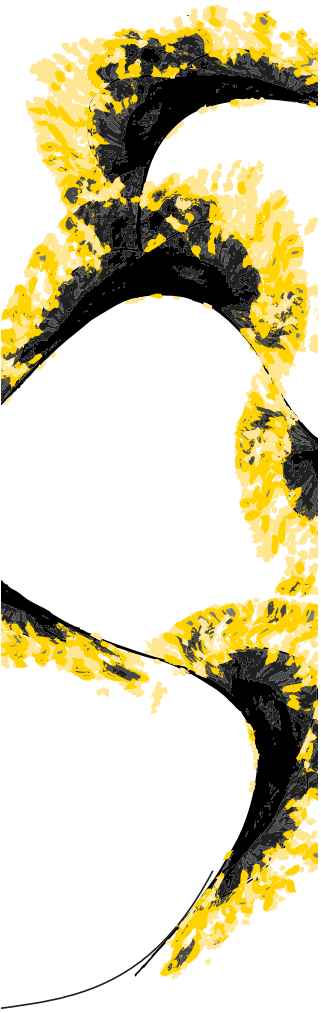


RAM ROBOTICS AND MECHATRONICS

SAFETY-LAYER DESIGN FOR THE RELBOT

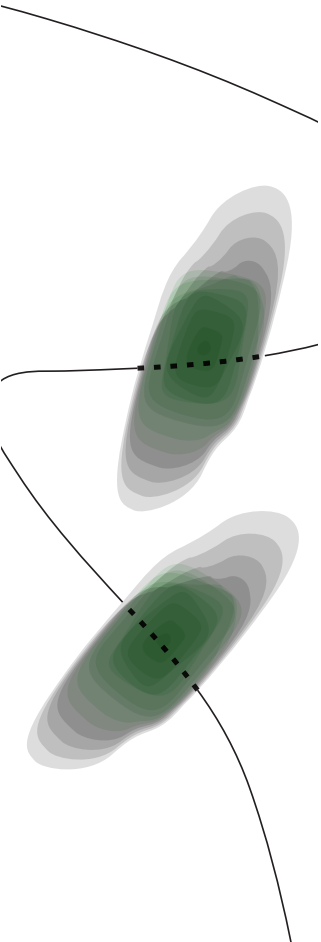
A.J.H. (Antonie) Veldwachter

ACADEMIC SKILLS PROJECT

Committee:
dr. ir. J.F. Broenink
dr. V. Groenhuis, MSc
dr. ir. G. van Oort

September, 2025

070RaM2025
Robotics and Mechatronics
EEMCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands



UNIVERSITY OF TWENTE. | TECHMED CENTRE UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Summary

This project aims to design and implement a safety layer for the RELbot. The goal of this safety layer is to prevent collisions and drop-offs during operation. Prior research was executed (Hofmann, 2024). The study recommends the use of multi-zone time-of-flight (ToF) sensors for the safety layer. These sensors have been integrated into the existing XRF2 framework.

A structured analysis identified stakeholder needs for a simple, predictable, and minimally intrusive safety layer. Technical requirements were identified, focusing on firm real-time detection, low CPU usage (<30%), and reliable intervention. A quadratic velocity-dependent threshold was selected for activation, and parameters were determined through brake tests. The chosen response behaviour is a direct motor lock, combined with a red LED indicator.

The final design split processing between ROS2 in Ubuntu (sensor readout, data analysis) and XRF2 in Xenomai (Threshold calculation, braking), minimising latency. Mechanical improvements included press-fitted roof-mounted sensor fixtures placed at an angle for optimal floor-edge detection. A functional calibration process ensured robustness to mounting variations and provided a reference view for its activation region. The safety feature focuses on objects in the straightforward path, ignoring objects it will drive past.

Testing demonstrated that object detection and collision avoidance succeeded in 96% of trials, meeting the ≥95% requirement. Drop-off detection was reliable up to 0.3 m/s, sufficient for typical elevated-surface use (<0.25 m/s). CPU usage averaged 12.2%, well within the specification.

Known limitations include reduced performance at higher speeds, misclassification of slopes, and occasional missed detection of angled drop-offs. Sensor noise was found to increase with distance and lighting conditions, reinforcing the importance of filtering and threshold margins for object and floor-edge detection. The main performance bottleneck was the I^2C communication speed, limiting the two-sensor setup frame-rate to 3.5 Hz. Recommendations for future work include increasing this speed, improving physical mounts, and making the system more modular by integrating the `ROS2_adapter`.

Overall, the safety feature meets its purpose of collision and drop-off avoidance within the defined requirements. It provides a robust and practical solution for protecting the RELbot in educational use.

Contents

1 Introduction **1**
1.1 Context . . . 1
1.2 Mission Statement . . . 1
1.3 Tasks . . . 1

2 Background **2**
2.1 The RELbot . . . 2
2.2 The ToF sensors . . . 5

3 Analysis **6**
3.1 Stakeholder wishes . . . 6
3.2 Activation of the Safety Feature . . . 6
3.3 Typical RELbot environments . . . 8
3.4 Project Boundaries . . . 8
3.5 Stakeholder Requirements . . . 8
3.6 Technical Requirements . . . 9

4 Design **10**
4.1 Safety Response . . . 10
4.2 Indication Method . . . 11
4.3 Threshold model . . . 11
4.4 Sensor Placement . . . 13
4.5 Calibration . . . 14
4.6 Data processing . . . 16
4.7 Integration within the XRF2 Framework . . . 18
4.8 Cyber-Physical System . . . 19
4.9 Hardware . . . 20
4.10 Functional Flow . . . 21

5 Tests **22**
5.1 Sensor-noise tests . . . 22
5.2 System Speed tests . . . 24
5.3 CPU usage . . . 25
5.4 The Dynamic Brake Tests . . . 25
5.5 Pass/Fail Summary . . . 27

6 Conclusions and Recommendations **28**
6.1 Conclusions . . . 28

6.2 Recommendations 29

A User Manual **30**
A.1 Installation 30
A.2 Calibration 32
A.3 Change I^2C Addresses 33
A.4 Reproducing the Demo 34

B Suggested Changes **35**
B.1 Revist threshold model 35
B.2 Integrating the `relbot_adapter` 35
B.3 Separate Floor Calibration 37

C Wheel Speed Relations **39**

D Sensor Mounting Angle Evaluation **40**
D.1 Setup 40
D.2 Method 40
D.3 Results 41

E Sensor Noise Tests **42**
E.1 Noise over Distance 42
E.2 Noise with External Interfaces 42

F Static Threshold Brake Test **45**
F.1 Setup 45
F.2 Method 45
F.3 Additional Data Collection 45
F.4 Results 45

G Dynamic Threshold Brake Tests **47**
G.1 Object Detection 47
G.2 Floor Detection 48

H CPU usage **50**

I Appendix: Survey **51**
I.1 Analysis 51
I.2 Raw results 52

J Matlab Threshold Estimation **56**
J.1 Result 56

References **58**

1 Introduction

1.1 Context

RELbots are educational platforms used by students, technicians, and teachers for various projects. Frequent use can lead to unintended behaviour and physical damage, such as collision with objects or falling from elevated surfaces like tables. To prevent the system from damaging itself, a collision avoidance feature was investigated, with a proposed method in Hofmann (2024). The study recommends integrating two multi-zone time-of-flight sensors into the RELbot system.

This project focuses on implementing that safety feature and evaluating its behaviour. The following mission statement was defined within this scope.

1.2 Mission Statement

- To develop and implement a safety feature for the RELbot using multi-zone time-of-flight sensors.

1.3 Tasks

To achieve the mission statement, the following tasks were completed within the scope of this project.

1. Analyse the desired system behavior.
2. Define system requirements for the safety feature.
3. Design a safety feature that uses the `VL53L7CX` multi-zone ToF sensor and integrates into the XRF2 framework
4. Evaluate the safety feature through testing.
5. Analyse the results and assess the effectiveness and limitations of the safety feature.
6. Reflect on system limitations and provide recommendations for future development.

2 Background

Before the project started, some important technologies had already been researched and/or developed. Applicable systems and studies for this study will be explained in this chapter.

2.1 The RELbot

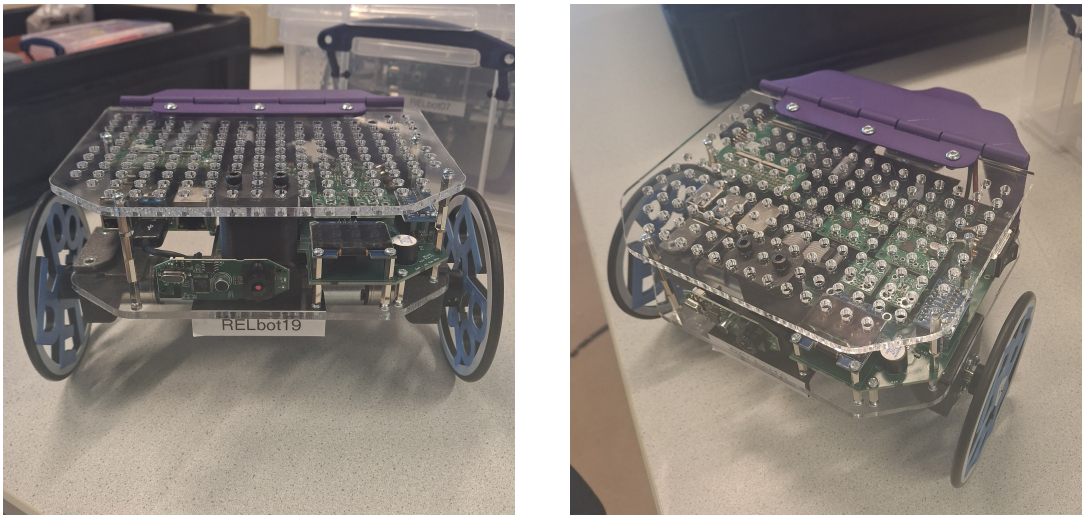
The RELbot is an educational robot, shown in Figure 2.1. It is a programmable cart with two actuated wheels and a castor wheel. It is designed to be modular and easy to add external interfaces to. This provides a platform where students can learn how to program and researchers can test their algorithms. The RELbot operates on a Raspberry Pi and an FPGA, allowing real-time software to run on it. The physical structure and the software infrastructure of the RELbot are presented below.

2.1.1 Physical System

The most recent RELbot version is shown in Figure 2.1. The RELbot has the following components available at the start of this project, which are categorized and listed below. How these components come together is schematically shown in Figure 2.2.

The RELbot components at the start of the project were:

1. Sensors
   (a) Raspberry Pi Camera Module
   (b) Encoders
   (c) Current Sensors
2. Controllers/Computers
   (a) Raspberry Pi 4
   (b) ico-board FPGA
   (c) Motor Drivers
   (d) Analog to Digital Converter (ADC)
3. Actuators
   (a) Wheel actuators 12 V
4. Indicators
   (a) I^2C Display
   (b) Buzzer



(a) Front View **(b)** Side View

Figure 2.1: RELbot physical system

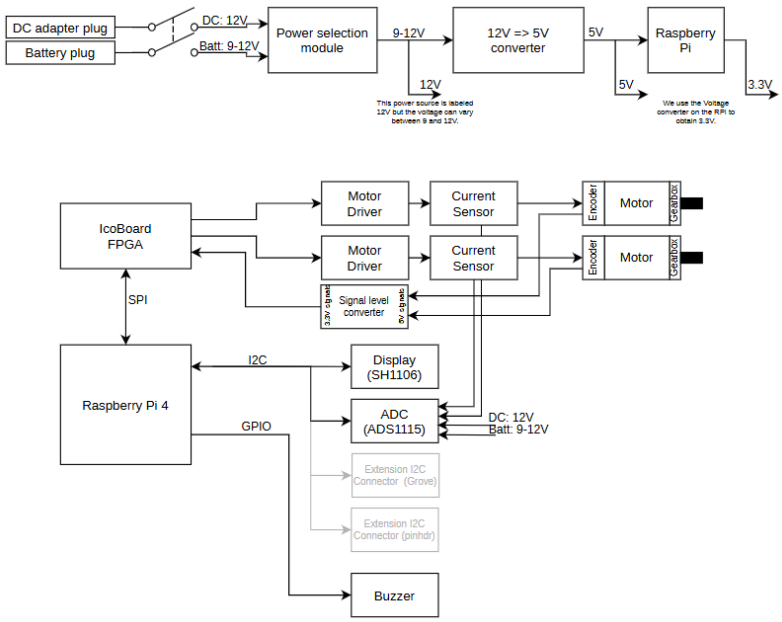


Figure 2.2: Schematic Overview (Oort, 2025)

2.1.2 The XRF2 framework

Before the introduction of the XRF2 framework, the Raspberry Pi ran a split software stack, as shown in Figure 2.3. On one side, a Linux (Ubuntu) environment hosts ROS2 in soft real-time. Alongside it, Xenomai provides a firm real-time environment. Xenomai communicated directly with the FPGA over the SPI bus, and the FPGA in turn interfaced with the actuators, encoders, and other low-level hardware.

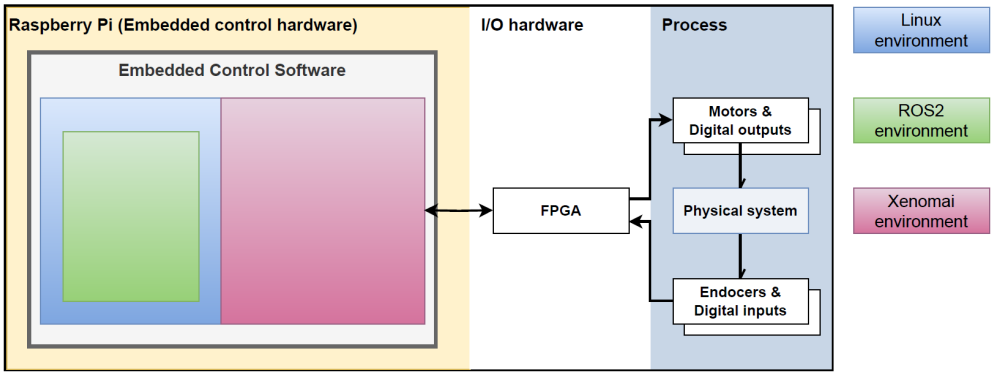


Figure 2.3: Software Layout (Raoudi, 2024)

The XRF2 framework, short for Xenomai–ROS2 Framework, builds on this base system to form the connection between the firm real-time domain in Xenomai and the soft real-time ROS2 environment. These conclusions originate from the study of Raoudi (2024). It consists of:

- The finite state machine (FSM) and controller loop running in firm real-time on the Xenomai core.
- The interface to the FPGA over the SPI bus (hard real-time link).
- The ROS2–Xenomai bridge, enabling communication between Xenomai and ROS2.

Only ROS2 components required for the ROS2–Xenomai interface are part of XRF2; all other ROS2 nodes are separate from the framework. The XRF2 framework is the firm real-time core of the system, bridging both to the soft real-time ROS2 layer and the hard real-time FPGA layer. This structure is shown in Figure 2.4.

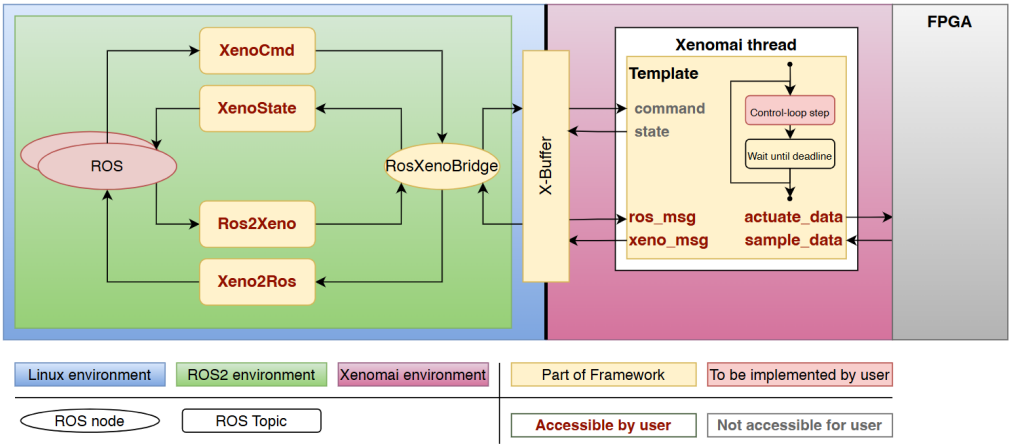


Figure 2.4: The XFR2 framework (Broenink, 2025)

2.2 The ToF sensors

The goal of creating a collision avoidance feature was previously researched (Hofmann, 2024). The focus of this project was to choose a detection method for a collision avoidance feature. The study concluded a pair of multi-zone ToF (time-of-flight) sensors. The sensor chosen was the `VL53L7CX` from ST Microelectronics. A testing algorithm has been developed for the RELbot in Python. Before this project, the ToF sensors were not yet integrated within the XRF2 framework.
Two sensors would create a large field of view that would be sufficient to detect objects within the path of the RELbot.
The result of the study of Hofmann (2024) was a hardware design including multi-zone ToF-sensors, A Python script to read out the sensors, and recommendations on how to integrate these sensors even further. The resulting physical RELbot system is shown in Figure 2.5.

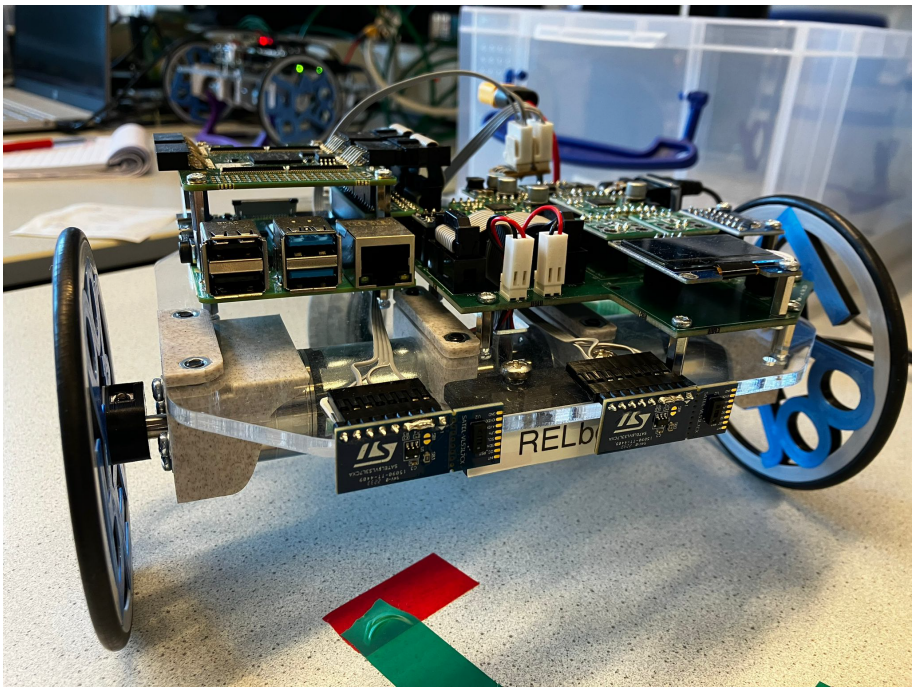


Figure 2.5: Integraded Multi-zone ToF sensors (Hofmann, 2024)

3 Analysis

Designing a safety feature for the RELbot requires a structured analysis to define its intended behaviour, activation conditions, and operational constraints. The outcome is a set of requirements that form the basis for the design and implementation.

The chapter is organised as follows:

1. Summarising the wishes of the stakeholders.
2. Defining the conditions for activating the safety feature.
3. Describing the typical environments of the RELbot.
4. Determining the scope of the project and selecting project boundaries.
5. Translating findings from the previous sections into stakeholder requirements.
6. Translating findings from previous sections into technical requirements.

Each section builds on the previous one, progressing from user expectations to operational analysis, and concluding with a set of requirements that guide the design phase.

3.1 Stakeholder wishes

Within this project, the stakeholders are students, technicians, teaching assistants (TAs), and teachers. They interact with the RELbot either through its use in various projects or during maintenance. To define the stakeholder wishes, field research was conducted and is presented in Appendix I. The results showed that most stakeholders were concerned about the system compensating too much. They wish for freedom, a predictable system, and minimal interference.

The system should be straightforward, avoiding complicated algorithms that significantly alter RELbot behaviour. The safety feature should indicate clearly and intuitively that it is active for predictability. In other words, stakeholders wish for an intuitive system that interferes as little as possible, but provides a clear indication when it does.

3.2 Activation of the Safety Feature

To determine when the safety feature should be activated, several key terms must be introduced. For a system to come to a full stop, it needs to brake, resulting in a braking distance d_{brake}, which depends on the system's velocity and inertia. In addition, there is a delay before braking begins due to measurement processing and control activation. This delay translates into a response distance d_{response}, obtained by multiplying the response time by the current velocity.

These two distances together form the stopping distance d_{stop}. A safety margin d_{margin} is also introduced to keep a fixed distance from an obstacle or drop-off. Figure 3.1 illustrates the relation between these distances.

The safety feature should be activated before d_{stop} exceeds the measured distance to an object or floor-edge. Adding the safety margin to the stopping distance yields the threshold distance $d_{\text{threshold}}$. When the measured distance becomes smaller than $d_{\text{threshold}}$, the safety feature should initiate an emergency brake. The threshold model depends on the system velocity and is derived from basic kinematics.

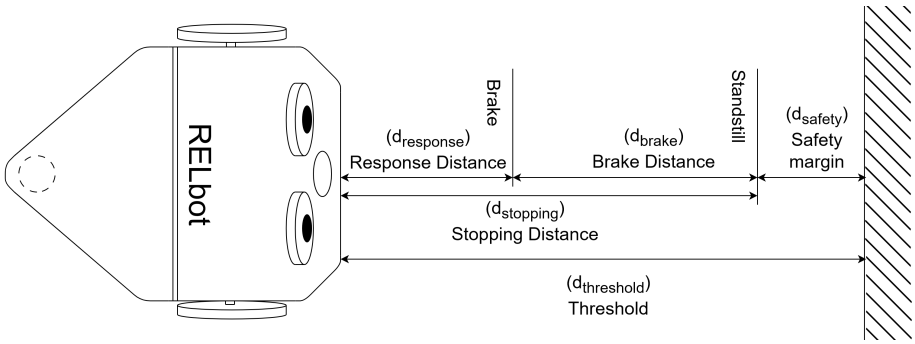


Figure 3.1: Stopping Distance Explained

The velocity of the system as a function of time is:

$$v_f(t) = v_i + at \tag{3.1}$$

Where v_i is the current speed, v_f is the future speed, and a is the constant deceleration. The displacement of the system as a function of time is:

$$d(t) = v_i t + \frac{1}{2}at^2 \tag{3.2}$$

Given these two equations, the distance to go from v_i to v_f is:

$$t = \frac{v_f - v_i}{a} \tag{3.3}$$

$$d = v_i * \left(\frac{v_f - v_i}{a}\right) + \frac{1}{2}a\left(\frac{v_f - v_i}{a}\right)^2 \tag{3.4}$$

$$d = \frac{v_i(v_f - v_i)}{a} + \frac{(v_f - v_i)^2}{2a} \tag{3.5}$$

$$d = \frac{2(v_i v_f - v_i^2) + (v_f^2 - 2v_i v_f + v_i^2)}{2a} = \frac{v_f^2 - v_i^2}{2a} \tag{3.6}$$

For a complete stop, $v_f = 0$, giving:

$$d_{\text{brake}} = \frac{-1}{2a}v^2 \qquad a < 0 \tag{3.7}$$

Assuming deceleration to prevent a negative brake distance.

The threshold model is then defined as:

$$d_{\text{threshold}} = c_1 v^2 + c_2 \qquad c_1 \geq \frac{-1}{2a};\ c_2 \geq d_{\text{Safety}} \tag{3.8}$$

where c_1 and c_2 are constants used to tune the system's sensitivity and offset.

The velocity v in (3.8) is the average wheel speed in m/s, calculated from the actuator speeds of the left and right wheels:

$$v = \frac{v_{\text{left}} + v_{\text{right}}}{2} \tag{3.9}$$

The model omits a linear term (response distance) because the quadratic term dominates braking distance. This simplification reduces complexity and avoids the need to tune an additional parameter.

3.3 Typical RELbot environments

RELbots are mostly used in educational environments, for example, in the courses *Software Development for Robotics* and *Advanced Software Development for Robotics*. In these courses, students develop programs to have the RELbot track a green ball. The advanced course extends this by incorporating real-time control principles.

These courses typically run in lab settings where RELbots operate on elevated surfaces. This environment introduces a significant risk of the robot unintentionally driving off the edge. Therefore, drop-off prevention is a concern for any safety system integrated into the RELbot.

In addition to manual or semi-automated tasks, RELbots are increasingly used in autonomous robotics projects, such as those in the *AI for Autonomous Robots* course. Although the specifics of these projects vary, the general goal is to develop software that enables the RELbot to autonomously navigate its environment.

In such autonomous contexts, the RELbot may traverse lab spaces containing objects, walls, and other robots. As a result, collision avoidance remains a critical requirement. A robust safety system must therefore address both forward collision risk and floor-edge detection.

3.4 Project Boundaries

Boundaries are necessary to outline the scope of this project and make sure it does not diverge from the main objective. The following project boundaries were set:

1. No significant changes will be made to the RELbot design besides the introduction of the safety feature.
2. The safety feature will use two multi-zone ToF sensors as per the recommendations of Hofmann (2024).
3. The safety feature will operate within the existing XRF2 framework, see Section 2.1.2.
4. No external processing units will be introduced.
5. The focus is on typical RELbot project speeds (<0.6 m/s on the floor and <0.25 m/s on elevated surfaces)

3.5 Stakeholder Requirements

The stakeholder requirements translate the wishes described in Section 3.1 into requirements for the safety feature. Stakeholders include students, teaching assistants (TAs), technicians, and teachers, all of whom use the RELbot in various projects or during maintenance.

The safety feature must be intuitive, interfere as little as possible, and behave predictably. It should be inactive under normal operation, taking control only in unsafe situations. When active, it must indicate this through a clear physical indication. The full system should not significantly reduce the RELbot's performance.

These expectations are formalised in Table 3.1, which lists the stakeholder requirements together with their identifiers, values, and MoSCoW priorities.

Nr.	Requirement	Value	MoSCoW
SR01.01	The user must be able to easily disable the safety feature by changing one boolean value	–	M
SR01.02	The total computational load of the system should not exceed 30% of the available CPU capacity.	<30%	S
SR01.03	When the safety feature is active, this must be indicated physically.	–	M
SR01.04	The safety feature should be compatible with existing RELbot use cases, allowing small adaptations	–	S
SR01.05	The user's control loop must be suspended while the safety feature is active.	–	M
SR01.06	The user's control loop should automatically resume once the safety condition has been resolved.	–	S

Table 3.1: Stakeholder Requirements

3.6 Technical Requirements

The technical requirements in this section translate the stakeholder expectations (Sections 3.5 and 3.1) and the analysis of RELbot operation (Sections 3.2–3.4) into requirements. While stakeholder requirements describe desired behaviour in general terms, technical requirements define measurable conditions to ensure the safety feature functions reliably and predictably.

The safety feature is expected to prevent collisions and drop-offs with minimal interference with the user's control. It must rely on defined distance thresholds (see Equation 3.8) and operate in a binary fashion: either the RELbot is in a safe state, or the emergency stop is triggered. The exact values of this threshold will be determined in Chapter 4.3. The safety margin is set at 10 mm given the observations of Hofmann (2024). The safety feature shall not interpret or predict the user's intended trajectory based on motion history or turning behaviour. This design choice is made to preserve system simplicity, predictability, and reliability.

With these restrictions, technical requirements have been defined in table 3.2

Nr.	Requirement	Value	MoSCoW
TR01.01	The safety feature must keep at least 10 mm distance from any floor-edge or object	> 10 mm	M
TR01.02	Objects and floor-edges outside of the forward path of the RELbot should be ignored.	-	S
TR01.03	The safety feature must only activate when objects or floor-edges are within the threshold distance	-	M
TR01.04	When the safety feature is activated, the system must come to a complete standstill	0 m/s	M
TR01.05	The safety feature must not estimate or infer the user's intended path; all obstacles within the threshold zone shall trigger braking regardless of rotation or motion history.	-	M
TR01.06	The safety feature must successfully detect and respond to obstacles and drop-offs with an emergency brake in at least 95% of the test cases	>95%	M

Table 3.2: Technical Requirements

4 Design

Given the abstract vision of the safety feature presented in Chapter 3 and the requirements outlined in Sections 3.5 and 3.6, some design choices remain open. The start of this chapter will motivate the design choices. The end of the chapter will show the final design

The goal of each decision is to create a system that is predictable, minimally intrusive, and responsive enough to prevent collisions and drop-offs. The following sections justify each design component individually:

1. The safety response behaviour.
2. The indication method and placement.
3. The threshold design.
4. The sensor placement.
5. The calibration method.
6. The measurement analysis.
7. The architecture within XRF2.

4.1 Safety Response

When the threshold is surpassed, the safety feature should respond appropriately. According to the stakeholder wishes in Section 3.1, the system should be simple. The safety response should be simple. Altering user paths or slowing the system down will result in unpredictable behaviour and complicate the system. The system response can take the shape of multiple alternatives. The goal of all systems is to slow the RELbot down until a full stop. The following ideas are generated:

1. **Locking the motors**
   When the threshold is surpassed, the safety feature locks the motors, i.e., no user input will be sent through to the FPGA.
2. **A response loop-controller slowing the system down**
   When the safety feature is triggered, the loop controller of the user is disabled, and a custom response loop controller will be enabled. This loop controller will decelerate as quickly as possible without generating a slip. This should have the lowest brake distance.
3. **Regulating the user loop-controller behaviour to slow down.**
   To give the user more freedom over the system, the safety feature will not take over the system, but only regulate it. The safety system will calculate a number between one and zero to indicate the risk of collision. Zero indicates high risk, and one indicates no risk. Consider risk as the product of the chance of collision and the damage that the collision will generate. The user control outputs will be multiplied by this number before being passed on to the FPGA. This will slow the system down in more risky scenarios, but still allows the user to make rotations or reverse since it does not fully take over.

To decide between these options, a decision table is created. Here, different criteria with different weights are created. Each option is graded 1 to 10 per criterion. The grade is multiplied by the weight of that criterion; the option with the highest sum is most optimal. The decision table is shown in Table 4.1.

The table criteria emphasise a simple system with minimal interference. CPU usage and user freedom reflect restrictions on the user's system, with higher scores meaning fewer limitations. A high predictability score indicates an intuitive system. Feasibility refers to how easily the system can be integrated, though it is less critical than the other factors.

Criteria	Weights	1. Motor Lock	2. Response loop	3. Limit regulator
CPU usages	6	8	7	7
User freedom	6	8	6	7
Predictability	8	7	6	6
Feasibility	3	8	7	6
Sum	-	**176**	147	150

Table 4.1: Decision table: Response Behavior

The optimal solution is a simple motor lock, chosen under the assumption that the RELbot will typically operate at low speeds, below 1 m/s. At these speeds, the risk of slipping is low enough that a separate controller is unnecessary. The other options are also viable, but add complexity and potentially reduce system intuitiveness. For these reasons, option one was chosen.

4.2 Indication Method

A physical indication is required when the safety feature is active, as per the stakeholder requirements in Section 3.5. The goal of this indication is to be immediately noticeable and intuitive for the user. The following options were considered:

1. I^2C display
2. Buzzer
3. Red LED

Although the I^2C display is already available on the RELbot, it is primarily used to show battery and supply voltages, and the IP addresses of the system. Re-purposing or sharing the screen for the safety feature would lead to confusion, especially since it is not as visible from all angles. This option is not noticeable enough.
The buzzer, while also present on the RELbot, was deemed less suitable as well. The option would be highly noticeable, but lacks specificity. A beeping noise could indicate a plural of states. The buzzer is also reserved for low-battery warnings. This aligns with common user expectations, such as smoke alarms.
In contrast, a red LED provides a clear visual cue that is comparable to other emergency shutdown indicators. Red lights indicate stop signals, e.g., traffic lights. This solution is also easily implemented via the Raspberry Pi's GPIO interface.
For these reasons, a red LED was selected as the indication method for the safety feature.

4.3 Threshold model

The threshold defines the distance at which the safety feature should intervene. As analysed in Chapter 3, this distance depends on velocity and the RELbot's breaking capabilities. The form of the equation is given in Equation 3.8.

A safety margin of 10 mm is selected to account for the measurement noise observed by Hofmann (2024). This is further tested in Appendix E, where these results were accurate up to 1 meter distance from the target.

The Equation 3.8 has two constants, c_1 represents the system's sensitivity and c_2 the offset. To determine these, brake tests were performed to record the stopping distance of the system. A report of this test is presented in appendix F. The RELbot was let go at one meter from a soft obstacle. It needs to detect that obstacle and brake at a given static threshold. The stopping distance at various speeds was recorded and shown in Figure 4.1.



Figure 4.1: Stopping Distance over Speed

The results indicate a clear increase in stopping distance when velocity increases. It is interesting to note that the spread of the stopping distances also increases with velocity. Additional testing at three selected speeds also confirmed this, shown in Figure 4.2.

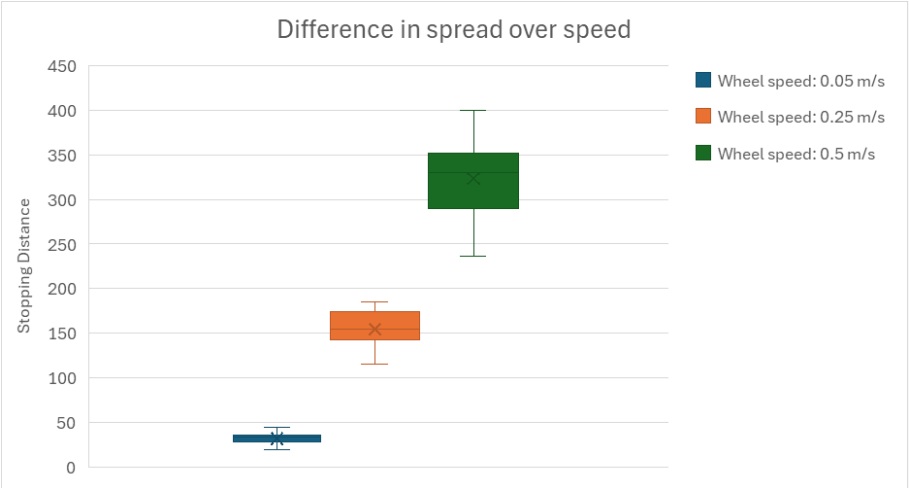


Figure 4.2: Stopping Distance Spread over Speed

To determine the optimal fit, a least-square error method was applied using MATLAB with the maximum values of the test results (see Appendix J). Using the maximum values ensures that the system is robust. Based on the actual wheel speed, the resulting fit has the parameters $c_1 = 951$ and $c_2 = 101$. However, computing real velocity during runtime requires additional calculations, which introduces unnecessary complexity. Actuator torque (Y) is already available within the XRF2 framework in millinewtonmeter. This can be used as a proxy for velocity, assuming they remain linearly related in the absence of external forces. The relationship between velocity and torque is provided in Appendix C. To change the formula to torque, the same model can be used, but the constants have to change. They are determined using the same method. The variable v^2 is exchanged for Y^2, and the parameters change to $c_1 = 0.0008$ and $c_2 = 95.6$.

The floor-edge detection has a limited resolution of three detectable values: 180mm, 240mm, and 300mm. These distances are calculated in the next section, Section 4.4, in Equations 4.9-4.11. Elevated surfaces carry a high risk since the damage from a drop is more significant than a collision. To address this, the detection threshold is rounded up twice to the next available value. The threshold calculation for object detection and floor-edge detection looks as follows:

$$D_{\text{threshold}} = 951 \times v^2 + 101 \quad \text{or} \quad D_{\text{threshold}} = 0.0008 \times Y^2 + 95.6 \tag{4.1}$$

$$D_{\text{floor_threshold}} = \begin{cases} 240\,\text{mm} & D_{\text{threshold}} \leq 180\,\text{mm} \\ 300\,\text{mm} & D_{\text{threshold}} > 180\,\text{mm} \end{cases} \tag{4.2}$$

4.4 Sensor Placement

The sensor placement proposed by Hofmann, 2024 used tape to secure the sensors. While sufficient for initial tests, this method is not robust for long-term use. Sensor misalignment during operation could compromise reliability. A mechanical redesign of the mounting is necessary.

The redesigned mounts are more rigid and have a greater floor-edge resolution. The redesign assumed that RELbots that operate on elevated surfaces generally move relatively slowly (<0.25m/s). The required sensor range for this speed is estimated using the threshold model from Section 4.3:

$$951.14 * 0.25^2 + 100.95 = 160mm \tag{4.3}$$

To ensure a sufficient safety margin, the desired range will be placed on 300 mm. This ensures consistent floor-edge detection.

To minimise obstruction of other components (e.g., the camera and number plate), the sensor will be mounted on the top plate of the RELbot and angled downward. The angle was calculated using initial measurements using the configuration of Hofmann (2024) (see Appendix D). These measurements show that when the RELbot is placed on a flat surface, the bottom row of the sensor detects distances between 190-220 mm, and the second-lowest row between 270-290 mm. This geometry is illustrated in Figure 4.3.

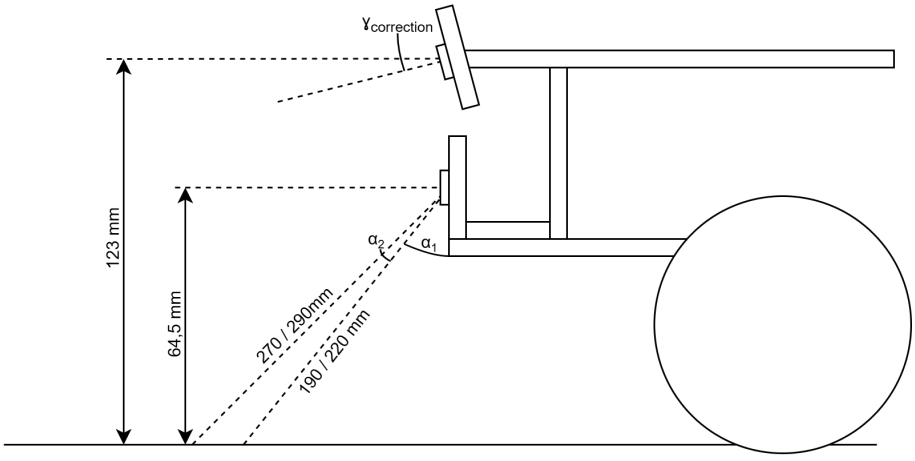


Figure 4.3: Sensor Placement on the RELbot

Using trigonometry, the detection angles are calculated as follows:

$$\alpha_1 = cos^{-1}\left(\frac{64.5}{190}\right) = 70.32^0 \quad \text{to} \quad cos^{-1}\left(\frac{64.5}{220}\right) = 72.95^0 \tag{4.4}$$

$$\alpha_2 = cos^{-1}\left(\frac{64.5}{270}\right) - \alpha_1 = 5.86^o \quad \text{to} \quad cos^{-1}\left(\frac{64.5}{290}\right) - \alpha_1 = 4.2^o \tag{4.5}$$

To gain resolution, it is desired to have three rows for floor-edge detection. Assuming linear spacing between sensor rows, the third row from the bottom corresponds to the following measurement angle:

$$\alpha_3 = \alpha_1 + 2\alpha_2 \simeq 80^o \tag{4.6}$$

To detect a floor-edge ahead with a vertical distance of 123 mm (the height of the RELbot top plate from the floor), the required angle is:

$$\beta_3 = tan^{-1}\left(\frac{300}{123}\right) = 67.70^o \tag{4.7}$$

To compensate for the difference between these two angles, a downward tilt is beneficial. The correction angle is:

$$\gamma_{correction} = \beta_3 - \alpha_3 = -12.3^o \tag{4.8}$$

Using these angles, the estimated detection distances are the following per row:

$$D_1 = 123 \times tan(\alpha_1 + 2\alpha_2 - \gamma_{\text{correction}}) \simeq 300mm \tag{4.9}$$

$$D_2 = 123 \times tan(\alpha_1 + \alpha_2 - \gamma_{\text{correction}}) \simeq 240mm \tag{4.10}$$

$$D_3 = 123 \times tan(\alpha_1 - \gamma_{\text{correction}}) \simeq 180mm \tag{4.11}$$

To calculate the exact distances observed by each row, the measurements are linearised by assuming that all cells within a given row detect the same distance. In practice, this assumption is only an approximation, as the actual measured distances vary slightly across the row due to the conical curvature of the sensor's field of view.

Concluding, the sensors will be mounted on the RELbot's roof at a downward angle of 12.3°. In this implementation, a custom press-fit mount was chosen as a practical solution that avoided additional modification work while providing sufficient robustness for testing and calibration purposes. For future redesigns of the RELbot, an integrated mechanical fixture is recommended.

4.5 Calibration

The technical requirements in Chapter **??** state that the safety feature must only respond to objects within the straightforward path of the RELbot. Objects outside of this region are ignored. To reliably distinguish between these zones, spatial calibration of the sensor is necessary.

Although it is theoretically possible to achieve the same goal using trigonometry, this will assume ideal sensor placement. In practice, small mounting variations can lead to significant deviations. As a result, relying solely on fixed geometry is not robust.

To address this, the system needs a reference, i.e. a calibration. The system requires the bounding box for the straightforward region to compare the measured data to. It can then determine if objects are placed within this region of interest or must be ignored. Using this method, sensor placement is irrelevant as long as it is consistent with the sensor placement during calibration.

The calibration method

The goal of the calibration process is to record a reference measurement of the forward path. This reference is stored as an 8×16 array corresponding to the sensor's measurement grid. These measurements indicate the edges of the region of interest, i.e. the straightforward region. To generate this reference, the RELbot is physically placed in a well-defined environment: a flat horizontal surface with two straight walls placed parallel to the RELbot's sides. The setup is schematically shown in Figure 4.4.



Figure 4.4: Calibration setup

To account for sensor noise, the calibration does not rely on one signal frame. Instead, the system collects a significant amount of data. This should contain at least 30 measurements, but is set to be 100 for this project. The calibration computes the median of these measurements for each cell in the 8x16 matrix. By choosing this approach, noise is assumed to be white and cancels out. The use of the median, rather than the mean, ensures robustness to occasional spikes in the data.

The calibration will return an 8x16 reference matrix that is stored as a numpy file on the RELbot. In practice, calibration only needs to be performed once after the sensors are physically mounted, provided the mounts remain fixed and undisturbed. For most educational and general-purpose use, initial calibration is enough.

The question remaining is how often the RELbot is used and how much the sensor mounts are disturbed during handling. Automatic recalibration is not feasible in this context. It requires a controlled environment as shown in Figure 4.4. The appropriate recalibration frequency is difficult to define, as it depends on the factors mentioned above. If the mounts shift slightly, recalibration is not necessary. However, if sensor placement is altered or higher precision is needed, a more recent calibration will be beneficial for system performance. It is recommended to create a more fixed mount for long-term use, which is bolted to the RELbot, so sensor disturbance becomes a smaller issue.

4.5.1 Limitations

By using this method, the safety feature evaluates only the straightforward region. While this prevents the RELbot from reacting to objects outside its width, it creates a risk during rotation. Obstacles outside the straightforward path may rotate into it without sufficient time or distance for the RELbot to stop, potentially leading to collisions. This limitation is illustrated schematically in Figure 4.5 and is a consequence of prioritising forward-path detection over some 360^o awareness.

Another limitation of this setup is the misclassification of slopes. Under the assumption of a uniform surface, slopes are detected either as obstacles or as drop-offs. This behaviour is expected and can be mitigated by adjusting the detection thresholds within the node configuration. Alternatively, the safety feature can be temporarily disabled in scenarios where such conditions are common.

Finally, the floor-edge detection is calibrated for the straightforward path. When approaching a drop-off on a steep angle, corner zones that would otherwise detect the edge may lie outside the region of interest. One potential improvement would be to introduce a separate calibration profile for floor-edge detection. However, this issue was identified too late in the project to be addressed.

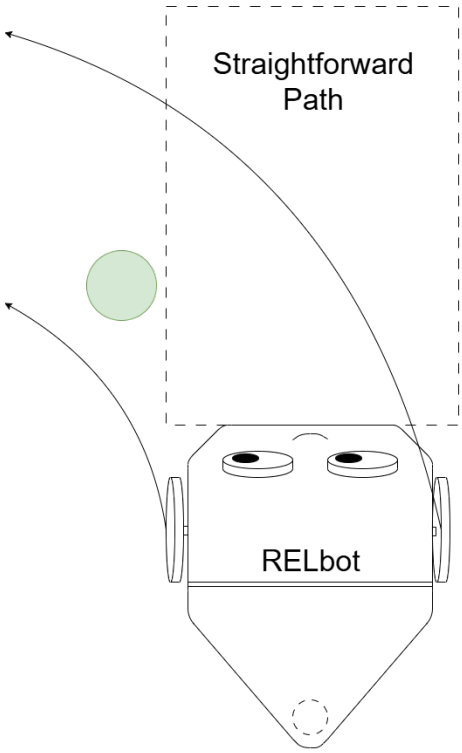


Figure 4.5: Limited field of detection issue

4.6 Data processing

The sensors return two 8x8 matrices filled with distances in millimetres. These two matrices are merged to create an 8x16 grid. The raw data is interpreted to return two scalar values: a distance to the nearest object and the floor-edge.

4.6.1 Floor Detection

Floor detection relies on the bottom three rows of the measurement grid. Each row corresponds to a constant distance to the front of the RELbot. When any of these measurements drop significantly, it indicates a missing floor. The setup is schematically shown in Figure 4.6

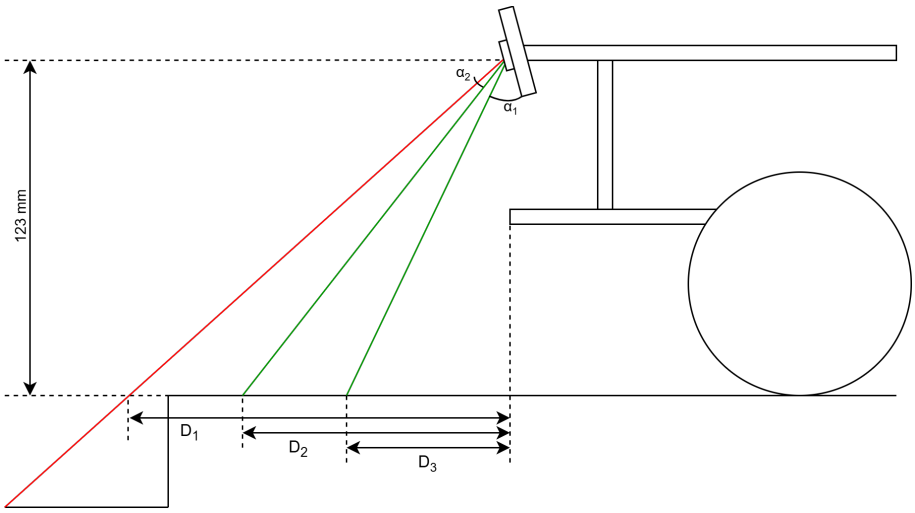


Figure 4.6: Floor Detection Explained

Here $D_1 = 300$, $D_2 = 240$, and $D_3 = 180$ as calculated in Equations 4.9-4.11.

To ensure reliable detection, the system monitors for significant drops in the measured distances. A drop is considered significant if the difference exceeds a row-specific threshold compared to its calibration value. This is done by subtracting the measurements from the calibration values and selecting the minimum value. This ensures robustness, especially when the RELbot approaches a drop at an angle. When multiple rows detect a drop, the closest drop (the smallest measured distance) is prioritised.
Because measurement noise increases with distance (as demonstrated in Appendix E), each row has a dedicated threshold to compensate for this effect. A drop is considered significant if it meets the following conditions:

- $C_1 - D_1 \leq -60mm$
- $C_2 - D_2 \leq -30mm$
- $C_3 - D_3 \leq -20mm$

Where:

- C_i is the calibration (expected) value for row i
- D_i is the latest measured value of row i

These thresholds control the system's sensitivity. When more tolerance is required, for example, when driving down a ramp, these thresholds should be increased to compensate.

4.6.2 Object Detection

To estimate the distance to the closest object in the straightforward path of the RELbot, the system simply takes the smallest valid measurement from the sensor grid. However, due to sensor noise, this value is not taken directly from a single frame.
To address sensor noise, the system maintains a sliding window of the last five frames. For each cell within the measurement grid, the median value of this sliding window is used. This median filters out outliers in the data. The median filtering is more robust than a simple average, as it effectively suppresses short-lived large spikes.
One complication arises in the 5^{th} row of the measurement grid. Although it is positioned in the upper region, it can sometimes detect the floor. This may be due to uneven terrain small movements within the sensor casing. If included in object detection, this may indicate a floor as an object nearby. For this reason, the 5^{th} row is excluded from object detection.
This approach ensures fast, noise-resilient detection.

4.6.3 Detection Regions

To summarise, the first 4 rows are only used for object detection. The 5^{th} row is neglected since it sometimes sees the floor and is unreliable. The bottom three rows are used for floor-edge detection and object detection, and positive values indicate objects, negative values indicate a floor-edge. This distribution is shown in Figure 4.7.

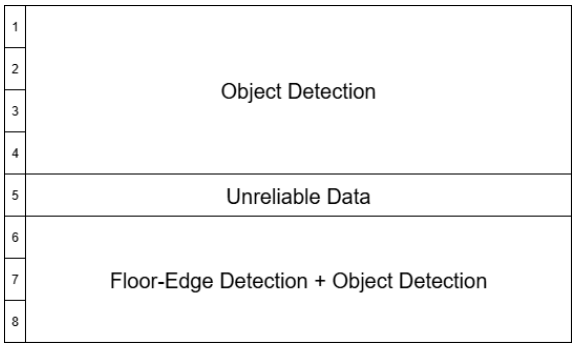


Figure 4.7: Detection Regions

4.7 Integration within the XRF2 Framework

To embed a safety feature in the XRF2 framework, the system can be decomposed into four functional components, processed in the following sequence:

1. **Sensor Readout:** This component interfaces with the hardware sensors via their drivers and retrieves raw distance measurements.
2. **Data Analysis:** Processes the sensor data to identify the closest object and the nearest floor-edge.
3. **Threshold Calculation:** Computes a dynamic safety threshold using the current velocity and the function defined in Equation 3.8.
4. **Brake Sequence:** Compares environmental data to the calculated threshold and triggers braking if the threshold is surpassed. It also indicates when the safety feature is active.

The connection between these modules, the system state, the sensors, and the indication LED is schematically shown in Figure 4.8.

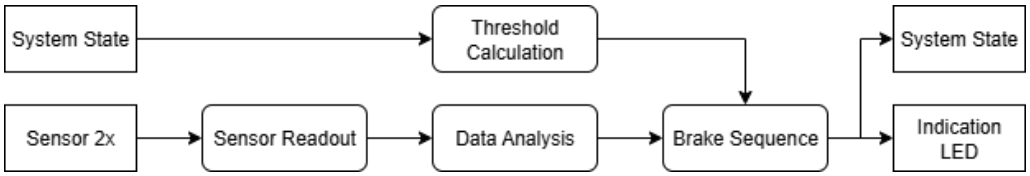


Figure 4.8: Functional flow between functional components

Each of these components must be implemented either within the firm-real-time domain in XRF2 within Xenomai or in the soft real-time environment of ROS2. The primary factor in making this decision is the communication overhead across the ROS2–Xenomai boundary. Data exchange via the `ROS2XenoBridge` (Fig. 2.1.2) introduces complexity and latency that should be minimised.

Sensor Readout (ROS2)

The sensor readout involves I2C communication with multizone ToF sensors, returning two full 8×8 distance matrices per frame. This interface does not behave reliably under firm-real-time conditions. ROS2 is better suited for handling this volume of data asynchronously from the control loop and reliably. For these reasons, the sensor readout is placed in ROS2.

Data Analysis (ROS2)

The raw measurement matrix from the sensors is relatively large and not ideal for transferring across the `ROS2XenoBridge`. Instead, it is more efficient to perform the initial data analysis in ROS2. This step will estimate the closest object and the nearest floor-edge as a result. Placing this within ROS2 makes the software computationally lightweight, as it only needs to run once per new frame. Additionally, the outputs of this step, two scalar values, are much easier to send to the XRF2 in Xenomai than the full 8x16 matrix.

Threshold Calculation (XRF2)

The threshold depends on the system velocity, which is monitored and controlled within XRF2. Reading this velocity directly within XRF2 avoids the need to transmit real-time control signals into ROS2. Furthermore, threshold evaluation must be consistent with the most recent system state, ensuring that decisions are based on precise timing. Hence, the threshold calculation is best embedded in XRF2.

Brake Sequence (XRF2)

The brake sequence is the most time-sensitive component in the safety feature. Once the threshold is surpassed, the response must be as fast as possible. Implementing this in ROS2 would introduce unacceptable latency and jitter, as ROS2 does not provide firm real-time guarantees. Alternatively, placing the brake sequence in ROS2 can be achieved by altering the user's control setpoints. However, altering the user's control setpoints assumes knowledge about the user's control strategy, which may be unknown or incompatible with safety requirements. Directly controlling actuators in XRF2 ensures the response is reliable and independent of the user's controller. Therefore, this component is placed in XRF2.

Conclusion

To ensure reliable operation and minimise communication overhead, the system is divided into components based on their individual needs. The sensor readout and data analysis are handled in ROS2, as they deal with a large amount of data and do not require strict timing. The threshold calculation and brake sequence are integrated within XRF2. Here, fast and predictable responses can be guaranteed. This setup keeps the safety feature responsive and efficient.

4.8 Cyber-Physical System

The cyber-physical system outlines the interaction between software and hardware, as illustrated in Figure 4.9. Within ROS2, sensor data is handled by the `tof_readout` node, which manages the drivers and reads data from the sensors. This node processes the raw input and publishes two key measurements: the closest object `/closest_object`) and the distance to the nearest floor-edge (`/floor_distance`).

The user provides a desired target setpoint via the `/safe_setpoint` topic. This data is purely used for the user's controller and does not alter the safety feature behaviour.

All data is packaged and transmitted to the XRF2 environment through the `RosXenoBridge`. Within the XRF2, the system evaluates whether the nearest detected object or floor-edge is within the calculated threshold. The threshold is calculated based on the current velocity, see equation 4.1. If so, it overrides the user control, triggers an emergency stop, and signals this state by activating the red emergency LED.

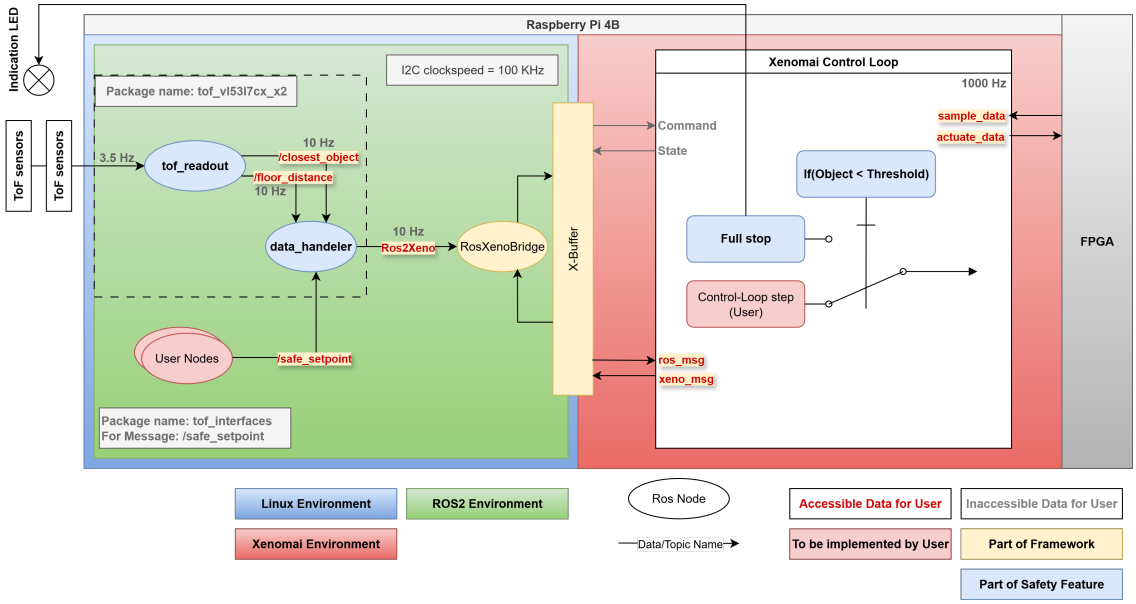
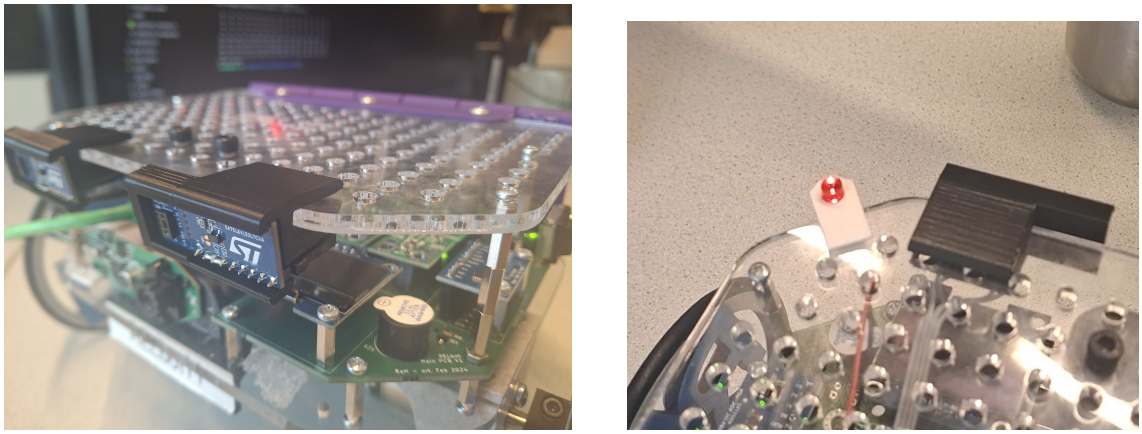


Figure 4.9: The Safety Feature Architecture

4.9 Hardware

Two hardware components were added to the RELbot. This includes an indication light for when the safety feature is active and improved mounts for the multi-zone ToF sensors. Both components are mounted close to each other to emphasise their functional association. The added hardware is shown in Figure 4.10.



(a) The improved sensor mounts

(b) The indication light

Figure 4.10: Hardware adaptations

The sensors require an I^2C connection. The indication light can be wired up to any GPIO output, but needs a resistor in between. The components in the current configuration are wired up in the following way:

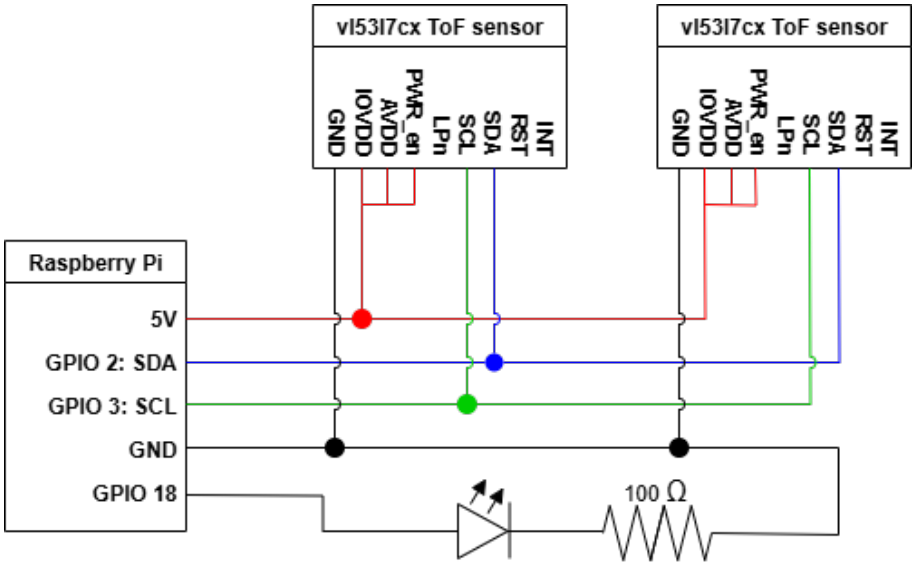


Figure 4.11: Electrical Circuit

4.10 Functional Flow

To display how the safety feature works within the overall system, the following functional flow diagram shows the sequence of operation from sensor input to actuator control. This is shown in Figure 4.12. The diagram provides a high-level overview of how sensor data is processed and how safety decisions are made. The sensor is read out and analysed in ROS2. The system state is read out from the FPGA to calculate the Threshold using the threshold model. If the detected values from the analysed data are smaller than the threshold, the system will activate the brake sequence. This will perform an emergency brake and activate the LED to indicate the safety feature is active. The elements at ROS2 operate at 10 Hz, and the XRF2 loop operates at 1000 Hz.

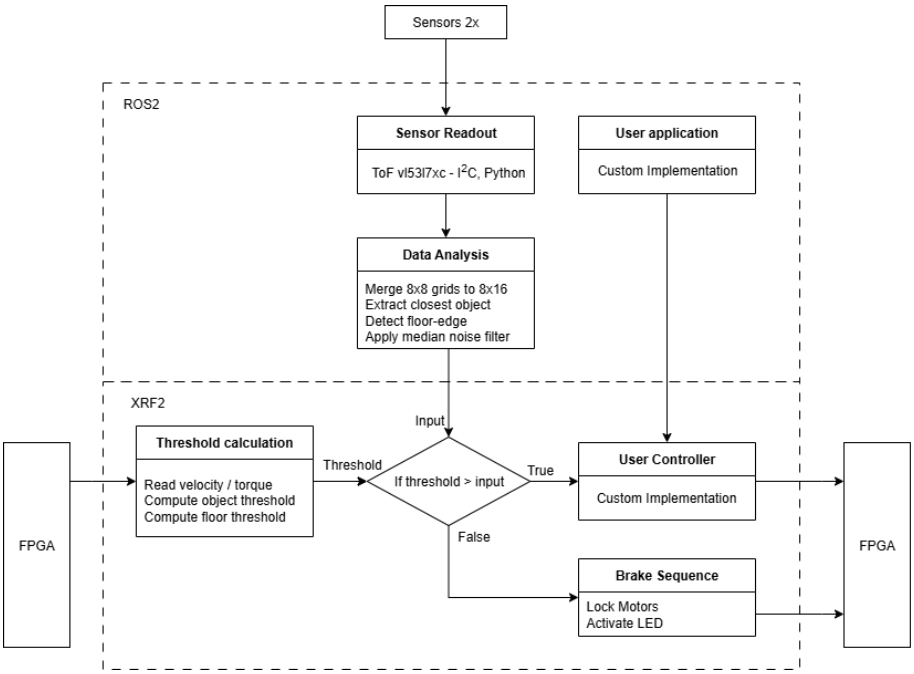


Figure 4.12: Functional Flow Diagram

5 Tests

This chapter evaluates the safety feature through testing, analyses the results and assesses its effectiveness and limitations. Each test will briefly introduce its goal and setup, show results and draw conclusions from those results. Irrelevant data and detailed test protocols are not shown in this chapter, but are added to the Appendix E-H.
This chapter will address the following key findings:

1. The influence of measurement noise within the system.
2. The bottleneck for overall system speed.
3. CPU usage of the system.
4. Fully integrated system brake performance.

5.1 Sensor-noise tests

Earlier work estimated the noise of the VL53L7CX sensor to be relatively constant and typically within ±10 mm, Hofmann, 2024. During development, the measurement noise was found to be significantly higher and not uniform. The goal of these tests was to map this behaviour. A Python script was written that calculated the standard deviation of each measurement cell over 100 measurements. This script was used in two different setups, one focusing on the influence of distance and on focusing on the effect of external factors. A detailed description of this test is added to Appendix E.

5.1.1 Setup

Two setups were created. The first setup focused on the influence of distance on the sensor noise under normal operating conditions. The setup for this test was a wooden wall placed at various distances in front of the RELbot.
The second setup focused on the influence of external factors. For this, the RELbot was placed in a controlled room where lighting conditions and human presence were systematically altered.

5.1.2 Results: The influence of distance

The result of this test includes a set op heat-maps, where larger deviations have deeper colours. The heat-maps generated from these measurements are shown in Appendix E. For context, one standard deviation (σ) indicates that approximately 68% of measurements fall within $\pm\sigma$ of the mean. Averaging across all cells, the following trend emerges: Standard deviation increases

Distance to the wall	Average standard deviation
0.5 m	5.23 mm
1.0 m	14.73 mm
1.5 m	109.03 mm
2.0 m	206.30 mm
2.5 m	190.80 mm

Table 5.1: Average standard deviation of sensor measurements at different distances from the wall.

exponentially with distance, peaking at around two meters. After this point, noise appears to decrease slightly but remains far too high for reliable measurements, indicating that the sensor becomes unreliable at longer ranges (>2m). It is assumed that noise becomes uniform after two meters and stays around $\pm 200mm$.

5.1.3 Results: The influence of external factors

The controlled room used for these tests was room CR3420. The sensor faced a wall at 1 m distance, and again 100 measurements were taken systematically changing conditions. The results are shown below: The results indicate that lighting conditions have a noticeable impact

Different conditions at 1 m in room CR3420	Average std (mm)
Lights on	14.13 mm
Lights on, no people	14.02 mm
Lights off	11.89 mm
Lights off, no people	11.16 mm

Table 5.2: Average standard deviation under different lighting and presence conditions at 1 m distance in room CR3420.

on measurement reliability. Turning off the lights reduced the standard deviation by roughly 3 mm. Human presence had only a minor effect, suggesting that reflected infrared noise from people is not a major noise contributor. Although these environmental effects are measurable, they are not significant enough to disrupt the system's functionality in normal use. However, they do reinforce the importance of thresholding and filtering when operating near the system's detection limits.
It is also interesting to note that under normal operating conditions (with people and lights on), both tests showed similar results at 1 meter. This indicates that no other significant noise factors are present.

5.1.4 Discussion

The results show that the VL53L7CX sensor exhibits non-uniform, distance-dependent noise with standard deviations increasing significantly beyond 1 meter. External conditions also affect measurements, but to a lesser extent. While the sensor remains reliable at close range ($\leq$ 1m), these findings reinforce the need for dynamic thresholding, filtering, and a proper safety margin. Overall, the noise characteristics of the sensor are accounted for to ensure safe and robust behaviour under different conditions.

5.2 System Speed tests

The safety feature operates in real-time. Generally speaking, faster response times allow for lower thresholds and tighter tolerances, improving overall system performance. The goal of these tests is to find the bottleneck for the current system speed.

Looking at figure 4.9, the primary bottleneck in the current setup lies in the sensor readout rate. Although the VL53L7CX datasheet specifies a maximum frame rate of 60 Hz, the actual performance within the RELbot system is significantly lower.

5.2.1 Setup

To investigate this, various software and hardware configurations were tested. This includes different sensor resolutions, different sensor setups, and different devices. Altering the I^2C clock speed on the Raspberry Pi of the RELbot was unsuccessful. A different Raspberry Pi (model 3) was used to prove that the 60 Hz frame rate was possible.

5.2.2 Results

All different setup performances are shown in Table 5.3.

Language	Platform	Sensors	Resolution	I2C Speed (kHz)	Average Sample Rate (Hz)
Python	Pi 4b	1	8×8	1000	9.3
Python	Pi 4b	2	8×8	1000	3.5
Python	Pi 4b-ROS	2	8×8	1000	3.4
Python	Pi 4b	1	4×4	1000	18.1
C++	Pico	1	4×4	1000	18.0
Python	Pi 4b	2	4×4	1000	9.4
Python	Pi 4b	1	4×4	1000	18.6
Python	Pi 3	1	4×4	4000	60.0
Python	Pi 3	1	8×8	4000	31.4

Table 5.3: Sample rate measurements under different software configurations and sensor setups.

The table shows that a frame rate of 60 Hz is possible with one sensor at 4×4 resolution with a 4000 kHz I^2C clock speed. For the required 8×8 resolution used in floor-edge detection, the maximum achievable frame rate with one sensor drops to approximately 30 Hz. However, this I^2C speed could not be set in the RELbot environment due to unresolved issues.

Furthermore, using two sensors divides the total throughput between them, reducing the frame rate to approximately 3.5 Hz in the current implementation.

5.2.3 Discussion

The test results show that neither the choice of programming language (Python vs C++) nor the addition of ROS2 introduces significant overhead to the sampling rate. The dominant performance limitation is the I^2C clock speed. Resolution and the use of two sensors are additional contributing factors. The maximum achievable frame rate for a dual sensor setup should hypothetically be half of 30 Hz, thus 15 Hz.
The most straightforward and impactful improvement is to increase the system's I^2C speed.

5.3 CPU usage

According to the requirements defined in Section 3.5, the safety feature may use up to 30% of the CPU capacity. The goal of this test is to evaluate the CPU usage of the system.

5.3.1 Setup

The test was performed by running the safety feature in isolation and using the `top` command in Linux to analyse its behaviour. The CPU usage is expected to be uniform since it operates at a fixed rate.

5.3.2 Results and Discussion

The raw results of this test are shown in Appendix H. The system consistently averaged a CPU usage of 12.2%. Therefore, requirement SR01.02 is met.

5.4 The Dynamic Brake Tests

To validate the full integration and performance of the safety feature, the system was tested in real-world braking scenarios. The goal of these tests was to find the limitations and performance of the full safety feature. For these tests to be successful, the system needs to detect an object or floor-edge, calculate the threshold distance, and intervene before a collision or drop-off. The tests are split up into two setups:

1. Object Detection and Collision Avoidance.
2. Floor-edge Detection and Drop-off Prevention.

A detailed description of these tests is provided in Appendix G.

5.4.1 Setups

For object detection and collision avoidance, the system was tested using the same setup as the first brake tests in Appendix F. The RELbot is placed at one meter from a soft obstacle and let go at various predetermined initial velocities.

For floor-edge detection and Drop-off Prevention, the RELbot was placed on a flat, empty table one meter from the table edge. The RELbot was then also let go at various predetermined initial velocities.

For both tests, the distance left to the object or floor-edge was recorded. The results are shown separately below.

5.4.2 Results: Object Detection and Collision Avoidance

The results of this test include the recorded safety margin over various initial velocities, i.e. the distance left between the RELbot and the obstacle. Velocities were tested up to 0.65 m/s. The system is expected to detect the object and engage the brakes in time to avoid contact. The results are shown in Figure 5.1.

The system failed in 5 out of 125 trials, resulting in a failure rate of 4%. The RELbot consistently stopped with at least 10 mm clearance from the obstacle, demonstrating reliable performance.

Interestingly, at higher speeds, the system achieved a larger safety margin. The trend in the graph shows promising behaviour. This shows that the safety feature will not fail in collision avoidance when the velocity increases.



(a) Results over speed **(b)** Error

Figure 5.1: Dynamic Threshold Braketest: Objects

5.4.3 Results: Floor-edge Detection and Drop-off Prevention

Drop-off avoidance was tested up to a maximum speed of 0.3 m/s. The results are shown in Figure 5.2



(a) Results over speed **(b)** Error

Figure 5.2: Dynamic Threshold Braketest: Drop-off

The graph shows a clear increase in braking threshold beginning at 0.15 m/s, which is the expected response to higher speeds. Due to limited floor-edge resolution, the system may fail after 0.3 m/s. This can also be seen in the overall trend of the graph. It was not tested to prevent significant damage to the system.

5.4.4 Extra Observation: Jitter at High Speeds

As speed increases, the dynamic threshold is placed further ahead of the RELbot. However, sensor noise also increases with distance, as shown in Appendix E. This causes the safety trigger to activate prematurely or inconsistently, which may result in jittering behaviour. Especially when the RELbot operates near the system's speed limits (>1 m/s), this behaviour becomes

noticeable. While this is an unavoidable result, this is only valid at higher speeds. Typical use cases should not trigger this behaviour significantly.

5.4.5 Discussion

The object detection and collision avoidance of the safety feature proved to be robust and met all requirements. The failure rate of these tests was 4%, which meets the requirement in TR01.06 (maximum 5% failures). The closest recorded safety margin without collision was 12 mm, satisfying the minimum safety margin of 10 mm.

The performance of the floor-edge detection and drop-off prevention of the safety feature declined above 2.5 m/s. This is primarily due to the limited floor-edge resolution. The results are lower than expected since the initial calculations in Chapter 4.4 showed a theoretical stopping distance of 160 mm at 0.25 m/s.

However, the system performed as desired, meeting its requirements within the typical operating velocities (<0.25 m/s on elevated surfaces and <0.6 m/s on the floor).

Overall, the system is deemed reliable for its intended educational use.

5.5 Pass/Fail Summary

Returning to the set requirements, each will be evaluated for its performance during the test. The table below will show the detected value if relevant and the overall pass/fail of that requirement

Nr.	Requirement	Measured Value	Pass/Fail
SR01.01	The user must be able to easily disable the safety feature by changing one boolean value	–	Pass
SR01.02	The total computational load of the system should not exceed 30% of the available CPU capacity.	12.2%	Pass
SR01.03	When the safety feature is active, this must be indicated physically.	–	Pass
SR01.04	The safety feature should be compatible with existing RELbot use cases, allowing small adaptations	–	Pass
SR01.05	The user's control loop must be suspended while the safety feature is active.	–	Pass
SR01.06	The user's control loop should automatically resume once the safety condition has been resolved.	–	Pass
TR01.01	The safety feature must keep at least 10 mm distance from any floor-edge or object	> 12 mm	Pass
TR01.02	Objects and floor-edges outside of the forward path of the RELbot should be ignored.	-	Pass
TR01.03	The safety feature must only activate when objects or floor-edges are within the threshold distance	-	Pass
TR01.04	When the safety feature is activated, the system must come to a complete standstill	0 m/s	Pass
TR01.06	The safety feature must successfully detect and respond to obstacles and drop-offs with an emergency brake in at least 95% of the test cases	96%	Pass

Table 5.4: Pass/Fail Summary Table

6 Conclusions and Recommendations

6.1 Conclusions

The objective of this project was "to develop and implement a safety feature for the RELbot using multi-zone time-of-flight sensors". This was integrated into the XRF2 framework to prevent collisions and drop-offs. This objective was achieved.

All defined tasks were completed. The desired system behaviour was analysed, stakeholder and technical requirements were established and met, and a safety feature was designed and integrated using custom press-fit sensor mounts. The system was implemented with ROS2 handling sensor readout and data analysis, and XRF2 executing threshold calculation and braking, minimising latency.

Testing confirmed that the system meets all key requirements. Collision and drop-off avoidance was successful in 96% of trials (≥95% required), the minimum safety margin recorded was 12 mm (>10 mm required), and CPU usage averaged 12.2% (<30% allowed). A clear visual indication is provided when the system is active, and interference with user control remains minimal and predictable.

Additional outcomes include a calibration method for defining the straightforward detection region, a velocity-dependent threshold model based on braking tests, and a filtering approach to reduce the impact of sensor noise. The resulting design achieves the intended balance between simplicity, responsiveness, and low computational load for use in educational settings.

Some limitations remain. These include reduced performance at higher speeds, misclassification of slopes, and reduced floor-edge detection accuracy when approaching at steep diagonal angles. Nonetheless, within the defined operating conditions, the safety feature performs reliably and fulfils its intended purpose.

6.2 Recommendations

The following recommendations are ordered according to priority in which they should be addressed. Higher-ranked items offer the greatest improvement to system performance relative to the effort required for implementation. The following recommendations are listed:

1. **Increase I^2C clock frequency:** by increasing the clock frequency to 4000 kHz, the bottleneck frame rate will boost up to approximately 15 Hz. This could improve filtering with larger sliding medians and reduce latency. This was not achieved within the current RELbot configuration, but should theoretically be possible.
2. **Change sensor I^2C addresses on boot:** In the current configuration, I^2C addresses of the sensor will reset every time the system boots. The sensors I^2C addresses can't be changed permanently. An alternative solution should be found using either GPIO pins or a multiplexer to communicate separately with the sensors. This way their I^2C address can be changed automatically after boot.
3. **Revisit threshold model:** The current quadratic model may overcompensate at higher speeds. The brake distance is less significant than first hypothesized. The most significant distance is the response distance. Changing the threshold model to a linear model or a combination of both may better map the stopping distance.
4. **Improve physical mounts:** Sensor alignment and reliability would benefit from fixed, rigid mounting solutions. Press-fit mounts are a great substitute, but not sufficient for the long term.
5. **Integrating the `relbot_adapter`:** Interchanging the `data_handeler` node from the current framework (Fig. 4.9) to the more commonly used `relbot_adapter` will increase modularity.
6. **Separate floor calibration:** Because floor detection benefits from the full field of view, not only the straightforward path, its reference calibration should reflect that. A separate calibration with its reference environment would improve robustness.
7. **Switch to a single ToF sensor:** Switching to a signal sensor will also double the frame rate of the system. While the field of view is reduced, this change may be a worthwhile trade-off for improved responsiveness. It rarely happened that only one sensor was seeing an object.

A more detailed description of how to achieve some of the recommendations is provided in Appendix B. This includes recommendations 3, 5, and 6.

A User Manual

This appendix provides a user manual for the RELbot safety feature. It is intended for students, technicians, and teaching staff who will install, configure, and operate the safety feature. The instructions cover essential setup steps, calibration procedures, and system configuration options required for correct operation. The goal is to enable end-users to integrate the safety feature into their RELbot projects. The focus lies on the version displayed in the report.

A.1 Installation

The safety layer is implemented in two parts:

- A **ROS2 component** that handles sensor communication, data processing, and message passing.
- A **XRF2 component** that applies the safety logic and controls the hardware response.

For an overview of their interaction, see Chapter 4.7. The overall software structure is shown in Figure A.1.

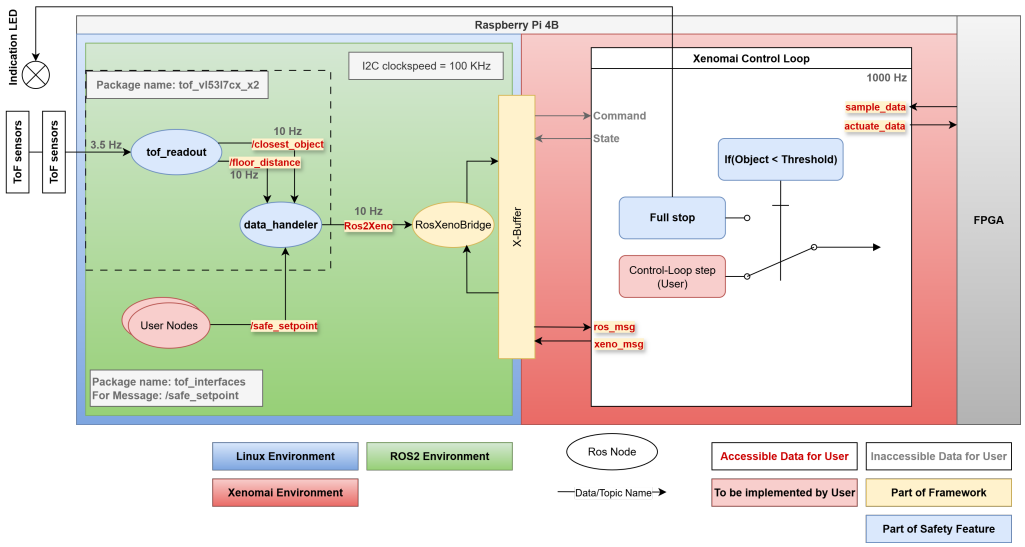


Figure A.1: Safety feature layout

A.1.1 ROS2 Packages

The ROS2 part consists of:

- `tof_interfaces`: contains the custom message `/safe_setpoint` used by the safety logic.
- `tof_vl53l7cx_x2`: contains the nodes for sensor readout and safety data handling:
  - `tof_readout`: Connects to the VL53L7CX sensors, handles their drivers, and publishes:
    - `/closest_object`: nearest obstacle distance.
    - `/floor_distance`: nearest floor-edge distance.

    Note: Sensors must use I^2C addresses `0x31` (left) and `0x32` (right), as seen from the front of the RELbot.
  - `data_handeler`: Combines `/closest_object`, `/floor_distance`, and the user's `/safe_setpoint`, then publishes them to the `/Ros2Xeno` topic for transfer to Xenomai.

The `/Ros2Xeno` message must be updated in:

`XRF2/XenoRosFramework/Common/XRF2_msgs/msg/Ros2Xeno.msg`

Change its contents to:

```
float64 x
float64 y
int32 floor_distance
int32 closest_object
```

A.1.2 XRF2 Component

The XRF2 part uses an adapted template from the XRF2 framework. This template can be found in the "Safety Feature Xenomai Template" folder of the Git repository. Its main logic is wrapped in an if-statement that checks whether the measured object or floor-edge distance falls below the calculated threshold. If so, the safety feature triggers.

The Xenomai code performs:

1. **Threshold calculation**: Based on the current velocity, using the calibrated quadratic relationship (Eq. **??**).
2. **Emergency brake activation**: Stops the wheels when a collision is detected.
3. **Visual indication**: Turns on the red LED.

A.2 Calibration

Calibration only has to be performed after the initial sensor mounting or after mechanical adjustments to the sensors. Software can be found within the calibration folder on the git repository (src/calibration_software calibration_tof.py). The calibration script stores a measured reference value, which is used by the `tof_readout` node. It stores in the folder where the software was run. The `tof_readout` node should be started in the same folder.

Procedure

1. Place the RELbot on a flat, uniform surface with two walls beside the RELbot's width, see Figure A.2. A real-world example is shown in Figure A.3.
2. Ensure the RELbot is stationary.
3. Run the calibration script:

```
sudo python3 calibration_tof.py
```

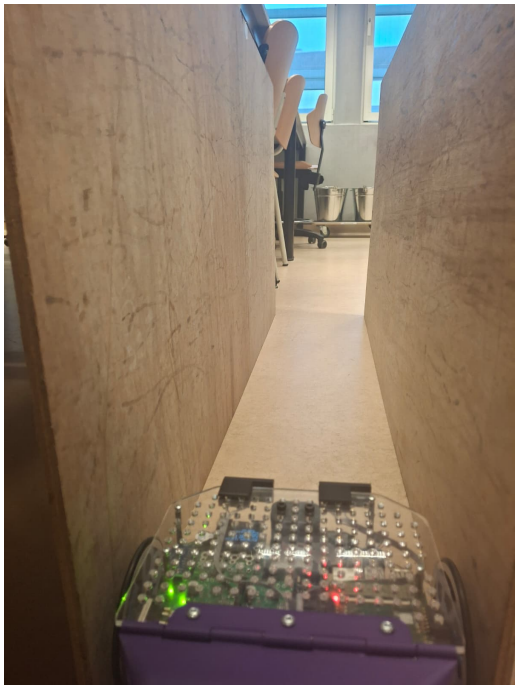
4. Run the script for at least 30 seconds.
5. The script will record the last 100 consecutive frames, calculate the median for each zone, and save the values to the calibration file.
6. After at least 30 of stationary measurements, press ctrl+c to close the application and store the measurements.
7. Once complete, verify that the calibration file is loaded correctly by restarting the safety layer node and checking the log output. It will state "Loaded calibration matrix." if done correctly.



Figure A.2: Ideal RELbot Calibration setup



(a) Top View



(b) Front View

Figure A.3: Calibration Setup Example

A.3 Change I²C Addresses

In the current configuration, changing the I²C addresses of the ToF sensors must be done manually. An automatic address change at startup is not possible in the current hardware configuration, as this would require either a dedicated GPIO control line or an I²C multiplexer to address each sensor individually during initialisation.

Current Procedure

1. Disconnect the *right* sensor when viewed from the front of the RELbot.
2. With only the left sensor connected. This sensor will respond to the default I²C address `0x29`.
3. within the `change_tof_ip.py` set the desired I²C address to `0x31`.
4. Run the address change script to set the left sensor to `0x31`:

```
python3 change_tof_ip.py
```

5. Reconnect the right sensor.
6. Now only the right sensor responds at `0x29`.
7. within the `change_tof_ip.py` set the desired I²C address to `0x32`.
8. Run the script again, this time changing the right sensor to `0x32`.

Addressing Standard

Throughout this software, the standard configuration is:

- Left sensor: `0x31`
- Right sensor: `0x32`

The default factory address for VL53L7CX sensors is `0x29`. Changing the I²C addresses to alternative addresses can also be done using this software; however, the standard configuration is assumed for the `tof_readout` node.

A.4 Reproducing the Demo

This section describes how to set up and run the demo of the RELbot safety feature. The demo shows the system detecting collisions and drop-offs in real-time.

Setup

1. Clone the `relbot_tof` repository into your own ROS2 workspace `src` folder:

```
cd ~/ros2_ws/src
git clone <repo-url>
cd ..
colcon build
```

2. If you are using the current default hardware configuration, a calibration file is already included. If you have changed the sensor setup or mounting position, perform a new calibration as described in Section A.2.

Running the Demo

Three separate terminal windows are required to run the full demonstration:

1. **Terminal 1: Xenomai real-time control** Navigate to the build folder and run the Xenomai safety module:

```
sudo ./relbot_tof/build/demo/demo
```

2. **Terminal 2 – ROS2 safety infrastructure** Launch the background processes for sensor readout, data processing, and safety logic. The following command should be executed in the same folder where the calibration matrix (calibration_matrix.npy) is located:

```
ros2 launch tof_python_node demo_setup.launch.py
```

3. **Terminal 3 – Keyboard teleoperation** Start the teleoperation script to manually control the RELbot with your keyboard. This script cannot be integrated into a ROS2 launch file and must always run in a separate terminal. Use the following command:

```
ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

Demo Behaviour

- Use your keyboard to drive the RELbot around. - When an obstacle or drop-off is detected, the safety feature will activate:

- The red indication LED will turn on.
- The teleoperation input will be ignored, and the wheels will be locked.

- If the RELbot is picked up, no floor will be detected, and the wheels will lock automatically. - To resume control, simply move the RELbot out of the collision or drop-off path. The teleoperation controller will automatically regain control.

B Suggested Changes

This chapter is a continuation of the recommendations given in chapter 6.2. The following recommendations will be explained in further detail:

3 Revisit threshold model.
5 Integrating the `relbot_adapter`
6 Seperate floor calibration

B.1 Revist threshold model

During the original analysis phase, the safety threshold was modeled as a quadratic function, see Chapter 3. However, during the testing phase, the braking distance seemed negligible at the current range of speeds. This means that the total stopping distance is dominated by the reaction distance, which scales linearly with velocity.

Reaction distance can be expressed as:

$$d_{\text{reaction}} = t_{\text{reaction}} \times v \tag{B.1}$$

Where:

- t_{reaction} is a constant range of reaction time (s)
- v is the current average velocity over both wheels (m/s)

Revisiting the brake test data in Appendix F. A linear model can be fitted on the maximum values, similar to the method chosen in Chapter 4.3. The results are shown in Figure B.1.

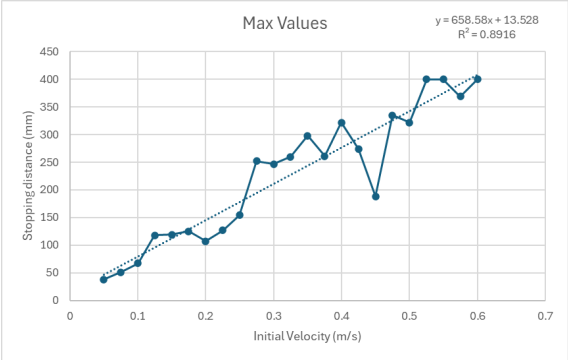


Figure B.1: Stopping Distances over Velocity

The updated linear threshold model is:

$$\text{Threshold} = 658.58\, v + 13.53 \tag{B.2}$$

where the constants are obtained from a least-squares fit to the measured data.

B.2 Integrating the `relbot_adapter`

The `relbot_adapter` is a recent addition to the XRF2 framework. It enforces safety regulations by limiting velocity values and also interfaces with the RELbot simulator. The node publishes to the `/Ros2Xeno` topic and shares similarities with the `data_handler` node used in this project.
To improve modularity and reduce redundancy, these two nodes should be merged into a single component.

To visualise how this merge could be achieved, pseudo-code was created and then translated to C++ using AI assistance. The pseudo-code is shown below. IT combines the functionalities of both software. The new node is designed to occupy the same role and space within the current framework as the existing `data_handler` node, with the added functionalities of the `relbot_adapter` node. Although this code has not yet been tested due to time constraints, it can serve as a starting point for further development. It can be found in the Git repository as `tof_relbot_adapter.cpp`.
For this change, the `/Ros2Xeno` topic must be adapted to carry the extra data. It can be found in:

`XRF2/XenoRosFramework/Common/XRF2_msgs/msg/Ros2Xeno.msg`

Change its contents to:

```
float64 x
float64 y
int32 floor_distance
int32 closest_object
```

B.2.1 Pseudocode

```
Node RELbotAdapterPlus:

  Variables:
    double x, y
    int floor_distance, closest_object
    double max_speed_mps, max_speed_rad, max_speed

  Constructor:
    - Declare parameters (use_twist_cmd, gain, robotmode, safety
   limits)
    - Create publishers:
        motor_command_sim  -> RelbotMotors
        ros2xeno_pub       -> Ros2Xeno (x, y, floor_distance,
   closest_object)
    - Create subscribers:
        if use_twist_cmd:
            sub /cmd_vel -> twistCallback()
        else:
            sub /left_motor/setpoint_vel  -> leftWheelCallback()
            sub /right_motor/setpoint_vel -> rightWheelCallback()
        sub /floor_distance  -> floorDistanceCallback()
        sub /closest_object  -> closestObjectCallback()
    - Calculate max_speed from safety params
    - Start timer for output_callback()

  twistCallback(msg):
    - Convert twist to left_wheel_vel, right_wheel_vel

  leftWheelCallback(msg):
    - store left_wheel_vel as x

  rightWheelCallback(msg):
    - store right_wheel_vel as y

  safeSetpointCallback(msg):
    - Read x,y from msg
```

```
  - If sqrt(x^2 + y^2) > max_speed_mps -> scale down
  - Store x,y
  - publishCombinedMessage()

floorDistanceCallback(msg):
  - Store floor_distance
  - publishCombinedMessage()

closestObjectCallback(msg):
  - Store closest_object
  - publishCombinedMessage()

publishCombinedMessage():
  - Fill Ros2Xeno with x, y, floor_distance, closest_object
  - Publish ros2xeno_pub

output_callback():
  - If |x| or |y| > max_speed:
      scale both to max_speed
  - Fill Ros2Xeno with x, y, floor_distance, closest_object
  - Publish ros2xeno_pub
```

B.3 Separate Floor Calibration

The current calibration procedure only focuses on the straightforward path, i.e., what is directly in front of the RELbot. While this approach is optimal for collision detection, it does not fully exploit the sensor's potential for floor-edge detection. The current calibration setup is shown in Figure B.2.



Figure B.2: Straightforward Calibration Setup

Drop-off avoidance carries a higher risk since it can cause more damage to the RELbot. By using the full FoV of the ToF sensors for floor-edge detection, the system can better detect drop-offs. This includes those slightly to the sides, which is currently a system limitation.

B.3.1 Proposed Approach

It is recommended to add a second calibration, dedicated to floor-edge detection. This can be achieved using the same software as the straightforward path calibration; the main difference being the environment. This results in two calibrated reference frames:

- **Straightforward path calibration**: Remains as the primary reference for object detection in the straightforward path.
- **Floor calibration**: A new calibration capturing the floor over the full field of view.

During operation, floor-edge detection would use the floor calibration as a reference, while object detection would continue to use the straightforward path reference.

B.3.2 Calibration Environment

The floor calibration should be performed in an environment that is flat (e.g., smooth floor or large table). It should not contain any obstacles within the full FoV of the sensors, extending at least *1 meter*. This ensures that the resulting reference is unobstructed.

C Wheel Speed Relations

During testing, motor commands were issued in terms of a wheel speed parameter, which by itself does not directly correspond to physical velocity. To interpret this parameter meaningfully, it can be calculated using the wheel diameter and gear reduction to an actual velocity. In addition, the parameter Y is introduced to represent the torque exerted by the actuators, measured in mNm. Real-world velocity was calculated and Actuator Torque shown within the `RelBotDemo.cpp` script. This is located on the git repository at:

`src/asai-addon-pkg/Xenomai/src/RelBotDemo.cpp`

The relationship between wheel speed, real-world velocity, and actuator torque is summarised in Table C.1.

Wheel Speed	Real Speed (m/s)	Y1 (mNm)	Y2 (mNm)	Y1 (Nm)	Y2 (Nm)
0.5	0.025	80	70	0.080	0.070
1.0	0.050	107	97	0.107	0.097
1.5	0.075	134	124	0.134	0.124
2.0	0.100	161	151	0.161	0.151
2.5	0.125	188	178	0.188	0.178
3.0	0.150	215	205	0.215	0.205
3.5	0.175	242	232	0.242	0.232
4.0	0.200	269	259	0.269	0.259
4.5	0.225	296	286	0.296	0.286
5.0	0.250	323	313	0.323	0.313
5.5	0.275	350	340	0.350	0.340
6.0	0.300	377	367	0.377	0.367
6.5	0.325	404	394	0.404	0.394
7.0	0.350	431	421	0.431	0.421
7.5	0.375	458	448	0.458	0.448
8.0	0.400	485	475	0.485	0.475
8.5	0.425	512	502	0.512	0.502
9.0	0.450	539	529	0.539	0.529
9.5	0.475	566	556	0.566	0.556
10.0	0.500	593	583	0.593	0.583
10.5	0.525	620	610	0.620	0.610
11.0	0.550	647	637	0.647	0.637
11.5	0.575	674	664	0.674	0.664
12.0	0.600	701	691	0.701	0.691
12.5	0.625	728	718	0.728	0.718
13.0	0.650	755	745	0.755	0.745

Table C.1: Measured torque values Y1 and Y2 as a function of wheel speed and real speed

D Sensor Mounting Angle Evaluation

Chapter 4.4 describes the process of selecting the new sensor placement. It uses angled down press-fit mounts. To calculate the angle at which it should be mounted, measurements were used. This appendix explains the setup, method, and results used to determine the correction angle.

D.1 Setup

The RELbot was placed on a flat, uniform surface with no obstacles within a 2-meter range in front. Ambient lighting and other noise conditions were kept constant to maintain Gaussian noise. Sensors were mounted straight on the RELbot floor for the initial tests, the setup in Hofmann, 2024. A prior iteration of the sensor mounts was placed on a 3.5-degree correction angle, mounted on the RELbot floor.

D.2 Method

For each mounting configuration, the sensor collected 100 measurement frames (8x16 measurement zones). The median value for each zone was computed to account for measurement noise and outliers. The testing software used is the calibration software that can be found on the git repository:

`src/callibration_software/calibration_tof.py`

The result of this approach provided average row values used in Chapter 4.4. The raw measurements are presented here; a heatmap is applied on top of the measurements to improve readability.

D.3 Results

The initial configuration followed the mounting approach in Hofmann, 2024. The sensors faced straight forward at a relative 0-degree tilt. The resulting measurement heatmap in millimeters is shown in Figure D.1.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
421	326	0	0	0	317	0	0	446	0	0	0	0	0	450	432
285	279	273	269	268	275	282	292	290	280	276	270	268	279	286	290
193	198	200	220	225	210	200	196	192	195	201	218	224	211	198	192

Figure D.1: 0° mounting configuration measurement

The desired target range is 300mm. A prior iteration of the mounts was placed on the RELbot floor on a 3.5-degree tilt downwards. The measurement results in millimeters for this configuration are shown in Figure D.2.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
371	361	362	377	361	359	355	363	424	397	390	412	410	409	369	0
274	275	276	278	276	277	272	268	333	328	324	330	328	324	325	320
213	214	218	219	217	215	210	208	248	249	253	257	258	254	249	244

Figure D.2: −3.5° mounting configuration measurement

The latest iteration of the sensor mounts is placed on the RELbot roof, on a 12.3-degree tilt downwards. The measurement results in millimeters for this configuration are shown in Figure D.3

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	250	261	270	0	259	0	0	303	279	285	322	294	301	276	335
300	290	302	309	313	301	302	306	252	254	253	262	260	254	256	254
216	216	217	222	223	225	222	216	180	185	196	198	198	192	186	182
159	160	163	171	171	166	162	157	136	137	141	145	144	141	137	135

Figure D.3: −12.3° mounting configuration measurement

E Sensor Noise Tests

The goal of these tests is to map the measurement noise behavior. This appendix presents the results of two noise measurement experiments:

- **Noise over distance**: Analysing how measurement noise changes with target distance.
- **Noise with external interfaces**: Assessing the effect of lighting and human presence on measurement stability.

E.1 Noise over Distance

Analysing how measurement noise changes with target distance.

E.1.1 Setup

A straight vertical wall was used as the detection target. The RELbot was placed on the floor at a set distance from the wall. No other objects were between the wall and the RELbot during measurements. The RELbot was moved backwards in discrete steps of 0.5 m to test different distances. The setup is shown in Figure E.1.



Figure E.1: Test setup: noise over distance

E.1.2 Method

At each distance, the RELbot recorded 100 measurement frames (8x16 measurements per frame). The standard deviation was calculated per zone. The code used for this test can be found on the git repository:

```
src/demo_software/SensorDistributionTest.py
```

The results were visualised as heatmaps for the distance 0.5 m till 2.5 m.

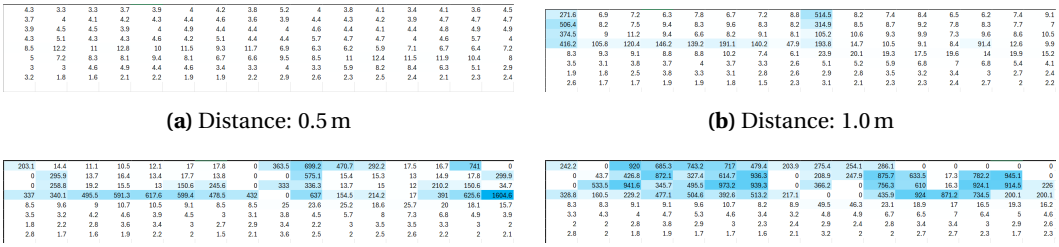
E.1.3 Results

Heatmaps for all distances are shown in Figure E.2.

The leftmost columns of measurements of both sensors at 1 m distance appear to see past the wooden wall. For the next steps, these columns are not included in the average standard deviation. The average standard deviation for each test distance is shown in Table E.1.

E.2 Noise with External Interfaces

Assessing the effect of lighting and human presence on measurement stability.



(a) Distance: 0.5 m

(b) Distance: 1.0 m

(c) Distance: 1.5 m

(d) Distance: 2.0 m

Figure E.2: Noise heatmaps for different wall distances (standard deviation per sensor zone)

Distance to the wall	Average standard deviation (mm)
0.5 m	5.23
1.0 m	14.73
1.5 m	109.03
2.0 m	206.30
2.5 m	190.80

Table E.1: Average standard deviation of sensor measurements at different distances from the wall.

E.2.1 Setup

The RELbot was placed 1 m from a wall on a table inside room CR3.420. Tests were performed with different lighting and human presence conditions. The setup is shown in Figure E.3.

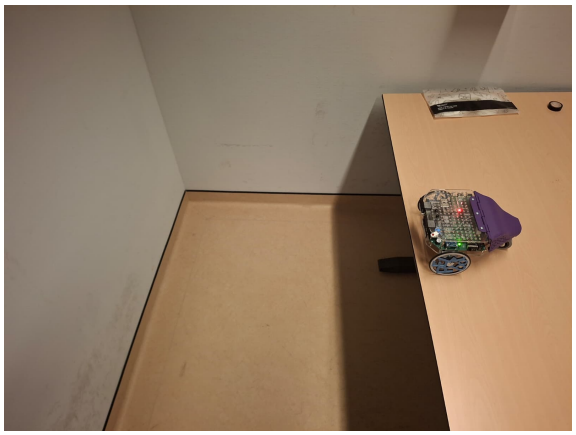


Figure E.3: Test setup: noise with external interfaces

E.2.2 Method

For each condition, the RELbot recorded 100 measurement frames. The standard deviation for each zone was calculated, similar to the distance-based tests. The same software was used.

Each condition will be systematically changed. The difference between the measurements was not significant enough for the heatmaps to be interesting. Results will only be shown in average values.

The following situations were tested:

- Lights on, with people (similar to the lab setting)
- Lights on, no people
- Lights off, with people
- Lights off, no people

E.2.3 Results

The average standard deviations for the four conditions are shown in Table E.2. It is important to mention that the "lights on, with people" setup has similar results to the 1 m measurement in Table E.1. This shows that the conditions between this case and the lab are similar.

Condition at 1 m in room CR3.420	**Average std (mm)**
Lights on, with people	14.13
Lights on, no people	14.02
Lights off, with people	11.89
Lights off, no people	11.16

Table E.2: Average standard deviation under different lighting and presence conditions at 1 m distance.

F Static Threshold Brake Test

The goal of this test is to estimate the stopping distance threshold, see Chapter 4.3. The test also improves the general understanding of the RELbot's braking performance. The results provide the basis for calculating the speed-dependent dynamic threshold.

F.1 Setup

The test setup consisted of a flat, smooth surface. A soft obstacle (a bag) reinforced with a bucket of sand behind it. A tape measure aligned along the approach path to record the remaining distance after braking. The setup is shown in Figure F.1.

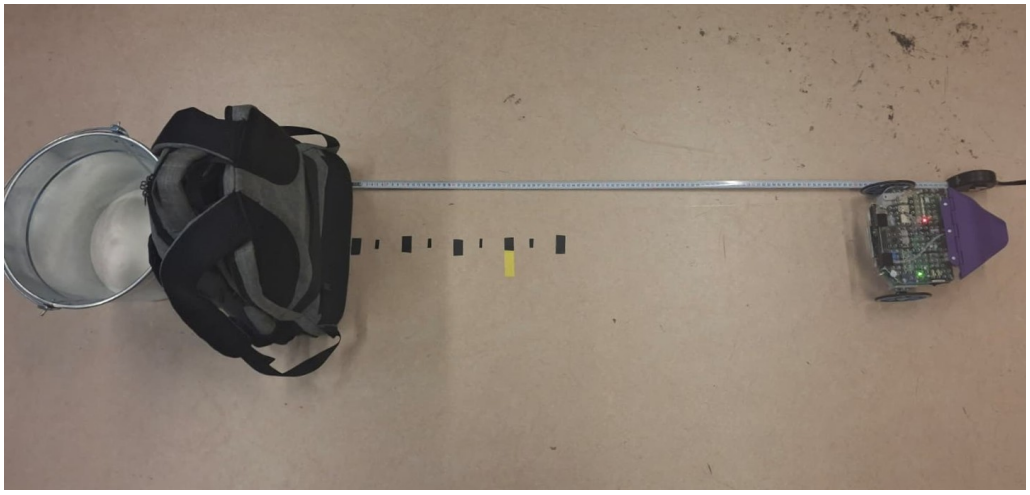


Figure F.1: Brake test setup with obstacle and distance measurement tape

F.2 Method

The RELbot was placed 1000 mm away from the obstacle and released at a predetermined velocity (v). A static threshold of 300 mm was used for speeds lower than 0.35 m/s, after which the stopping distance became greater than 300 mm and the static threshold was increased to 400 mm.

When the safety feature detected an object within the threshold, the RELbot performed an emergency stop, and the remaining distance to the obstacle was measured. This distance can be subtracted from the predetermined static threshold, resulting in a stopping distance.

To ensure reliable results, each test speed was repeated three times. A soft obstacle was intentionally used to prevent damage in the event of a failed stop.

F.3 Additional Data Collection

During testing, the spread of stopping distances noticeably increased at higher approaching speeds. To better map this behavior, additional data was collected at three representative speeds: 0.05 m/s, 0.25 m/s, and 0.5m m/s. For each speed, 30 repeated measurements were collected to assume a Gaussian spread. The data is used to analyse the overall variation.

F.4 Results

The measured stopping distances for all tested speeds are shown in Figure F.2, while the spread at the selected speeds is shown in Figure F.3.



Figure F.2: Stopping distance over approach speed

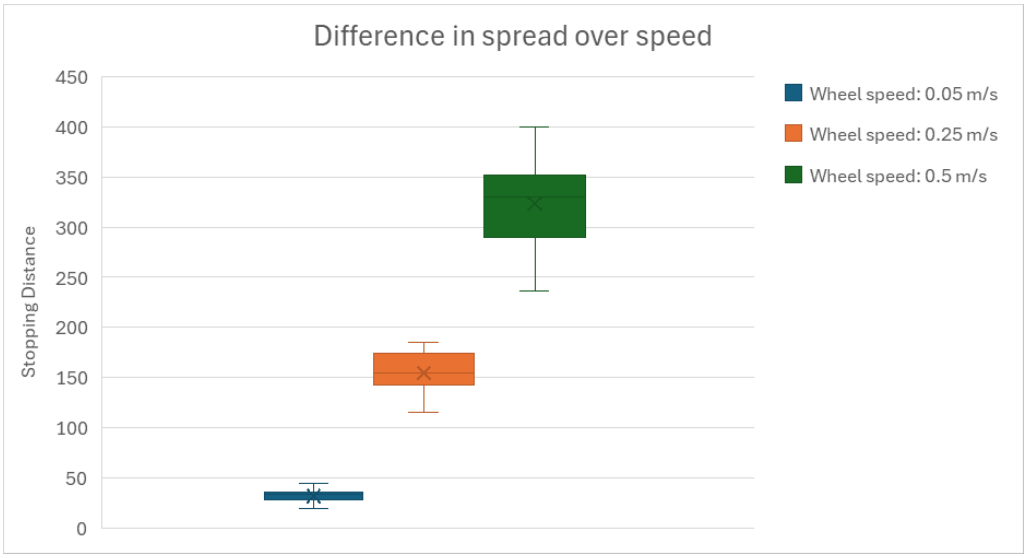


Figure F.3: Spread of stopping distances over approach speed

G Dynamic Threshold Brake Tests

The goal of these tests is to evaluate the overall performance of the safety feature. For this test to be successful, the system must detect the object or floor-edge, calculate the appropriate threshold, and brake before impact or dropping off. This appendix presents results from two brake tests:

- **Object detection and collision avoidance**
- **Floor-edge (drop-off) detection and avoidance**

Both tests validate the dynamic threshold logic under normal operating conditions.

G.1 Object Detection

G.1.1 Setup

Similar to the static threshold brake tests in Appendix F, the setup contains a soft obstacle (a bag) reinforced with a bucket of sand behind it. A tape measure is aligned along the approach path of the RELbot to measure the remaining distance after braking. The setup is shown in Figure G.1.

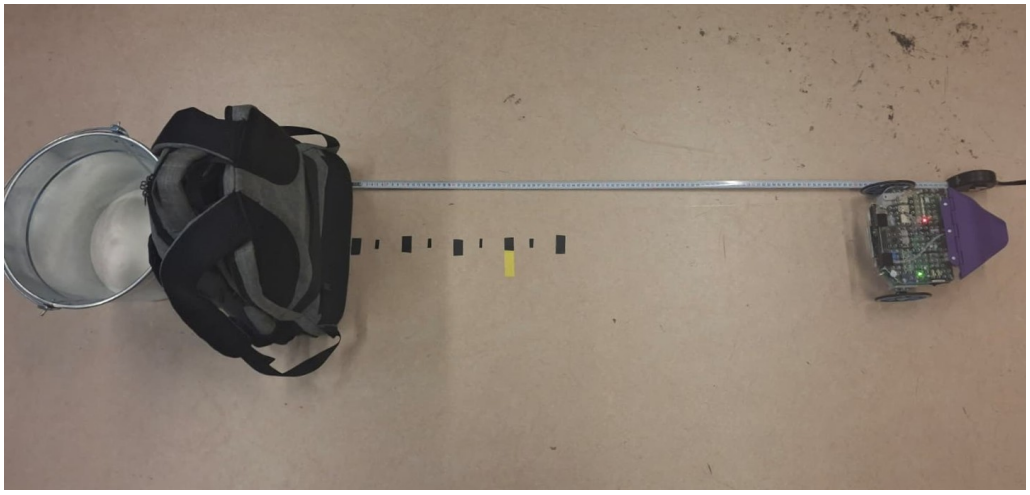


Figure G.1: Brake test setup for object detection

G.1.2 Method

The RELbot was placed on the floor 1 m away from the obstacle and driven towards it at different speeds. The system has to detect the obstacle, compare it to the calculated stopping threshold, and brake when necessary. The distance between the object and the closest point of the RELbot was recorded. For each speed, this test is repeated 5 times to account for outliers and ensure reliable data.

G.1.3 Results

Figure G.2 shows the remaining safety margin left between the RELbot and the object. Ideally, this stays constant, indicating an ideal behavior and threshold model. The left graph shows the minimum, median, and maximum values for each speed. The error bar indicates the variation between all attempts. From the 125 measurements conducted, only 5 hit the soft object and were marked as "failed".



(a) Stopping distance over speed **(b)** Error distribution

Figure G.2: Dynamic threshold brake test results: object detection

G.2 Floor Detection

G.2.1 Setup

The drop-off tests were conducted on a table with a clear edge. A soft chair was positioned at the ledge of the table to prevent damage in case of test failure. The setup is shown in Figure G.3.

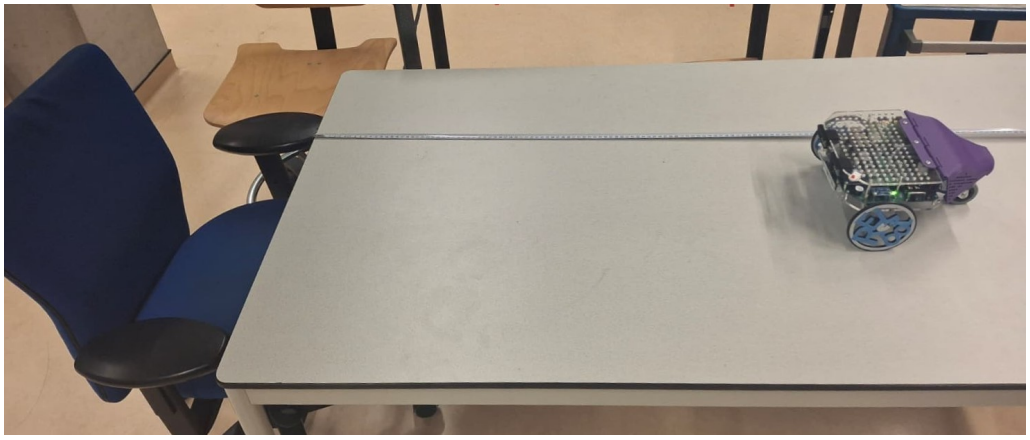


Figure G.3: Brake test setup for drop-off detection

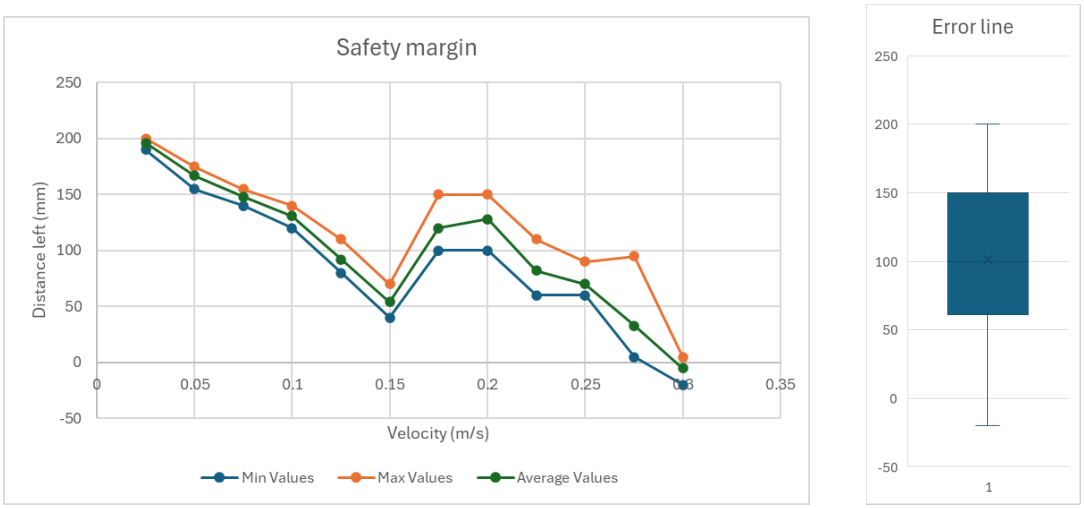
G.2.2 Method

Similar to the object detection brake tests, the RELbot was positioned 1 m away from the table-edge and released at various speeds. The system must detect the table-edge, calculate the stopping threshold dynamically, and perform the emergency brake in time. For each speed, the distance between the table edge and the front-most part of the RELbot was recorded. Negative values indicate an overhang.
This test is only done up to 0.3 m/s since the system started to fail after this speed. The experiment was repeated 5 times per set velocity.

G.2.3 Results

Figure G.4 shows the remaining safety margin. The left graph shows the minimum, maximum, and median values of the measurement as a function of velocity. The right graph shows the

overall value of this safety margin and that distribution. It is interesting to mention that the dynamic threshold calculation only has a resolution of two values, see Chapter 4.3. The switch between these two values can be observed after 0.15 m/s.



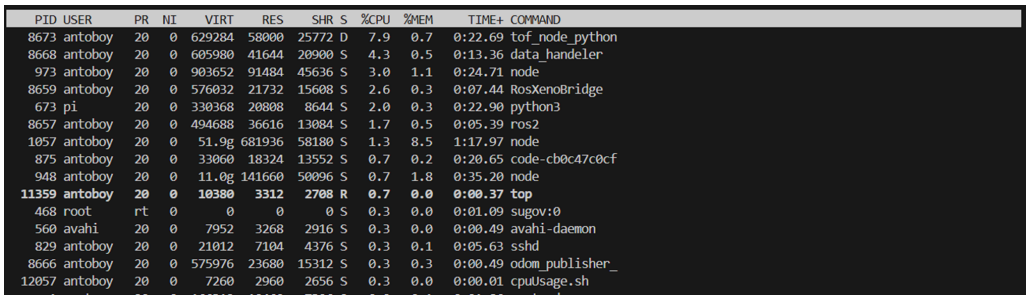
(a) Stopping distance over speed

(b) Error distribution

Figure G.4: Dynamic threshold brake test results: drop-off detection

H CPU usage

After using the `top` command when the system is running, the following data was observed. Some program names have changed after performing this test, but not the functionality, nor



PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
8673	antoboy	20	0	629284	58000	25772	D	7.9	0.7	0:22.69	tof_node_python
8668	antoboy	20	0	605980	41644	20900	S	4.3	0.5	0:13.36	data_handeler
973	antoboy	20	0	903652	91484	45636	S	3.0	1.1	0:24.71	node
8659	antoboy	20	0	576032	21732	15608	S	2.6	0.3	0:07.44	RosXenoBridge
673	pi	20	0	330368	20808	8644	S	2.0	0.3	0:22.90	python3
8657	antoboy	20	0	494688	36616	13084	S	1.7	0.5	0:05.39	ros2
1057	antoboy	20	0	51.9g	681936	58180	S	1.3	8.5	1:17.97	node
875	antoboy	20	0	33060	18324	13552	S	0.7	0.2	0:20.65	code-cb0c47c0cf
948	antoboy	20	0	11.0g	141660	50096	S	0.7	1.8	0:35.20	node
11359	**antoboy**	**20**	**0**	**10380**	**3312**	**2708**	**R**	**0.7**	**0.0**	**0:00.37**	**top**
468	root	rt	0	0	0	0	S	0.3	0.0	0:01.09	sugov:0
560	avahi	20	0	7952	3268	2916	S	0.3	0.0	0:00.49	avahi-daemon
829	antoboy	20	0	21012	7104	4376	S	0.3	0.1	0:05.63	sshd
8666	antoboy	20	0	575976	23680	15312	S	0.3	0.3	0:00.49	odom_publisher_
12057	antoboy	20	0	7260	2960	2656	S	0.3	0.0	0:00.01	cpuUsage.sh

Figure H.1: CPU usages of the system

the CPU usage. The sum of all CPU usage by the `tof_vl53l7cx_x2` package is 12.2%. These include the tof_node_python and `data_handeler`.

I Appendix: Survey

I.1 Analysis

To gain a perspective of opinions from the stakeholders, field research is conducted. This is researched in multiple ways. The technicians were represented by Gijs van Oort. He was interviewed about his opinions on the RELbot system and what they thought the biggest flaw was. Students were given a survey; see Appendix I for the raw results. This survey was given to over 100 students with RELbot experiences, from which 11 gave a detailed answer. For both parties, questions were non-solution-oriented to identify their desires and issues with the current RELbot configuration. The results are summarized in the next two sections, first desires then issues.

I.1.1 Stakeholder Desires

When asked about how to improve the system, the technician was open-minded and spoke very solution-oriented. He did not specify any direct desires that could be implemented. This seems logical since he implemented the system and is familiar with its design. Students, however, spoke about how they found the RELbot system slightly complicated. They spoke about how the user manual was unclear and the troubles they had when trying to integrate their prepared code. They desire a more Intuitive Interface and a user manual.
When delving deeper into the topic and asking for their opinion about the introduction of a safety layer, both the students and the technician were concerned about the interference of their software. They feared that a safety layer might cancel their software when it was operating as expected. They both desire as much freedom within the system as possible.
Lastly, to add to the previous statement. It can be stated that stakeholders want to know when their software does something unexpected and when the safety feature is taking over. They desire a predictable system.

To summarize, stakeholders desire a *predictable* and *intuitive* system with as much *freedom* as possible.

I.1.2 Stakeholder Issues

Below is a list of the mentioned stakeholder issues within the field research. It is also hypothesized what caused these issues and how they could be solved. Most of the issues the students had with the system could also have been their code and should be fixable without changing the RELbot hardware or XRF2 framework. These problems are still interesting to mention; however, they generate an image of problems they experience, where an experienced user might say "That is easy".
Disclaimer: The mentioned "likely cause" is only theoretical and might not contain the true problem from which these issues occurred.

1. **Upside-down camera feed**
   (a) **Likely cause:** Incorrect camera settings and/or mounted upside down
   Fix: Change camera settings
2. **Broken Camera**
   (a) **Likely cause:** Fell from a significant height ($\geq 10cm$).
   Fix: Surface Detection + Emergency Brake
   (b) **Likely cause:** Loose connection through oscillations
   Fix: Check for alternating patterns on the encoders and raise an error when this occurs.

3. **Difficulty when setting up the RELbot**
   (a) **Likely cause:** Complicated system with too little explanation about its functionality.
   Fix: A user manual
4. **Lag and frozen connection to the RELbot**
   (a) **Likely cause:** Limited Bandwidth on the RAM network.
   Fix: Upgrading the Bandwidth.
   Fix: Downsizing group lab sizes.
5. **Memory leaks/Corrupted Files**
   (a) **Likely cause:** Incorrectly shutting down the Raspberry Pi.
   Fix: Use the "sudo shutdown now" command and inform students of this.
   (b) **Likely cause:** Not releasing frame buffers
   Fix: controlling CPU temperature and raising errors when the temperature gets too high.
6. **Hard to prepare code for the system**
   (a) **Likely cause:** Complicated cyberphysical setup and no testing environment for Raspberry Pi. Code is usually prepared on Windows and then transferred to Linux.
   Fix: Create a Linux-based development environment
7. **Motors nonfunctional**
   (a) **Likely cause:** Addressing the motors wrong or a bug in the loop controller.
   Fix: Fix the bug
   (b) **Likely cause:** Disconnected through vibrations.
   Fix: Check for alternating patterns on the encoders and raise an error when this occurs.
   (c) **Likely cause:** Broken through use.
   Fix: Replace part.
8. **Motors Jitter at start up**
   (a) **Likely cause:** Uninitialized PWM outputs
   Fix: Introducing a pull-down resistor to give an initial PWM signal of zero (Electronics Tutorials, 2024).
   (b) **Likely cause:** Wrongly implemented Loop Controller
   Fix: Fix the bug in the loop controller.
9. **Broken wheels**
   (a) **Likely cause:** Aggressive handling of the RELbot when picking it out of its box.
   Fix: Making the wheel basis shorter, so it fits better in the box.

I.2 Raw results

So if you have received this form, it is assumed you are working or have worked with the RELbot in a course. If you are not familiar with the name, these are the robots you have programmed in the (Advanced) Software Development for Robotics course. For my Academic Skills project, I am trying to improve the framework of these robots and make them more foolproof. I am therefore interested in your previous experience working with these machines.

What course did you use the RELbot with?

1. Advanced Software Development for Robotics, Acedemic Skills Project
2. Advanced Software Development for Robotics
3. Advanced Software Development for Robotics
4. Advanced Software Development for Robotics
5. Advanced Software Development for Robotics
6. Software Development for Robotics
7. Advanced Software Development for Robotics
8. Advanced Software Development for Robotics

9. Advanced Software Development for Robotics
10. Advanced Software Development for Robotics
11. Advanced Software Development for Robotics

How would you rate the difficulty of setting up the RELbot?

1. 4
2. 5
3. 2
4. 2
5. 2
6. 3
7. 2
8. 1
9. 3
10. 2
11. 2

Were there any specific issues when trying to connect to the RELbot?

1. No clue how to connect to the system. The SSH connection was hard to set up.
2.
3.
4. At the first couple of sessions, the RAM connection did not work properly, which had us connecting to the RELbot directly.
5.
6. Camera feed was upside down, TA's said it was a common problem that could not be resolved easily.
7. bad manual
8. Not an issue of the RELbot itself, but when there are a lot of people in the lab, the bandwidth is quite limited. Sometimes it even completely froze and disconnected.
9.
10. getting the ssh working via visual studio code took a lil bit of troubleshooting
11.

How did you get your code on the RELbot?

1.
2. Using github
3. Copy and paste it through SSH
4. using SSH connection with Visual Studio Code
5. Copy and paste it through SSH
6. Using github
7. Copy and paste it through SSH
8. SSH copy paste but with GUI (WinSCP)
9. Using github
10. Copy and paste it through SSH
11. Copy and paste it through SSH

How well did your preparations translate to the real system?

1. 3
2. 2
3. 4
4. 3
5. 1
6. 3

7. 4
8. 4
9. 3
10. 4
11. 3

What interface did you use to code?

1. VS code with remote - SSH extension
2. VS code with remote - SSH extension
3. VS code with remote - SSH extension
4. VS code with remote - SSH extension
5. VS code with remote - SSH extension
6. VS code with remote - SSH extension
7. VS code with remote - SSH extension
8. VS code with remote - SSH extension
9. VS code with remote - SSH extension
10. both linux terminal and vs code with ssh
11. VS code with remote - SSH extension

How would you rate this experience?

1. 4
2. 5
3. 5
4. 4
5. 4
6. 4
7. 5
8. 4
9. 4
10. 5
11. 4

Have you experienced unexpected behavior? If so, what was this?

1. The motor djittered due to an unpropper loop controller.
2.
3. At some point we had camera failure problems, and when the camera failed the entire RELbot froze.
4. Not really
5. 1) FPGA failure on first power-up. 2) Memory leaks if the processes are not ended properly. 3) On one RELbot the motors were not moving even though the controller's output was 100+. Worked on another one.
6.
7. no
8. Not really. If there's overshoot it's our own controller that is to blame.
9.
10. motor jiggle at controller startup, constant flipping and unflipping of the camera (our own fault). At some point, we also broke the RELbot and it did not want to startup again.
11.

Do you think the RELbot system should prevent certain user actions automatically? If so, which?

1. Yes, these high frequency oscillations should be raising errors.
2.

3.
4. I think it was quite nice that the system was not too limited in what you could do.
5. No idea
6.
7.
8. No. User should be in full control. I think limiting certain actions that could be dangerous to the RELbot itself only complicates troubleshooting for students.
9.
10. I think the magic of Linux is that everything is possible...but changing that might not be smart.
11.

What features would make your RELbot experience significantly better?

1. A debugging platform enabling easier testing of the code, especially remotely. Maybe a replica using a digital box or second Raspberry Pi.
2.
3.
4. The main issue was the project manual. Lots of missing or incorrect information.
5. Simulation!! We need to be able to work offline with an actual simulator. URDF, Gazebo, RViz, etc.
6.
7.
8. Better FOV on the camera. Bracket for fisheye lens.
9. Better and accurate documentation of the model and system of the RELbot.
10. A graphical user interface for easier programming, rather than through SSH.
11.

J Matlab Threshold Estimation

```
% Data (from excel) I use the maximum values for a greater
   safety margin
y = [38; 51; 67; 118; 119; 125; 107; 127; 154; 252; 247;
   260; 298; 261; 322; 274; 188; 335; 322; 400; 400; 369;
   400;];

% x is just index use whatever you use as input, for my this
   is the Y factor, see the excel sheet for translations to
   wheel speeds
x = (75:27:(75 + (length(y)-1)*27))';

% Fit y = c1 * x^2 + c2 using linear regression
X = [x.^2, ones(size(x))]; % Design matrix [x^2, 1]
coeffs = X \ y;            % Least-squares solution

c1 = coeffs(1);
c2 = coeffs(2);

% Display coefficients
fprintf('Fitted model: y = %.4f * x^2 + %.4f\n', c1, c2);

% Plot original data and fitted curve
x_fit = linspace(min(x), max(x), 200)';
y_fit = c1 * x_fit.^2 + c2;

figure;
plot(x, y, 'bo', 'MarkerFaceColor', 'b'); hold on;
plot(x_fit, y_fit, 'r-', 'LineWidth', 2);
xlabel('Index (x)');
ylabel('Value (y)');
title('Fit: y = c_1 x^2 + c_2');
legend('Data', 'Quadratic Fit');
grid on;
```

J.1 Result

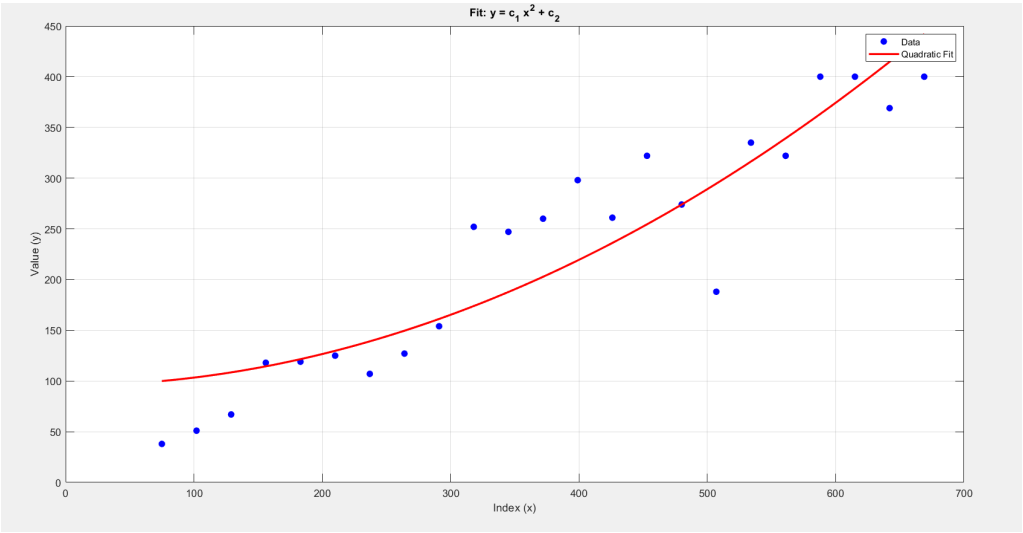


Figure J.1: Threshold Estimation

References

Broenink, Jan (Apr. 2025). *SWdevTools*. https://cloud.ram.eemcs.utwente.nl/index.php/s/RELbotDownloads?dir=undefined&openfile=6863373. Accessed: 2025-05-02.

Electronics Tutorials (2024). *Pull-up Resistor*. Accessed: 2025-04-30. URL: https://www.electronics-tutorials.ws/logic/pull-up-resistor.html.

Hofmann, Moritz (2024). *Development of a Collision-Avoidance Feature for the RELbot*. URL: https://cloud.ram.eemcs.utwente.nl/index.php/s/Q3T6RQ5Y6QkHd4o.

Oort, Gijs van (Apr. 2025). *RELbot User Manual*. https://cloud.ram.eemcs.utwente.nl/index.php/s/RELbotDownloads?dir=undefined&openfile=6863373. Accessed: 2025-05-02.

Raoudi, Ilyas (2024). 'ROS2-Xenomai4 Real-Time Framework on Raspberry Pi'. MA thesis. University of Twente. URL: https://cloud.ram.eemcs.utwente.nl/index.php/s/g6SijsJ4P8igszo.