CHALMERS
UNIVERSITY OF TECHNOLOGY

Closing the loop in a treadmill-based 2-wheeler simulator

Development of a Communication Bridge and PI-Controller for a Treadmill

Bachelor degree thesis project report in Mechatronics engineering

Ahmad Yasin
Ehsan Hassen Mahamed

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Degree project report 2026

Closing the loop in a treadmill-based 2-wheeler simulator

Development of a Communication Bridge and PI-Controller for a Treadmill

Ahmad Yasin
Ehsan Hassen Mahamed

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and maritime sciences
Chalmers University of Technology
Gothenburg, Sweden 2026

Closing the loop in a treadmill-based 2-wheeler simulator
Development of a Communication Bridge and PI-Controller for a Treadmill
Ahmad Yasin, Ehsan Hassen Mahamed

Supervisor: Marco Dozza, Mechanical Engineering
Examiner: Fredrik Bruzelius, Mechanical Engineering

Degree project report 2026
Department of Mechanics and Maritime Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Closing the loop in a treadmill-based 2-wheeler simulator
Development of a Communication Bridge and PI-Controller for a Treadmill
Ahmad Yasin
Ehsan Hassen Mahamed
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

This thesis project aims to further develop an existing simulator that uses a large-scale treadmill from Rodby AB as a driving platform to simulate a real driving environment for 2-wheeled vehicles (e-scooter). To keep the vehicle centered on the treadmill a rope is attached to the vehicle, which introduces unwanted interference. Additionally, a motion capture system from Qualisys AB is used to track the movement and position of the rider. This serves as a controlled environment to study behaviors of riders on 2-wheeled vehicles, such as riding under the influence of alcohol. For a more realistic simulation, a closed-loop control system that adjusts the speed of the treadmill based on the position of the e-scooter was implemented, replacing the rope. The objective of the project was to stabilize a 2-wheeled vehicle at the center of the treadmill without the use of physical restraints. Position measurement was implemented using high precision Qualisys optical motion capture system that transmits position information in the 6 Degrees of Freedom (6DoF) in real time to a custom control system built in Python.

A custom Proportional-Integral (PI) control logic was implemented to calculate and control the speed of the treadmill depending on the riders position. It was able to adjust the speed of the treadmill depending on the riders position but was not fast enough to keep the rider centered on the treadmill when the rider accelerated aggressively, due to the treadmills limited acceleration capabilities compared to the e-scooter. In addition to software latency which further slows the system. Further development is required for the control system to operate accurately to simulate a real driving environment for 2-wheeled vehicles.

Keywords: Closed-Loop system, PI Controller, Motion Capture, Qualisys Track Manager, E-scooter

Acknowledgements

We would like to express our sincere gratitude to our supervisors Marco Dozza and Fredrik Bruzelius for their guidance, expertise, and continuous support throughout the development of this thesis. We also extend a special thank you to Henrik Hörlin for his technical assistance with the configuration and calibration of the Qualisys Track Manager system in the laboratory.
Furthermore, we wish to acknowledge Chalmers University of Technology for providing the advanced hardware and facilities necessary to complete this project.

Ahmad Yasin, Gothenburg, June 2026
Ehsan Hassen Mahamed Gothenburg, June 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

API	Application Programming Interface. Protocols that allows different software programs to communicate and share data with each other.
CRC	Cyclic Redundancy Check. An algorithm used to detect errors in digital data.
DTR	Data Terminal Ready. A signal showing that a device is ready to communicate
GUI	Graphical User Interface. A visual interface for interacting with electronic devices.
HMI	Human-Machine Interface. A user interface or dashboard that connects a person to a machine, system, or device
PI	Proportional-Integral. Used to fixe errors by combining current and past feedback to keep systems like speed perfectly steady.
RTS	Request to Send. A hardware flow control signal in serial communication used to indicate that a device has data ready to transmit.
RTU	Remote Terminal Unit. A device that links physical hardware to a control system.
RS	Recommended Standard. A series of physical standards for serial data transmission that define electrical characteristics and signal timing.
SDK	Software Development Kit. A collection of software development tools and libraries provided by a vendor to facilitate the creation of applications for specific platforms
UART	Universal Asynchronous Receiver-Transmitter.A physical hardware component that handles asynchronous serial communication by translating data between parallel and serial forms.
QTM	Qualisys Track Manager.A proprietary motion capture software application used for capturing, processing, and analyzing 3D kinematic data.
6DoF	6 Degrees of Freedom, six variable. The measurement of a rigid body's position (X, Y, Z) and orientation (pitch, yaw, roll) in three-dimensional space.

List of Figures

Contents

1
Introduction

1.1 Background

The use of e-scooter as a means of urban transport has increased significantly in recent years. Consequently, the number of traffic accidents involving e-scooters has also increased. In 2024, five people died and 4599 people were injured in e-scooter accidents in Sweden [1], representing an increase of approximately 30% compared to previous years.

One contributing factor to these accidents is human behavior in traffic, such as riding with one hand, using a mobile phone simultaneously, and riding under the influence of alcohol [2],[3]. Studying the impact of human behaviors on e-scooter accidents is therefore crucial to reducing accident risks in the future. However, studying behaviors such as drunk driving in a real traffic environment is both dangerous and unethical. A controlled environment is therefore essential for conducting such studies, as it ensures safety and allows control over variables that can affect the result, such as traffic conditions, time of day, and weather. In Addition, controlled environments improve repeatability, enabling the same scenario to be replicated and thus increasing the reliability of the findings.

To study how people control and interact with two-wheeled vehicles in a controlled laboratory environment, a large-scale treadmill combined with a high-precision motion capture system is used in the FUSE laboratory at Chalmers University of Technology. The treadmill serves as a platform for riding two-wheeled vehicles, whereas the motion capture system tracks the movement and position of the rider. This setup provides a platform for simulating near-realistic riding conditions, particularly for studying two-wheeled vehicle riders, riding in a close to straight-line. A virtual environment is presented to the rider using a TV screen, allowing interaction with simulated surroundings.

For safety, the treadmill is equipped with an emergency stop button that immediately stops the treadmill when activated. In addition, a separate emergency switch is mounted on the ceiling of the lab connected to a harness worn by the rider. If the rider loses control, the harness is pulled, triggering the switch and causing the treadmill to stop immediately. This prevents the rider from being pushed into the surrounding environment.

For current studies, a rope is attached to the vehicle to control its longitudinal position while riding on the treadmill. This design allows the rider to focus on the lateral movement rather than the longitudinal position. However, the rope introduces unintended constraints on the lateral movement of the rider. Furthermore, the speed of the treadmill is controlled manually using the manufacturers interface (Rodby Control System 2.0), which prevents the rider from naturally accelerating and decelerating.

1.2 Purpose

The projects aim is to create a more realistic rider simulator, by removing the rope used for longitudinal positioning and replacing it with an automated control system. Implementing a feedback loop, by using position data from the motion capture system to control the treadmill speed, allowing the rider to remain centered while driving at their preferred pace.

Since regular treadmills are designed to operate based on moving at a certain speed, they lack the dynamic response required for 2-wheeled vehicles. The purpose of this research is therefore to fill the gap between human centric treadmill and vehicle dynamics through the use of a motion capture mechanism together with a specialized software program.

1.3 Goals

The primary goal of this project is to create a working prototype that combines both hardware and software. For an successful project, the following goals must be met:

- **Communication and Actuation:**
  Create a two-way interface that could read and send data to the treadmill motherboard.
- **Real-time Data Stream:**
  Capture the riders position in real-time with Qualisys Six Degrees of Freedom (6DoF) Tracking system.
- **Control System (PI):**
  Develop a PI controller that maintains the stability of the e-scooter in relation to its positioning on the treadmill by altering the belt speed accordingly to keep the e-scooter centered on the treadmill.
- **System safety:**
  The system should feel safe to ride on with an e-scooter. With a kill-switch boundary on the edges of the treadmill.

1.4 Limitations

- **Environmental Constraints:**
  Because the system relies on the Qualisys Track Manager (QTM), the solution is strictly limited to indoor laboratory environments. The system requires uninterrupted data from the motion cameras. Extended visual occlusion cannot be actively compensated for beyond the implemented safety gear.
- **Riders Safety:**
  Because operating a vehicle on a treadmill introduces significant risks to the rider, the project is strictly limited to only be tested in an already mechanically safe environment. The developed software control operates under the assumption that the rider is constantly secured by the harness and a mechanical spring to kill-switch. The software does not attempt to predict human fall or balance loss.
- **Rodby Hardware:**
  There exists an infrared camera installed in the Rodby treadmill for automatically controlling the belt speed depending on the longitudinal location of the runner. Nevertheless, this native control arrangement was invalidated for two reasons.

  Firstly, no data could be extracted from the infrared camera installed by Rodby nor the automatic control for the belt speed. Secondly, the system was built to read a runner position and does not account for 2-wheel vehicles. Both reasons eliminates any type of modification to the control system and future research of human biomechanics. Only motion cameras provided by Qualisys will be used through out the project.

2

Theory

This section presents the theory necessary to understand the design and implementation of the control system developed during the thesis project. The system consists of a treadmill connected to an external computer via an RS-232 communication line that uses the Modbus RTU protocol over it to send commands to the treadmill. In addition to the motion capture system that provides the position of the rider in real-time which serves as the input to the control system. The control algorithm is based on a proportional-integral controller and the entire control system is implemented in Python, using three libraries, namely QTM Real-time SDK, PySerial, and asyncio.

2.1 Rodby Skate Band RL3500E

Rodby Innovation AB is a Swedish engineering company specializing in the design and manufacturing of advanced, custom-built treadmills. Rodby primarily develops high performance equipment for elite sports testing, medical rehabilitation, and physiological research.

One of Rodbys products is the skate band RL3500Ex2500, a large-scale motorized treadmill developed for sports testing. While conventional treadmills are strictly dimensioned for standard human biomechanics for running or walking, the RL3500E is explicitly engineered as a skate treadmill with the hardware specification listed below

- Belt length: 3500 mm.
- Belt width: 2500 mm.
- Speed: 0-50 km/h.
- Elevation: 0-6°.
- Pulse Control with pulse band / clock Bluetooth®.
- Track Simulation GPX.
- Motor Effect: 10 – 14 kW.
- Emergency Stop Switch.

2.1.1 Rodby Control System 2.0

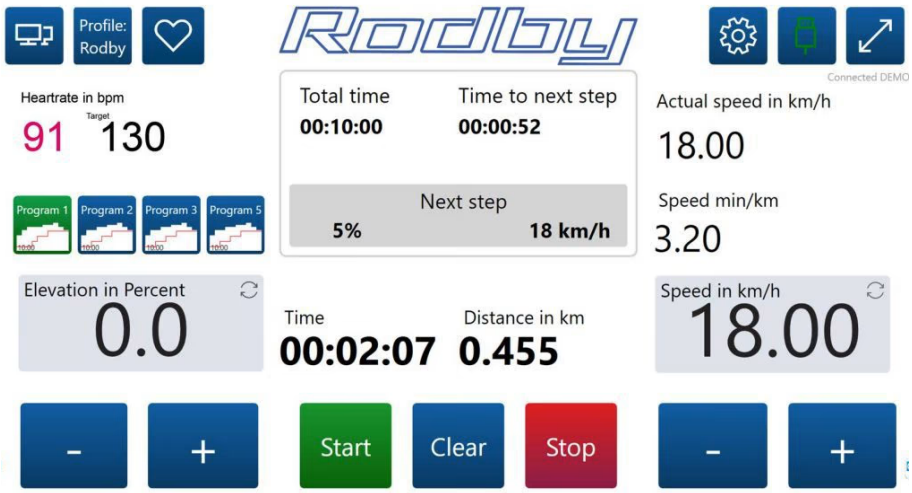

Figure 2.1: Rodby Interface

Rodby software (Rodby Control System 2.0) has a built in interface that control the treadmill manually. Rodby Control System 2.0 is installed with every treadmill on an private computer to prevent disturbances. The interface provide the the user to manually interact with treadmills speed, elevation and data such as heart-rate, actual speed, and time shown in Figure 2.1.

2.2 Serial Communication And RS-232

Serial communication is a method used to transfer data between devices, where data is transmitted sequentially, one bit at a time, over a single communication line [4]. Serial communication can be classified into synchronous and asynchronous communication [5]. Asynchronous communication relies on predefined communication parameters between the sender and the receiver.

One of the most widely used asynchronous serial communication standards is the RS-232 standard [6]. The standard is commonly used in industrial control systems, laboratory equipment, and embedded hardware.

For successful communication over RS-232, both devices must agree on a set of parameters before transmission begins [7]. These parameters define how each data frame is structured and include the baud rate, number of data bits, a start bit, a stop bit, and parity. A start bit indicates the beginning of a transmission frame, and a stop bit indicates its end.

The baud rate defines the transmission speed in bits per second (bps)[7]. The number of data bits specifies how many bits constitute each transmitted byte. Parity provides a basic form of error detection and can be set to even, odd, or none.

In addition to the communications parameters, the standard defines a set of hardware control signals that are independent of the data stream[7]. This includes RTS (Request to Send) and DRT (Data Terminal Ready). These control signals can be set on or off regardless of the data being transmitted.

Native RS-232 serial ports are typically no longer found in modern computers. Therefore, USB-to-serial adapters are used to convert between USB signals into RS-232 compatible signals that are expected by the target hardware.

2.3 Modbus RTU Protocol

Modbus is a widely used communication protocol based on a Master-Slave Architecture, the Master device initiates the communication, and the Slave device responds [8]. There are several variants of the Modbus protocol, including the Modbus RTU (Remote Terminal Unit) which is designed for serial communication interfaces such in RS-232.

Modbus RTU message frame consists of several fields, address, function code, data, and CRC (Cyclic Redundancy Check) field. The start of the massage is indicated with a silent interval equivalent to at least 3.5 character times [9]. The address field contains an 8-bit address to the Slave [9], which corresponds to the treadmill in this project. The function code field, also 8 bits, defines the operation requested by the Master. The data field contains the parameters associated with the requested function. The CRC field provides error detection by using a 16-bit value that is the result of a CRC-calculation preformed on the content of the message. Lastly, the end of the transmission is marked by a silent interval of at least 3.5 character times.

2.4 PI-Controller

A proportional controller (P-controller) is a controller that generates a control signal proportional to the error of a closed-loop control system, given by the difference between the setpoint and the current value of the system [10, kap.4]. The proportional controller is defined by equation (2.1):

$$u = K_p \cdot e \tag{2.1}$$

where,

u = control signal
e = error
K_p = proportional gain

A small K_p leads to a stabile but slow system [10, Cha.4]. Whereas a large K_p leads to a fast but unstable system which in this case would cause the rider to fall off the

treadmill. Selecting an appropriate K_p value is therefore crucial to achieving a fast and stable system.

A disadvantage with a P-controller, is that it gives rise to steady-state error [10, kap.4]. To eliminate this error, the P-controller is often combined with an integral controller (I-controller) forming a PI-controller. The I-controller accumulates the error over time and is defined by equation 2.2:

$$u = K_i \int e dt \tag{2.2}$$

where,

$K_i = \frac{K_p}{T_i}$ integrating gain
$T_i =$ integration time

The PI-controller is defined by adding equation 2.1 and 2.2:

$$u = K_p \cdot e + K_i \int e dt \tag{2.3}$$

A consequence of using the I-controller is a phenomenon called integrator windup [11, Cha.3]. It occurs when the control signal reaches the limits of the actuator, but the I-controller continues to accumulate the error over time. This leads to a large error and controller output while the control signal remains saturated causing a large transients. This is prevented by including an anti-windup protection such as conditional integration. This is a method that switches off the integral term when certain conditions are fulfilled.

2.5 Qualisys Motion Capture

Motion capture (mocap) is a technology that is used to track and record motion in a computer system to generate various types of data that can be used for different applications [12]. There are different types of mocap technology, including optical passive, which use cameras that emit infrared light and track retroreflective markers.

In addition to QTM which is the software that processes the data captured by the cameras and computes the three-dimensional position of each marker[13]. QTM can process data in real-time, which can be acquired by other applications in real-time over the network using Qualisys own protocol called QTM RT protocol.

In QTM, a rigid body can be defined to obtain only one position of an object instead of getting position data of the individual markers. A rigid body is a set of at least three markers whose relative positions are fixed. QTM computes the position of the entire rigid body as a single entity, even if one marker from the rigid body is not detected by the cameras, QTM is able to compute the position of the rigid body

based on the current visible markers.

2.6 Six Degrees of Freedom (6DoF) Tracking

Six Degrees of Freedom (6DoF) refers to the ability of a rigid body to move freely in a three dimensional environment. An object total position description within a 3D space needs a set of six independent variables. There are two kinds of independent variables, translation and rotation as seen in Figure 2.2.

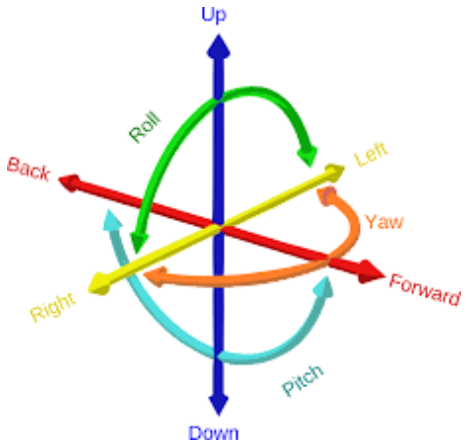

Figure 2.2: Translation And Rotation

2.6.1 Translation (Position)

Translation describes the object displacement along three axes originating from a defined coordinate center as seen below:

- **X-axis** Forward and backward movement.
- **Y-axis** Lateral, side to side movement.
- **Z-axis** Vertical, up and down movement.

2.6.2 Rotation (Orientation)

Rotation describes the object angular orientation around Translation axes, commonly represented as Euler angles as seen below:

- **Roll:** Tilting side to side around the X-axis.
- **Pitch:** Tilting forward and backward around the Y-axis.
- **Yaw:** Turning left or right around the vertical Z-axis (the steering direction).

2.6.3 Rotation Angle Calculations in QTM

Optical motion capture systems, such as QTM, do not inherently track 6DoF from a single point. A single reflective marker only provides 3D positional data. To calculate the rotational angles, the system requires a rigid body.

A rigid body is defined by placing at least three reflective markers on a physical object. Because the physical distance between these markers remains constant, the tracking software can use the relative 3D positions of the individual markers to mathematically compute the exact orientation and center point of the entire object shown in Figure 2.3.

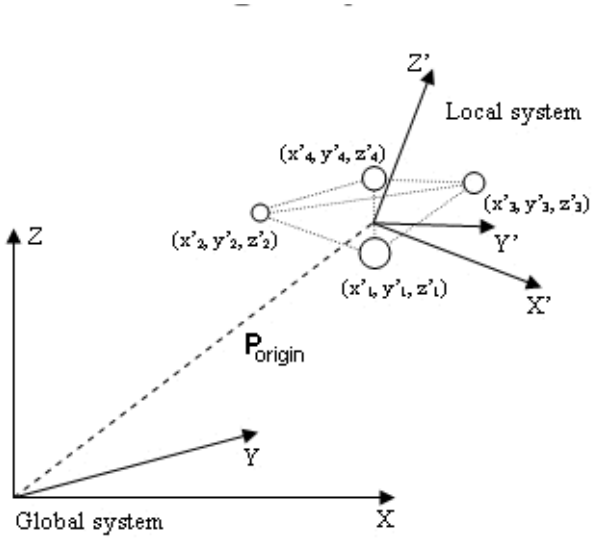

Figure 2.3: 6DoF Calculation

Rigid body definition to compute origin, the positional vector of the origin of the local coordinate system in the global coordinate system, and R, the rotation matrix which describes the rotation of the rigid body.

The rotation matrix (R) can then be used to transform a position P_{local} (e.g. x'_1, y'_1, z'_1) in the local coordinate system , which is translated and rotated, to a position P_{global} (e.g. x_1, y_1, z_1) in the global coordinate system. The following equation is used to transform a position 2.4

$$P_{global} = R \cdot P_{local} + P_{origin} \tag{2.4}$$

2.7 Python Libraries

Python is a high-level programming language [14] that is widely used due to its extensive library support and readability. Serial communication, communication between QTM and the control system, and concurrent programming, can be implemented in Python using different libraries.

Serial communication in Python can be implemented by the pySerial library, which is a third party library that provides an interface to a physical serial port [15]. Using pySerial, the serial connection can be easily configured with a few lines of code, where the communication parameters, baud rate,data bits and the stop bits are specified directly. This abstraction makes it possible to avoid direct interaction with lower-level operating systems to establish serial communication, decreasing the complexity of the implementation. In addition to configuring the hardware control signals RTS and DTR independently of the data channel.

Another important library is the asycnio library in Python, used for concurrent code [16]. This enables a program to run concurrently using async/await syntax within a single thread avoiding the complexity of using multi-threading. For instance, it can be used to let a program receive position data from the motion capture system and send commands simultaneously to the treadmill control system.

To obtain position data in real-time, the Qualisys real-time SDK for Python can be used. Which implements the QTM RT protocol, the library is called `qtm_rt`, it establishes a network connection to QTM, enabling a Python program to receive a continuous stream of data.

3
Methods

3.1 System Overview

The developed system is set up for Rodby RL3500Ex2500 treadmill year model 2021, found in the physiology facility in Chalmers university. The system designed to regulate the speed of a motorized treadmill based on real time position of an e-scooter. To achieve a smooth integration between human driven vehicle dynamics and machine actuation, the system is divided into three primary subsystems: Position Tracking (Input), Control Logic (Processing), and Actuation (Output).

3.1.1 Position Tracking (Input)

The origin of the coordinate system, X=0, Y=0, is calibrated at the center of the treadmill using Qualisys calibration tools. The coordinates of the e-scooter are then captured using Qualisys motion capture system. The cameras tracks markers that are manually selected and mounted on the e-scooter representing a rigid body. QTM streams simultaneously the 6 Degrees of Freedom (6DoF) data in real-time.

3.1.2 Control Logic (Processing)

Input from QTM is sent over the network and with the use of Qualisys real-time SDK writen in Python. Continuously processing and calculating the positional error along the X-axis in real-time. The error is then evaluated by a PI controller with an Anti-Windup to control the speed of the belt.

3.1.3 Actuation And Communication (Output)

The control logic sets the velocity at which the belt will be driven. Once the intended velocity has been established, the command can then be formatted for communication to the motor controller via the Modbus RTU protocol. The floating-point representation of the desired velocity is first converted into an integer, and then the integer is broken down into two bytes, one high and one low, using bit-shifting operations. Newly constructed byte sequences will include an appended CRC-16 to ensure the integrity of the data being sent and received. If the treadmill receives a message containing a CRC that was not correct, it will disregard the command and ask the control logic to send another.

3.2 Qualisys Track Manager Setup

After configuring the origin point ($X = 0, Y = 0$) in the center of the treadmill belt using the QTM calibration tools, a cluster of reflective markers is attached to the e-scooter. The system utilizes an array of 12 infrared cameras positioned around the laboratory to continuously track these markers. In QTM this cluster is defined as a rigid body, allowing the software to calculate the vehicles 6DoF. This spatial data is then streamed in real-time over the network to the Python environment using the Qualisys SDK shown in Figure A.5.

3.3 Main And The Control-Loop

The software architecture driving the closed-loop system is built upon a continuous, asynchronous execution loop in Python. To ensure stability and rider safety, the script is divided into an initialization phase (Safe Start) and a 10 Hz real-time control loop.

Before the treadmills motor is engaged, the software must establish and verify all external connections. During the initialization phase, the script opens the serial port connection to the treadmill. By configuring the communication parameters, to a baud rate of 57600 bps, 8 data bits, no parity, and one stop bit (8N1). In addition to the two RS-232 control signals, DTR and RTS which are disabled as shown Figure A.2. Subsequently, the connection to QTM is established and then the software subscribes to the 6DoF data stream from QTM.

3.3.1 Safety Boundary

A kill-switch is integrated directly into the main execution loop. The algorithm continuously monitors the lateral position of the vehicle (Y-axis). If the scooter drifts beyond the predefined physical safety boundaries (Y_{max} or Y_{min}), the software instantly terminates the loop and transmits a false command to system_running, forcing an immediate emergency stop to protect the rider. The implementation of the emergency stop is shown in Figure 3.1.

```
# Emergency Stop (Y-Axes)
if current_y_position > Y_MAX or current_y_position < Y_MIN:
    print(f"\n[!!!] NÖDSTOPP: Y = {current_y_position:.1f}")
    system_running = False
    await asyncio.to_thread(stop, conn)
    break
```

Figure 3.1: Emergency Stop (Y-Axis)

3.4 Position Regulation

Speed regulation algorithm was designed to keep the treadmill belt velocity in constant alignment with the constantly changing longitudinal positions of an e-scooter. This is achieved through continuous measurement of an e-scooter's position, which allows the rider to remain stabilized at a calibrated origin point ($X = 0$). A smooth and responsive motor command can now be sent to the treadmill using a PI loop.

3.4.1 Treadmill Step Response

To understand the dynamic behavior of the treadmill, four step response tests were conducted. By commanding a sudden change in belt speed and recording the actual speed until reaching steady-state speed.

Each test was conducted in three phases using a Python program that interfaced with the treadmill via the Modbus RTU protocol. In the first phase, the belt was set to a start speed of 2 km/h and stabilized for five seconds, to ensure that the belt reaches steady state. In the second phase, time t=0 was defined using the function `time.time()` and immediately after, the target speed was commanded by writing in the treadmill speed register, then the actual belt speed was sampled with 10 Hz during a 15 second interval. In the third phase, the belt was stopped and the step response data obtained.

Using the recorded speed data, the rise time, settle time, overshot, and steady-state error were obtained. Rise time was defined as the time for the speed to increase from 10% to 90% of the commanded step size. The settle time was defined as the time until the belt speed remained within 5% above and below the target speed. The overshot was defined as the maximum speed above the target speed, expressed as a percentage of the step size.

3.4.2 Communication Time

To examine how long it takes to receive a response from the treadmill, the actual time for the read operation was measured as shown in Figure 3.2.

```
# Read actual speed from treadmill
before = time.time()
actual = get_actual_speed(conn)
after = (time.time() - before)/2
print(after)
```

Figure 3.2: Time of the read operation

3.4.3 Mathematical Formulation of the Discrete PI-Controller

Since the control software executes at a fixed sampling frequency of 10 Hz ($\Delta t = 0.1$ s), a discrete-time representation of the PI controller is implemented. The system continuously evaluates the positional error e_k at the current time step k, defined as the difference between the target reference position X_{target} and the actual measured position from the motion capture system X_k shown in equation 3.1:

$$e_k = X_{\text{target}} - X_k \tag{3.1}$$

The control signal u_k, representing the commanded treadmill belt velocity, is the summation of three independent control components shown in equation 3.2:

$$u_k = P_k + I_k \tag{3.2}$$

Where each control term is mathematically defined as follows:

Proportional Term (P_k): Compenses for the immediate current error. It scales the control action linearly based on the distance from the target center point shown in equation 3.3:

$$P_k = K_P \cdot e_k \tag{3.3}$$

Integral Term (I_k): Accumulates past errors over time to eliminate steady-state tracking errors and maintain the baseline cruising velocity of the vehicle shown in equation 3.4:

$$I_k = I_{k-1} + K_I \cdot e_k \cdot \Delta t \tag{3.4}$$

3.4.4 Anti-Windup

To prevent integral windup a phenomenon where the integral term I_k grows indefinitely when the treadmills motor reaches its maximum physical speed limit (V_{max}) a conditional integration (clamping) algorithm was implemented. Before applying the integral accumulation in Equation 3.4, the algorithm evaluates if the control signal u_k is already saturated and if the error e_k is pushing in the same direction. If these conditions are met, the integration is temporarily paused. This anti-windup mechanism ensures the controller remains highly responsive and prevents dangerous overshoot when the vehicle begins returning toward the setpoint.

The discrete-time PI controller logic is represented in the block diagram in Figure 3.3. The control loop begins by computing the positional error e_k between the static spatial origin X_{target} and the real-time vehicle position X_k acquired from the QTM system. This error is simultaneously fed into a proportional gain block K_p and a discrete integral block. The integral term is represented by its z-domain equivalent function $K_i \frac{\Delta t}{z-1}$, effectively mirroring the backward Euler approximation executed in the Python environment. The preliminary control signal u_{prelim} is subjected to a saturation block, enforcing the mechanical and safety constraints of the treadmill $[V_{min}, V_{max}]$ before the final command u_k is formatted for Modbus transmission. Crucially, the diagram illustrates the logic of the anti-windup clamping mechanism

via the dashed feedback loop. If the preliminary velocity exceeds the physical saturation limits while the error maintains its sign, the clamping switch is triggered, actively halting further integral accumulation and preventing dangerous windup behavior.

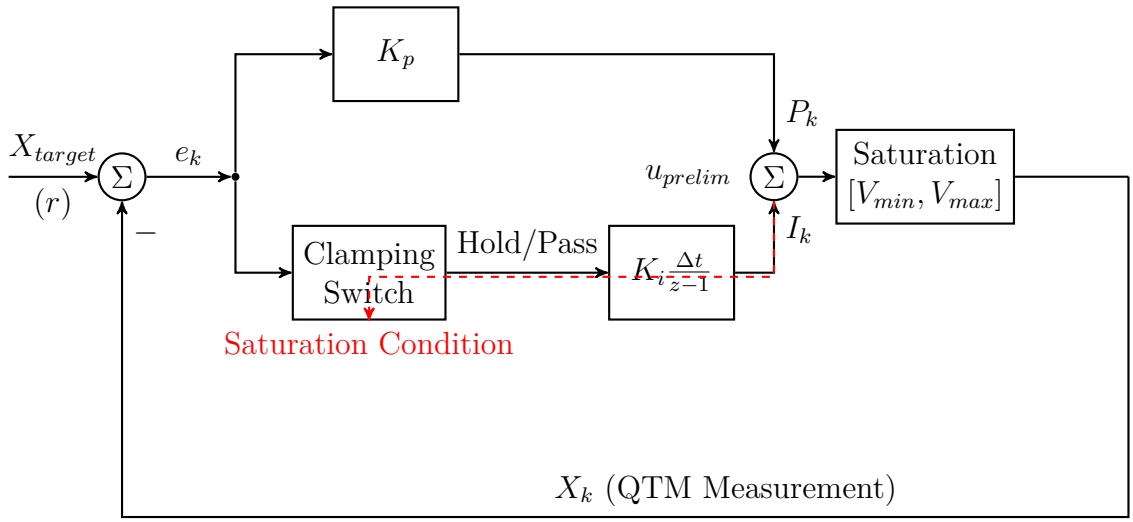

Figure 3.3: Block diagram of the discrete-time PI controller

3.5 Communication Protocol

Once the control logic calculates the desired speed for the treadmill, that digital value must be translated into a physical hardware command. Because the manufacturer provides no official API or SDK for third-party developers, a custom two-way communication bridge was developed to bypass the standard user display and interact directly with the treadmills motherboard.

To achieve this, the system utilizes the industry standard Modbus RTU protocol. The communication is structured around a Master-Slave Architecture. In this configuration, the Python control environment operates as the active Master, while the treadmills frequency inverter acts as the passive Slave.

3.5.1 Data Formatting and Byte Shifting

The target velocity calculated by the Python script is initially a floating-point number. To fit the Modbus protocol requirements, this value is first multiplied by a scaling factor of a 100 and converted into an integer. Because standard Modbus registers are 16 bits wide, this integer is then split into two 8-bit bytes (a High byte and a Low byte) using bit-shifting operations. These bytes are packaged into a specific data frame alongside the treadmills designated Slave address and the write command function code.

Since Rodby RL3500Ex2500 receives data with Modbus RTU from serial communication RS-232 on port COM3 the CRC16 make sure to send data at 8 bits and translate it to Modbus RTU shown in Figure 3.4.

```python
#Modbus Crc16
def crc16_modbus(data):
    crc = 0xFFFF
    for byte in data:
        crc ^= byte
        for i in range(8):
            if crc & 0x0001:

                crc = (crc >> 1) ^ 0xA001
            else:
                crc >>= 1

    return crc
```

Figure 3.4: CRC16 Modbus

3.5.2 Function Code & Data Frame

To communicate with the treadmills embedded motor controller, every transmission must strictly use Modbus RTU byte-frame architecture. The specific payload structure varies depending on the complexity of the command being executed. The different commands are shown in table 3.1

Table 3.1: Byte structural breakdown of the Modbus RTU data frame utilized

Action	Function Code	Data Frame (Byte Array)
Start	`0x05` (Write Coil)	1, 5, 0x08, 0x00, 0xFF, 0x00
Stop	`0x05` (Write Coil)	1, 5, 0x08, 0x00, 0x00, 0x00
Set Speed	`0x10` (Write Multi)	1, 16, 0x1B, 0xC2, 0x00, 0x02,0x04, hi, lo, 0x00, 0x00

Basic operational commands such as starting and stopping the motor utilize Function Code `0x05` (Write Single Coil) shown in Table 3.2. This concise data frame directly toggles the internal relays of the frequency inverter, providing a simple binary state change (ON/OFF) without requiring an large amount of data.

Table 3.2: Write Coil

Array	Write Coil	Description
0	Slave	Identify the receiving node on the port
1	Function code	`0x05` for Write Single Coil
2	Coil Address	The high byte of the internal relays specific memory address
3	Coil Address	The low byte of the internal relays specific memory address.
4	Data Value	0xFF for ON, `0x00` for OFF
5	Data Value	Not needed, `0x00` for the satisfaction of array

The dynamic speed regulation requires a more comprehensive data structure, outlined in Table 3.3. By utilizing Function Code `0x10`, Python can transmit the calculated 16-bit target velocity to the memory addresses.

Table 3.3: Write Multiple Registers

Array	Multiple Registers	Description
0	Slave	Identify the receiving node on the port
1	Function code	`0x10` for Write Multiple Registers
2	Register Address `hi`	The specific 16-bit memory address for velocity control
3	Register Address `lo`	Low byte of the memory address that controls velocity
4	Quantity of Registers `hi`	Total number of registers to be overwritten
5	Quantity of Registers `lo`	Total number of registers to be overwritten `0x02`
6	Byte Count	Specifies that exactly 4 bytes of data payload will follow
7	Data for Register `hi`	The calculated target speed
8	Data for Register `lo`	The calculated target speed
9	Data for Register `hi`	Not needed, `0x00` to satisfy the length of array
10	Data for Register `lo`	Not needed, `0x00` to satisfy the length of array

Figure 3.5 illustrates the practical construction of this velocity command as a decimal byte array. The upper sequence shows the raw data frame, while the lower sequence demonstrates the final payload just before transmission.

```
def set_speed(conn, speed):
    raw = int(round(speed * 100))
    hi = (raw >> 8) & 0xFF
    lo = raw & 0xFF
    reply = send(conn, [1, 16, 0x1B, 0xC2, 0x00, 0x02, 0x04, hi, lo, 0x00, 0x00])
    return len(reply) >= 6
```

Figure 3.5: Set speed

3.6 Hardware Communication

To send the calculated speed to the treadmill, the Python script uses the Modbus RTU protocol over a serial connection (RS232 at 57600 baud rate). Since the Modbus standard requires data to be sent as integers, the script converts the calculated speed for example, 5.5 km/h into a whole number (550) before splitting it into bytes, shown in Figure A.3.

Finally, to guarantee that the treadmill never acts on corrupted signals
the script calculates a CRC16 checksum. This checksum is attached to the end of every message sent to the motor controller.

4

Results

In this section the results of this thesis project is presented. Including the different tests conducted during the project and the resulting prototype of the implemented control system. The tests were carried out using a radio-controlled car and an e-scooter. The objectives of the project were tested with Radio-controlled car first for safety of the rider, after safely operating the radio-controlled car the e-scooter was used for testing the system. The resulting prototype could safely regulate the speed of the treadmill based on the vehicle position data.

4.1 Results of the Step Response

Four step response tests were conducted, starting from an initial speed of 2 km/h to a target speeds of 4, 8, 10 and 15 km/h. The results are presented in table 4.1.

Table 4.1: Step response results

Test	Start (km/h)	Target (km/h)	Step size (km/h)	Rise time (s)	Settle time (s)	overshoot (%)	steady-state error (km/h)
1	2.0	4.0	2.0	1.04	2.08	3.5%	0.06
2	2.0	8.0	6.0	3.11	4.15	1.3%	0.08
3	2.0	10.0	8.0	4.16	5.20	1.4%	0.08
4	2.0	15.0	13.0	6.21	>15	1.3%	0.09

Dividing the step size by the rise time for each test yields:

Test 1: $\frac{2.0}{1.04} = 1.92$ km/h/s

Test 2: $\frac{6.0}{3.11} = 1.93$ km/h/s

Test 3: $\frac{8.0}{4.4} = 1.92$ km/h/s

Test 4: $\frac{13.0}{6.21} = 2.09$ km/h/s

Mean acceleration= 1.97 km/h/s = 0.55 m/s^2
This indicates that the treadmill´s internal control system limits acceleration to a fixed rate regardless of the commanded speed change.

4.1.1 Communication Time

The measured time was approximately 0.502 seconds using the code shown in Figure 3.2. This results in each call taking 1 second, which leads to the effective sampling rate of the step response program to 1 Hz even though the program was configured with a sampling rate 10 Hz.

The step time was measured from the moment the step command was issued, rather than the moment the treadmill began responding, all the measured times include this communication latency. However, the rise time is computed as the difference between two timestamps, and the delay cancels out and does not affect the rise time measurement.

4.2 Communication and Actuation

The Modbus RTU serial bridge was successfully implemented. The system consistently converted the digital floating point speeds into physical motor actuation. The bridge successfully opened communication with our third party system, as long as the port was not occupied by other software such as Rodby. The bridge successfully started, stopped the treadmill, wrote speed and read speed but failed to elevate the treadmill.

4.3 Tracking Reliability

Incorporation of the Qualisys Python SDK resulted in tracking the rigid body of both the radio-controlled car and the e-scooter. There was an uninterrupted flow of data for rigid body measurements, and since NaNs were handled correctly, there was no computation of wrong errors during the brief periods when the markers were not visible. The markers used to define the rigid body of both the radio-controlled car and the e-scooter are shown in Figure 4.1 and 4.2.

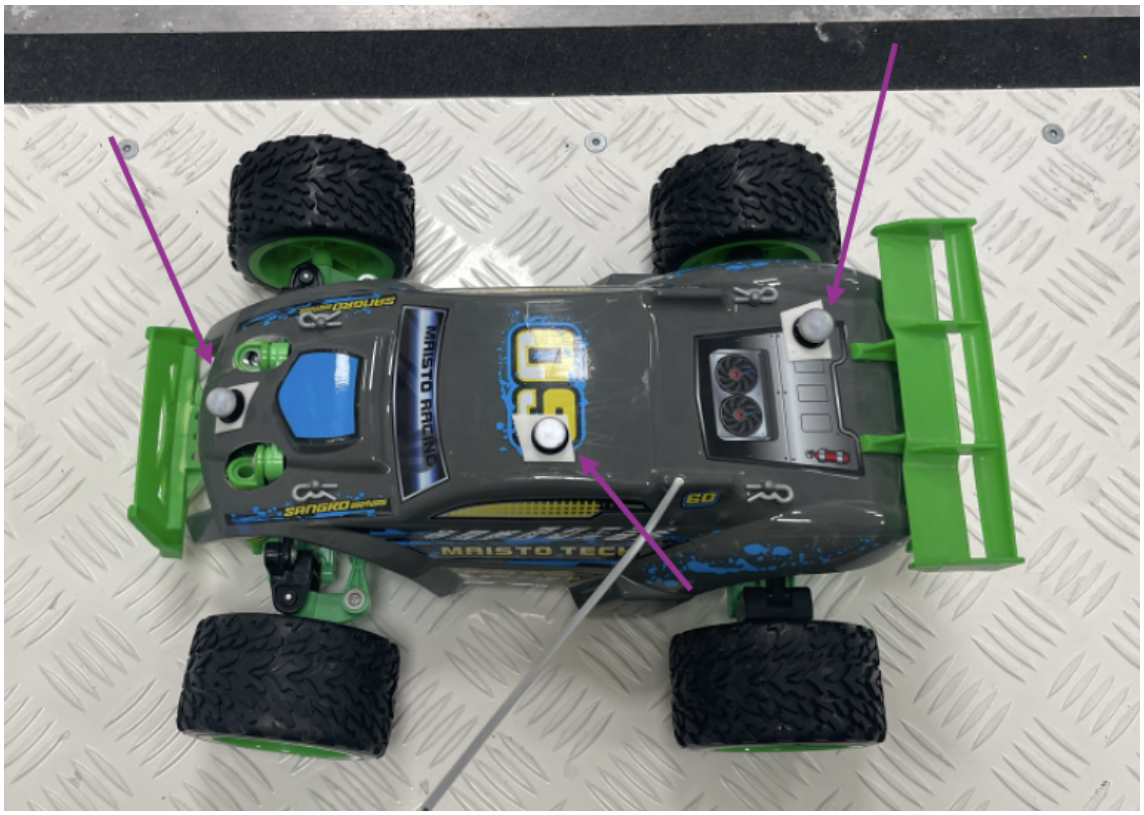

Figure 4.1: QTM-markers for radio controlled car

Figure 4.2: QTM-markers for E-scooter

4.4 Control System Performance

The implemented PI controller did increase the speed of the treadmill to keep the e-scooter near the central point ($X = 0$) when the scooter went past the center point, and decrease the speed of the treadmill when the e-scooter drifted behind the center point. Since the acceleration of the treadmill is limited to $0.55m/s^2$ it was not fast enough to keep the e-scooter centered when it accelerated, allowing the e-scooter to move past the central point. This is illustrated in the graph shown in

Figure 4.3.

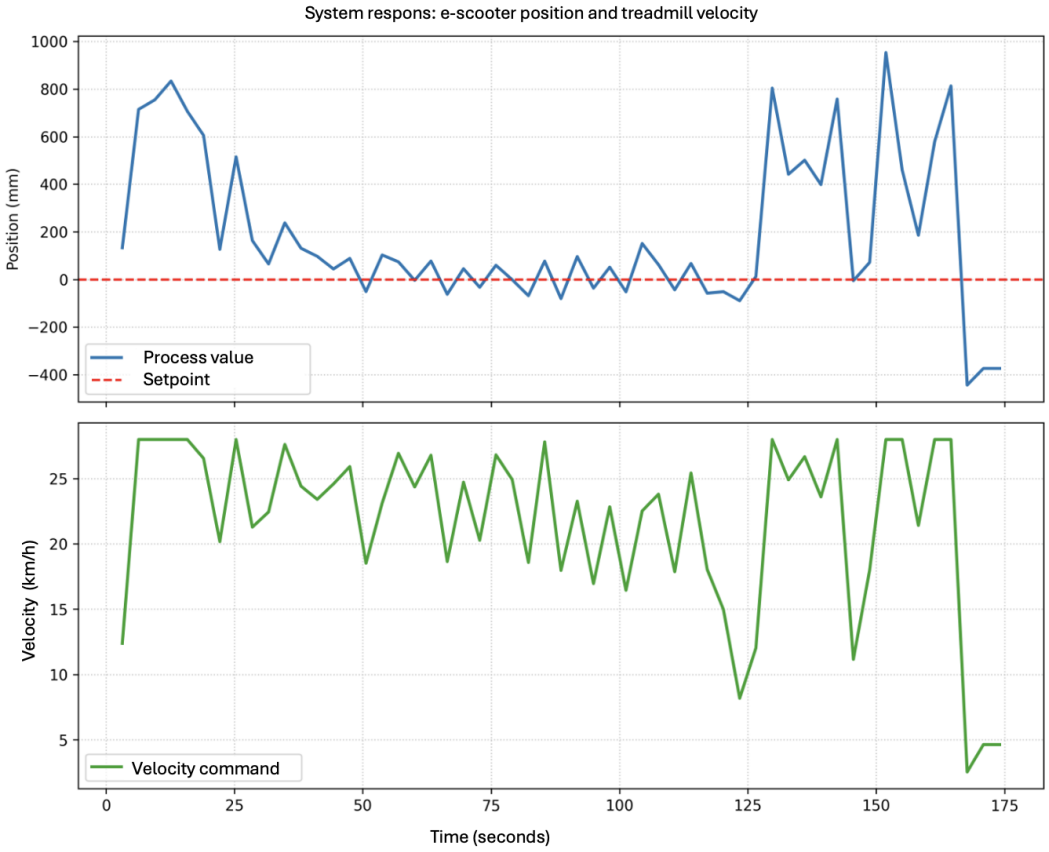

Figure 4.3: E-scooter position and belt speed over time

The position graph shows how at the start, when the e-scooter accelerates fast, it´s able to move past the middle point even though the treadmill speeds up to bring it back to the middle point. When the e-scooter stops accelerating as fast, it´s brought back to the middle point by the treadmill as can be observed in the middle of the graph. The last part of the position graph shows again how the treadmill fails to keep up with the e-scooter when it accelerates aggressively.

4.5 Safety System Validation

The software based safety boundaries shown in Figure 3.1 functioned as intended. During simulated tests, intentionally driving the radio-controlled car beyond the defined Y boundary limits instantly triggered the software kill-switch, the radio controlled car flew of the treadmill with a high velocity. The e-scooter was not tested for safety validation of the rider.

5
Conclusion

The project has proven that it is possible to build an automated Closed-Loop test system for 2-wheeled vehicles from a skate treadmill. The projects success was made possible by creating a custom Modbus RTU communication bridge to bypass the restrictions of the proprietary hardware and directly connecting the mocap system to the treadmill for velocity control. In addition, by integrating QTM into the system as a tracking tool, we were able to obtain very reliable real-time low-latency spatial information which allowed us to establish a well-defined control architecture.

The discrete time PI controller developed for this project with an integrated clamping anti-windup was capable of controlling the treadmill belt speed in order to stabilize the e-scooter under both steady-state and gradual speed transition conditions.

It is also worth noting that the response of the physical hardware used for the treadmill were asymmetric in nature. The e-scooter has a very high capability to accelerate instantaneously. However, the treadmill belt requires a significant amount of time to achieve higher speed setpoints because of the high amount of mechanical resistance present in the motor and the band. This means that during periods of aggressive rider acceleration, the scooter will be able to travel a significant distance from its origin before the treadmill belt reaches the target speed setpoint. Conversely, the treadmill has a very fast response time when decelerating. The asymmetric nature of the hardware will limit the tuning of the control system and therefore will continue to inhibit the ability of the PI controller to accurately track the acceleration profiles produced by human riders when they accelerate sharply.

5.1 Future Work

In conclusion, this thesis has demonstrated that commercially available hardware can be modified to support high-end automated testing. With this as an initial platform, future improvements to both hardware and software will be necessary to enable aggressive real-world riding conditions.
To support future development, future motor upgrades should include a higher frequency inverter to support faster control loops and faster acceleration. On the software side, predictive feed forward control algorithms should be used to better anticipate the e-scooters motion before actual motion occurs, thereby eliminating some of the delay time in starting the treadmill. Finally, a separate Graphics User

Interface (GUI) should be developed to replace the bypassed HMI so laboratory operators in the future can independently adjust control settings, monitor the spatial boundaries, and view the real-time computer tracking data without having to work through the Python terminal.

Bibliography

[1] TRANSPROT STYRELSEN, "Antal skadade i olyckor med elsparkcyckl ökar kraftig,"[Online]. Available: https://www.transportstyrelsen.se/sv/om-oss/pressrum/nyhetsarkiv/2025/antal-skadade-i-olyckor-med-elsparkcykel-okar-kraftigt/ (Accessed Apr. 19, 2026)

[2] forskning.se, "Förarnas beteende orsakar olyckor med elsparckcyklar" 2025.[Online]. Available: https://forskning.se/2025/03/26/orsaker-olyckor-elsparkcykel/ (Accessed May. 15, 2026)

[3] S.Wallhagen "Electric scooters: A review of international literature and analysis of Swedish accident data" Statens väg-och transportsforskningsinstitut, Linköping, Sweden, VTI PM 2023:16, 2023. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1822411/FULLTEXT01.pdf (Accessed May. 15, 2026)

[4] E.Zhou,"List of Serial Communication Protocols: A Complete Guide", COMESTAR, [Online]. Sep. 20, 2025. Available: https://www.come-star.com/blog/serial-communication-protocols/(Accessed Apr. 20, 2026)

[5] A.Pandit "Serial Communication Protocols", CIRCUIT DIGEST,[Online].Apr. 29, 20219. Available: https://circuitdigest.com/tutorial/serial-communication-protocols#rs232-serial-communication-implementation (Accessed Apr. 20, 2026)

[6] ANALOG DEVICES,"Fundamentals of RS-232 Serial Communications," 2001. [Online]. Available: https://www.analog.com/en/resources/technical-articles/fundamentals-of-rs232-serial-communications.html (Accessed Apr. 21, 2026)

[7] Chromateq,"RS232 PROTOCOL,"[Online]. Available: http://filedownload.chromateq.com/Tutorials/RS232%20protocol.pdf(Accessed Apr. 21, 2026)

[8] Y. Yuanyuan, C. Meng,"The design of adaptive communication frame supporting high-speed transmission based on ModBus protocol," in 10th International Confrance of Information and Communication Technology, 2022, pp.2-3. [Online]. Available:https://www.sciencedirect.com/science/article/pii/S187705092100572X (Accessed Apr. 21, 2026)

[9] modbus tools, "Modbus protocol Description", [Online]. Available: https://www.modbustools.com/modbus.html#function16 (Accessed Apr. 21, 2026)
[10] B. Thomas,Modern Reglerteknik. 5th ed., Stockholm, Sweden: Libre, 20216.
[11] K. J. Åström and T. Hägglund, Advanced PID Control. Research Triangle Park, NC, USA: ISA - International Society of Automation, 2006. [Online]. Available: https://app.knovel.com/hotlink/toc/id:kpAPIDC001/advanced-pid-control.
[12] Adobe, "What is motion capture and how dose it work?", [Online]. Available:https://www.adobe.com/uk/creativecloud/animation/discover/motion-capture.html#what-is-motion-capture (Accessed June. 13, 2026)
[13] Qualisys, "Qualisys motion capture systems" [Online]. Available:https://docs.qualisys.com/qtm/content/front_matter/about_qtm_manual.htm (Accessed Apr. 21, 2026)
[14] Python,"General Paython FAQ", [Online].Available:https://docs.python.org/3/faq/general.html#what-is-python-good-for (Accessed June. 13, 2026)
[15] pySerial," Welcome to PySerial´s documentation",[Online]. Available:https://pyserial.readthedocs.io/en/latest/#(Accessed Apr. 21, 2026)
[16] Python Software Foundation," asyncio-Asynchronus I/O",[Online]. Available: https://docs.python.org/3/library/asyncio.html(Accessed Apr. 21, 2026)

A

Appendix 1

The code that was used during this thesis for the control loop is shown below.

```
# ========================================
# 1. Settings
# ========================================

QTM_IP = "129.16.73.---"
PORT = "COM3"
BAUD = 57600
X_TARGET = 0.0   # Central point

KP = 0.015 # Proportionell Gains
KI = 0.010 # Integrerande Gains


DYNAMIC_THRESHOLD = 100 # mm from X_target to activate dynamic gains

# Kill-Switch
Y_MAX = 1500.0
Y_MIN = -1350.0

#Max Velocity
V_MIN = 0.0
V_MAX = 25.0

# Filter
FILTER_SIZE = 2

#Global Variabals
x_history = []
filtered_x = None
current_y_position = None
system_running = True
current_belt_speed = V_MIN
```

Figure A.1: Settings

```
# ========================================
# 2. Treadmil Communication
# ========================================

#To open the bridge for the software to the hardware (To the treadmills motherboard)
def open_connection():

    conn = serial.Serial(
        PORT, BAUD, bytesize=serial.EIGHTBITS, parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE, timeout=3, rtscts=False, dsrdtr=False
    )
    conn.setRTS(False)
    conn.setDTR(False)
    time.sleep(1)
    conn.reset_input_buffer()
    return conn

#Modbus Crc16
def crc16_modbus(data):
    crc = 0xFFFF
    for byte in data:
        crc ^= byte
        for i in range(8):
            if crc & 0x0001:

                crc = (crc >> 1) ^ 0xA001
            else:
                crc >>= 1

    return crc
```

Figure A.2: Open Connections and CRC16

```python
# send signal
def send(conn, data):
    crc = crc16_modbus(data)
    lo = crc & 0xFF
    hi = (crc >> 8) & 0xFF
    msg = bytes(data) + bytes([lo, hi])
    conn.reset_input_buffer()
    conn.write(msg)
    time.sleep(0.05)
    return conn.read(16)

#Converting integers to Modbus dataformat.
def set_speed(conn, speed):
    raw = int(round(speed * 100))
    hi = (raw >> 8) & 0xFF
    lo = raw & 0xFF
    reply = send(conn, [1, 16, 0x1B, 0xC2, 0x00, 0x02, 0x04, hi, lo, 0x00, 0x00])
    return len(reply) >= 6
#start signal
def start(conn):
    return send(conn, [1, 5, 0x08, 0x00, 0xFF, 0x00])

#Stop signal
def stop(conn):
    return send(conn, [1, 5, 0x08, 0x00, 0x00, 0x00])
```

Figure A.3: Send, Set speed, Start and Stop Functions

```python
# ==========================================
# 3. Soft-Stop "Ctrl + C"
# ==========================================

async def Mjukstop(conn, start_speed):
    print(f"\n[!] Mjukstopp från {start_speed:.1f} km/h...")
    speed_to_set = start_speed
    while speed_to_set > V_MIN:
        speed_to_set = max(V_MIN, speed_to_set - 1.0)
        print(f"    Saktar ner... {speed_to_set:.1f} km/h")
        await asyncio.to_thread(set_speed, conn, speed_to_set)
        await asyncio.sleep(0.5)

    print("[!] Stänger av motorn.")
    await asyncio.to_thread(stop, conn)
```

Figure A.4: Soft Stop

```python
# ========================================
# 4. QTM CALLBACK OCH FILTER (QTM SDK)
# ========================================
def on_packet(packet):
    global filtered_x, current_y_position, x_history
    try:
        header, bodies = packet.get_6d_euler()
        if bodies and len(bodies) > 0:
            body = bodies[1]   #Make sure its connected to the right Rigid-body
            position, _ = body
            raw_x = position.x if hasattr(position, 'x') else position[0]
            current_y_position = position.y if hasattr(position, 'y') else position[1]
            x_history.append(raw_x)

            if len(x_history) > FILTER_SIZE:
                x_history.pop(0)

            filtered_x = sum(x_history) / len(x_history)

    except Exception:
        pass
```

Figure A.5: QTM Packet

,

```python
# 5. PI-REGULATOR
# ========================================
async def control_loop(conn):
    global filtered_x, current_y_position, system_running, current_belt_speed
    last_time = time.time()
    integral = 0.0
    start(conn)
    try:
        while system_running:
            await asyncio.sleep(0.1)
            if filtered_x is None or current_y_position is None:
                continue
            # Emergency Stop (Y-Axes)
            if current_y_position > Y_MAX or current_y_position < Y_MIN:
                print(f"\n[!!!] NÖDSTOPP: Y = {current_y_position:.1f}")
                system_running = False
                await asyncio.to_thread(stop, conn)
                break
            now = time.time()
            dt = now - last_time
            last_time = now

            # 1. Error
            error = filtered_x - X_TARGET

            # 2. P-Gain (Standard proportionell)
            P = KP * error

            # 3. I-Gain & INTEGRAL WINDUP
            preliminary_integral = integral + (error * dt)
            preliminary_speed = P + (KI * preliminary_integral)

            # Windup:
            if (preliminary_speed >= V_MAX and error > 0) or (preliminary_speed <= V_MIN and error < 0):
                pass  # Pause integration, 'integral' not uppdating
            else:
                integral = preliminary_integral  # uppdating secured

            I = KI * integral
            # 4. Calculation of velocity (PI-Reglation)
            new_speed = P + I
            new_speed = max(V_MIN, min(V_MAX, new_speed))
            # 5. Constant update
            success = await asyncio.to_thread(set_speed, conn, new_speed)
            if success:
                current_belt_speed = new_speed

    except asyncio.CancelledError:
        pass
```

Figure A.6: Control Loop

```python
# ========================================
# 6. MAIN
# ========================================
async def main():
    global system_running
    print("Initierar systemet...")

    try:
        treadmill_conn = open_connection()
        print("Ansluten till löpband.")

    except Exception as e:
        print(f"Kunde inte ansluta till löpbandet: {e}")
        return
    connection = await qtm.connect(QTM_IP)

    if connection is None:

        print("Kunde inte ansluta till QTM.")
        treadmill_conn.close()

        return

    print("Ansluten till QTM. Startar dataström...")

    await connection.stream_frames(components=["6deuler"], on_packet=on_packet)

    control_task = asyncio.create_task(control_loop(treadmill_conn))

    try:
        print("\nSystemet är redo. Tryck Ctrl+C för mjukstopp.")
        while system_running:
            await asyncio.sleep(1)

    except KeyboardInterrupt:
        print("\n[!] Ctrl+C mottaget!")
        system_running = False
        control_task.cancel()
        await Mjukstop(treadmill_conn, current_belt_speed)
    finally:
        treadmill_conn.close()
        print("Löpband stoppat och frånkopplat.")

if __name__ == "__main__":
    asyncio.run(main())
```

Figure A.7: Main

DEPARTMENT MECHANICS AND MARITIME SCIENCE
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY