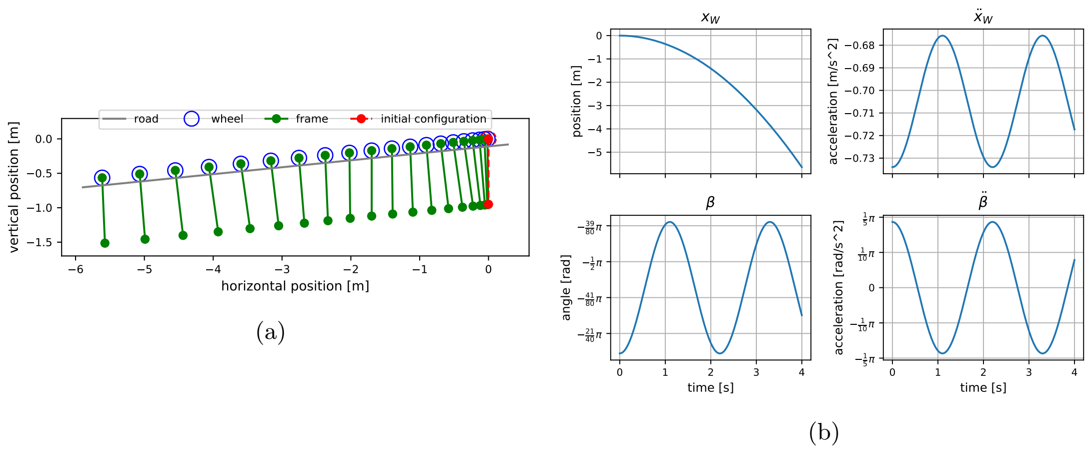
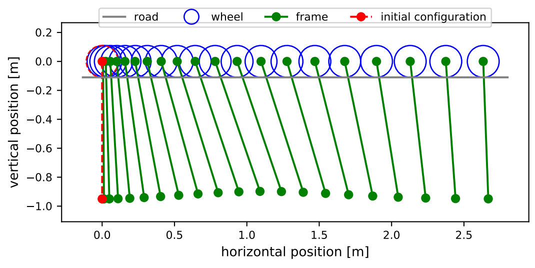
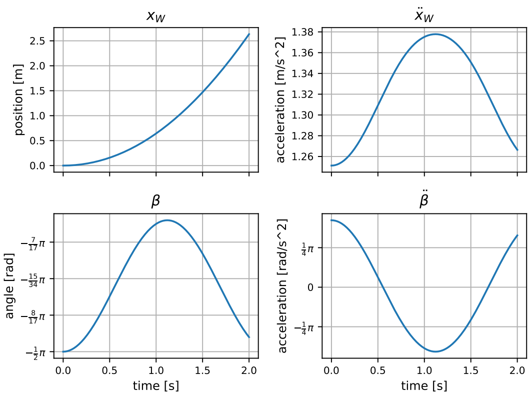


Figure 2.5: Dynamic model behavior for $\gamma > 0$.

**Motor torque** Lastly, the model behavior is analyzed when a constant positive motor torque of 1 N is applied. At first sight the model behavior, depicted in fig. 2.6, seems to be counterintuitive due to the acceleration of $\beta$ being in the same direction as the wheel acceleration. This is however to be expected, since the applied motor torque is exerted on both the wheel and the frame in opposite direction; in order to rotate the wheel clockwise a counter-clockwise torque needs to be exerted on the trolley frame as well. This counter-clockwise reaction torque pushes the trolley frame upwards in the same direction as the trolley's acceleration.



(a)



(b)

Figure 2.6: Dynamic model behavior for $T_m > 0$.

Following these analyses it can be concluded that the model likely behaves as expected in relevant scenarios and therefore it can be used for representation of the trolley.

## 2.2 Simplifying the model

Dynamic model equation 2.7 contains four free variables, namely output $\ddot{x}_W$ and trolley inputs $F_{HX}$, $F_{HY}$ and $T_M$; all other parameters are assumed to be known. A problem arises when the model is used to simulate the trolley being pulled by a human; now the forces on handle $H$ are nonzero. In this case $F_{HX}$ is specified, either as a deliberately set input in simulation or through measuring on the real trolley, as is motor torque $T_M$ since it is controlled by the controller, however balancing force $F_{HY}$ is not. Thus, whenever the forces on handle $H$ are not zero, not only $\ddot{x}_W$ but also $F_{HY}$ remains in the model equation as an unknown variable. The system is therefore underdetermined, and a value of $\ddot{x}_W$ can only be calculated using the model when $F_{HY}$ is known. Thus, some additional information is needed to obtain a value for $F_{HY}$. Furthermore, the current equation is nonlinear which requires the use of advanced nonlinear control methods to control. It is therefore desired to linearize it in order to simplify the controller design process.

### 2.2.1 Making the equation solvable

Dynamic model equation 2.7 can be made determined (solvable) by adding some additional information which makes one of the unknown variable dependent on other, known, variables in the system. This is achieved by adding an additional constraint, which establishes a relationship between vertical handle force $F_{HY}$ and other forces acting on the trolley: *The height of trolley handle $H$ perpendicular to the road surface will be set constant.* In order to keep $H$ on a constant height, the trolley must be balanced by $F_{HY}$. Therefore this constraint adds a relation between $F_{HY}$ and the remaining trolley dynamics, which can now be dropped as unknown variable from the dynamic equation.

The constraint is based on three assumptions. Firstly, the user's body is assumed to be always perpendicular to the road, as depicted in fig. 2.7. Justification for this assumption can be found in section C.1, where it is shown that on a slope with the maximum allowed inclination ratio as defined by Dutch law [15] of 1:10 (approx. 6°), this assumption changes the value of trolley frame angle beta less than 1 % with respect to its value when the human body orientation is vertical.

The second assumption states that the human keeps the trolley handle at point $H$ at a constant height with respect to his/her body. It is hypothesized that the human needs to lift the trolley handle continuously to keep the trolley upright. Therefore, it is assumed that the human arm is approximately stretched and oriented along the axis of the body during use of the trolley, as depicted by the dashed lines in fig. 2.7. This means that the distance between the feet of the user and point $H$ can be considered constant with respect to the orientation of the user's body.

Lastly it is assumed that walking over a piece of road where the road slope changes does not have an effect on frame angle $\beta$. This assumption is justified in section C.2, where it is shown that the maximum error in the estimated linear trolley acceleration caused by passing a change in road slope is around 0.21 %.

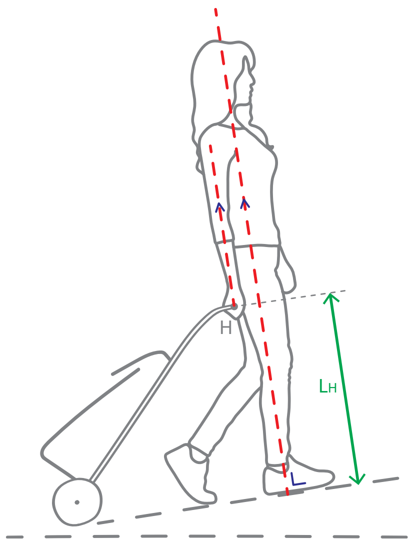


Figure 2.7: Assumed orientation of human and trolley with respect to the road.

Together, these three assumptions fix the trolley's orientation with respect to the road. Therefore they do not only eliminate $F_{HY}$ from the dynamic model equation, they also cause trolley frame angle $\beta$ to become constant and therefore $\dot{\beta}$ and $\ddot{\beta}$ to become zero. This can be seen more easily when writing $\beta$ in terms of $L_H$:

$$\beta = \sin^{-1}\left(\frac{L'_H}{L_{WH}}\right); \qquad L'_H = L_H - R_W.$$

Now both $L_{H'}$ and $L_{WH}$ are fixed, $\beta$ becomes fixed as well. This effect further simplifies the trolley's dynamic model equation, as will be shown in the next section.

### 2.2.2 Constrained dynamic equation

The constraint on trolley handle $H$ can be described mathematically using the parameters of the FBD in fig. 2.3:

$$y_W + L_{WH}\sin(\beta) = C \tag{2.8}$$

where $C$ denotes an arbitrary constant value. It can be added to the trolley's system of dynamic equations eqs. 2.6 by following the same procedure as described in section 2.1.2: 'Dynamic model equation' to obtain original dynamic model equation 2.7. First, eq. (2.8) needs to be written in terms of accelerations. Differentiating twice results in:

$$\ddot{y}_W - L_{WH}\big(\sin(\beta)\dot{\beta}^2 - \cos(\beta)\ddot{\beta}\big) = 0 \tag{2.9}$$

Equation (2.9) is consequently added to trolley dynamics system of eqs. 2.6 by appending it in an additional row to vector $\mathbf{a}(\mathbf{x}, \dot{\mathbf{x}})$ (2.6d) and matrix $\mathbf{B}$ (2.6g):

$$\mathbf{a}^*(\mathbf{x}, \dot{\mathbf{x}}) = \big(L_{WH}\sin(\beta)\dot{\beta}^2\big) \tag{2.6d*}$$

$$\mathbf{B}^* = \begin{pmatrix}0 & 1 & 0 & 0 & 0 & L_{WH}\cos(\beta)\end{pmatrix} \tag{2.6g*}$$

Rewriting the updated system into a single equation using substitution of variables results in:

$$\begin{aligned}
&\Big[\mu m_{FT}\big(L_{WF}\sin(\beta)+W_{WF}\cos(\beta)\big)+L_{WH}\cos(\beta)(m_{FT}+3m_W)\Big]\ddot{x}_W\\
&=\Big[L_{WH}\big(\mu\sin(\beta)+\cos(\beta)\big)\Big]F_{HX}+2\Big[L_{WH}\cos(\beta)R_W^{-1}-\mu\Big]T_M\\
&-2\Big[L_{WH}\big(\mu\cos(\gamma)+\sin(\gamma)\big)\cos(\beta)\Big]F_{WG}-2\Big[\Big(\mu L_{WF}\big(\sin(\beta)\sin(\beta+\gamma)-\cos(\gamma)\big)\\
&\qquad\quad+L_{WH}\cos(\beta)^2\big(\mu\cos(\gamma)+\sin(\gamma)\big)+\mu W_{WF}\big(\sin(\beta)\cos(\beta+\gamma)+\sin(\gamma)\big)\Big)\\
&\quad\Big(L_{WH}\cos(\beta)(m_{FT}+3m_W)+\mu m_{FT}\big(L_{WF}\sin(\beta)+W_{WF}\cos(\beta)\big)\Big)\\
&\quad\Big(L_{WF}m_{FT}\mu\sin(2\beta)+\big((m_{FT}+3m_W)L_{WH}+\mu m_{FT}W_{WF}\big)\big(\cos(2\beta)+1\big)\Big)^{-1}\Big]F_{FG}\\
&+2\Big[\Big(L_{WH}\cos(\beta)(m_{FT}+3m_W)+\mu m_{FT}\big(L_{WF}\sin(\beta)+W_{WF}\cos(\beta)\big)\Big)\\
&\quad\Big(I_F\mu\sin(\beta)+\mu m_{FT}\sin(\beta)(L_{WF}^2+W_{WF}^2)+m_{FT}L_{WH}\cos(\beta)(L_{WF}+\mu W_{WF})\Big)\\
&\quad\Big(L_{WF}m_{FT}\mu\sin(2\beta)+\big((m_{FT}+3m_W)L_{WH}+\mu m_{FT}W_{WF}\big)\big(\cos(2\beta)+1\big)\Big)^{-1}\Big]\dot{\beta}^2
\end{aligned}$$

where

$$m_{FT}=m_F+m_B;\qquad F_{FG}=m_{FT}g;\qquad F_{WG}=m_Wg;$$

$$\beta=\sin^{-1}\left(\frac{L'_H}{L_{WH}}\right);\qquad L'_H=L_H-R_W.$$

The assumptions described in the previous section cause trolley frame angle $\beta$ to be assumed constant, $\dot{\beta}$ becomes zero and therefore drops out of the equation. The resulting dynamic model equation thus becomes:

$$\boxed{\begin{aligned}
&\Big(\mu m_{FT}\big(L_{WF}\sin(\beta)+W_{WF}\cos(\beta)\big)+L_{WH}\cos(\beta)(m_{FT}+3m_W)\Big)\ddot{x}_W\\
&=\Big(L_{WH}\big(\mu\sin(\beta)+\cos(\beta)\big)\Big)F_{HX}+2\Big(L_{WH}\cos(\beta)R_W^{-1}-\mu\Big)T_M\\
&-2\Big(L_{WH}\big(\mu\cos(\gamma)+\sin(\gamma)\big)\cos(\beta)\Big)F_{WG}\\
&-2\Big(\Big(L_{WH}\cos(\beta)(m_{FT}+3m_W)+\mu m_{FT}\big(L_{WF}\sin(\beta)+W_{WF}\cos(\beta)\big)\Big)\\
&\quad\Big(\mu L_{WF}\big(\sin(\beta)\sin(\beta+\gamma)-\cos(\gamma)\big)+L_{WH}\cos(\beta)^2\big(\mu\cos(\gamma)+\sin(\gamma)\big)\\
&\qquad+\mu W_{WF}\big(\sin(\beta)\cos(\beta+\gamma)+\sin(\gamma)\big)\Big)\Big(L_{WF}m_{FT}\mu\sin(2\beta)\\
&\qquad+\big((m_{FT}+3m_W)L_{WH}+\mu m_{FT}W_{WF}\big)\big(\cos(2\beta)+1\big)\Big)^{-1}\Big)F_{FG} \qquad (2.10)
\end{aligned}}$$

where

$$m_{FT} = m_F + m_B; \qquad F_{FG} = m_{FT}g; \qquad F_{WG} = m_W g; \qquad \beta = \sin^{-1}\left(\frac{L_H - R_W}{L_{WH}}\right).$$

It can be seen that $F_{HY}$ is no longer explicitly present in the equation, and the system is now determined with only $\ddot{x}_W$ remaining as an unknown variable. Therefore, equation (2.10) can be used to obtain the value for $\ddot{x}_W$ when the trolley is being held by the user. Furthermore, since both $\beta$ as well as $\gamma$ are assumed constant, all nonlinearities present in the equation become constant, making eq. (2.10) linear.

### 2.2.3 Verification

Equation (2.10) can be verified by examining the equation and comparing it against our expectations by using some simple calculations.

In section 2.1 of the current chapter, eq. (2.10) is obtained by subsitution of variables. During this substitution, rolling resistance torque $T_{ROLL}$ is eliminated from the equation. In order to compare eq. (2.10) with our expectations, it is desirable to explicitly show the contributions of all forces in the equation. Therefore eq. (2.10) is rewritten into a form where $T_{ROLL}$ is explicitly present:

$$\left(m_{FT} + 3m_W\right)\ddot{x}_W = F_{HX} - (F_{FG} + 2F_{WG})\sin(\gamma) + \frac{2}{R_W}(T_M - T_{ROLL}) \tag{2.11}$$

Since it is assumed that the trolley's orientation with respect to the ground is constant and no damping-like behavior is taken into account in the dynamic model equation, its dynamics are expected to behave like a mass-system consisting of a non-rotating body and two rotating wheels. A simple, arbitrary mass-system complying to this description can be seen in fig. 2.8.

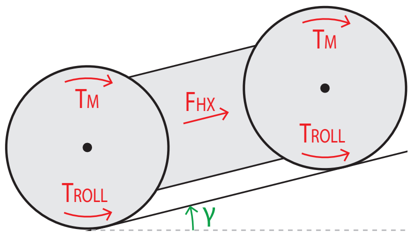


Figure 2.8: A mass system taken as a reference to validate the simplified dynamic trolley model eq. (2.10) obtained in section 2.2.2.

To verify the dynamic behavior of eq. (2.11), a dynamic equation for the system seen in fig. 2.8 can be constructed for comparison. Applying Newton's second law of motion, $F = ma$, to the mass-system of fig. 2.8 yields:

$$\begin{aligned}&(m_{FT} + 2m_W)\ddot{x}_W + 2\frac{I_W}{R_W}\ddot{\theta}\\&= F_{HX} - (F_{FG} + 2F_{WG})\sin(\gamma) + \frac{2}{R_W}(T_M - T_{ROLL})\end{aligned}$$

Rewriting the expression to linear coordinates, replacing $I_W$ by the inertial expression for a cylinder rotating around its axis $\frac{1}{2}m_W R_W^2$ and reordering the equation to group elements of forces results in:

$$(m_{FT} + 3m_W)\ddot{x}_W = F_{HX} - (F_{FG} + 2F_{WG})\sin(\gamma) + \frac{2}{R_W}(T_M - T_{ROLL}) \tag{2.12}$$

Comparing eq. (2.12) with eq. (2.11) it can be seen that they are identical, and therefore we can assume that the behavior of equation (2.10) is correct.

## 2.3 Model limitations

Due to the model of the trolley system being a simplification of reality, it will never perfectly represent the real trolley behavior. Whilst the model is designed so that these differences between model and reality will stay within reasonable bounds for considered situations, a deviation from these situations might significantly increase the model inaccuracies. It is important to be aware of these situational limitations, since a controller based on a highly inaccurate model can cause the controller performance to decrease substantially and may even cause instability. The most relevant model limitations are discussed.

**Model dimensions.** The model gives a 2D representation of the trolley system and thus can only give information about dynamics in that plane. Therefore, any controller or analysis based on this model will not take into account transverse dynamic behavior like effects of turning or of any force/torque applied transverse to the plane. Therefore, the model accuracy drops when such behavior occurs.

**Operating velocity.** The no-slip- and air-drag conditions are made under the assumption that the operating velocity of the trolley will remain low, as these forces will have negligible magnitude with respect to the other forces acting on the trolley system. This will no longer be the case when significant velocities are reached ($> 6\,\mathrm{m\,s^{-1}}$ [13]), and therefore the model's accuracy declines in such situation. This is however unlikely to happen under normal circumstances, since it is quite uncommon for elderly to travel at a speed of $20\,\mathrm{km\,h^{-1}}$ by foot.

Furthermore, in composing the model, rolling resistance torque $T_{ROLL}$ is assumed to be linear and pointing in negative direction. However, in reality $T_{ROLL}$ changes direction with the direction of movement and becomes zero when the wheel velocity is zero, which is nonlinear behavior. Therefore this assumption only holds under the condition that the trolley is continuously moving in positive x-direction.

**Vertical movement of trolley handle.** As mentioned in section 2.2, the perpendicular distance between trolley handle $H$ and the road surface is assumed to be constant whilst in reality it will likely vary to some degree. If this vertical movement becomes too big, the model will not represent reality sufficiently anymore. This will decrease the controller performance and the risk of the trolley becoming unstable might increase. Also, if trolley handle $H$'s distance perpendicular to the road changes excessively, the assumption made about a constant frame angle $\beta$ becomes doubtful as well.

**Road inclination.** As explained in section 2.2: 'Simplifying the model', it is assumed that trolley frame angle $\beta$ is constant, effectively stating that the user's body is always oriented perpendicular to the road. Although it is shown that for the maximum allowed ramp inclination this does not yield a significant change in $\beta$ with respect to a more realistic scenario where the user aligns itself with the gravitational vector, this can change for a situation where the inclination is significantly increased. Therefore, in such situations the controller performance cannot be guaranteed.

## 2.4 Summary

In this chapter, a dynamic model equation describing the trolley dynamics is formulated. The goal of this dynamic model is to provide a relation between the trolley inputs, namely the human force acting on the handle and the torque provided by the trolley's wheel motors, and its output, namely the trolley's linear acceleration in the direction of travel. This dynamic model equation will later be used in four different applications within this thesis: an analysis of suitable control types for use in the assistive trolley, the development of an inverse nominal model as part of the controller, a closed-loop stability analysis of the trolley-controller system and the verification of a simulation which is used to test the controller performance.

For the formulation of the trolley's dynamic model equation, firstly a schematic representation of the trolley dynamics in the form of a Free Body Diagram (FBD) is made. Here the trolley is represented by an inverted pendulum on wheels with a bag attached, residing on a sloped surface. The human force on the trolley handle is separated into components along- and perpendicular to- the slope's surface in order to distinguish between its elements responsible for balancing the trolley and pulling the trolley forward.

Using this FBD as a reference the Newton-Euler equations of motion and constraint equations, which together form the trolley's system of dynamic equations, are formulated. The system of dynamic equations is rewritten into the form of a single equation using substitution, with its output being the trolley's linear velocity in the direction of travel. The equation is verified by simulating its response over time for several elemental situations in which its principal dynamic behavior is shown.

The obtained dynamic model equation is initially underdetermined and therefore it needs to be modified to remove an excessive unknown variable, which in this instance is the component of the human force on the trolley handle perpendicular to the road surface. This is done by adding an additional constraint to the system of equations, which states that the vertical distance between the trolley handle and the road surface is stay constant.

This constraint is based on three assumptions. Firstly, the assumption is made that the orientation of the user's body is always perpendicular to the road, independently of the road slope. This assumption is justified by showing that the maximum error it induces in the trolley's frame orientation is less than 1 % with respect to a situation where the user's body is held vertically. Secondly, it is assumed that the trolley handle generally needs to be lifted by the user, and therefore the human arm will be approximately stretched along his/her body during use of the trolley. Lastly the effect of driving over a change in road slope angle is assumed to have a negligible effect on the change of the trolley frame orientation with respect to the road surface. This assumption is validated by calculation.

Adding the additional constraint to the trolleys system of dynamic equations results in a dynamic model equation which is determined and can therefore be used to represent the trolley behavior for use in the intended applications. Moreover the constraint makes the dynamic model equation linear, therefore simplifying the controller design. The newly obtained, determined model equation is again verified, this time by showing that it conforms with a simple dynamic model representing the expected behavior.

The model obtained in this chapter is a simplification of reality, and it will therefore only represent the real trolley behavior sufficiently within certain bounds. In the final part of this chapter some important corresponding limitations on the model and controller performance are discussed.

# Chapter 3

# Control Method Overview

In the previous chapter a dynamic model equation describing the trolley's dynamic behavior is formulated. In this chapter some of the properties of this dynamic model are used to determine a suitable control system for the assistive shopping trolley. A literature review was performed in order to determine a suitable power assist control method for use in the assistive shopping trolley[1]. In this literature review several control types found in literature are analyzed based on criteria relating to the objective formulated in section 1.1: 'Objective'. These control types are subsequently compared in this chapter using a Harris profile, from which the most suitable control type is chosen to use in the assistive shopping trolley.

This chapter is constructed as follows: In section 3.1: 'Focus' the focus area of the review is defined. In section 3.2: 'Comparative method' the method used for evaluating the found control types on their suitability to the task at hand is introduced, and the criteria used for this evaluation are explained. In section 3.3: 'General controller structure' the general structure between the found control systems will be explained. Subsequently, in sections 3.4.1–6 and 3.5, the different types of control will be discussed, along with the types of human input conversion that are encountered. Lastly, in section 3.6: 'Comparison', a comparison is made between control elements and a controller is proposed to be used in our assistive shopping trolley.

## 3.1 Focus

McRuer's crossover model states that humans have a preference for controlling systems which behave like a first order system [16]. As shown in section 2.2, the relationship between force and acceleration in our trolley can be approached by a regular (zeroth order) equation, and therefore the force-velocity relation is first order. Consequently it is assumed that the human will attempt to control the trolley's velocity.

Besides other human-held, two-wheeled power-assist trolleys the focus of the review lies on power-assist bicycles due to the similarities between riding a power-assist bicycle and pulling the power-assisted trolley; Both deal with a human force/torque that is directly applied to the plant (which is in this case the bicycle or trolley), and which is used for propulsion as well as a control input. Furthermore, both can be modeled as a first order system when looking at the relation between force/torque input and forward velocity [17]. Finally, similar factors influence the model variables as well as the effect of external elements on the model; both face large differences in weight (the bicycle due to different riders, the trolley due to different weight of groceries being carried) as well as obstacles like slopes and edges. In the remainder of this report, any general dynamic system will be referred to as 'plant'.

[1]This chapter builds on the previously conducted literature review [9]. The important aspects of this literature review are used and complemented with additional information and insights to form the overview found in this chapter.

## 3.2 Comparative method

After a qualitative analysis of the papers found within the defined focus area, six different control types are identified. As described in section 1.1: ‘Objective’, the two main design goals for our controller are to decrease the effort of pulling a trolley and to gain the acceptances of the elderly user. In order to determine which control type is best suited to achieve these objectives, a Harris profile is used; Composing a Harris profile is a useful technique for evaluating the relative performance of different designs with respect to important requirements [18]. All designs are rated on a scale of ’- -’ to ‘++’ for a set of criteria relevant to the design objectives. These scores are then expressed in a graphical manner for easy comparison.

From the design objectives, five different criteria are derived for the rating of the control types. An additional criterion is added to distinguish between controllers that can operate independently and the ones that need an additional controller to work properly. These criteria are discussed in detail below.

### Assist

The assist of a control system is a measure for the amount of help the controller provides given the human input. It therefore gives an indication of the effort that is needed to propell the controlled plant. An important aspect of this is the amount of lag that a controller induces; if the system reacts slowly to situational changes the user will experience a high deviation from the intended force profile in order to keep the desired velocity. Practically speaking this means that, in situations such as the user wanting to accelerate or the encountering of a slope, much of the additional force required to keep a constant velocity will have to come from the user him/herself. Thus, in these cases lag (temporarily) increases the effort of pulling the trolley, which is undesirable.

### Disturbance handling

Variable uncertainties within the plant as well as variable external influences acting on the plant are seen as disturbances. These disturbances have an influence on the system output, and therefore need to be attenuated in order to make the system behave as desired. Relevant examples of disturbances are additional weight caused by groceries and changes in road sloping. ‘Disturbance handling’ relates to the way the disturbances acting on the plant are being handled by the controller.

The disturbance handling methods are classified into three categories: ‘explicit’, ‘implicit’ and ‘non-existent’. Explicit disturbance handling methods are those in which the controller actively targets the disturbances by attenuating them in a feedback loop. For implicit disturbance handling the controller does not target disturbances explicitly, but it does react on other variables which are impacted by disturbances. Therefore the relation between disturbances and controller action for implicit methods is indirect. Controllers with non-existent disturbance handling do not react on disturbances at all.

Another (related) aspect of influence for the quality of disturbance handling is whether or not system dynamics are taken into account; considering dynamics is expected to improve the accuracy of the disturbance estimation since it provides information regarding dynamic effects which are not part of the disturbance.

### Passivity

This point relates to the objective of gaining the user’s acceptance, as described in section 1.1: ‘Objective’. A distinction can be made between ‘active’ and ‘passive’ controllers; passive controllers being controllers where human input is required to actuate the plant whereas active controllers do not need human input to operate. A passive controller is preferred since it removes the threat of the system acting ‘on its own’.

### Predictability

Another factor which is presumed to influence the user acceptance is the system's predictability. A high predictability is desired to increase the sense of trust in the trolley. Here once more system dynamics are of influence; taking these into account results in more natural system behavior, which is presumed to provide a more instinctive human interaction.

Additionally, the order of the system is assumed to play a role in its predictability. McRuer's crossover model states that humans prefer a system under their control to behave like a first order system [16]. Therefore it is presumed that a system which behaves differently will be less predictable. Since the human aims to control the trolley velocity (as described in section 3.1), we are interested in the order of the trolley's force-velocity input-output relation. As is shown in section 2.2: 'Simplifying the model' the dynamic relation between human input force $F_{HX}$ and position $\theta$ is second order, and therefore the trolley behaves like a first order system with respect to velocity. This means that, in order for the system to behave as desired, the addition of a controller should not change the system order. The system order will be evaluated by its transfer function, of which the derivations can be found in appendix D.

Note that, as discussed in section 1.1, predictability is enhanced by proper disturbance attenuation as well. This controller feature is however already discussed under 'disturbance handling'.

### Human input conversion

Another point of interest is human input conversion. This points to the way in which the conversion from physical force/torque exerted by the human on the system to electronic signal which can be used in the control system is made. Since this topic is not bound to any specific controller type it will be discussed in a separate section.

### Independence

This last criterion is not based on the design objective, but rather relates to the extent to which additional control needs to be implemented in order to make the controller work properly. This is of interest since it is an indication of the accuracy of the scores in this analysis; the influence of this additional control action on the controller performance cannot be determined from the literature used in this review, and therefore the performance scores for the system including these controllers become more uncertain.

## 3.3 General controller structure

After a qualitative analysis, 9 control systems are used for the comparison. Between all found control systems a common structure of user, controller and plant can be identified [9]. A schematic overview of this structure can be found in fig. 3.1.

In all situations the human input acts as a propelling entity on the plant as well as a control signal for the controller. Furthermore a controller is present, which exerts influence on the plant through the actuation of motors. The combination of controller and plant is called the 'system'. Other elements which have an unwanted effect on the plant dynamics are seen as disturbances. Important disturbances in the scope of this thesis are differences in road inclination and parameter uncertainties within the plant such as varying mass. The user him/herself implements feedback control on the system velocity as well; he/she can be seen as an enveloping controller which senses the trolley velocity through force-feedback on the handle and can change the input force on the system to reduce the deviation from the desired velocity. The combination of user and system thus includes dynamics caused by this feedback loop as well.

As explained in section 1.1, an objective of the controller is to minimize the influence of disturbances on the system so the human does not need to compensate for them manually. Therefore, in a system with a well-performing controller it is expected that the dynamics caused by the human feedback loop are of little influence to the dynamics of the structure as a whole. Therefore this human feedback loop is not taken into account when analyzing the found control systems, unless it is of vital importance to the controller's working principle.

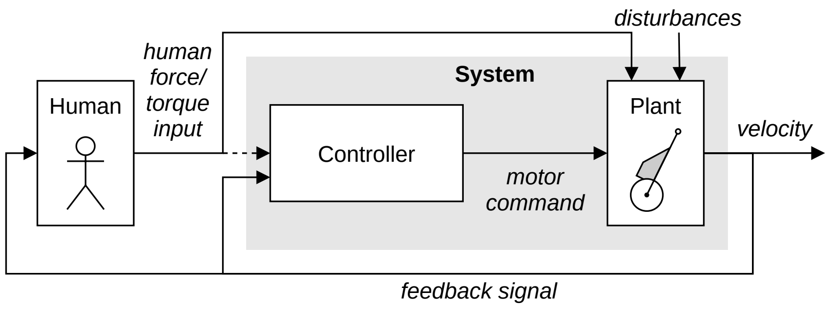


Figure 3.1: This schematic overview shows the common structure of all analyzed control systems. For the ‘Controller’ component, seven alternatives are evaluated in this chapter.

Some of the controllers under review output a reference velocity instead of a direct motor command. In these cases, an additional controller needs to be implemented to follow this reference velocity. The structure of this controller can be chosen independently from the controller that provides the reference velocity, and therefore no general assumptions can be made regarding its influence on the system dynamics. Consequently, its dynamics cannot be explicitly taken into account in this analysis. Such controller elements will be indicated as a block with a dashed line and the label ‘$C^*$’ in the corresponding schematic overview, and its contribution to transfer functions will be assumed to be of zeroth order.

## 3.4 Control types

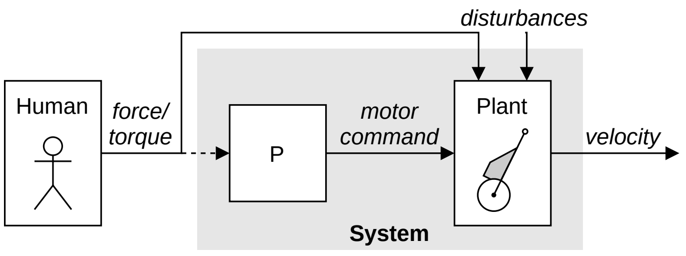
The found controller types are discussed below. The specific parts of the structure of user, controller and plant will from now on be referred to as labeled in fig. 3.1.

### 3.4.1 Proportional Control

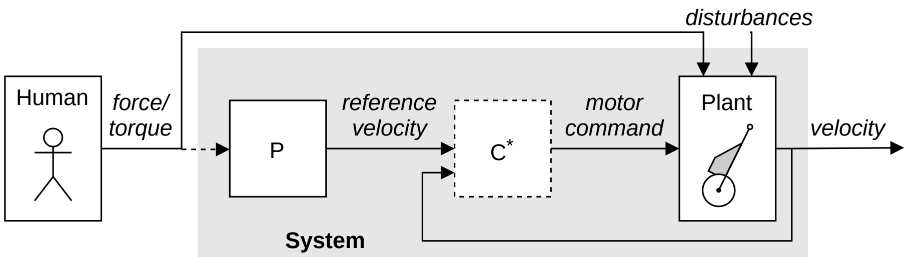
This type of control proportionally relates the human input with the system output [19], [20]. The proportional gain function can be chosen to fit the purpose of the controller.

A distinction can be made between proportional velocity control ([19]), where the human input is proportional to the reference velocity, and proportional torque control ([19], [20]), where the human input is proportional to the motor torque. In the case of proportional velocity control an additional control element needs to be implemented to make sure the plant follows the reference velocity generated by the proportional controller. The controller mentioned in [19] uses a PID controller for this, however this can be easily replaced by any other velocity controller without changing the function of the proportional element. Therefore, the dynamics of the velocity controller element will not explicitly be taken into account in the analysis.

A schematic representation of both controller types can be seen in fig. 3.2, where ‘P’ represents the proportional gain function which determines the relation between controller input and output, ‘Plant’ represents the plant dynamics and $C^*$ represents the arbitrary velocity controller.



(a) Proportional Torque controller structure. Here, P represents a proportional gain function.



(b) Proportional Velocity controller structure. Here, ‘P’ represents a proportional gain function and ‘C$^*$’ represents an arbitrary velocity controller.

Figure 3.2: Schematic overview of the Proportional controller structures.

### Assist

This controller type is zeroth order and therefore imposes virtually no lag, which allows such a system to react quickly to given inputs. Therefore, it will likely be able to assist well during situational changes. Because of this the proportional control type is rated ‘++’on assist.

### Disturbance Handling

With proportional control, the relation between controller -input and -output is determined by an arbitrary function which does not take system dynamics into account. Therefore it has no information regarding disturbances on the plant dynamics and cannot act on them explicitly. The disturbances will have virtually no effect on the torque delivered by the motors, and therefore the proportional torque controller will not react on them at all. The disturbance handling in this type of controller is thus ‘non-existent’. This yields a disturbance handling rating of ’- -’ for the proportional torque controller.

The disturbances however do have an influence on the plant velocity. In the case of proportional velocity control the sequential controller will compensate for this velocity deviation, and thus the controller will react on the disturbances. Therefore we can state that a proportional velocity controller ‘implicitly’ handles the disturbances. Because of this, proportional velocity control will be rated ‘+’ on disturbance handling.

### Passivity

Due to the controller scaling the current input signal given by the human, the output signal of the proportional part of the controller will become zero when no input is applied. In case of a proportional torque controller, this means that there will be no torque on the motors and the system therefore will be strictly passive. It will therefore be rated ‘++’ on passivity.

For proportional velocity control this means that the reference velocity for the sequential controller will become zero, but the motor torque will only become zero when a the velocity error becomes zero. This means that the controller will continue to actuate the motors until the trolley comes to a standstill. Theoretically this controller type is therefore not completely passive. This problem can however easily be overcome by implementing an additional rule that the motors can only provide a torque whenever an input force is present. Therefore, proportional velocity control is rated ‘+’ with respect to passivity.

**Predictability**

The general transfer functions of the systems using proportional-torque and proportional-velocity control can be respectively written as:

$$H = \frac{AC}{s} \tag{3.1}$$

and

$$H = \frac{AC^*C}{s + AC^*} \tag{3.2}$$

where $H$ is the system's transfer function from human input force/torque to trolley velocity, $A$ is a term containing all constants relating to the trolley dynamics transfer function, $C$ represents the proportional-control transfer function ($P$ in fig. 3.2) and $C^*$ is the transfer functions of the additional velocity controller (fig. 3.2b), respectively. Their derivations and block diagrams can be found in sections D.2 and D.3.

A proportional controller does not include an integrator or differentiator, and is therefore zeroth order. Since $A$ and $C^*$ are considered to be zeroth order as well, from eqs. (3.1) and (3.2) it can be derived that transfer function H for both proportional-torque control and proportional-velocity control are first order. It can therefore be argued that our system using proportional control will likely be predictable to elderly. This type of control however does not take into account the plant dynamics, which can make its behavior deviate slightly from the behavior of an uncontrolled trolley. This might be experienced as unexpected behavior by the elderly user. Therefore, both proportional controllers are rated '+' on predictability.

**Independence**

The output of proportional torque control directly commands the motor torque, and therefore no additional control is needed to make the system work properly. Therefore it scores '++' on independence. This does not hold up for proportional velocity control; this control type produces a reference velocity which must be kept by an additional controller element ($C^*$). Without this additional controller element the proportional velocity controller would not work. Therefore it is rated '-' on independence.

## 3.4.2 Integral Velocity Control

This type of controller relates human input and reference velocity by using an integrator; the human input can be considered to be the desired system acceleration [19]. In order for the controller to keep this reference velocity, an additional controller element is needed. [19] Mentions the use of a PID controller for this, however this can be easily replaced by any other velocity controller without changing the function of the integral element. Therefore, the dynamics of the velocity controller element will not explicitly be taken into account in the analysis.

A schematic representation of such structure can be found in fig. 3.3, where 'I' represents the integral controller consisting of an integrator and an integrator gain, 'Plant' represents the plant dynamics and '$C^*$' represents the arbitrary velocity controller.

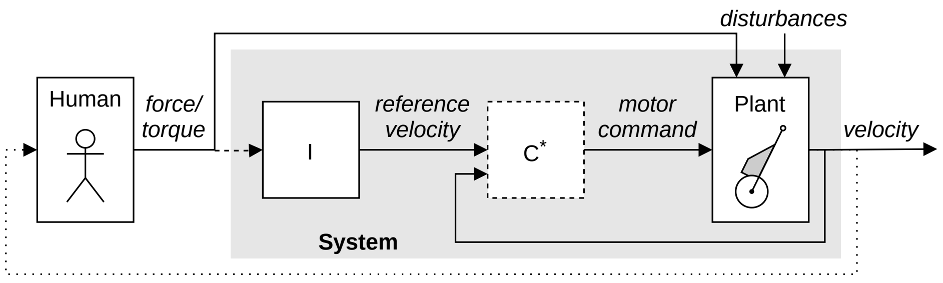


Figure 3.3: Schematic overview of the Integral Velocity controller structure. Here, 'I' represents an integral gain function and '$C^*$' represents an arbitrary velocity controller.

### Assist

An integral controller uses the accumulated error signal to determine its output signal. The accumulation of this error takes time, and therefore the controller reacts relatively slow. Because of this, such a controller might react sluggish to the user's input and might not be fast enough to assist significantly during situational changes. Therefore, this controller type will be rated '-' on assist.

### Disturbance Handling

An integral velocity controller uses the integral of the human input force to determine the reference velocity. System dynamics are therefore not taken into account in this relation. A sequential controller is implemented to follow this reference velocity. It therefore is this sequential controller which compensates for disturbances, since these disturbances cause the plant velocity to deviate from the reference velocity. Since this controller reacts on the velocity error caused by disturbances and not on the disturbances themselves, this controller type can be stated to be 'implicit'. Therefore integral velocity control is rated '+' on disturbance handling.

### Passivity

The reviewed controller in [19] takes the controller output to be the reference velocity, which makes that the human input can be seen as the desired acceleration of the system. Therefore, when the desired velocity is reached, the desired acceleration will become zero and the system will continue at a constant speed. Since it now operates with a human input of 0, the system using this controller type can considered to be 'active'. This also means that the user actively needs to slow the trolley down in order for it to stop. Consequently this controller will be rated '-' on passivity.

### Predictability

The general transfer function of the system using integral control can be written as:

$$H = \frac{AC^*C}{s + AC^*} \tag{3.3}$$

where $H$ is the system's transfer function from human input force/torque to trolley velocity, $A$ contains all constants relating to the trolley dynamics transfer function, $C$ represents the proportional-control transfer function ($I$ in fig. 3.3) and $C^*$ is the transfer functions of the additional velocity controller (fig. 3.3). its derivation and block diagram can be found in section D.3.

A basic integral controller is a first order system. Together with $A$ and $C^*$ both being zeroth order, it can be derived from eq. (3.3) that transfer function $H$ becomes second order; one order higher than the preferred open-loop system for humans to control. Therefore the system using this controller type is expected to be more difficult to predict than with a zeroth order controller. Furthermore, no plant dynamics are taken into account in this control type, which is expected to make the system even more unpredictable. Overall, this control type is rated '-' on predictability.

### Independence

The output of the integral velocity controller is a reference velocity and not a motor command. Thus, an additional controller elements ($C^*$) is needed to make this conversion. Therefore the integral velocity controller is scored '-' on independence.

### 3.4.3 Disturbance Observer Control

An observer is a dynamical system which computes unmeasured states using a mathematical model and other measurements [14] [21] [20] [22]. One specific type of observer is the Disturbance Observer (DOB). This type of observer estimates the effect of disturbances on the plant by comparing the measured plant input with an estimation based on output measurements and an inverse nominal plant model. A nominal plant model is a model of the plant where none of the disturbances and uncertainties are taken into account, and therefore represents the desired situation. This error can be used in a feedback loop to counteract the effects of the disturbances, making the real system behave like the nominal model.

Besides an inverse nominal model, a disturbance observer generally also includes a Q-filter which filters the relevant info from the estimated disturbance signal. Moreover a Q-filter is necessary to make the system 'proper', which means that it is feasible, by compensating for the derivative terms introduced in the inverse model. In all considered papers concerning DOB's the implemented Q-filter is a low-pass filter; this type of filter will let the disturbances of interest pass whilst removing high-frequency noise.

A schematic overview of a disturbance observer can be found in fig. 3.4, where 'Plant' represents the plant dynamics, 'Inverse model' represents the inverse nominal model and 'Q-filter' represents the above mentioned Q-filter.

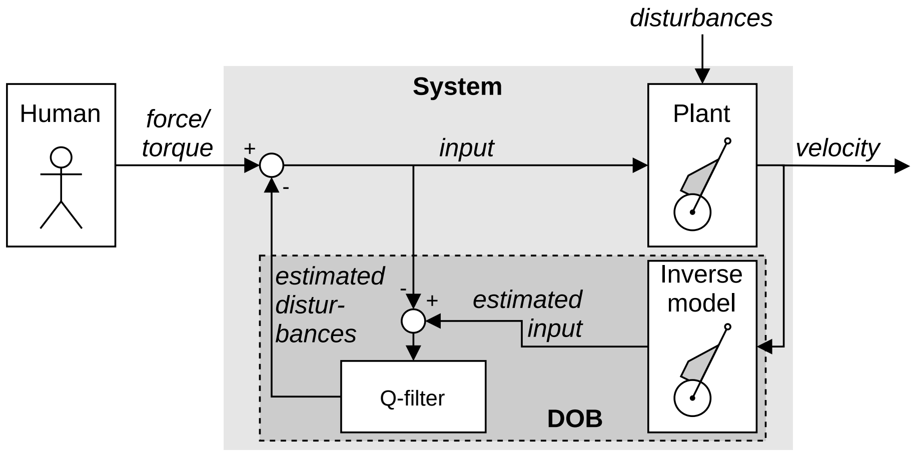


Figure 3.4: Schematic overview of the Disturbance Observer (DOB) controller structure.

#### Assist

A disturbance observer aims to make the system behave like the (inverse) model that is implemented in the controller, given the applied human input. The amount of assist therefore depends on the perceived disturbances acting on the plant. Since a dynamic model is used for the estimation of these disturbances, this controller takes into account model dynamics like inertia and damping. The Q-filter which is part of the DOB imposes a lag on the signal, which might partially compromise the amount of assist that can be given during situational changes. Therefore, this control type will be rated '+' on assist.

### Disturbance Handling

DOB controllers estimate the disturbances acting on the plant and use this estimation to attenuate them. It does so by using an (inverse) model, thus system dynamics are taken into account in the estimation. Because the controller reacts only on (estimated) disturbances, it can be labeled 'explicit' and this control type is rated '++' on disturbance handling.

### Passivity

The controller's goal is to mimic the behavior of the nominal plant model. This means that when no input force-or torque is applied to the plant, the controller can keep actuating the motors until the nominal plant model would have come to a standstill. Therefore, this controller will not be completely passive. As mentioned in section 3.4.1 this problem can be easily solved by only allowing the motors to be actuated when human input is present. Because of this, the DOB control type will be rated '+' on passivity.

### Predictability

DOB control presents an 'intelligent' type of assist; the use of a dynamic model in the controller allows us to take into account dynamics factors such as damping and inertia when determining the desired amount of assist. Therefore it is argued that a DOB controlled system will react more naturally to given human inputs than control types that do not take plant dynamics into account.

The general transfer function of the system using DOB control can be written as:

$$H = \frac{AB}{s\left((A-B)Q+B\right)} \tag{3.4}$$

where $H$ is the system's transfer function from human input force/torque to plant velocity, $A$ and $B$ are terms containing all constants relating to the trolley dynamics and controller model transfer function respectively and $Q$ is the transfer functions of the Q-filter (fig. 3.4). its derivation and block diagram can be found in section D.4.

A disadvantage of using the inverse of the model, like done in DOB control, is the need for an additional Q-filter with integrator to make the system proper. Since both $A$ and $B$ are constants, from eq. (3.4) it can be derived that the integrator in Q makes transfer function $H$ (at least) second order; which is higher than preferred for a human to control. Because of this, the predictability of DOB control is rated '+'.

### Independence

DOB control aims to estimate the disturbance force or torque acting on the plant, and compensate for it using motor torque. Because our trolley will take force as a controller input and therefore the disturbance is also given as a force, this force needs to be mapped to a torque value before it can be used as a motor command. This mapping is not inherent to DOB control and it can therefore be argued that an additional controller element is needed to perform this action. This mapping is however very simple, since the relation between force and torque is well known:

$$T = Fr \tag{3.5}$$

where $T$ is the motor torque command, $F$ is the estimated disturbance force and r is the radius of the wheel on which the motor delivers its torque. This relation is zeroth order and therefore it can be safely assumed that the implementation of this mapping will not have a large influence on the performance of the controller as a whole. Hence, DOB control is rated '+' on independence.

### 3.4.4 Impedance Control

*The control structure presented in this section is based on a paper which was disqualified from use in the preliminary literature review due to discrepancies in the proposed controller structure and nondescriptive experimental results*[2]*. Nevertheless, with some minor adjustments proposed in the literature review, the paper's proposed controller could still be used to serve as a base for the general control structure provided in the current section.*

Impedance control predicts the output of a virtual model with desired parameter settings and uses this as a reference for the real system [23]. This way, the real system is made to behave similar to the virtual model and thus its perceived properties can be altered by adjusting the virtual model parameters. This approach is similar of that of a DOB controller; but instead of using a model in the feedback loop to determine the error it is now used to determine a reference in the open loop which is kept by an additional velocity controller.

In order for the system to keep this reference velocity, an additional controller element needs to be used. In the controller proposed by [23] a PI controller is used; however this can be easily replaced by any other velocity controller without changing the function of the impedance element. Therefore, the dynamics of the velocity controller element will not explicitly be taken into account in the analysis.

A schematic representation of an impedance controller structure can be seen in fig. 3.5, where 'Model' represents the virtual model, 'Plant' represents the plant dynamics and 'C$^*$' represents the arbitrary velocity controller.

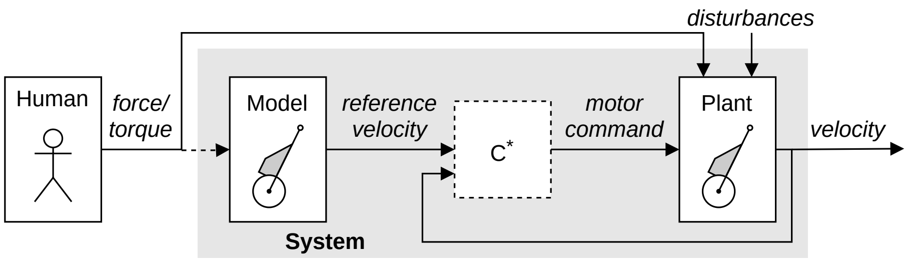


Figure 3.5: Schematic overview of the Impedance controller structure. Here, 'C$^*$' represents an arbitrary velocity controller.

#### Assist

The amount of assist this type of control provides is similar as that of a DOB (section 3.4.3), since both are determined using a plant model. Hence, this controller also takes into account model dynamics. An essential difference however is that in impedance control the used model is not inverse and implemented in the open-loop part of the controller. Since the model is first order, it imposes a lag on the system which might inhibit the controller from providing sufficient assist during situational changes. On the other hand, unlike for an inverse model, there is no need for an additional lag-inducing filter to make the model proper. Therefore, the impedance controller scores '+' on assist.

[2] [9], section 3.1.5.

#### Disturbance Handling

In impedance control the reference velocity for the system is determined using a dynamic model, which increases the accuracy with respect to the desired velocity. However, since the following of this reference velocity is taken care of by a sequential controller which does not have any knowledge of these system dynamics, the disturbance handling is still identified as being 'implicit'. Even though, because the system dynamics are taken into account, it is expected that the difference with explicit disturbance handling will be small. Therefore, this control type is rated '++' on disturbance handling.

#### Passivity

As for DOB control (section 3.4.3), since the controller aims to follow the model's behavior, when no human force input is given the controller can keep actuating the motors until the model would have come to a standstill. Therefore this controller is not completely passive. This problem can however be easily solved by only allowing the motors to be actuated when a human input force is present. Impedance control is consequently rated '+' on passivity.

#### Predictability

A model of the plant, which contains information regarding plant dynamics, is used to obtain the reference velocity. Therefore dynamic properties are taken into account in determining it, which is argued to improve predictability for elderly. Since a forward model is used, there is no need for an additional integrating filter. On the other hand, adding the first order model to the system this way will cause the model itself to increase the system order by one.

The general transfer function of the system using impedance control can be written as:

$$H = \frac{AC^*C}{s + AC^*} \tag{3.6}$$

where $H$ is the system's transfer function from human input force/torque to plant velocity, $A$ is a term containing all constants relating to the trolley dynamics transfer function, $C$ represents the model transfer function and $C^*$ is the transfer functions of the additional velocity controller (fig. 3.5). its derivation and block diagram can be found in section D.3.

From eq. (3.6) it can be derived that including the first order trolley model in $C$ will result in an transfer function $H$ which is second order. This is an order higher than the preferred system controlled by the human, and is therefore expected to lower predictability. Because of this, impedance control is overall rated '+' on predictability.

#### Independence

The impedance controller aims to provide a reference velocity for the plant to follow. An additional controller element needs to be implemented in order to follow this reference velocity, otherwise the impedance controller would be useless. Therefore, impedance control is scored '-' on independence.

### 3.4.5 Cruise Control

In the case of cruise control a reference velocity for an arbitrary velocity controller is set using a control panel [20]. Although the human force/torque input on the plant is not taken directly as an input of the cruise controller, they both have an influence on the plant velocity. Therefore, the cruise controller supplements the human input force/torque such that this reference velocity is reached. The cruise controller described in can be separated into two parts; one part which sets the reference velocity and another which makes the plant keep to this velocity. The reviewed controller in [20] uses a PI controller for velocity keeping, however this can be easily replaced by any other velocity controller without changing the function of the element which sets the reference velocity. Therefore, the dynamics of the velocity controller element will not explicitly be taken into account in the analysis.

A schematic representation of an impedance controller structure can be seen in fig. 3.5, where 'Control panel' represents the interface on which a reference velocity can be set, 'Plant' represents the plant dynamics and 'C$^*$' represents the arbitrary velocity controller.

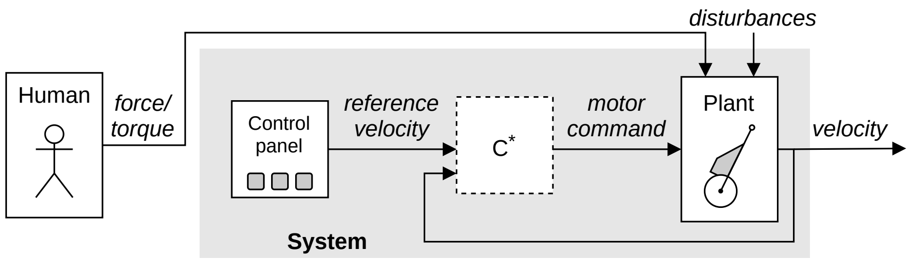


Figure 3.6: Schematic overview of the Cruise controller structure. Here, 'C$^*$' represents an arbitrary velocity controller.

### Assist

The amount of assist for a cruise controller fluctuates in such a way that the motor will compensate for any deviation from the reference speed. The cruise control structure itself does not impose any lag; Therefore, this controller type scores '++' on assist.

### Disturbance Handling

As for proportional velocity control (section 3.4.1), integral velocity control (section 3.4.2) and impedance control (fig. 3.5), cruise control makes use of a sequential controller for the following of the reference velocity. It therefore does not act explicitly on disturbances but they are of influence on the velocity error. Cruise control is therefore identified as practicing 'implicit' disturbance handling. Furthermore no dynamic model is implemented for the consideration of dynamic effects in the trolley behavior, resulting in an overall disturbance handling rating of '+'.

### Passivity

Since the reference velocity in a cruise-controller is set using a control panel and thus is not dependent on the human input/force torque, a cruise controller does not need any human force/torque input in order to actuate the plant. Therefore the system can considered to be 'active'. Because changing the reference velocity requires deliberate interaction with the control panel, it is relatively difficult to stop the controller from actuating the plant. This is especially the case for unexpected situations where an intuitive reaction from the user is to be expected. This results in a passivity rating of '- -'.

### Predictability

The general transfer function of the system using cruise control can be written as:

$$H = \frac{AC^*}{s + AC^*} \tag{3.7}$$

where $H$ is the transfer function from the reference velocity set using the control panel to the plant velocity, $A$ is a term containing all constants relating to the trolley dynamics transfer function and $C^*$ is the transfer functions of the additional velocity controller (fig. 3.6). its derivation and block diagram can be found in section D.5.

From eq. (3.7) it can be seen that the cruise controller itself does not impose any additional dynamics on the system transfer function. It is therefore expected not to change the system predictability.

Since no plant model is used in determining the reference velocity, no dynamic effects will be taken into account. This is expected to lower the predictability. Another important aspect to consider is that the reference velocity of this controller type needs to be set using a control interface. The use of such an interface is often not intuitive for elderly, which can lower the predictability significantly. In consequence, cruise control is rated '-' on predictability.

**Independence**

The function of cruise control is to set a fixed reference velocity and keep to this velocity. The velocity-control part of this control type is arbitrary, and therefore this is not taken into account in the analysis. Thus, only the setting of the reference velocity is considered. Since this part of the controller does need a velocity controller to work properly it can be seen as dependent and will therefore be rated '-' on independence.

### 3.4.6 Fuzzy Control

From [9]: "Fuzzy logic provides a nonlinear mapping of a set of input data to scalar output data by making use of fuzzy sets, that is, classes with a continuum of grades of memberships. These fuzzy sets stand in contrast to 'crisp' variables, which are either 100% true or 100% false. For example, an arbitrary colour is made up of different percentages of primary colours, or a car can be made of parts produced in different countries. These fractions can be represented in fuzzy logic by the variable's grade of membership to different classes, where in crisp set theory this is not possible.

A schematic overview of a system using fuzzy control can be found in fig. 3.7, where 'Fuzzy' represents the fuzzy controller and 'Plant' represents the plant dynamics. This control type is used by [24] to implement several types of power assist. It is chosen to use fuzzy control because the authors state that it is difficult to obtain a plant model (which in their case is a bicycle) and fuzzy control does not require a model to operate. However, from several other papers it can be concluded that obtaining a plant model is perfectly feasible and therefore this type of control is not being considered as a candidate for our trolley control system.

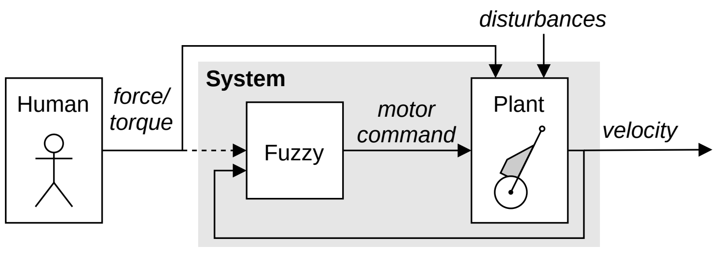


Figure 3.7: Schematic overview of the Fuzzy controller structure.

## 3.5 Human Input Conversion

Human input conversion points to the way that the physical human input is transformed to electronic input for the controller to use. Two types of human input conversion are found in literature; measurement ([19], [21], [14], [24]) and estimation ([20], [22]). Measuring the human input is done using force or torque sensors, depending on the type of human input. Human input estimation is done using a structure similar to a disturbance observer, only now the human input itself is estimated instead of the disturbance. This estimated human input can then be used as the input for additional controller elements. A schematic representation of such a human input estimator can be found in fig. 3.8.

In both controllers which use human input estimation a filter for removing high-frequency noise is implemented in order to obtain the human input.

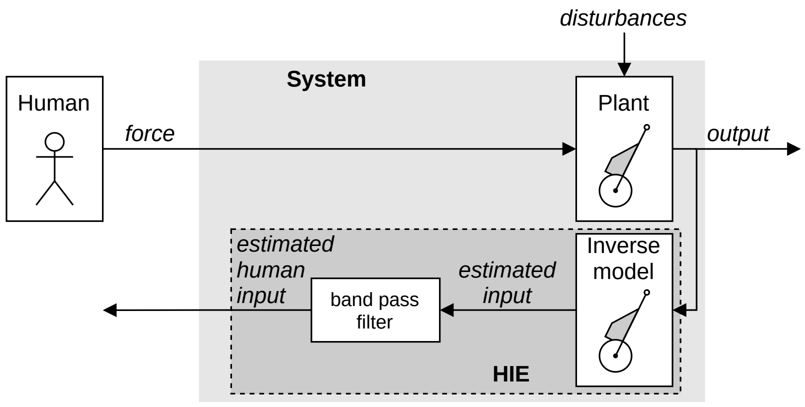


Figure 3.8: Schematic overview of a Human Input Estimator (HIE): a structure for estimating the human input force, similar to the Disturbance Observer described in section 3.4.3.

While the measuring of the human input might be the most straight-forward option, human input estimation will remove the need of force sensors; this can lead to cost reduction as well as an increased system robustness.

## 3.6 Comparison

In previous sections several different optional control methods for use in the assistive shopping trolley were analyzed based on criteria related to the review objective formulated in section 1.1: 'Objective'. In this section a comparison will be made between the analyzed control methods, and the most suitable control type as well as human input conversion method is chosen.

### 3.6.1 Control types

The scores given in the preceding analysis produce the Harris profile seen in table 3.1. It can clearly be seen that different control types have different advantages and disadvantages, and no single one can be clearly called the most suitable for use in the assistive shopping trolley. A distinction can be made nonetheless based on the notion that the four criteria based on the design objectives, namely 'assist', 'disturbance handling', 'passivity' and 'predictability' are of high importance, and therefore it is desirable to implement a controller which performs well on these criteria. Proportional torque control lacks on disturbance attenuation since no control loop is present. Cruise control scores badly on predictability and passiveness due to its un-intuitive handling and lack of requiring human input force/torque for actuation. Integral control scores badly on almost all criteria due to its lag, high order transfer function and not being passive as well. Therefore those three options are deemed unfit.

Table 3.1: Harris profile of reviewed control types

| | Prop. torque control | | | | Prop. velocity control | | | | Integral control | | | | DOB control | | | | Impedance control | | | | Cruise control | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Assist | | | 🟩 | 🟩 | | | 🟩 | 🟩 | | 🟧 | | | | | 🟩 | | | | 🟩 | | | | 🟩 | 🟩 |
| Disturbance | 🟧 | 🟧 | | | | | 🟩 | | | | 🟩 | | | | 🟩 | 🟩 | | | 🟩 | 🟩 | | | 🟩 | |
| Passive | | | 🟩 | 🟩 | | | 🟩 | | | 🟧 | | | | | 🟩 | | | | 🟩 | | 🟧 | 🟧 | | |
| Predictable | | | 🟩 | | | | 🟩 | | | 🟧 | | | | | 🟩 | | | | 🟩 | | | 🟧 | | |
| Independent | | | 🟩 | 🟩 | | 🟧 | | | | 🟧 | | | | | 🟩 | | | 🟧 | | | | 🟧 | | |

The remaining three options, namely proportional velocity control, disturbance observer control and impedance control, all score relatively well on these four criteria. All score equally well on passivity and predictability. Proportional velocity control scores particularly well on assist due to it having virtually no lag, whilst the other two do have a slower response. On the other hand DOB control and impedance control excel in disturbance attenuation because of their incorporation of a dynamic model, whilst proportional velocity control is lacking this. Both assist and disturbance handling are important in the design of the controller; therefore no further distinction can be made on the basis of these criteria.

In order to make a decision between the three we must look at the remaining criterion; 'independence'. For this criterion proportional torque control as well as impedance control score relatively low; both rely on an arbitrary velocity control element to make the system follow the reference velocity they provide. Since the type of velocity control which can be used is not related to the control types under review, no assumptions can be made regarding its dynamics. This imposes an amount of uncertainty regarding the controller's dynamic behavior. DOB control scores better in this respect; although the transformation from estimated disturbance force to the motor torque needed to compensate for this disturbance is not inherent to the control type and therefore can be seen as an additional control element, this transformation (eq. (3.5)) is straightforward and its dynamics can be stated to have little influence on the controller performance.

Concluding, from this analysis it can be stated that Disturbance Observer Control is deemed the most suitable for use in the assistive shopping trolley.

### 3.6.2 Human input conversion methods

With respect to human input conversion, human input estimation looks promising; it provides the possibility of reducing cost and increasing robustness. Also, it would be relatively efficient to design since it could make use of the same inverse model that is already composed for use in the DOB. Even though human input estimation seems to be advantageous, it also presents an additional effort to properly design it. Considering workload, it is therefore chosen to use force measurement as human input conversion method for the scope of this thesis.

## 3.7 Controller proposal

Based on the conclusions of the comparison in previous section, a suitable controller structure for use in the assistive shopping trolley is now proposed. The main element of the controller will be a Disturbance Observer (DOB) which has the function to attenuate disturbances such as the effects of road inclination and additional mass of groceries acting on the trolley. Furthermore it is decided to use force sensors to obtain the human input, since these are readily available on the assistive trolley prototype. A schematic overview of the proposed control structure is given in fig. 3.4, revisited on the current page.

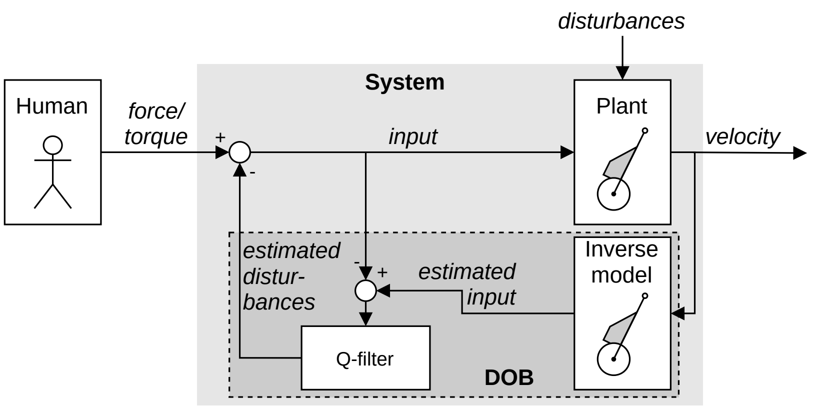


Figure 3.4 (revisited): Schematic overview of the proposed Disturbance Observer (DOB) controller structure.

## 3.8 Conclusion

This chapter has aimed to find the most suitable control structure for use in an assistive shopping trolley. Several controllers found in literature were analyzed. The analysis focused on two aspects: the type of control and the type of human input conversion, which indicates the way a controller obtains information about the human force input provided to the trolley. The suitability per control type is determined by rating them on several criteria based on the design objective formulated in section 1.1. Subsequently they are compared using a Harris Profile, from which the most suitable one is chosen to use as part of the controller design for the assistive shopping trolley. A suitable human input conversion method is chosen based on a combination of effectiveness of the method and the effort required for implementation.

Several different control types are found in literature: proportional-torque control, proportional-velocity control, integral-velocity control, disturbance observer control, impedance control, cruise control and fuzzy control. The proportional controllers relate the input force applied on the system with the system output using a simple proportional gain. Here the system output is either the motor torque or trolley velocity, depending on the type of proportional controller. Integral velocity control on the other hand relates the controller input to the desired acceleration of the system using an integrator. In disturbance observer control the effect of disturbances like road slope and additional weight is estimated using a dynamic model of the controlled plant, which is subsequently attenuated in a feedback loop. An impedance controller includes a dynamic plant model as well, but here it is used to predict the output of the desired plant behavior given the provided input force. This prediction is then used as a reference signal for the real plant. Cruise control aims to keep a reference velocity which can be manually set through a control panel, and therefore does not relate the applied human force to the system output at all. Lastly, fuzzy control determines which control action to take by classifying a situation based on the amount in which it fits into several relevant 'fuzzy sets'. The classification is purely based on inputs and therefore does not require a dynamic model of the controlled plant, which is mentioned to be the reason this control type is used for the controller in question. Since the formulation of an appropriate dynamic model of an assistive shopping trolley is however considered achievable this control type is not taken into further account.

The criteria on which these control types are assessed are 'assist', 'disturbance handling', 'passivity', 'predictability', 'human input conversion' and 'independence'. 'Assist' indicates the amount of help the controller provides given the amount of human input. The measure of 'disturbance handling' regards the amount and accuracy of the attenuation of disturbances like the influence of heavy groceries or a sloped road on the trolley dynamics. 'passivity' expresses to what extent a control type can control the trolleys behavior when no user input is given. 'predictability' is a measure which provides information about how intuitive the controller's response is expected to be for the elderly user. This is influenced by the order of the resulting trolley-controller system and whether the controller takes into account the trolley dynamics. The last criterion, 'independence', describes the extent to which additional control needs to be implemented in order to make the controller work properly. This criterion is not based on the design objective but is still of interest since it gives an indication of the accuracy of the given scores.

The resulting Harris Profile, which combines the scores of all control types into one overview, can be found in table 3.1 of section 3.6. From the Harris Profile it is concluded that both disturbance observer control and impedance control are well-equipped for use in the assistive shopping trolley. However, due to the former having the largest independence score, disturbance observer control is considered to be the most suitable in general.

With respect to human input conversion two methods are found between the analyzed controllers; measurement, which obtains the applied human force using sensors, and estimation, which uses an observer to derive the applied human input using other available information. Although input estimation seems promising, due to the additional effort required for its implementation in the controller it is chosen to use measurement as means to obtain the applied human force on the trolley handle.

A controller is proposed, combining the most favourable control type and human input conversion method. Additionally it is decided to add a proportional controller with a manually adjustable gain in front of the disturbance observer controller in order to further improve the assistive function of the shopping trolley. A schematic overview of the proposed controller can be found in fig. 3.4 (revisited) of section 3.7.

We now have obtained the desired control structure. In the next chapter, the desired control structure is used to analyse the closed-loop response of the trolley-controller system and the different elements of the controller are designed.

Chapter 4

# Controller Design

In the previous chapter, the structure for a suitable control system for an assistive shopping trolley was determined. The main goals of the controller, as described in section 1.1, are to provide assist to- as well as to gain acceptance of- the elderly user when using the assistive shopping trolley. A structural analysis of several control types found in literature has revealed that a controller implementing a Disturbance Observer is best suited for these goals. Furthermore, it is decided that the human input force will be measured using force-sensors in the trolley handle. In the current chapter, the elements of the proposed controller structure are designed in detail.

## 4.1 General design

The general design of the assistive shopping trolley control system has been determined in chapter 3: ‘Control Method Overview’. It consists of a Disturbance Observer (DOB), which uses the human handle force acting on the trolley handle along with the trolley’s acceleration and applied motor torque to estimate the unwanted effects, called disturbances, acting on the trolley.

The DOB intends to counteract the undesired disturbance forces acting on the trolley by making an estimation of these disturbance forces and using the estimation in a feedback loop to attenuate them. A DOB consists of two main elements; an inverse nominal model and a Q-filter. The nominal model represents the trolley’s desired dynamic behavior, without any disturbances present. It used to make an estimation of what the trolley’s total input would have been when it would have behaved in this desired manner. The difference between this estimation and the intended trolley input, consisting of the measured input force applied on the trolley handle, is caused by unwanted disturbances. This disturbance estimation is passed through a Q-filter, which ensures causality of the feedback loop and removes high-frequency noise. Finally the filtered disturbance signal is used as a control signal to the trolley’s motors, which then produce a torque opposing the disturbances.

A detailed schematic overview of the proposed controller system is provided in fig. 4.1. This schematic depicts the chosen controller structure as shown in section 3.7: ‘Controller proposal’, while some small adjustments have been made to provide more detail regarding the exact implementation. Both trolley inputs, namely the handle force and the motor torque signal, are now depicted separately as they are separate signals. Also, the measured force signal is specified explicitly. Furthermore the trolley output is now specified to be the measured acceleration (in contrast to the velocity used in chapter 3), since this is the signal that will be used by the actual controller.

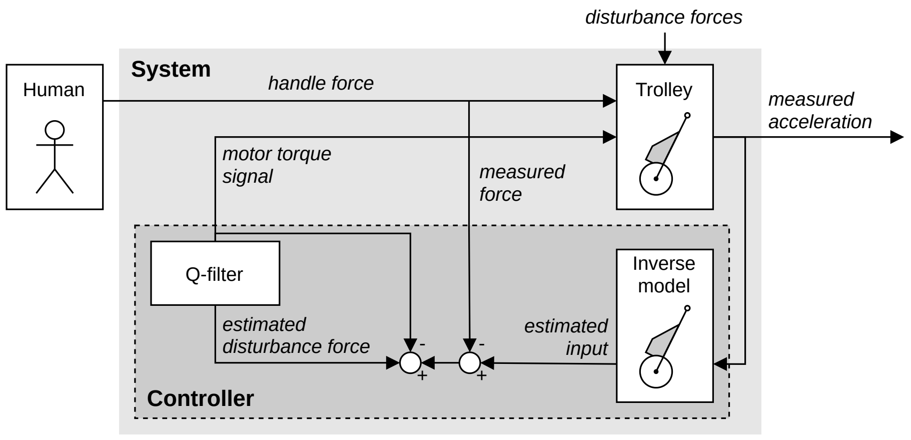


Figure 4.1: Detailed overview of proposed controller structure for the assistive shopping trolley.

## 4.2 Inverse nominal model

This section details the design of the controller's inverse nominal model, which represents the trolley's desired behavior.

### 4.2.1 Nominal behavior

The trolley's desired behavior is represented by the controller's inverse nominal model. The desired behavior is specified in section 1.1: 'Objective', where the objective is set to make the trolley feel "as if it is empty and on a flat road". From this statement, three main considerations for the controller's nominal model can be derived: Firstly, the trolley is desired to behave like an actual trolley, which means that the model should include dynamics representing a trolley. Therefore, the equation (2.10) (revisited below) produced to represent the trolley dynamics in chapter 2 is used as the basis for the nominal model design.

$$
\begin{aligned}
&\Big(\mu m_{FT}\big(L_{WF}\sin(\beta)+W_{WF}\cos(\beta)\big)+L_{WH}\cos(\beta)(m_{FT}+3m_W)\Big)\ddot{x}_W\\
&=\Big(L_{WH}\big(\mu\sin(\beta)+\cos(\beta)\big)\Big)F_{HX}+2\Big(L_{WH}\cos(\beta)R_W^{-1}-\mu\Big)T_M\\
&-2\Big(L_{WH}\big(\mu\cos(\gamma)+\sin(\gamma)\big)\cos(\beta)\Big)F_{WG}\\
&-2\Bigg(\Big(L_{WH}\cos(\beta)(m_{FT}+3m_W)+\mu m_{FT}\big(L_{WF}\sin(\beta)+W_{WF}\cos(\beta)\big)\Big)\\
&\qquad\Big(\mu L_{WF}\big(\sin(\beta)\sin(\beta+\gamma)-\cos(\gamma)\big)+L_{WH}\cos(\beta)^2\big(\mu\cos(\gamma)+\sin(\gamma)\big)\\
&\qquad\quad+\mu W_{WF}\big(\sin(\beta)\cos(\beta+\gamma)+\sin(\gamma)\big)\Big)\Big(L_{WF}m_{FT}\mu\sin(2\beta)\\
&\qquad\quad+\big((m_{FT}+3m_W)L_{WH}+\mu m_{FT}W_{WF}\big)\big(\cos(2\beta)+1\big)\Big)^{-1}\Bigg)F_{FG}
\end{aligned}
\tag{2.10 revisited}
$$

where

$$
m_{FT}=m_F+m_B; \qquad F_{FG}=m_{FT}g; \qquad F_{WG}=m_Wg; \qquad \beta=\sin^{-1}\left(\frac{L_H-R_W}{L_{WH}}\right).
$$

Secondly, the trolley's perceived weight is desired to equal its empty weight at all times. This requirement is included in the nominal model by defining the trolley bag mass $m_B$ to be that of an empty bag.

Lastly, the trolley is desired to act like it is always traveling on a flat road, which is implemented in the nominal model by setting the model's road slope parameter $\gamma$ to a constant of 0 rad.

In order to estimate the disturbances acting on the trolley as well as possible, the remaining parameters of the nominal model must match the parameters of the real situation as close as possible. A prototype of the assistive shopping trolley is available, which is used as a reference to obtain the remaining weights and dimensions. An overview of the chosen numerical values for all parameters can be found in table 4.1.

The trolley frame height $L_{WH}$ is measured, as is frame depth $W_F$, bag height and -depth $L_B$ and $W_B$ and wheel radius $R_W$. The masses of frame, empty bag and wheels $m_F$, $m_B$ and $m_W$ are weighed. Furthermore, the gravitational acceleration $g$ is taken to be the conventional standard value of $9.81\,\mathrm{m\,s^{-2}}$. The height of the trolley handle with respect to the ground $L_H$ during use is unknown, therefore its value is set to the fist height of the average Dutch elderly person of 60+ years of age when standing with his/her arms along his/her body [25]. The distances between the center of mass of the bag-frame combination and the frame's point of rotation $W$, $L_{WF}$ and $W_{WF}$, which are defined colinear and perpendicular to the orientation of the trolley frame, are estimated based on the configuration and weights of the frame and bag. The resulting relations are:

$$L_{WF} = \frac{1}{m_F + m_B}\left(\frac{m_F}{2}L_{WH} + \frac{m_B}{3}L_B\right); \qquad W_{WF} = \frac{m_B}{m_F + m_B}(\frac{1}{2}W_F + \frac{1}{3}W_B) \tag{4.1}$$

The complete estimation can be found in appendix A.

An essential component of the nominal trolley model is the rolling friction; This friction ensures that a constant nonzero input force from the user on the trolley handle is required to keep a constant velocity. Without it, no form of friction would be included in the model. Since the controller uses the model as a reference of the desired behavior, this means that any friction acting on the trolley would be seen as a disturbance, resulting in its attenuation. A frictionless trolley is not only expected to feel unnatural to the user, therefore reducing its predictability, it also lacks a braking force to stop the trolley in the absence of human input. This has a negative effect on the trolley's passivity, since the trolley will therefore not brake on its own; a negative human input force should be applied to bring the trolley to a standstill. Although the situation of a 'runaway trolley' could (and should[1]) be easily prevented by implementing a check independently from the inverse nominal model to only enable controller feedback whenever an input force is applied, it is preferred to prevent the need for such a system for the controller to work properly in the first place. Rolling friction exists mainly of friction due to hysteresis caused by deformation of the tires and road surface [12]. The corresponding rolling friction coefficient $\mu$ is based on car tires on a smooth surface [26, 27].

[1]A check is especially important when the system will be used in real life, and is not yet implemented in the scope of this thesis.

Table 4.1: Fixed and nominal parameter values of the trolley model

| | **Nominal value** | **Description** |
|---|---|---|
| *Fixed* | | |
| $L_B$ | 0.65 | Height of trolley bag [m] |
| $W_B$ | 0.22 | Depth of trolley bag [m] |
| $L_{WH}$ | 0.95 | Length of trolley frame [m] |
| $W_F$ | 0.015 | Depth of trolley frame [m] |
| $R_W$ | 0.11 | Trolley wheel radius [m] |
| $m_F$ | 0.9 | Mass of trolley frame [kg] |
| $m_W$ | 1.0 | Mass of trolley wheel [kg] |
| $g$ | 9.81 | Gravitational acceleration [m/s$^2$] |
| *Variable* | | |
| $L_H$ | 0.742 | Perpendicular distance between road and trolley handle [m] |
| $m_B$ | 1.2 | Mass of trolley bag [kg] |
| $\gamma$ | 0.00 | Road inclination angle [rad] |
| $\mu$ | 0.01 | Rolling friction coefficient [-] |

### 4.2.2 Inverse nominal model equation

As described in the previous section, equation (2.10) is used as a basis for the controller's internal nominal model. The inverse nominal model is obtained by rewriting eq. (2.10) to a form where the input force on the trolley handle $F_{HX}$ is the output and the trolley's acceleration $\ddot{x}_W$ becomes an input along with motor torque $T_M$. Meanwhile defining $\gamma$ to be 0 as specified in section 4.2.1 results in the following equation to be used as inverse nominal model:

$$\begin{aligned} F_{HX} = \Bigg[ & \Big(\mu m_{FT}\big(L_{WF}\sin(\beta) + W_{WF}\cos(\beta)\big) + L_{WH}(m_{FT} + 3m_W)\cos(\beta)\Big)\ddot{x}_W \\ & - 2\Big(L_{WH}\cos(\beta)R_W^{-1} - \mu\Big)T_M + \Big(2\mu L_{WH}\cos(\beta)\Big)F_{WG} \\ & + \mu\Big(L_{WH}\cos(\beta) + W_{WF}\sin(\beta) - L_{WF}\cos(\beta)\Big)F_{FG}\Bigg] \\ & \Big(L_{WH}\big(\mu\sin(\beta) + \cos(\beta)\big)\Big)^{-1} \end{aligned} \tag{4.2}$$

where

$$m_{FT} = m_F + m_B; \qquad F_{FG} = m_{FT}g; \qquad F_{WG} = m_W g;$$
$$\beta = \sin^{-1}\left(\frac{L'_H}{L_{WH}}\right); \qquad L'_H = L_H - R_W.$$

The remaining parameter values as determined in section 4.2.1 can be found in table 4.1.

**Nonlinear implementation of rolling friction**

As mentioned in section 2.3: 'Model limitations', the trolley's equation (2.10) on which the inverse nominal model equation is based is only valid for positive velocities. This is due to the linear implementation of rolling friction torque $T_{ROLL}$ (eq. (2.1), revisited on this page) on the trolley wheels in the dynamic model equation; the rolling friction is assumed to always act in counterclockwise direction. In reality however, $T_{ROLL}$ will always act in opposite direction of the direction of movement, thus its orientation will flip whenever the trolley travels backwards. Furthermore, $T_{ROLL}$ will be zero when the trolley is at a standstill. To improve controller performance for the situations where the trolley's velocity is negative or zero, eq. 4.2 is updated by replacing the linear implementation of $T_{ROLL}$ with an implementation which takes the nonlinearities of $T_{ROLL}$ into account. The improved form of $T_{ROLL}$ is presented in eq. (4.3):

$$\text{Old implementation:} \qquad T_{ROLL} = \mu F_N R_W \tag{2.1 revisited}$$

$$\text{New implementation:} \qquad T_{ROLL} = \begin{cases} -\mu F_N R_W & \dot{x}_W < 0 \\ 0 & \dot{x}_W = 0 \\ \mu F_N R_W & \dot{x}_W > 0 \end{cases} \tag{4.3}$$

Substituting eq. (4.3) for eq. (2.1) in the initial equations of motion eqs. (2.3) established in chapter 2: 'Trolley Dynamics' yields the following modified version of inverse nominal model equation 4.2:

$$\boxed{\begin{aligned} F_{HX} = \Bigg[ & \Big(\widetilde{\mu} m_{FT}\big(L_{WF}\sin(\beta) + W_{WF}\cos(\beta)\big) + L_{WH}(m_{FT} + 3m_W)\cos(\beta)\Big)\ddot{x}_W \\ & - 2\Big(L_{WH}\cos(\beta)R_W^{-1} - \widetilde{\mu}\Big)T_M + \Big(2L_{WH}\widetilde{\mu}\cos(\beta)\Big)F_{WG} \\ & + \bigg(\Big(L_{WH}\cos(\beta) + W_{WF}\sin(\beta) - L_{WF}\cos(\beta)\Big)\widetilde{\mu} \\ & \quad + L_{WH}\cos(\beta)\bigg)F_{FG}\Bigg]\Big(L_{WH}\big(\widetilde{\mu}\sin(\beta) + \cos(\beta)\big)\Big)^{-1} \qquad (4.4) \end{aligned}}$$

where

$$m_{FT} = m_F + m_B; \qquad F_{FG} = m_{FT}g; \qquad F_{WG} = m_W g;$$

$$\beta = \sin^{-1}\left(\frac{L_H - R_W}{L_{WH}}\right); \qquad \widetilde{\mu} = \text{sign}(\dot{x}_W)\mu.$$

Comparing the newly obtained eq. 4.4 with eq. 4.2 shows that the new implementation of $T_{ROLL}$ has effectively made the direction of rolling friction coefficient $\mu$ dependent of the direction of trolley velocity $\dot{x}_W$.

It must be noted that this new implementation of $T_{ROLL}$ has as a consequence that the inverse nominal model equation is no longer linear, which complicates the remainder of the controller design. Noise in the velocity signal can cause problems when the trolley's velocity is low; the direction of $T_{ROLL}$ might flip while the trolley is not actually changing direction, potentially causing instability. This problem is solved by implementing a velocity deadband; a velocity-range around 0 in which the velocity will be assumed 0. Whenever the trolley's velocity measures within this deadband, the rolling friction as defined in eq. (4.3) will become zero as well. If the deadband range is chosen to be larger than the noise in the velocity data, it will prevent the incorrect flipping of $T_{ROLL}$, therefore preventing possible problems.

## 4.3 Q-filter

The name ‘Q-filter’ is generally used to indicate a filter with a given ‘Q’- or ‘Quality’ factor; A term referring to the filter’s selectiveness with respect to different frequencies [28]. The Q factor is related to a filter’s resonance frequency[2], and therefore it only applies to second- and higher order systems. Although the filter within a DOB is commonly called a Q-filter, the above mentioned definition does not necessarily apply since a first order filter might be used as well.

Within the DOB, the Q-filter serves multiple purposes [29]. Firstly the inverse nominal model implemented in the controller might not be causal, causing the system to be improper. That is, the inverse nominal model may contain differential elements which as-is cannot be solved in real-time. The DOB’s Q-filter can satisfy properness of the control loop by providing compensatory integral action. Furthermore, the design of the Q-filter can be used to ensure stability of the closed-loop system of trolley and controller and influences the robustness of the controller against disturbances and noise.

The current section concerns the design of the Q-filter for use in the trolley controller. The boundaries for the Q-filter design are determined by analyzing the controller’s properness and performing a stability analysis of the trolley-controller system’s closed-loop behavior. Furthermore, an analysis of the system’s sensitivity functions is performed to decide on the optimal parameter values within the boundaries. All analyses are performed on the system’s transfer functions; therefore an appropriate representation of the trolley-controller system in the frequency domain will be presented first.

### 4.3.1 System representation in the frequency domain

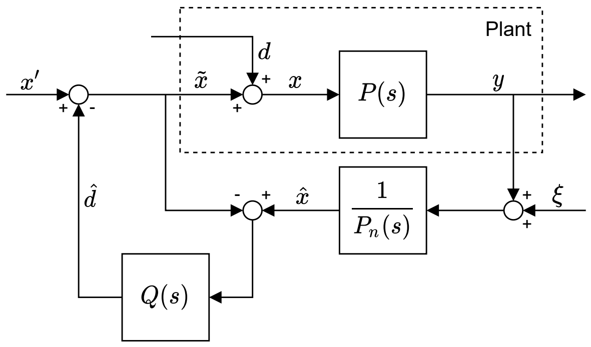
In the frequency domain, dynamic elements can be represented by their transfer functions. A general system of plant and DOB can be represented by the block scheme depicted in fig. 4.2a, where $P$, $P_n$ and $Q$ represent the tranfer functions of the plant, the plant’s nominal model and the DOB’s Q-filter, respectively [29, 30]. Furthermore, signal $x'$ represents the deliberate input to the complete system of plant and controller, $\tilde{x}$ the total deliberate plant input, $x$ the actual plant input and $\hat{x}$ an estimation of the plant input by the internal nominal model, signals $d$ and $\xi$ represent the disturbances acting on the plant and the measurement noise, $y$ represents the plant output signal, and $\hat{d}$ represents the disturbance estimation made by the DOB. It is desired to obtain a similar structure for the trolley system in order to perform the analyses using conventional methods.

#### 4.3.1.1 System structure

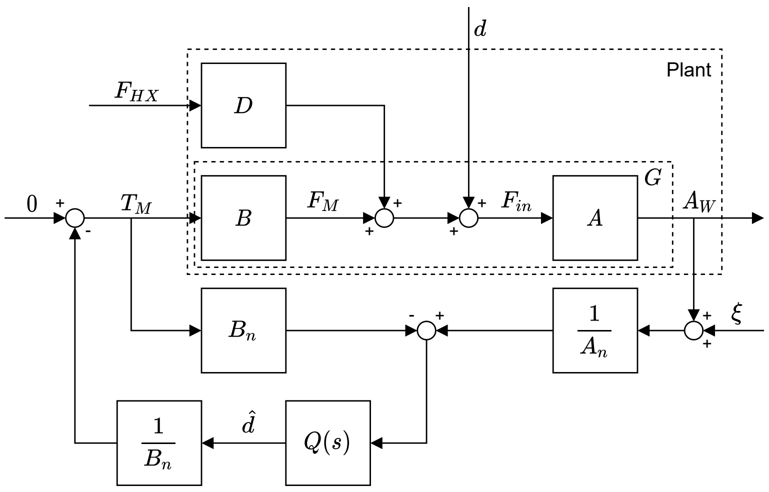
Figure 4.2a cannot directly be used to represent the trolley-controller system due to an essential difference in the way the inputs on both systems are applied. In the general representation seen in fig. 4.2a, all inputs are assumed to be having the same dynamic effects on the plant output. For example, in power-assist bicycles both the human input exerted on the paddles and the motor torque exerted by the wheel motor often apply on the same point of the plant, namely the axis of the rear wheel. Therefore, their effect on the bicycle’s acceleration is the same and they can effectively be seen as one signal, shown as $\tilde{x}$ in fig. 4.2a. In the case of the trolley however, the human input force $F_{HX}$ and the motor torque $T_M$ are applied on different parts of the trolley, therefore having their own, distinct dynamic effects. Because of this, they cannot be represented as a single input signal acting on the plant. Consequently, trolley inputs $T_M$ and $F_{HX}$ are represented as separate plant inputs.

The resulting structure can be seen in fig. 4.2b. Since not all dynamics are shared among all inputs, the plant dynamics are divided into three blocks, each containing the dynamics specific to a certain input; Block $D$ and $B$ include the dynamics specifically corresponding to plant inputs $F_{HX}$ and $T_M$, and block $A$ contains the remaining dynamics which apply equally to all inputs of the plant (including disturbance signal $d$). $F_M$ represent the force signal resulting from the dynamic transformation of $T_M$ by block $B$. Note that $F_{FG}$ and $F_{WG}$ and their corresponding dedicated dynamics are not included in the block diagram; Although they are also inputs of the trolley-controller system, they are not part of the feedback loop and are therefore not of interest for any of the analyses. To avoid unnecessary cluttering of the block scheme, they are therefore left out.

[2]The frequency where the largest gain is induced.



(a) General system of plant and DOB [29, 30]



(b) Block scheme of the trolley-controller system, excluding dynamics related to gravitational forces $F_{FG}$ and $F_{WG}$.

Figure 4.2: Block schemes representing the structure of general- and trolley closed-loop system dynamics.

Additional adjustments are made so that the combined dynamics again match the behavior of the general system shown in fig. 4.2a. For this, a similar division is made in the plant's nominal representation ($P_n$ in fig. 4.2a), where $A_n$ and $B_n$ are introduced as the nominal versions of blocks $A$ and $B$. The trolley-controller system's closed-loop dynamics now correspond to those of the general system, where the combination of blocks $A$ and $B$ correspond to block $P$, and $A_n$ and $B_n$ consequently correspond to $P_n$. Note that the dynamics in block $D$ are not considered to be part of $P$; $F_{HX}$ is seen as an independent input, therefore the dynamics in $D$ do not effect the controller feedback loop of which $P$ is part. The combined dynamics of blocks $A$ and $B$ of the trolley plant, which correspond to the dynamics for an arbitrary plant $P$ within a general feedback system, will be references as the trolley's loop dynamics $G$:

$$G = AB \tag{4.5}$$

#### 4.3.1.2 Transfer functions

The transfer functions represented by blocks $A$, $B$ and $G$ can be obtained by transforming the corresponding dynamics from time to frequency domain. Equation (2.10) (revisited below) is used as a representation for the trolley plant dynamics; the differences between its simplified dynamics and the trolley's real dynamic behavior due to the neglect of certain dynamic effects can be taken into account as neglected-dynamics uncertainty, as will be explained later.

$$\begin{aligned}
&\Big(\mu m_{FT}\big(L_{WF}\sin(\beta)+W_{WF}\cos(\beta)\big)+L_{WH}\cos(\beta)(m_{FT}+3m_W)\Big)\ddot{x}_W \\
&=\Big(L_{WH}\big(\mu\sin(\beta)+\cos(\beta)\big)\Big)F_{HX}+2\Big(L_{WH}\cos(\beta)R_W^{-1}-\mu\Big)T_M \\
&-2\Big(L_{WH}\big(\mu\cos(\gamma)+\sin(\gamma)\big)\cos(\beta)\Big)F_{WG} \\
&-2\Bigg(\Big(L_{WH}\cos(\beta)(m_{FT}+3m_W)+\mu m_{FT}\big(L_{WF}\sin(\beta)+W_{WF}\cos(\beta)\big)\Big) \\
&\qquad\Big(\mu L_{WF}\big(\sin(\beta)\sin(\beta+\gamma)-\cos(\gamma)\big)+L_{WH}\cos(\beta)^2\big(\mu\cos(\gamma)+\sin(\gamma)\big) \\
&\qquad\quad+\mu W_{WF}\big(\sin(\beta)\cos(\beta+\gamma)+\sin(\gamma)\big)\Big)\Big(L_{WF}m_{FT}\mu\sin(2\beta) \\
&\qquad\quad+\big((m_{FT}+3m_W)L_{WH}+\mu m_{FT}W_{WF}\big)\big(\cos(2\beta)+1\big)\Big)^{-1}\Bigg)F_{FG} \qquad \text{(2.10 revisited)}
\end{aligned}$$

where

$$m_{FT}=m_F+m_B; \qquad F_{FG}=m_{FT}g; \qquad F_{WG}=m_Wg; \qquad \beta=\sin^{-1}\left(\frac{L_H-R_W}{L_{WH}}\right).$$

Equation (2.10) is arranged so that the dynamics applying to the same signals are grouped, which makes it easy to identify the dynamic components relating to blocks $A$ and $B$:

$$A: \quad \ddot{x}_W(t)=\Big[\mu m_{FT}\big(L_{WF}\sin(\beta)+W_{WF}\cos(\beta)\big)+L_{WH}\cos(\beta)(m_{FT}+3m_W)\Big]^{-1}F_{in}(t) \qquad (4.6a)$$

$$B: \quad F_M(t)=2\Big[L_{WH}\cos(\beta)R_W^{-1}-\mu\Big]T_M(t) \qquad (4.6b)$$

where

$$m_{FT}=m_F+m_B; \qquad \beta=\sin^{-1}\left(\frac{L_H-R_W}{L_{WH}}\right).$$

The dynamics between the in- and output signals in eqs. (4.6a) and (4.6b) contain only constants, which makes their transformation to transfer functions in frequency domain straightforward using Laplace transforms:

$$A: \tag{4.7a}$$

$$A_W(s) = \Big(\mu m_{FT}\big(L_{WF}\sin(\beta) + W_{WF}\cos(\beta)\big) + L_{WH}\cos(\beta)(m_{FT} + 3m_W)\Big)^{-1} F_{in}(s);$$

$$A(s) = \frac{A_W(s)}{F_{in}(s)} = \Big(\mu m_{FT}\big(L_{WF}\sin(\beta) + W_{WF}\cos(\beta)\big) + L_{WH}\cos(\beta)(m_{FT} + 3m_W)\Big)^{-1} \tag{4.7b}$$

$$B: \tag{4.7c}$$

$$F_M(s) = 2\Big(L_{WH}\cos(\beta)R_W^{-1} - \mu\Big)T_M(s);$$

$$B(s) = \frac{F_M(s)}{T_M(s)} = 2\Big(L_{WH}\cos(\beta)R_W^{-1} - \mu\Big) \tag{4.7d}$$

The transfer function for block $G$ can now be obtained as well by making use of eq. (4.5):

$$\boxed{G(s) = A(s)B(s) = 2\,\frac{L_{WH}\cos(\beta)R_W^{-1} - \mu}{\mu m_{FT}\big(L_{WF}\sin(\beta) + W_{WF}\cos(\beta)\big) + L_{WH}\cos(\beta)(m_{FT} + 3m_W)} \quad (4.8)}$$

where

$$m_{FT} = m_F + m_B; \qquad \beta = \sin^{-1}\left(\frac{L_H - R_W}{L_{WH}}\right).$$

### 4.3.2 Properness

A system is considered to be ‘proper’ if the degree of the numerator of its transfer function does not exceed the degree of the denominator [14, 31]. In other words, a proper system needs no netto differential action to obtain the system output, it is only dependent on present and previous inputs. An improper system are not physically realizable since it will have to see into the future to obtain its current output.

Disturbance Observers often include an improper inverse nominal model, due to the model being the inversion of the proper plant dynamics. This improperness must then be balanced by the Q-filter to result in an implementable controller, therefore putting a constraint on the Q-filter’s design. Although the real trolley dynamics are indeed proper, it’s first- and higher order dynamics such as position- and velocity-dependent effects are neglected in equation (2.10), as described in more detail in section 2.2. The trolley model used as the basis for the controller’s inverse nominal model is therefore 0’th order, which makes the inverse nominal itself of 0’th order as well. Since a 0’th order system is proper, no restrictions have to be imposed on the controller’s Q-filter to ensure properness of the trolley-controller system as a whole.

### 4.3.3 Closed-loop analysis

Instability in closed-loop systems can occur if the feedback signal reinforces itself when it passes through the closed-loop circuit. This will cause the controller output to blow up, and therefore the trolley to go out of control. An important aspect of the controller design is therefore to ensure that the closed-loop system of trolley and controller will be stable. This needs to be the case under all reasonable circumstances; the controller must therefore be 'robust' against unmodeled changes in the trolley's behavior.

The robustness of a control system is an indication of its sensitivity to differences between the system's expected and actual behavior, generally called 'model uncertainty' [32]. This model uncertainty can cause the real system to become unstable, even though the controller is designed to stabilize the corresponding model. A robust control system can handle large model uncertainties without the system destabilizing. Having a robust control system is especially important in the case of the assistive shopping trolley; the trolley system includes large unknown disturbances such as changing weight and road inclination, and on top of that a failing control system can have a detrimental effect on the trust of the elderly user. For this reason a stability analysis is performed, which establishes criteria for robustness on the Q-filter design.

#### 4.3.3.1 $H_\infty$-approach

The behavior of closed-loop controlled systems can be analyzed by means of their transfer functions. This type of analysis is called 'loop analysis' [33]. The system's stability can be analyzed by determining how sinusoidal signals propagate through the feedback loop, and for which conditions they will reinforce themselves. For these conditions the system is unstable, since the signal will now grow exponentially and independent of any external inputs, resulting in the system spinning out of control. The analysis makes use of the system's loop transfer function, which represents the signal's journey around the loop.

In order to ensure the trolley system's stability in the presence of uncertainty, an $H_\infty$-stability analysis is performed on the closed-loop controlled system [14, 29, 32, 31]. The key idea of this type of analysis is to generate a set of all possible dynamic plant models within the scope of the model uncertainty, called the 'uncertainty set', and use loop analysis to determine conditions for stability of the closed-loop system for all models within the set. The analysis is performed in the frequency domain; therefore, the system behavior will be described with respect to the complex plane.

The implementation of the $H_\infty$-method will be revealed in more detail in future sections. First, the sources of model uncertainty present within the trolley-controller system will be discussed.

#### 4.3.3.2 Sources of model uncertainty in the trolley-controller system

Model uncertainty can be composed from different sources [32]. Within the trolley system, the most important contributors to model uncertainty are presumed to be unknown parameter deviations from their nominal value, called 'parameter uncertainty'. Any disturbances acting on the system fall within this category.

In reality, there is a certain amount of uncertainty present in every parameter included in the model, since measurements and estimations of those parameters are never infinitely accurate. However, only parameter uncertainties which are assumed to have a significant impact on the trolley behavior are taken into account. Among these are the two often-encountered disturbances of additional mass and road inclination. Based on interviews with Dutch elderly, it is determined that the assistive shopping trolley should be able to carry up to 20 kg of additional weight [8]. Given the original trolley bag weight $m_B$ of 1.2 kg[3], a bag mass lying anywhere in the range of 1.2 kg - 21.2 kg must therefore be considered. Dutch law states that a ramp should not have an inclination angle of more than 0.10 rad [15]. Therefore the road inclination $\gamma$ is considered to fall within a range of $-0.10$ rad - 0.10 rad.

Another important addition to the trolley's parameter uncertainty comes from the range in height of different users, which translates to a variation in trolley handle height $L_H$. The range of $L_H$ is set to include the fist height between the 5th and 95th percentile of Dutch elderly of 60+ years of age [25], which results in a range of 0.661 m - 0.823 m. Lastly, the variation in rolling friction coefficient $\mu$ is taken into account, which is caused by several different factors such as road type and softness of the trolley's

[3]As determined in section 4.2.1: 'Nominal behavior'.

Table 4.2: Estimated ranges of uncertain parameter values

| | Value | | | |
|---|---|---|---|---|
| **Parameter** | Range | Avg | Nom | **Description** |
| $m_B$ | 1.2 - 21.2 | 11.2 | 1.2 | Mass of trolley bag[1] [kg] |
| $L_H$ | 0.661 - 0.823 | 0.742 | 0.742 | Trolley handle height with respect to the road[3] [m] |
| $\gamma$ | -0.10 - 0.10 | 0.00 | 0.00 | Road inclination angle[2] [rad] |
| $\mu$ | 0.102 - 0.22 | 0.156 | 0.01 | Rolling friction coefficient[4] [$-$] |

[1]Based on original trolley-bag mass of 1.2 kg and maximum additional weight of 20 kg [8].
[2]Maximum angles according to Dutch law [15].
[3]Fist height of P5-P95 of Dutch elderly of 60+ years of age [25].
[4]Based on friction coefficients of bicycle tires on concrete, car tires on gravel and rotary sliding-contact bearings [27, 34].

tires. Friction within the wheel bearings is taken into account here as well[4]. The range in rolling friction due to contact between the tires and the road is set to be 0.002 - 0.02, based on bicycle tires on concrete and car tires on gravel [27]. Rotary sliding-contact bearings generally have a friction coefficient between 0.05 and 0.1 [34]. Based on the wheels of the assistive shopping trolley prototype taken as reference for the controller design, two of such bearings are assumed per wheel. Taking both types of friction into account, the total rolling friction coefficient per wheel is therefore estimated to lie anywhere in the range of 0.102 - 0.22. The ranges of all considered uncertain parameters are shown in table 4.2, along with the corresponding average and nominal parameter values.

Besides parameter uncertainty, other types of uncertainty are present within the trolley-controller system. 'Neglected dynamics uncertainty' is caused by effects of dynamic elements which are deliberately left out of scope during the design of the controller. For the trolley, some elements were neglected for the linearization of its dynamic model; these include the no slip- and air drag conditions, as described in section 2.1, as well as the linearized implementation of $T_{ROLL}$. Another source of neglected dynamics uncertainty, which does not originate from the dynamic model design, is the delay in the feedback loop caused by the sensors and controller. The last source of uncertainty is 'unmodeled dynamics uncertainty'; this covers unknown dynamic effects, for example the system's behavior at high frequencies.

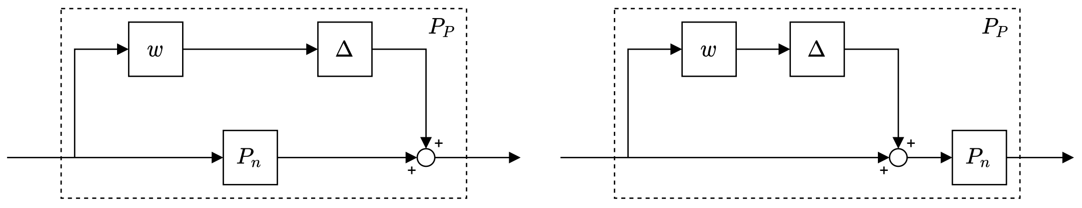
Since parameter uncertainty is considered to be the dominant type of model uncertainty, this type will be the focus of the stability analysis. Nevertheless, the addition of other uncertainty types in the analysis was kept in mind during the selection of the method for analysis, so that in any future work the remaining types may be easily included.

#### 4.3.3.3 Including uncertainty in the model description

**Representing uncertainty.** The system's model uncertainty is taken into account in the analysis by replacing the transfer function of the trolley plant with an uncertainty set ($\Pi$), which represents al possible variations of the plant model due to the model uncertainty [32]. The uncertainty within the set is represented by a stable '$H_\infty$-norm-bounded perturbation' ($\Delta$) around the average plant model, which defines a region of possible variations in the trolley's behavior in the complex plane for every frequency. This area is called the 'uncertainty region'. The perturbation can work in any direction[5] and the maximum absolute magnitude of any of its components is less than 1:

$$||\Delta(s)||_\infty \leq 1$$

[4]The inclusion of bearing friction within the rolling friction coefficient again introduces uncertainty; the rolling friction in the trolley model is implemented to act exclusively on the trolley wheel and not on the frame, whereas in reality friction in the bearings will act on both bodies. It is however assumed that this simplification is of negligible influence on the trolley dynamics.
[5]on the real and/or imaginary axis in the complex plane



(a) Additive uncertainty (b) Multiplicative uncertainty

Figure 4.3: Common methods for including uncertainty [32].

**Uncertainty descriptions.** Perturbation $\Delta$ can be represented in the model either through additive (absolute) or multiplicative (relative to the plant) uncertainty description:

*Additive uncertainty:* $P_p(s) = P_a(s) + w(s)\Delta(s);\qquad ||\Delta(s)||_\infty \leq 1$ (4.9a)

*Multiplicative uncertainty:* $P_p(s) = P_a(s)\big(1 + w(s)\Delta(s)\big);\qquad ||\Delta(s)||_\infty \leq 1$ (4.9b)

where $P_p$ represents any model within the uncertainty set, $P_a$ represents the set's average model or value and $w$ represents a weighing function for the perturbation [32]. A schematic representation for both additive and multiplicative uncertainty is given in fig. 4.3. Note that for Single Input Single Output (SISO) systems like the trolley-controller system, it doesn't matter if the multiplicative uncertainty is added before or after the average plant model since they obtain the same result.

For SISO plants, additive and multiplicative uncertainty representations are essentially the same since their weights can be chosen such that their equations are equal. Nevertheless, it is chosen to represent uncertainty in the trolley as multiplicative uncertainty as its weighing function $w$ directly provides useful information regarding the stability of the uncertain system. When $w$ becomes larger than 1, the uncertainty of the plant becomes larger than the plant itself; In this case there will be a possibility that the transfer function of plant $P_p$ becomes 0, meaning that the system will be uncontrollable. This notion is later used for the analysis of the uncertain trolley-controller system.

**Combining different sources of uncertainty.** In order to simplify the analysis, multiple sources of uncertainty within a system are generally represented as a single, 'lumped' perturbation [32]. Combining multiple perturbations into one often results in a lumped perturbation with a complicated shape, which is difficult to work with. Therefore, its area is approximated by a disk, which covers the original surface. This disk can again be described as a multiplicative uncertainty, as defined in eq. (4.9b), where the weighing function $w$ equals the radius of the disk. The uncertainty regions and their disk-shaped approximations around the Nyquist curve of an average plant are illustrated in fig. 4.4. In the next section, the lumped perturbation representing the parameter uncertainty within the trolley-controller system will be defined.

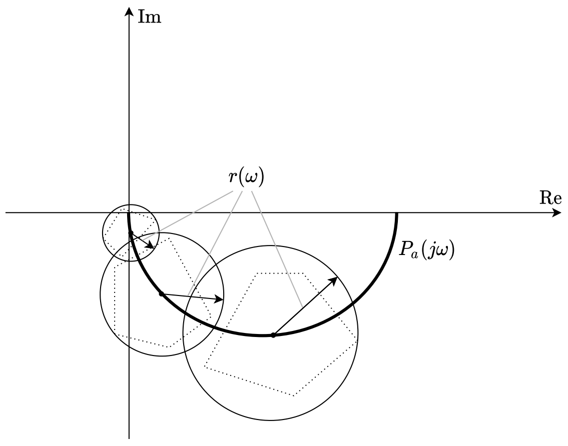


Figure 4.4: Uncertainty regions ····· around the Nyquist-curve ▬ of an average model $P_a$ and their disk-shaped approximations — with radius $r$ [32].

#### 4.3.3.4 Representation of the trolley-controller system uncertainty

As described in the previous section, it is desired to represent all uncertainty within the trolley-controller system by a disk-shaped perturbation with radius $r$ around an average plant model $P_a$. The uncertainty will be implemented in the trolley plant representation of the system's loop transfer function as a multiplicative uncertainty as defined in eq. (4.9b). Since, as described in section 4.3.1, not all dynamics of the trolley plant are included in the feedback loop, general plant representation $P$ is replaced by the trolley's loop dynamics $G$. The uncertain trolley plant will therefore be represented as:

$$G_p(s) = G_a\big(1 + w(s)\Delta(s)\big); \qquad ||\Delta(s)||_\infty \leq 1 \tag{4.10}$$

The transfer function of $G$ is derived to be eq. (4.8) in section 4.3.1:

$$G(s) = 2\,\frac{L_{WH}\cos(\beta)R_W^{-1} - \mu}{\mu m_{FT}\big(L_{WF}\sin(\beta) + W_{WF}\cos(\beta)\big) + L_{WH}\cos(\beta)(m_{FT} + 3m_W)} \tag{4.8 revisited}$$

where

$$m_{FT} = m_F + m_B; \qquad \beta = \sin^{-1}\left(\frac{L_H - R_W}{L_{WH}}\right).$$

**Average trolley-plant loop dynamics.** The trolley's average plant model $G_a$ represents the average behavior of its loop dynamics in the loop transfer function. An equation for $G_a$ is obtained by filling in the average values of uncertain parameters in transfer function $G$ provided by eq. (4.8), along with the fixed values of the remaining parameters. Since all its parameter values are now fixed, $G_a$ acts as a constant. The average values of uncertain parameters are defined in table 4.2 on page 44, the fixed values can be found in table 4.1 on page 37.

**Uncertainty weighing function.** Weighing function $w$ scales perturbation $\Delta$ to match the correct amount of uncertainty with respect to average trolley plant model $G_a$. The uncertainty within the trolley-controller system is represented by a disk-shaped perturbation, which is a simplification of the combined parameter uncertainty present within the system [32]. The disk's radius $r$ therefore will be used as a value for $w$. To ensure the disk covers the entire uncertainty region defined by the original lumped uncertainty, the radius of the disk-shaped area should be equal to the maximum deviation of the plant behavior with respect to its default value in the complex domain:

$$r(\omega) = \max_{G_P \in \Pi}\left|\frac{G_p(j\omega) - G_a}{G_a}\right| \tag{4.11}$$

where $r$ is the radius of the disk-shaped area, $\Pi$ is the uncertainty set, $G_p$ is any plant within $\Pi$ and $G_a$ is the average plant model. Since $G_p$ can be any plant within $\Pi$, the values of the uncertain parameters which are responsible for the uncertainty within $G_p$ are unknown. Therefore, an optimization is performed to solve eq. (4.11). The goal of the optimization is to find the uncertain parameter values of trolley bag mass $m_B$, trolley handle height $L_H$ and rolling friction coefficient $\mu$ such that the relative difference between a plant model $G_p$ from the uncertainty set and average model $G_a$ is maximal. Note that, although road inclination $\gamma$ also contains uncertainty, it is not used since it is not part of $G$.

***Optimization problem definition.*** A suitable definition of the optimization problem is derived by making some adjustments to eq. (4.11). An optimization problem is generally defined as

$$\min_{x\in\mathbb{R}^n} f(x); \qquad x = \big(x_1, x_2, ..., x_n\big)$$
$$\text{s.t. } g(x) = 0$$

where $f$ is the objective function, $g$ is a set of functions defining the constraints and $x$ is the vector containing all arguments of $f$ and $g$ [35]. The optimization problem provided by eq. (4.11) can initially be written as:

$$\max_{x\in\mathbb{R}^3} \left|\frac{G_p(x) - G_a}{G_a}\right|; \qquad x = \big(m_B, L_H, \mu\big) \tag{4.12a}$$

$$\begin{aligned} \text{s.t.} \quad 1.2 \leq{} & m_B \leq 21.2 \\ 0.661 \leq{} & L_H \leq 0.823 \\ 0.102 \leq{} & \mu \leq 0.22 \end{aligned} \tag{4.12b}$$

This optimization problem can be simplified by noting that the objective function is maximal where its numerator is maximal:

$$\arg\max_{x\in\mathbb{R}^3} \left|\frac{G_p(x) - G_a}{G_a}\right| = \arg\max_{x\in\mathbb{R}^3} |G_p(x) - G_a|$$

Optimization algorithms often search for the minimum of the objective function by default. For convenience, the current objective is therefore transformed into a minimization problem by negating the objective function:

$$\arg\max_{x\in\mathbb{R}^3} |G_p(x) - G_a| = \arg\min_{x\in\mathbb{R}^3} -|G_p(x) - G_a|$$

The arguments contained in $x$ all have constraints imposed on them, which complicates the optimization. Since these constraints are linear, they can be eliminated by integrating them in the objective function. This is done by replacing the constrained variables in the objective function by mappings, which map a new, unconstrained variable to the constrained space of the old variable:

$$\Omega : \widetilde{x} \to x \qquad \text{s.t.} \{x | x = \Omega(\widetilde{x}), \widetilde{x} \in \mathbb{R}^m\} = \{x | x \in \mathbb{R}^n, a \leq x \leq b\}$$

where $\Omega$ represents the mapping, $x$ is the original, constrained variable, $\widetilde{x}$ is a new, unconstrained variable and $a$ and $b$ are the limits of the constraint on $x$. The mapping ensures that any value of unconstrained variable $\widetilde{x}$ is transformed to another value which honors the original constraint on $x$; therefore, the explicit constraint $a \leq x \leq b$ can be removed. Variable $\widetilde{x}$ can now be used for solving the optimization problem. The optimal value for the original variable $x$ can subsequently be obtained by applying the same mapping $\Omega$ to the found optimum for $\widetilde{x}$.

The mapping for the elimination of the constraints in (4.12b) is defined as follows:

$$\Omega : \qquad x = x_a + \sin(\widetilde{x})\delta x; \qquad \delta x = x_{max} - x_a \tag{4.13}$$

where $x_a$ and $x_{max}$ are the average and maximum value of original argument $x$ according to its constraint. In the mapping, $\sin(\widetilde{x})$ keeps the value of $\widetilde{x}$ within bounds, whilst $\delta x$ scales its results to match the allowed range and $x_a$ provides the correct offset.

Implementing the above mentioned adaptations to the original formulation of eq. (4.12) and substituting $G_p(x)$ for its definition provided in eq. (4.8) results in the following description for the optimization problem:

$$\min_{\tilde{x}\in\mathbb{R}^3} - \left| 2 \, \frac{L_{WH}\cos(\beta)R_W^{-1} - \mu}{\mu m_{FT}\big(L_{WF}\sin(\beta) + W_{WF}\cos(\beta)\big) + L_{WH}\cos(\beta)(m_{FT} + 3m_W)} - G_a \right|; \tag{4.14}$$
$$\widetilde{x} = \left(\widetilde{m}_B, \widetilde{L}_H, \widetilde{\mu}\right)$$

where

$$m_{FT} = m_F + m_B; \qquad \beta = \sin^{-1}\left(\frac{L_H - R_W}{L_{WH}}\right);$$
$$m_B = m_{B,a} + \sin(\widetilde{m}_B)\delta m_B; \qquad \delta m_B = m_{B,max} - m_{B,a};$$
$$L_H = L_{H,a} + \sin(\widetilde{L}_H)\delta L_H; \qquad \delta L_H = L_{H,max} - L_{H,a};$$
$$\mu = \mu_a + \sin(\widetilde{\mu})\delta\mu; \qquad \delta\mu = \mu_{max} - \mu_a.$$

***Algorithm.*** The final optimization problem is unconstrained and has a nonlinear objective function. An analytic expression for the objective function is available, from which the gradient (Jacobian) can be analytically derived. Based on these characteristics, it is chosen to use the Davidon-Fletcher-Powell Quasi-Newton algorithm for performing the optimization, which uses a quadratic approximation of the objective function to find its minimum [35]. The optimization is performed multiple times using different initial argument values to increase the chance of finding the global minimum. The resulting optimum is compared to the results of a grid search for validation. The grid search computes the solution for radius $r$ for a number of argument combinations spread over their constrained range. If $r$ resulting from the optimization is indeed the maximum, all values for $r$ obtained in the grid search should be equal or lower than the value obtained from optimization.

***Result.*** The optimization results in a maximum $r$ of 1.41, found at the arguments' lower bounds $m_B = 1.2\,\text{kg}$, $L_H = 0.661\,\text{m}$ and $\mu = 0.102$. Therefore, weighing functions $w$ should be set to 1.41. As described briefly in section 4.3.3.3, an uncertainty weight $w$ larger than 1 can cause problems with the controllability of the system; When the transfer function of plant $G$ becomes 0, the feedback signal provided by the controller will not have any effect on the system output, making it uncontrollable. $w$ in multiplicative uncertainty denotes the magnitude of the perturbation with respect to the dynamic effects of the average plant. Thus, when $w$ becomes larger than 1, the effect of uncertainty on the system output can become larger than the influence of the plant itself[6]. In this situation there will be a chance that the combination of plant and uncertainty results in a transfer function of 0, which is undesired. Such a situation can also be seen graphically in fig. 4.4, where the smallest disk-shaped uncertainty region overlaps the origin of the Nyquist plot, indicating that the real transfer function of $P_a$ possibly is 0.

To ensure the trolley-controller system can never become uncontrollable due to uncertainty, some amount of uncertainty should be eliminated. This is no straightforward task, since trolley transfer function $G$ is nonlinear and therefore the effect of changing uncertain parameter values on its output is not self-explanatory. More research needs to be done on this topic to determine what measures can be taken to reduce the amount of uncertainty to a safe level. It is important to note that controllability and stability are two different things: controllability indicates the ability of the controller to assert influence on the system output, where stability relates to the self-reinforcement of the feedback signal when it travels through the loop. In fact, uncontrollability and instability are mutually exclusive.

[6]Note that the disk-shaped uncertainty region used to determine $w$ is a conservative estimate of the possible effects of uncertainty on the plant dynamics; It assumes that the plant's Nyquist curve can change the maximum amount in any direction in the complex plane, where in reality the possible deviation due to uncertainty might differ between directions. Therefore, it might be that in reality uncertainty cannot make the system uncontrollable, even if $w$ is larger than 1.

#### 4.3.3.5 Loop analysis of the trolley-controller system

Now that the system's uncertainty is taken into account as a perturbation to the trolley plant, conditions for robust stability and performance can be obtained by analyzing the system's closed-loop behavior. This is done using loop analysis, which studies the way sinusoidal signals propagate through the system's feedback loop. This propagation is characterized by the system's loop transfer function, which represents the relation between a signal before and after it passes through the feedback loop. For the purpose of this analysis the trolley-controller system loop transfer function will be defined with respect to the system output $A_W$, which represents the trolley's acceleration in frequency domain.

**Loop transfer function** The trolley-controller loop transfer function $L$ can be derived from fig. 4.2b, revisited on page 51, by following the tracing the signal through the loop from $A_W$ to $A_W$ and multiplying by $-1$ to account for negative feedback:

$$L = \frac{ABQ(s)}{A_nB_n - A_nB_nQ(s)} = \frac{GQ(s)}{G_n(1 - Q(s))}$$

where $AB = G$ and $A_nB_n = G_n$. The uncertainty present in the system is implemented as described in the previous section, by replacing the plant loop dynamics $G$ by its uncertain representation $G_p$ as defined in eq. (4.10):

$$\begin{aligned} L_p &= \frac{G_pQ(s)}{G_n(1 - Q(s))}; \\ &= \frac{G_a(1 + w\Delta)Q(s)}{G_n(1 - Q(s))}; \qquad ||\Delta(j\omega)||_\infty \leq 1 \\ &= \frac{G_aQ(s)}{G_n(1 - Q(s))}(1 + w\Delta); \\ &= L_a(1 + w\Delta); \end{aligned} \tag{4.15}$$

where $L_p$ and $L_a$ represent the loop transfer functions when the trolley's loop dynamics $G$ are replaced by respectively the uncertain trolley plant model $G_p$ or its average loop dynamics $G_a$. It can be seen that the uncertainty in loop transfer function $L_p$ also takes the form of a multiplicative uncertainty acting on the average loop transfer function $L_a$, similar to the definition of uncertain plant model $G_p$. Therefore, the uncertainty can once again represented by a disk centered around the Nyquist curve in the complex plane, as depicted in fig. 4.4.

**Robust stability** A system is stable as long as its loop transfer function never becomes smaller than $-1$; for all values below this, the system's feedback signal will amplify itself as it passes through the loop, causing the signal to blow up. Therefore, the Nyquist curve of a system's loop transfer function should never circle the point $-1$, called the 'critical point' [33]. For an uncertain system like the trolley-controller system, this is the case when the critical point never gets covered by the disk-shaped uncertainty region of $L_p$, provided that $L_a$ is stable.

From this notion, a requirement can be derived to ensure stability of the closed-loop trolley-controller system [32, 33]. Considering the Nyquist plot of uncertain loop transfer function $L_p$ provided in fig. 4.5, the distance between any point on the Nyquist curve of $L_a$ and critical point $-1$ is given by the difference between their positions:

$$d_{L_a \to \text{-}1} = |1 + L_a|$$

The magnitude of the uncertainty within uncertain loop transfer function $L_p$ can be obtained from eq. (4.15):

$$L_a(1 + w\Delta) = L_a + wL_a\Delta$$

Noting that the maximum absolute magnitude of perturbation $\Delta$ is 1, the radius of the disk-shaped uncertainty region $r$ becomes:

$$r = |wL_a\Delta|, \quad ||\Delta||_\infty = 1 \qquad \Rightarrow \qquad r = |wL_a|$$

Since the uncertainty region should not overlap the critical point, it can be stated that its radius may not be larger than the distance from $L_a$ to the critical point:

$$\begin{aligned} & |wL_a| < |1+L_a|, \ \forall\omega & \Rightarrow \quad & \left| w\frac{L_a}{1+L_a} \right| < 1, \ \forall\omega \\ \Rightarrow \quad & |wT_a| < 1, \ \forall\omega & \Rightarrow \quad & ||wT_a||_\infty < 1 \end{aligned} \tag{4.16}$$

where $T_a$ is the trolley-controller system's average complementary sensitivity function. Replacing $L_a$ with the trolley-controller system's average loop transfer function provides the stability criterion in its final form:

*Stability criterion:*
$$\left\| w\frac{G_aQ(s)}{G_n(1-Q(s))+G_aQ(s)} \right\|_\infty < 1 \tag{4.17}$$

where $G_a$ and $G_n$ represent the trolley plant's loop dynamics including respectively its average- and nominal parameter values, as defined by eq. (4.8): 'Transfer functions' (revisited on on page 46), $w$ represents the weighing function relating to the system's model uncertainty as determined in section 4.3.3.4: 'Representation of the trolley-controller system uncertainty', and $Q$ represents the yet to be defined Q-filter implemented in the controller. When the structure of Q is known, eq. (4.17) can be used to provide conditions for its arguments that ensure stability of the closed-loop system.

**Robust performance** Equation (4.16) shows that there is a relation between the weight function $w$, which is related to the amount of uncertainty present in the system, and the closed-loop system's complementary sensitivity function $T$. The complementary sensitivity function of the trolley-controller system is the transfer function providing the relation between the incoming disturbance and the corresponding estimation made by the controller, represented as $d$ and $\hat{d}$ in fig. 4.2b, respectively. The controller performs best when $T$ is 1, since then all disturbances acting on the trolley will be taken into account in the disturbance attenuation.

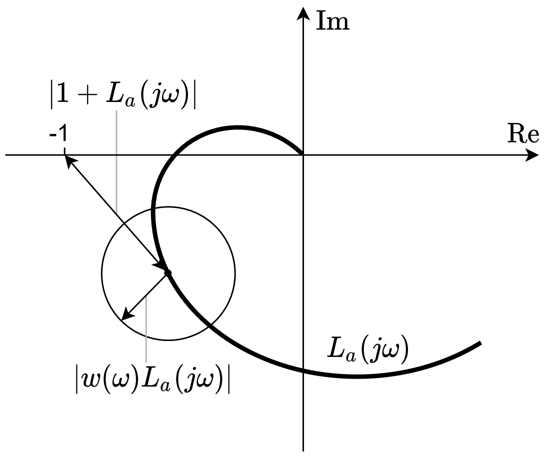


Figure 4.5: Uncertainty region — around the Nyquist curve of the loop transfer function ━ [32].

By reorganizing eq. (4.16), the following criterion for the controller performance can be obtained:

$$|T_a(j\omega)| < \frac{1}{|w|}, \ \forall\omega \tag{4.18}$$

The criterion shows that, in order to both ensure closed-loop stability and achieve optimal disturbance estimation, the magnitude of $w$ should stay below a value of 1. In section 4.3.3.4 is was determined that $w$ had to be set to 1.41 in order to include all considered uncertainty of the trolley-controller system, which

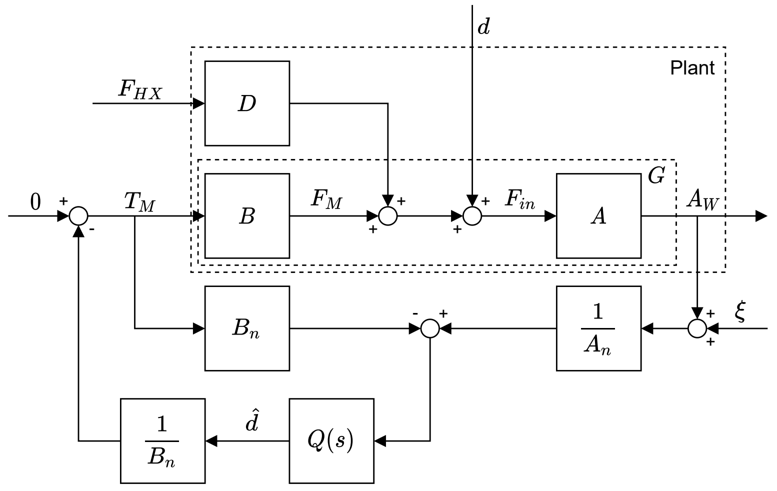


Figure 4.2b (revisited): Block scheme of the trolley-controller system structure

well exceeds this boundary; with this amount of uncertainty, $T_a$ cannot be larger than approximately 0.72. It can therefore be concluded that, in order to accomplish robustness of the trolley-controller system whilst considering the parameter uncertainties as defined in section 4.3.3.2, a significant compromise has to be made with respect to the controller's disturbance-attenuating capabilities.

Lowering $T$ reduces the controller's ability to completely attenuate all disturbances, since the Q-filter will now reduce the estimated disturbance signal before it is passed to the trolley's wheel motors. This is undesirable since it subtracts from the trolley's goal of reducing the burden of carrying groceries for elderly. Moreover, as described in section 4.3.3.4, a weighing function in excess of 1 could potentially cause the closed-loop system to become uncontrollable, rendering the assistive system useless. Further research should be done to gain a better understanding of the uncertainty present in the trolley-controller system, and to find ways to reduce it to safe levels. For the current controller design, an uncertainty weight function $w$ of less than 1 will be assumed in anticipation of this improved knowledge.

### 4.3.4 Q-filter design

In the previous sections, the trolley's closed-loop system properness, stability and performance are analyzed. Based on the results, a design for the controller's Q-filter can now be proposed.

#### 4.3.4.1 Filter order

The filter's order has an influence on the slope of its frequency response, as well as the lag it imposes on the filtered signal. The higher the filter's order, the sharper the transition between its pass- and stopband, but also the greater the lag it introduces in the output. A sharp slope is desired, since it will allow the Q-filter to better pass frequencies close to the cutoff frequency compared to a more shallow slope. On the other hand, minimal lag should be introduced in order to achieve the fastest controller response. between these contradictory requirements, the controller's reaction speed is deemed to be most critical to the overall controller performance; a large lag in controller response can result in unnatural dynamic behavior of the trolley system, which will reduce its predictability. It will furthermore cause the controller to respond slow to changing disturbances, thereby exposing the elderly user to its effects for a longer period of time. However, although unwanted disturbances with frequencies close to $\omega_c$ might not be attenuated as well as in higher-order filters, this is predicted to have far less impact on the controller's predictability and overall disturbance-attenuation capacity. This is especially true since the disturbances that get attenuated less accurately are of relatively high frequency, meaning that they are over relatively quickly. It is therefore chosen to use a Q-filter of first order:

$$Q(s) = \frac{a}{s+b} \tag{4.19}$$

where $a$ and $b$ are arbitrary real values, to be determined in the upcoming sections.

#### 4.3.4.2 Parameter values

Parameters $a$ and $b$ determine several characteristics of the Q-filter, such as its bandwidth and gain over different frequencies. Ideally the bandwidth of the DOB is infinitely wide; it would then be able to estimate disturbances over all frequencies and it would not impose any additional lag on the controller's response [29]. Its bandwidth is however limited by its desired functionality.

Besides ensuring stability of the trolley-controller system, as discussed in the previous section, the Q-filter's job is to remove any noise from the controller's estimated disturbance signal before it is passed to the trolley motors. On the other hand, the signal's component relating to any disturbances should not be effected. These two tasks provide a conflict, since both noise and disturbances are present at all frequencies.

Although disturbances are present on the entire frequency spectrum, the low-frequency disturbances like added mass and road inclination are assumed to be the main culprit of the increased physical burden the assistive shopping trolley aims to reduce. It is therefore desired that the gain of low-frequency signals is reduced as little as possible when passed through the Q-filter. Low-frequency transmittance by the Q-filter is accomplished by defining the filter's transfer function (4.19) to be 1 at zero-frequency:

$$\frac{a}{s+b} = 1, \quad s = 0 \qquad \Rightarrow \qquad a = b$$

$$\Rightarrow \qquad Q(s) = \frac{b}{s+b} \tag{4.20}$$

The remaining parameter $b$ is chosen based on the performed stability analysis and the characteristics of often-encountered disturbances; these topics will be discussed in the upcoming sections.

#### 4.3.4.3 Stability

In section 4.3.3.5, a criterion is determined for robust stability with respect to the trolley-controller system's loop transfer function, which can now be used to establish the stable range of Q-filter parameter $b$. To do so, eq. (4.20) is substituted in the stability criterion of eq. (4.17), and consequently the resulting equation is solved for $b$, as demonstrated below:

$$\left\| w \frac{G_a Q(s)}{G_n(1-Q(s)) + G_a Q(s)} \right\|_\infty < 1 \tag{4.17 revisited}$$

$$\Rightarrow \qquad \left\| w \frac{G_a \frac{b}{s+b}}{G_n + \frac{b}{s+b}(G_a - G_n)} \right\|_\infty < 1$$

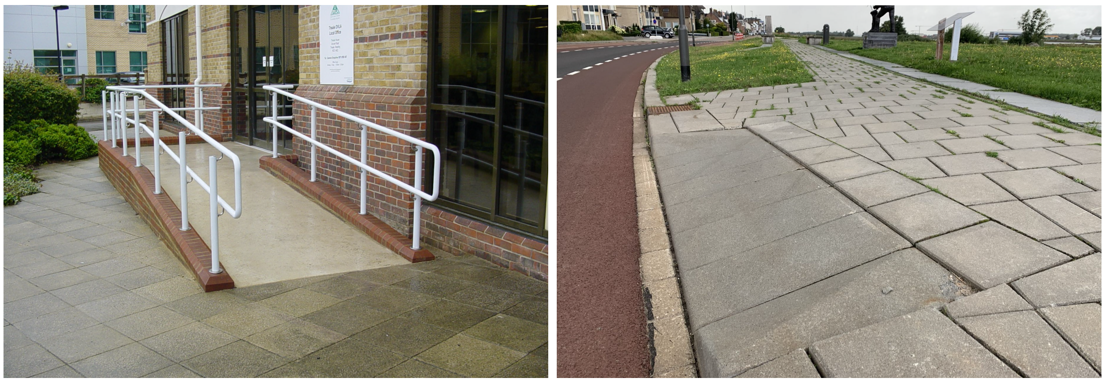
$$\Rightarrow \qquad \left\| w \frac{G_a b}{G_n s + G_a b} \right\|_\infty < 1$$

$$\Rightarrow \qquad \left| w \frac{G_a b}{G_n j\omega + G_a b} \right| < 1, \forall\ \omega$$

$$\Rightarrow \qquad \frac{|w G_a b|}{\sqrt{(G_n \omega)^2 + (G_a b)^2}} < 1, \forall\ \omega$$

The left-hand-side of this equation is largest when frequency $\omega$ is lowest. Therefore, if its inequality condition holds for $\omega = 0$, the trolley-controller system will be stable for all values of $\omega$:

$$\frac{|w G_a b|}{\sqrt{(G_n \cdot 0)^2 + (G_a b)^2}} < 1 \qquad \Rightarrow \qquad \left| \frac{w G_a b}{G_a b} \right| < 1 \qquad \Rightarrow \qquad |w| < 1 \tag{4.21}$$



(a) A ramp in front of a public building [36]. (b) A curb ramp providing access to the sidewalk [37].

Figure 4.6: Different types of ramps one can encounter during a trip to the store

Equation (4.21) shows that the system is stable in the presence of uncertainty when weighing function $w$ is lower than 1. Parameter $b$ is not part of this relation, therefore no restrictions have to be imposed on its range in order to ensure robust stability.

#### 4.3.4.4 Disturbance frequency analysis

The contradictory nature of disturbance attenuation and noise reduction requires a trade-off between the transmittance and cancellation of low- and high frequency signals. The disturbances most relevant to the goal of the assistive shopping trolley reside in the low-frequency region. Therefore, a compromise between reducing noise and passing through estimated disturbance signals is made by defining the Q-filter's cutoff frequency $\omega_c$ in a way that allows low-frequency disturbances to pass, whilst blocking any signal components of higher frequency.

To determine the desired cutoff frequency, some often-encountered disturbances are studied. The main goal of the assistive shopping trolley is to enable Dutch elderly to independently make the trip from their home to the grocery store, which is generally located within the same neighborhood. Therefore, the focus is on disturbances which are encountered during a walk here. It is assumed that the sidewalks are well-maintained; if not, elderly are at risk of tripping over any protrusions, which may deter them from traveling that route regardless.

Among the many disturbances that can be present during a trip to the grocery store, three disturbances are deemed to both occur regularly and require a reasonable amount of additional force to overcome. Firstly, groceries are added to the trolley along the way, increasing the trolley's weight. Secondly, many sidewalks and buildings have ramps, requiring additional force to overcome. Lastly, when crossing roads, curb ramps are often implemented to provide a smooth pathway from the sidewalk to the lower-positioned road, which can be difficult to cross as well. An example of both types of ramps can be seen in fig. 4.6.

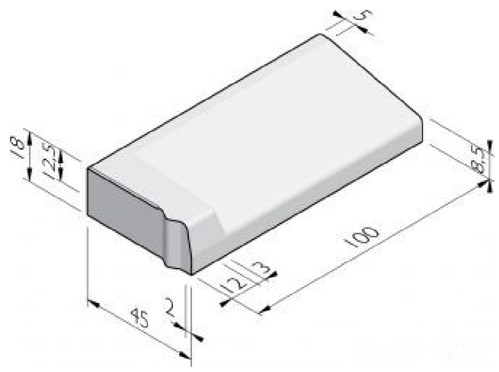


Figure 4.7: Dimensions of a steep curb ramp. [38].

Between these types of disturbances, curb ramps are expected to cause disturbances of the highest frequency, since they require the shortest distance to cross. Therefore, the value for the crossover frequency is based on this type of disturbance. The disturbance frequency is estimated by considering the time necessary to cross a curb ramp for an elderly person. Since we are interested in the highest frequency, the maximum elderly gait speed and a relatively steep curb ramp are examined.

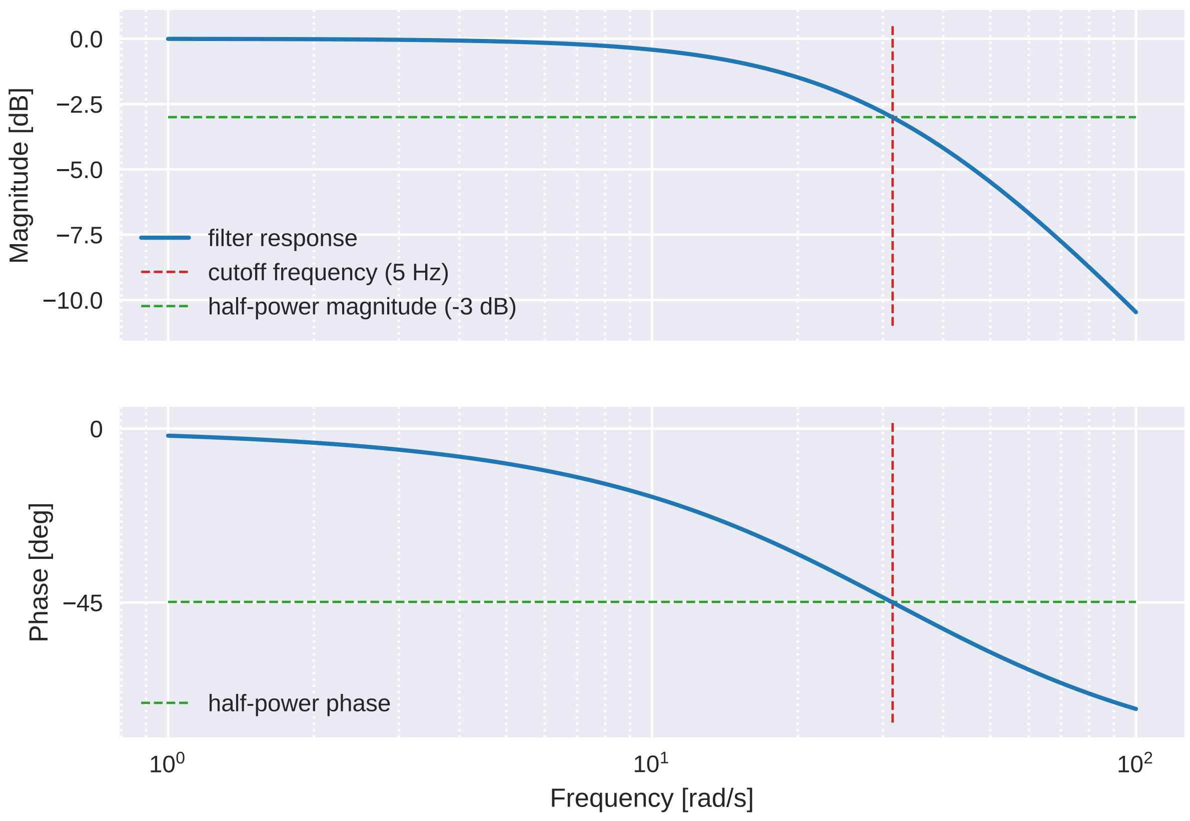


Figure 4.8: Bode plot showing the frequency response of the Q-filter defined as eq. (4.22).

The maximal gait speed of US elderly between 60 and 80 years of age with no mobility-related conditions is found to be around $1.9\,\mathrm{m\,s^{-1}}$ [39]. For the curb ramp, the ramp in fig. 4.7 is taken as reference [38], which is relatively steep in comparison to others available. Its sloping edge has a width of 40 cm and a height of 12.5 cm, resulting in a sloped surface of about 0.42 m in length. Combining the two results reveals that it takes roughly 0.2 sec for a fast-moving elderly person to cross a steep ramp, which corresponds to a disturbance frequency of a little under 5 Hz. The desired crossover frequency of the Q-filter is therefore 5 Hz, which equals $10\pi\,\mathrm{rad\,s^{-1}}$.

#### 4.3.4.5 Result

In first order low-pass filters like the current Q-filter in eq. (4.20), the cutoff frequency is equal to the filter parameter $b$. Therefore, the final form of the Q-filter is defined as:

$$\boxed{Q = \frac{10\pi}{s + 10\pi}} \tag{4.22}$$

The frequency response of the Q-filter is shown in a Bode plot in fig. 4.8. It can be seen that the filter's cutoff frequency lies at $5\,\mathrm{Hz} \approx 31\,\mathrm{rad\,s^{-1}}$, as designed. At the cutoff frequency the output has a phase lag of 45 deg, which is to be expected from a first order filter. It can thus be concluded that the design works properly.

#### 4.3.4.6 Discretization

The Q-filter is implemented in software as a discrete-time filter. First, the Q-filter transfer function is transformed to the discrete frequency domain using a zero-order hold method:

$$Q(s) = \frac{a}{s+b} = \frac{Y(s)}{U(s)} \quad \xrightarrow{\text{ZOH}} \quad \frac{Y(z)}{U(z)} = \frac{c}{z+d} = \frac{cz^{-1}}{1+dz^{-1}}$$

where $U(s)$ and $Y(s)$ are the system in- and output in continuous frequency domain, and $U(z)$ and $Y(z)$ their discretized versions. Subsequently the discrete-frequency system is converted to time domain using the inverse Z-transform:

$$\begin{aligned} Y(z) + d\,Y(z)\,z^{-1} = c\,U(z)\,z^{-1} \quad &\xrightarrow{Z} \quad y[n] + d\,y[n-1] = c\,u[n-1] \\ &\rightarrow \qquad\qquad y[n] = c\,u[n-1] - d\,y[n-1] \end{aligned} \tag{4.23}$$

The values of parameters $c$ and $d$ are determined automatically within the controller.

## 4.4 Summary

In the current chapter, the design of the assistive shopping trolley controller is formulated. The controller consists of two main parts; an inverse nominal model, which represents the trolley's desired dynamics, and a Q-filter, which adds robustness and noise reduction to the controller functionality.

The controller's inverse nominal model is based on the linearized dynamic model equation obtained in chapter 2: 'Trolley Dynamics', whilst its parameters are based on the trolley's nominal (desired) behavior. Insight regarding the system's stability and performance in the presence of model uncertainty, such as added weight and road inclination, is obtained through an $H_\infty$ analysis. In this analysis, the system's loop transfer function is analyzed for a set of plant models, which represent all possible deviations in dynamics due to uncertainty present in the system. For this purpose, a block scheme of the trolley-controller structure is created and the dynamic trolley model as constructed in chapter 2 is transformed from time- to frequency domain. Some adjustments are made to the trolley structure with respect to the general structure of plant-DOB systems to account for the different dynamic effects of the plant's human handle force- and motor torque input.

Different sources of model uncertainty within the trolley-controller system are identified, and the range over which they are expected to fluctuate is quantified based on literature. A suitable mathematical representation for the model uncertainty based on complex perturbations is constructed for their implementation in the system's loop transfer function. The perturbations are scaled to cover the effects of all model uncertainty; the corresponding weighing function is obtained via optimization.

The result shows that the dynamic effects of uncertainty are currently larger than the expected effects of the trolley dynamics itself. This imposes a risk of the system becoming uncontrollable when the combined dynamics of the average trolley and uncertain elements result in a transfer function of 0, therefore breaking the feedback loop. A subsequent analysis of the system's loop dynamics with respect to the controller performance yields a similar result; robust stability of the trolley-controller system can only be ensured by lowering the Q-filter's permeability for low frequency signal components. This is undesired, since most prominent disturbances are of slow-varying nature, and they will therefore not be fully attenuated when such measures are taken. These results lead to the conclusion that more research should be done regarding the present uncertainty acting on the trolley system, and future effort should be made to lower them to a safe level. For the design of the current controller, a safe measure is taken for the weighing function, in anticipation of changes in considered uncertainties and improved knowledge.

The design for the Q-filter is based on a first-order low-pass filter of the form $b/(s+b)$. Using the stability criterion obtained in the system's closed-loop analysis it is determined that, given an uncertainty weighing function below 1 for all frequencies, no restrictions have to be imposed on $b$ to ensure closed-loop stability. A compromise between disturbance attenuation and noise canceling is made in the determination of the value of $b$, since these tasks are contradictory. An appropriate value for $b$ is found by analyzing often-encountered disturbances during a trip to the store. It is presumed that curb ramps cause the highest-frequency deviations within this category, therefore the filter's crossover frequency is based on an estimation of such disturbances. The estimation results in a frequency of 5 Hz, i.e. $10\pi\,\mathrm{rad\,s^{-1}}$, therefore the transfer function of the Q-filter becomes $Q = 10\pi/(s + 10\pi)$.

# Chapter 5

# Implementation

In Chapter 4 the trolley controller is designed in theory, based on the design goal specified in section 1.1: ‘Objective’, namely to make the shopping trolley feel like it is always empty and on a horizontal road, and the trolley dynamics derived in chapter 2: ‘Trolley Dynamics’. The current chapter will describe the implementation of the controller design into a functioning program using Robotic Operating System (ROS).

This chapter will go into detail on the implementation of the controller in Robotic Operating System (ROS). First of all, a description of why ROS and its accompanying simulation program Gazebo were chosen is given in section 5.1. Subsequently, a short overview is given of ROS’s functionality in section 5.2. In section 5.4, the implementation of the controller design in ROS is detailed, and finally a summary of the most important points regarded in the current chapter is provided in section 5.6.

## 5.1 Why ROS and Gazebo

For the implementation of the controller it is decided to use the ‘Robotic Operating System (ROS)’ framework, combined with the ‘Gazebo’ dynamic robot simulation. ROS is the standard in the use of mobile robotics, and has a big community of users situated all over the world [40]. As an aspiring robotics engineer it is beneficial to be able to use ROS for robotics. Furthermore, DoBots is one of many companies and institutions that use ROS for the operation and control of their robots. Therefore the use of ROS and Gazebo for this thesis project will make this work particularly useful to them.

An advantage of ROS is that it uses a modular code architecture, making it easy to write modules, named ‘nodes’ in ROS terminology, each responsible for a separate part of the controller. Every module can be built using several programming languages according to the needs, adding to its flexibility. ROS can furthermore directly be used to communicate states and commands from and to the ‘Gazebo’ simulation, making for a powerful integration.

A useful feature of ROS is the portability of controllers that are written within this framework. There are ROS implementations for PC but also for ARM development boards in robots. This makes it possible to use the same software first in simulation and then on a real robot. When running on a robot, the simulation in-and outputs are simply replaced with measurements from sensors and actuator signals to the actual robot. This makes way for fast, efficient testing of ROS code without the hazards and additional time paired with implementation on a real robot. Moreover, this transition between simulation and real-world implementation can be done gradually by only replacing a few simulation elements by sensors/actuators at a time, increasing control over the implementation of the controller on actual hardware.

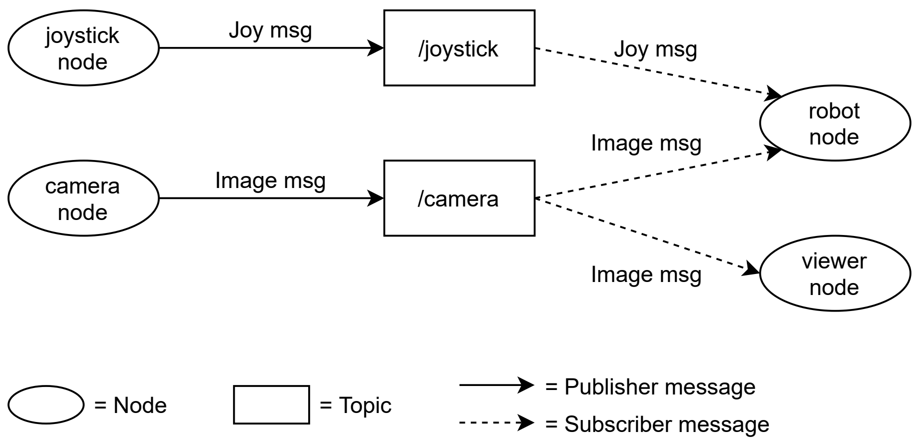


Figure 5.1: Example of a simple ROS scheme

## 5.2 The ROS framework

ROS is a flexible, open-source meta-operating system that aims to simplify the task of creating and managing robot behavior across a wide variety of robotic platforms [41]. ROS makes use of a decentralized system of processes, called nodes, which communicate with each other using the ROS communication infrastructure. This structure includes synchronous communication over channels called services, asynchronous streaming of data over topics, as well as data-storage on ROS's parameter server [42]. These nodes can run from different devices. They all have specific tasks and run independently from each other, making it easy to build complicated software structures and share and re-use nodes in other applications. For example, a node may be written to process and publish data from a specific sensor, contain code to provide a user-friendly interface for a hardware actuator, or perform any of the processing steps that a controller would perform between measurements and outputs. Compatible nodes for many types of hardware are readily available, (partially) eliminating the need for a detailed understanding of the hardware's workings in order to use it.

An example of a basic ROS-structure can be seen in fig. 5.1, which schematically depicts a situation where a robot is controlled using a joystick and camera. Both the joystick and camera have their own nodes, which process the raw sensor data and publish it on dedicated topics using messages. The robot has its own node as well, which reads data from the joystick and camera topics and uses it to control the robot. A viewer node is added, which can be used to monitor the information getting published on the camera topic. This example shows only a few nodes, but the modular structure of ROS allows making as many elements as needed. Therefore the ROS functionality is easily scalable and can be used for very simple, single-robot control as well as complicated, multirobot collaborative purposes [42].

A node in the ROS framework is a piece of software which can be written in a number of languages, wrapped with ROS functionality. It runs independently from other nodes. ROS nodes can operate in two different ways; they can spin, which means that they are continuously running their code in the fastest way possible, or they can they can cycle through their code at a fixed rate. The code itself can contain anything and does not have to be ROS-related, however to make use of the functionality of ROS it usually contains some form of ROS-related communication such as a publisher, subscriber or service, discussed below. Note that a single node can contain multiple of these elements, and can thus be a publisher as well as a subsriber simultaneously.

A subscriber node deals with incoming data on a topic by means of a callback function: when a node subscribes to a topic, it registers a callback function, or callback for short. The callback will be called as soon as a message is published on a topic, and will get as argument the contents of that message. It can then process this received data in any way desirable. Concurrently, the node that originally published the message on the topic can continue its own process. This makes the process of sending and receiving data asynchronous.

Data can be published on topics by nodes as well, such a node is called a publisher. Publishing data can be done either incidentally, for example when a node's callback function or service is called, but a node can also publish code in a fixed-rate loop. For example, a real-time controller might have a control loop running at 50 Hz, independent from when or at what rate messages come through on the topics the node is subscribed to. This is important for instance when a node's job is to process data on fixed intervals, which is the case for for example digital filters. If the filter step is done whenever some data is received, no guarantess can be given on the rate at which the filter is applied since that would depend on when the unfiltered messages come in. Instead, the callback of the unfiltered data topic can just store the data temporarily, until it is processed. The node can now apply a digital filter to the stored data at a fixed rate.

To summarize, ROS uses the following concepts, which will be refered to in the rest of this chapter:

| | |
|---|---|
| **Node** | a piece of software written in any language, that runs as an independent process. It can publish messages on a topic, subscribe to a topic to get input data from other nodes, and respond to or activate services that directly perform a specific task. |
| **Topic** | a communication pipeline between nodes. Nodes can publish data to a topic in the form of messages. Nodes that are subscribed to that topic will each get a copy of the published message as soon as it becomes available. Each topic only transports a single message type. |
| **Publisher** | a node that publishes messages to a topic. |
| **Subscriber** | a node that subscribes to a topic by registering a callback function to it, which will be called with the message contents as soon as a message is published. It can then process the contents of the message. |
| **Message** | a piece of data that is transmitted over a topic. Each message has a message type which describes the format of the contained data. This type can be simple like integer, float or string, or a more elaborate data structure tailored to a specific purpose. |
| **Service** | an advertised procedure call within a node: a node can call a service from another node to make it execute a certain function. The biggest difference with topics is that service calls happen synchronously: the node calling a service is halted until the service call finishes and the result is returned to the caller. |

## 5.3 The trolley controller structure in ROS

A structure is designed to implement the controller programmatically using the ROS framework. This section provides a general description of this structure and the different types of elements it contains. Section 5.4: 'Implementation of core nodes' provides an in-depth elaboration on the controller's key components.

### 5.3.1 Design requirements

In designing the structure, some requirements have to be taken into account. Firstly, it has to be taken into consideration that the controller's functionality will be tested in the Gazebo simulation environment, and therefore the structure should allow for easy integration with this environment. On the other hand, the controller is ultimately intended for use on a real trolley, and thus the eventual integration with real hardware needs to be taken into account as well. Finally, it is important to consider data analysis and debugging in the structure design.

### 5.3.2 General layout

The resulting structural overview is shown in fig. 5.2. The controller and its supplementary functions are subdivided into specific tasks, which are each implemented in their own node. This not only keeps the code organized, the topics used by the nodes to communicate between each other also make it easy to view and track intermittent data streams for analysis and debugging. Furthermore, partitioning the controller's functionality into separate building blocks facilitates the use of node-level unit- and integration tests to ensure the node's performance during the overall development of the control system implementation. A disadvantage of this approach is that the increase in communication steps accompanying the increasing number of consequential nodes adds a delay to the processing of the data, which might be problematic when becoming too large.

Other than the core controller design, other nodes are implemented to facilitate additional operations necesseray for the controller's implementation. These operations include communication with hardware or the simulation engine and operations required for data collection. From the overview in fig. 5.2, nodes with different types of functionality can be distinguished. Groups of nodes with corresponding functionality are indicated using different shades. Their purpose will be described generally in the paragraphs below.

**Core nodes.** The core nodes (shaded purple in fig. 5.2) contain the controller design as specified in chapter 4 and are therefore at the center of the structure's performance. The controller design is split into several separate nodes, with each node performing a dedicated subtask. Together, they implement the complete controller design as specified in chapter 4. The functions and implementation of the core nodes will be discussed in depth in section 5.4.

Most core nodes read their input data from sensor topics, indicated by the '`sensors/`' inclusion in the topic name. The input data published on these topics has already been processed by the reader nodes into a format which can directly be used for computation within the core nodes. The controller produces an output command which is published on a dedicated topic. This data will be processed by a writer node after which it is used by the trolley's wheel joint effort plugin, which powers the motors in the Gazebo simulation. Details about the operation of plugins will be discussed in chapter 6.

**Reader/writer nodes.** Reader- and writer nodes (shaded green in fig. 5.2) take care of the necessary processing of data between the controller and the simulation- or hardware components. Having this functionality separated from the controller's main components in dedicated nodes allows for easy switching between different sources of input data or actuator components. A different data source or target might produce/require data in a different format or frequency, and by performing the data conversion in a separate node the controller can easily accommodate a different source or target by swapping the node. The operation of the controller's core components therefore become independent on the format in which the raw in- and output data is provided/required. This ubiquity is especially convenient when switching from the controller implementation in simulation to the implementation on real hardware. Only the reader nodes should be replaced with (often readily available) nodes which are compatible with the used hardware in order for the controller to work.

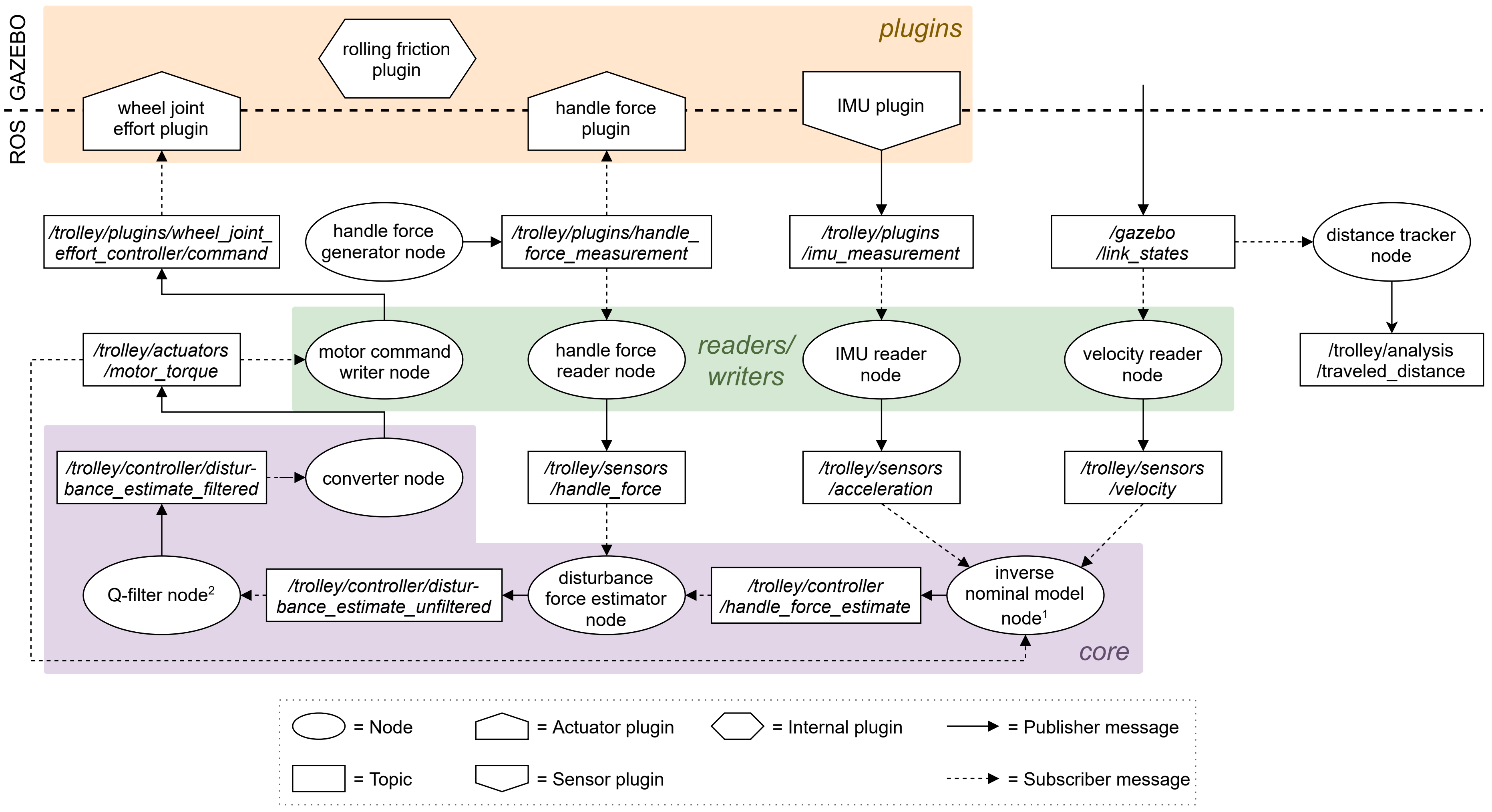

GAZEBO
ROS
plugins
rolling friction plugin
wheel joint effort plugin
handle force plugin
IMU plugin
/trolley/plugins/wheel_joint_effort_controller/command
handle force generator node
/trolley/plugins/handle_force_measurement
/trolley/plugins/imu_measurement
/gazebo/link_states
distance tracker node
/trolley/actuators/motor_torque
motor command writer node
handle force reader node
readers/writers
IMU reader node
velocity reader node
/trolley/analysis/traveled_distance
/trolley/controller/disturbance_estimate_filtered
converter node
/trolley/sensors/handle_force
/trolley/sensors/acceleration
/trolley/sensors/velocity
Q-filter node²
/trolley/controller/disturbance_estimate_unfiltered
disturbance force estimator node
/trolley/controller/handle_force_estimate
inverse nominal model node¹
core
= Node
= Actuator plugin
= Internal plugin
= Publisher message
= Topic
= Sensor plugin
= Subscriber message

Figure 5.2: Block diagram showing the structural overview of the controller in the ROS/Gazebo framework. The dotted line shows the interface between the ROS implementation of the controller and the Gazebo simulation.

[1]Inverse nominal model inverse nominal model equation 4.4 is implemented in ‘inverse nominal model node’.
[2]The Q-filter tranfer function (4.22) is implemented in ’Q-filter node’.

**Plugins.** Plugins (shaded orange in fig. 5.2) provide a way of communication between regular ROS nodes and the Gazebo simulation environment. They can be used to manipulate the simulation or extract data from it, for example by exerting a force on the simulated model or by monitoring the normal force between model and ground. Plugins can (but not necessary have to) include nodes, for example to publish extracted simulation data for regular nodes to use, or to advertize services which can be used to trigger an event in simulation by a regular node. A more detailed explanation regarding the plugins used for the controller implementation will be provided in chapter 6: 'Simulation'.

**Other nodes.** Some of the nodes specified in the structural overview in fig. 5.2 do not belong to any of the above described groups. These nodes (unshaded in fig. 5.2) provide functionality which is exclusively needed for the execution of the simulation and or analysis of the data.

A handle force generator node is implemented to provide simulated handle force data. In a real-life situation, the user would apply a force on the trolley handle. This force propells the trolley forward and its measurement is used as an input of the controller. No human is present in the simulation, thus the handle force sensor data is mimiced by the handle force generator node which publishes a predefined handle force profile on the handle force sensor topic. Furthermore, a node is implemented for keeping track of the trolley's traveled distance. The trolley's traveled distance is used for the analysis of simulation data, however this information is not directly available from the simulation or controller structure. Therefore, a node is implemented which integrates position data available from the simulation by default in order to obtain the traveled distance. This information is then published on a dedicated topic so it can easily be monitored and recorded. More information on these nodes is provided in chapter 6.

In the structural overview, the integration between ROS and Gazebo is shown using a bold dotted line. This integration and the corrensponding Gazebo-related plugins are explained in detail in chapter 6 as well.

## 5.4 Implementation of core nodes

The previous section discussed the general description of the total structure of controller and supplementary elements in the ROS framework. The current section provides a more in-depth explanation of the controller's core implementation in the ROS framework. Section 5.4.1 describes how the controller's functionality is partitioned into tasks, which will be implemented as separate nodes. Subsequently, in section 5.4.2 each node is described in detail, along with important parts of the source code produced to implement the nodes.

### 5.4.1 Core task division

As described in chapter 4, The controller design consists of two main elements; an inverse nominal model and a Q-filter. A recurrence of the controller schematic provided in this chapter can be found in fig. 4.1, revisited on the next page. The inverse nominal model makes an estimation of what the forces on the trolley should have been to produce the measured output given the desired trolley behavior, whilst the Q-filter is responsible for tweaking the controllers performance by filtering the resulting signal. Both are implemented in the form of a single function which is applied to the relevant data, and therefore they are each implemented as a single node, called '`inverse nominal model node`' and '`Q-filter node`', respectively.

Besides these two main elements, some smaller, intermediate step need to be performed. Firstly, the estimated force input acting on the trolley needs to be compared to the intended input. This is done by subtracting the applied motor torque signal and measured handle force from the estimated trolley input. Another dedicated node, called '`estimate disturbance force node`', is reserved for this. Finally, the filtered estimated disturbance force signal generated by the Q-filter must be transformed to torque. This processing step is implemented in a node called '`converter node`'. All mentioned nodes will be described in more detail in section 5.4.2.

In order to minimize the delay in the controller's response, it is desired to have the controller process the incoming data as soon as it becomes available. Therefore most nodes do not run independently, but rather respond directly to incoming data provided by other nodes. The only exception on this behavior

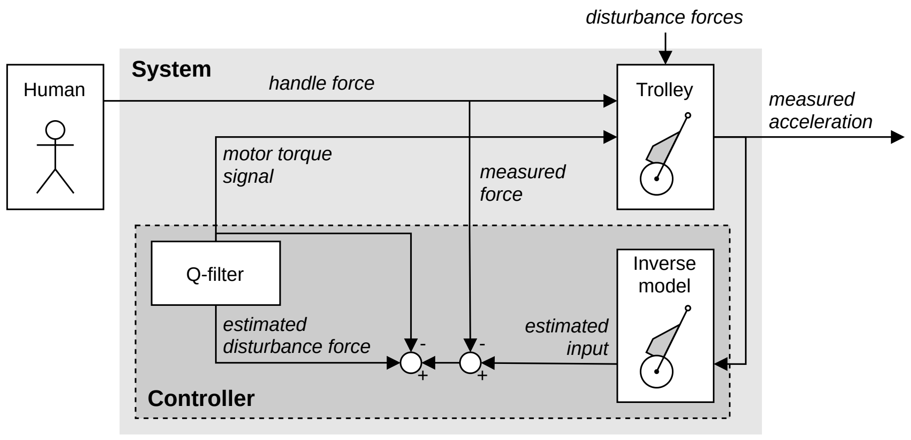


Figure 4.1 (revisited): Detailed overview of proposed controller structure for the assistive shopping trolley.

is the Q-filter node, which' internal discrete filter needs to be applied on data with a fixed interval in order to achieve an accurate result.

### 5.4.2 Implementation per node

This section describes the implementation of the controller's core nodes in the ROS framework in more detail. For every node, the way incoming data is processed is described in paragraph 'Data handling', where 'Computations' includes the implementation of the node's core computations. The latter paragraph also specifies whether these core computations are executed at a fixed rate or in reaction to another event. When applicable, any major additions and/or adjustments made to improve the node's functionality in practice are described in a paragraph 'Practical adjustments'.

The node descriptions include references to the relevant lines of code, provided in the section itself or in the appendix. The complete scripts are available at `https://gitlab.com/v.heusdens/assistive-shopping-trolley-control`.

#### 5.4.2.1 Inverse nominal model node

The inverse nominal model node contains the controller's inverse nominal model. It has the function of estimating the handle force applied on the trolley handle in the current situation when the trolley would behave as desired. The inverse nominal model equation has been composed in section 4.2: 'Inverse nominal model' as:

$$\begin{aligned} F_{HX} = \Bigg[ &\Big(\widetilde{\mu} m_{FT}\big(L_{WF}\sin(\beta) + W_{WF}\cos(\beta)\big) + L_{WH}(m_{FT} + 3m_W)\cos(\beta)\Big)\ddot{x}_W \\ &- 2\Big(L_{WH}\cos(\beta)R_W^{-1} - \widetilde{\mu}\Big)T_M + \Big(2L_{WH}\widetilde{\mu}\cos(\beta)\Big)F_{WG} \\ &+ \bigg(\Big(L_{WH}\cos(\beta) + W_{WF}\sin(\beta) - L_{WF}\cos(\beta)\Big)\widetilde{\mu} \\ &+ L_{WH}\cos(\beta)\bigg)F_{FG}\Bigg]\Big(L_{WH}\big(\widetilde{\mu}\sin(\beta) + \cos(\beta)\big)\Big)^{-1} \end{aligned} \tag{4.4 revisited}$$

where

$$m_{FT} = m_F + m_B; \qquad F_{FG} = m_{FT}g; \qquad F_{WG} = m_W g;$$

$$\beta = \sin^{-1}\left(\frac{L_H - R_W}{L_{WH}}\right); \qquad \widetilde{\mu} = \operatorname{sign}(\dot{x}_W)\mu.$$

The relevant snippets of the inverse nominal model node's source code are provided in snippet 5.1, in which the lines directly relevant to the implementation of the inverse nominal model equation are highlighted.

**Data handling.** The inverse nominal model node receives its velocity- and acceleration input data from sensors via reader nodes over the corresponding sensor topics. The motor torque input data is obtained from the '`actuators/motor_torque`' topic, which is the topic used by the controller to publish its output signal. The node contains subscribers for all these topics, which directly pass any data published on these topics to a corresponding callback function when it is received. The callback function saves the most recently received input data within the node, so it can be called upon for use in the computation when the inverse nominal model output is calculated. This method of saving data allows the node to operate independently from the nodes which provide the data; the inverse nominal model node can always access the saved input data for its computations, and the callback functions of the subscribers listening to the input topics will update the saved input values whenever more recent data becomes available. The node's three callback functions are defined in snippet 5.1, line 49 to 51, line 53 to 62 and line 64 to 68. The node attributes defined for saving the most recently received input data are defined at line 24 to 26.

**Computations.** The inverse nominal model Inverse nominal model equation 4.4 is stored in the controller's configuration file. Upon initiation of the node, the equation is loaded and saved as an attribute within the node. Its parameters are substituted by the nominal parameter values as determined in section 4.2.1: 'Nominal behavior' (table C.1). The equation is called whenever the node's acceleration input data callback function is triggered, and the resulting handle force estimation is then published on a dedicated '`handle force estimate`' topic. This node's update frequency is therefore dependent on the rate at which the acceleration input data is received. The initiation of the inverse nominal model equation is implemented on line 14 to 17 and line 37 to 42. The loaded instance of the inverse nominal model equation is called and the result is published on the output topic in a function defined on line 70 to 73, which is called by the accelaration callback function on line 68. The source code of the controller configuration file is included in section E.4, where the inverse nominal model equation is defined on line 6.

**Practical adjustments.** As mentioned in section 4.2.2, a velocity deadband should be implemented around 0 due to the nonlinear nature of rolling friction $T_{ROLL}$; this deadband will prevent errors in the handle force estimation due to the effect of velocity data noise on the direction of $T_{ROLL}$. This deadband is implemented in the inverse nominal model node's velocity data callback function; whenever the absolute value of the received velocity data is lower than the threshold of the deadband, the node's internally saved velocity will be updated to be 0 instead of the received value. The threshold value is set to be 0.01, which has been determined using trial-and-error. In the node's source code provided in 5.1 the velocity deadband threshold is loaded into the node at line 3 from the general configuration file (section E.1, line 3).

The velocity output data from the Gazebo simulation engine in which the controller is tested happens to have an offset, which varies between scenarios. This offset induces errors in the handle force estimate calculated by the node, which degrades the controller's performance. Moreover it caused the velocity measurement value to surpass the velocity deadband whilst the trolley was stationary in some instances. The resulting false disturbance estimate induced a correcting motor torque response by the controller, which caused the trolley to start moving independently of any force- or torque inputs. To compensate for the offset in velocity data, a velocity calibration function has been added to the inverse nominal model node which is called once at the start of the simulation, before the node starts running its main processes. The calibration function saves the velocity value provided by the simulation environment when the trolley is at a standstill. This calibration value is subsequently subtracted from the incoming velocity data before it is stored to compensate for the measured offset. The source code for this procedure can be found in snippet 5.2, line 44 to 47, where the calibration function is defined, and line 35, where it is called from within the node's main process.

```
1 VELOCITY_TOPIC = '/trolley/sensors/velocity'
2 ACCELERATION_TOPIC = '/trolley/sensors/acceleration'
3 FORCE_ESTIMATE_TOPIC = '/trolley/controller/handle_force_estimate'
```

```python
MOTOR_TORQUE_TOPIC = '/trolley/actuators/motor_torque'

class InverseNominalModelNode:

    def __init__(self):
        #@ load info from config files
        config_general = load_config('trolley_general')
        config_controller = load_config('trolley_controller')
        self.loop_delay = config_general['loop_delay']

        #@ load inverse nominal model equation
        self.n = ParamsNumeric()  #@ load parameter values
        self.force_model = self._load_model(config_controller['model_Fhx'])  #@ load inverse nominal
        ↪  model equation from config file
        self._inverse_nominal_model_eq = lambdify([self.n.d_xw, self.n.dd_xw, self.n.T_m],
        ↪  self.force_model)  #@ transform equation to usable function

        #@ init instances for saving incoming data
        self.velocity_deadband = config_general['velocity_deadband']  #@ configure velocity deadband
        self.velocity_offset = None  #@ used for storing velocity offset
        self.active = False  #@ trigger computation and publishing of data

        self.velocity = None  #@ used for storing most recent velocity input
        self.acceleration = None  #@ used for storing most recent acceleration input
        self.motor_torque = 0  #@ used for storing most recent motor torque input

        #^ init publishers/subscribers/services
        self.pub = rospy.Publisher(FORCE_ESTIMATE_TOPIC, Float64, queue_size=1)
        rospy.Subscriber(VELOCITY_TOPIC, Float64, self._callback_velocity)
        rospy.Subscriber(ACCELERATION_TOPIC, Float64, self._callback_acceleration)
        rospy.Subscriber(MOTOR_TORQUE_TOPIC, Float64, self._callback_motor_torque)

        #^ other
        self._calibration_velocity()

    def _load_model(self, model):
        """Load inverse nominal model equation using nominal parameter values"""
        self.n.set_params_nom()  #@ set params to their nominal values
        self.n.set_params_symbolic(['dd_xw', 'd_xw', 'T_m', 'F_hx', 'F_hy'])  #@ set symbolic params
        self.n.d_beta = 0  #@ set velocity of beta to 0
        return convert_string_to_sympy(model, self.n)  #@ convert config equation to usable function

    def _calibration_velocity(self):
        """Determines obtained velocity from gazebo during standstill"""
        hold_ros(0.2, self.loop_delay) #@ wait untill simulation is running properly
        self.velocity_offset = rospy.wait_for_message(VELOCITY_TOPIC, Float64).data

    def _callback_motor_torque(self, msg):
        """Store average motor torque."""
        self.motor_torque = msg.data

    def _callback_velocity(self, msg):
        """Store velocity measurement after deadbanding the velocity value."""
        if self.velocity_offset is None:  #@ do nothing before velocity offset is set
            return
        velocity = msg.data - self.velocity_offset  #@ apply offset

        #@ apply velocity deadband
        if abs(velocity) < self.velocity_deadband:
            velocity = 0
        self.velocity = velocity
```

```
63
64      def _callback_acceleration(self, msg):
65          """Store acceleration measurement and process data."""
66          self.acceleration = msg.data
67          if self.active:  #@ perform computations if node is active
68              self._process()  #@ call main process
69
70      def _process(self):
71          """Compute handle force estimate and publish on output topic."""
72          handle_force_est = self._inverse_nominal_model_eq(self.velocity, self.acceleration,
            ↪  self.motor_torque)  #@ compute output data
73          self.pub.publish(handle_force_est)  #@ publish result on output topic
74
75      def spin(self):
76          """Main loop"""
77          while not rospy.is_shutdown():
78              if self.acceleration is not None and self.velocity is not None:  #@ wait until all sensor
                ↪  data is received
79                  self.active = True
80              rospy.spin()
```

Snippet 5.1: The relevant Python code of the controller's inverse nominal model node, which implements inverse nominal model equation 4.4 as designed in section 4.2.

#### 5.4.2.2 Disturbance force estimator node

The disturbance force estimator node performs the task of comparing the measured and estimated handle force, and publishing the resulting difference on the dedicated 'disturbance estimate unfiltered' topic. The disturbance force estimator node's relevant source code is included in snippet 5.2, where the elements directly related to the computation of the disturbance estimate are highlighted.

**Data handling.** The measured- and estimated handle force input data is obtained using subscribers, which fetch the data from the corresponding '`/controller/handle_force_estimate`' and '`/sensors/handle_force`' topics and process them through their callback functions. In these callback functions, the input data of each type is saved into an attribute of the node, so it can be accessed at all times by other node elements. The callbacks for the handle force measurement- and estimation input data are defined at snippet 5.2, line 21 & 23, and line 25 to 28, respectively. The node attributes where the input data is saved are defined at line 13 and 14.

**Computations.** The disturbance force estimator node computes the difference between the node's handle force estimate- and handle force measurement values by subtraction; the difference between them is considered to be the disturbance force. The computation is implemented in a function called `process()` which is called via the handle force estimate topic's callback function. The callback is triggered as soon as new data arrives on the corresponding topic, therefore triggering the computation from here ensures that the most up-to-date handle force estimation data is processed with minimal delay. Right after the disturbance estimate is computed, it is published on the node's output topic. The `process()` function is defined on line 30 to 33, it is called from within the handle force estimate callback on line 28.

```
1 FORCE_MEASUREMENT_TOPIC = '/trolley/sensors/handle_force'
2 FORCE_ESTIMATE_TOPIC = '/trolley/controller/handle_force_estimate'
3 PUBLISHER_TOPIC = '/trolley/controller/disturbance_estimate_unfiltered'
4
5 class EstimateDisturbanceForceNode:
6
```

```python
    def __init__(self):
        #@ load info from config files
        config_general = load_config('trolley_general')
        self.loop_delay = config_general['loop_delay']

        #@ init instances for saving incoming data
        self.handle_force_measurement = 0
        self.handle_force_estimate = 0

        #^ init publishers/subscribers/services
        self.pub = rospy.Publisher(PUBLISHER_TOPIC, Float64, queue_size=1)
        rospy.Subscriber(FORCE_MEASUREMENT_TOPIC, Float64, self._callback_handle_force_measurement)
        rospy.Subscriber(FORCE_ESTIMATE_TOPIC, Float64, self._callback_handle_force_estimate)

    def _callback_handle_force_measurement(self, msg):
        """Store handle force measurement."""
        self.handle_force_measurement = msg.data

    def _callback_handle_force_estimate(self, msg):
        """Store handle force estimate and process data."""
        self.handle_force_estimate = msg.data
        self._process()

    def _process(self):
        """Compute \disturbance force estimate and publish on output topic."""
        disturbance_force_estimate = self.handle_force_estimate - self.handle_force_measurement  #@
        ↪  compute output data
        self.pub.publish(disturbance_force_estimate)  #@ publish result on output topic

    def spin(self):
        """Main loop"""
        while not rospy.is_shutdown():
            rospy.spin()
```

Snippet 5.2: The relevant Python code of the controller's disturbance force estimator node, which calculates the unfiltered disturbance force estimation given the handle force estimation and -measurement.

#### 5.4.2.3 Q-filter node

The Q-filter node is responsible for filtering the estimated disturbance force data. It implements the discrete Q-filter designed in section 4.3.4:

$$y[n] = c\,u[n-1] - d\,y[n-1] \tag{4.23 revisited}$$

The relevant source code of the Q-filter node's implementation is provided in snippet 5.3, where the elements directly related to the computation of the disturbance estimate are highlighted.

**Data handling.** The Q-filter node's only input data comes from the topic containing the disturbance estimation provided by the disturbance force estimator node; the '`disturbance_estimate_unfiltered`' topic depicted in fig. 5.2. Like in the previously discussed nodes, the callback of the subscriber monitoring this topic saves the received data in an attribute[7]within the node, where it can be accessed at any time by other node elements. Besides this data, the Q-filter's output data is saved in an attribute of the node as well, as it is an input for the Q-filter's computation in the next iteration. The unfiltered disturbance estimate callback is defined in snippet 5.3, line 42 to 44. The attributes in which this data and the Q-filter output is saved are defined in line 34 and 35, respectively.

**Computations.** The discrete Q-filter provided by eq. (4.23) is saved as an attribute of the Q-filter node in the form of a function which takes the node's in- and outputs of the previous time step and returns the current output. The initialization of this function happens when the node itself is initialized. First, the Q-filter's continuous-time numerator and denominator are loaded from the controller's configuration file, after which they are transformed to discrete time using a specialized function from Python's 'SciPy' library. The discrete Q-filter elements are then combined with symbolic input variables to obtain the Q-filter function as described above. The function loading the Q-filter's continuous-time numerator and denominator from the controller configuration file is defined on snippet 5.3, line 4 to 11, where the function converting the numerator and denominator to the final discrete-time function can be found on line 13 to 20. Both functions are called from within the node's initialization method, in line 30 and 31 respectively. The controller configuration file can be found in section E.4, where the Q-filter's numerator and denominator are defined on line 9 and 10, respectively.

In order to yield correct results when applying the discrete Q-filter to the unfiltered disturbance estimate data, the computation needs to happen in fixed intervals. Therefore, the core process function in which the Q-filter is used is called from within the node's main loop, which is set to run at a fixed frequency. Note that this is in opposition to the behavior of the other core nodes, in which the core processes are triggered by the arrival of new data. Furthermore, the main process function takes care of storing the filtered disturbance force estimate data outputted by the Q-filter, as well as publishing the data on the node's output topic ('`disturbance_estimate_filtered`' in fig. 5.2). The node's process function, in which the Q-filter is used, is called from within the main loop at snippet 5.3, line 56. The process function itself is defined on line 46 to 50.

```python
1  PUBLISHER_TOPIC = '/trolley/controller/disturbance_estimate_filtered'
2  SUBSCRIBER_TOPIC = '/trolley/controller/disturbance_estimate_unfiltered'
3
4  def load_q_filter(config='trolley_controller'):
5      """Get controller q_filter numerator and denominator from 'config_controller'."""
6      config_controller = load_config(config)  #@ load configuration file
7
8      #@ transform numerator and denominator to Python-code
9      num = [float(convert_string_to_sympy(x, np.pi, 'pi')) for x in config_controller['numerator']]
10     denom = [float(convert_string_to_sympy(x, np.pi, 'pi')) for x in
       ↪  config_controller['denominator']]
11     return num, denom
12
13 def generate_q_filter_discrete(num_cont, denom_cont, dt):
14     """Returns difference equation of 1st order continuous-time transfer function. Uses 0-order hold
       ↪  to transform to discrete time. Output is in the form of y[n] = c u[n-1] - d y[n-1]."""
15     num_discr, denom_discr, _ = scp.cont2discrete((num_cont, denom_cont), dt)  #@ to discrete-time
16     num_discr = num_discr.flatten()[-1]
17     denom_discr = denom_discr.flatten()[-1]
18
19     y_cur = lambda u_prev, y_prev: num_discr * u_prev - denom_discr * y_prev  #@ create Q-filter
20     return y_cur
```

[7]Unlike in the other nodes, the data in the Q-filter node is saved in a list-like attribute type called a 'deque'. This is done so that the attribute can be easily adjusted to hold more than one piece of data, in case a higher order Q-filter is implemented.

```python

class QFilterNode:

    def __init__(self):
        #^ init params
        config_controller = load_config('trolley_controller')
        self.rate = config_controller['update_rate']

        #@ initiate Q-filter
        q_num, q_denom = load_q_filter()  #@ get Q-filter fraction from config file
        self.q_filter = generate_q_filter_discrete(q_num, q_denom, 1. / self.rate)  #@ transform to discrete-time function in the form of y[n] = c u[n-1] - d y[n-1]

        history_size = 1
        self.input_history = deque(np.zeros(history_size), maxlen=history_size)  #@ used for storing previous input
        self.output_history = deque(np.zeros(history_size), maxlen=history_size)  #@ used for storing previous output

        #^ init publishers/subscribers
        self.pub = rospy.Publisher(PUBLISHER_TOPIC, Float64, queue_size=1)
        rospy.Service(SERVICE_TOPIC, EmptyTrigger, self._srv_handle)
        rospy.Subscriber(SUBSCRIBER_TOPIC, Float64, self._callback)

    def _callback(self, msg):
        """Store incoming \disturbance estimate data."""
        self.input_history.append(msg.data)

    def _process(self):
        """Compute filtered \disturbance data, publish result on output topic and store for next iteration."""
        dist_filtered = self.q_filter(self.input_history[-1], self.output_history[-1])  #@ compute output data
        self.pub.publish(dist_filtered)  #@ publish result on output topic
        self.output_history.append(dist_filtered)  #@ store result

    def spin(self):
        """Main loop"""
        r = rospy.Rate(self.rate)  #@ set loop frequency
        while not rospy.is_shutdown():
            self._process()  #@ call main process
            r.sleep()  #@ hold until next iteration should begin
```

Snippet 5.3: The relevant Python code of the controller's Q-filter node, which applies discrete Q-filter eq. (4.23) to the unfiltered disturbance estimatation data.

#### 5.4.2.4 Converter node

The final element of the controller is the converter node, which converts the filtered disturbance-force estimation data to the required motor torque that should be applied per wheel to compensate for the disturbances. This data is later transformed to a motor command by the motor command writer node. The relevant source code snippets of this node are provided in snippet 5.4, in which the lines relating to the node's computations have been highlighted.

**Data handling.** The converter node takes the filtered disturbance force estimation data outputted by the Q-filter node as its input. This data is sent to a callback function by the node's subscriber whenever it becomes available on the corresponding '`disturbance_estimate_filtered`' topic. The proper convertion of the input data to motor torque is not dependent on any fixed rate, and every incoming data point is converted to motor torque independently of any other variable data. Therefore, unlike in the other core nodes, the data is not saved in an attribute of the node but rather directly processed from within the callback. The callback is defined in snippet 5.4, line 15 to 20.

**Computations.** The convertion of the filtered disturbance force estimation is fairly straightforward; the estimated force is first converted to torque and then divided in half to arrive at the motor torque per wheel. The conversion of force to torque requires the wheel radius, which is loaded from a configuration file containing the trolley parameters. Since the conversion is performed in the node's callback, the node's update frequency depends on the rate at which input data is received. The conversion of the input data from total force to torque per wheel is performed in line 18 and 19. The configuration file containing the trolley parameters is included in section E.5, where the wheel radius is defined on line 18.

```
 1 SUBSCRIBER_TOPIC = '/trolley/controller/disturbance_estimate_filtered'
 2 PUBLISHER_TOPIC = '/trolley/actuators/motor_torque'
 3
 4 class ConverterNode:
 5
 6     def __init__(self):
 7         #^ init params
 8         model_config = load_config('trolley_description')
 9         self.R_w = model_config['wheel_radius']  #@ configure wheel radius
10
11         #^ init publishers/subscribers/services
12         self.pub = rospy.Publisher(PUBLISHER_TOPIC, Float64, queue_size=1)
13         rospy.Subscriber(SUBSCRIBER_TOPIC, Float64, self._callback)
14
15     def _callback(self, msg):
16         """Compute motor torque per wheel."""
17         force = msg.data
18         torque_tot = -self.R_w * force  #@ compute total torque
19         torque_per_wheel = 0.5 * torque_tot  #@ compute torque per wheel
20         self.pub.publish(torque_per_wheel)  #@ publish result on output topic
21
22     def spin(self):
23         while not rospy.is_shutdown():
24             rospy.spin()
```

Snippet 5.4: The relevant Python code of the controller's converter node, which converts the filtered disturbance force estimation data to the motor torque that should be applied per wheel to compensate for the disturbances.

## 5.5 Verification

During the course of this thesis, there have been multiple attempts to create the controller software in ROS and implement the simulation in Gazebo. This chapter shows the final attempt, which succeeded to run a simulation with a controller that functions correctly. However, in previous attempts, problems were encountered ranging from inaccurate simulation, noise in the measurements, and a controller not performing as expected.

In order to trust the simulation and controller results, and find the reasons for the aforementioned problems, verification procedures were implemented in the form of automated software tests. These include unit tests, that test small units of code without any ROS-specific functionality, and integration tests, which test not only the general code within the nodes but also the node's integration with ROS and Gazebo. These tests are performed using ROS's 'rosunit' and 'rostest' Python modules respectively, which provide a wrapper for Python's built-in 'unittest' module so it can be used within the ROS framework[8]. Integration take a part of the computations and test if the implemented ROS node reflects the relation between inputs and outputs correctly. This is done by running the to-be-tested node along with a node that will perform the test. The to-be-tested node is called the 'node under test' in this case, while the node that performs the test is called the 'testing node'.

Integration tests work as follows:

1. Design an input to a node, i.e. a fake measurement or intermediate result. This test input should yield an easily verifiable output in the node under test.
2. A testing node publishes this input on the topic that the node under test subscribes to.
3. The node under test calculates its response based on this input and publishes that response on its output topic.
4. The testing node subscribes to that output topic, gathering the outputs of the node under test.
5. The testing node verifies whether these outputs are as expected based on the inputs.

This automated approach verifies both the integration with ROS and the underlying computations. Note that every test is performed in an isolated environment, making it possible to test just the node under test without interference from any other running nodes.

For each of the nodes specified in the structural overview provided in fig. 5.2, one or more of these tests were designed and implemented. Tests are implemented to check the functionality of groups of nodes as well. In total, a number of 74 tests were created for these purposes. The simplest tests are those for the reader nodes, which currently only have the task of passing along data, while the tests for the correct functioning of Gazebo elements are the most complex due to the relatively intricate and non-transparent workings of the Gazebo simulation environment.

As an example of the testing procedures, the Q-filter-node test is described here. The Q-filter node is tested by verifying the relationship between successive output messages by applying a step function on its input topic. At first, the Q-filter output is equal to its initial state of 0. At 50 Hz, it will keep outputting this "filtered" value of 0, which is the first thing that the testing node verifies. Then, the testing node will publish a fake "unfiltered" input value of 1. This is picked up by the node under test, which will effectively start outputting the step response of the filter. The relationship between successive output messages of the filter is checked by comparing the output to expectation based on the applied input value and the difference equation that the node under test is supposed to implement. The relevant code involved in performing this test is included in appendix F for illustrative purposes.

[8]Although the unit tests do not test any ROS-specific code, the 'rosunit' module enables the automated calling of the test script and the proper logging of test data from within the ROS framework.

## 5.6 Summary

The current chapter describes the implementation of the controller design, obtained in chapter 4: 'Controller Design'. This is done using the Robotic Operating System (ROS) framework; a flexible meta-operating system specialized for the control of robotic devices across a wide variety of robotic platforms. ROS implements a decentralized system of processes, called 'nodes', which operate independently from each other. Nodes can be written in various programming languages. Communication between them happens mainly through topics, on which messages containing various types of data can be published. The messages can subsequently be read by any node listening to that topic. Nodes can directly respond to published data or they can run at a fixed frequency, depending on their required functionality.

For implementation in ROS, the controller design and additionally required functionality are subdivided into separate tasks, which are implemented as separate nodes. This allows for easy tracking of data over the intermittent topics, as well as convenient testing of specific elements through unit- and integration tests. In the resulting structure (fig. 5.2), four types of nodes can be distinguished. The functionality of the controller design provided in chapter 4 is divided over four nodes, which are labeled the 'core' nodes. The controller's inverse nominal model as formulated in section 4.2 is implemented in its own node, as is the Q-filter specified in section 4.3. The remaining two core nodes contain the disturbance force estimation and the conversion of the filtered disturbance estimate to a required torque per wheel motor for attenuation of the disturbance.

Additional 'reader' and 'writer' nodes (considered as a single group) are implemented to facilitate the conversion of data between the controller and hardware- or simulation elements. Every communication channel has its own reader or writer node, which allows for easy adjustment of the conversion whenever a different element requiring a different data format is implemented.

A 'plugin' group contains nodes which handle the manipulation of the simulation. Dedicated plugins are implemented for the control of the trolley's wheel motors, the application of the imposed force on the trolley handle, the acquisition of acceleration data and the application of a variable rolling friction torque on the trolley wheels. The velocity measurement used in the controller does not require an additional plugin, since this data is made available by the simulation engine per default. More information regarding these type of nodes is provided in chapter 6 on the facing page.

The last group of nodes is formed by the two nodes that do not belong to any of the above-mentioned categories. These nodes provide functionality for the execution of the simulations and the analysis of the resulting data.

In principle, nodes are triggered to perform their computations by the arrival of new data on their input topics, to induce as little delay as possible. The only exception to this rule is the node containing the Q-filter, which needs to be executed at a fixed rate for the discrete filter to work correctly.

During the implementation of the controller structure in ROS, several complications have arisen, hindering its proper operation. This chapter's final section describes the resulting implementation of automated unit- and integration tests for verification of the implemented controller's functionality. A total of 74 test were written, checking the proper functioning of all separate nodes, as well as specific elements within the nodes and the functionality of several nodes combined. As an example, the tests performed for the verification of the Q-filter node are examined, which test the functioning of the node as a whole as well as the Q-filter function explicitly.

# Chapter 6

# Simulation

In the introduction of this thesis, the goal is set to reduce the physical burden of carying groceries through means of an assistive shopping trolley, with the intention of increasing elderly mobility. To achieve this goal, a controller for the assistive shopping trolley is designed with the intention of making the trolley feel as if it is always empty and on a flat road, independently of any changing system parameters and environmental variables (also called 'disturbances'). In the previous chapters, the design and implementation of this controller has been detailed. Now the question arises: how well does the controller accomplish the goal that has been set for it? An attempt is made to answer this question by putting the controller to the test in a series of simulations.

The current chapter provides an elaboration on the design, implementation and results of these simulations. Firstly, in section 6.1, the theoretical setup of the simulations is described, where section 6.2 deals with their implementation in Robotic Operating System (ROS) and Gazebo. To conclude, section 6.3 discusses the obtained results and their implications.

## 6.1 Setup

As mentioned before, the controller is designed to attenuate disturbances acting on the assistive shopping trolley, always making it feel like it is empty and on a flat road. This is called the 'nominal' behavior, and the corresponding situation is accordingly called the nominal situation. Any deviation from the nominal behavior can be considered an error in the controller performance. The controller performance can therefore be determined by comparing the controlled assistive shopping trolley behavior with its nominal counterpart. This is done in several scenarios, where different types of disturbances are encountered.

The quantitative results obtained from the controller performance analysis by themselves are not very meaningful, since they do not provide any information about the improvement the controller provides with respect to the performance of the current solutions. Therefore, for every scenario the behavior of a regular, non-assistive shopping trolley is used as a reference to put the controller results into perspective. To make a clear distinction between the two, the assistive shopping trolley will be called the 'controlled' trolley, whereas the regular, non-assistive shopping trolley will be referred to as the'uncontrolled' trolley.

### 6.1.1 Scenarios

To properly analyze the controller performance, its behavior is tested whilst different disturbances are applied. During a trip to the grocery store one often encounters small slopes such as (curb) ramps, which can be seen as a disturbance in road inclination. Furthermore, the trolley's weight can vary with up to 20 kg [8] due to groceries that are added during the trip, presenting a disturbance in trolley mass. To test the attenuation of these types of disturbances by the controller, they are each implemented in a simulation scenario; in one scenario a slope of the maximum allowed angle [15] is added, in the other the trolley will carry an additional 20 kg of weight.

Equally important is the controller behavior when no disturbances are present; The behavior of the controlled trolley in this situation is an indication of how well the controller is able to predict the desired

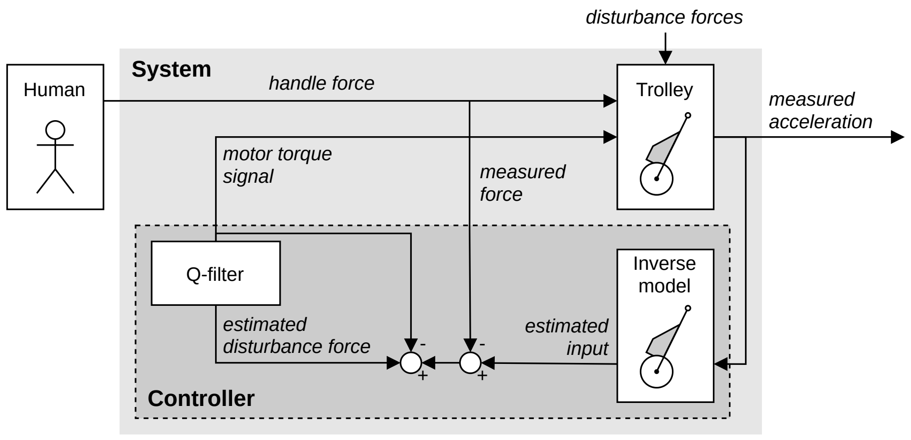


Figure 4.1 (revisited): Detailed overview of proposed controller structure for the assistive shopping trolley.

behavior. Ideally, the desired and actual behavior for this scenario are the same. Differences between the two can be caused by inconsistencies between the controller's nominal model and the real desired behavior. Therefore this scenario is simulated as well.

### 6.1.2 Measures

This section discusses the measures used to obtain and depict the simulation results.

#### 6.1.2.1 System in- and outputs

The relation between in- and output of a system is descriptive of it's behavior. During the tests, the predefined parameter is called the 'input' measure, since this can be seen as the provided input to the system. The 'output' measure is defined as the parameter which is chosen to represent the corresponding reaction of the system on the given input. As mentioned in chapter 2: 'Trolley Dynamics', the in- and output of the trolley system are set to be the force applied on the trolley handle ($F_{HX}$) and the trolley's acceleration ($\ddot{x}_W$). An overview of the trolley system is provided in fig. 4.1, revisited on the current page. Since the controller attempts to control the system's behavior, the trolley's handle force and -acceleration will be used as in- and output measures in the analysis of the controller performance.

Initially, it was planned to perform experiments in addition to the simulations in order to analyze the difference between simulated and experimental results. Because of this, it was of the essence that the chosen input and output measures for the simulation were also suitable as inputs and outputs in the experiments. It was therefore decided to use the trolley's acceleration as the input measure and obtain the horizontal force on the handle as output; a human subject would be better able to follow a certain acceleration instruction (for example by following targets on the floor which have to be reached at a fixed frequency) than an instruction to keep a certain force on the handle. Furthermore, by setting the acceleration as input measure the trolley's velocity can be easily controlled, and therefore it is easy to obtain a steady-state situation regardless of any present disturbances. A disadvantage of this approach however is the need to model a human arm[1]in order to obtain representable force readings on the trolley handle in simulation.

Unfortunately, this arm modeling proved to be non-viable for this thesis research due to problems encountered with the Gazebo simulation engine. Furthermore, it was later decided that the comparison between simulation and experiment was out of scope for this thesis research, thus the requirement for the measures to be suitable for an experimental setup was eliminated. Therefore, for the final simulations the horizontal handle force is chosen as input measure and the trolley acceleration is taken as output measure for the simulated system.

[1]In the current dynamic model described in chapter 2, the constraint imposing a fixed height on the trolley handle represents the human arm dynamics, since in reality this constraint would be enforced by the human arm. Hoever, this basic relation is assumed to be insufficiently accurate to obtain representable force readings when the horizontal handle force would be used as the output measure.

#### 6.1.2.2 Measuring performance

As mentioned earlier, analysis of the trolley system's output measure, which is chosen to be the acceleration, provides us with information about the system's behavior. Any deviation from the nominal acceleration[2]is considered an error in performance. In order to conveniently compare the controlled trolley performance with the performance of an uncontrolled trolley, a new 'performance error' measure is defined. This measure consists of the accumulated error between the measured- and nominal acceleration:

$$\text{performance error} = \sum_{i=0}^{N} \big(\ddot{x}_W[i] - \ddot{x}_{W,nom}[i]\big) \tag{6.1}$$

The performance error represents the 'total' error in performance of the trolley at any given time. Therefore, the best-performing control type is simply the type of which the performance error is lowest.

#### 6.1.2.3 Input profile

The most direct way to test the controller in the proposed simulation setup is to repeatedly apply the same, constant input force on the controlled and uncontrolled trolley in every scenario and compare their resulting performance error. This approach however imposes a problem: the trolley in simulation does not contain any damping, thus applying a force on it will always result in a persistent constant acceleration. Since this acceleration will differ between scenarios due to the different conditions they impose, it will be challenging to keep the trolley's velocity within bounds for all scenarios. Also, continuously applying the same, constant force would not be very intuitive, and therefore the resulting graphs will be more difficult to interpret than when a more natural force trajectory is applied. Therefore it is chosen to roughly mimic the natural force input profile of an elderly person pulling a trolley from a standstill in the nominal situation. Initially, a large force will be applied to accelerate the trolley to a satisfying velocity, after which the force input will lower to approximately the trolley's resistance force in order to obtain a steady-state velocity.

The input force profile is based on the average comfortable walking speed of elderly of 60+ years of age [39], which is roughly $1.3\,\mathrm{m\,s^{-1}}$, and an estimated acceleration duration of $2\,\mathrm{s}$ before the comfortable walking speed is reached. The corresponding acceleration- and steady-state forces are calculated using a restructured form of equation (2.10). In calculating the desired acceleration force, the trolley mass is set to its nominal mass plus the additional $20\,\mathrm{kg}$ of weight as applied in the 'added mass' scenario; this is done because the calculated force when only the nominal trolley weight was taken into account barely moved the trolley in this scenario, which resulted in it coming to a standstill before the end of the simulation. Note that, since the same input force profile is used in all simulations, the steady-state velocity in the nominal situation will no longer be $1.3\,\mathrm{m\,s^{-1}}$.

#### 6.1.2.4 Plotted measures

The data obtained from the simulations will be plotted for analysis. A selection of interesting measures is made for this purpose, which together provide a clear overview of the controller performance. The data is shown in a grid of plots, where data corresponding to the different scenarios is separated into columns, and different types of measures are divided over different rows.

The selected data includes the previously defined controller performance measure, as it is the main measure of interest, as well as the corresponding input measure, namely the applied horizontal force on the trolley handle. Additionally, the trolley's acceleration and the motor torque command are included to help with the interpretation of the results.

[2]The trolley's nominal acceleration equals the acceleration of a nominally behaving trolley in the nominal situation.

#### 6.1.2.5 x-Axis measure

The x-axis measure determines the common variable over which the data is compared. The most suitable measure can differ per scenario. The trolley behavior is simulated over time, and accordingly 'time' would overall be the most intuitive measure to plot the data against. Consequently, time will be used as x-axis measure for the 'no disturbance' and 'added mass' scenarios. However, in the 'slope' scenario, the beginning- and end of the slope might be encountered at different times between simulations. This would make it difficult to compare the trolley behavior at these points since the corresponding data will not end up at the same location in the plots. The slope's position is independent of the trolley's behavior, hence a position-related measure will be better suited here. Therefore, the trolley's traveled distance[9]will be used as the x-axis measure for this scenario.

## 6.2 Implementation in Gazebo

The simulation setup described in section 6.1 is performed using the Gazebo simulation environment. This section first provides some general information regarding this simulator and how it is used. Subsequently, the implementation of the trolley dynamics and simulation scenarios is described, as well as the methods used for the acquisition of different types of data from the simulation environment.

### 6.2.1 The Gazebo simulator

Gazebo is an open-source simulator, specialized for use in the field of robotics. It provides a physics engine to perform elaborate dynamic simulations, which can include a variety of different sensors and actuators. The basic communication structure of Gazebo is very similar to that of ROS, and therefore communication between the two is straightforward. This makes them easily integratable with each other, providing a powerful combination for the design and implementation of robotics software.

The communication between ROS-nodes and the Gazebo environment happens using messages and plugins. Via these channels, the components within the simulation can be manipulated and simulation data such as model states can be obtained. For example, plugins can be used as sensors or to actuate certain model parts. Several standard plugins are available, but custom ones can be built as well when a different functionality is required.

In Gazebo, the simulation environment can be specified programmatically using Simulation Description Format (SDF) and its ROS-specific sibling Unified Robot Description Format (URDF), both based on the XML file-format. Components within the simulation are built from different elements describing their geometry, dynamics and appearance, as well as potential other properties. A robot model is built from a combination of `<link>` and `<joint>` elements. Links represent elementary physical segments like boxes, cylinders and spheres, which can contain dynamic properties such as inertia and frictional characteristics. These links can be connected to each other using several types of joints, which set the range of relative movement of the connecting links. Joints can contain dynamics properties, such as friction and damping, as well. Properties of the simulated world itself, such as gravity, atmosphere and magnetic field, can be described in a similar fashion. The SDF- and Unified Robot Description Format (URDF) specifications can be found in [1] and [2], respectively.

Gazebo performs its simulations in 3D, whilst 2D simulation results are desired. This problem is overcome by building the simulation components to be symmetric along the plane in which the trolley is moving. This way, the third dimension will not have an effect on the relevant output data.

[9]Note that the traveled distance is used as x-axis measure rather than the trolley's horizontal position, as the traveled distance takes the trolley's vertical displacement into account as well.

### 6.2.2 Trolley model implementation

Due to Gazebo's component-based type of modeling, the equation (2.10) describing the trolley, as derived in chapter 2, cannot be directly implemented in simulation. The current section describes the implementation of the trolley dynamics in the Gazebo environment. The related relevant code is included in appendix G.

#### 6.2.2.1 URDF description

The URDF description of the trolley's basic elements is included in section G.1. The trolley model is composed of several `<link>` and `<joint>` elements, which together define the trolley's physical configuration. The trolley frame is modeled as a rectangular box, with the bag attached as a rectangular element by a 'fixed' joint. Its two wheels are attached to the base of the frame using a 'continuous' type of joint, which allows the wheels to spin freely around a specified axis. The geometry of the trolley's representation in Gazebo can be seen in fig. 6.1.

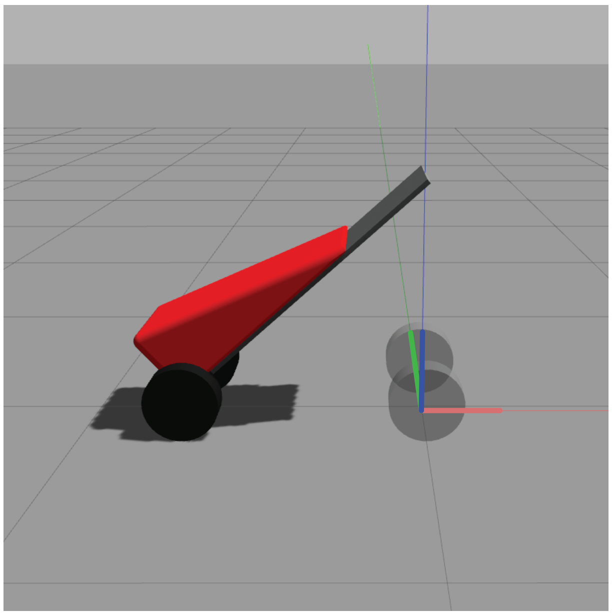


Figure 6.1: The trolley model with its local reference frame in the Gazebo environment. The transparent wheels are used as support to fix the height of the trolley handle with respect to the ground.

Besides these regular trolley elements, two support wheels are defined below the trolley handle. These are used for the implementation of the constraint set on the y-position of the handle, as will be explained in more detail later.

Physical properties can be included to links and joints via nested elements. The dimensions of the trolley model components can be declared via a `<geometry>` element included in the component's `<link>` element, whilst their masses and other inertial properties are specified through a nested `<inertial>` element. Some properties are defined from within a separate `<gazebo>` element, which is referenced to the link they apply on. The static friction $F_{STAT}$ acting on the trolley wheels is implemented by setting their Coulomb friction coefficient to 1 from within such element.

#### 6.2.2.2 Computations within Gazebo

Using the model's physical properties defined by its URDF description, Gazebo is able to compute several forces acting between model components and the simulation environment. The constraint forces between the frame and wheels, $F_{WX}$ and $F_{WY}$, are imposed by the joints between them. Furthermore, the gravitational forces $F_{FG}$ and $F_{WG}$ are computed using the components' masses and the gravitational acceleration $g$, which can be specified in the description of the simulation world. The normal force $F_N$ acting on the wheels is automatically computed by Gazebo as well.

The effect of road inclination $\gamma$ is implemented by the simulated terrain on which the trolley is situated. The simulation's default ground plane provides a flat, horizontal surface, but its orientation can be set manually to a different angle. Furthermore, static elements can be added to the simulation environment to provide all kinds of non-level surfaces over which robots can travel.

As explained in section 2.2.1, handle-force component $F_{HY}$ is defined via a constraint imposed on the y-position of the trolley handle. Furthermore, the constraint fixes the orientation of the trolley frame with respect to the road, therefore defining $\beta$ as well. A similar constraint is imposed within the simulation, by specifying the distance between the trolley handle and a set of support wheels, which are added underneath the handle. These additional wheels are very light in comparison to the other trolley elements, to minimize any additional effects on the trolley dynamics. A visual representation of the trolley model with support wheels can be seen in fig. 6.1.

#### 6.2.2.3 Plugins

The trolley model's inputs $F_{HX}$ and $T_M$ are imposed by the ROS structure from outside the Gazebo simulation environment, and are therefore applied using plugins. To achieve its desired operation, rolling friction torque $T_{ROLL}$ is implemented using a plugin a well. The plugins are described in more detail below.

**Motor plugin.** Motor torque $T_M$ is applied through the use of Gazebo's readily-available 'wheel joint effort controller' plugin [3]. This plugin acts as a motor on the trolley's wheels: it listens to torque commands on a specified topic and applies them to its target joints via a transmission interface, implemented in the joint's URDF description. The plugin is loaded in the code snippet of section G.2, line 16 to 19. The transmission element is specified in the wheel-joint components in section G.1, line 74 to 82.

**Handle force plugin.** When implementing the input force on the trolley handle $F_{HX}$, its orientation must be considered; $F_{HX}$ represents the force applied by the user to move the trolley forward, thus the handle force should always point in the trolley's direction of travel. This can be achieved by simply defining the handle force in the trolley's local reference frame (shown in fig. 6.1), since this frame moves and rotates along with the trolley model. By default, Gazebo provides a service which should allow a wrench[10]defined in any specified reference frame to be applied on a model link. Unfortunately, despite several attempts, this approach has failed to succeed; despite explicitly assigning the trolley's local frame as a reference according to the service's documentation [4], the specified wrench was always interpreted in the global reference frame. Therefore, a custom plugin was built to provide this functionality.

The plugin makes use of the `AddLinkForce()` function from Gazebo's Application Programming Interface (API) [5] to apply a force to a specified link, defined in the link's local reference frame. The force data is obtained from a specified topic by a subscriber, which saves the most recently published value as an attribute within the plugin. The plugin's relevant code is included in section H.1. Here the `AddLinkForce()` function is called on line 17, where the incoming force value is stored on line 22 in an attribute defined on line 33.

**Rolling friction plugin.** The friction caused by deformation of the road and wheels is included in the trolley dynamics as torque $T_{ROLL}$, acting on the trolley wheels. As can be seen in eq. (2.1), revisited on this page, $T_{ROLL}$ depends on normal force $F_N$ acting on the wheels. Gazebo provides the option to define a constant static friction element as part of the wheel joint[11], however this torque is not dependent on $F_N$. Furthermore, since it is applied within the joint instead of on the link of the wheel, the friction applies to both the wheel and the trolley frame to which the joint is attached. Applying the rolling-friction torque this way will therefore induce modeling errors in simulation, which contaminate the simulation results. A custom plugin is therefore designed for the implementation of a variable rolling friction, applied on the wheel bodies.

$$T_{ROLL} = \mu F_N R_W \tag{2.1 revisited}$$

The custom rolling friction plugin calculates the rolling friction according to eq. (2.1) and applies it exclusively on the trolley's wheel links using Gazebo API's `AddRelativeTorque()` function [5]. The used normal force acting on the wheels is calculated by addition of the vertical force acting on the wheel joints and the gravitational force of the wheel itself. The gravitational force of the wheel can be computed using the wheel's mass and the gravitational acceleration, where the vertical force acting on the wheel joints can be obtained from Gazebo by making use of its API's `GetForceTorque()` function [6]. The joint force is obtained from both wheels and their average value is used in the computation to keep the symmetry within the simulation. Furthermore, the trolley's angular wheel velocity is used to determine the direction in which the rolling friction torque should be applied. This is obtained from the Gazebo simulator as well, via the API's `RelativeAngularVel()` function [5]. A velocity deadband is applied by

[10]A wrench is a combination of a force and torque.

comparing the obtained value with a deadband threshold; the rolling friction is set to 0 whenever the velocity measurement does not surpass the threshold.

During testing of the plugin in simulation it was discovered that the angular velocity measurements obtained from Gazebo included a variable offset, which in some cases caused the plugin to apply a torque on a stationary trolley. A solution to this problem is implemented by adding a velocity calibration step. This step is performed at the start of the simulation on a stationary trolley, before any rolling friction torque is applied. During this step, the measured velocity of the stationary trolley is saved as a plugin attribute. The stored offset value is subtracted from incoming velocity measurements during the operation of the plugin to compensate.

The relevant code of the rolling friction plugin can be found in section H.1. The relevant parameters from the trolley's parameter configuration file and the ROS parameter service are loaded on line 20 to 24. The velocity calibration is performed by a function defined on line 49 to 54. The normal force and velocity are obtained from the Gazebo environment in functions defined on line 68 to 76 and line 78 to 87, respectively. Finally, the magnitude of the rolling friction is calculated and applied in a function on line 56 to 66, and the deadband is applied in a function specified on line 3 to 10. The corresponding attributes in which obtained parameter values are saved are defined on line 105 to 110.

### 6.2.3 Data acquisition

Similarily as to the application of several parameter values described in section 6.2.2, the relevant simulation output data is obtained from Gazebo through messages and plugins. Most importantly, as mentioned in section 6.1: 'Setup', the trolley's acceleration is needed to analyze the controller performance. To obtain the trolley's linear acceleration, a default Inertial Measurement Unit (IMU) sensor [7] is used. The IMU sensor provides information about the corresponding body's position, angular velocity and linear acceleration by publishing it on a specified topic. The information on this topic can consequently be saved for later use in analysis. The implementation of the IMU plugin in the trolley's URDF description is specified in the code of section G.2, line 22 to 35.

Furthermore, the trolley's x- and y position should be obtained for the calculation of the traveled distance, which will be used as a measure for plotting. Gazebo publishes this information by default on a '`link_states`' topic, and therefore no additional implementation needs to be provided to obtain it[12]. The same holds for the velocity measurements, which are published by default as well.

### 6.2.4 Scenario implementation

As described in section 6.1.1, three different scenarios will be used for testing the controller performance. The first one is a scenario where no disturbances are present. Since this scenario is consistent with the default simulation environment, no additional elements need to be added for its implementation.

The disturbance of additional weight on the trolley is tested in a scenario where the trolley is given a high mass. This is implemented through the trolley's URDF description, in which an additional mass can be added to the bag's total mass. The magnitude of the additional mass is specified in the trolley parameter configuration file, included in section E.5.

Finally, a scenario is used where the trolley drives over a slope to induce a disturbance in the road inclination. For this purpose, a sloped element is added to the simulation environment. The slope element is modeled as a rectangular block which is tapered at the end to form a ramp, as depicted in fig. 6.2. Since the ramp has a relatively complex shape, rather than defining it using basic links it is designed as a 3D shape in an external tool. The shape is subsequently loaded into the simulation environment as a static object.

During testing of the simulation environment it was discovered that the abrupt transition between the slope and the horizontal resulted in a strange oscillatory behavior in the trolley's acceleration measurement. This is presumably caused by errors in the trolley position due to the discrete nature of the

[12]The IMU sensor provides position- and velocity data as well, but it is chosen to use the data directly available from Gazebo for simplicity.

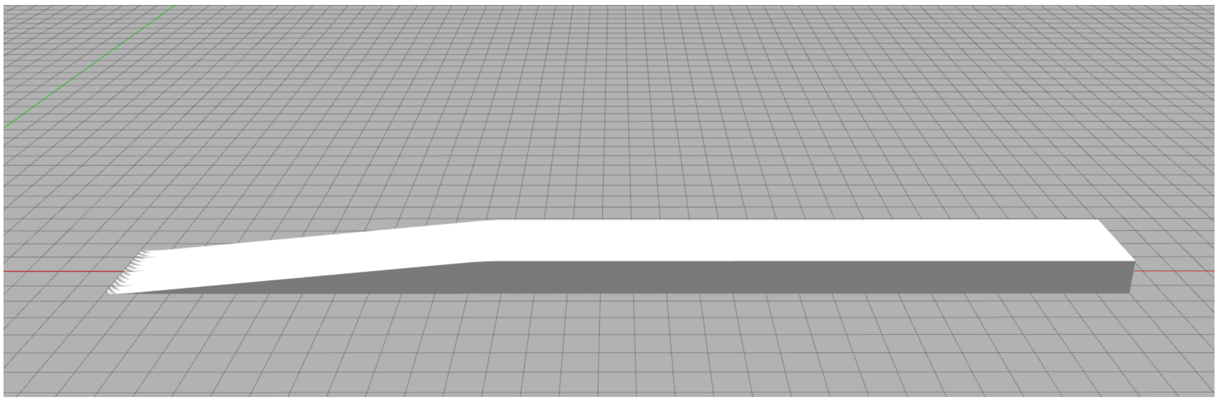


Figure 6.2: The slope used for testing disturbances in road inclination in the Gazebo environment.

simulation steps, resulting in high accelerations when the position is corrected. Improvements have been made by rounding the ramp's vertices to avoid as much as possible abrupt changes of the road surface. The ramp curvature is implemented as a mesh of triangular surfaces, and therefore it still contains non-smooth vertices. As a result, the smoothing of the ramp's edges reduces the amplitude of the oscillations in the acceleration signal but increases the distance over which they are introduced. Comparison of acceleration measurements between scenarios using a sharp-edged and rounded-edged slope shows that using rounded edges overall introduces the lowest performance error due to the oscillations, and therefore it is decided to use it.

## 6.3 Results

The experimental results of the three different tests are compiled in fig. 6.3, included on the next page. The test are performed by a controlled trolley, on which the Disturbance Observer (DOB) controller under analysis is implemented, and an uncontrolled trolley, which does not provide any assist. The data of these trolleys is depicted as an orange and blue line, respectively. The plots show the trolley's input- and output data: the applied handle force in the direction of travel and the trolley's linear acceleration, respectively. The motor torque command provided by the controller is included as well. Most importantly, the performance measure is presented, which represents the accumulated error between the measured- and nominal acceleration. This measure defines the controller's performance; it's magnitude is descriptive of the total difference between the desired and actual acceleration of the trolley over the course of the simulation.

### 6.3.1 Observations

In the upcoming sections, the results shown fig. 6.3 are discussed.

#### 6.3.1.1 No disturbance scenario

The left most column depicts the results of the scenario where no disturbances are present. This scenario is set as the 'nominal' scenario for the controller; the controller's goal is to make the trolley behave like this scenario at all times. This can be seen from the fact that the acceleration data of the uncontrolled trolley (blue line) exactly overlaps with the dotted line representing the nominal behavior. Consequently, the uncontrolled trolley has a performance error of $0\,\mathrm{m\,s^{-1}}$.

Since the uncontrolled trolley already behaves in the nominal way without intervention of the controller, it is to be expected that the controller will stay idle in this scenario. It can indeed be seen that, overall, very little motor torque is applied during the course of the simulation. Hoverer, at certain locations some activity can be noticed. This can be seen even better when looking at the trolley's acceleration; whenever the input force changes, the acceleration experiences an overshoot. This behavior is peculiar since an overshoot like this is typical for second order systems, whereas the controller only includes a first order filter.

An explanation is found by taking a closer look at the response time of intermediate signals when a change in handle force is applied. Figure 6.4 shows different signals for the uncontrolled trolley in the nominal scenario. Note that all intermediate signals are still present, only the feedback of the motor commands to the motor controllers is disabled. The graph includes the controller's relevant signals[13], as well as the acceleration obtained directly from the IMU sensor.

From the graph it can be seen that, after the application of a nonzero handle force, the acceleration signal (in orange) does not immediately reach its steady-state value. Since no damping is implemented in the simulation, after an initial delay caused by sensor update rate, the trolley's acceleration is expected to jump directly to its corresponding steady-state value. The acceleration measurement's gradual increase therefore introduces a measurement error, inducing a perceived disturbance in the controller. This induced disturbance estimate can also be seen in the graph's unfiltered disturbance estimate signal (in olive). The controller will try to compensate for the disturbance by ordering a motor torque, which consequently causes the overshoot in acceleration.

The acceleration signal obtained from the IMU sensor in the Gazebo simulation (in grey) first passes through a reader node in the control structure before the processed signal (in orange) is used in any computations. The IMU sensor data cannot be seen in the graph; this is because the processing in this case does not alter the data and induces almost no delay, which makes the two signals virtually identical. It can therefore be concluded that the overshoot seen in the controlled trolley's acceleration data is the result of a discrepancy in the sensor data obtained from the simulation, rather than an error in the controller itself.

Furthermore, a slight deviation between the steady-state acceleration of the controlled trolley and the nominal situation can be noticed whenever an input force is applied. A steady-state acceleration error

[13]The velocity used for determining the direction of the rolling friction is not taken into account.

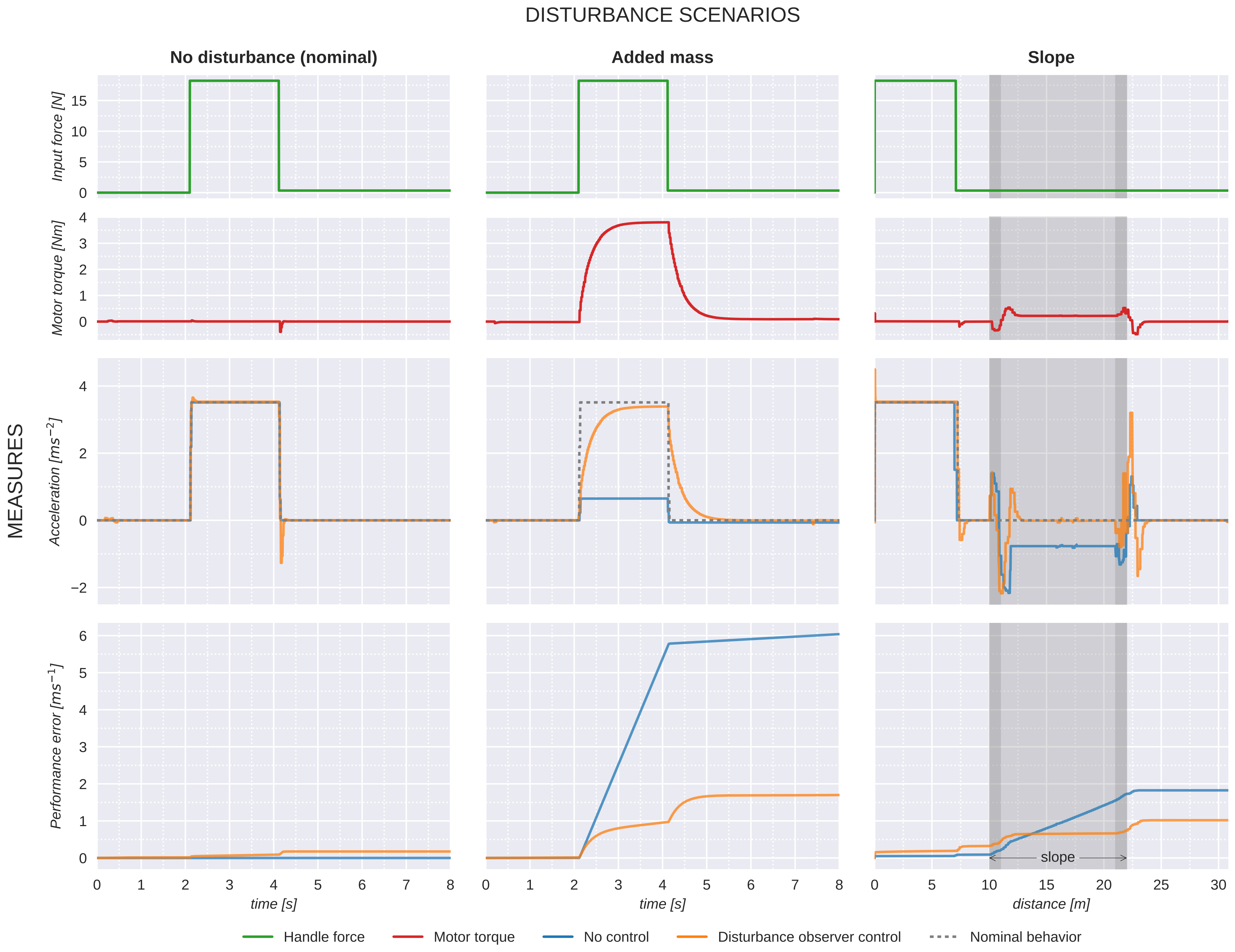


Figure 6.3: Simulation results for the three simulation scenarios. More information regarding the scenarios and measures is provided in section 6.1.

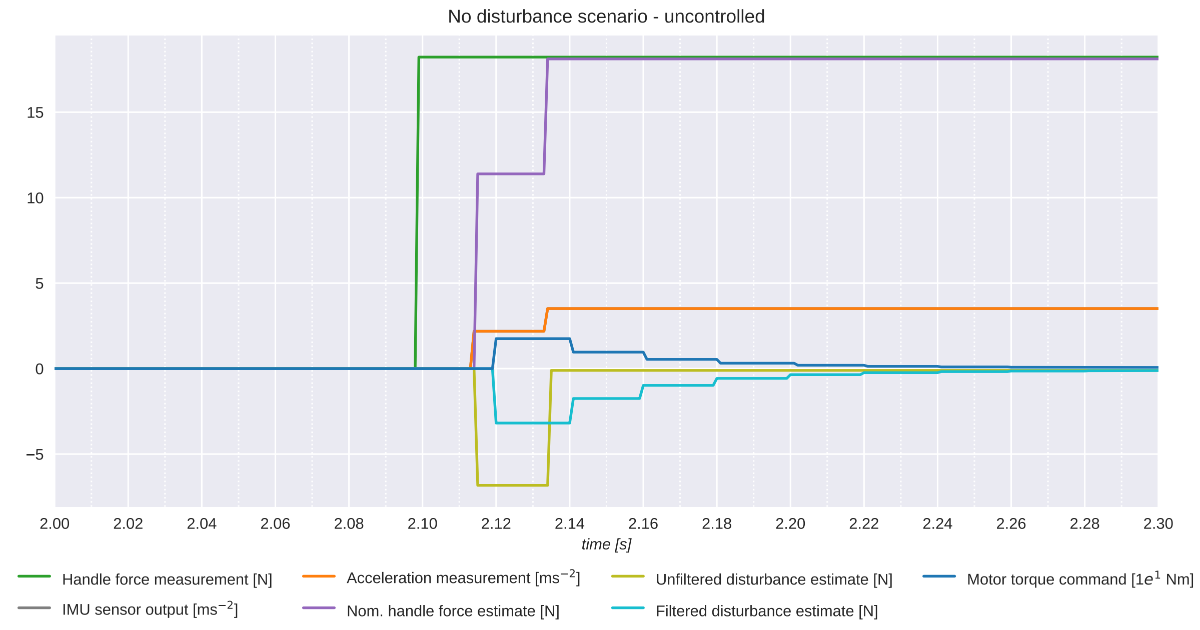


Figure 6.4: Response of different controller signals to an increase in applied handle force when no disturbance is present. Feedback of motor command to the motor controllers has been disabled. The sensor- and controller update rates are set to 50 Hz = 0.02 s per iteration.

implies a discrepancy between the controller's inverse nominal model and the 'real' situation, which in this case is the simulation. This difference is due to the addition of support wheels to the trolley model in Gazebo, as described in section 6.2.2. Therefore, it is actually the simulated nominal behavior that is slightly off, instead of the behavior of the controlled trolley.

In conclusion: for the scenario without any disturbances, the controller is expected to stay idle. For most of the simulation this is indeed the case. Some unwanted deviations from the nominal behavior are however present at specific locations, causing the performance error to increase slightly. These deviations are likely caused by discrepancies in the simulation environment, and not by the controller itself. Therefore it can be concluded that for this scenario the controller performs as expected.

#### 6.3.1.2 Added mass scenario

In the added mass scenario, an additional weight is added to the trolley bag to simulate a disturbance in the trolley's mass. The corresponding acceleration plot shows that the additional weight causes the acceleration of the uncontrolled trolley in reaction to the application of input force to be significantly reduced. The goal of the controller is to compensate for this reduction in order to maintain the trolley's nominal behavior. Therefore it is expected that the controller will start applying a motor torque whenever a difference in nominal and measured acceleration is detected, which will cause the acceleration to converge to its nominal value.

The motor torque data in fig. 6.3 shows the controller's reaction to the acceleration disturbance caused by the additional trolley weight. From the detailed view in fig. 6.5 it can be seen that the controller is quick to respond; it takes about one update cycle of 0.02 seconds for a compensatory motor torque command to be applied after the initial increase in handle force is measured. From the motor torque values plotted in fig. 6.3 it can furthermore be seen that the motor torque does not immediately assumes its steady-state value, but rather converges exponentially due to the controller's internal first-order Q-filter. The same behavior is seen when the handle force is lowered to obtain a steady-state velocity.

The gradual adjustment of the motor torque to a change in the disturbance estimation makes that its compensation happens over a period of time rather than instantaneously. Consequently, during this period a performance error is introduced. This can also be seen from the performance error graph for the current scenario; whenever a different handle force is applied the controlled trolley's performance error increases, following a similar shape as the change in acceleration.

Although the controller experiences a certain level of performance error when an additional mass is introduced to the trolley, its performance is significantly better than when no controller is used; when looking at the corresponding performance error plot in fig. 6.3 it can be observed that the controller action flattens the performance error curve, where the performance error in the uncontrolled trolley keeps rising

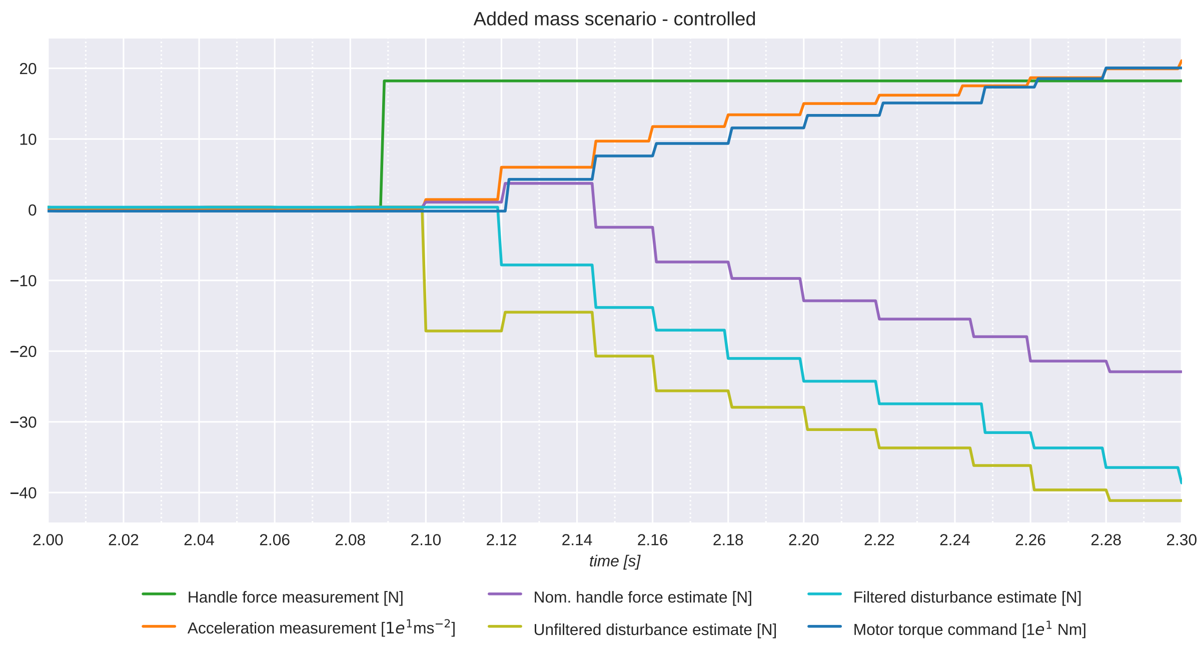


Figure 6.5: Response of different controller signals to an increase in applied handle force when an added mass is present. The sensor- and controller update rates are set to $50\,\mathrm{Hz} = 0.02\,\mathrm{s}$ per iteration.

lies around $6.0\,\mathrm{m\,s^{-1}}$, where this is approximately $1.7\,\mathrm{m\,s^{-1}}$ for the controlled case; this amounts to an improvement of roughly $72\,\%$ in performance when the designed controller is implemented.

#### 6.3.1.3 Slope scenario

The slope scenario tests the controller performance whenever a disturbance in road inclination is encountered by placing a ramp on the trolley's path. From the plot it can be seen that the acceleration of the uncontrolled trolley becomes negative when entering the slope. Again, in the assistive shopping trolley the controller should compensate for this deviation from the nominal acceleration. Note that the x-axis measure for the plots in fig. 6.3 corresponding to this scenario is set to be the trolley's traveled distance, opposed to 'time' for the other two scenarios. This choice is explained in section 6.1.

The first part of this scenario is identical to the situation where no disturbances are present. Therefore, it is expected that the controlled trolley's behavior will be identical to this scenario as well. When the trolley enters the slope, its acceleration will drop due to the change in gravitational orientation with respect to the direction of travel. The controller is expected to apply a compensatory torque in order for the acceleration to converge back to its nominal value, similarly as observed in the added mass scenario.

It can be observed from the corresponding acceleration data that the controller is indeed able to attain the desired acceleration over the central, flat part of the slope. However, contrary to expectation, the trolley's acceleration data displays strange behavior around the slope's entry and exit; A remarkable oscillation is present whilst the trolley is (partially) driving on the smooth edges of the ramp[14]. A similar oscillation can be observed in the acceleration data of the uncontrolled trolley, therefore the cause appears to relate to the dynamics of the simulation rather than the controller. The extra wave in the controlled trolley data oscillation is likely due to the controller's reaction to these perceived disturbances; the controller applies a motor torque to compensate for the acceleration error, effectively enforcing the oscillation. This effect can also be observed from fig. 6.6, where the controller signals are plotted for both the uncontrolled and controlled trolley around the period where the oscillations first occur. Both plots show the same signal types, only in the uncontrolled case the motor torque command is decoupled from the simulated motors, preventing the torque from being applied in simulation. In the controlled situation the torque command causes the acceleration to rise quicker than in the uncontrolled case, this rise in combination with the rise due to the oscillation causes the acceleration to overshoot its target.

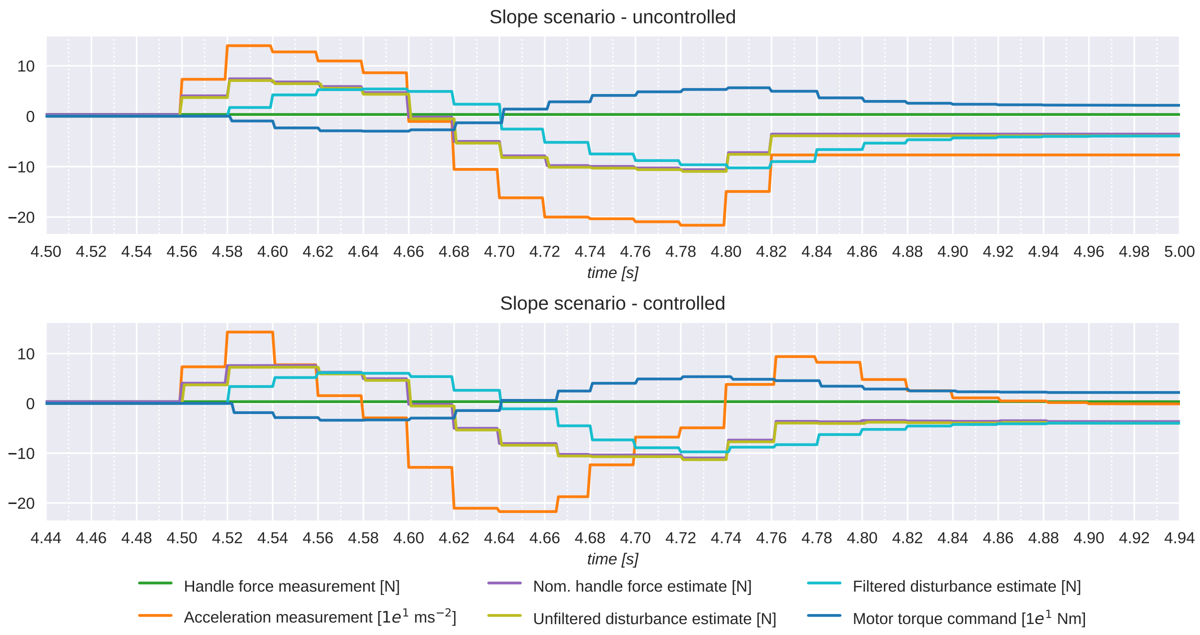


Figure 6.6: Responses of different controller signals to a changing road inclination. Feedback of motor command to the motor controllers has been disabled for the 'uncontrolled' case. The sensor- and controller update rates are set to 50 Hz = 0.02 s per iteration.

These oscillatory discrepancies locally have a big impact on the measured acceleration, therefore it is impossible to obtain a reliable performance error value in these areas. Moreover, other readily described disparities relating to the simulation are present, further reducing the reliability of the performance error. Consequently, the performance error data will only be analyzed qualitatively. As discussed previously under 'No disturbance scenario', before encountering the slope, the controlled trolley overall behaves in a nominal manner. Whilst driving on the ramp, a situation similar to the added mass scenario can be observed; where the uncontrolled trolley's performance error steadily increases due to the presence of a constant disturbance, the increase in performance error for the controlled trolley levels off fairly quickly. This is a good indication of the proper performance of the controller in this scenario. It is therefore reasonable to assume that the controlled trolley performs significantly better than a regular, uncontrolled trolley when a disturbance in road inclination is encountered.

### 6.3.2 Overall conclusions

Given these results, the following main conclusions can be drawn. The controller does not intervene when no disturbances are present, which means that it is able to estimate the nominal behavior correctly. Therefore, it can be concluded that its inverse nominal model, responsible for the estimation, is valid. When a disturbance is introduced, the controller picks up on it quickly; the controller is able to respond to a measured change in acceleration within one update iteration, which equals a maximum delay of 0.02 seconds. Provided that the sensors run at 50 Hz as well independently of the controller, the total response time to an encountered disturbance is maximally 0.04 seconds. After a response is initiated, it takes less than a second for the controller to attenuate 90 % of the applied disturbance.

Since no performance error is introduced by the controller in the nominal scenario, the controlled trolley does not provide a disadvantage with respect to the use of a regular trolley in this situation. When disturbances are introduced, the initial increase in performance error is equal between the controlled and uncontrolled trolley. However, when the disturbances are detected by the controller it applies a compensatory torque to eliminate their effects, leveling the induced rise in performance error. In the case of the uncontrolled trolley this does not happen; the effect of the disturbances will persist until

[14]The ramp's edges are rounded to reduce the performance error which is introduced by the unnatural oscillatory acceleration that appears when the trolley encounters the edges. More information can be found in section 6.2.4.

the disturbances themselves go away. It can therefore be concluded that, when any disturbances are present during operation of the trolley, the controlled trolley will outperform its uncontrolled equivalent. Moreover, the controlled trolley's performance advantage will increase with an increase in disturbance duration due to its ability to flatten the curve, whereas the uncontrolled trolley's performance error will keep rising over the full duration of the disturbance.

### 6.3.3 Future improvements

Figure 6.3 also shows various points for future improvement. First, it can be seen that the majority of the controller performance error is introduced at points where changes in disturbances occur, due to the controller's gradual adaptation to the new situation. Accordingly, the controller performance can be improved by shortening the adaptation period. The gradual convergence of the controller response to its new steady-state value is caused by the controller's Q-filter, in an effort to remove high-frequency noise from the estimated disturbance signal. Reducing the Q-filter's rise time speeds up the convergence and therefore will possibly decrease the accumulated performance error when changes in disturbances occur. Conversely, this reduces the Q-filter's capability to suppress noise, which in turn can increase the performance error and introduce bothersome jittering in the trolley's movements. More research should be done with respect to the controller's response to noise in order to finetune the rise time for optimal performance.

Moreover, it is hypothesized that a faster controller convergence will result in less overshoot in response to an oscillatory disturbance, such as seen around the slope edges in the slope scenario data; the controller will be able to adapt its output torque more quickly to the oscillating acceleration, reducing the lagging torque's reinforcing effect as the acceleration changes direction. To further diminish the overshoot, the controller's response time could be reduced. This can be done either by reducing the maximum delay between the acquisition and processing of data through increasing the Q-filter's processing rate, or by providing the Q-filter with more recent data through increasing the measurement rate of the sensors.

Lastly, in the nominal scenario, a small discrepancy between the steady-state accelerations of the controlled and uncontrolled trolley is introduced when a nonzero input force is applied. This is caused by the simulated trolley being slightly to heavy, due to the addition of two support wheels. Although in this instance the induced acceleration error is caused by a disparity on the side of the simulation, it clearly demonstrates the effect of dynamic differences between the controller's inverse nominal model and reality. These effects are however not taken into account in the current analysis, since the simulated environment is built using the same parameters as used in the inverse nominal model. More research is required to determine if the current inverse dynamic model is still sufficiently accurate when the controller is implemented on a real trolley.

# Chapter 7

# Conclusion

Our aging society puts a heavy strain on our healthcare system. Assisting the elderly with robotic devices can help increase their mobility, therefore reducing healthcare costs and increasing elderly satisfaction. In previous research [8], a novel concept is proposed for an assistive, motorized shopping trolley, with the purpose of reducing the physical burden for elderly of carrying groceries over uneven terrain. Continuing on this concept, the aim of this thesis has been to design a control system for use in the assistive shopping trolley.

Since elderly generally have an above-average distrust of technology, the above mentioned goal was refined to include gaining the acceptance of the elderly user. The goal was approached by focussing on the attenuation of effects caused by varying parameters and environmental influences such as added mass and road inclination, also called disturbances, on the trolley's dynamics. Moreover, the controller should behave in a predictable manner to prevent scaring the user.

The Disturbance Observer (DOB) is chosen to be used as a controller on the basis of a literature review, in which seven different control types found in other assistive shopping trolleys and power-assist bicycles have been compared on aspects relating to their predictability and disturbance attenuating behavior. A Disturbance Observer (DOB)uses a dynamic model of the controlled plantto estimate the influence of present disturbanceson the plantdynamics. This estimation can consequently be used to produce an opposing force, thereby canceling the effects of the disturbances. A DOB moreover includes a filter which provides robustness to uncertainty within the model and filtering of high-frequency noise from the data. The DOB was deemed most suitable over other control types due to its overall adequate performance with respect to all the imposed criteria, as well as its relatively high disturbance attenuation accuracy due to the use of the plant's dynamics in making an estimation of the disturbances at play.

A model of the trolley dynamics has been designed for use in the DOB, as well as for use in a stability analysis of the trolley-controller closed-loop behavior. The model was based on the dynamics of an inverted pendulum on wheels, with an additional constraint on the position of the trolley handle to represent the interaction with the human user. The constraint fixes the height of the trolley handle with respect to the road, following the hypothesis that during normal use, the user will keep his/her arm approximately stretched due to the trolley's weight. By neglecting small, nonlinear dynamic effects such as variations in the trolley's orientation and air drag, the general model is simplified to a linear equation. A basic simulation of the model's nonlinear, unconstrained dynamics as well as an analysis of its linearized mathematical construction have been used to verify its behavior.

A nominal model was created by combining the dynamic model with a set of parameters that correspond with the desired trolley behavior: empty bag and flat road. The controller was designed by analyzing the loop transfer function of the closed loop system in the presence of parameter uncertainty of friction, trolley mass and handle height due to differences in the user's length. Using $H_\infty$ analysis it was found that the uncertainty range is too large, and the system may become uncontrollable. This may be resolved by making the uncertainty range smaller or using a less conservative uncertainty approximation, but this is left for further research. It was also found that if this would be resolved, any Q-filter design will lead to a stable closed-loop system. A first-order low-pass filter was chosen as design. To filter high-frequency noise but not frequently-encountered low-frequency disturbances such as ramps, its cut-off frequency was set to 5 Hz.

The controller has been implemented using the Robotic Operating System (ROS) robotics framework in combination with the Gazebo simulator. In ROS, functional structures are implemented as a system of independent processes called 'nodes'. Consequently, the controller's main functionality has been subdivided into multiple components. Additional nodes are implemented for the conversion of exchanged data between sensors/actuators and the main controller. Finally, several elements are added for the purpose of the simulation and the corresponding data analysis. Various unit- and integration tests were built to test all implemented controller- and simulation components, in an effort to ensure their proper functionality.

The controller's performance was tested in simulation by comparing the controlled trolley's acceleration error to that of a regular, uncontrolled trolley in several scenario's, where different types of disturbances are applied. The results are promising: the data reveals that the controller succeeds in reducing the effects of disturbances on the trolley's behavior with respect to a regular, uncontrolled trolley in the simulated situations. The controller is able to recognize the trolley's changing behavior due to an encountered disturbance, which indicates that its inverse dynamic model is properly able to predict the trolley's nominal behavior. Additionally, the controller correctly attenuates the effects of the disturbances by ordering the required compensatory motor torque.

The research done for this project yields a positive first impression to the use of DOB control in assistive shopping trolleys. More research is needed for further improvement of the controller; Firstly, the controller's response to noise should be analyzed in order to finetune the Q-filter's rise-time for optimal performance. Secondly, the fact that the uncertainty range is too large should also be further researched. In addition, a more extensive stability analysis including time delay and high-frequency unmodeled dynamics should be performed to further ensure the controllers safety. Finally, the controller should be tested on a prototype in order to determine the accuracy of the current inverse nominal model with respect to the trolley's real-life dynamics.

On a more general level, but certainly not of less importance: it is recommended to validate the accordance between the measure used to define the controller's performance and the preferences of the elderly user; the assistive shopping trolley will not be of much use when it will not be utilized by the people that it is intended to help.

# Appendices

# Appendix A

# Estimation of trolley center of mass position

In this appendix an estimation of the Center of mass (COM) position of the combined trolley frame and -bag is made. A schematic overview of the trolley with relevant parameters can be seen in fig. A.1.

**Frame** The trolley frame can be approximated by a beam of uniform mass with its center of mass $F$ in the middle of its length $L_{WH}$ and width and $W_F$. Its position with respect to wheel center $W$ is thus:

$$\begin{aligned} L_{FC} &= \frac{1}{2} L_{WH} \\ W_{FC} &= 0 \end{aligned}$$

**Bag** The trolley bag can be approximated by a triangle of uniform mass, with its center of mass $B$ on 1/3 of its length $L_B$ an d width $W_B$. its position with respect to point $W$ is thus:

$$\begin{aligned} L_{BC} &= \frac{1}{3} L_B \\ W_{BC} &= \frac{1}{3} W_B + \frac{1}{2} W_F \end{aligned}$$

**Combined** The combined center of mass $T$ will lie somewhere between the center of masses $F$ and $B$, to ratio of the frame- and bag masses $m_F$ and $m_B$:

$$\begin{aligned} L_{WF} &= \frac{m_F}{m_F + m_B} L_{FC} + \frac{m_B}{m_F + m_B} L_{BC} && = \frac{1}{m_F + m_B} \left( \frac{m_F}{2} L_{WH} + \frac{m_B}{3} L_B \right) \\ W_{WF} &= \frac{m_B}{m_F + m_B} W_{BC} && = \frac{m_B}{m_F + m_B} \left( \frac{1}{2} W_F + \frac{1}{3} W_B \right) \end{aligned}$$

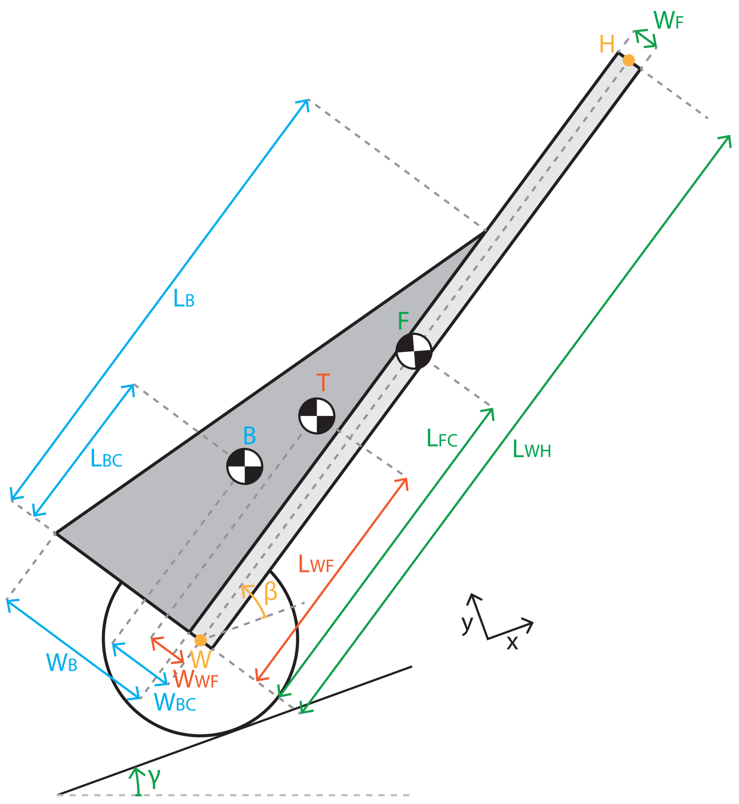


Figure A.1: Schematic overview of the trolley with relevant parameters.

# Appendix B

# Model equation verification simulation parameters

This appendix states all parameter values used in the simulations for the verification of the dynamic model equation described in section 2.1.3.

| **Parameters** | | **Frame horizontal initial condition** | **Inclined road** | **Motor torque** |
|---|---|---|---|---|
| | | *Constant values* | | |
| $F_{HX}$ | [N] | 0 | 0 | 0 |
| $F_{HY}$ | [N] | 0 | 0 | 0 |
| $T_M$ | [N m] | 0 | 0 | 1 |
| $m_F$ | [kg] | 0.9 | 0.9 | 0.9 |
| $m_B$ | [kg] | 1.2 | 1.2 | 1.2 |
| $m_W$ | [kg] | 3.9 | 3.9 | 3.9 |
| $\mu$ | [−] | 0 | 0 | 0 |
| $g$ | [m s$^{-2}$] | 9.81 | 9.81 | 9.81 |
| $L_{WH}$ | [m] | 0.95 | 0.95 | 0.95 |
| $L_B$ | [m] | 0.65 | 0.65 | 0.65 |
| $W_{WF}$ | [m] | 0 | 0 | 0 |
| $R_W$ | [m] | 0.11 | 0.11 | 0.11 |
| $\gamma$ | [rad] | 0 | $\tan^{-1}(1/10) \approx 0.1$ | 0 |
| | | *Initial conditions* | | |
| $\theta$ | [rad] | 0 | 0 | 0 |
| $\beta$ | [rad] | 0 | $-0.5\pi - \gamma$ | $-0.5\pi$ |

# Appendix C

# Model assumption calculations

This appendix supports the assumptions made in section 2.2: ‘Simplifying the model’ with the goal of simplifying the trolley’s dynamic model equation with calculations. For these calculations the values specified in table C.1 are used, which are obtained by measuring or estimated based on literature.

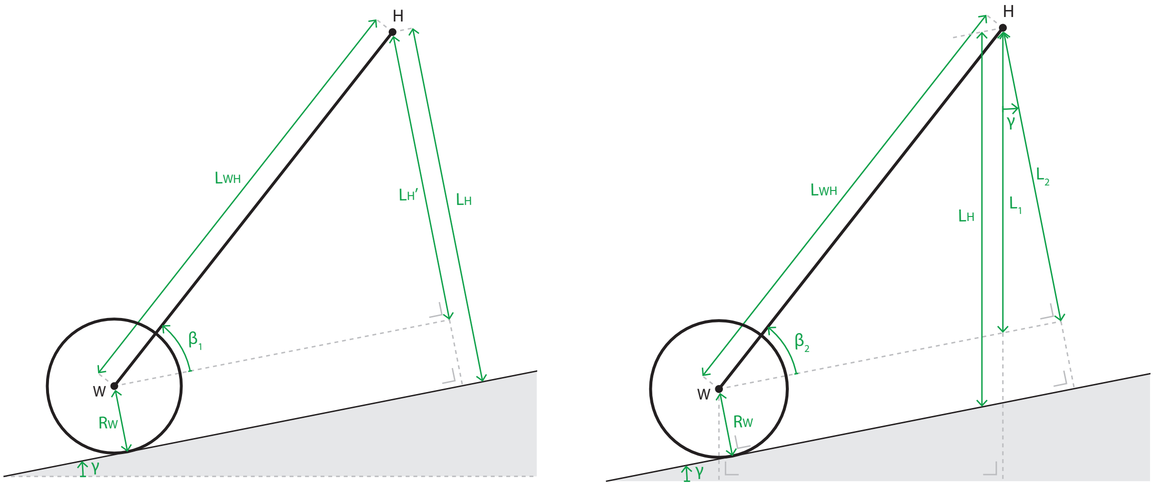
Table C.1: Parameter values used for model assumption calculations

| Parameter | | Description |
|---|---|---|
| $L_{H,avg}$ | $0.742\,\text{m}$ | Average trolley handle height with respect to the road[1] |
| $L_{WH}$ | $0.95\,\text{m}$ | Trolley frame height |
| $R_W$ | $0.11\,\text{m}$ | Trolley wheel radius |
| $\mu$ | $0.01$ | Nominal rolling friction coefficient |
| $\gamma_{max}$ | $\tan^{-1}(1/10)$ $\approx 0.10\,\text{rad}$ | Maximum allowed ramp inclination as defined by Dutch law [15] |

[1]Based on data of the fist height of standing Dutch adults of 60+ years of age (2004) [25].

## C.1 Effect of human body orientation on frame angle

This appendix calculates the maximum error in frame angle $\beta$ due to assumption that the orientation of the user’s body is always perpendicular to the road.



(a) $L_H$ measured perpendicular to road

(b) $L_H$ measured vertical

Figure C.1: Schematic overview of trolley orientation for different definitions of handle height $L_H$.

The maximum error that this assumption induces on frame angle $\beta$ occurs when road slope $\gamma$ is maximal, and can be reviewed by comparing the values of $\beta$ for $\gamma_{max}$ in two scenarios:

**Scenario 1** Height $L_H$ from the road to trolley handle $H$ is measured *perpendicular to the sloped road* as seen in fig. C.1a

It is more realistic to assume that humans position their bodies such that their center of gravity is above their feet in order to prevent themselves from tipping over. Therefore a more realistic situation will be:

**Scenario 2** Height $L_H$ from the road to trolley handle $H$ is measured *vertically* as seen in fig. C.1b.

The corresponding values for $\beta$ are calculated as follows:

**Scenario 1**:

$$L'_H = L_H - R_W \text{ [m]} \tag{C.1a}$$
$$\beta_1 = \sin^{-1}\left(L'_H/L_{WH}\right)$$
$$\approx 0.728\,\text{rad} \tag{C.1b}$$

**Scenario 2**:

$$L_1 = L_H - R_W/\cos(\gamma_{max}) \text{ [m]} \tag{C.2a}$$
$$L_2 = \cos(\gamma_{max})L_1 \text{ [m]} \tag{C.2b}$$
$$\beta_2 = \sin^{-1}(L_2/L_{WH})$$
$$\approx 0.723\,\text{rad} \tag{C.2c}$$

where the used parameter values are listed in table C.1. The maximum error in frame angle $\beta$ due to fixing the position of trolley handle $H$ perpendicular to the road is:

$$e_{\beta,max} = |\beta_2 - \beta_1| \qquad \approx 0.005\,\text{rad} \qquad \approx 0.30° \qquad \approx 0.73\,\%$$

where the error percentage is defined with respect to the value of $\beta$ in Scenario 2, eq. (C.2c). This error percentage is considered to be negligible and therefore the assumption that trolley handle $H$ is kept at a constant perpendicular distance from the road surface is validated.

## C.2 Effect of changing road slope on acceleration estimation error

This appendix calculates the maximum error in the estimated linear trolley acceleration due to the assumption that a change in road slope $\gamma$ does not have an influence on the value of trolley frame angle $\beta$. This error is calculated by comparing the estimated acceleration for a situation where $\gamma$ is constant with an estimation for when the change in $\gamma$ is largest.

**Frame angle for constant road slope.** If road slope $\gamma$ remains constant during use, the trolley orientation is fixed and frame angle $\beta$ becomes constant as well. This situation is depicted in fig. C.1a. The value of $\beta$ ($\beta_1$ in fig. C.1a) in this case can be calculated as shown in section C.1:

$$L'_H = L_H - R_W \text{ [m]} \tag{C.1a revisited}$$
$$\beta_1 = \sin^{-1}\left(L'_H/L_{WH}\right)$$
$$\approx 0.728\,\text{rad} \tag{C.1b revisited}$$

**Frame angle for variable road slope.** If road slope $\gamma$ changes during use of the trolley, it will have an effect on frame angle $\beta$ and consequently on estimated acceleration $\ddot{x}_W$. Since trolley handle $H$ at one end of the trolley frame usually moves in front of wheel center $W$ at the other side, handle $H$ will reach the slope before wheel center $W$ and will therefore start to rise before point $W$ does. Whenever wheel center $W$ is still on horizontal ground, the rise of $H$ will cause the frame to tilt upwards and therefore cause $\beta$ to increase. As soon as the trolley wheel starts to travel up the slope, $\beta$ returns to its value for constant $\gamma$.

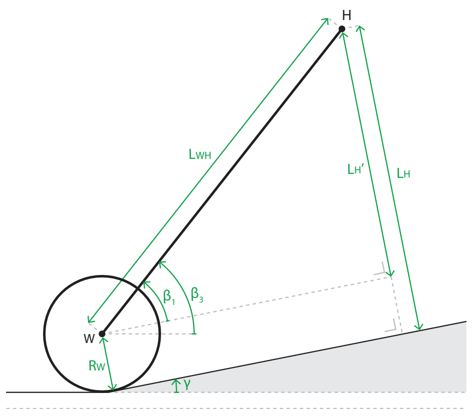


Figure C.2: Schematic overview of trolley orientation whilst driving over a change in road slope.

Frame angle $\beta$ is largest when road slope $\gamma$ changes abruptly from 0 to its maximum value of $\tan^{-1}(1/10)$ [rad] (as defined by Dutch law [15]) during use and the trolley wheels just reach the start of the slope; at this point $\beta$ is still measured with respect to the horizontal road whilst trolley handle $H$ is at its highest point with respect to wheel center $W$. A schematic overview of this situation can be seen in fig. C.2. In this situation frame angle $\beta_3$ can be calculated as follows:

$$\beta_3 = \beta_1 + \gamma_{max} \qquad \approx 0.828\,\text{rad} \tag{C.3}$$

Using dynamic model equation (2.10), the estimated linear trolley accelerations belonging to $\beta_1$ and $\beta_3$ can be calculated:

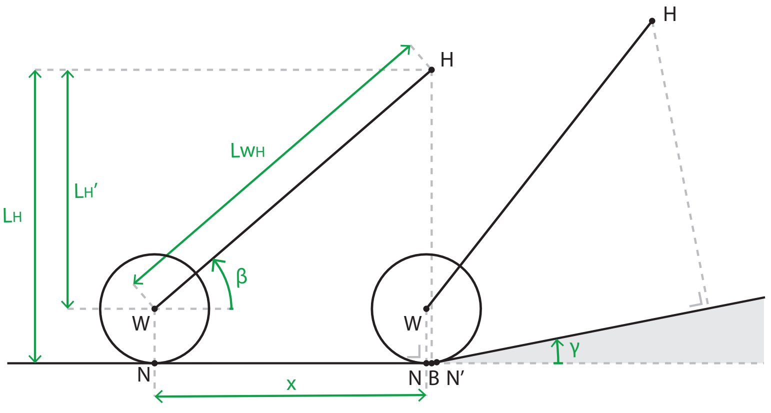
$$\begin{aligned} \ddot{x}_{W,\beta_1} &\approx 3.5543\,\text{m}\,\text{s}^{-2}; & \ddot{x}_{W,\beta_3} &\approx 3.5528\,\text{m}\,\text{s}^{-2} \\ \Delta\ddot{x}_W &\approx -1.6\times10^{-3}\,\text{m}\,\text{s}^{-2} & &\approx -0.21\,\% \end{aligned} \tag{C.4}$$

where the error percentage is defined with respect to $\beta_1$.

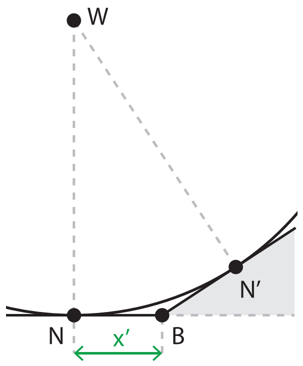
**Time-span of changing road slope.** The time-span in which the trolley is affected by a change in road inclination depends on the distance the trolley needs to travel to overcome this change, as well as the speed at which it is traveling. The distance over which the trolley is affected by the change in road slope $\gamma$ is the traveled distance of the trolley from the point that it starts entering the slope to the point where it is completely on it. A graphical representation of these situations can be seen in fig. C.3a, where the traveled distance is denoted by $x$. The trolley starts entering the slope when handle $H$ is above the slope's base $B$, and it is completely on the slope whenever the wheel touches the slope at point $N'$. The latter situation happens just before wheel center $W$ is above slope base $B$, as can be seen in fig. C.3b. The horizontal distance between points $W$ and $B$ is however negligible, thus it can be assumed that $W$ is above $B$ at the moment the trolley wheel touches the slope. Therefore traveled distance $x$ can be approximated by the horizontal projection of the frame with endpoints $W$ and $H$:

$$x = L_{WH}\cos(\beta) \qquad = \sqrt{(L_{WH}^2 - L_H'^2)} \qquad \approx 0.71\,\text{m} \tag{C.5}$$

The minimal comfortable walking speed of elderly without pulling a trolley is a little above $1\,\text{m}\,\text{s}^{-1}$ [39], therefore a minimal walking speed of $1\,\text{m}\,\text{s}^{-1}$ whilst pulling the trolley is assumed. With this speed, it takes the elderly user approximately 0.7 s to overcome the distance in which the trolley is affected by changing road slope $\gamma$.



(a) Overview of the affected range.



(b) Closeup of trolley wheel when touching the slope.

Figure C.3: Range $x$ in which the trolley is effected by a change in road slope $\gamma$.

**Conclusion.** Equation (C.4) shows that the maximum deviation of linear trolley acceleration $\ddot{x}_W$ from its estimated real value due to considering road slope $\gamma$ constant is approximately 0.21 %, which is considered to be negligible. Moreover, the error in $\beta$ gradually increases from 0 over distance $x$ after the trolley starts entering the slope and its maximum is only reached when the trolley is about to be completely on the slope. Also, if the ascending part of the slope is shorter than distance $x$ of 0.71 m or the slope gets less steep along this distance, the maximum error isn't reached at all. The average error is therefore even lower. Furthermore, the time span in which this error occurs is shown to be maximal 0.7 s, after this time the trolley will be completely on the slope and the error will be brought back to 0. From these results it can be concluded that assuming road slope $\gamma$ to be constant can be considered acceptable in the scope of this thesis project.

# Appendix D

# Control method transfer functions

The current appendix deduces the transfer functions of the control types examined in chapter 3: 'Control Method Overview'.

## D.1 Plant

The following derivation is based on the block diagram seen in fig. D.1. Here $G$ represents the transfer function of the controller.

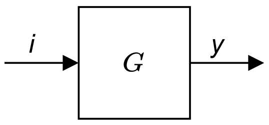


Figure D.1: Block diagram of a general plant.

Given that the plant input $i$ represents force or torque and output $y$ represents velocity, we can derive the following plant transfer function from eq. (2.10):

$$G = \frac{A}{s} \tag{D.1}$$

where A represents all constant terms. Consequently the transfer function of plant model M can be given as

$$M = \frac{B}{s} \tag{D.2}$$

where $B$ represents al constant terms.

## D.2 Proportional torque control

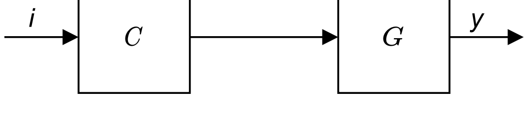


Figure D.2: Block diagram of Proportional Torque Control.

The following derivation is based on the block diagram seen in fig. D.2. Here $C$ and $G$ represent the transfer function of the controller and plant respectively.

$$y = GCi$$
$$H = \frac{y}{i} = GC$$

Filling in eq. (D.1) results in

$$H = \frac{AC}{s}$$

## D.3 Velocity control

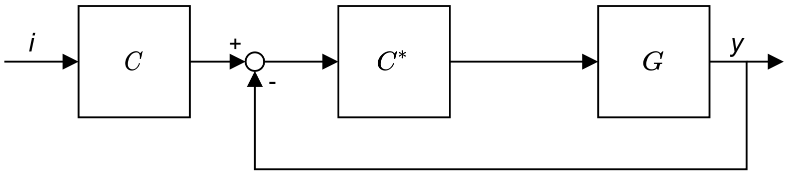


Figure D.3: Block diagram of Velocity Control.

The following derivation is based on the block diagram seen in fig. D.3. Here $C$, $C*$ and $G$ represent the transfer functions of the main controller element, additional velocity controller element and plant respectively.

$$y = GC^*(C_1 i - y)$$
$$H = \frac{y}{i} = \frac{GC^*C}{1 + GC^*}$$

Filling in eq. (D.1) results in

$$H = \frac{AC^*C}{s + AC^*}$$

## D.4 Disturbance Observer control

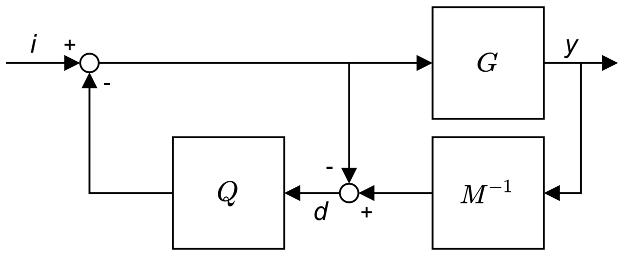


Figure D.4: Block diagram of Disturbance Observer Control.

The following derivation is based on the block diagram seen in fig. D.4. Here $G$, $M$ and $Q$ represent the transfer functions of the plant, model and Q-filter respectively.

$$
\begin{aligned}
d &= \frac{1}{M}y - (i - Qd) \\
d &= \frac{1}{M(1-Q)}y - \frac{1}{1-Q}i \\
y &= G(i - Qd) \\
&= G\left(i - Q\left(\frac{1}{M(1-Q)}y - \frac{1}{1-Q}i\right)\right) \\
&= \frac{G}{1-Q}i - \frac{GQ}{M(1-Q)}y \\
y &= \left(1 + \frac{QG}{M(1-Q)}\right)^{-1} \frac{G}{1-Q}i \\
H &= \frac{y}{i} = \frac{GM}{M(1-Q) + GQ}
\end{aligned}
$$

Filling in eq. (D.1) and eq. (D.2) results in

$$
H = \frac{AB}{s\left((A-B)Q + B\right)}
$$

## D.5 Cruise control

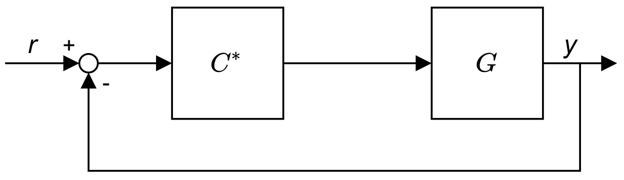


Figure D.5: Block diagram of Cruise Control.

The following derivation is based on the block diagram seen in fig. D.5. Here, $C*$ and $G$ represent the transfer functions of the additional velocity controller element and plant, respectively.

$$
\begin{aligned}
y &= GC^*(r - y) \\
H &= \frac{y}{r} = \frac{GC^*}{1 + GC^*}
\end{aligned}
$$

Filling in eq. (D.1) results in

$$
H = \frac{AC^*}{s + AC^*}
$$

# Appendix E

# Configuration files

This appendix includes the configuration files in which controller- and simulation settings are specified. These files are generally loaded using the `load_config()` function, which returns the configuration file as a Python dictionary.

## E.1 General configuration

YAML configuration file containing settings used in multiple elements.

```yaml
loop_delay: 0.05 #^ [s]

velocity_deadband: 0.01 #@ [rad/s] Safety range around velocity 0 for determining rolling friction
↪  direction
```

Snippet E.1: `config_general.yaml`

## E.2 Sensor configuration

YAML configuration file containing settings specific to sensors.

```yaml
update_rate: 50. #^ [Hz]

#% handle_force_generator_node
publish_delay: 2. #^ how long before handle force should be applied
```

Snippet E.2: `config_sensors.yaml`

## E.3 Simulation configuration

YAML configuration file containing settings specific to the simulation. Note that the target velocity specified here is different from the target velocity in the actual simulations. This is due to the use of a large trolley mass during the calculation of the simulation's input-force profile. The motivation behind this action is given in section 6.1.2.3.

```yaml
target_velocity: 1.3 #^ [m/s]
acceleration_duration: 2. #^ [s]

support_wheel_mass_ratio: 0.01 #@ Part of total wheel weight
```

Snippet E.3: `config_simulation.yaml`

## E.4 Controller configuration

YAML configuration file containing settings specific to the controller.

```yaml
## Config file for trolley controller elements ##
#% General
update_rate: 50

#% Inverse nominal model
model_Fhx: (s.F_fg*s.L_wf*s.R_w*s.mu*sp.sin(s.beta)*sp.sin(s.beta + s.gamma)*sp.sign(s.d_xw) -
↪  s.F_fg*s.L_wf*s.R_w*s.mu*sp.cos(s.gamma)*sp.sign(s.d_xw) +
↪  s.F_fg*s.L_wh*s.R_w*s.mu*sp.cos(s.beta)**2*sp.cos(s.gamma)*sp.sign(s.d_xw) +
↪  s.F_fg*s.L_wh*s.R_w*sp.sin(s.gamma)*sp.cos(s.beta)**2 +
↪  s.F_fg*s.R_w*s.W_wf*s.mu*sp.sin(s.beta)*sp.cos(s.beta + s.gamma)*sp.sign(s.d_xw) +
↪  s.F_fg*s.R_w*s.W_wf*s.mu*sp.sin(s.gamma)*sp.sign(s.d_xw) +
↪  2.0*s.F_wg*s.L_wh*s.R_w*s.mu*sp.cos(s.beta)**2*sp.cos(s.gamma)*sp.sign(s.d_xw) +
↪  2.0*s.F_wg*s.L_wh*s.R_w*sp.sin(s.gamma)*sp.cos(s.beta)**2 -
↪  s.I_f*s.R_w*s.d_beta**2*s.mu*sp.sin(s.beta)*sp.sign(s.d_xw) -
↪  s.L_wf**2*s.R_w*s.d_beta**2*s.m_ft*s.mu*sp.sin(s.beta)*sp.sign(s.d_xw) -
↪  s.L_wf*s.L_wh*s.R_w*s.d_beta**2*s.m_ft*sp.cos(s.beta) +
↪  s.L_wf*s.R_w*s.dd_xw*s.m_ft*s.mu*sp.sin(2.0*s.beta)*sp.sign(s.d_xw)/2.0 -
↪  s.L_wh*s.R_w*s.W_wf*s.d_beta**2*s.m_ft*s.mu*sp.cos(s.beta)*sp.sign(s.d_xw) +
↪  s.L_wh*s.R_w*s.dd_xw*s.m_ft*sp.cos(s.beta)**2 + 3.0*s.L_wh*s.R_w*s.dd_xw*s.m_w*sp.cos(s.beta)**2
↪  - 2.0*s.L_wh*s.T_m*sp.cos(s.beta)**2 + 2.0*s.R_w*s.T_m*s.mu*sp.cos(s.beta)*sp.sign(s.d_xw) -
↪  s.R_w*s.W_wf**2*s.d_beta**2*s.m_ft*s.mu*sp.sin(s.beta)*sp.sign(s.d_xw) +
↪  s.R_w*s.W_wf*s.dd_xw*s.m_ft*s.mu*sp.cos(s.beta)**2*sp.sign(s.d_xw))/⌋
↪  (s.L_wh*s.R_w*(s.mu*sp.sin(s.beta)*sp.sign(s.d_xw) +
↪  sp.cos(s.beta))*sp.cos(s.beta))

#% Q-filter
numerator: [10*pi]
denominator: [1, 10*pi]

#% Motor controller
#& Effort controller for both wheels ----------------------------
wheel_joint_effort_controller:
type: "effort_controllers/JointGroupEffortController"
joints: ["left_wheel_joint", "right_wheel_joint"]
pid: {p: 1.0, i: 0.0, d: 0.0}

#& Publish all joint states -----------------------------------
joint_state_controller:
type: joint_state_controller/JointStateController
publish_rate: 50
```

Snippet E.4: `config_controller.yaml`

## E.5 Trolley model parameters

YAML configuration file containing model- and environment parameter values.

```yaml
## Frame
frame_height: 0.95 #@ [m] - L_WH
frame_width: 0.31 #@ [m]
frame_depth: 0.015 #@ [m] - W_F
frame_mass: 0.9 #@ [kg] - m_F

## Bag
bag_height: 0.65 #@ [m] - L_B
bag_width: 0.35 #@ [m]
bag_depth: 0.22 #@ [m] - W_B
bag_mass: 1.2 #@ [kg] - m_B

bag_mass_added: 20 #@ [kg]
#bag_mass_added: 10 #@ [kg]
bag_mass_added_nom: 0 #& [kg] Empty bag

## Wheel
wheel_radius: 0.11 #@ [m] - R_W
wheel_thickness: 0.08 #@ [m]

#@ m_W
#wheel_mass: 3.9 #@ [kg] wheel + motor
wheel_mass: 1.0 #@ [kg] wheel + motor

## Handle
#@ [m] Height at which the trolley handle is held during use (dined)
#@ L_H
handle_height_min: 0.661 #@ [m]
handle_height_max: 0.823 #@ [m]
handle_height_nom: 0.742 #& [m] Avg

## Coefficients
#@ [-] Rolling friction coefficient based on surface and bearing friction
#@ mu
friction_coeff_min: 0.102 #@ [-]
friction_coeff_max: 0.22 #@ [-]
friction_coeff_nom: 0.01 #& [-] Friction of car tires on smooth surface

## Other
gravity: 9.81 #@ [m/s^2] - g
```

Snippet E.5: `config_description.yaml`

# Appendix F

# Q-filter node unit- and integration tests

This appendix contains code of the different files involved in performing unit- and integration tests of the Q-filter node and its internal computations. The relevant code of the Q-filter node itself is specified in chapter 5: ‘Implementation’, snippet 5.3.

## F.1 Q-filter-node test script

Relevant Python code of unit- and integration tests used for testing the Q-filter node’s implementation.

```python
SUBSCRIBER_TOPIC = '/trolley/controller/disturbance_estimate_filtered'
SERVICE_TOPIC = '/trolley/tests/set_disturbance_estimate_unfiltered'

class TestQFilterNode(unittest.TestCase):

    def setUp(self):
        #^ init params
        config_general = load_config('trolley_general')
        config_controller = load_config('trolley_controller')
        config_tests = load_config('trolley_tests')

        self.loop_delay = config_general['loop_delay']
        self.timeout = config_tests['timeout']
        rate = config_controller['update_rate']

        #@ load Q-filter from controller config file
        q_num = [float(convert_string_to_sympy(x, np.pi, 'pi')) for x in
        ↪  config_controller['numerator']]
        q_denom = [float(convert_string_to_sympy(x, np.pi, 'pi')) for x in
        ↪  config_controller['denominator']]
        self.q_filter = generate_q_filter_discrete(q_num, q_denom, 1. / rate)  #@ transform num and
        ↪  denom to discrete Q-filter equation using the same function as used by the Q-filter node
        ↪  itself

        self.trigger_message_received = False
        self.trigger_gather_data = False
        self.output_received = []  #@ used for storing input data

        #^ init publishers/subscribers/services
        rospy.Subscriber(SUBSCRIBER_TOPIC, Float64, self.callback)
        rospy.wait_for_service(SERVICE_TOPIC, timeout=self.timeout)
        self.set_disturbance_estimate_srv = rospy.ServiceProxy(SERVICE_TOPIC, SetDisturbanceEstimate)

    def callback(self, msg):
        """Store incoming Q-filter output data."""
        self.trigger_message_received = True
        if not self.trigger_gather_data:  #@ wait until triggered
            return

        self.output_received.append(msg.data)  #@ store data
        if len(self.output_received) >= 5:  #@ gather 5 datapoints
            self.trigger_gather_data = False

    def test_q_filter_difference_equation(self):
        """Test output of Q-filter equation y[n] = a x[n-1] - b y [n-1] generated in the Q-filter node with expectations."""
        dt = 0.02  #@ time step
        var = 5 * 2 * np.pi  #@ transform from Hz to rad/s
        q_filter = generate_q_filter_discrete([var], [1, var], dt)  #@ transform num and denom to discrete Q-filter equation using the same function as used by the Q-filter node itself

        #@ specify (x[n-1], y[n-1], y[n]) for different test-scenarios. a and b are expected output values based on the discrete Q-filter fraction.
        a = 0.46651191
        b = -0.53348809
        inputs = [(0, 0, 0), (1, 0, a), (0, 1, -b), (1, 1, a - b)]
        for x_prev, y_prev, y_exp in inputs:  #@ loop over scenarios
            y_calc = q_filter(x_prev, y_prev)  #@ apply internal Q-filter
            self.assertAlmostEqual(y_calc, y_exp)  #@ test equality between calculated and expected output

    def test_q_filter_node_IO(self):
        """Test if Q-filter node processes input data correctly."""
        #@ let input-value publisher node publish a unit step on the Q-filter's input topic
        self.set_disturbance_estimate_srv(0)
        self.hold(0.5)  #@ wait
        self.set_disturbance_estimate_srv(1)
        self.hold(0.1)  #@ wait
        self.trigger_gather_data = True  #@ start storing data

        #@ wait until enough data is received
        timeout = rospy.get_time() + self.timeout
        while self.trigger_gather_data:
            self.hold(0.1)

        #^ test received data
        for i in range(1, len(self.output_received)):  #@ loop over stored data
            output_exp = self.q_filter(1, self.output_received[i - 1])  #@ calculate expected output using the Q-filter function directly
            self.assertAlmostEqual(self.output_received[i], output_exp)  #@ test equality between output received from Q-filter node and expected output.
```

Snippet F.1: Relevant lines of `test_q_filter_node.py`

## F.2 Q-filter-node test launch file

Content of Robotic Operating System (ROS)-test XML launch file, which launches all elements necessary for the execution of the Q-filter node tests in an isolated environment.

```xml
<launch>
    <node name="disturbance_estimate_unfiltered_publisher" pkg="trolley_tests"
    ↪  type="disturbance_estimate_unfiltered_publisher.py"/>
    <node name="q_filter_node" pkg="trolley_controller" type="q_filter_node.py"/>

    <test test-name="test_q_filter_node" pkg="trolley_tests" type="test_q_filter_node.py"/>
</launch>
```

Snippet F.2: `test_q_filter_node.test`

# Appendix G

# Simulation-element descriptions

This appendix contains the relevant code of the trolley description for its implementation in simulation, using the URDF format. Parameters, recognizable as `${...}`, are obtained from the model- and controller configuration files, which are included in sections E.4 and E.5.

## G.1 Trolley-model general components

The trolley model's general components describe the trolley physical structure, as well as its physical properties and possible interfaces used for the coupling of plugins.

```
<robot name="trolley"
    <xacro:include filename="$(find trolley_description)/urdf/trolley.gazebo" />

    <!--& Frame  -->
    <link name="frame_link">
        <collision>
            <origin xyz="-${frame_height/2} 0 0" rpy="0 0 0"/>
            <geometry>
                <box size="${frame_height} ${frame_width} ${frame_depth}"/>
            </geometry>
        </collision>
        <visual>
            <origin xyz="-${frame_height/2} 0 0" rpy="0 0 0"/>
            <geometry>
                <box size="${frame_height} ${frame_width} ${frame_depth}"/>
            </geometry>
        </visual>
        <xacro:box_inertia m="${frame_mass}" w="${frame_width}" d="${frame_height}"
        ↪  h="${frame_depth}" x="-${frame_height/2}"/>
    </link>

    <joint name="frame_joint" type="fixed">
        <parent link="base_footprint"/>
        <child link="frame_link"/>
        <origin xyz="0 0 ${handle_height}" rpy="0 -${PI/2 - trolley_angle} 0"/>
    </joint>

    <!--& Bag  -->
    <link name="bag_link">
        <collision>
            <geometry>
                <box size="${bag_height} ${bag_width} ${bag_depth}"/>
            </geometry>
        </collision>
```

```xml
        <visual>
            <geometry>
                <mesh filename="package://trolley_simulation/meshes/trolley_bag.stl"/>
            </geometry>
        </visual>
        <xacro:triangle_inertia m="${bag_mass_total}" w="${bag_width}" d="${bag_depth}" h="${bag_height}"/>
    </link>

    <joint name="bag_joint" type="fixed">
        <parent link="frame_link"/>
        <child link="bag_link"/>
        <origin xyz="${-frame_height + bag_height*0.5} 0 ${(frame_depth + bag_depth)*0.5}"/>
    </joint>

    <!--& Wheels -->
    <xacro:macro name="wheel" params="prefix reflect">
        <link name="${prefix}_wheel_link">
            <visual>
                <origin rpy="${-PI*0.5} 0 0"/>
                <geometry>
                    <cylinder radius="${wheel_radius}" length="${wheel_thickness}"/>
                </geometry>
            </visual>
            <collision>
                <origin rpy="${-PI*0.5} 0 0"/>
                <geometry>
                    <cylinder radius="${wheel_radius}" length="${wheel_thickness}"/>
                </geometry>
            </collision>
            <xacro:cylinder_inertia m="${wheel_mass}" r="${wheel_radius}" t="${wheel_thickness}" roll="${-PI*0.5}"/>
        </link>

        <joint name="${prefix}_wheel_joint" type="continuous">
            <parent link="frame_link"/>
            <child link="${prefix}_wheel_link"/>
            <origin xyz="${-frame_height} ${(frame_width + wheel_thickness)/2*reflect} 0"/>
            <axis xyz="0 1 0"/>
            <!--<dynamics friction="${wheel_friction}"/>-->
        </joint>

        <transmission name="${prefix}_wheel_transmission">
            <type>transmission_interface/SimpleTransmission</type>
            <joint name="${prefix}_wheel_joint">
                <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
            </joint>
            <actuator name="${prefix}_wheel_motor">
                <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
            </actuator>
        </transmission>
    </xacro:macro>
    <xacro:wheel prefix="left" reflect="1"/>
    <xacro:wheel prefix="right" reflect="-1"/>
</robot>
```

Snippet G.1: Relevant lines of `trolley.urdf.xacro`

## G.2 Trolley-model Gazebo-specific components

The trolley model's Gazebo-specific components contain additional physical properties, as well as the plugins used for the communication between different elements within Robotic Operating System (ROS) and Gazebo.

```
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
    <!--& Static wheel friction settings -->
    <xacro:macro name="wheel_friction" params="prefix support">
        <gazebo reference="${prefix}${support}_wheel_link">
            <mu1 value="1.0"/>
            <mu2 value="1.0"/>
            <kp value="500000.0" />
        </gazebo>
    </xacro:macro>
    <xacro:wheel_friction prefix="left" support=""/>
    <xacro:wheel_friction prefix="right" support=""/>
    <xacro:wheel_friction prefix="left" support="_support"/>
    <xacro:wheel_friction prefix="right" support="_support"/>

    <!--& wheel joint effort plugin -->
    <gazebo>
        <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so"/>
        <robotNamespace>/trolley/plugins</robotNamespace>
    </gazebo>

    <!--& IMU plugin -->
    <gazebo>
        <plugin name="imu_plugin" filename="libgazebo_ros_imu.so">
            <alwaysOn>true</alwaysOn>
            <robotNamespace>/trolley/plugins</robotNamespace>
            <serviceName>imu_service</serviceName>
            <topicName>imu_measurement</topicName>
            <bodyName>base_footprint</bodyName>
            <frameName>base_footprint</frameName>
            <gaussianNoise>0.0</gaussianNoise>
            <xyzOffset>0 0 ${wheel_radius}</xyzOffset>
            <rpyOffset>0 0 0</rpyOffset>
            <updateRate>${update_rate}</updateRate>
        </plugin>
    </gazebo>

    <!--& rolling friction plugin -->
    <!--@ allow feedback from wheel joints -->
    <xacro:macro name="wheel_feedback" params="prefix">
        <gazebo reference="${prefix}_wheel_joint">
            <provideFeedback>true</provideFeedback>
        </gazebo>
    </xacro:macro>
    <xacro:wheel_feedback prefix="left"/>
    <xacro:wheel_feedback prefix="right"/>

    <!--@ load plugin -->
    <gazebo>
        <plugin name="rolling_friction_plugin" filename="librolling_friction_plugin.so">
            <target1>left_wheel</target1>
            <target2>right_wheel</target2>
            <mu>${mu}</mu>
            <wheel_mass>${wheel_mass}</wheel_mass>
        </plugin>
    </gazebo>

    <!--& handle force plugin -->
    <gazebo>
        <plugin name="handle_force_plugin" filename="libapply_force_plugin.so">
            <topic>/trolley/plugins/handle_force_measurement</topic>
            <link>base_footprint</link>
            <x>0</x>
            <y>0</y>
            <z>${handle_height}</z>
        </plugin>
    </gazebo>
</robot>
```

Snippet G.2: Relevant lines of `trolley.gazebo`

## G.3 World description

The world description includes general physics elements.

```
<sdf version="1.6">
  <world name="sim_world">
    <!-- Add components -->
    <include>
      <uri>model://ground_plane</uri>
    </include>

    <!-- Set gravity -->
    <gravity>0 0 -9.81</gravity>

    <!-- Physics engine settings -->
    <physics type="ode">
      <ode>
        <solver>
          <type>quick</type>
          <iters>150</iters>
        </solver>
      </ode>
    </physics>
  </world>
</sdf>
```

Snippet G.3: Relevant lines of `sim.world`

# Appendix H

# Custom plugins

This appendix contains the relevant code of the custom plugins, built for the implementation of custom functionality within the simulation.

## H.1 Handle force plugin

The handle force plugin applies a provided force command to a link in simulation. The force command is provided via a specified topic, and it is applied in the target link's local reference frame.

```
namespace gazebo
{
  class ApplyForce : public ModelPlugin
  {
  public:
    void Load(physics::ModelPtr _parent, sdf::ElementPtr _sdf)
    {
      this->updateConnection_ =
      ↪  event::Events::ConnectWorldUpdateBegin(std::bind(&ApplyForce::OnUpdate, this)); // Listen
      ↪  to Gazebo's update event. This event is broadcast every simulation iteration.

      this->rosNode.reset(new ros::NodeHandle("apply_force_plugin_node")); // Create ROS node
      this->topic = "/trolley/plugins/handle_force_measurement"; // Set subscriber topic
      this->rosSub = this->rosNode->subscribe<geometry_msgs::Vector3>(this->topic, 1,
      ↪  &ApplyForce::OnRosMsg, this); // Initialize subscriber
    }

    void OnUpdate() // Called by the world update start event
    {
      this->link_->AddLinkForce(this->force, this->pos); // Apply stored force
    }

    void OnRosMsg(const geometry_msgs::Vector3ConstPtr &_msg) // Subscriber callback
    {
      this->force = ignition::math::Vector3d(_msg->x, _msg->y, _msg->z); // Store force
    }

  private:
    physics::ModelPtr model_; // Pointer to the model
    physics::LinkPtr link_; // Pointers to the joints
    std::unique_ptr<ros::NodeHandle> rosNode; // A node used for ROS transport
    ros::Subscriber rosSub; // A ROS subscriber

    std::string topic; // Topic name
    ignition::math::Vector3d pos; // Store force application position
    ignition::math::Vector3d force; // Store most recent force value
  };
}
```

Snippet H.1: Relevant lines of `apply_force_plugin.cc`

## H.2 Rolling friction plugin

The rolling friction plugin applies a variable rolling friction torque on the trolley wheels.

```
namespace gazebo
{
  float calculate_sign_with_deadband(float value, float deadband)
  { // Return the sign of value, return 0 if absolute(value) < deadband
    if (value > deadband) // Outside deadband, positive
      return 1.;
    else if (value < -deadband) // Outside deadband, negative
      return -1.;
    return 0.; // Within deadband
  }

  class RollingFriction : public ModelPlugin
  {
  public:
    void Load(physics::ModelPtr _parent, sdf::ElementPtr _sdf) // Initialize plugin
    {
      this->_model = _parent; // Store the pointer to the model

      // Set params from config files and parameter server
      this->gravity = config_description["gravity"].as<float>();
      this->deadband = config_general["velocity_deadband"].as<float>(); // deadband threshold
      this->mu = _sdf->GetElement("mu")->Get<float>(); // friction coefficient value
      this->wheel_radius = config_description["wheel_radius"].as<float>();
      this->wheel_mass = _sdf->GetElement("wheel_mass")->Get<float>();

      // Set pointers to links/joints
      std::string target1_name = _sdf->GetElement("target1")->Get<std::string>();
      std::string target2_name = _sdf->GetElement("target2")->Get<std::string>();
      this->_link1 = this->_model->GetLink(target1_name + "_link");
      this->_link2 = this->_model->GetLink(target2_name + "_link");
      this->_joint1 = this->_model->GetJoint(target1_name + "_joint");
      this->_joint2 = this->_model->GetJoint(target2_name + "_joint");

      this->updateConnection =
      ↪  event::Events::ConnectWorldUpdateBegin(std::bind(&RollingFriction::OnUpdate, this)); //
      ↪  Listen to the update event. This event is broadcast every simulation iteration.
    }

    void OnUpdate() // Called by the world update start event
    {
      if (ros::Time::now() > ros::Time(0.2)) // Wait for params values to stabilize
      {
        if (this->trigger_calibration)
          this->CalibrateVelocity(); // Save velocity measurement offset
        this->GetAngularWheelVelocity(); // Save angular wheel velocity obtained from Gazebo
        this->GetNormalForce(); // Save normal force obtained from Gazebo
```

```cpp
        this->SetRollingFriction(); // Apply friction torque to targets
      }
    }

    void CalibrateVelocity() // Save velocity measurement offset
    {
      this->GetAngularWheelVelocity(); // Save angular wheel velocity obtained from Gazebo
      this->velocity_offset = this->wheel_angular_velocity; // Save current velocity as offset
      this->trigger_calibration = false; // Disable calibration
    }

    void SetRollingFriction() // Apply friction torque to targets
    {
      double rolling_friction_value = this->mu * this->normal_force * this->wheel_radius; //
      ↪  Calculate rolling friction
      float direction = -calculate_sign_with_deadband(this->wheel_angular_velocity, this->deadband);
      ↪  // Apply deadband
      this->rolling_friction = rolling_friction_value * direction;

      // Apply rolling friction to targets
      ignition::math::Vector3d rolling_friction_vector(0, this->rolling_friction, 0);
      this->_link1->AddRelativeTorque(rolling_friction_vector);
      this->_link2->AddRelativeTorque(rolling_friction_vector);
    }

    void GetNormalForce() // Save normal force obtained from Gazebo
    {
      // Get force applied by parent link on wheel joint, specified in parent link frame
      float wheel_force_1 = this->_joint1->GetForceTorque(0).body1Force.Z();
      float wheel_force_2 = this->_joint2->GetForceTorque(0).body1Force.Z();

      float wheel_force_gravity = this->wheel_mass * this->gravity; // Calculate wheel gravity force
      this->normal_force = -(wheel_force_1 + wheel_force_2) / 2 + wheel_force_gravity; // Calculate
      ↪  and save normal force per wheel
    }

    void GetAngularWheelVelocity() // Save angular wheel velocity obtained from Gazebo
    {
      // Get angular velocity per wheel from Gazebo
      ignition::math::Vector3d angular_velocity_vector_1 = this->_link1->RelativeAngularVel();
      ignition::math::Vector3d angular_velocity_vector_2 = this->_link2->RelativeAngularVel();
      double angular_velocity_1 = angular_velocity_vector_1.Y() - this->velocity_offset;
      double angular_velocity_2 = angular_velocity_vector_2.Y() - this->velocity_offset;

      this->wheel_angular_velocity = (angular_velocity_1 + angular_velocity_2) / 2.; // Calculate and
      ↪  save average
    }

  private:
    // Pointers
    physics::ModelPtr _model;
    physics::LinkPtr _link1;
    physics::LinkPtr _link2;
    physics::JointPtr _joint1;
    physics::JointPtr _joint2;

    // Constants
    float gravity;
    float wheel_radius;
    float wheel_mass;
    float mu; // Friction coefficient
    double deadband; // Velocity deadband threshold

    // Get from Gazebo simulation
    float normal_force = 0;
    double wheel_angular_velocity = 0;
    double velocity_offset = 0;

    // Get from plugin
    double rolling_friction = 0;
    bool trigger_calibration = true;
  };
}
```

Snippet H.2: Relevant lines of `rolling_friction_plugin.cc`

# Bibliography

## Documentation

[1] *SDFormat Specification.* SDFormat.org. URL: http://sdformat.org/spec.

[2] *URDF Specification.* ROS Wiki. URL: http://wiki.ros.org/urdf/XML.

[3] *GazeboRosControlPlugin Class Reference.* ROS Documentation. URL: http://docs.ros.org/en/melodic/api/gazebo_ros_control/html/classgazebo__ros__control_1_1GazeboRosControlPlugin.html.

[4] *Gazebo ApplyBodyWrench Documentation.* ROS Documentation. URL: http://docs.ros.org/en/melodic/api/gazebo_msgs/html/srv/ApplyBodyWrench.html.

[5] *Gazebo Link Class Reference.* Gazebo API Reference. URL: http://osrf-distributions.s3.amazonaws.com/gazebo/api/9.0.0/classgazebo_1_1physics_1_1Link.html#a2b5d6a125fba8b6faea62de2611ef991.

[6] *Gazebo Joint Class Reference.* Gazebo API Reference. URL: http://osrf-distributions.s3.amazonaws.com/gazebo/api/9.0.0/classgazebo_1_1physics_1_1Joint.html.

[7] *IMU Class Reference.* ROS Documentation. URL: http://docs.ros.org/en/melodic/api/gazebo_plugins/html/classgazebo_1_1GazeboRosIMU.html.

## References

[8] Reka Hajnovicsova. “Pull-E- The Assistive Shopping Trolley”. Masters Thesis. Delft: Delft University of Technology, 2017.

[9] V. Heusdens. “Assistive Shopping Trolley Control for the Elderly - A Literature Review”. Literature review. Delft: Delft University of Technology, 2019.

[10] Nederlandse Zorgautoriteit (NZa). *Zorg Voor Ouderen.* Monitor. 2018.

[11] *Basic Shopping Trolley.* URL: https://www.bol.com/nl/p/luxery-shopping-trolley-antracite/9200000096361148/.

[12] Adam Zeidan. *Friction.* In: *Encyclopædia Britannica.* Encyclopædia Britannica, inc., 2020.

[13] J. D.G. Kooijman, A. L. Schwab, and J. P. Meijaard. “Experimental Validation of a Model of an Uncontrolled Bicycle”. In: *Multibody System Dynamics* 19.1-2 (Feb. 2008), pp. 115–132.

[14] Xuan Fan. “Intelligent Power Assist Algorithms for Electric Bicycles”. Dissertation. Berkeley: University of California, 2010.

[15] Rijksoverheid. *Technische bouwvoorschriften uit het oogpunt van veiligheid.* Bouwbesluit. 2012.

[16] D.A. Abbink. *Frequency-Domain Analyses and Modeling.* Lecture notes. Delft: Delft University of Technology, 2017.

[17] J. P. Meijaard and A. L. Schwab. “Linearized Equations for an Extended Bicycle Model”. In: *European Conference on Computational Mechanics, Solids, Structures and Coupled Problems in Engineering.* Lisbon, 2006.

[18] A.G.C. van Boeijen et al. *Delft Design Guide.* Ed. by Annemiek van Boeijen et al. 1st ed. Delft: BIS Publishers, 2013.

[19] Jakob Fjellström and Adam Maaninka. “Shopping Cart Dynamic Assistance System”. Masters Thesis. Gothenburg: Chalmers University of Technology, 2014.

[20] P Padmagirisan, R Sowmya, and V Sankaranarayanan. “Power-Assist Control of a Human–Electric Hybrid Bicycle with Energy Regeneration and Cruise Control”. In: *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 233.2 (Feb. 2019), pp. 179–191.

[21] Zhu Deyi, Zhang Kai, and Yin Dejun. “A New Torque Control Approach for Electric Power Assisted Bicycle Based on Model-Following Control”. In: *2017 International Conference on Information, Communication and Engineering (ICICE)*. IEEE, Nov. 2017, pp. 36–39.

[22] Hiroshi Niki, Junichi Miyata, and Toshiyuki Murakami. “Power Assist Control of Electric Bicycle Taking Environment and Rider’s Condition into Account”. In: ed. by Farrokh Janabi-Sharifi. International Society for Optics and Photonics, Dec. 2005.

[23] D. S. Cheon and K. H. Nam. “Pedaling Torque Sensor-Less Power Assist Control of an Electric Bike via Model-Based Impedance Control”. In: *International Journal of Automotive Technology* 18.2 (Apr. 2017), pp. 327–333.

[24] Chi-Ying Liang, Wai-Hon Lin, and B. Chang. “Applying Fuzzy Logic Control to an Electric Bicycle”. In: First International Conference on Innovative Computing, Information and Control. Vol. 1. IEEE, 2006, pp. 513–516.

[25] Human-Centered Design Department. “Anthropometric Database”. Delft University of Technology.

[26] Ian Hutchings and Philip Shipway. “Applications and Case Studies”. In: *Tribology - Friction and Wear of Engineering Materials*. Elsevier, Jan. 2017, pp. 303–352.

[27] *Rolling Resistance*. Engineering Toolbox. 2008. URL: `https://www.engineeringtoolbox.com/rolling-friction-resistance-d_1303.html`.

[28] Julius O. Smith. *Introduction to Digital Filters with Audio Applications*.

[29] Emre Sariyildiz and Kouhei Ohnishi. “A Guide to Design Disturbance Observer”. In: *Journal of Dynamic Systems, Measurement, and Control* 136.2 (Mar. 1, 2014), p. 021011.

[30] Tony Kelman. *Disturbance Observers*. Lecture notes. UC Berkeley, 2016.

[31] Kemin Zhou and John C Doyle. *Essentials of Robust Control*. Prentice Hall, 1999.

[32] Sigurd Skogestad and Ian Postlethwaite. *Multivariable Feedback Control - Analysis and Design*. 2nd ed. John Wiley & Sons, 2001.

[33] Karl Johan Åström and Richard M Murray. *Feedback Systems - An Introduction for Scientists and Engineers*. Princeton, Oxford: Prinseton University Press, 2009.

[34] Alexander Slocum. *Bearings*. 2008.

[35] Tom van den Boom and Bart de Schutter. *Optimization in Systems and Control*. Lecture notes. Delft: Delft Center for Systems and Control; Delft University of Technology, 2018.

[36] *DDA Handrail Guidelines*. SG System Products. URL: `https://www.handrailsuk.co.uk/dda-handrail-guidelines`.

[37] Altena TV. *Burgerstem Altena nodigt wethouder uit voor ritje in rolstoel*. Altena TV. Aug. 20, 2019. URL: `https://www.altenatv.nl/2019/08/20/burgerstem-altena-nodigt-wethouder-uit-voor-ritje-in-rolstoel/`.

[38] Struykverwoinfra. *Inritband*. struykverwoinfra.nl. URL: `https://www.struykverwoinfra.nl/productselector/banden/trottoirbanden-13-15/inritbanden-13-15/inritbanden-45x18x100-vb.html`.

[39] Richard W Bohannon. “Comfortable and Maximum Walking Speed of Adults Aged 20-79 Years: Reference Values and Determinants”. In: *Age and Ageing* 26 (1997), pp. 15–19.

[40] Reka Bérci-Hajnovics. “Introduction to ROS”. Almende show-and-tell (Rotterdam).

[41] *About ROS*. URL: `https://www.ros.org/about-ros/`.

[42] *ROS Introduction*. ROS Wiki. URL: `http://wiki.ros.org/ROS/Introduction`.