FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Cockpit for high-level teleoperation, monitoring and task assignment to a set of mobile robots in internal logistics applications

Diogo Torres Correia

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: António Paulo Gomes Mendes Moreira

August 1, 2022

Resumo

A automatização de veículos para uso em operações de logística é uma realidade atual e cuja necessidade tem vindo a aumentar. Para além de reduzir o número de lesões e acidentes presentes muitas vezes neste contexto, abre também a porta a pessoas que sejam incapacitadas ou necessitem de estar isoladas uma vez que conseguem controlar vários veículos a partir de um único local.

O controlo presente nos dias de hoje deixa ao encargo dos robôs a escolha da tarefa a realizar, o que nem sempre leva à melhor solução já que se trata de um problema de grande complexidade. Assim, é necessária a intervenção de um operador para que possa alocar as tarefas aos robôs e intervir quando estes não conseguem tomar decisões autonomamente.

Este trabalho apresenta uma solução para este problema ao dar ao operador um sistema de teleoperação de alto nível no qual o operador pode controlar vários veículos no modo autónomo ou remoto e ainda ter informação relativa ao estado dos mesmos bem como supervisionar estes veículos através de uma interface. Para a testagem deste sistema foi necessário adaptar uma empilhadora já automatizada no que toca à navegação autónoma e ao controlo remoto. Assim, após as adaptações feitas a nível de software e hardware, foi possível validar as funcionalidades implementadas no que toca ao controlo remoto e à autonomia da empilhadora na tarefa de retirar e colocar uma palete num dado lugar.

Por fim, esta dissertação sugere algum trabalho futuro para melhorar este sistema e aproximar a sua aplicação na indústria.

Abstract

Automating vehicles for logistics operations is a current reality with increasing demand. It not only reduces the number of injuries and accidents that occur in this context, but it also allows people who are disabled or need to be isolated to control multiple vehicles from a single location.

Today's control leaves it up to the robots to choose the task, which does not always result in the best solution since it is a highly complex problem. As a result, the solution to this problem requires an operator who can assign tasks to robots and intervene when they cannot make decisions autonomously.

This work proposes a solution by providing the operator with a high-level teleoperation system through which the operator can control several vehicles in autonomous or remote mode, obtain status information and supervise these vehicles via an interface. There was an already automated forklift with remote control and autonomous navigation features but needed some adaptations to be used in a high-level teleoperation system. As a result of the software and hardware modifications, it was possible to test it in both modes of operation, with great results obtained for the operator's performance in remote control and the forklift's autonomy in removing and placing a pallet in a given location.

Finally, this dissertation suggests future work to improve this system and bring its application in the industry closer.

Agradecimentos

No decorrer desta dissertação houveram várias pessoas que apoiaram em todas as dificuldades que senti e às quais gostaria de agradecer.

Em primeiro lugar, quero agradecer aos meus orientadores António Paulo Moreira e Manuel Silva, por toda a ajuda, paciência, acompanhamento e disponibilidade que me dedicaram para que pudesse ter sucesso na execução desta dissertação. Sem a experiência e conhecimentos transmitidos por eles, o caminho percorrido durante a execução desta dissertação seria bem mais difícil.

Também quero agradecer àqueles que, embora não sendo meus orientadores, se preocuparam com o sucesso na realização desta dissertação e estiveram disponíveis para me ajudar sempre que precisei, nomeadamente Héber Sobreira, Paulo Rebelo, Rui Coutinho e André Baltazar.

Um obrigado também aos meus colegas de laboratório, com os quais tive o prazer de partilhar vários momentos e que me acolheram desde o primeiro dia e com os quais aprendi sobre diversas áreas e com quem pude contar sempre que precisei de ajuda. Foi este ambiente de entre-ajuda e boa disposição que fizeram com que os dias mais exigentes fossem mais toleráveis e enriquecedores.

Também não posso deixar de agradecer aos meus amigos, com quem tive a oportunidade de viver todo o tipo de momentos, desde as grandes alegrias aos momentos mais desafiantes e que também proporcionaram experiências memoráveis para além das vividos no ambiente académico.

Quero agradecer também à minha família, que me acompanhou durante todo este percurso, em especial à minha mãe, por todos os sacrfícios que fez para me permitir chegar mais longe e a quem devo grande parte daquilo que sou hoje.

Por fim quero agradecer a todos aqueles que, ao longo destes anos, contribuiram de uma ou de outra forma para o meu desenvolvimento académico e pessoal.

Diogo Correia

"Each failure brings you one step closer to success"

Zig Ziglar

Contents

Abbreviations xvi

1 Introduction 1
- 1.1 Context . . . 1
- 1.2 Goals . . . 1
- 1.3 Motivation . . . 1
- 1.4 Dissertation Structure . . . 2

2 Background 3
- 2.1 Introduction . . . 3
- 2.2 Mobile Robots . . . 3
  - 2.2.1 Perception . . . 3
  - 2.2.2 Localization . . . 4
  - 2.2.3 Navigation . . . 6
  - 2.2.4 Path planning . . . 6
- 2.3 Teleoperation . . . 8
- 2.4 Image Transmission . . . 10
- 2.5 High-level Command and Task Assignment to the Vehicles . . . 11

3 State of the Art 13
- 3.1 Improvements on Teleoperation . . . 13
- 3.2 Applications in Logistics Operations . . . 17
- 3.3 Surveys on Teleoperation . . . 20
- 3.4 Discussion . . . 20

4 Implementation 23
- 4.1 System Overview . . . 23
- 4.2 Forklift Adaptation . . . 24
  - 4.2.1 Hardware Adaptation . . . 25
  - 4.2.2 Software Adaptation . . . 33
- 4.3 Remote Operation Center . . . 42
  - 4.3.1 Autonomous Operation . . . 42
  - 4.3.2 Remote Control . . . 44
  - 4.3.3 User Interface . . . 45

5 Results 53
- 5.1 User Interface . . . 53
- 5.2 Remote Control . . . 53
- 5.3 Autonomous Pick and Place Operation . . . 57

5.3.1 Pallet Detection 58
5.3.2 Autonomous Navigation 58
5.4 Discussion 59

6 Conclusion **61**
6.1 Outcomes 61
6.2 Challenges 61
6.3 Conclusions and Future Work 62

References **65**

List of Figures

3.1 Software Architecture [1] . . . 14
3.2 Perspective view of the mixed reality environment shown within HMD to the human operator [2] . . . 14
3.3 Telepresence interface [3] . . . 15
3.4 The proposed Human-Robot Interaction (HRI) method, which is built upon real-time robotic mapping and online Virtual Reality (VR) visualization, where the robot and remote control station communicate with each other by a wireless network. [4] . . . 16
3.5 Two Augmented Reality teleoperation interfaces designed in this work: Left - Realtime Virtual Surrogate (RVS), Right - Waypoint Virtual Surrogate (WVS) [5] 17
3.6 Hardware architecture [6] . . . 18
3.7 System architecture of forklift teleoperation system, which consists of a laptop, an Arduino microcontroller, four cameras, a PS3 controller. [7] . . . 19
3.8 Classification of teleoperation enhancement techniques [8] . . . 21

4.1 System Block Diagram . . . 24
4.2 Laser scanners placement . . . 25
4.3 Forklift's hardware placement. The hardware is as follows: 1 - laser scanners; 2 - emergency buttons; 3 - fork encoder; 4 - camera; 5 - pallet sensor; 6 - wireless emergency button; 7 - access point and network switch . . . 26
4.4 Fork Encoder . . . 26
4.5 Data transmission of the fork encoder (image from the manual available on LIKA's website [9]) . . . 27
4.6 Fork's encoder circuit schematic . . . 28
4.7 Fork's encoder circuit implemented on a Veroboard . . . 28
4.8 Raspberry Pi Camera . . . 29
4.9 Camera placement . . . 29
4.10 Pallet sensor web interface . . . 31
4.11 Pallet sensor placement . . . 31
4.12 Remote emergency system: emitter (button) - left and receiver - right . . . 32
4.13 Overview of the software architecture . . . 33
4.14 Software already developed in the forklift's PC . . . 35
4.15 Software implemented in the forklift's PC . . . 37
4.16 ROS node selection given the forklift's intended mode of operation . . . 38
4.17 Pick order state machine . . . 40
4.18 Place order state machine . . . 41
4.19 Representation of the software implemented in the forklift's Arduino . . . 43
4.20 Racing wheel and pedals and its functions . . . 45

4.21 Joystick and its functions 46
4.22 Configurations page of the user interface 47
4.23 Task assignment page of the user interface 48
4.24 Remote control page of the user interface 49
4.25 Load selection page of the user interface 50

5.1 Delays in the image transmission from the camera to the interface 54
5.2 Delays in the commands transmission from the forklift's PC to the operator's PC 55
5.3 Delays in between the click and the forklift's unlocking to start to move 55
5.4 Operator's responses to the NASA TLX questionnaire on the workload felt during the remote control experience 56
5.5 Average of the operator's responses to the NASA TLX questionnaire on the workload felt during the remote control experience 57

List of Tables

3.1 Overview of the technologies used in the previous works 22

4.1 Contents of the message used to send the operator commands in remote control mode 36
4.2 Contents in the feedback message sent to the interface 42

5.1 NASA TLX quesionnaire to be filled by the operators after performing a pick and place task using the remote control functionality 56

Table of Acronyms

AVC	Advanced Video Coding
ACO	Ant Colony Optimization
ARMD	Augmented Reality Head-Mounted Display
AGV	Automatic Guided Vehicles
AuRA	Autonomous Robot Architecture
CAN	Controller Area Network
DWA	Dynamic Window Approach
ERP	Enterprise Resource Planning
EKF	Extended Kalman Filter
FOV	Field of View
GA	Genetic Algorithms
GNSS	Global Navigation Satellite Systems
HDMI	High Definition Multimedia Interface
HEVC	High Efficiency Video Coding
IMU	Inertial Measurement Unit
IEEE	Institute of Electrical and Electronics Engineers
JSON	JavaScript Object Notation
JPEG	Joint Photographic Experts Group
KF	Kalman Filter
LiDAR	Light Detection And Ranging
LTE	Long Term Evolution
MPC	Model Predictive Control
MJPEG	Motion JPEG
PC	Portable Computer
PRM	Probabilistic Roadmap
PLC	Programmable Logic Controller
RRT	Rapid Exploring Random Trees
RTSP	Real Time Streaming Protocol
RTP	Real-time Transport Protocol
ROS	Robot Operating System
SLAM	Simultaneous Localization And Mapping
SCADA	Supervisory Control And Data Acquisition
TEA	Time Enhanced A*
TCP	Transmission Control Protocol
UDP	User Data Protocol
WiFi	Wireless Fidelity
WLAN	Wireless Local Area Network
YAML	Yet Another Markup Language

Chapter 1

Introduction

1.1 Context

The teleoperation of mobile robots is highly relevant when performing systematic and repetitive tasks or paths that often cause fatigue to the operators performing them manually. A typical case where this happens is in logistics operations when moving goods in a warehouse, using forklifts or pallet jacks. The automation of these vehicles appears then as a growing necessity nowadays since, in the face of the COVID-19 pandemic, the volume of orders has increased considerably, and it is not always possible to hire a sufficient number of people to deal with this problem.

Thus, a high-level teleoperation system capable of supervising these robots comes as a relevant solution by allowing their control using the remote or autonomous modes.

1.2 Goals

Some requirements should be satisfied for the correct operation of this high-level teleoperation system. There should be at least one operator to supervise these autonomous robots' behavior and task assignment.

This system should allow the operator to:

- visualize the warehouse and the vehicles through strategically positioned cameras using an interface;
- create a pick and place task and assign it to a vehicle to perform it autonomously;
- assume the control of the forklift when required, using the remote control mode;
- switch the forklift's control between remote and autonomous.

1.3 Motivation

This proposal appears as a solution that allows:

- the control of vehicles in potentially dangerous environments for humans from a safe and comfortable place for the operator;
- the reduction of the number of injuries associated with repetitive handling of this equipment;
- the reduction of the number of accidents through a system to avoid collisions and their severity since there are fewer people physically present in this environment;
- the simultaneous control of vehicles by remote operators covering even people who may have motor disabilities or need to be isolated.

Aside from that, in the current teleoperation systems, the task assignment is automatic and does not always provide the expected performance due to being a problem with multiple constraints. As a result, an operator should assign these robots tasks and intervene when required.

1.4 Dissertation Structure

Besides the introduction, this dissertation contains five more chapters. In chapter 2, it is given an understanding of how autonomous mobile robots work and an insight in the methods used in teleoperation and image transmission systems. In chapter 3, a state of the art of some solutions in high level teleoperation is presented. Chapter 4 describes the adaptations made to the forklift to fit it in the developed high-level-teleoperation system, both software and hardware, and how to use it. Some tests on the main parts of this system were performed and chapter 5 presents the obtained results. Finally in chapter 6, the outcomes of this work are presented as well some of the future work that can be done to improve this high-level teleoperation system.

Chapter 2

Background

2.1 Introduction

The teleoperation of a mobile robot involves multiple systems that must all work together to achieve optimal performance. This chapter demonstrates some of the most widely used approaches used in this type of system. As a result, the section on mobile robots examines the modules that contribute to their autonomy, such as perception, localization, navigation, and path planning. Following that are the teleoperation and image transmission sections, which describe some of the most common methods used, and, finally, the normal mode of operation of mobile robots used in logistics at this time, which leaves it up to the robot to decide which task to perform, thus differentiating this work.

2.2 Mobile Robots

Mobile robots have played a significant role in automating repetitive tasks often present in logistics processes, such as loading/unloading and moving loads from one point to another. These robots can move autonomously, making decisions regarding the trajectory they should follow based both on the destination they have to reach and on the information they get from their surroundings through their sensors and a previous map. This process can be divided into five main sub-processes such as: perception, localization, navigation, obstacle avoidance and path planning.

2.2.1 Perception

As presented in [10], there are multiple sensors that can be used to provide information about various aspects related to the robot's state (proprioceptive) and the environment around him (exteroceptive). The former includes some of the most used sensors such as:

- Encoders - measures the position or velocity of the robot's wheels
- Gyroscopes - measures the orientation of the robot in relation to a fixed frame

- Accelerometer - measures the acceleration

There are also sensors that combine some of the sensors stated above, such as the Inertial Measure Unit (IMU) that uses gyroscopes, magnetometers and accelerometers. Whereas in the latter category the sensors used vary according to the type of environment as some work better at outdoors (like Global Navigation Satellite Systems (GNSS)) and others are more suitable for indoors. Some of the sensors that work well indoors are:

- Laser range finders - measures the distance based on the phase difference between the emitted and reflected wave
- Artificial Vision - captures environment information in an image (2D) or in a point cloud (3D)

2.2.2 Localization

The challenge of self-localization is of the utmost importance for the robot to know where it is and, thus, move autonomously. There are multiple approaches to this problem, some of which can be used simultaneously to achieve better performance [10].

One possible solution is localization by using natural or artificial landmarks. In this solution, the robot needs to know its initial position to look for landmarks in a limited range around it. If beacons are used, the robot must know the location of at least three beacons to use some methods to calculate its position: trilateration and triangulation. The former uses the distances to and between beacons whereas the latter measures the angles to the beacons to a fixed frame. There are some issues related to this method. It can only be accurate when it is near landmarks so it may lose itself if it fails to detect a landmark, requiring multiple beacons dispersed through the environment to prevent that.

Using odometry can also help solve this challenge by tracking the data from the wheels' encoders. This method can calculate not only the linear and angular velocities of each robot, but also the distance traveled by counting the number of impulses in each encoder and knowing the distance between each wheel. However, this metric can be affected by two types of errors: systematic and non-systematic. Differences in wheel diameters and inaccuracy in measuring the distance between wheels are examples of the former. Among the latter are floor irregularities, wheel slippage, and encoder resolution issues.

Another highly used methods are the Kalman Filter (KF) and the Extended Kalman Filter (EKF). These algorithms take into account the measurements made by every sensor used and their uncertainties to estimate a state, which is, in this case, robot's position and orientation, and update it every iteration. The difference between both algorithms is that the former cannot be used in nonlinear models whereas the latter can achieve this by using linear approximations of nonlinear functions. A more detailed description is presented in [11]. Some downsides of these algorithms are the low performance if the sensor's model cannot be approximated by a Gaussian function and that it may never recover once lost either by a high estimation error or by ambiguous situations as described in [10].

Within the probabilistic approaches, apart from the Kalman Filter, is the Markov localization algorithm explained in detail in [10]. This algorithm uses a 3D grid map where each cell represents the probability of the robot to be in a position and orientation given the value of the probability density function chosen. While it can succeed in the recovery problem, it comes with a high computational and memory cost since, for every iteration, all cells in the grid map are updated. Thus, it is highly impractical in environments with large dimensions and where robots' control cycles need to be short.

Monte Carlo Filter, most commonly called Particle Filter, is also a probabilistic approach to solve the localization problem. It represents the probability of the robot to have a certain pose by a set of particles that are updated based on the sensors' measurements and is described in more detail in [12]. Whereas it may be simpler to implement, it can have a high computational cost depending of the number of particles used.

Map matching-based algorithms also seem to have high performance in solving this problem. Essentially, they compare the measurements taken with a previously stored map, measuring the error between them and aiming to reduce it by minimizing a cost function taking this error into account. In [13] is made a comparison of three algorithms inside this category: Perfect Match, Iterative Closest Point, and Normal Distribution Transform. It showed that the former was more computationally efficient and had better performance. Furthermore, in [14] it is evidenced the superior performance compared to the particle filter in terms of time efficiency, with the Perfect Match algorithm being 4 or 10 times faster than a particle filter with 200 or 500 particles, respectively.

In the algorithms above, the robot requires a map to solve the self-localization problem by comparing it with the obtained measurements. When a robot does not have access to a map, it must create one. However, to achieve that, it needs to know its current position in the environment, which leads to the Simultaneous Localization And Mapping (SLAM) problem that is explained in detail in [12]. The two most used approaches to this problem are the EKF-SLAM and the Fast SLAM.

EKF-SLAM associates the robot's pose, with the features it detects, to a state. This way, both landmarks' and robot's positions are estimated and updated in each iteration. The uncertainty associated with the measurements made tends to increase through the mapping process, capable of achieving high values and thus, failing the mapping process. To prevent that, the robot needs to go to the initial position where it started mapping. By "closing the loop" the uncertainty reduces drastically as there can only be a limited set of points in the map that can lead back to the initial position. Apart from having high precision, it inherits the EKF problems being computationally expensive if multiple landmarks are used. Besides that, some ambiguity may be present as it may fail to differentiate between a new landmark and an already detected one. A more detailed description can be consulted in [12].

Another approach that comes to solve the SLAM problem is the Fast SLAM. This method combines in a way the Particle Filter and the EKF by using the former to solve the robot's localization problem and a set of the latter to track the landmark positions. This way, each particle

has the number of EKF correspondences to the landmarks to be detected. In a situation where the landmark association is ambiguous, it just accepts the one that has a higher probability. M. Montemerlo *et al.* describes this approach in detail in [15] whereas, in [16], an improved version of this method is also presented, revealing higher performances than the EKF.

2.2.3 Navigation

When it comes to navigation, the robot may use paths already defined in the environment, using magnetic or electrical wires placed on the floor, or only use the environment. Examples of the latter are laser navigation and natural navigation.

When using fixed paths, the robot's position is well defined, allowing for accurate positioning with a navigation that is reliable and robust to interferences. Aside from that, defining a path for the robot to follow is simple and inexpensive. While this strategy suits well for fixed paths, the same does not apply to dynamic environments, where they are sometimes complex and need to be updated, becoming expensive to replace the wire or the tape regularly [17].

Laser navigation is one of the most used types of navigation [17]. By sending rotating laser arrays in a 360-degree pattern, they can reach reflectors positioned at the same height as the navigation system. This system can provide accurate positioning at high speeds with a simple installation. However, apart from the complexity of the algorithms involved, the environment needs to be previously prepared by adding reflectors in strategic positions.

Another type of navigation that is becoming increasingly popular is natural navigation which uses the information from the robot's surroundings to define its path and position using SLAM strategies to track the environment. On the upside, it provides high flexibility of paths and cheap installation and maintenance. On the downside, it is difficult to achieve accurate navigation due to having uncertainties in the robot's position. Apart from that, it may not be very reliable when included in high dynamic scenarios as stated in [17].

2.2.4 Path planning

Having solved the previous problems, robots need to be able to plan a path to follow given a destination point and the information about its environment. Usually, the goal is to choose the path that leads to the shortest traveled distance. However, others can be used, like finding the fastest or the most energy-saving path. Some of the path planning methods are presented in [18], some developed in early stages: road map; cells decomposition; potential fields; and others which are becoming more popular like the random sampling.

Starting with the roadmap, there are two main algorithms: Visibility graph and Voronoi diagram. The former connects all obstacles' corners to the starting and ending points, as well as the obstacles themselves. It is thus guaranteed to obtain the shortest path by applying a graph search algorithm (A*, Dijkstra, etc.). However, this approach lacks in the proximity of the robot to obstacles, therefore, increasing the probability of crashing. On the other hand, the Voronoi diagram seems to solve this issue by creating a path equidistant to all obstacles thus being more secure.

Apart from that, this algorithm is more efficient by having a complexity of $nlogn$ vs n^2logn where n is the number of obstacles [18].

One of the most used methods is cell decomposition, which can be divided into two subcategories: exact and approximate. The cells are built based on either a grid or some simple geometric shapes, and connections are made between them and the starting and ending point. The exact method represents the environment by two types of cells that classify the space as empty or occupied, whereas the approximate method uses one more type of cell that classifies the space as partially occupied. One problem that arises with this type of method is that it may not find a path even if there is one because it is highly dependent on the size of the shape used as stated in [18]. The grid size used can either be fixed or adjusted to the state of the environment (quadtree decomposition). While the former provides a better trajectory, the latter is more memory efficient, making it more suitable for real-time scenarios.

The potential fields method leads the robot to its destination point using two fields, similar to magnetic fields: attractive created by the goal; and repulsive created by the obstacles. While it is more efficient than the other algorithms, it fails when these fields cancel each other in a zone different than the goal, thus, being stuck in that point [18]. A possible solution to this problem is using algorithms capable of escaping this zone (local minima) like the Simulated Annealing [19] or the Tabu Search [20] algorithms.

In the previous algorithms, as the dimensions of the environment increase, so does the map, leading to a high computational time needed to find a path. The random sampling method tries to solve this problem by using a probability distribution to sample nodes randomly, thus reducing the number of nodes to be visited. As presented in [18] there are two algorithm classes: multi-query and single-query, also explained in detail in [21]. An example of the former is Probabilistic Roadmap (PRM) which computes an accurate roadmap in a short amount of time and then samples the nodes according to the probability distribution, saving those nodes as milestones, linking them if there is a collision-free path between them. As [18] states, the performance of these types of algorithms is dependent on the probability distribution used, for example, a uniform distribution, may lead to bad results. The latter creates a roadmap on the fly by only sampling nodes that contain either the starting or the ending points. The roadmap consists of two trees, each one rooted in the initial and final point, that expand by randomly sampling new nodes and inserting them into the roadmap until there is a connection between both trees. One highly popular example is the Rapid Exploring Random Trees (RRT) that can provide fast paths although not optimal.

In an highly dynamic environment where multiple machines or people are present, it may be difficult to have an efficient planned path. Besides, when there are multiple AGV involved, some ambiguous situations can occur due to share of a resource (or routes) thus, leading to deadlocks. As some algorithms are not well suited for this type of environment, like A*, multiple researches were made to develop algorithms that could adapt the path in real time. Time Enhanced A* (TEA*) is an A* variant that takes time into account thus enabling to allocate resources in different time windows and avoid deadlocks. J. Santos *et al.* presents some results from this algorithm in which paths are assigned to multiple vehicles that may share the same resources [22]. It is also clear that

in some cases, an AGV should wait for another to move and, as a result, have a combined total path time execution lower than if the opposite occurs. Another variant is the Dynamic A* (D*). This family of algorithms allows for the robot to follow a path and update its path if necessary when new information is acquired like the dynamic obstacles.

Dynamic Window Approach (DWA) is a method for planning a route that takes vehicle kinematics into account. Paths are composed of arcs that lead to the next trajectory with the linear and angular velocities that the robot can achieve in the next cycle.Thus, knowing the current robot speed and the maximum accelerations it can achieve, allows for a filtering process in which every path that conducts to a possible collision is ignored. As a result, the best path can be chosen based on an objective function that prioritizes forward motion while maintaining a safe distance from obstacles with the desired final orientation. If the distance between each position and the goal is known, it can also be used to determine the best path to follow [10].

The number of path planning algorithms in the literature is vast, and multiple algorithms have been developed over the years, indicating that path planning is still a research area. In [23] it is made a comparison between some of the most used algorithms where the field D* stands out due to needing a significantly less computational time compared to the other algorithms from the D* family. Other popular algorithms that are among the most widely used are the nature-based algorithms, such as Genetic Algorithm (GA) or Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and Potential fields [24]. However, some of them, as the nature-based algorithms, are not recommended for real-time applications, lacking their performance by being stuck in local minima, as [23] states. According to [25], the Model Predictive Control (MPC) is preferred over the potential fields method as not only does it allow for a future position of the robot, but also is more optimal because it considers the robot's dynamics without too much computational cost associated. For a broader comparison of path planning algorithms, [26] presents an overview of these algorithms as well as their applications.

2.3 Teleoperation

Teleoperation is the process of remotely controlling something in a location other than the one to which it will move. It is not a new concept as it has long been used in navigation in hazardous or difficult-to-access scenarios, and efforts have been made to improve the remote control's performance. Within the teleoperation methods developed, direct teleoperation and supervisory teleoperation are within the most used [8]. In direct teleoperation, the vehicle relies on the commands transmitted by the operator to drive and perform decisions whereas, in supervisory teleoperation, the operator monitors the vehicle and aids in the decisions the robot has to make. Thus, the robot may have an obstacle avoidance system to aid the path following providing an extra level of security and autonomy. To remotely drive efficiently, the operator must have the closest possible feeling to driving the vehicle in person. Humans get more than 90% of their perception through vision, performing a significant role in a good performance in teleoperation. Sound can also help to provide an immersive experience to the operator as it better perceives not only the vehicle's

state, as sounds may sometimes reveal malfunctions, but also its environment. Some works take a step further by using virtual reality and providing operators with Head Mounted Displays (HMD). While it should increase the driving performance, it showed that if delays are evident between the head movements and the visual information, it may lead to motion sickness [27]. To provide visual perception, cameras are mounted in strategic positions such as the front, back, and sometimes the sides, allowing for a global view of the robot's surroundings. One closely related aspect of the use of cameras is its Field of View (FOV), which has been shown to have a significant impact on the operator performance [3]. The work [3] makes a comparison of the operator's performance when using cameras with different FOV lenses where a 180 degrees diagonal FOV and a 180 degrees horizontal FOV with pan and tilt, produces the best results and requires the least cognitive effort from the operators. Even though cameras can provide such vast amounts of information, sending that information takes time, and as the number of cameras increases, so does the amount of time required. Apart from that, even with cameras, there may be some blind spots. A more efficient way of perceiving the environment may be provided by using a camera and a Light Detection And Ranging (LiDAR) sensor simultaneously. When mounted in the front, the camera can provide the majority of the visual feedback required, while the LiDAR can provide the distance to the objects in the robot's surroundings at each rotation, effectively covering any blind spots that may exist.

To control the robot, the operator should have an interface suitable to the operator's needs. This interface should be intuitive and highly informative, presenting all necessary information while not dividing too much of the operator's attention and thus lowering the cognitive effort associated.

The controller used by the operator may even have some feedback, haptic or force feedback, according to the robot's movement or the force used by them to execute a task such as loading or unloading loads.

Another relevant aspect of teleoperation is the transmission delays, which is the time it takes for the information to be sent and received by both the operator and the vehicle. These delays have a significant impact on the operator performance [28] as a longer delay is associated with less confidence in the movements ordered to perform. Although the delay duration is important, the variance between them seems to play a higher role as [29] states. As a result, using an efficient communication system with low and consistent delays is desirable. Internet communication has been used by most of the teleoperation strategies, providing not only a greater range and bandwidth but also faster speeds, compared to Bluetooth or mobile networks [30]. Some popular protocols used are Transmission Control Protocol (TCP) and User Data Protocol (UDP). TCP allows for reliable transmissions of information by having error detection methods that trigger retransmissions in case packets are lost. However, due to the timeouts required to trigger the retransmission, it may not be worth retransmitting some information because it is outdated. As a result, this protocol may not be suitable for real-time applications. UDP on the other hand, provides a fast way of transmitting data at the expense of not ensuring that all of the data is transmitted and, as a result, some packets may be lost in the process. If information is being sent at high rates the teleoperation may not suffer from this issue as a control command that was not sent in the previous iteration may have already been sent in the next iteration. If significant delays are associated with the transmis-

sion of these commands it is clear that it would not be wise to operate the vehicle at high speeds since, as mentioned in [31], this delay limits the reaction times and, therefore, an effort should be made to mitigate the operator's perception of these delays. In any case, the safety of the vehicle should always be ensured, so there should always be a constant concern in the constant reception of commands from the operator, otherwise, the vehicle should stop until the situation returns to normal, thus allowing to maintain safety in case of connection failures that may arise.

2.4 Image Transmission

Efficient image transmission is critical to provide reliable teleoperation as the operator can see the AGV's view in real-time. Apart from that, it is expected that a reliable supervisory system will be able to visualize the fleet of AGVs in the environment in real-time with delays as minimal as possible or, at the very least, imperceptible to the operator. One factor that influences the length of the delays is the quality of the image, as higher quality implies longer transmission time. As a result, there is a trade-off between time and quality, so to achieve real-time transmission, one should prioritize time over quality, resulting in a better response from the operator. According to [32], this requirement is backed-up by the results obtained when comparing the operator's performance with different quality and delay values.

By reducing the image's quality and resolution it is possible to reduce the delays considerably without losing too much of its perception. Another approach is to apply a compression algorithm to the images before transmitting them, reducing the amount of information sent over the network. Some of the most used are Motion JPEG (MJPEG), codec H.264/Advanced Video Coding (AVC) and their suitability depends on the scenarios where they are applied. In the MJPEG algorithm each frame is represented by an independent Joint Photographic Experts Group (JPEG) image requiring little processing. On the other hand, in H.264/AVC there is a relationship between frames in which the I-frame contains the essential image data and the P and B frames contain the changes that occur in the scene, resulting in a higher processing time but lower bandwidth required [33]. As a result, MJPEG seems to be more suitable to real-time applications as it does not require so much processing, it allows for faster transmissions. There is, however, another algorithm, the H.265/High Efficiency Video Coding (HEVC) that is an improvement of the previous and seems to perform better compressions at lower resolutions at the cost of an increased computational power required [34].

The protocol used to transmit the images, just like in teleoperation, has high relevance in the transmission time. Real-time Transport Protocol (RTP) is a network protocol that can be used with either TCP or with UDP to deliver streaming audio and video media over the internet. Since a video is composed of a constant stream of images, the TCP protocol should not be used as, in addition to requiring more time for transmission, it will attempt to retransmit something that, given the time lag, may no longer be necessary. Therefore, UDP should be used, although it does not detect errors in the transmission, which is not a serious issue since the image to be transmitted has already been replaced by a more recent and thus more relevant one. In [33], a comparison

between the compression methods is made, along with the protocol used (TCP or UDP). It was shown that MJPEG had shorter transmission times (300 ms lower) but higher throughput (Mbps vs kbps), compared to H.264. Aside from that, the latency associated with TCP was not only higher but also more unevenly distributed than with UDP. Overall, the use of the MJPEG along with UDP provided the best transmission times, followed by MJPEG with TCP and, finally, H.264 with UDP.

Finally, as stated in the teleoperation section, it is necessary to pay attention to whether images are being received continuously or a connection problem has occurred. As a result, if the vehicle came to a stop, it would not be possible to distinguish between these two situations since the image representing that situation would always be the same.

2.5 High-level Command and Task Assignment to the Vehicles

Automating logistics operations requires a system capable of handling the creation of specific tasks related to moving loads and assigning them to forklifts. Starting with task creation, this can be achieved through a system that records a series of clicks made by the operator in the interface used to monitor the forklifts, such as 1) selecting the desired load in the image; 2) choosing the forklift; and 3) defining a destination to place the load. So, with the task already created, this system should be able to send this sequence of steps to the assigned forklift, along with all pertinent information.

To meet these requirements, a connection can be made between the interface, which can be a Supervisory Control And Data Acquisition (SCADA) and fleet management system. This way, when the operator performs the required clicks to create an order, this information can be passed to the fleet management system which can then communicate with a master-slave architecture, composed by a master and several Programmable Logic Controller (PLC) slaves corresponding to each vehicle. The master PLC communicates with the slave PLC to assign the task and the slave reports the status of the task execution to the master.

Some planning algorithms can be used to make better use of resources, removing the operator's responsibility for the AGV's allocation. These algorithms can be tailored to achieve a variety of goals, such as minimizing completion time, reducing the likelihood of deadlock occurrence, reducing the total distance traveled, or balancing the overall workload of AGV. Some of the most commonly used algorithms in these type of optimization problems are Tabu Search, Genetic Algorithms (GA) and Ant Colony Optimization (ACO) and an application can be found in [35].

Tabu Search is an algorithm that begins with an acceptable initial solution and then searches for neighbors of these solutions by making minor changes. In the first phase, it selects the best solution from among them and adds some aspects of that solution to a Tabu list. This way, in the next iteration, it will penalize solutions with the same characteristics to avoid becoming stuck in local minima. Some drawbacks of this algorithm are that it is a highly parameterized algorithm and that many iterations may be required to find a good solution.

GA is an algorithm that attempts to replicate natural selection by retaining the best solutions in each generation. Each individual has a chromosome on which the solution's parameters are stored. Each individual is assigned a fitness value, and the individuals with the lowest fitness level are

eliminated from the population in each generation. As a result, the remaining individuals generate successors by combining solutions with the others (crossover) resulting in the appearance of new solutions in the population. Furthermore, in each generation, a mutation factor is introduced to maintain diversity in the solutions generated and thus avoid being stuck in local minima. As a result, after a specified number of generations, a good solution can be obtained, resulting in an efficient task allocation to each AGV.

ACO is an algorithm inspired in the ant colony behavior in which each ant begins randomly searching for a solution and communicates this solution to the others by trailing a temporary pheromone. When applied to an allocation problem, each ant starts with a random selection (assignment of a task to an AGV) and then moves on to the next task, which is assigned to an AGV [36]. This allocation is done probabilistically using a heuristic based on the pheromone value. As a result, the higher these values, the more likely it is that one AGV will be chosen over another. When each individual discovers the solution the pheromone's values decrease and are updated in proportion to the quality of the solutions. This process is repeated until an acceptable solution is found or a specific termination criterion is met.

Chapter 3

State of the Art

Some research has been conducted in the field of high-level teleoperation, which enables the supervision or direct control of a robot. Some works are more concerned with advancements in teleoperation, whereas others present work that is used in logistics operations.

3.1 Improvements on Teleoperation

Some work has been done to improve teleoperation, either using raw images from cameras or virtual reality.

The work developed by X. Shen *et al.* provides immersive teleoperation to the operator using components that are easily available [1]. A car is teleoperated by an operator who has a view inside the vehicle provided by a stereo camera (42° FOV) with pan-tilt capability. Thus, the operator's head movements are translated into pan-tilt movements in the camera to replicate the operator's view if he was inside the vehicle. As depicted in Figure 3.1 a client (operator)-server (vehicle) architecture is used in which two computers communicate with each other over Institute of Electrical and Electronics Engineers (IEEE) 802.11n, allowing for connections via Wireless Fidelity (Wifi), 3G, 4G, or Wireless Local Area Network (WLAN). The video is transmitted in real-time to the operator using RTP and compressed using the H.264 codec and the GStreamer framework. The operator controls the vehicle using a racing wheel and pedal console game controller, with commands sent to the vehicle using UDP for the messages and services present in Robot Operating System (ROS). Furthermore, the latency present in the various connections used is compared, concluding that WLAN allows for better video quality; however, the fact that 3G and 4G have greater range opens the door for their future use in teleoperation. This solution enables immersive teleoperation and opens the door to future network applications such as 5G, which could significantly improve the results. However, using the compression algorithm MJPEG instead of H.264 would reduce the image transmission time.

A. Hosseini *et al.*, evaluates the performance of an interface that uses mixed reality to improve vehicle teleoperation [2]. Three cameras are used to provide a 210° view of the vehicle's surroundings. The captured images are compressed with the H.264 codec and sent over Long Term

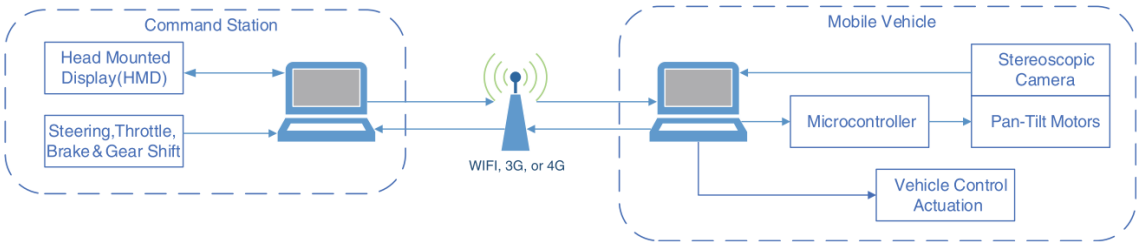


Figure 3.1: Software Architecture [1]

Evolution (LTE) using UDP. To supplement the image provided to the operator, information from two LiDARs is used to create a topological map with fixed-size cells, as can be seen in Figure 3.2. The cells are classified as free, occupied, or unknown in terms of the presence of obstacles, and the detected obstacles have the same height because they are captured at the same level. Driving simulations were carried out using the SILAB software [37], which allowed for the introduction of delays inherent in image transmission and commands of 500 and 100 ms, respectively. In these simulations, the operators demonstrated improved task performance, better perception of their surroundings, and less cognitive effort required to complete them. The use of LiDAR data appears to be interesting in terms of allowing a better perception of the surrounding environment and having additional information such as the distance to specific obstacles and mitigating the effect of some blind spots that may exist. As a result, it can be used to keep the operator from colliding with obstacles during teleoperation.

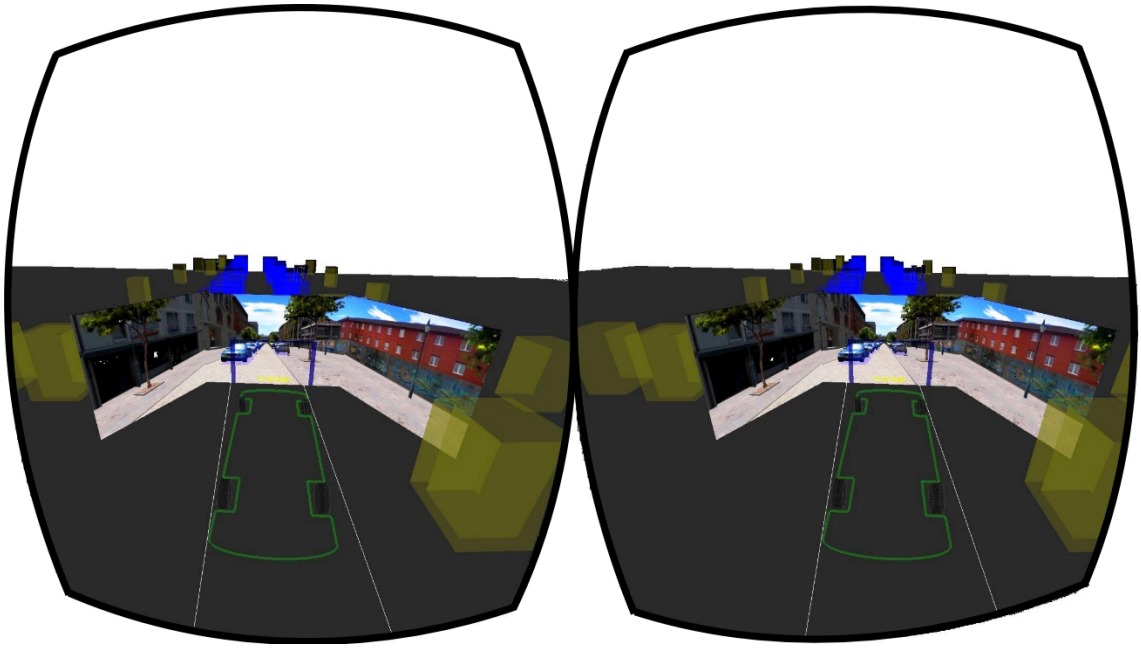


Figure 3.2: Perspective view of the mixed reality environment shown within HMD to the human operator [2]

F. Bazzano *et al.*, make a comparison about which is the best camera configuration to use in a robot for telepresence purposes, taking into account its FOV [3]. ROS is used for control, and Javascript is used for the interface, with a library that allows ROS to be used over the web. The operator can control the vehicle in two ways: teleoperation or remote control. In the former, the operator clicks on the image in the web interface, and the coordinates of the point selected by the operator are converted to a point on the real map via ray-tracing. All points on the image are assumed to correspond to positions on the map with axis $z = 0$. As a result, for each pixel (x,y), there is an intersection with the $z = 0$ plane that corresponds to the location to which the robot will move. This movement is accomplished through the use of planning algorithms, and deviation from both fixed and moving obstacles is present. Aside from that, the planned path is shown directly on the video as it moves until it reaches its destination, as showed in Figure 3.3. The remote control is accomplished by using the keyboard to communicate directly with the robot via the Roslibjs library, allowing the user to change both the orientation and speed of the robot. While the web interface provides the most responsiveness to the operator, it suffers from delays associated with data display and transmission.



Figure 3.3: Telepresence interface [3]

The work done by J. Xiao *et al.* uses virtual reality to aid the robot control [4]. The operator interacts with the robot he will control via virtual reality. He has two control modes available to him via a wireless HTC Vive controller: direct and more autonomous. In the former, the controller buttons are used to change the robot's direction and speed. In the latter, the operator points to a virtual position, and the robot calculates the best path to take to reach that point (on the ground) autonomously. ROS is used to control the robot, and the data (commands and images) are encoded

in JavaScript Object Notation (JSON) and sent via UDP over a wireless network. To provide this type of interface, intensive processing is required to generate the point clouds, which may result in some delays that may interfere with teleoperation.

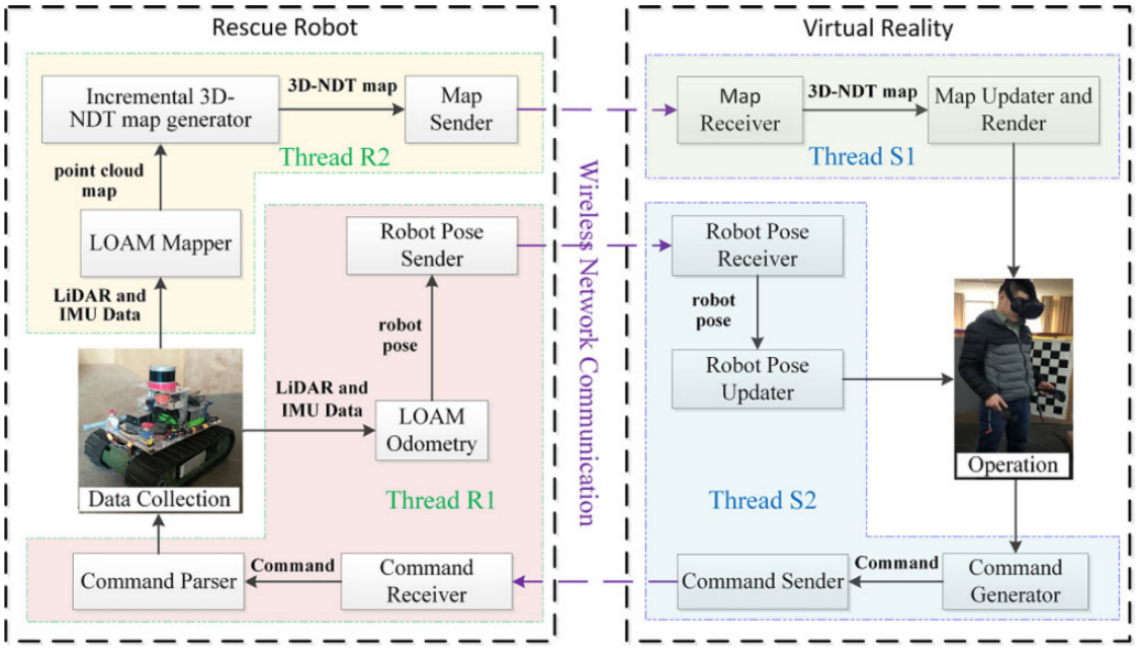


Figure 3.4: The proposed Human-Robot Interaction (HRI) method, which is built upon real-time robotic mapping and online Virtual Reality (VR) visualization, where the robot and remote control station communicate with each other by a wireless network. [4]

M.E. Walker *et al.*, implemented a teleoperation's system where the operator controls a drone using an Xbox controller and an Augmented Reality Head-Mounted Display (ARHMD) [5]. Augmented reality shows the operator a "surrogate" as well as an image of the actual drone. They support two modes of navigation: direct and waypoint navigation. In the former, the operator directs the "surrogate" to the desired location, and the drone immediately follows, replicating the movement. In the latter, the operator uses the "surrogate" to select the waypoints where the drone should move and then commands the drone to do so. The application is built with the Unity game engine, and commands are communicated via UDP over a wireless connection. Users have improved their path completion time and reduced their cognitive effort as a result of this interface. Although the interface was developed for an aerial robot, it could also be used for mobile robots to provide insight into the robot's future position. If there are significant delays, it could even represent a way to mitigate the impact of delays because the vehicle's current and future positions are given in real-time. Furthermore, if the operator can see the future robot's trajectory, he will be able to predict what will happen to the robot and react accordingly.

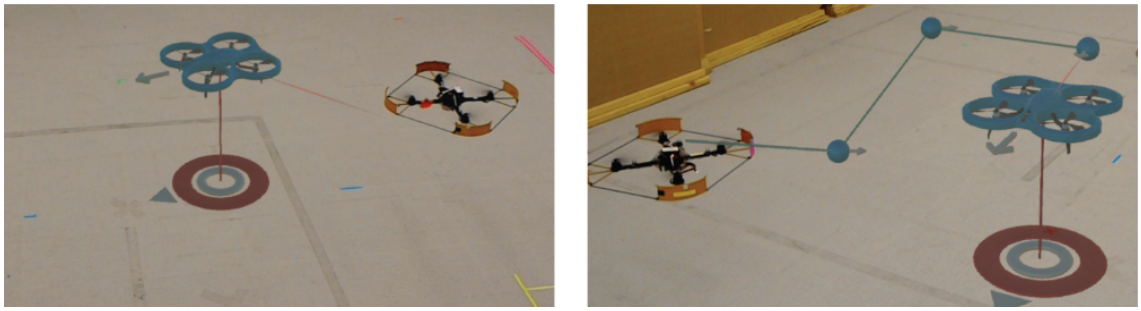


Figure 3.5: Two Augmented Reality teleoperation interfaces designed in this work: Left - Realtime Virtual Surrogate (RVS), Right - Waypoint Virtual Surrogate (WVS) [5]

3.2 Applications in Logistics Operations

There have already been some forklift teleoperation implementations in logistics operations.

M.C. Mora *et al.* developed a system that allows a fully autonomous forklift to be controlled in two modes of operation: path tracking and teleoperation [6].

Path tracking is the forklift's autonomous mode of operation in which a path is calculated and followed by the forklift based on its initial and destination positions. Because the forklift has implemented the Artificial Potential Fields algorithm along the path, there may be some deviations from the initial trajectory if an obstacle is detected.

Teleoperation is accomplished through the use of a client-server architecture (Figure 3.6), in which the operator serves as the client and the computer in each forklift serves as the server. The server was developed with Labview and communicates with the forklift's PLC via RS232, sending information to the client such as its orientation and position. If this communication fails, the forklift will remain stationary until the situation normalizes. The client was developed in Darkbasic and receives operator commands, such as orientation via a steering wheel with force feedback and speeds representing the accelerating and braking function. The communication between these applications takes place over WLAN using TCP/IP, resulting in a robust but not very efficient control due to the inherent delay in command transmission and the possibility of retransmissions. A camera with a TCP/IP server is installed in each forklift, which transmits images to the operator at a fixed rate. This paper also presents a high-level system, which involves communication between an Enterprise Resource Planning (ERP) system and a SCADA interface for order assignment and control. Because TCP is used for both control commands and image transmission, the transmission time is longer, whereas UDP would result in faster transmissions.

An hybrid (deliberative and reactive) Autonomous Robot Architecture (AuRA) control architecture is implemented using the MissionLab framework in [38]. Some agents have already been implemented in this framework to control the forklift and provide relevant information to the operator, such as an interface to show the current position, map, and obstacles detected; an order editor; an observer of the robot's state and sensors; robot movement control using a joystick; and communication between all agents. The operator may either take control of the forklift while it

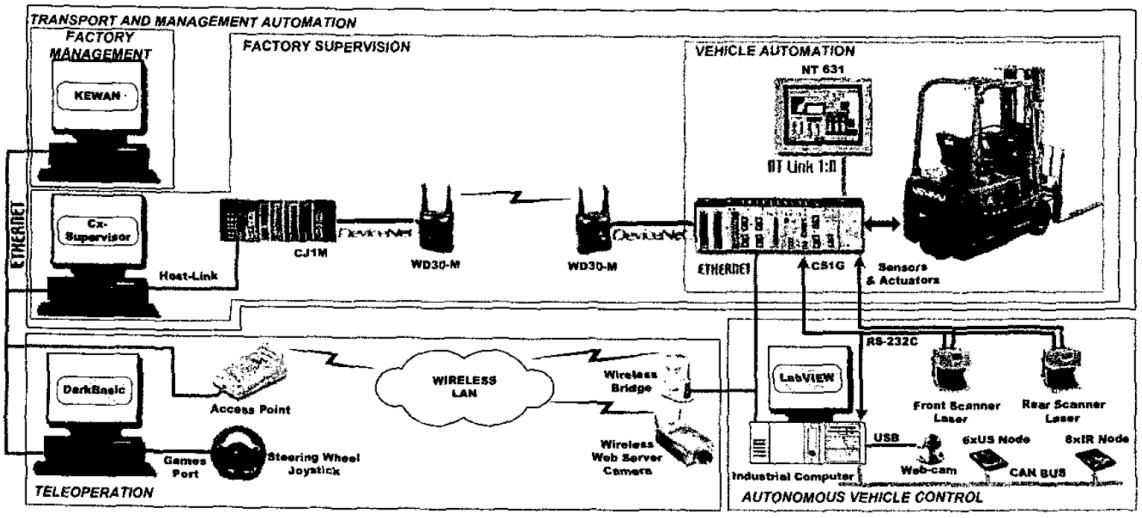


Figure 3.6: Hardware architecture [6]

performs its tasks autonomously or be requested to intervene if a sensor fails. Nonetheless, the obstacle avoidance agent monitors the teleoperation to prevent the operator from colliding.

H.S. Ahn *et al.*, present a work in which a forklift is controlled by two computers: one in the forklift and the other in the control station [7]. The Player platform employs a client (operator) - server (forklift) architecture, with data transmitted via TCP as depicted in Figure 3.7. If the latency inherent in the transmission exceeds a certain threshold, the stacker is immobilized until the situation returns to normal. Two threads are used in this architecture: one updates the information sent from (direction, speeds) the server and the other updates information provided by the operator, such as controller handle movement or button presses. A PlayStation 3 controller is used to control the direction, speed, and position of the forks, and its vibration (haptic feedback) allows the user to feel the position of the wheel. The GStreamer framework sends stereo audio and images captured by four cameras mounted on the forklift, providing a better perception of the surrounding environment. Each camera was fitted with a 180º fisheye lens to provide a 360º view of their surroundings. One of these cameras has been modified to include pan and tilt functionality. As a result, the operator has the option of observing in one of two directions: downward, to see the path and nearby obstacles, or forward to see a greater distance. These images were transmitted using the GStreamer framework, where they were compressed with the H.264 codec and sent over TCP. The protocol was chosen because it appeared that there were no significant delays detected when compared to the UDP protocol. Furthermore, it is claimed that when the latter protocol was used, there were packet losses and deformations in the video sent to the operator. The data transmission delay between the Player platform and a forklift at a distance of 10 km was measured, yielding an average round-trip time of 110 ms.

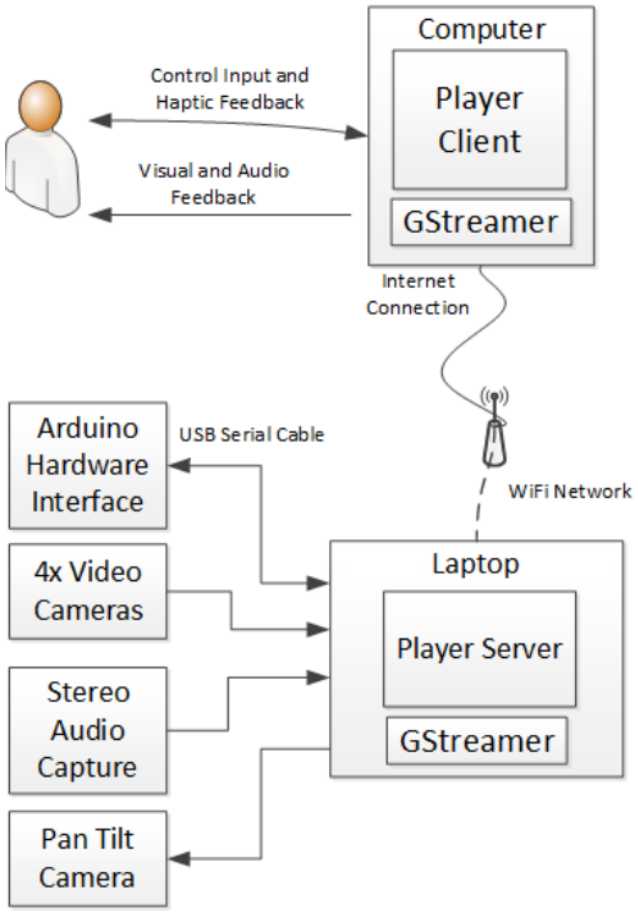


Figure 3.7: System architecture of forklift teleoperation system, which consists of a laptop, an Arduino microcontroller, four cameras, a PS3 controller. [7]

3.3 Surveys on Teleoperation

Some works have already investigated the field of teleoperation, aggregating numerous works and the methods used to teleoperate various types of robots.

S. Opiyo *et al.* discuss several works done in the field of teleoperation, including the type of the interface used and the method of connection between operator and robot [30].

In terms of interface, the HMD provides the best sense of presence, but problems such as motion sickness can occur due to poor quality devices (the equipment is relatively expensive) or significant delays between head movement and visualized image.

When it comes to wireless technology (Wifi, mobile network, and Bluetooth), most works adopt Wifi because, while it has a shorter range than mobile network, it is easier to use and consumes less energy. However, with the introduction of the 5G network, the mobile network may become more popular because it not only has a longer range but also a faster data transmission speed, which is critical for teleoperation.

In teleoperation, it highlights that the operator's performance is affected not only from the information received from the environment but also by the reliability of the communication channel. Apart from that, the authors suggest that more research could be done on using Big Data and Machine Learning to help the robot learn its environment. In this way, the robot could provide the operator with decision-making suggestions, reducing cognitive load significantly because it does not need to constantly evaluate the situation.

M.D. Moniruzzaman *et al.* present a comprehensive aggregation of the various methods used in teleoperation and enhancement techniques [8]. As it is stated, the most common teleoperation mode is supervisory control followed by direct control. The former is still evolving and consists of the robot's ability to autonomously plan paths and avoid obstacles. The latter, also known as manual remote control, is the most basic method of teleoperation and is highly dependent on transmission delays. Finally it resumes the enhancements that can be made to teleoperation in 5 main categories, presented in Figure 3.8, thus being the operator perception, interface, control system, latency and a multimodal teleoperation system. Some of these solutions were used in the works presented above as, for example, in [2] both map merging and mixed reality were used to improve the operator perception and interface.

3.4 Discussion

As it is seen in Table 3.1, HMDs are used occasionally to provide the operator with greater immersion when performing teleoperation. These interfaces, however, require some image processing, which results in longer delays. Furthermore, if there are delays between the images seen by the operator and the movements made with his head, it can create a sense of unease, which does not occur with a flat screen interface. The images sent can be compressed using H.264 to use less bandwidth, as some of the works do, or MJPEG, which requires more bandwidth but takes less time to process. In terms of transmission algorithms, using UDP results in faster transmissions

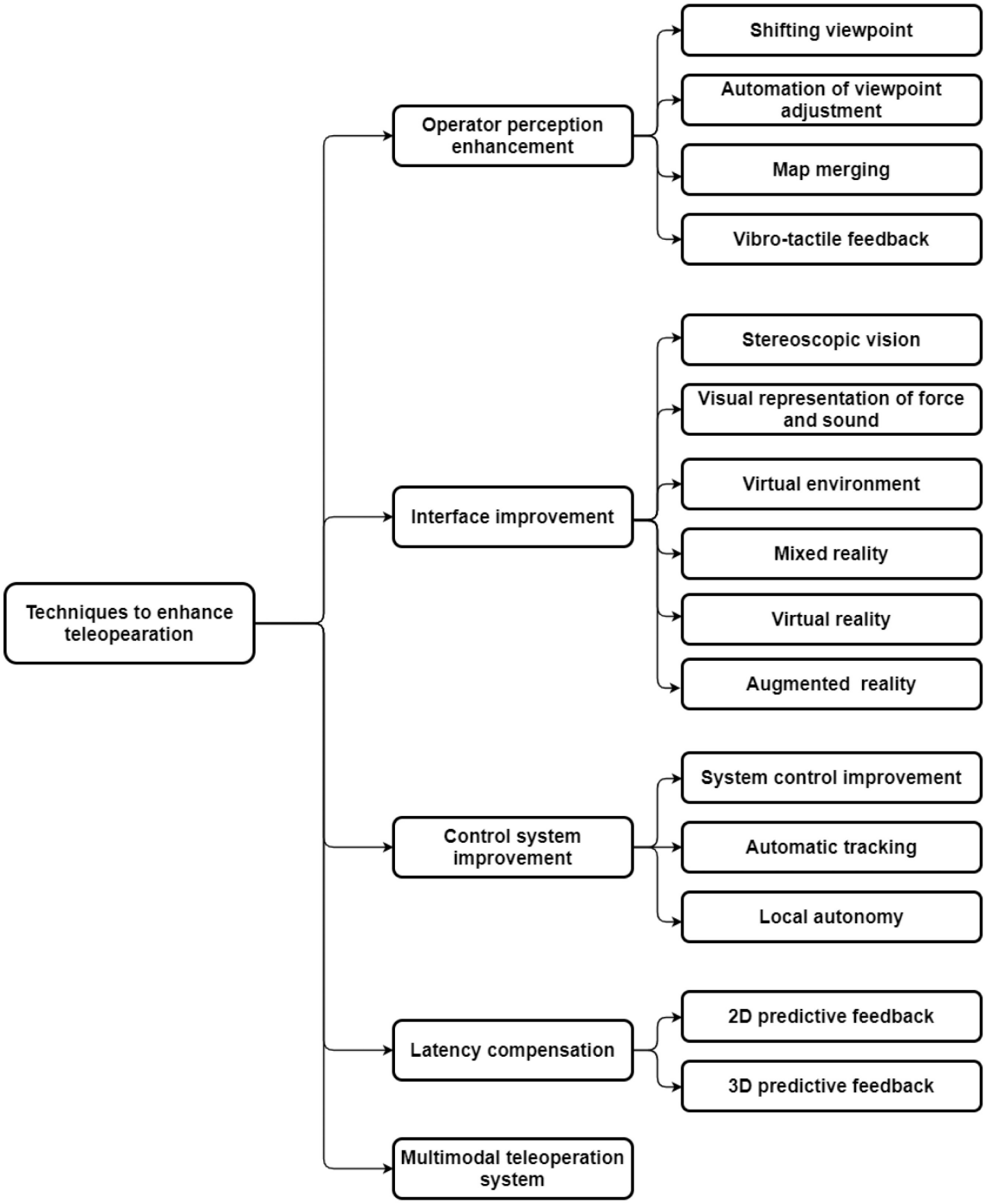


Figure 3.8: Classification of teleoperation enhancement techniques [8]

than using TCP, and the mode of communication is mostly via Wifi. However, with the introduction of 5G, this may no longer be the case, as 5G allows for a higher range and faster transmission speeds.

In industry, delays are relevant when performing teleoperation. As a result, a simple interface could be used to reduce processing delays. If this is not an issue, then using HMDs might improve teleoperation performance as long as the equipment has enough quality to match the transmitted images to the operator's movements. To achieve the shortest possible delay MJPEG compression method in conjunction with the UDP protocol can be used, either over Wifi or, if possible, over 5G.

Table 3.1: Overview of the technologies used in the previous works

Work	**Interface**	**Communication**	**Compression algorithm**	**Transmission protocol**
[1]	Head mounted display	Wifi	H.264	RTP, UDP
[2]	Head mounted display	LTE	H.264	UDP
[6]	Flat screen	WLAN	Information not available (N.A.)	TCP
[38]	Flat screen	N.A.	N.A.	N.A.
[3]	Flat screen	N.A.	N.A.	N.A.
[7]	Flat screen	Wifi	N.A.	TCP
[4]	Head mounted display	Wifi	N.A.	UDP
[5]	Head mounted display	Wifi	N.A.	UDP

Chapter 4

Implementation

This chapter describes the tasks performed to develop a high-level teleoperation system capable of being tested on a forklift, beginning with a general overview of how the system works and progressing to specifics about the adaptations made and how the operator controls the forklift in the two modes of operation: remote and autonomous.

4.1 System Overview

Developing a high-level teleoperation system requires identifying its main components. The input devices and the interface enable the communication between the operator and the forklift. The sensors, on the other hand, collect information about the environment as well as the status of the forklift. Finally, two computers process this information and complete the assigned tasks: one for the operator and one for the forklift. Figure 4.1 depicts these system components and their interactions.

In this system, the operator can control the forklift using two modes of operation and an interface that allows him to visualize and interact with his surroundings.

The ZeroMQ (ZMQ) protocol allows exchanging data between the forklift computer, the operator computer, and the interface.

In the forklift remote control mode, two input devices are used: the steering wheel and pedals to control the forklift's direction and speed and the joystick to control the raising and lowering of the forks. When it comes to steering control, the computer in the forklift uses Controller Area Network (CAN) messages to communicate directly with the steering driver installed in the forklift. An Arduino is used to control both the traction speed and the forks. It communicates with the traction and fork motors, converting the commands it receives into CAN messages that are sent to a single CAN bus. The Arduino sends speed commands to the traction motor and receives feedback to estimate its position and actual speed. An absolute encoder measures the distance between the forks and the floor, communicating with this Arduino via the Synchronous Serial Interface (SSI) protocol.

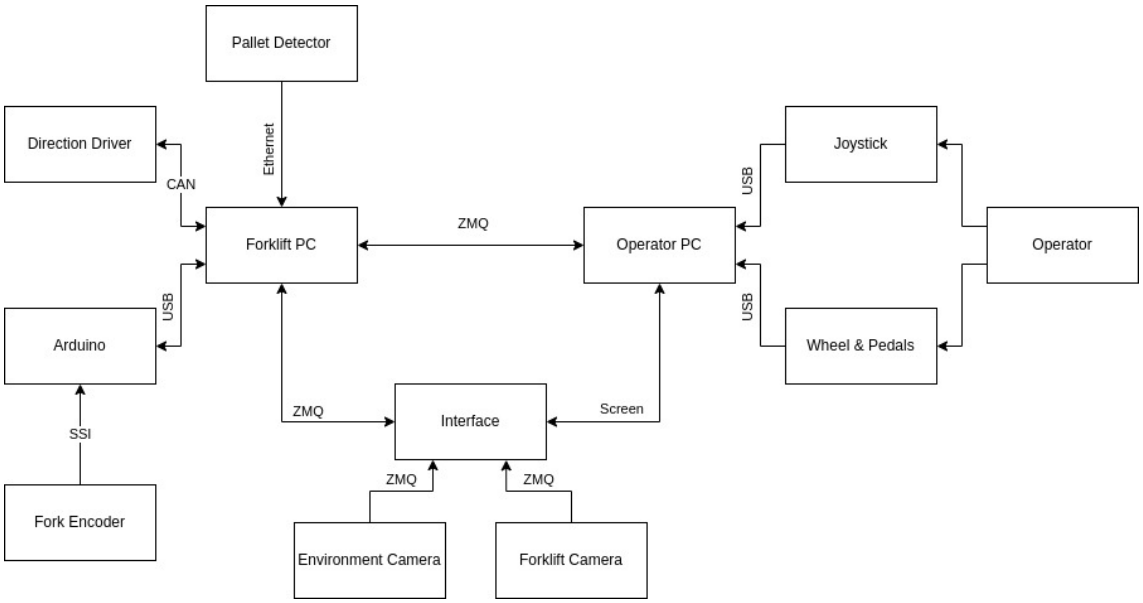


Figure 4.1: System Block Diagram

In the autonomous mode, the operator performs a series of steps in the interface that allows him to send information to the forklift computer and thus autonomously pick or place a pallet in a given location.

With the interface, the operator can also view images captured from two cameras: one fixed and one mobile while operating the forklift in these two modes. The first provides an overview of the forklift's environment. The second camera is mounted on the forklift and shows what the operator would see if driving it.

Finally, the pallet sensor aids in inserting the forks into the pallet and sends the pallet detection information to the computer in the forklift via an Ethernet cable.

4.2 Forklift Adaptation

Testing a high-level teleoperation system requires that the used vehicles are suitable to this type of control. The forklift available for this purpose (a Toyota SWE140) required some additional adaptations from the ones already made to make it remotely controllable.

This forklift was already controllable in two operating modes: remote and autonomous. A joystick (Logitech Gamepad F710) controlled its direction and speed in the remote mode. As for the autonomous mode, the navigation stack developed by Institute for Systems and Computer Engineering, Technology and Science (INESC TEC) - Centre for Robotics in Industry and Intelligent Systems (CRIIS) had already been installed, which allowed moving between two mapped points on a given map autonomously.

4.2.1 Hardware Adaptation

The forklift already had some hardware to enable its control on both operation modes Two SICK safety lasers were installed around the forklift to provide 360-degree coverage of the environment, as presented in Figure 4.2, allowing the SLAM algorithm to map the surroundings and the forklift's location on it. Figure 4.3 depicts the placement of these lasers, identified by the number 1. It is also possible to define different safety zones based on the information provided by these lasers so that the forklift can be immobilized if there are any obstacles in the way.



(a) Back (b) Front

Figure 4.2: Laser scanners placement

Finally, two emergency buttons on each side were installed on this forklift to be immobilized if necessary, as depicted in Figure 4.3, identified by the number 2.

With these features in mind, the high-level teleoperation system required some additional components, as described in the following sub-sections.

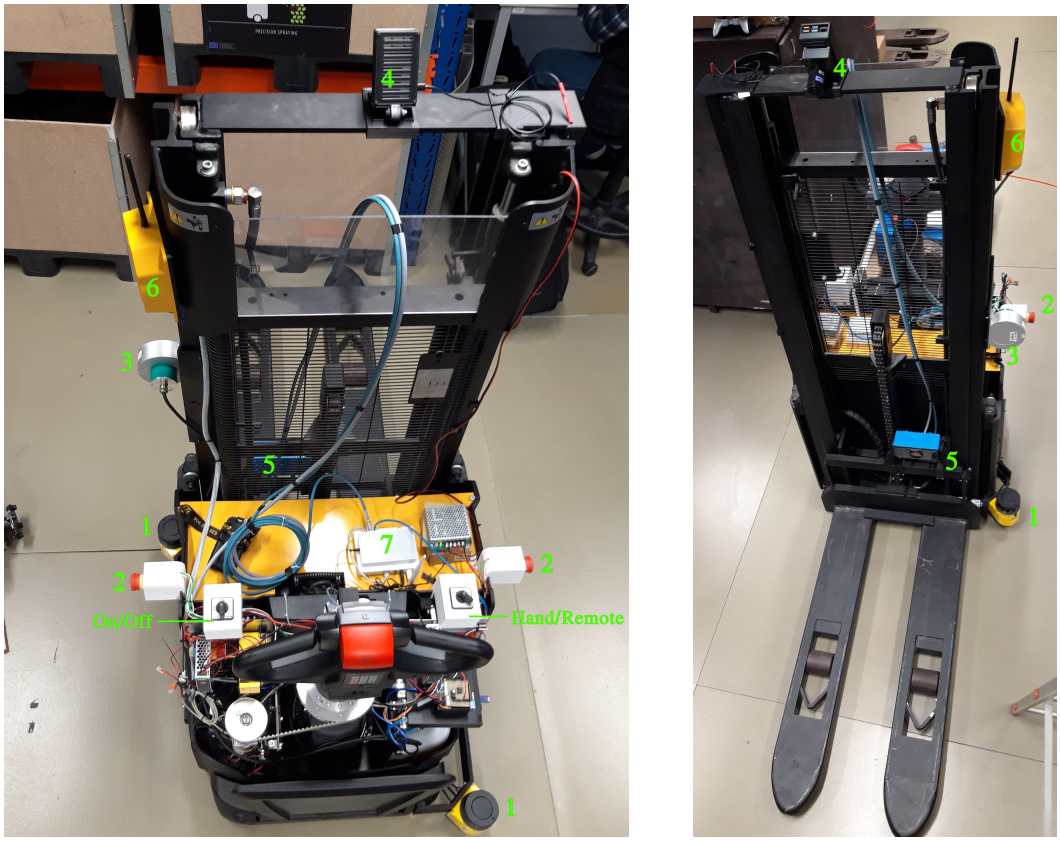
4.2.1.1 Fork Encoder

A draw wire encoder (LIKA), as presented in Figure 4.4, was required to measure the forklift's forks position, which is essential when operating a forklift via teleoperation.

This sensor measures the length of the wire pulled at any given time and can, thus, be used to determine the height of the forklift's forks as depicted in Figure 4.3, identified by the number 3. Since this sensor is absolute, the measured value is clear. It also has two additional features for configuring the sensor after installation: reversing the counting direction and resetting the counting at a specific position. This sensor uses the SSI protocol, and its measurements are resistant to noise because it employs the differential transmission mode.

The SSI protocol is a data transmission protocol based on the RS-422 standard that allows the data transmission from a slave (encoder) to a master (Arduino). For this transmission to take place, these two parties must be synchronized using a clock signal.

Thus, the encoder stores the position value at the first falling edge of the clock (logic level from high to low). Then, as presented in Figure 4.5, at the next rising edge of the clock (logic



(a) Front view (b) Back view

Figure 4.3: Forklift's hardware placement. The hardware is as follows: 1 - laser scanners; 2 - emergency buttons; 3 - fork encoder; 4 - camera; 5 - pallet sensor; 6 - wireless emergency button; 7 - access point and network switch



Figure 4.4: Fork Encoder

level from low to high), the transmission of this value begins, starting with the most significant bit until the last bit in each rising edge. If the number of rising edges exceeds the resolution + 1, the encoder will send zeros for each rising edge; at least 24 rising edges are required for a 23-bit resolution encoder. If the number of rising edges sent is higher than the required, the encoder will continue to send zeros. When the reading is complete, the clock signal must be set to logic level HIGH for a predetermined amount of time (16 microseconds) to signal a new transmission.

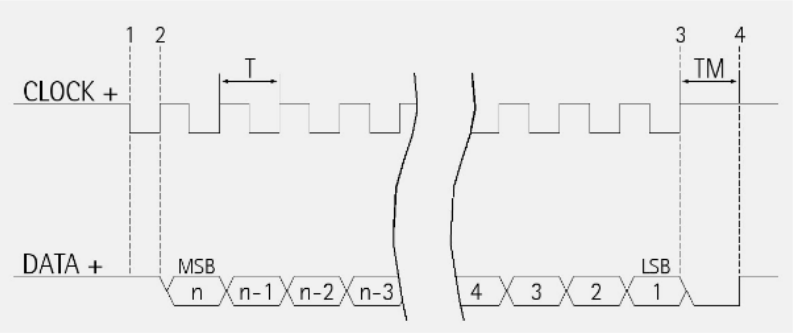


Figure 4.5: Data transmission of the fork encoder (image from the manual available on LIKA's website [9])

Reading the encoder required a microcontroller (Arduino) that could send the clock signal to the encoder and receive the data sent by it. Given the sensor's differential transmission, converting the differential signal into a single signal and vice versa required an integrated circuit (Max490). As a result, aside from the power pins, only two Arduino pins are used to communicate with the encoder: one for the clock signal and other for the data. The circuit was implemented on a Veroboard and adapted to fit the existing Arduino shields. As a result, this shield can be easily placed or removed without interfering with previous work and is a safer circuit with less risk of loosening connections.

Figure 4.6 represents a schematic representation of the circuit, whereas Figure 4.7 depicts the Veroboard circuit. Adding two buttons to the circuit makes accessing the sensor reset, and count inversion features simple. According to the datasheet, these two pins must be connected to the encoder power supply (7.5- 30 V) for at least 100 microseconds before pressing the button.

Given the similarity in reading the data using the same protocol, the encoder's reading process implementation uses an adaption of the code developed by Atar Babgei [39].

In this manner, obtaining the encoder position measurements is possible while taking special care to generate a clock signal that is as stable as possible. That means that no interruptions or other functions should be called during data transmission after the encoder reading has begun, or else noisy measurements far from the actual value will result. Another factor to consider is using pins free from any other function for the Clock and Data signals; otherwise, the same errors described above can occur.

Finally, when it comes to the encoder's installation, the best-found location, according to the dimensions of the forklift, was attached to the side. The encoder was a little above the ground,

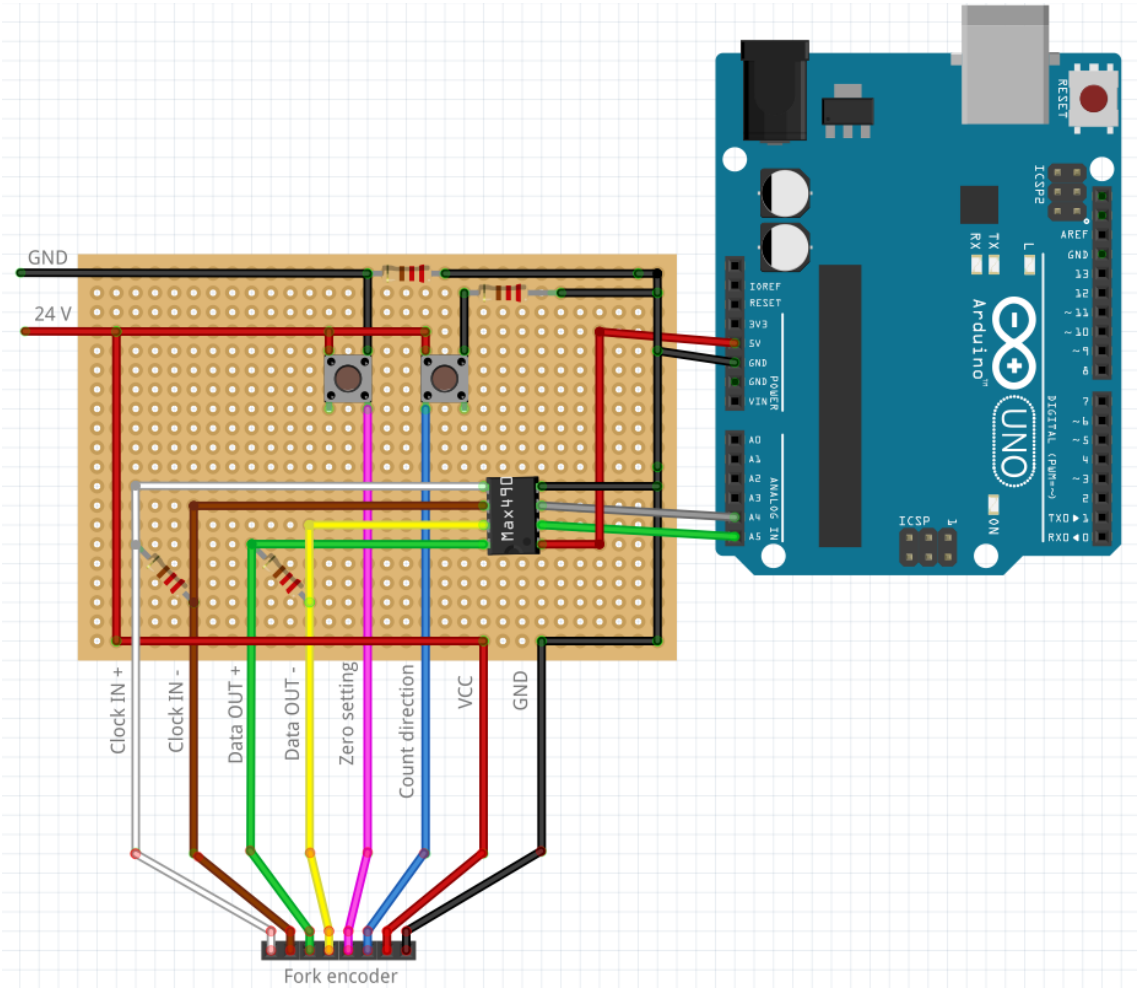
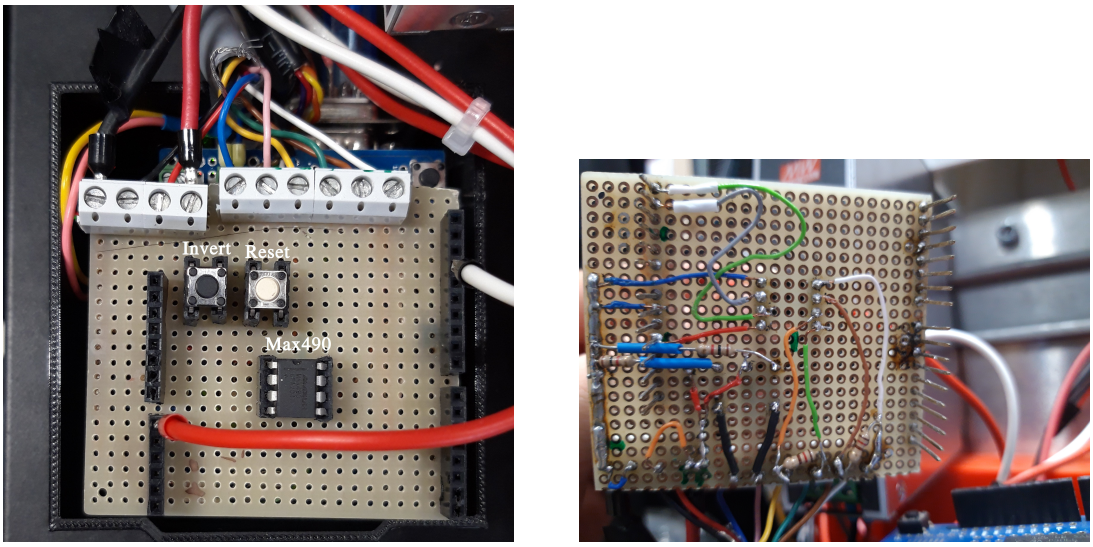


Figure 4.6: Fork's encoder circuit schematic



(a) Top

(b) Bottom

Figure 4.7: Fork's encoder circuit implemented on a Veroboard

directed downwards, with part of the wire stretched and attached to the forklift platform via a pulley, as shown in Figure 4.4. That ensured that the forks' position was measured in a single direction regardless of height. This linearity would not be possible if, for example, the encoder had been turned upwards with the pulley on top. In that case, the wire would retract initially and then stretch after a certain height, creating ambiguity in the data reading because there would be two different positions with the same value given by the encoder.

4.2.1.2 Cameras

In order to remotely control a vehicle, it is necessary to visualize it, and its surroundings, to avoid obstacles and have a view closer to the one the operator would have if he was driving it. As a result, the installation of two cameras, each connected to a RaspberryPi, as depicted in Figure 4.8 provided both views: one on the forklift and the other fixed near the ceiling to provide a broader view of the forklift's surroundings. Figure 4.9 presents the placement of both cameras. Two lenses were attached to both cameras to improve the image and provide a sharper and more expansive view.



Figure 4.8: Raspberry Pi Camera



(a) Camera installed in the forklift

(b) Camera fixed to a ceiling structure

Figure 4.9: Camera placement

When installing the forklift's camera, the initial thought was to place it at the operator's eye level while he was remotely driving the camera. However, this would not be feasible because, in

addition to a grid blocking most of the view, it would no longer be helpful in two scenarios: when a box blocked the field of view or when the forks were above the camera's level. As a result, placing it in an area of the lifting surface separate from the forks showed improvement. This surface has the particularity that it only rises when the forks reach a certain height (and then rise together). This placement allows the operator to have a broader field of vision and can track the elevation of the pallet throughout its course, as depicted in Figure 4.9 and in Figure 4.3, identified by the number 4.

Regarding the image captured by the camera, being connected to a RaspberryPi makes it possible to program and change the parameters of the image captured. Furthermore, it enables them to be accessible with an Ethernet or an High Definition Multimedia Interface (HDMI) cable to have an interface that can be seen by the user. Besides that, it can also be accessed using the Virtual Network Computing (VNC) application. This access is possible due to the execution of a script that creates a VNC server and launches the application responsible for capturing and publishing the image whenever they are connected.

Regarding the compression method, the MJPEG method provides faster transmission of the captured images and requires more bandwidth. Following compression, the images are sent to an interface via the ZMQ protocol, employing the publisher-subscriber communication pattern. As a result, the camera publishes the image on its IP address, and the interface subscribes to it at the same address.

4.2.1.3 Pallet Sensor

Installing a pallet sensor in the forklift came to be a way to help pick or place a pallet in a specific location. This sensor, widely used in the industry, consists of a 3D camera (Visionary-T SICK) that uses infra-red lightning and comes with an application capable of recognizing where the forks should enter to load the pallet. This machine learning-based algorithm employs point clouds generated by measuring the time-of-flight between the emitted and reflected light signals to the objects.

This sensor's image can be accessed directly in a browser by entering its IP address. This window, as presented in Figure 4.10, displays the image captured by the sensor and some configuration parameters such as integration time, camera installation position, maximum height to be searched, and whether or not to average the measurements. Following the detection of the pallet, the image has a rectangle outlined, corresponding to the pallet's prediction. Apart from that, there are also three points: two relating to each pallet's pockets and one relative to the center of the pallet.

These points, with coordinates x, y, z, belong to the camera reference frame, where the z coordinate is the distance to the pallet. When this detection occurs, a 3D view of the captured image with the shape of the detected pallet is also available. With these coordinates, it is possible to calculate the pallet's orientation concerning the camera reference frame and thus correct the forklift's orientation to align it with the box, as well as its distance from the box, which indicates the distance to be traveled to pick the pallet.

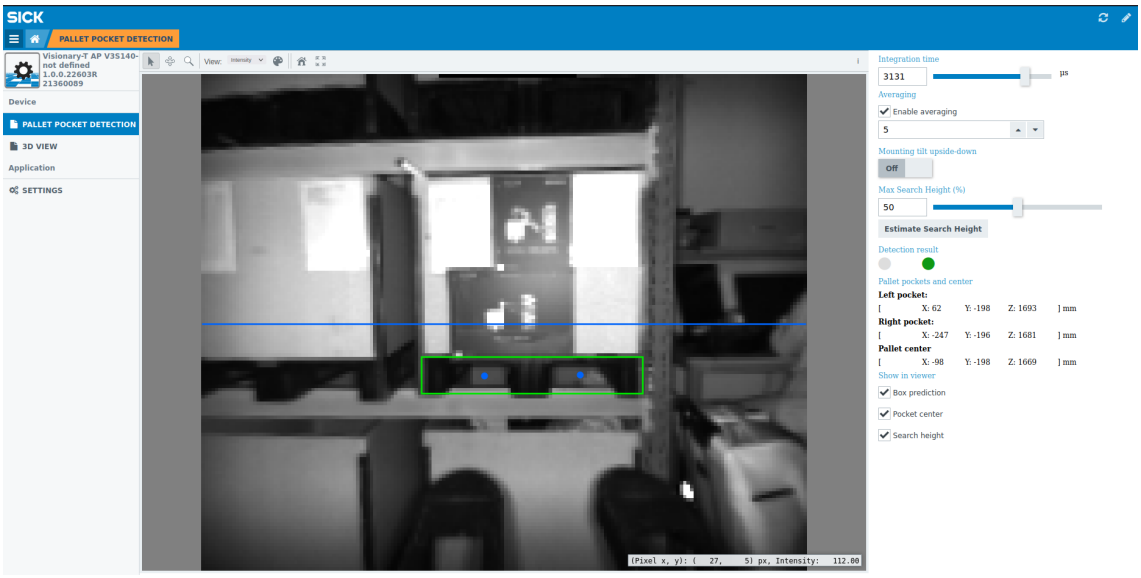


Figure 4.10: Pallet sensor web interface

Regarding the installation, mounting the sensor on the same structure where the forks are connected allowed it to always follow their movement, as depicted in Figure 4.11 and in Figure 4.3, identified by the number 5. However, it required tilting the sensor due to space constraints so it would not collide with a structure on the forklift when lifting the forks. As a result, the captured image shows a portion of the forklift's forks.

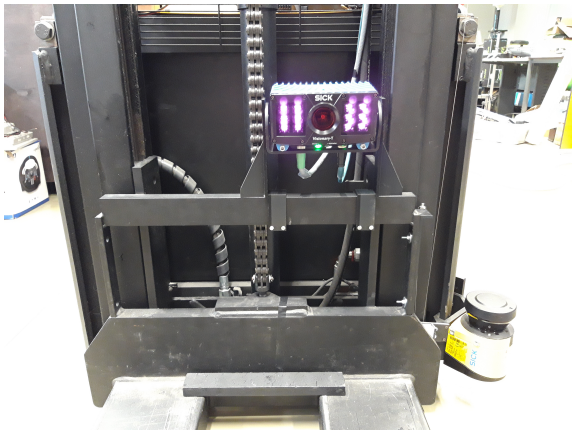


Figure 4.11: Pallet sensor placement

4.2.1.4 Wireless Emergency Button

The emergency system installed in the vehicle is effective when someone is close to the vehicle, ready to push the emergency button. However, this is not always possible and requires someone

close to the forklift when remotely controlled. Using a remote emergency button allowed to address this issue as the operator could use it without being next to the truck. This emergency system consists of a transmitter (emergency button) and a receiver installed in the forklift that continuously receives messages from the active transmitter, as depicted in Figure 4.12 and in Figure 4.3, identified by the number 6. By pressing this button, it blocks the transmission of messages which causes the emergency system to be activated, causing the forklift's immobilization. This behavior is also triggered when the transmitter is further from the receiver than the stipulated distance, when the transmitter's battery runs out, or when it is charging.



Figure 4.12: Remote emergency system: emitter (button) - left and receiver - right

Although the installation recommendation for this sensor suggested connecting it to a permanently active power source, that seemed unnecessary as it would lead to spending unnecessary battery when the forklift was not in use. As a result, the circuit that powered the forklift's components also powered the receiver circuit, ensuring that it was only active when the forklift was also active. Since the emergency button receiver can only switch its contacts when powered, these contacts remain open when the forklift is turned off and only close when the remote emergency button is powered and unlocked. Otherwise, the forklift will remain in the emergency mode.

4.2.1.5 Access Point and Network Switch

Receiving the information provided by the pallet sensor requires this sensor to be connected via an Ethernet cable. When initially linked to the forklift's computer, and if chosen the cable connection, the sensor interface would be accessible by entering the sensor's IP address into the browser. However, this blocked access to the internet, as well as the data transmission between the two computers. Even with both types of connections on, only one of them would work. An Access Point (D-Link DAP-1665) and a network switch (TL-SF1008D) allowed solving this issue. Thus,

the access point worked as a client of the laboratory's existing network. The switch links to this access point and has two network cables: one for the forklift computer and another for the pallet sensor, as depicted in Figure 4.3, identified by the number 7. This way, anyone on the same network could access the pallet sensor, and the computer in the forklift would still have internet access.

4.2.2 Software Adaptation

The existing software on the forklift computer was created with ROS and developed using C++. The already implemented software primarily employs the publisher-subscriber communication pattern, in which publishers publish information on a specific topic and subscribers subscribe to those topics. Thus, the adaptations had to fit into this system using the already implemented nodes. Figure 4.13 depicts the overall interaction of several systems, including the software already developed in the forklift's PC, the software implemented in the forklift's PC, the software present in the Arduino, and the software developed in the operator's PC. Each connection's content represents either the protocol used or the topic where the message is published. What is in square brackets, on the other hand, refers to the type of message sent, whereas what is in curved brackets only specifies the message's function.

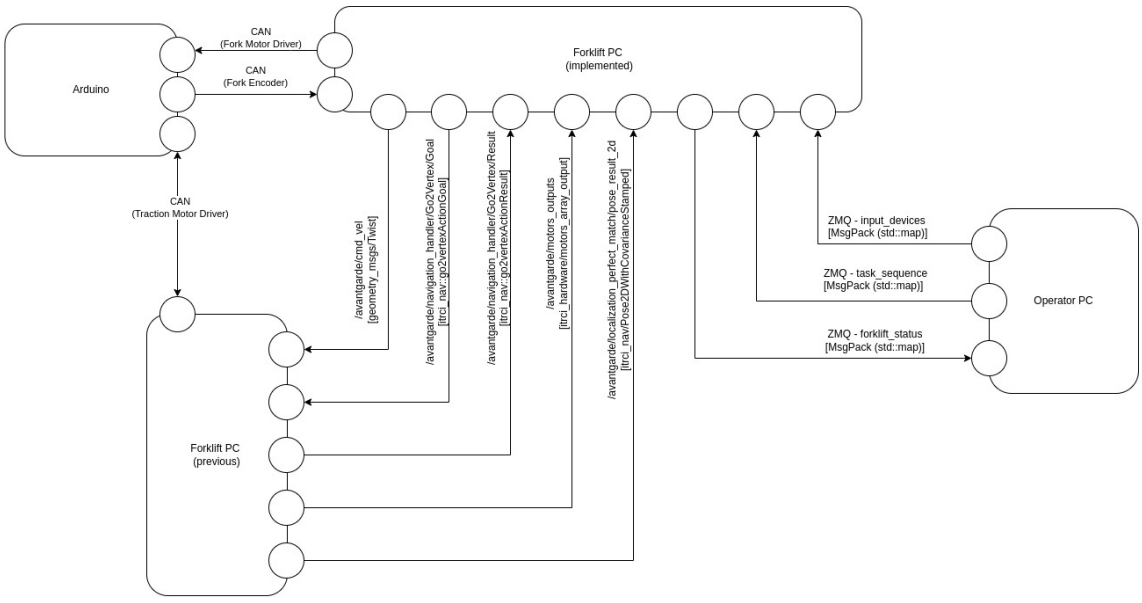


Figure 4.13: Overview of the software architecture

To better understand how this system works, there will be a detailed description of each component depicted in Figure 4.13.

4.2.2.1 Previous Software in Forklift's PC

Starting with the software already developed in the forklift's PC, this is composed of multiple nodes, Figure 4.14 depicts the most relevant ones.

The *driver_joy* node reads the buttons and axes on the Xbox joystick. It sends the reading to the *omnijoy* node, which receives the reading and converts it to linear and angular velocity values before sending them to the *odometry_processor* node. The *odometry_processor* node converts these values into values compatible with the traction and steering motor drivers, sending them to the *odometry_interface* node and receiving status feedback from the same node. After receiving these values, the *odometry_interface* node converts the received values into messages compatible with the *arduino_communication* node, which controls the forklift's steering driver, and the *master_canopen* node, which controls the forklift's steering driver. The *arduino_communication* node converts the received traction value into a CAN message that the Arduino will receive. Furthermore, it will receive CAN messages from the Arduino, allowing it to obtain feedback from the drive traction motor. Once these messages are received, this node converts them into a value compatible with the *odometry_interface* node. Regarding the forklift's direction, the *master_canopen* node allows its control in two ways: speed or direction. Any angular velocity value translates into steering wheel rotation speed in velocity mode. In steering mode, this value must be between -pi/2 and pi/2 because it relates to the steering wheel position (which must not exceed its mechanical limits). So, after receiving the steering reference value, it will exchange some CAN messages with the *Direction Motor Driver* to perform the requested steering and provide feedback on the current position of the steering wheel.

Regarding autonomous navigation, the already developed software also contains several nodes for the localization and mapping of the environment surrounding the forklift, which uses information from the installed lasers. Because these nodes allow the forklift to navigate autonomously, only two nodes containing relevant information for this work will be covered: *localization_perfect_match* and *navigation_handler*. The former node provides information about the forklift's current position and orientation, whereas the latter, on the other hand, enables the forklift to move autonomously between two locations. In a first step, the forklift creates a map of its surroundings using lasers. After creating this map, it is possible to select specific points on the map and connect them using a path defined by a Bezier curve. This process creates a graph, with the selected points serving as vertices and the curves serving as edges. These vertices contain information about the forklift's position and final orientation when it arrives. On the other hand, the edges contain information about the start and end nodes and the speed at which the forklift should travel the route. So, for autonomous driving, the system only requires receiving the node's id defined on the map. Then this operation's result is known by listening to a topic that shows the completeness of the path.

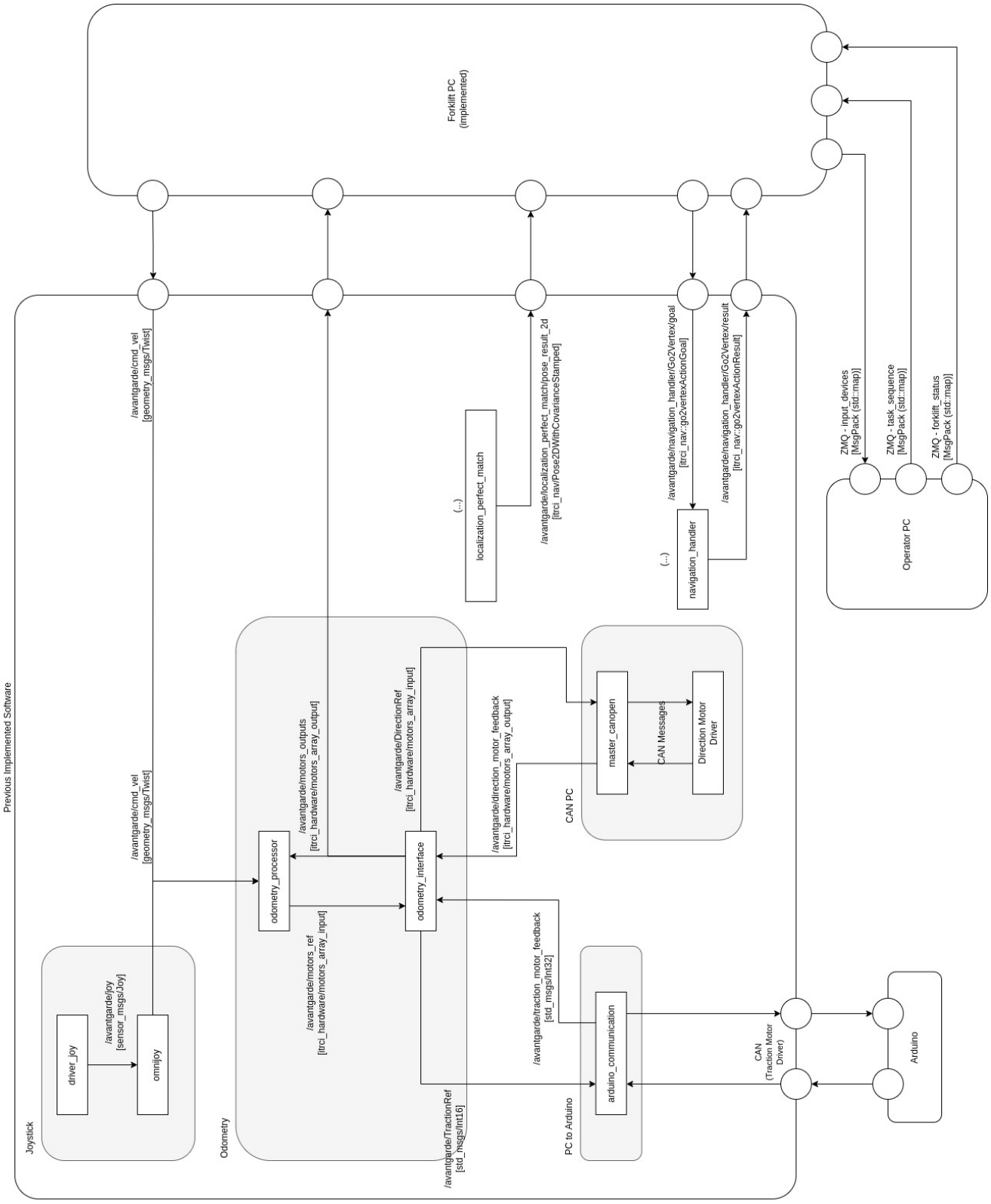


Figure 4.14: Software already developed in the forklift's PC

4.2.2.2 Implemented Software in Forklift's PC

Although the developed software allowed forklift control in both modes of operation, it required some modifications before its use in a high-level teleoperation system. In Figure 4.15, there is a representation of the implemented nodes and their interaction with the other system's components.

As a result, creating a node that could receive data and control the forklift in one of these two modes of operation was necessary. The creation of the *commands_handler* node serves for this purpose as it receives information compressed using the MessagePack compression method via the ZMQ protocol. This node has implemented a subscriber that listens to a specific topic (the vehicle name) and thus ensures that it only performs operations if the vehicle is selected. This compression makes the message lighter, allowing for faster transmission than the JSON method. Thus, after the message's decompression, it is determined the desired mode of operation and the instructions on how to operate: either the commands in remote control mode or a sequence in autonomous mode. Depending on this mode, some nodes may intervene as opposed to others, as illustrated in Figure 4.16. In remote mode, the nodes *ArduinoForkHandler* and *odometry_processor* are the ones that communicate with *commands_handler*. On the other hand, in the autonomous mode, the node used is the *picknplace_handler*, which communicates with both the *navigation_handler* and *move_fork*.

Regarding the remote control mode, the message sent contains information about the operation mode and the actions performed on the input devices. This data is stored in a data structure (map) with keys corresponding to the type of information and the values stored in those keys. Table 4.1 displays the various types of information and their possible values.

Table 4.1: Contents of the message used to send the operator commands in remote control mode

Category	Description	Possible Values	Converts to
mode	mode of operation	1 - Auto, 2 - Remote	none
gas	value of the accelerator pedal	0 to 1	0 to 0.2
brake	value of the brake pedal	0 to 1	0 to 1
wheel	position of the racing wheel	-1 to 1	-pi/2 to pi/2
lift	value of the joystick movement	-1 to 1	(- 1 to 1) * 0.2 * 200
moving_direction	steering wheel direction	1 - Forward, 2 - Backward	none
wheel_state	if the start button was pressed	0 or 1	none
is_wheel_alive	if the clutch is pressed	0 or 1	none
is_joystick_alive	if the trigger is pressed	0 or 1	none

After receiving these values, they must be converted into messages suitable to those the nodes expect to receive. Thus, the steering wheel position and acceleration pedals values are converted into linear and angular velocities and included in the message published in the topic */avant-garde/cmd_vel*. Now that the *odometry_processor* node, as previously described, has subscribed to this topic, it will initiate the execution of the set of tasks required to place the forklift at the desired speed and orientation. The received joystick position value, which is responsible for fork movement, is also converted using a scale factor defined by the operator and a maximum velocity value and sent to the *ArduinoForkHandler* node, which translates the CAN messages exchanged

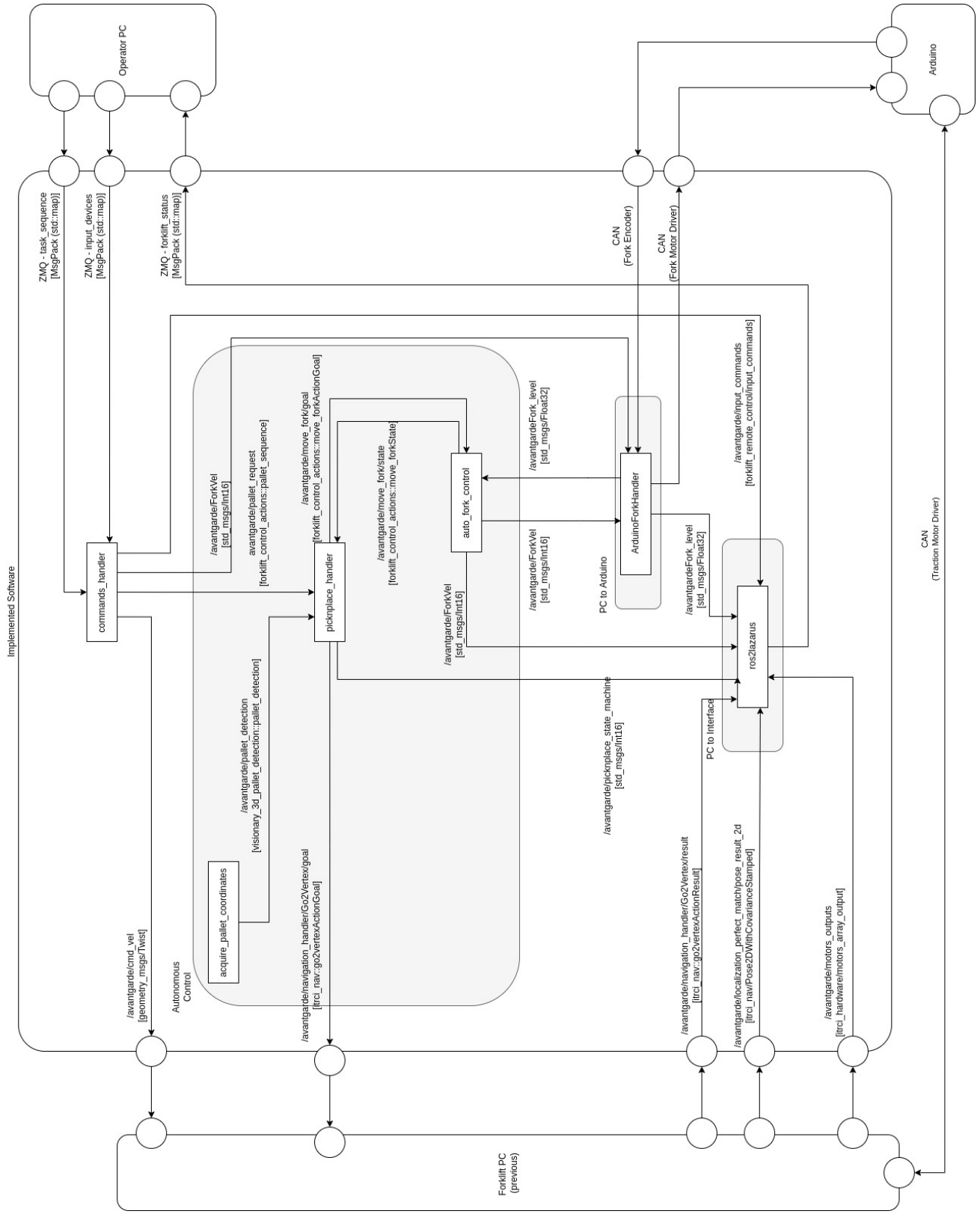


Figure 4.15: Software implemented in the forklift's PC

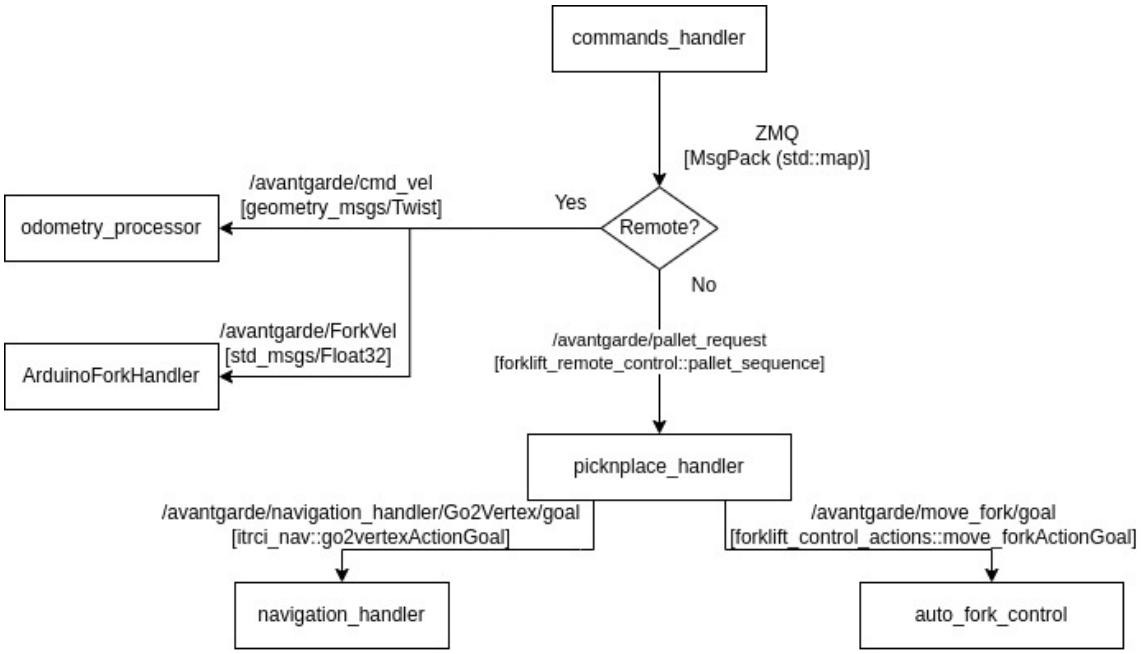


Figure 4.16: ROS node selection given the forklift's intended mode of operation

with the Arduino. When the *ArduinoForkHandler* node receives this message, it will convert the desired speed into CAN messages sent to the Arduino.

One aspect to consider is that, after experimental verification, there was an offset in the desired values to begin the fork's movement. So, after finding out this value, it had to be added to the conversion's calculation. Thus, the CAN message sent to the Arduino contained four bytes:

- an id that identified the information in the message
- a code that represented the message size
- a byte that indicated whether it was a positive speed or not
- a byte that indicated the desired speed

Regarding the autonomous mode of operation, the message received is different. It includes the intended mode of operation and two positions corresponding to where to pick and place the pallet. A Yet Another Markup Language (YAML) document maps these two positions, each corresponding to a node id (used for autonomous navigation), a height value corresponding to the height of the pallet in the warehouse, and a variable related to the intention of adjusting the forklift orientation towards the pallet. The *commands_handler* node converts the received data into a message that specifies the pick and place nodes and the heights at which the forks should be placed and sends it to the *picknplace_handler* node. This node manages all the operations required for the pick and place task. It has two state machines whose transitions are triggered based on the state of the various operations. A pick operation, as represented in Figure 4.17, must first ensure

that the forklift does not move until the forks are on the ground. If that is not the case, there is a request to lower the forks, as moving a forklift with the forks raised is considered dangerous.

After this check, the forklift will move to a node near the pallet. Then, upon receiving confirmation that it has arrived, the operator can request the forklift to correct its orientation so it aligns with the pallet. If that is the case, the forks need to be raised to a height near the one received so that the pallet sensor can be at the pallet's level. Following this, the orientation correction algorithm is run based on the detected pallet's coordinates, received from the *acquire_pallet_coordinates* node developed to communicate with the pallet sensor and extract this information. After aligning with the pallet, the forklift may raise the forks to the pallet's height and move to the node mapped beneath the pallet so that, after inserting the forks beneath the pallet, the forklift can lift the entire pallet. When the pallet arrives at this node, it performs a slight climb to lift it. After confirming this step, it returns to the previous node, where it lowers the forks to the ground level and completes its task when it receives confirmation that they are down.

In the place operation, the steps are similar, as depicted in Figure 4.18, except for a skip in the orientation check step. This difference is due to the autonomous navigation system that already assures the travel of a path with a defined final orientation. Furthermore, when it is in the pallet's node, the pallet moves down slightly rather than up as in the pick operation. Another difference is that the forks start at a minimal height that allows not to place the pallet on the floor; otherwise, the pallet would not move with the forklift.

These operations require continuous communication with two nodes: *navigation_handler* and *auto_move_fork*. As previously stated, the former node is in charge of autonomous navigation and only receives the id of the desired vertex (which is on the map). In contrast, the latter allows moving the forks to a given height autonomously, using a Proportional Derivative (PD) controller, whose parameters are configurable with a YAML file.

The *auto_move_fork* node checks the forks' height value at a 50 ms rate to calculate the position error, calculates a speed to correct it, and sends it to the *ArduinoForkHandler* node. This last node also reads the CAN messages sent by the Arduino regarding the forks' height, converts them into a message, and sends it to the *move_fork* node.

After selecting the operation to be performed, the operator must receive feedback about the instructions it gave. As a result, it is necessary to have a node that can send relevant information about the forklift's state to the interface and feedback, e.g., on the remote control, of the commands it has sent. The *ros2lazarus* node communicates with multiple nodes to obtain this feedback and sends it via ZMQ in a compressed message. Table 4.2 displays the message's contents.

4.2.2.3 Implemented Software in Forklift's Arduino

Besides the software developed on the forklift's PC, the Arduino code also had some additions. By already having the traction motor control implemented, it only required the development of the control of the forks' motor and reading the data received by the fork encoder.

There is a single CAN bus to which the frames relevant for controlling and reading the hardware present in the forklift are sent and picked from. The software available in this Arduino

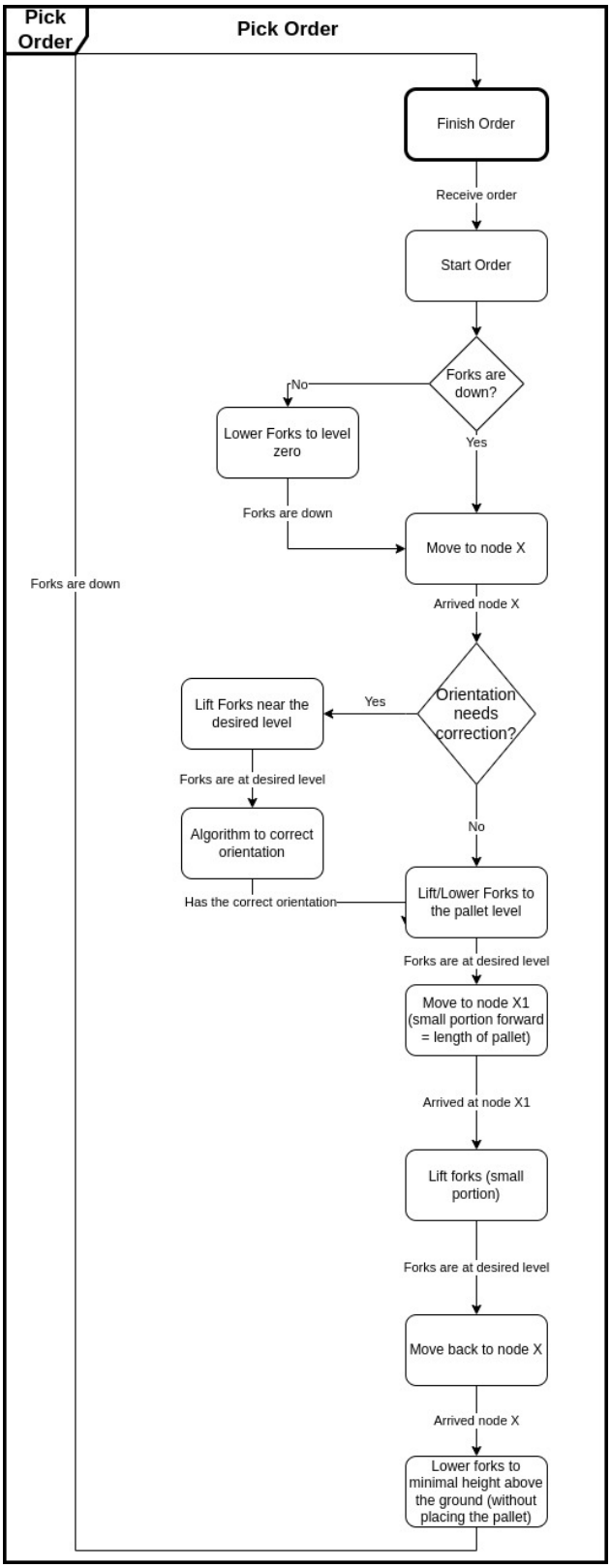


Figure 4.17: Pick order state machine

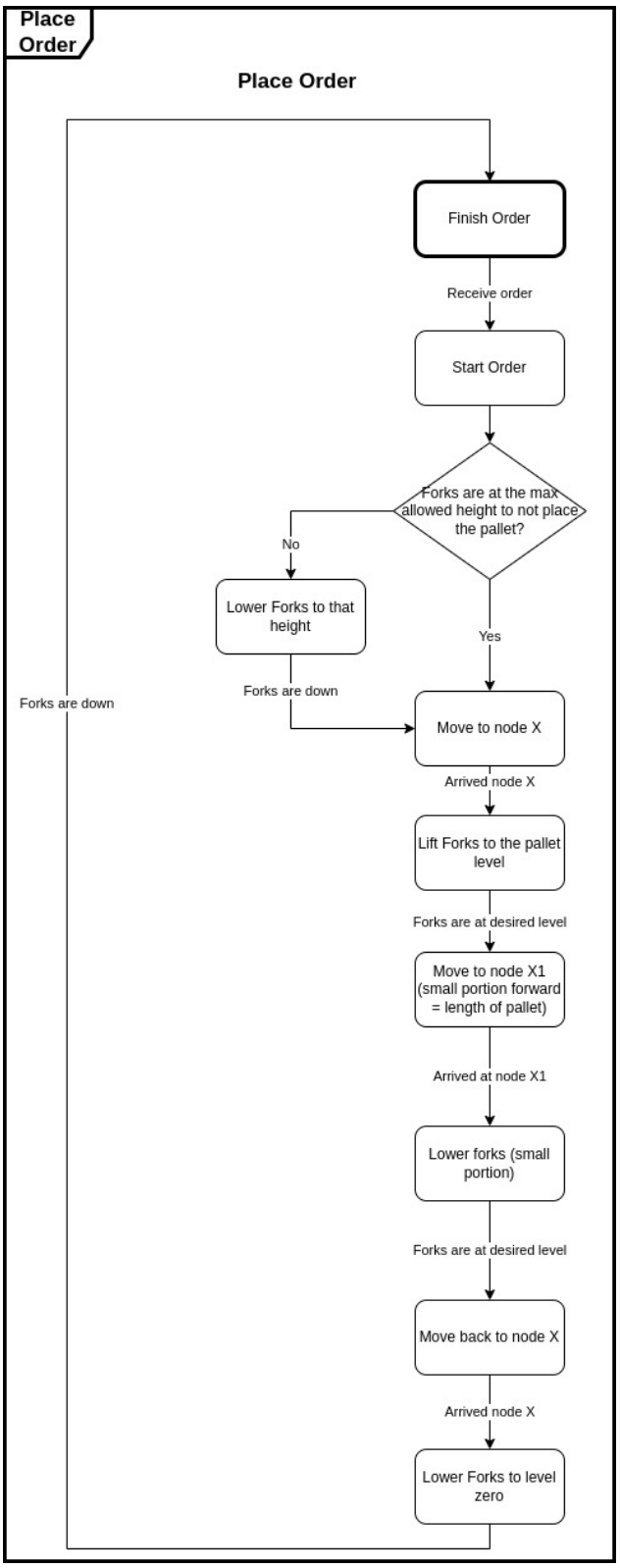


Figure 4.18: Place order state machine

Table 4.2: Contents in the feedback message sent to the interface

Parameter	Node
vel_linear	*odometry_interface*
vel_angular	*odometry_interface*
vel_lift	*move_fork*
pos_fork	*ArduinoForkHandler*
mode	*commands_handler*
gas	*commands_handler*
lift	*commands_handler*
brake	*commands_handler*
wheel	*commands_handler*
moving_direction	*commands_handler*
wheel_state	*commands_handler*
is_wheel_alive	*commands_handler*
is_joystick_alive	*commands_handler*
pose_x	*localization_perfect_match*
pose_y	*localization_perfect_match*
orientation	*localization_perfect_match*
state_feedback	*picknplace_handler*

allowed to control the linear speed and the forks' movement via the keyboard. This way, it was only necessary to check the commands sent to the fork motor and adapt the information. In this manner, the Arduino receives the speed value from the forks and converts the message into a CAN frame containing the address for the up or down function and its speed. Regarding the fork encoder's data, the Arduino receives this data via the SSI protocol, converts this measure, puts it in a CAN message, and sends it to the CAN bus. Figure 4.19 represents the functions performed by the forklift's Arduino.

4.3 Remote Operation Center

After the adaptations made to the forklift to be suitable in a high-level teleoperation system, each vehicle in this system must know what information to receive for its autonomous or remote control.

4.3.1 Autonomous Operation

Some requirements must be satisfied for the operator to instruct the forklift to perform a task autonomously. The forklift PC must be prepared to receive ZMQ messages and decompress them to determine the pallet's initial and final positions. Furthermore, the forklift must already have a sequencer capable of performing all necessary steps, such as moving autonomously between two points (previously mapped in the forklift's map) and precisely positioning the forks. To ensure this autonomy, safety systems capable of preventing accidents, such as a laser scanner system, and a mapping of the upper and lower limits of each pallet's compartment based on fork encoder measurements, are required.

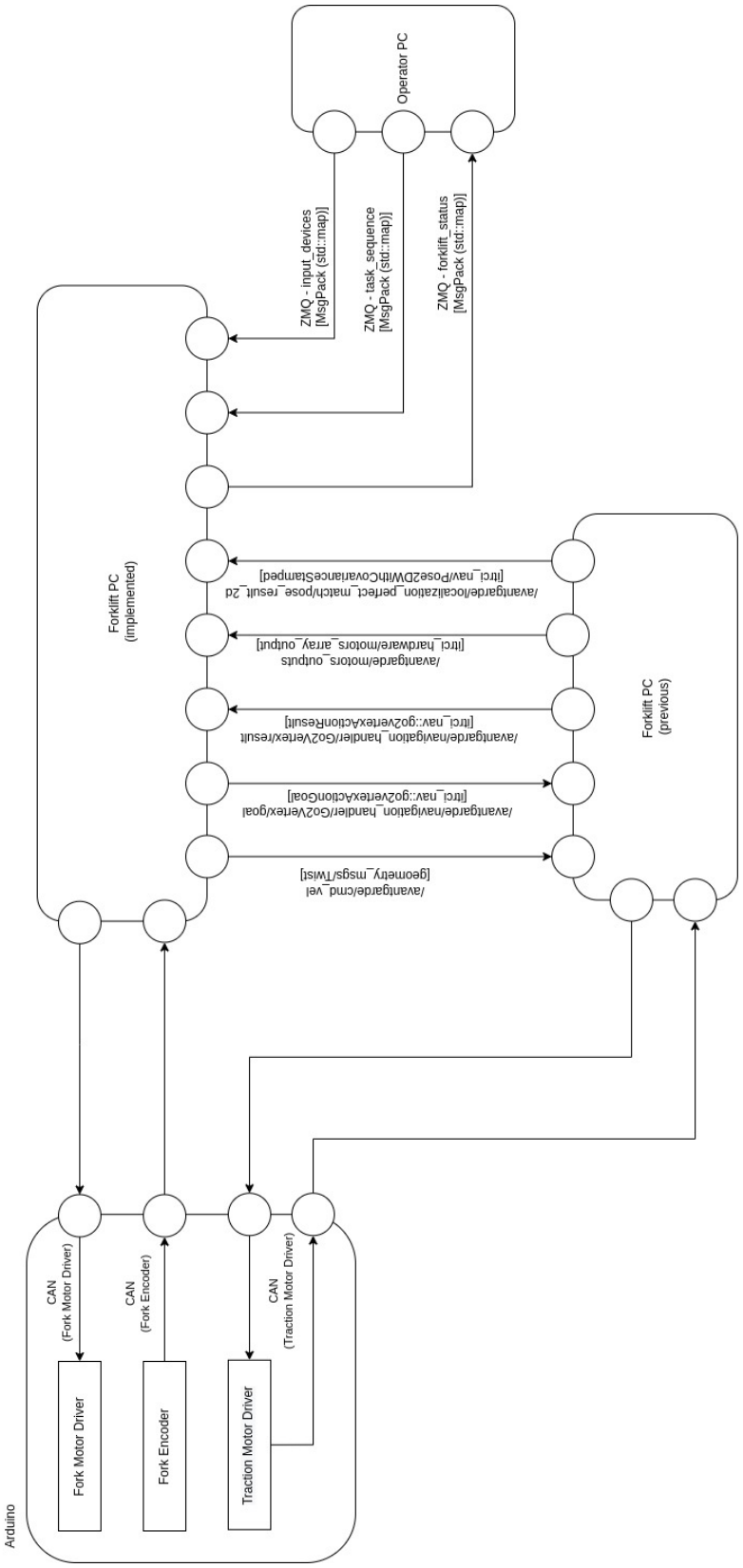


Figure 4.19: Representation of the software implemented in the forklift's Arduino

As a result, when performing the pick or place operation, the forks do not move more than they should. To help with this task and, if desired, correct the forklift's orientation with the pallet, the computer must be able to communicate with a pallet sensor to extract relevant information about the pallet detection and, if desired, display it to the operator at the interface.

Thus, having a vehicle with these functionalities implemented allows the operator to request and see a pick and place task executed autonomously, along with an interface that allows seeing the cameras' images and the information relevant to the task he wants to perform.

4.3.2 Remote Control

To remotely control the forklift, the controllers should be similar to those used by the operator in person, and there should be cameras to allow him to see what he is doing. Two cameras provide two views to aid the forklift's remote control: one as if the operator was on the forklift and another that provides a more general view of the entire environment. If something does not go as planned and the operator needs to stop the forklift when there is no one around, he can use the remote emergency button to avoid accidents.

Regarding how the operator remotely controls the forklift, although using a joystick, similar to a PlayStation controller, allows to control the forklift remotely, it does not resemble the actual forklift control. As a result, using a Racing Wheel and Pedals seemed a more suitable option as they are widely used in simulated driving systems and, therefore, making a smoother transition to this type of control as they are closer to the actual system. Regarding the forklift's forks, the operator could lift or lower the forklift's forks by using the joystick's push or pull movements.

4.3.2.1 Racing Wheel and Pedals

Regarding the forklift's steering and linear speed control, the Logitech G29 offered handy features such as several integrated buttons and three pedals, like a car: clutch, brake, and accelerator.

In the first step, using the *jstest-gtk* application allowed the visualization of the mapping of each button and axis to identify them when reading the steering wheel data. Reading this input device was possible due to the ROS node *Joy*, which could convert the information it gave. This node's output were values of 0 or 1 for buttons and values between -1 and 1 for axes (racing wheel and pedals). After implementing the reading process, the need to develop some safety measures to prevent accidents arose. For that reason, sending data is only possible when the steering wheel is "turned on" i.e., toggling the starting button to allow the reading of this data. Following this action, the steering wheel data is read when the clutch pedal is pressed, working as a deadman switch. Thus, while pressing the pedal, the steering wheel's turning allows the forklift's direction to change, and pressing the accelerator pedal changes its speed. When releasing the clutch pedal, the direction value remains at the last value given, and the speed remains at 0 until both clutch and accelerator are pressed again, immobilizing the forklift in both speed and direction. This implementation requires the operator's movements to be intentional and reduces the risk of accidental movements.

This driving wheel also allows to change the forklift's speed direction, i.e., whether the forklift moves forwards or backward. The automatic gear shift handles on the steering wheel are used for this: the left one for forward and the right for reverse. Figure 4.20 depicts the features implemented in this input device.

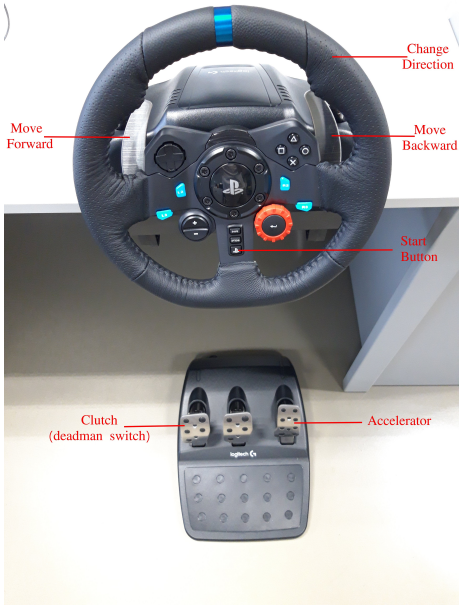


Figure 4.20: Racing wheel and pedals and its functions

4.3.2.2 Joystick

Regarding forklift's forks control, the choice of a joystick similar to the one used in flight simulation systems seemed suitable as it resembles the lever used to lift/lower the forks. It has several buttons and three axes: one rotational and two translational, with only one used for fork movement. The push movement raises the forks, and the pull movement lowers them. Similar to the previous input device, the same application allowed the visualization of the mapping of the axes and buttons and the same ROS node for reading the data.

On the safety measures, this controller also assures the intentionality of the movements. As a result, the *trigger* button ensured that the operator intended to raise and lower the forks, implying that if he stopped pressing it, the speed values would remain at zero. Figure 4.21 presents the features implemented in this device.

4.3.3 User Interface

In high-level teleoperation of a vehicle, it is essential to have an interface that can communicate with each vehicle and supervise it using the images captured by the cameras.

There was already an application developed using Lazarus, an application development environment, to receive the cameras' images. As a result, it seemed opportune to use this application



Figure 4.21: Joystick and its functions

and expand it to include the features required for the high-level teleoperation system. Thus, the developed interface now has a set of pages with a specific function that the user can easily navigate using the buttons at the top of the application.

The *Configurations* page, shown in Figure 4.22, allows the user to change the addresses of the vehicles and cameras they want to connect.

The addresses, previously defined in a configuration file, are displayed, and the user can edit them while on this page. After confirming the addresses, the user can connect to them by clicking the *connect* button. Next to each address, there is a circle that indicates the connection status: red for disconnected and green for connected. As presented in the software adaptation section, this application employs the publisher-subscriber communication pattern because the camera's images were obtained by subscribing to the topic each camera was publishing. The application subscribes to three addresses and publishes on its own. It must connect to the vehicle's computer, which corresponds to the first address, to receive vehicle status information such as pose, fork height, and linear and angular velocities. The following addresses refer to the cameras used to view the vehicle: the one installed on the vehicle and the environment camera. Finally, by pressing the *connect* button next to the last address, the user allows sending the data to the vehicle assigned for the task.

The *Task Assignment* page, depicted in Figure 4.23, is intended to supervise the vehicle's task execution. It is possible to switch between the available cameras: the environment and the one installed in the vehicle, and the vehicle's operation mode: remote or autonomous. A click on the *remote control* button takes the user to the remote control page. In contrast, in autonomous mode, before executing a task, it needs to meet some requirements such as the vehicle selected and the pallet's pick and place positions. The autonomous pallet's pick and place operation requires the

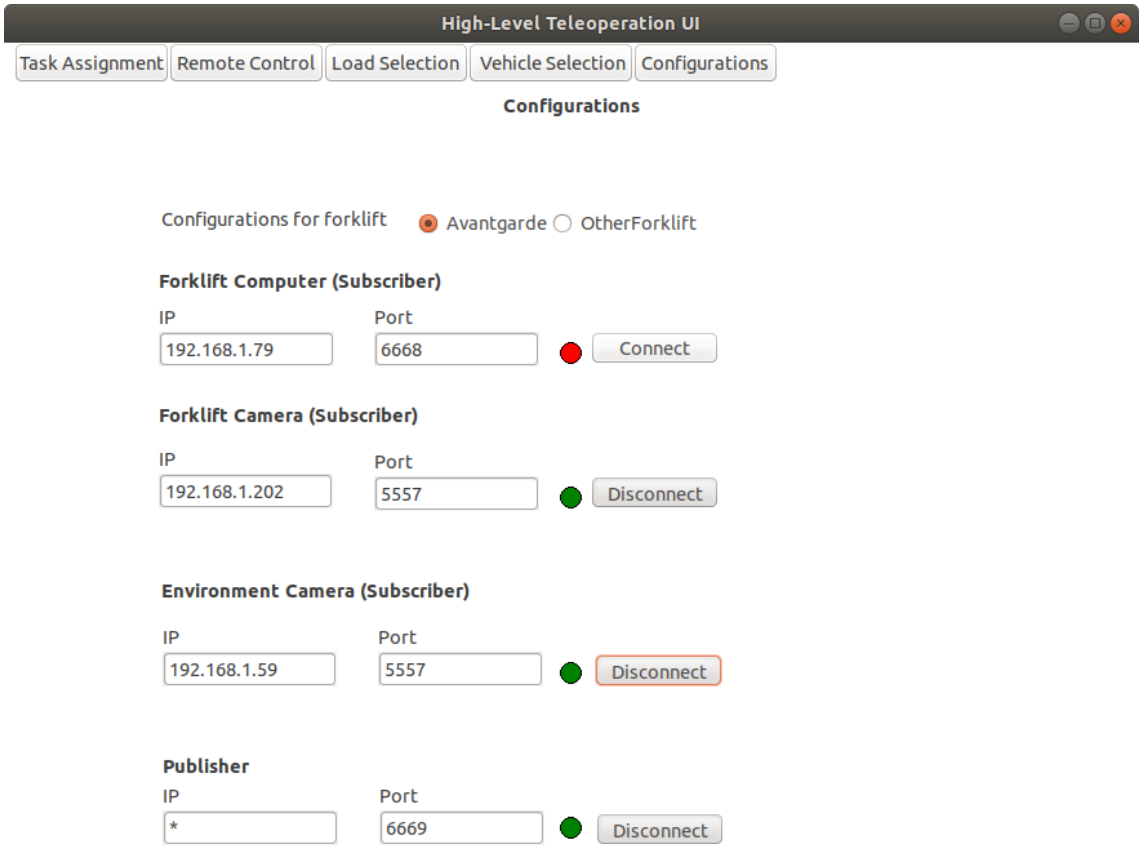


Figure 4.22: Configurations page of the user interface

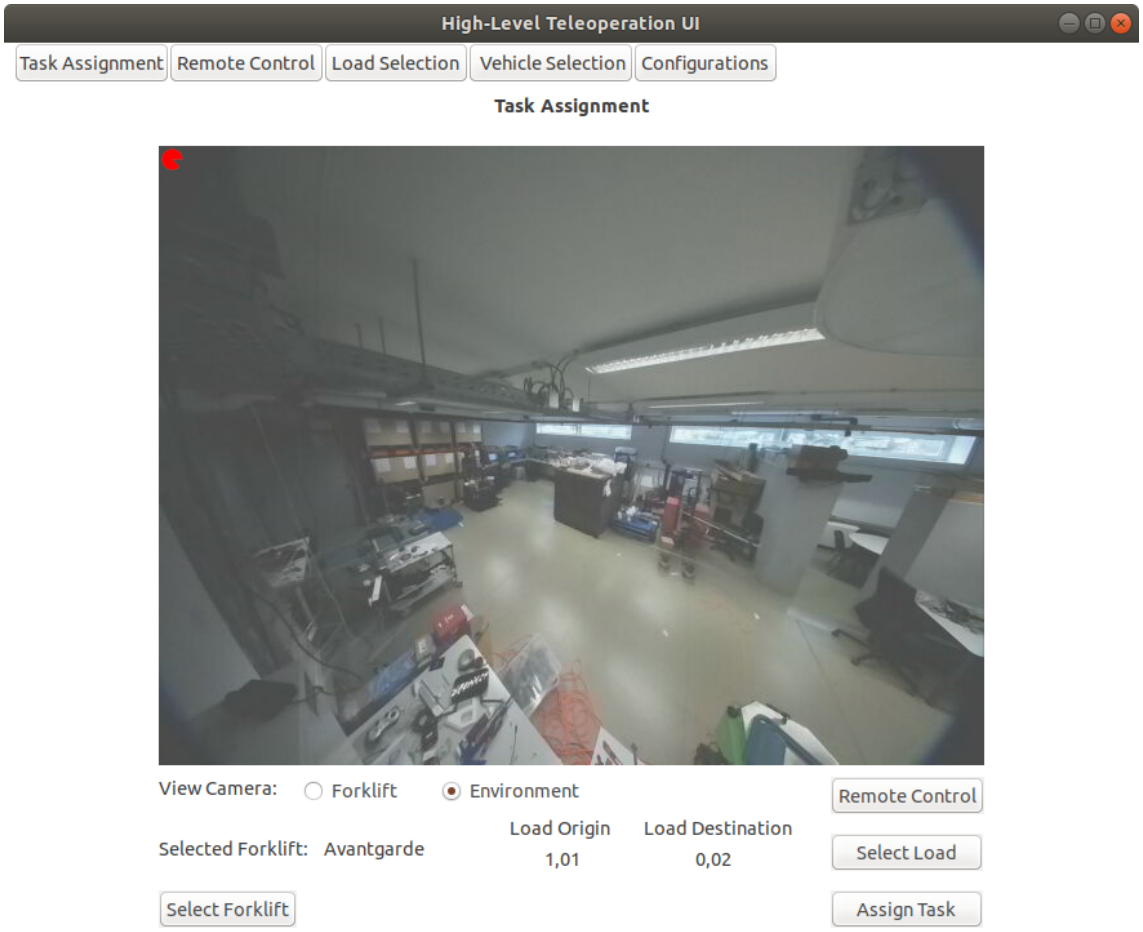


Figure 4.23: Task assignment page of the user interface

selection of the pallet's start and end positions and the vehicle that will perform this task. This application can receive information about multiple vehicles; however, to avoid diverting the user's attention and increasing communication traffic in the interface, it was decided to display only one vehicle's information at a time. Nevertheless, this functionality could be added in future versions if more vehicles operate in this teleoperation mode.

As a result, if the user clicks on the *Assign Task* button without first selecting the pallet's positions, the user is directed to the *Load Selection* page, where an interface is presented for the user to select the positions available for the movement he intends to perform. After fulfilling this requirement, the user is directed to the *Vehicle Selection* page to assign the vehicle to the newly created task. After selecting the vehicle, the user is taken to the *Task Assignment* page and sees created task's data, specifically the pick and place positions of the pallet and the vehicle in charge. As a result, whenever the user wants to begin the execution of the task, all it requires is to press the *Assign Task* button to send the data to the selected vehicle.

The *Remote Control* page, depicted in Figure 4.24 is for vehicle remote control.

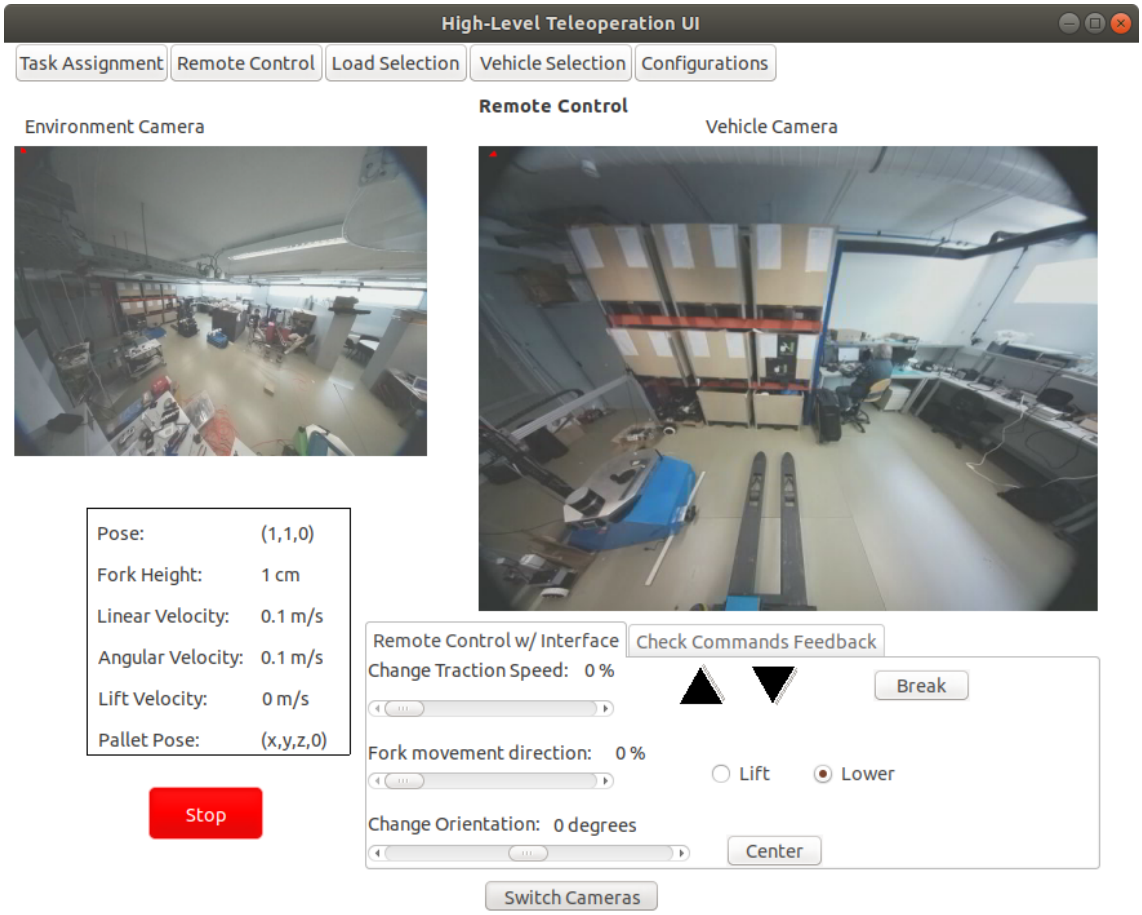


Figure 4.24: Remote control page of the user interface

Aside from the vehicle information, there are two cameras to choose from for vehicle control

by clicking on the *Switch Cameras* button, which changes the arrangement of the cameras, highlighting one of them. Furthermore, it is possible to control the forklift remotely using only the interface's controls. To do so, the user must press the *Start* button on the remote control, and when it wants to stop this type of control, the user should press the same button again, labeled now as *Stop*. As a result, even if the user's input devices fail, he retains control of the forklift. It has two scrollers for adjusting the linear and fork speed and another for adjusting the steering wheel position.

On the *Load Selection* page, as represented in Figure 4.25 the user can select which pallet to remove and where to place it.

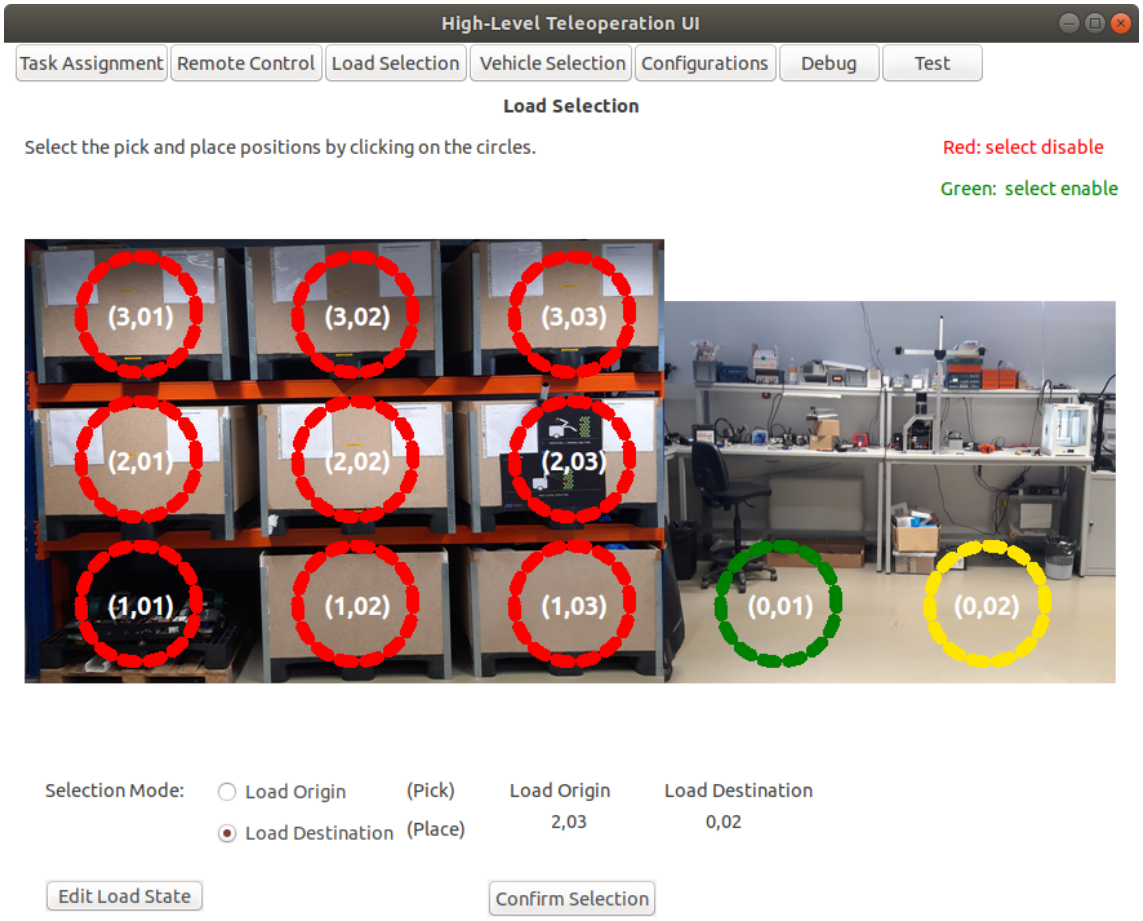


Figure 4.25: Load selection page of the user interface

In each pallet, a circle can have a red or green color, representing its selectability for the intended task to avoid some situations, and yellow indicates the current selected pallet. For example, if the user wants to select the pallet he wants to remove, he cannot do it in a place where there is no pallet. Similarly, when choosing a location for the pallet, the user cannot put it in an already occupied place. In addition, the user can change the warehouse state by clicking the *Edit Load State* button. After this click, the user can change the states of which pallets or locations are available

for each operation by simply clicking on them. Each location has an associated coordinate to keep track of its location, with the first number before the comma representing the row and the second representing the column. With two digits in the column field, the application can run the same algorithm if there are more than nine columns. Since there is a conversion of all values to float types, a task required by the type of map used to store the values, using only one digit would result in ambiguity because columns 1 and 10 would be the same. After finishing the editing process, the user can confirm the pallet and location selection and then proceed to the *Vehicle Selection* page or the *Task Assignment* page, depending on the status of the requirements for creating the task.

Chapter 5

Results

Following the development of the high-level teleoperation system, it is necessary to validate the forklift's modes of operation and check if the interface used to operate the forklift worked properly, allowing to control and visualize the forklift. As a result, this chapter will address the results obtained with the interface, remote control, and autonomous mode.

5.1 User Interface

Regarding the interface, it was confirmed that it had the implemented functionalities, including being able to:

- visualize the images from both cameras
- remotely control the forklift
- request a pick and place operation
- obtain info about the forklift's status

5.2 Remote Control

Regarding the remote control, confirming the developed functionalities and determining the delay in the image and commands transmission is necessary. As a result, the users asked to check the remote control functionality confirmed that they could control the forklift's steering using the racing wheel, the forklift's speed using the pedals, and the fork's height using the joystick. This mode of operation can be seen in some videos that were recorded. The video available at [40] shows the operator's actions and the interface side by side, whereas the video at [41] presents the view of the pallet sensor interface.

As for the delay, it is essential to know the delay between an action performed by the operator and when he sees it in the interface. There are mainly three delays in this operation:

- The delay in the image transmission between the cameras and the interface - visualization delay
- The delay in the command's transmission between the operator's PC and the forklift's PC - commands transmission delay
- The delay between the reception of the orders from the forklift's PC and its execution - actuation delay

An alternative solution to check the first delay had to be found, as the program running in the cameras should remain unchanged. This solution consisted of having a website that had the current time with a precision of milliseconds, showing it to the camera, and comparing it to the one displayed on the operator's PC, being both clocks synchronized over the internet. Although this was not a really precise way to measure the delay as there is a sampling factor, it could provide a rough estimate of this value. As depicted in Figure 5.1, this delay was around 160 ms, which proved suitable for the forklift's remote control.

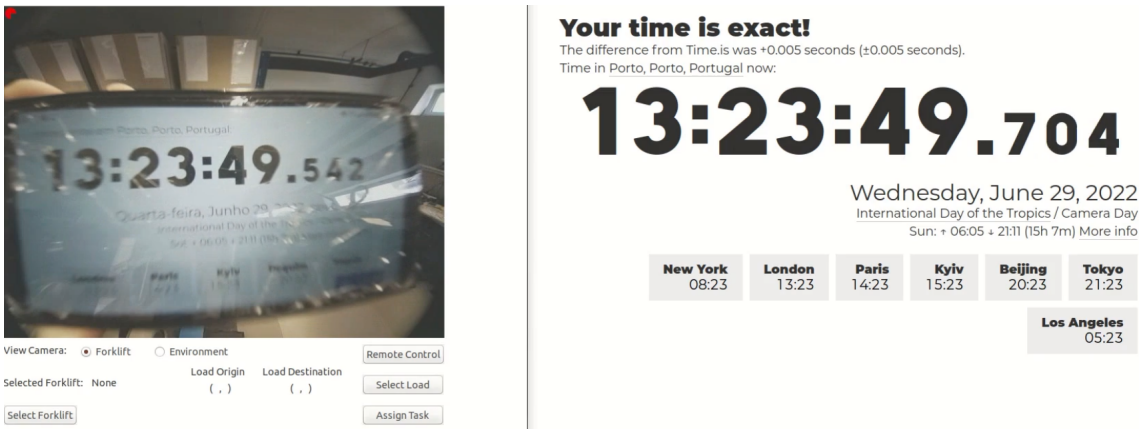


Figure 5.1: Delays in the image transmission from the camera to the interface

The delay in transmitting the commands included the total time from the message's compaction until the forklift's PC unpacked it. As depicted by the measurements in the graphic of the Figure 5.2, apart from some spikes in latency, this delay remained below 10 ms, having an average of 5ms.

Finally, for the actuation delay, the way used to find this delay relied on the forklift's unlocking mechanism. As the forklift makes a sound every time it moves (due to its unlocking), the solution consisted of measuring the delay between two sounds: the click (if controlled by the interface) and the forklift's unlocking, as it can be seen in Figure 5.3.

The measured delay was near 260 ms, which leads to a total delay since the operator performs an action and sees it executed using the interface of 420 ms.

When testing the operator's experience in driving the forklift, the operators had to perform a pick and place task that required using all the controls available (direction, speed, fork's speed)

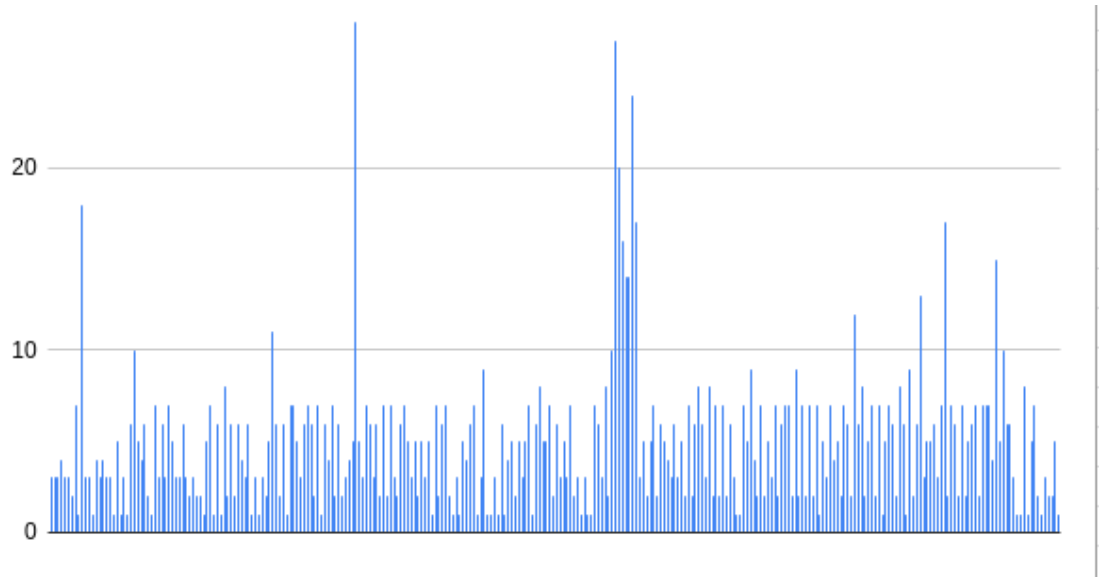


Figure 5.2: Delays in the commands transmission from the forklift's PC to the operator's PC

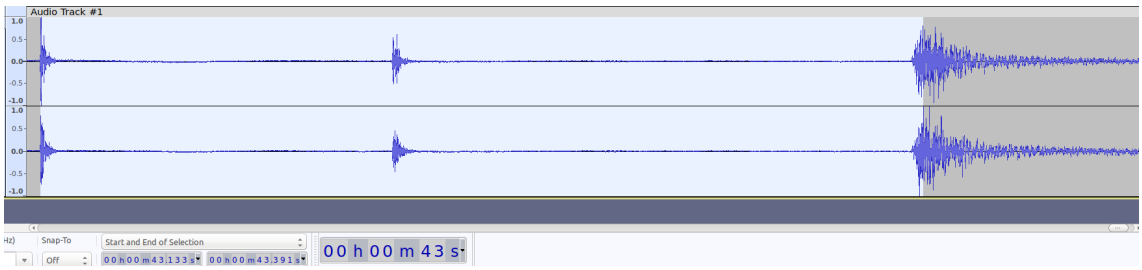


Figure 5.3: Delays in between the click and the forklift's unlocking to start to move

and the images captured by the cameras without seeing the forklift directly. The NASA Task Load Index (NASA TLX) is a widely used tool for determining the effectiveness of a task by assessing the workload felt by the operator, Table 5.1 shows an example of a questionnaire to be filled by each operator. Figure 5.4 depicts the operators' responses and in Figure 5.5 is presented their mean values.

Table 5.1: NASA TLX quesionnaire to be filled by the operators after performing a pick and place task using the remote control functionality

Category	**Question:**	**Response (0-20)**
Mental Demand	How mentally demanding was the task?	
Physical Demand	How physically demanding was the task?	
Temporal Demand	How hurried or rushed was the pace of the task?	
Performance	How successful were you in accomplishing what you were asked to do?	
Effort	How hard did you have to work to accomplish your level of performance?	
Frustration	How insecure, discouraged, irritated, stressed, and annoyed were you?	

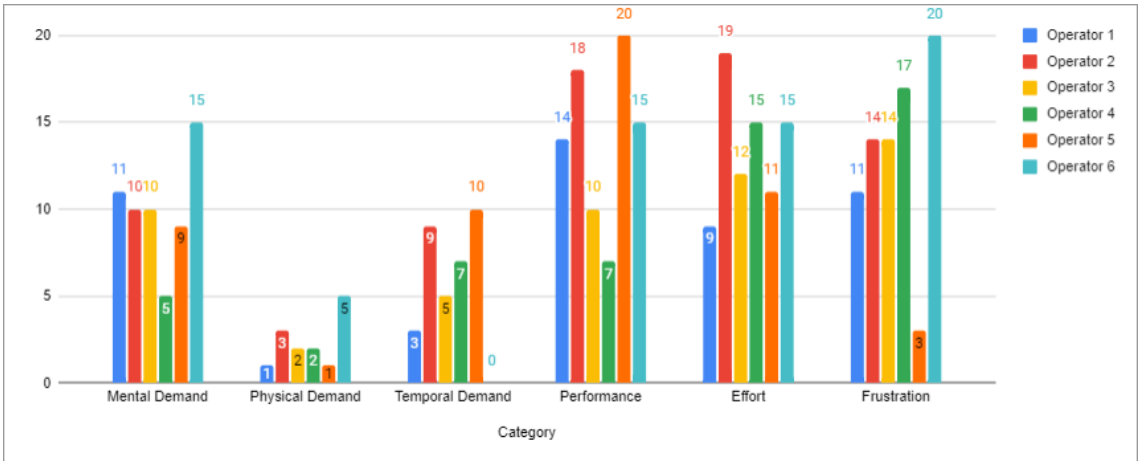


Figure 5.4: Operator's responses to the NASA TLX questionnaire on the workload felt during the remote control experience

As can be seen, the operators performed reasonably well even though they had little experience driving a forklift. Aside from that, as the operator is not used to controlling the forklift without directly seeing it, he is more cautious with his driving.

Driving a forklift is different from driving a normal vehicle since it is required to pay attention to the load it carries. As a result, the traction driver of the forklift has some acceleration and deceleration ramps that prevent sudden stops or accelerations. Thus, when performing a movement, the operator must take these ramps into account to safely control the forklift.

According to the obtained results, the operators did not experience a significant physical or mental strain; however, they did appear to feel some level of effort and frustration while controlling the forklift. Since the operators heavily rely on the images given by the cameras, it is difficult for them to have the same perception as if they were driving in person. Thus, if there were more

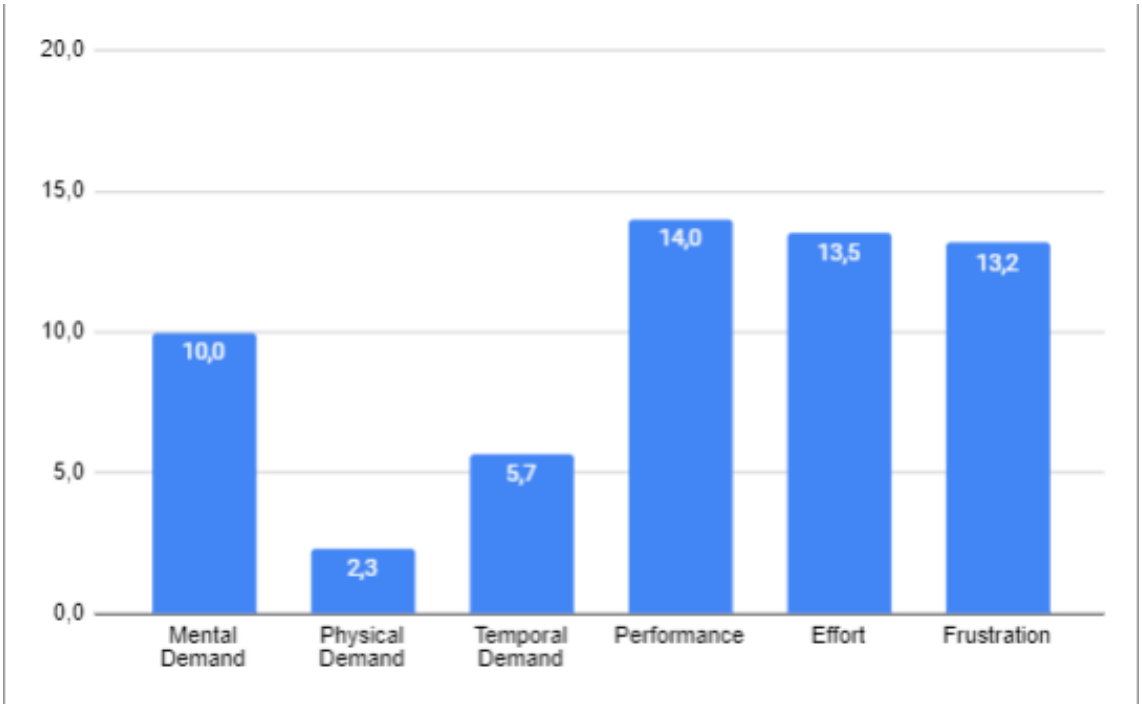


Figure 5.5: Average of the operator's responses to the NASA TLX questionnaire on the workload felt during the remote control experience

cameras with different perspectives, for example, to know how far away from the pallet he was, and a view from the side to check if the forklift was aligned with it, these two values could be reduced.

Despite this, the operators could remove and place the pallet in the desired locations, albeit with minor deviations in the pallet's orientation concerning the structure where it is located. According to the operators, vehicle motion and fork control were smooth and responsive, which increased their confidence in the system and its performance. Finally, some operators' inexperience in driving this vehicle may have impacted their effort and performance, making the task more difficult. Nonetheless, because these tests were conducted in the first contact with this control mode, it is expected that the effort and frustration values would decrease with habituation to this system and driving this type of vehicle.

5.3 Autonomous Pick and Place Operation

When performing an autonomous pick and place task, two factors must be considered for the operation's success: autonomous navigation and pallet detection, if the orientation needs to be corrected.

5.3.1 Pallet Detection

First, the pallet sensor must be studied to determine under which conditions it can detect the pallet. Therefore, one way to know its capabilities is to test it with the pallets in the pallet rack to see how far this sensor can detect the pallet and the maximum allowed deviation from the sensor before it stops detecting it. This measure is important because, while the orientation of the pallets should be consistent in the warehouse, it may not be the case when they are on the floor and outside the pallet rack. As a result, if the pallet deviation from the camera is higher than the camera can detect it, the forklift will be unable to align itself with the pallet and, as a result, will be unable to insert the forks and lift the pallet.

According to the results, the distance from which the sensor detected the pallet varied, resulting in it not having a continuous detection or detecting the wrong pallet. The deviation, however, depended on the distance and was also variable. This detection required changing parameters such as the integration time or the maximum search height, lifting/lowering the forks, or approaching or distancing itself from the pallet. Besides, the experiments showed that parameters such as the fork's height or distance to the pallet could change depending on the previous detection. Thus, a more in-depth sensor study is necessary to check the optimal parameters for greater pallet detection.

5.3.2 Autonomous Navigation

After the analysis made of the pallet sensor, it is necessary to evaluate the autonomous navigation of the forklift as there may be a cumulative error in this navigation due to errors in the laser scanning, odometry, orientation, and the maximum allowed error in the trajectory controller. Through the tests performed, the error in the autonomous navigation did not impact the performance of the assigned task.

Finally, it was decided to test the operation of the sequencer responsible for performing the pick and place task, which, depending on the task, requires both the pallet sensor and the autonomous navigation to function properly.

Given the results of the pallet sensor, the task designed to test this operation did not include correcting the orientation, relying, instead, solely on the fork's height and the destination's orientation to where the forklift should go. According to the results obtained, the sequencer meets the intended requirements as it was able to pick and place the pallet from a higher to a lower position in the pallet rack and vice-versa.

There was a video recording of the task being performed. The video at [42] shows two different perspectives on the completion of a task. Although these perspectives do not correspond to the same task because they were recorded at different times, after synchronizing them, it is clear that the forklift behaves the same way in both videos. The video available at [43], on the other hand, shows the required steps to create the task and allows the operator to visualize the task's execution using the interface.

5.4 Discussion

The implemented functionalities for remote control proved to be functional, allowing the operator to drive the forklift remotely using only the images from the cameras. Regarding the performance of the remote control, the latency of 420 ms had no impact on the operators' driving at lower speeds and they could successfully execute the assigned task, requiring some effort given the operator's inexperience in this type of control.

The autonomous operation mode was also validated and proved capable of autonomously performing the task of picking and placing a pallet. Given the pallet sensor's characteristics, it seemed better to ignore the sensor's detection since it could affect the performance of the task when false or wrong detections occurred.

Chapter 6

Conclusion

6.1 Outcomes

During the work of this dissertation, several objectives were achieved, such as:

- Adaptation of the forklift to allow its control in a high-level teleoperation system
- Development of an interface through which the operator sends tasks to multiple vehicles, monitors their execution, and intervenes when necessary, controlling it remotely
- Development of a remote control system for a forklift truck using a racing wheel and a joystick
- Development of a sequencer that allows for the autonomous execution of pick and place tasks
- Publication of the paper "A survey of high-level teleoperation, monitoring and task assignment to autonomous mobile robots." at IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC) [44]

6.2 Challenges

During the execution of this work, there were some challenges to overcome. Some of them took more or less time, depending on the method used. For example, the time required to install the encoder would be shorter if the first version of the circuit which read the encoder had already included the Max490 integrated circuit to convert the differential transmission mode. Furthermore, further analysis of the software present in the Arduino could have reduced the duration of this task by avoiding conflicts in the use of resources by this microcontroller. As for the remote control development, starting with trying to read the input devices on the Windows operative system turned out to be way more challenging than on Linux. Reading the data through Windows complicated the process, leading to a change of operating system to complete the task successfully. In addition,

it would have been advantageous to have previously studied the pallet sensor's characteristics to integrate it into the forklift's autonomous task execution easily.

Furthermore, the execution of this work led to the use of a vast spectrum of tools such as:

- Use and installation of hardware
- Development of electronic circuits
- Development of firmware for the Arduino
- Use of software tools such as ROS, with programming languages like C++ and Python
- Development of compatible interfaces with the Pascal programming language through the Lazarus application development environment

The multidisciplinarity involved in this dissertation made it more challenging and enriching as there was contact with a greater variety of tools than if it was only about software or hardware.

6.3 Conclusions and Future Work

The work done in this dissertation enabled a forklift to be used in the developed high-level teleoperation system. Moreover, this dissertation made possible the control of this forklift in the two available operation modes: remote and autonomous, by using the interface and the input devices available to the operator. However, some opportunities exist to improve this system and allow its use in the industry.

Regarding remote control, an additional camera in the opposite direction of the one already installed and a side-view camera near the pallet rack would allow the operator to have another view while driving without relying on the environment camera. Besides that, it would be interesting to allow the operator to regulate the speed so it could adjust the precision of the movement. In addition, there can be a more in-depth analysis of the visualization and actuation delays. Thus, another transmission protocol, or even 5G, would increase transmission speed, reducing this delay. Furthermore, a study of the actuation delay could be conducted to determine which component of this system is primarily responsible for it:

- if some of the ROS nodes
- if the transmission of information to the Arduino via the CAN bus
- if between the forklift's reading of the CAN bus and the corresponding execution

Regarding the autonomous mode, there can be an analysis of the error present in autonomous navigation by measuring the deviations in the distance the forklift should be from a given point and the orientation error concerning the desired orientation. Furthermore, there can be further research on the pallet sensor to integrate it into the system responsible for autonomously performing the pick and place task. This research can concentrate on:

- the detection differences between plastic and wooden pallets
- the impact that various camera parameters can have on pallet detection
- the range of distance and deviation values that the pallet can have concerning the sensor position for it to continue to detect the pallet

References

[1] X. Shen, Z.J. Chong, S. Pendleton, G.M.J. Fu, B. Qin, E. Frazzoli, and Jr. Ang, M.H. Teleoperation of on-road vehicles via immersive telepresence using off-the-shelf components. *Advances in Intelligent Systems and Computing*, 302:1419–1433, 2016. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84945947886&doi=10.1007%2f978-3-319-08338-4_102&partnerID=40&md5=874f1ba5eda699b8605ba10b78397711, doi:10.1007/978-3-319-08338-4_102.

[2] Amin Hosseini and Markus Lienkamp. Enhancing telepresence during the teleoperation of road vehicles using hmd-based mixed reality. In *2016 IEEE Intelligent vehicles symposium (IV)*, pages 1366–1373. IEEE, 2016. doi:10.1109/IVS.2016.7535568.

[3] F. Bazzano, F. Lamberti, A. Sanna, and M. Gaspardone. The impact of field of view on robotic telepresence navigation tasks. *Communications in Computer and Information Science*, 983:66–81, 2019. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85061159037&doi=10.1007%2f978-3-030-12209-6_4&partnerID=40&md5=58efc104021c1e120b0d377cf6b4e009, doi:10.1007/978-3-030-12209-6_4.

[4] J. Xiao, P. Wang, H. Lu, and H. Zhang. A three-dimensional mapping and virtual reality-based human–robot interaction for collaborative space exploration. *International Journal of Advanced Robotic Systems*, 17(3), 2020. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85086001420&doi=10.1177%2f1729881420925293&partnerID=40&md5=bbcc562bb685db7cf8ccb83bf391aa3c, doi:10.1177/1729881420925293.

[5] Michael E Walker, Hooman Hedayati, and Daniel Szafir. Robot teleoperation with augmented reality virtual surrogates. In *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 202–210. IEEE, 2019. doi:10.1109/HRI.2019.8673306.

[6] Marta C Mora, Veronica Suesta, Leopoldo Armesto, and Josep Tornero. Factory management and transport automation. In *EFTA 2003. 2003 IEEE Conference on Emerging Technologies and Factory Automation. Proceedings (Cat. No. 03TH8696)*, volume 2, pages 508–515. IEEE, 2003. doi:10.1109/ETFA.2003.1248741.

[7] Ho Seok Ahn, Samuel McArdle, Greg Sumner, and Bruce A MacDonald. Development of user interfaces for an internet-based forklift teleoperation system with telepresence. In *Proceedings of Australasian Conference on Robotics Automation*, 2014.

[8] M.D. Moniruzzaman, A. Rassau, D. Chai, and S.M.S. Islam. Teleoperation methods and enhancement techniques for mobile robots: A comprehensive survey.

Robotics and Autonomous Systems, 150, 2022. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85122643154&doi=10.1016%2fj.robot.2021.103973&partnerID=40&md5=5af74ed18ddd18d71516d1b209dd0c42, doi:10.1016/j.robot.2021.103973.

[9] Products - draw wire encoders, sfe-5000 sfe-10000. Accessed June. 6, 2022 [Online]. URL: https://www.lika.it/eng/products/draw-wire-encoders/-1/sfe-5000-sfe-10000.

[10] Roland Yves Siegwart and Illah Reza Nourbakhsh. *Introduction to autonomous mobile robots*. MIT, 2004.

[11] Youngjoo Kim and Hyochoong Bang. Introduction to kalman filter and its applications. In Felix Govaers, editor, *Introduction and Implementations of the Kalman Filter*, chapter 2. IntechOpen, Rijeka, 2019. URL: https://doi.org/10.5772/intechopen.80600, doi:10.5772/intechopen.80600.

[12] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT Press, 2005.

[13] H. Sobreira, C.M. Costa, I. Sousa, L. Rocha, J. Lima, P.C.M.A. Farias, P. Costa, and A.P. Moreira. Map-matching algorithms for robot self-localization: A comparison between perfect match, iterative closest point and normal distributions transform. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 93(3-4):533–546, 2019. doi:10.1007/s10846-017-0765-5.

[14] M. Lauer, S. Lange, and M. Riedmiller. Calculating the perfect match: An efficient and accurate approach for robot self-localization. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4020 LNAI:142–153, 2006. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-37249048510&doi=10.1007%2f11780519_13&partnerID=40&md5=cb9ea4c5c97df3bf4481356cb8f6ec8a, doi:10.1007/11780519_13.

[15] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002.

[16] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *IJCAI*, volume 3, pages 1151–1156, 2003.

[17] Agv navigation: Methods, comparison, pros and cons - illustrated guide. URL: https://www.agvnetwork.com/types-of-navigation-systems-automated-guided-vehicles.

[18] Shuzhi Sam Ge and Frank L. Lewis. *Autonomous Mobile Robots: Sensing, control, decision-making, and applications*. CRC / Taylor & Francis, 2006.

[19] Frank Liang. Optimization techniques-simulated annealing. Accessed April. 14, 2022 [Online], Apr 2020. URL: https://towardsdatascience.com/optimization-techniques-simulated-annealing-d6a4785a1de7.

[20] Frank Liang. Optimization techniques-tabu search. Accessed April. 15, 2022 [Online], Jul 2020. URL: https://towardsdatascience.com/optimization-techniques-tabu-search-36f197ef8e25.

[21] Howie M. Choset, Kevin Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, algorithms, and implementation*. MIT Press, 2005.

[22] J. Santos, P.M. Rebelo, L.F. Rocha, P. Costa, and G. Veiga. A* based routing and scheduling modules for multiple agvs in an industrial scenario. *Robotics*, 10(2), 2021. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85107229389&doi=10.3390%2frobotics10020072&partnerID=40&md5=008fc74715668faf6bc51744b7068264, doi:10.3390/robotics10020072.

[23] Karthik Karur, Nitin Sharma, Chinmay Dharmatti, and Joshua E. Siegel. A survey of path planning algorithms for mobile robots. *Vehicles*, 3(3):448–468, 2021. URL: http://dx.doi.org/10.3390/vehicles3030027, doi:10.3390/vehicles3030027.

[24] Han-ye Zhang, Wei-ming Lin, and Ai-xia Chen. Path planning for the mobile robot: A review. *Symmetry*, 10(10):450, 2018. URL: http://dx.doi.org/10.3390/sym10100450, doi:10.3390/sym10100450.

[25] Michael Hoy, Alexey S. Matveev, and Andrey V. Savkin. Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey. *Robotica*, 33(3):463–497, 2015. doi:10.1017/S0263574714000289.

[26] J.R. Sánchez-Ibáñez, C.J. Pérez-Del-pulgar, and A. García-Cerezo. Path planning for autonomous mobile robots: A review. *Sensors*, 21(23), 2021. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85120914893&doi=10.3390%2fs21237898&partnerID=40&md5=2ea01c38381baecc32c4ccb8779f3ace, doi:10.3390/s21237898.

[27] R.S. Allison, L.R. Harris, M. Jenkin, U. Jasiobedzka, and J.E. Zacher. Tolerance of temporal delay in virtual environments. In *Proceedings IEEE Virtual Reality 2001*, pages 247–254, 2001. doi:10.1109/VR.2001.913793.

[28] I Scott MacKenzie and Colin Ware. Lag as a determinant of human performance in interactive systems. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, pages 488–493, 1993. doi:10.1145/169059.169431.

[29] Jason P Luck, Patricia L McDermott, Laurel Allender, and Deborah C Russell. An investigation of real world control of robotic assets under communication latency. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 202–209, 2006. doi:10.1145/1121241.1121277.

[30] S. Opiyo, J. Zhou, E. Mwangi, W. Kai, and I. Sunusi. A review on teleoperation of mobile ground robots: Architecture and situation awareness. *International Journal of Control, Automation and Systems*, 19(3):1384–1407, 2021. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85093838441&doi=10.1007%2fs12555-019-0999-z&partnerID=40&md5=e9d540ed6a49b2c80f348a7c5ee73ce9, doi:10.1007/s12555-019-0999-z.

[31] Jörg Stückler, Max Schwarz, Mark Schadler, Angeliki Topalidou-Kyniazopoulou, and Sven Behnke. NimbRo Explorer: Semiautonomous Exploration and Mobile Manipulation in Rough Terrain. *Journal of Field Robotics*, 33(4):411–430, 2016. doi:10.1002/rob.21592.

[32] Maik Riestock, Frank Engelhardt, Sebastian Zug, and Nico Hochgeschwender. User study on remotely controlled uavs with focus on interfaces and data link quality. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3394–3400. IEEE, 2017.

[33] A. Kaknjo, M. Rao, E. Omerdic, L. Robinson, D. Toal, and T. Newe. Real-time video latency measurement between a robot and its remote control station: Causes and mitigation. *Wireless Communications and Mobile Computing*, 2018, 2018. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85058934145&doi=10.1155%2f2018%2f8638019&partnerID=40&md5=38e9defa5c199f6bba13e45f51f7fe35, doi:10.1155/2018/8638019.

[34] Safin Ramil, Roman Lavrenov, Tatyana Tsoy, Mikhail Svinin, and Evgeni Magid. Real-time video server implementation for a mobile robot. In *2018 11th International Conference on Developments in eSystems Engineering (DeSE)*, pages 180–185. IEEE, 2018.

[35] P. Udhayakumar and S. Kumanan. Task scheduling of agv in fms using non-traditional optimization techniques. *International Journal of Simulation Modelling*, 9(1):28–39, 2010. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-77949830406&doi=10.2507%2fIJSIMM09%281%293.139&partnerID=40&md5=11db6eac030582a7a1c10775ac2a9d1c, doi:10.2507/IJSIMM09(1)3.139.

[36] Marco Dorigo. Ant colony optimization. Accessed April. 11, 2022 [Online]. URL: http://www.scholarpedia.org/article/Ant_colony_optimization.

[37] Driving simulation and silab. Accessed April. 6, 2022 [Online]. URL: https://wivw.de/en/silab.

[38] F.J.S. Rodríguez, J.F. Rodríguez-Aragón, B.C. Diego, and V.M. Rodilla. Multi-agent cooperation for advanced teleoperation of an industrial forklift in real-time environment. *Advances in Intelligent and Soft Computing*, 88:57–62, 2011. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-80053022137&doi=10.1007%2f978-3-642-19875-5_7&partnerID=40&md5=e0aebc54c44010763a8e1f129d5ac1a1, doi:10.1007/978-3-642-19875-5_7.

[39] Babgei Atar. absoluteencoderssi, 2021. URL: https://github.com/atarbabgei/absoluteEncoderSSI.

[40] Remote operation - external pov and interface, Jul 2022. URL: https://youtu.be/Ah8fOBAuKX0.

[41] Remote operation - pallet sensor, Jul 2022. URL: https://youtu.be/oZVhWlFLjTI.

[42] Autonomous operation - external pov, Jul 2022. URL: https://youtu.be/JrTswUYkstc.

[43] Autonomous operation - interface, Jul 2022. URL: https://youtu.be/hp7s05EzX04.

[44] Diogo Correia, Manuel F. Silva, and Antonio Paulo Moreira. A survey of high-level teleoperation, monitoring and task assignment to autonomous mobile robots. In *2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 218–225, 2022. doi:10.1109/ICARSC55462.2022.9784769.