Faculty of Mechanical Engineering
Schmalkalden University of Applied Sciences

Learning Mobile Robots

Concept Design and Implementation of a Tactical Planning Model for Mobile Logistics Solution in Unstructured Urban Environment

By

Hajar EL FAKIR
Matrikel-Nr.: 313053

Supervised by

Prof. Dr. Frank Schrödel
Prof. Dr. Lutz Huxholl

Mechanical Engineering Faculty
Schmalkalden University of Applied Sciences

March, 2023

This is to certify that:

(i) The thesis comprises only my original work toward the Master Degree of Science (M.Sc.) at Schmalkalden University of Applied Sciences,

(ii) Due acknowledgment has been made in the text to all other material used

---

Hajar EL FAKIR
28^{th}March, 2023

Acknowledgments

I hereby thank everyone who have been a part of this thesis, directly or indirectly. I would like to thank in first place Prof. Frank Schrödel for giving me this great opportunity to be part of this interesting project which made me challenge myself and work on new topics. I'm so indebted to him for opening new doors for me in the mechatronics automotive and robotics industry and giving me a good insight on what this industry needs as well as what is an Automotive Mechatronics engineer. He has inspired me and always motivated me not only technically but also in all aspects of my professional career. He has been the perfect guidance and advisor in my professional life, and I appreciate that truthfully and wholeheartedly.

I also want to convey my gratefulness for all my colleagues in the Robotics Research Lab generally the Gera Project team and more specifically Joe Francis who inspired my model and Tarun Parmar who was a big help in testing my model; and with whom I have worked for a couple of years and exchanged experiences as well as different results in multiple meetings and discussions with Prof. Frank supervision, which gave me the chance to share and gain knowledge from them and this helped me strengthen not only my technical but also my soft skills.

A special thanks goes to Professor Carsten Behn for his guidance and Professor Lutz Huxholl for being my second supervisor. Both of them were among the best professors I had in my master studies.

Finally, my acknowledgement goes to my parents for all the support and encouragement that always kept me motivated but not only their mental support, their financial support as well which I will always be grateful for. I also thank my family, my sister Mariam and good friends for their presence during my hardest times.

Contents

List of Figures

List of Tables

Acronyms

AD	Automated/Autonomous Driving
ADS	Automated Driving Systems
AI	Artificial Intelligence
AV	Autonomous Vehicle
CA	Collision Avoidance
DDT	Dynamic Driving Task
DFSM	Deterministic Finite State Machine
FD	Free Drive
FS	Full Stop
FSM	Finite State Machine
LMD	Last Mile Delivery
MDP	Markov Decision Process
ML	Machine Learning
MPC	Model Predictive Control
NDFSM	Non-Deterministic Finite State Machine
OA	Obstacle Avoidance
ODD	Operational Design Domain
PP	Pure Pursuit
RL	Reinforcement Learning
ROS	Robot Operating System
SD	Slow Down
SU	Speed UP
VFH	Vector Field Histogram
V2X	Vehicle-to-Everything

Chapter 1

Introduction

Nowadays, *mobile robotics* is one of the fastest expanding fields of scientific research. Due to their abilities, mobile robots can substitute humans in many fields. Applications include surveillance, planetary exploration, patrolling, emergency rescue operations, reconnaissance, petrochemical applications, industrial automation, construction, entertainment, museum guides, personal services, intervention in extreme environments, transportation, medical care, and so on, as well as many other industrial and nonindustrial applications. Most of these are already available on the market. Mobile robots can move autonomously (in an industrial plant, laboratory, planetary surface, etc.), that is, without assistance from external human operators. A robot is autonomous when the robot itself has the ability to determine the actions to be taken to perform a task, using a perception system that helps it. It also needs a cognition unit or a control system to coordinate all the subsystems that comprise the robot. [20]

A *delivery robot* is an autonomous robot that provides "last mile" delivery services. An operator may monitor and take control of the robot remotely in certain situations that the robot cannot resolve by itself such as when it is stuck in an obstacle. Delivery robots can be used in different settings such as food delivery, package delivery, hospital delivery, and room service. [21] There are mainly three forms of products - autonomous delivery vans, sidewalk robots and autonomous delivery drones [22].

The robot delivery space has exploded in the past few years, in part due to the global pandemic and the rise of mobile ordering. The global market for delivery robots was valued at $300 million in 2021, according to Quince Market Insights. It's estimated to grow at a compound annual growth rate (CAGR) of 30.3 % from now until 2030 [23].

According to the researchers, it's estimated that last mile delivery accounts for more

than 20 % of the pollution in cities, something that could potentially be resolved with more efficient autonomous electric robots.They also can operate at times where there's less traffic congestion. The researchers also noted that these devices would lower transportation costs, with last mile costs currently accounting for 40 % of the total cost of delivery. Factors like "urban congestion, lack of parking spaces for loading and unloading commercial vehicles, and local regulations make the management of urban distribution of goods a very high cost for logistics companies," the press release reads. These autonomous delivery vehicles would represent "a significant reallocation of the carrier's costs and would make the service more economical and efficient than with conventional vehicles," it continues. [24]

Motivation

Transporting goods via freight rail networks and container ships is often the most efficient and cost-effective manner of shipping. However, when goods arrive at a high-capacity freight station or port, they must then be transported to their final destination. This last leg of the supply chain is often less efficient, comprising up to 53 % of the total cost to move goods This has become known as the "last mile problem.". The last mile problem can also include the challenge of making deliveries in *urban areas*. Deliveries to retail stores, restaurants, and other merchants in a central business district often contribute to congestion and safety problems. A number of companies are actively using small delivery robots to do the last-mile delivery of small packages such as food and groceries just using the pedestrian areas of the road and travelling at speed comparable with a fast walking pace. [25]

In this project we are using a side-walk autonomous delivery robot (Pioneer 3-Dx) which will deliver goods from REWE supermarket to Begegnungszentrum Eichenhof meeting center in Gera, Germany with a distance of 600m as part of 'Smart city' project for the city of Gera. For that a Hardware setup and a Software structure are adopted and will be discussed in detail in this report.

Outline of the Thesis

In this thesis, the main task is to build a model for the tactical planning module of our use case using MATLAB/Simulink as Software tool for building the model as well as ROS (Robot Operating System) for our model implementation on the Pioneer 3-Dx (text

vehicle). For that, it is important to go through the main points which are discussed in detail in the following chapters:

- *Chapter 2:* This chapter describes the Fundamental Architecture of the Automated Driving System (ADS). It also provides an overview on the Decision-Making Layer citing both approaches: the rule-based approach (If-Else and Finite State Machine) and Artificial Intelligence (AI) approach, focusing on the first approach and introducing different methods used in this thesis for solving automated driving problems. Finally, a comparison of each method and a small introduction of the Software Framework exploited.

- *Chapter 3:* This chapter describes the use-case; defining the scenarios and maneuvers. It also gives an overview of the hardware and software architecture of the project.

- *Chapter 4:* This chapter includes the concept's method chosen as well as the implemented model in this thesis. It summarizes the approach used considering If-Else and Finite State Machine (State Flow) methods for tactical planning as well as the architecture implemented. A small glimpse of Machine Learning solution method is mentioned for a wider vision.

- *Chapter 5:* This chapter includes the validation of the concept model solution where all the tests and results are explained in detail.

- *Chapter 6:* This chapter provides a conclusion and formal discussion about the results achieved in this research and proposes possible future improvements.

Chapter 2

Theoretical Basics

2.1 Autonomous Last-Mile Delivery Robots

In recent years, most efforts to develop autonomous vehicles have focused on last-mile delivery, driven by the rapid rise of e-commerce. The goal is to create machines that can deliver food and small packages directly to consumers in an affordable, quick, reliable and safe manner. [26]

FIGURE 2.1: Autonomous Delivery Robot: Delivers.ai. [1]

Last Mile robots are mobile robotic vehicles capable used to deliver small goods, post or groceries in an autonomous and fast way. These units, also called Last Mile Delivery (LMD) robots, have different sizes, shapes and modes of propulsion. As explained, last mile represents a huge cost in the "delivery chain". To battle these cost drivers, the tech industry, in pursuit of optimizing efficiencies and blending robots and supply chain management, has developed the Last Mile Delivery Robots. These robots work to aid in the delivery of packages and purchases from the local distribution center or store directly

to the consumer. If successful, supply chain management could then prioritize employee power on deliveries and activities that can't be done by robots. Efficiencies rise and costs decrease. However, in industrial or commercial environments everything (almost) can be controlled and made to be symmetrical and standardized. This isn't exactly how the world works outside the closed environment of an engineered facility. Public Roads have hills, curves, bumps, holes, pedestrians, bikes, cars, and most importantly changing weather conditions. [27]

To tackle these challenges, especially in urban environments, it crucial to decide on a set of hardware components and software tools that will help these robots to navigate more optimally in different areas.

2.1.1 Hardware Components: Sensors

In order to carry out truly autonomous action, robotic systems must be equipped with sensory components, information-processing capabilities, and some form of actuation. This part looks at the most common sensor types in robotics used for autonomous delivery and explains how they contribute to the autonomous vehicle systems.

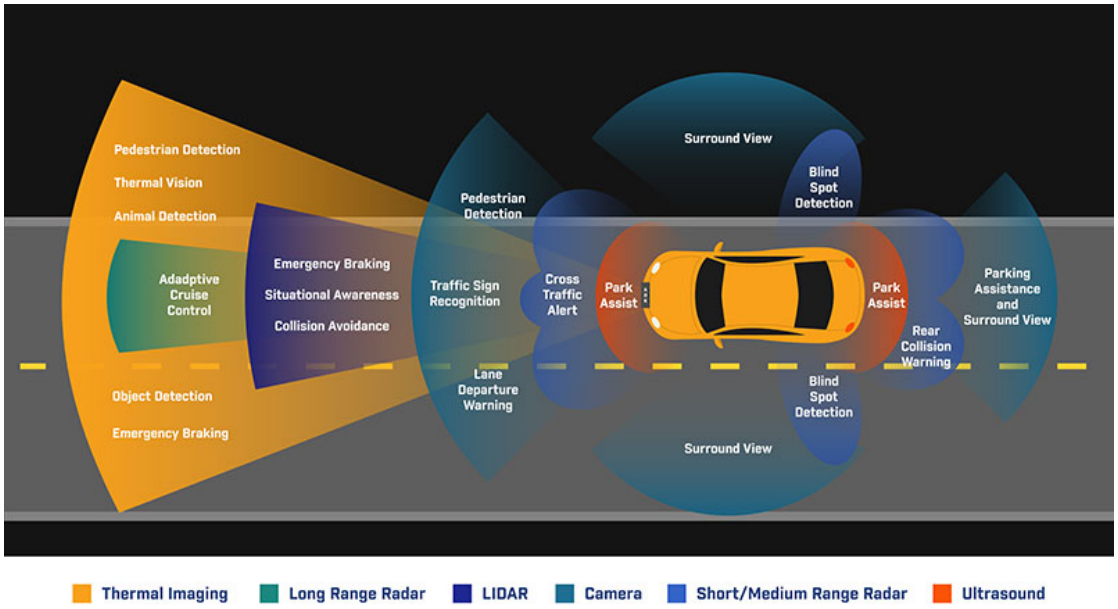


FIGURE 2.2: Autonomous Vehicles Sensors and their Ranges. [2]

2.1.1.1 LIDAR

Light detection and ranging sensors (better known as LIDAR) are at the forefront of autonomous vehicle technology. LIDAR robots use near-infrared light to detect the

distance between a transmitter and a reflective 'point,'. However, LIDAR systems create millions of light points per second in a 360-degree field of view, allowing each point to contribute to a 'point cloud'. The LIDAR sensors can detect pedestrians up to 250 meters away and cars at up to 500 meters away. These point clouds create a fully measurable, 3-dimensional map of the environment around the LIDAR system and the AV. Using AI neural networks and self-driving algorithms to process LIDAR's point cloud data, autonomous vehicles are able to create virtual, dimensioned worlds in which they can identify known objects such as cars, people and bicycles, and then navigate through them.

FIGURE 2.3: LIDAR for Autonomous Vehicles. [3]

2.1.1.2 Radar

Similar to LIDAR, radar uses a transmit and receive methodology to find objects. RADAR, which takes its name from an acronym for Radio Detection and Ranging, transmits radio waves outward in a specified direction. It receives the rebound of those waves to understand the distance from objects. Radar utilizes much larger wavelengths than LIDAR. In turn, it can detect objects further away but at a much lower resolution. It requires far less compute power than LIDAR, which makes it advantageous for specific functions within an autonomous vehicle. For example, radar is excellent at determining the speed of a vehicle relative to other objects, as well as detecting large objects such as houses, cars and other obstructions. Generally speaking, radar is used supplementally to complete functions that would be unnecessary to complete with LIDAR sensors. [28]

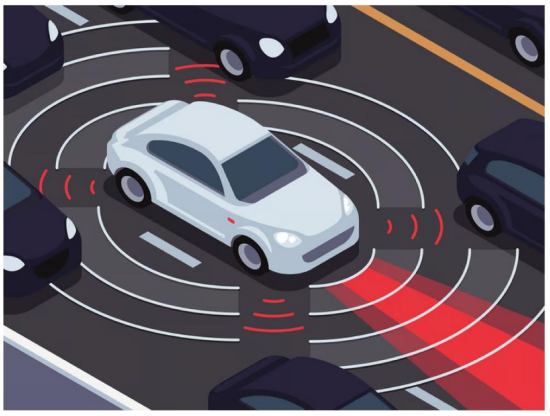


FIGURE 2.4: Radar for Autonomous Vehicles. [4]

2.1.1.3 Thermal Cameras

Thus far, autonomous vehicles have a hard time detecting humans in many different environments. Whether in a bustling city center or a quaint suburban neighborhood, humans are surprisingly good at blending into their environment in the eyes of a LIDAR or radar system. As such, thermal imagers and cameras are used to supplement LIDAR and radar models by providing an entirely different set of signature data. Given that humans themselves emit light, thermal cameras can be used to help detect if a human or other animal is present but is being misrepresented by call-and-response-based technologies. For example, if a human is sitting on a bench next to a trashcan and that trashcan is in the line of sight of an autonomous delivery robot's LIDAR, the LIDAR system may only identify the trashcan as it cannot 'see' the entirety of the human. Thermal cameras, however, would be able to see the human's body parts and/or heat signatures that are within view of the camera, thereby helping the autonomous vehicle understand that a human is by the trashcan. Thermal cameras are also able to detect potential systems threats, such as fires, open manholes, or even freshly poured concrete.[28]

FIGURE 2.5: Thermal Camera for Autonomous Vehicles. [5]

2.1.1.4 GPS

Global positioning sensor (GPS) receivers are devices used within Earth's planetary atmosphere to determine its precise geographical location. These receivers are used in conjunction with over 20 orbiting Earth satellites to triangulate the exact three dimensional position based off of the receiver's position about multiple satellite signals. It is important that the receiver be in contact with at least four of these satellites in order for an accurate location to be determined. For this reason, the orbital strategy of the GPS systems always has at least four satellites visible from any point on earth at any time. GPS devices can determine the location of the receiver to within a few feet of accuracy about the earth's surface. To do this there has to be a time sync between the satellites and the receiver. The manner in which they sync is rather complex, but with an accurate clock on both transmitter and receiver, the delays between the transmissions allow the receiver to make a very accurate distance measurement between each satellite. When using the distance information from four different satellites, it is possible to determine three-dimensional locational accuracy.[29]

2.1.1.5 Camera

Of all of the sensor technologies listed, cameras are the oldest and, subsequently, the most advanced. Some autonomous vehicle makers such as Tesla prefer to use cameras as their main source of data (as opposed to LIDAR/radar systems) given their natural human-level understandability. Since humans do not see in LIDAR point clouds, identifying and labeling LIDAR data can make neural network training even harder. Cameras, however, process natural reflected light. Humans are spectacularly well equipped to identify, label and understand camera sensor data, making AI training and data understanding very easy.[28]

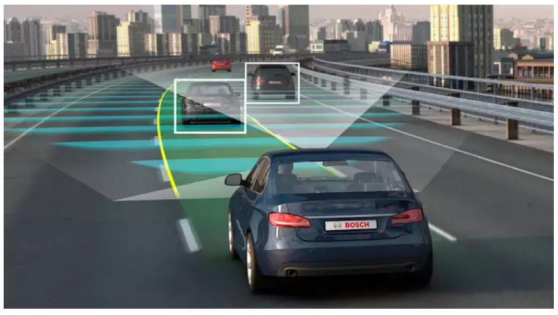


FIGURE 2.6: Camera for Autonomous Vehicles. [6]

2.1.2 Functional Architecture for Automated Driving

Choosing an appropriate functional architecture for automated driving (AD) is fundamental for making the right decisions in line with requirements regarding the intended driving behavior. Mastering the complexity of AD, especially in urban traffic scenarios, while simultaneously adhering to functional safety requirements is a challenging task that has to be considered in the architecture and the methods of decision-making from scratch. Two classical architecture approaches can be pointed out in automated systems. The first one is referred to as function-based, top-down, or knowledge-driven architecture. It uses an explicit world model for decision-making and planning as well as couples deliberative agents with explicit knowledge. An over-all mission for each agent is decomposed into sub-goals within a hierarchical control structure. That is, higher layers in the hierarchy create sub-goals for the lower layers. This includes the way of *sense-model-plan-act* paradigm. In consequence, the high-level side has long planning cycles and no fast reactions to dynamic environment changes. Deliberative agents are well suited for structured and highly predictable environments. This approach provides also the benefits of mission-oriented planning and high-level intelligence. However, top-down architectures require a general world model that is hard to represent and also computationally expensive. Therefore, the world representation is often reduced to topological and semantic maps. Furthermore, purely deliberative agents are not sufficient to realize intelligent behavior be-cause of over-simplification. The problem of static world representation is the absence of intuitive interpretation and solutions in dynamic environments.

The second architecture approach, in turn, focuses on reactive agents. Reactive agents implement decision-making and planning functionality by bottom-up or data-driven models that react to their sensory input directly without high-level planning and overall missions . The aim is a timely robotic response in dynamic and unstructured worlds. The main ad-vantage is the robustness of the directly coupled approach of perception and action (behavior). For a purely reactive approach to be applicable, however, actions (or driving functions in this case) have to be explicitly defined and made available in advance, leading to a deficiency of flexibility in the architecture.

Different hybrid architectures were developed to combine the benefits of both deliberative and reacting agents and to eliminate or at least minimize the drawbacks of each approach.

In AD hybrid models are in fact the most common approach and gain a lot of attention

in research. Reactive and deliberative functions are typically combined within a three-layered architecture: the deliberative layer, the sequencing layer, and the reactive layer. The deliberative layer is responsible for long-term strategic mission planning including mission adaption. The sequencing layer, in turn, coordinates the interaction between the deliberative and the reactive layer to execute appropriate tactics using context-dependent rules. The term tactic, in this context, refers to a predefined, ordered, and structured set of actions. The reactive layer acts with metric information and numerical control. The difficulty lies in the interface of these different layers because they work asynchronously, with different time scales and data representations.

Choosing the most appropriate functional architecture for AD usually depends on the application and the corresponding requirements.These architectures were the breakthrough for developing functional hybrid architectures for AD and are particularly characterized by their hierarchical planning. Furthermore, the well-known DARPA challenge gave rise to various hybrid architectures, where each architecture was specialized for solving the course defined by the challenge. Urban scenarios, however, were only partially considered. Later architectures focused more on urban environments. Further research in functional architectures for AD finally led to a 3-layered decision-making approach consisting of a strategical, a tactical, and an operational layer.[7]

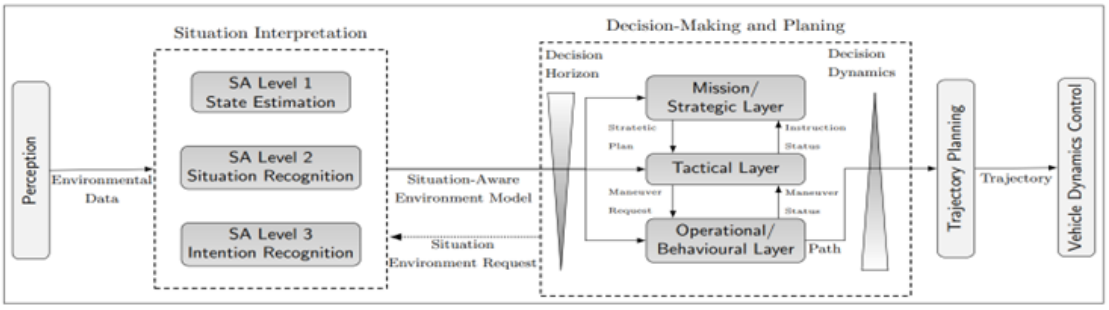


FIGURE 2.7: Functional Architecture for Automated Driving Systems.[7]

Figure 2.7 gives a rough overview of our functional architecture:

- The *Perception* cluster is responsible for perceiving the environment.
- The *Situation Interpretation* cluster translates the environmental data into an environmental model (or situation model).
- The *Decision-Making and Planning* cluster requires the respective information (Situation Interpretation Data). Information maximization and reduction (e.g., by selection or abstraction), as well as information representation, are key aspects. Both clusters are indeed tightly coupled and allow for situation-aware decision-making and planning.

- The *Trajectory Planning* cluster translates geometric paths with velocity annotations into trajectories or even alternative representations.
- The *Vehicle Dynamics Control* cluster is responsible for generating control variables such as steering angle and acceleration.

2.1.3 DECISION-MAKING

The decision-making process for intelligent high-level planning vehicles requires a hierarchical and situation-aware representation of the world. The focus of this section is presenting methods that handle, in a safe manner, decisions based on uncertain and incomplete information. Therefore, this section presents the underlying decision theory and gives a short overview of different applications and tasks in automated driving.

2.1.3.1 Intelligent Decision-Making for Maneuver Selection

Decision-making is essential for planning, selecting, and finally performing maneuvers. In Figure 2.7, the decision-making and planning cluster is hierarchically structured into multiple layers:

- *Mission Layer:* The mission layer provides all long-time goals and tasks that come with the intention of the human driver. It encompasses the global graph search algorithm to find the route in the road network under consideration of constraints like the fastest ways or resource-saving.
- *Strategic Layer:* The strategic layer includes the graph search algorithms to find the optimal route of roads in a predefined horizon and even driving lane-specific segments to reach the driving destination. The strategical layer and the mission layer mainly incorporate global sensor information for decision-making.
- *Tactical Layer:* The tactical layer can be considered as the interface between the high-level planning side and the low-level behavioral planning side, where the data structure between these layers may vary. This layer uses global and intern sensor information as the foundation of decision-making. The tactical layer is responsible for modifying the planned lane-specific navigation in such a way that it fits with the driving maneuvers of other traffic participants. Also, this layer generates a sequence of predefined operational functions for achieving more complex maneuvers such as overtaking maneuvers or it may generate acceleration and deceleration recommendations for the operational layer. Depending on the operational layer and the implemented behaviors, the tasks of the tactical layer can be very different

and includes symbolic (maneuver selection) and also metric (e.g., advise of acceleration) decisions. In any case, these decisions always have to take prevailing and emerging situations into account.

- *Operational Layer:* The operational layer subsumes everything necessary to translate maneuvers selected on the tactical decision-making layer into valid paths and trajectories. The corresponding driving functionality may also receive parameter sets from the tactical layer to realize the same maneuver in different scenarios and to adapt optimization functions.

[7]

2.1.3.2 Top-Down Decomposition

For better understanding the interaction between the different layers; Based on the environmental data received from the sensory setup, an environmental model is being generated, by mean of situation interpretation, in order to be used later on by the decision making layer on request. After that, a valid trajectory is being generated and the vehicle dynamics parameters of the vehicle will be controlled accordingly. By adopting this kind of functional architectures, the mobile robot can navigate autonomously from location A to location B by selecting the most efficient path and by avoiding all statics and dynamics obstacles that may be encountered during its route. The LMD robots as stated can operate with more flexibility and adaptability to change of behavior and environments. Moreover, LMD robots can rely on more advanced controller than a classical PID such as a model predictive control (MPC). It is important to mention that the autonomous mobile robots depend heavily on the operational design domain (ODD), in other terms where the robot will operate if in indoor, outdoor, unstructured, crowded environments, or on public roads... Also LMD robots depend on another factor, known as the dynamic driving tasks (DDT) such as accelerating, decelerating, yielding, driving at constant speed, following another vehicle, overtaking, avoiding an obstacle and rejoining the desired path... Both ODD and DDT can highly affect the development of the functional architecture [Figure 2.7] and the algorithm design, where the latter can be simplified a lot or become more complex depending on the situation as discussed.

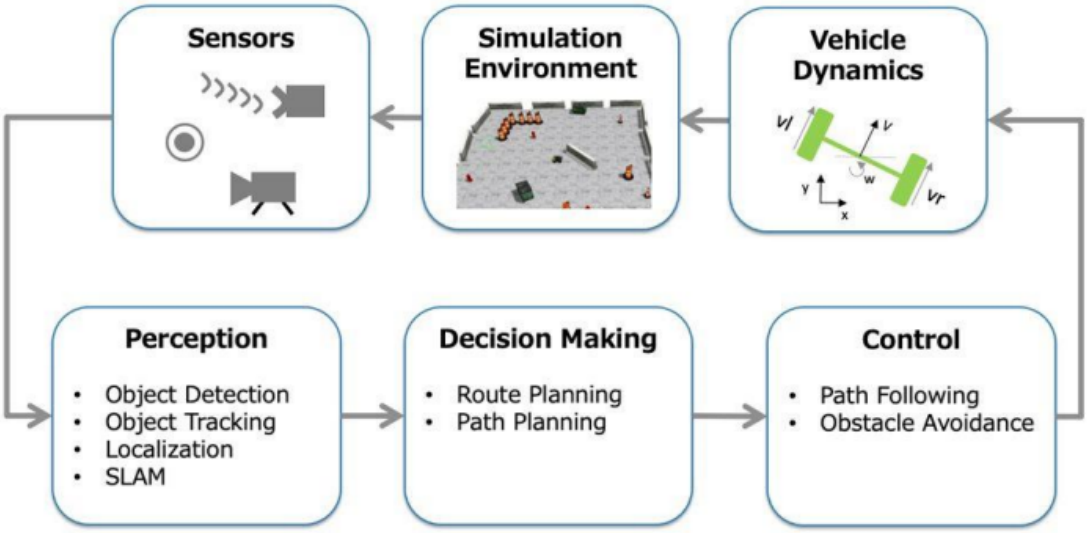


FIGURE 2.8: Advanced Functional Blocks Architecture Loop.[8]

The generic functional architecture presented in Figure 2.7 can be visualized more clearly as a simplified closed loop in Figure 2.8. The different functional blocks are connected in series from perception, decision making, control, vehicle dynamics to implementation either in virtual simulation environment or on a real vehicle.

2.1.3.3 Composition Strategies and Methods

By using building blocks to compose complex functionality on a higher hierarchy layer, we can flexibly apply different composition methods on different layers while taking decision dynamics and horizons into account. Composition, in this context, refers to coming up with a plan to achieve the goal on the next higher hierarchy layer. Since automated vehicles have to deal with a non-deterministic, highly dynamic environment, planning, executing a plan, and revising a plan might also be tightly interwoven especially on the lower hierarchy layers.

A very common and straightforward approach to compose high-level functionality is to encode all possible plans in state machines when the AD-System is designed. Roughly speaking, depending on a current state (or situation) and according to predefined transitions with transition conditions between the states, a state machine clearly defines in which situation and under which condition an action (e.g., a complex or basic maneuver) may be selected. Although this approach is indeed very intuitive, it is also very restricted: Knowledge, which was not encoded into the state machines during design

time, is not available when the AD-System is online and running. That is, the developers and domain experts have to consider all situations that may occur in the desired application and precisely define all relevant transitions and corresponding conditions in advance. For a very restricted ODD and a limited set of DDTs, this approach usually works pretty well, since a thorough investigation of possible situations and necessary maneuvers is still manageable, while state machines that encode the result of the investigation are still maintainable. With increasing AD functionality, however, this approach is not feasible anymore and should be replaced by AI approaches in terms of Automated Planning and ML techniques.[7]

2.2 The Analyses of Decision-Making Relevant Solutions for Autonomous Driving

The tactical decision-making approaches for autonomous vehicles roughly fall into three main directions: classical approaches, utility/reward-based approaches, and machine learning approaches as shown in Figure 2.9.

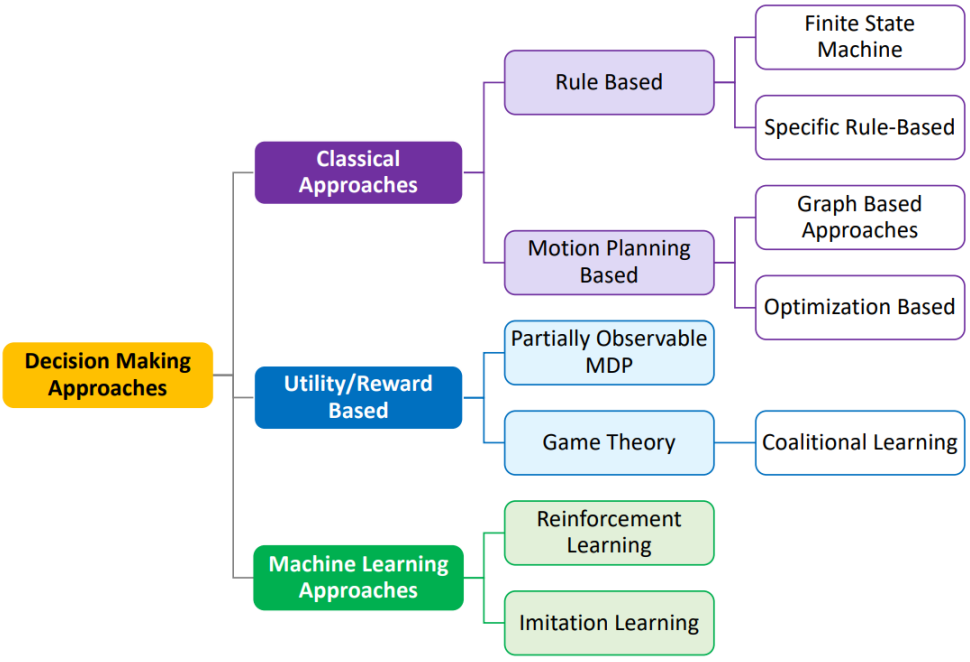


FIGURE 2.9: Categorization of Decision-Making Approaches for Autonomous Vehicles.[9]

Since the topic of this thesis focuses specifically on rule-based and machine learning approaches, they will be discussed in detail in the next subsection.

2.2.1 Rule-Based Approach

A rule-based approach uses rules as the knowledge representation. These rules are coded into the system in the form of if-then-else statements. The main idea of a rule-based system is to capture the knowledge of a human expert in a specialized domain and embody it within a computer (or robotic) system. Hence, knowledge is encoded as rules.[16] The common representative of this approach is the Finite State Machine (FSM) method.

2.2.1.1 Finite State Machine

A finite state machine (sometimes called a finite state automaton) is a computation model that can be implemented with hardware or software and can be used to simulate sequential logic and some computer programs[10]. A sample of FSM shown in Figure 2.10 is a mathematical model with discrete inputs and outputs where the actions are generated to react to the external events resulting in transitioning the states of the agents to another state.[8]

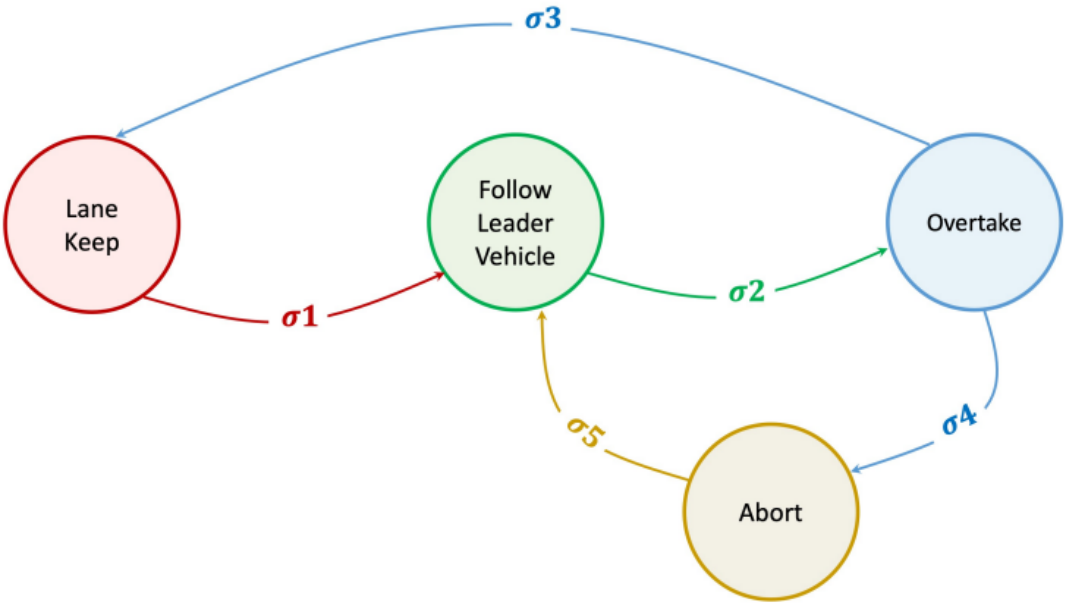


FIGURE 2.10: Simple Finite State Machine for Autonomous Vehicle Overtaking Maneuver.[8]

The circles are "states" that the machine can be in. The arrows are the transitions.

There are two types of basic state machines:

Deterministic finite state machine

A deterministic FSM is that for every state for any symbol there is at most one transition from that state labeled with that symbol. This means that there is never a choice in how the machine is to proceed when it sees a symbol: there will be at most one state to move to given that symbol. e.g. the "if" statement in that if 'x' then 'doThis' else 'doThat'; the computer must perform one of the two options.

A deterministic finite machine state (DFSM) is described by a five-element tuple:

$(Q,\sum,\delta,q_0,F)$.

Q = a finite set of states

$\sum$ = a finite, nonempty input alphabet (letters, characters, or digits)

δ = a series of transition functions

q_0 = the starting state or initial state

F = the set of accepting states or final states

There must be exactly one transition function for every input symbol in $\sum$ from each state. DFSMs can be represented by diagrams of this form:

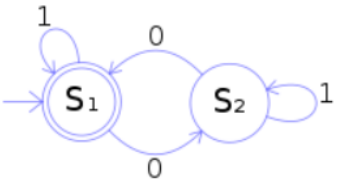


FIGURE 2.11: Diagram for Deterministic Finite State Machine.[10]

Nondeterministic finite state machine

Non-deterministic finite state machines are finite state machines where a given input from a particular state can lead to more than one different state. Unlike DFSMs, NDFSMs are not required to have transition functions for every symbol in $\sum$, and there can be multiple transition functions in the same state for the same symbol. Additionally, NDFSMs can use null transitions, which are indicated by ϵ. Null transitions allow the

machine to jump from one state to another without having to read a symbol.

NDFSMs can be represented by diagrams of this form:

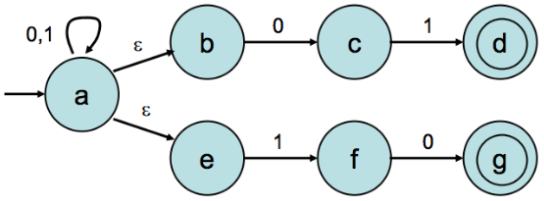


FIGURE 2.12: Diagram for Nondeterministic Finite State Machine.[10]

2.2.2 Artificial Intelligence Approach

Artificial Intelligence (AI), the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings. The term is frequently applied to the project of developing systems endowed with the intellectual processes characteristic of humans, such as the ability to reason, discover meaning, generalize, or learn from past experience. Since the development of the digital computer in the 1940s, it has been demonstrated that computers can be programmed to carry out very complex tasks—as, for example, discovering proofs for mathematical theorems or playing chess—with great proficiency. Still, despite continuing advances in computer processing speed and memory capacity, there are yet no programs that can match human flexibility over wider domains or in tasks requiring much everyday knowledge. On the other hand, some programs have attained the performance levels of human experts and professionals in performing certain specific tasks.[30]

Autonomous Agents

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors. A human agent has eyes, ears, and other organs for sensors, and hands, legs, mouth, and other body parts for effectors. A robotic agent substitutes cameras and infrared range finders for the sensors and various motors for the effectors.[31]

The concept of agent and its application in Artificial Intelligence (AI) needs to be understood. According to Wooldridge, “An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives”.

However, in reality, autonomous agents are complex systems to be deployed. Agents need to be designed in such a way that they can take decisions based on the environments. Actions taken by agents should maximize the chance of to fulfill a goal. Agents need to understand and remember it's past experiences that can be a deciding factor in future to take actions. Autonomous agents are the best fit in these scenarios. Relating to the previous experiences with present scenarios of environments, agents learn to take actions.

According to Wooldridge there are agents which are called goal-oriented agents. Goal information describes the situation which is desirable from any agent. Agents choose a way from different possibilities, selecting the best possible option to reach the goal. By different planning and calculating independent mathematical results of multiple possibilities, chooses a actions sequence which helps to reach the goal.

One of the very promising techniques in autonomous agents is model-based agents. It can handle partially observable environments. It stores the current state with some structure which helps to determine the future part of the environment. It stores some internal model based on the history and based on that, it able to determine some unobserved sides of the current state.[11]

2.2.2.1 Machine Learning

Machine learning is a subset of artificial intelligence. It focuses on improving how a machine performs some task. Here's the most important part: learning means that the machine goes beyond the training data. Equipped with machine learning algorithms, a computer can apply induction and form knowledge structures. In other words, where traditional programming fails, custom software development powered by machine learning and artificial intelligence can [32]

Machine Learning mainly consists of four types of processes:

- *Supervised Learning:* This is kind of task performed on labelled data to learn a function, which maps input data to output data based on example input-output pair.
- *Semi-supervised Learning:* Semi-supervised learning is a class of machine learning tasks and techniques that that applied on unlabeled data as well as labelled data. Typically, in this task the amount of labelled data is small amount in compare to unlabeled data.

- *Unsupervised Learning:* This task is performed on huge amount of unlabeled data to make some inference out of it.
- *Reinforcement Learning:* Reinforcement learning is an area of machine learning which determine of autonomous agents can take actions in an environment to reach goal by maximizing the defined rewards from the environment.

[11]

Machine learning (ML) has recently attracted attention in the field of autonomous driving due to advances in deep learning. The advantage of this approach is that it does not rely on hand-crafted rules and scales well with data, improving performance as more data is utilized for training. Consequently, this method has a great deal of potential to manage a wide range of driving scenarios. A plethora of ML approaches have been investigated and are contributed by the research community for different use cases enabling a higher level of autonomous driving.[8]

2.2.2.2 Reinforcement Learning

Reinforcement learning (RL) is an extensive learning on how to map situations to actions maximizing a numerical reward signal. In Reinforcement learning learner are not being told what action is to be taken in different scenarios, rather the learner learns based on the environment and rewards that it gets whenever a new action is to be taken. The trial-and-error search is the most important feature of reinforcement learning.

Breaking it down, the process of Reinforcement Learning involves these simple steps:

1. Observation of the environment.
2. Deciding how to act using some strategy.
3. Acting accordingly.
4. Receiving a reward or penalty.
5. Learning from the experiences and refining our strategy.
6. Iterate until an optimal strategy is found.

There are two main types of RL algorithms. They are *model-based* and *model-free*. A *model-free* algorithm is an algorithm that estimates the optimal policy without using or estimating the dynamics (transition and reward functions) of the environment. Whereas

a *model-based* algorithm is an algorithm that uses the transition function (and the reward function) in order to estimate the optimal policy. [33] The RL agent not necessarily always requires complete knowledge or control of the environment; but it should be able to interact with the environment and collect information. There are two types of setting for Reinforcement learning

- *Offline* settings where the experience is obtained prior to the learning and based on that the agent learns how to behave. This is called Batch Reinforcement Learning..
- *Online* settings where the data becomes available in sequential manner and based on the data, progressive behavioral update of the agent takes place.

In both cases, the core learning algorithms are essentially the same, but the main difference is that in an online setting, the agent can influence how it gathers experience which produces most effective learning. The online settings in Reinforcement Learning are an advantage as the agent is getting continuous feedback from the environment and thus it helps to learn in more efficient manner. For that reason, even when the environment is fully known, RL approaches may provide the most computationally efficient approach in practice as compared to some dynamic programming methods that would be inefficient due to this lack of specificity.

The general RL problem is formalized as a discrete time stochastic control process where an agent interacts with its environment in the following way: the agent starts, in a given state within its environment S_0‘ $\in S$, by gathering an initial observation. At each time step t, the agent has to take an action $A_t \in A$.

It then follows three consequences: the agent obtains a reward $R_{t+1} \in R$, the state transitions to $S_{t+1} \in S$.[11]

2.2.2.3 Reinforcement Learning and Markov decision Process

Markov Decision Process (MDP) is a mathematical framework to describe an environment in reinforcement learning. More specifically, the agent and the environment interact at each discrete time step, t = 0, 1, 2, 3... A_t each time step, the agent gets information about the environment state S_t. Based on the environment state at instant t, the agent chooses an action A_t. In the following instant, the agent also receives a numerical reward signal R_{t+1}. This thus gives rise to a sequence like S_0, A_0, R_1, S_1, A_1, R_2... The random variables R_t and S_t have well defined discrete probability distributions. [34]

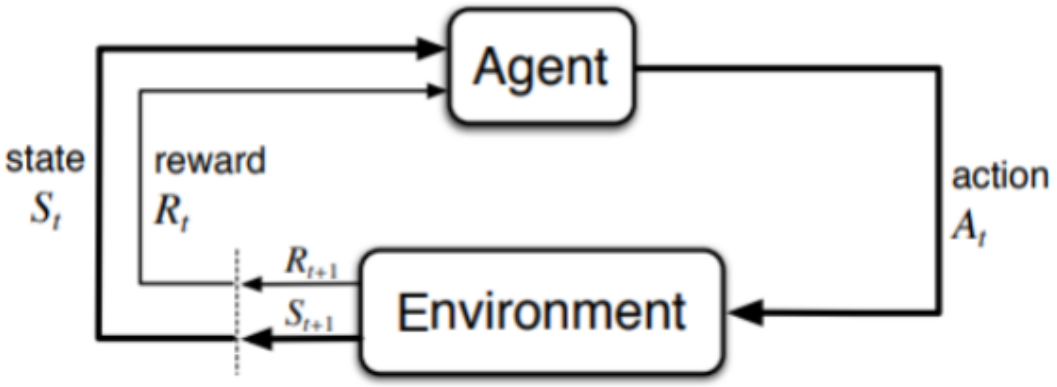


FIGURE 2.13: The agent environment interaction in a Markov decision process.[11]

2.3 Tools and Frameworks

The software used in this thesis is *Matlab/Simulink* which is a numeric computing environment developed by MathWorks, it allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages. An additional package, Simulink, adds graphical multi-domain simulation and model-based design for dynamic and embedded system.

Matlab includes various Modeling tools for different uses. For that, we considered for the tactical planning model representation using *Stateflow* which is a graphical programming environment based on finite state machines consisting of states, transitions and data and can model combinatorial and sequential decision logic, which can be simulated as a block within a Simulink model or executed as an object in MATLAB. Stateflow was used in this thesis to determine the conditions and states which were represented in flowchart form. Flow chart is a graphical construct that models logic patterns, it represents combinatorial logic in which one result does not depend on prior results.

A very commonly used robotic framework was used in this thesis to implement the Simulink model into the test vehicle. This framework is non other than the *Robotic operating system* (ROS).

Chapter 3

Concept

This chapter focuses on the concept of this thesis. The use case is elaborated in detail followed by an overview of the hardware and software architecture adapted in this project work.

3.1 Use case

3.1.1 General Idea

The main goal of this subsection is to answer on one trivial question : Why is defining a use case so important for the present thesis ? First, in the literal meaning ; a use case is a set of steps that are required to accomplish a specific task or goal. A use case can have multiple paths to reach the goal; each of them is considered a use case scenario. In simple words, a use case is a goal with various processes, and a case scenario represents a linear and straight path through one of the operations.[35]

To define a Use Case in this thesis, the following aspects should be specified:

- The start and goal position.
- The path (described as a set of predefined waypoints).
- The scenarios.
- The maneuvers.

More precisely, the following block diagram will give a rough overview about the different parts that will be the subject of this thesis: The input of this block (1) diagram are

FIGURE 3.1: Block diagram of the tactical planning layer.

the different scenarios present in our use case which can occur in different environments with different environmental conditions. These scenarios can be devided as follow:

- Road infrastructure: slopes and narrow sites.
- Traffic participants: pedestrians, cargos, pets and other vehicles (cars, bikes ...).
- Common areas: crosswalks and crossroads, house entrances, parking lots and garages (maybe parks).

The tactical planning block (2) is a black box in this section and will be discussed in detail in Chapter 4. The tactical layer as explained before in Chapter 2 is the interface between the high-level planning side and the low-level behavioral planning side. It is the main block of this thesis which will be elaborated later on.

The output (3) represents the different driving maneuvers which has to be performed by the test vehicle for different reasons, in different situations, and under different environmental conditions. It is selected very carefully by the tactical planning block. These maneuvers can be categorized as follow:

- *Longitudinal maneuvers:* responsible for the regulating the vehicle's cruise velocity, e.g., full stop (FS), free drive (FD), slow down (SD), speed up (SU), obstacle avoidance (OA_v).
- *Lateral maneuvers:* steers the vehicle's wheels for path tracking, e.g., obstacle avoidance (OA_ω).

It is important to add that there are two types of maneuvers: basic and complex maneuvers. The tactical layer is responsible for decomposing complex maneuvers into sequences of basic maneuvers. For example: Collision avoidance is a complex maneuver that is

decomposed into two basic maneuvers: full stop and free drive. This maneuver is very essential and will be mentioned frequently in the course of this thesis.

3.1.2 Costumer requirement

A *costumer requirement* will give us more information regarding the motive and needs of the customer. So that the use case will provide us with the details of how to accomplish the target and all the scenarios that the system can encounter while performing the task.

In our case the robot, which is a Pioneer 3-DX, has to follow a path from the Market center REWE, Zeulsdorfer Str. 85 til the residential area Begegnungszentrum Eichenhof, Eichenstraße 11B in Gera. The distance between the start and end point is about 600 m as specified in google maps (Figure 3.2).

This operation is limited to a small, controlled environment and only on specific roads,

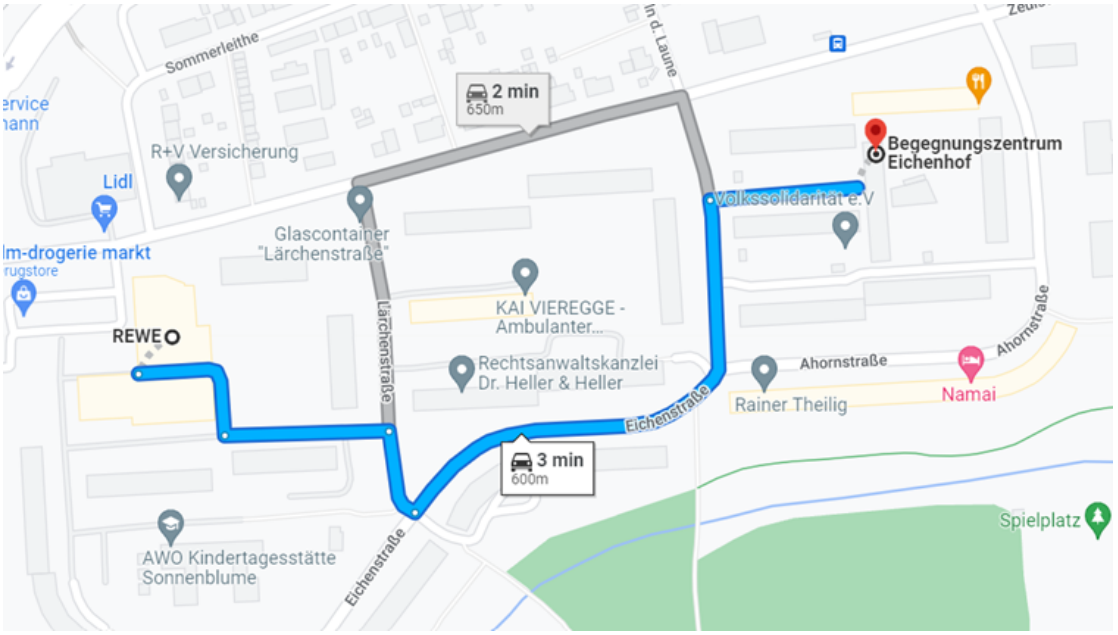


FIGURE 3.2: Route options (source: google maps).

in daytime and with mild weather conditions which makes the task less complex. It is considered a low-cost solution for delivery purposes.

FIGURE 3.3: real map of Gera route (source: google maps).

Components		Cost
Microcontrollers	Nvidia Jetson Xavier	645,05 €
Sensors	Real Sense Camera model Intel Real Sense D435i	433,29 €
	Multi-Sensor RTK	15587,81 €
I/O devices	HMI display model Wave share 11.6-inch LCD	195 €
	Wireless keyboard model Rapoo E2700	29,99 €
Electronical components	DC-DC Converter model QSKJ DC-DC (2×)	2×59,01 €
	Wires, cables, adapters, etc.	100 €
Mechanical components	Caster wheels and Aluminum setup	500 €
Robot	Pioneer 3-DX	2,657.20 €
Total		21525,37 €

TABLE 3.1: Hardware components costs.

3.1.3 Scenarios set

Due to local conditions, two preferred route options emerged after inspecting the surrounding streets and footpaths as shown in figure 3.4. The starting and ending points are the entrance to the Eichenhof meeting center and the main entrance to the REWE supermarket. The selected routes differ primarily in the nature of their paths and the number of hazardous areas that need to be considered separately for the use of the robot. But the optimal route presented as orange line in figure 4 is the one which will be considered in this thesis and discussed in this subsection. The marked positions stand for existing demanding circumstances along the way, such as road crossings. The set

of scenarios will be discussed in detail in the course of this subsection. Route 2 runs a

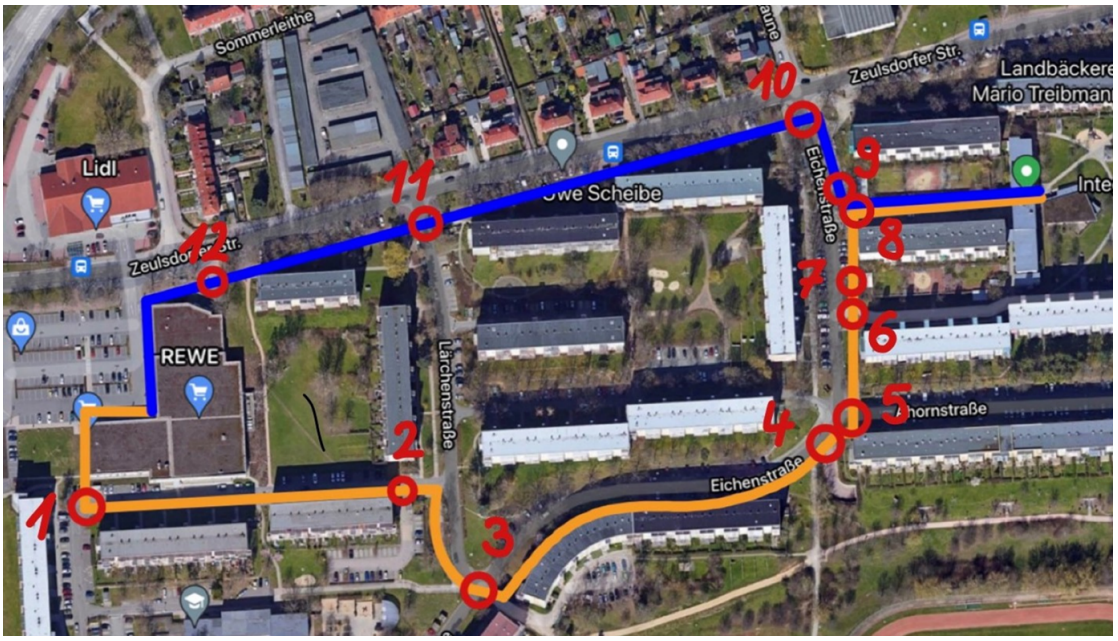


FIGURE 3.4: Preferred routes for the Pioneer 3-DX in Gera-Lusan (blue – route 1; orange – route 2; red – danger spots).[12]

footpath within a 30km/h zone along less busy streets in the residential area of Gera-Lusan. Due to the use of secondary roads, traffic is calmer and there are fewer cars and pedestrians. The scenarios represented as danger spots in Figure 3.4 (red marks) are depicted the Figure 3.5. The green line represents the supermarket entrance; it is a foot-

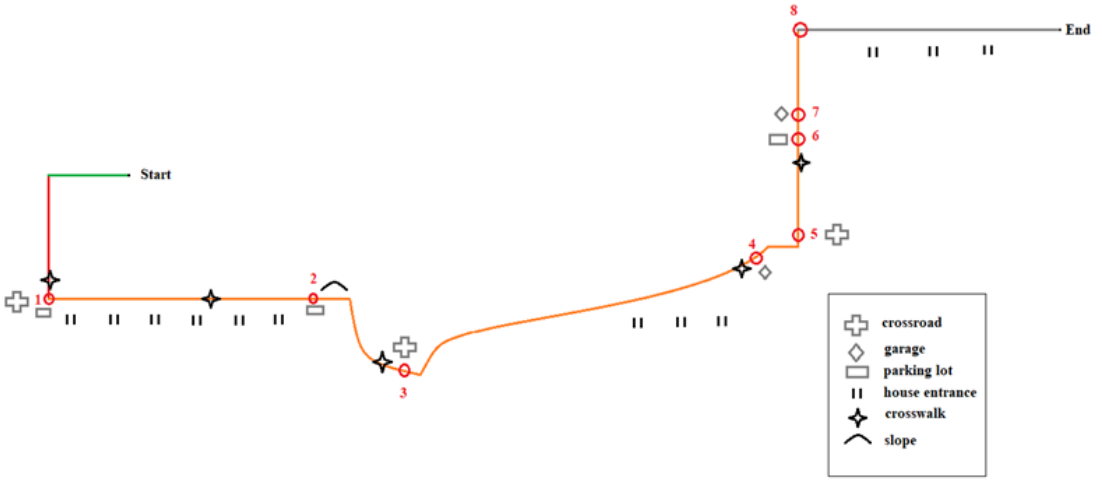


FIGURE 3.5: Simple schematic representing different scenarios.

path with high traffic density, from pedestrians walking in different direction to cargos. The danger of collision is considered high in this area. The red line represents a narrow footpath of 2,5 m. At the end of the path is a high traffic area with three scenarios: crosswalk (pedestrians and pets), exit of parking lot and crossroad with slopes (cars).

Near house entrances is high percentage of pedestrians (with pets in some cases). The grey line represents the entrance of the residential buildings (pedestrians) and is a functional road or a driving route (cars); which means a high traffic area with participants of different speeds. The scenarios can be generally categorized as follow:

FIGURE 3.6: Left: Illustration of the first danger zone (crossroad). Right: Illustration of danger zone 2 (parking lot).

FIGURE 3.7: Left: Illustration of danger zone 3 (crossroad). Right: Illustration of danger zone 4 (garage).

FIGURE 3.8: Left: Illustration of danger zone 5 (crossroad). Right: Illustration of danger zone 6 (parking lot).

FIGURE 3.9: Left: Illustration of danger zone 7 (garage). Right: Illustration of danger zone 8 (residence main road).

- *High Dynamic obstacles:* obstacles with high velocity (e.g., cars) defined as the 8 red marked spots and can be either a crossroad, parking lot, or an underground car park we can simply call garage.
- *Low Dynamic obstacles:* obstacles with low velocity (e.g., pedestrians, pets, bikes, etc.) are the ones present in between every critical spot and can be a crosswalk,

house entrance, or generally a sidewalk or a pedestrian area.

- *Infrastructural obstacles:* lightly present in some areas and defined as few slopes and a narrow area.

3.1.4 Maneuver Selection

Due to the fact that the delivery robot is using a pedestrian and cycle path (not enough cars, one defined lane, no traffic lights and no need for parking), the maneuvers performed are simple longitudinal and lateral maneuvers. It is also extremely important to notice that in case of a critical situations like unexpected behavior of the robot in some scenarios, the automatic mode should be immediately switched to manual mode for a user intervention as a part of vehicle safety. The maneuvers in automatic mode are already defined in subsection 3.1.1 and will be selected depending on the scenarios presented in section 3.1.3. The results are shown as follows in Table 3.2.

- *Collision Avoidance:* A full stop after detecting a high-speed moving obstacle (cars or other vehicles) or a low speed (pedestrians, pets, etc) moving obstacle then a free drive after the path is clear from any obstacle.
- *Obstacle Avoidance:* An obstacle avoidance algorithm is responsible for detecting an obstacle then controlling the robot automatically. Both longitudinal (v) and lateral (ω) deviations will take place.
- The speed should be decreased before any elevation of the ground up to 14.04°(slopes) or as to run smoothly in narrow areas.

	Scenarios	Maneuvers
High Dynamic obstacles	Crossroads	Collision Avoidance (Full Stop, Free Drive)
	Parkings	Collision Avoidance
	Garages (underground parking lots)	Collision Avoidance
Low Dynamic obstacles	Crosswalks	Collision Avoidance (Full Stop, Free Drive)
	House entrances	Obstacle Avoidance
	Pedestrian area	Obstacle Avoidance
Infrastructural obstacles	Slopes	Slow Down
	Narrow area	Slow Down

TABLE 3.2: Maneuver selection for different scenarios.

N.B Maneuvers as, turn left, turn right, and follow lane can be skipped in this case since the robot is using a predefined set of waypoint as way of path generation then a

controller is used for path tracking purposes (pure pursuit) which will be discussed in detail in the following chapters.

3.2 Hardware Architecture

As stated previously the Pioneer 3-DX robot is the main test vehicle used in this project. It is a two-wheeled robot, relying on differential drive type. The robot is well-known by its reliability, robustness and it is widely used by the educational institutions and laboratories. As shown in Figure 3.10, a set of hardware components is used with the

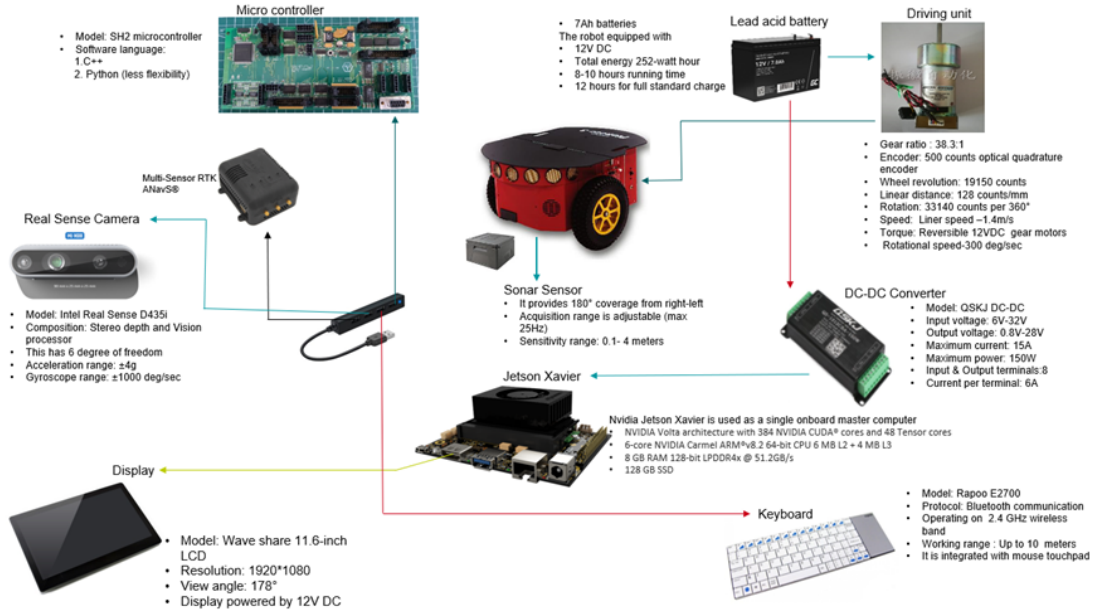


FIGURE 3.10: Hardware setup.

test drive and can be divided into two main categories : *Pioneer 3-DX hardware* and *external hardware*. The Pioneer 3-DX hardware are the different components already equipped on the robot:

- The three 7Ah (21Ah in total) lead acid batteries delivering 12V DC voltage with a total energy of 252 Wh. The Pioneer 3-DX can run continuously for 8- 10 hours with these batteries and up to 4 hours with onboard computer.
- The front Sonar arrangement can provide an 1800 sensing coverage, from the most right to the most left. The Sonar setup provides object detection and range information for collision avoidance.
- The driving unit is composed of two high-torque reversible 12V DC gearmotors, one on each side with a gear ratio of 38.3:1. The motors are equipped with 500

counts high-resolution optical quadrature shaft encoders for precise position and speed sensing. The robot can achieve a maximum linear speed of 1.4 m/s and rotational speed of 300 deg/s. in addition to that, the pioneer 3-DX can overcome a step of 20 mm and a gap of 89 mm. It can drive over a slope with a maximum grade of 25% and it is suitable for the wheel-chair accessible paths.

- The embedded SH-2 microcontroller supported by the ARIA library built originally in C++ language. This library can be used to develop algorithms either fully in C++ (or Python with less flexibility) in Microsoft Visual Studio for example, or it can be used within ROS (Robotic Operating System) middleware using the node RosAria. The SH-2 microcontroller cannot work independently and operate the robot. It needs a constant communication connection through the host port with a master computer that can be a laptop/PC or any other mini-computer board (e.g., NVIDIA Jetson Xavier), which can run the main algorithm. The SH-2 is considered more, as a low-level microcontroller that receive the data from sensors such as Sonar and send action commands to the actuators such as driving motors. The SH-2 is mainly manipulated by the master computer device as it is a master-slave communication type protocol.

The external hardware are the additional components installed on the Pioneer 3-DX. These components are not equipped originally on the robot; they were added additionally in the framework of the commissioning, to make the robot more suitable with the applications of this thesis:

- The Nvidia Jetson Nano a single on-board master computer which runs the main algorithm. The board has a 4GB of RAM 64-bit LPDDR4 at 25.6 gigabytes/second, quadcore ARM® A57 CPU and 128- core NVIDIA Maxwell™ architecture-based GPU. The Jetson board operates on Linux system with Ubuntu distribution. This development board is special because it combines CPU and GPU capabilities in one compact board, where the latter is essential in running computer vision and AI algorithms.

- The Intel RealSense D435i depth stereo camera plays an important role in the perception setup as it is considered somehow the eye of the robot and helps to reinforce the environmental data and by its turn help in building a better environmental model. It is very efficient in outdoor environment, which is the main domain of operation of the project. The advantage of using this camera in the project is for its compatibility with ROS middleware that is been used in the development of the overall algorithms, where several ROS topics are available for use.

- The HMI display to allow monitoring and interfacing with the robot, an 11.6-inch LCD capacitive touch. The choice of this particular display monitor is due to the fact that is fully functional with touch ability with the NVIDIA Jetson board. It has a resolution of 1920x1080 accessible through the HDMI port with IPS toughened glass display. To enable the touch functionality, the monitor should be connected by standard USB cable to the Jetson board. The display is powered by a 12V DC source.

- The wireless Keyboard For efficient teleoperation control, the robot was equipped with a mini wireless keyboard with an integrated mouth touchpad working via Bluetooth communication protocol with the onboard computer. By this keyboard, the robot can be driven manually as desired, select features or stop the robot in case of emergency.

- The two QSKJ DC-DC converter which are very useful in terms of providing adjustable regulated voltage for different additional accessories that will be mounted on the robot. The converter in use, can accept an input voltage range between 6V to 32V and can deliver an output voltage between 0.8V to 28V.

- The ANavS® Multi-Sensor module provides precise position, velocity and attitude information. It includes up to 3 Multi-frequency, Multi-GNSS (GPS + Galileo + Glonass + Beidou) receivers, a MEMS IMU, a barometer, a CAN interface for reception of vehicle data (wheel odometry and steering angle), an LTE module for reception of RTK/ PPP corrections and the powerful ANavS® Sensor Fusion on a single board.

- The adapter to include all external components to the Pioneer 3-DX.

3.3 Software Architecture

As shown in Chapter 2 section 2.1.2,the automated mobile robots rely on more complex functional architecture and control structure. The Generic functional architecture in Figure 2.7 incorporates 5 main layers: *Perception layer*, *Situation Interpretation layer*, *Decision-Making and Planning layer*, *Trajectory Planning layer* and *Vehicle Dynamics Control layer*, as discussed in details in Chapter 2 section 2.1.2. Therefore, The robot's algorithm for the autonomous application was being developed following the functional architecture introduced before. The strategy of the algorithm development is to begin with a smaller functional algorithm and then expand it as necessary according to the AD generic architecture. This strategy allows to reach more reliable result step by step. The algorithm for the P3-DX robot will be developed within MATLAB/Simulink according

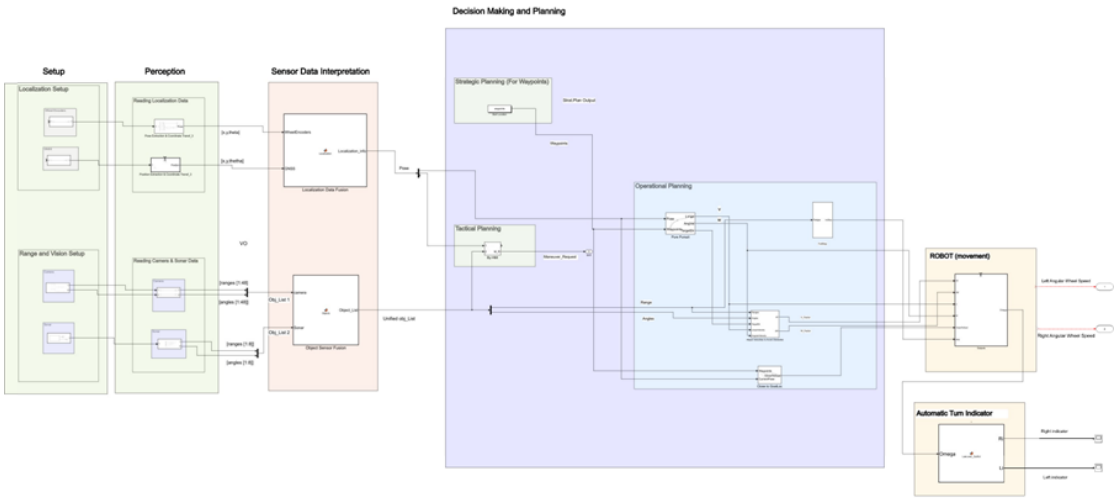


FIGURE 3.11: Software Architecture via Simulink.

to the sense-plan-act concept which is inspired more closely by the functional architecture in Figure 3.11. The selection of this development environment is due to many reasons, the most important one is that Simulink is very efficient to be used by engineers especially from control background, instead of relying solely on pure programming languages such as C++ and python that need high programming skills and especially when dealing with complex autonomous algorithms. It is important to mention that C++ and python are being used also during the algorithm development, but in an integrated way within MATLAB/Simulink. MATLAB/Simulink offer a high degree of flexibility during development, as it offers several important toolboxes, several ready functions blocks to be used and it offers the ability to create custom functions by programming language within MATLAB and then merge it with the main algorithm in Simulink. Moreover, Simulink is characterized by the hardware support for several devices and development board and by a very important feature known as automatic code generation into C, C++... Additionally, MATLAB/Simulink can be coupled easily with other platforms; one of the most important is the *Robotic operating system* (ROS). ROS is very beneficial to be used with Simulink, as many robotics hardware are not directly supported by Simulink such as the case of the Pioneer 3DX-robot. Herewith, it is necessary to have ROS as middleware, on the NVIDIA master robot's computer, between Simulink and the P3-DX robot's hardware. ROS offers the direct implementation of the Simulink model on the physical robot.

The Simulink model in Figure 3.11 consists of several main functional blocks from sensors setup, perception, sensor data interpretation, decision making and planning and low-level control. Each functional block includes sub-functional blocks where all the

main algorithms and architectures are developed. The *Perception block* consists of extracting environmental data from the different sensors (Camera, Sonar, GPS) equipped on the robot. Mainly from wheel encoders, the pose for localization and wheels speeds are extracted. For obstacle detection sonar is used. The main controller in the *oper-*

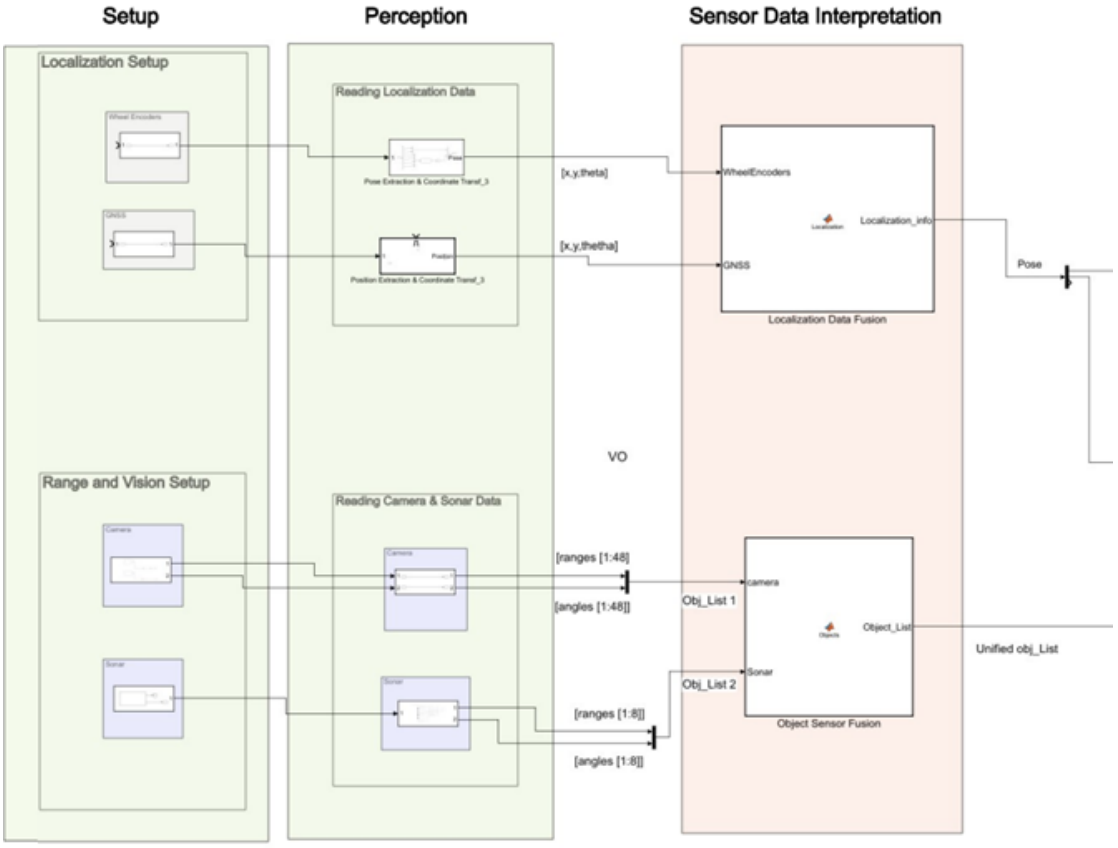


FIGURE 3.12: The perception part of the functional architecture.

ational planning block is chosen to be the pure-pursuit controller type which is very known in mobile robotics applications. It needs as inputs, the actual pose vector of the robot $[xy\theta]^T$ and a waypoints matrix as output of the *strategic planning* block which consist of several desired (x,y) coordinates points for the robot to follow in order to generate a path from start to goal location. Based on the mentioned required inputs and parameters, the controller will calculate the actual linear velocity (v) and heading rate (ω) in order to generate a valid trajectory for the robot.

After defining start, goal locations and set of waypoints for the robot to follow and after receiving the outputs data from the sensor data interpretation block as well as the maneuver selection from the *tactical planning* block, these information are sent to the control and decision-making functional block per request. Based on the pose and

waypoint information, the linear velocity and heading rate are calculated by the pure pursuit controller as discussed before. Then, the range information from the Sonar, is forwarded to MATLAB function block named obstacle avoidance in the operational Layer. The algorithm used in this block is the vector field histogram (VFH) algorithm which computes obstacle-free steering directions for a robot based on range sensor readings (e.g., sonar).

The actuating block will transform the linear velocity (v) and heading rate (ω) received from the functional *low level controller* block into left and right angular wheel speeds. The process in actuating block will occur in the low-level SH-2 robot's microcontroller. The main focus of this thesis will be solely on the decision-making cluster and more pre-

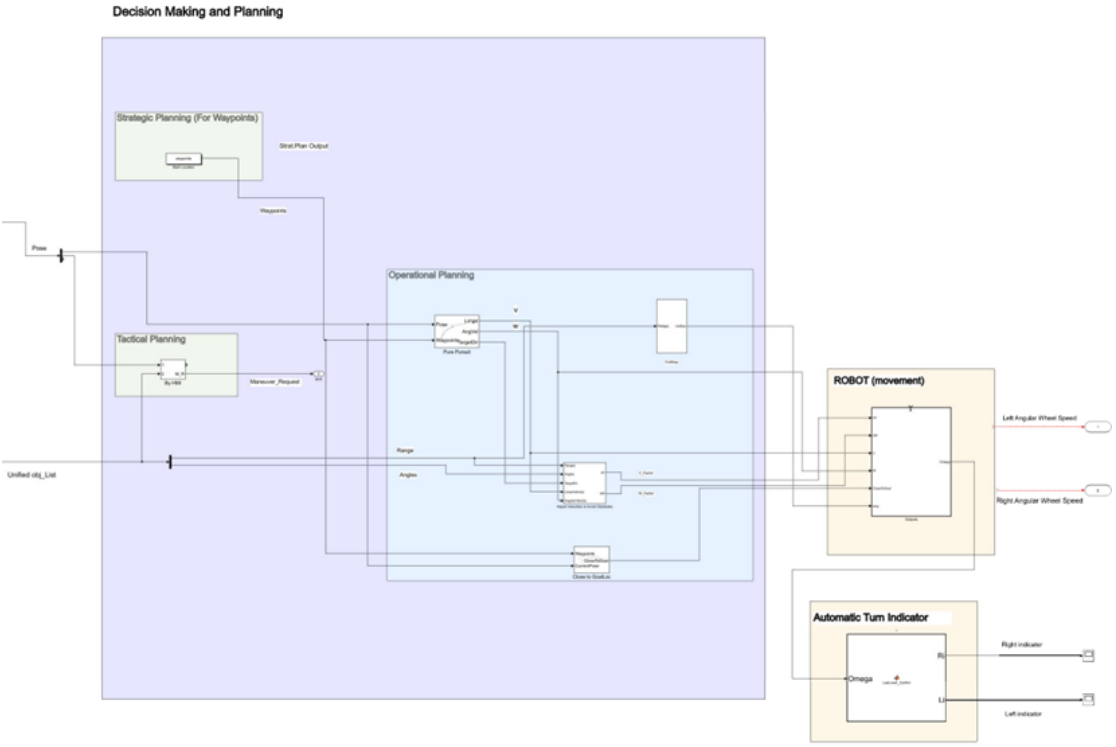


FIGURE 3.13: The planning and control part of the functional architecture.

cisely the tactical planning layer which will be developed and described in detail later on in this thesis.

Chapter 4

Implementation

As mentioned previously in Chapter 3, Figure 4.1 defines the general block diagram which represents the kick starter of this thesis concept design. The resulting block

FIGURE 4.1: Block diagram of the tactical planning layer.

diagram after a detailed study of the use case is shown in Figure 4.2.

To be able to design a concrete tactical planning model for the specified use case it is

FIGURE 4.2: Block diagram of the tactical planning layer for this use case.

important to first answer the following questions : 1. What kind of inputs should be

fed to the robot in order to create a certain Environment Model? 2. What is the most optimal method to transform this environment model described into maneuver requests? 3. What kind of outputs will be the translation of the maneuvers selected?

So, before defining the Tactical planning block which until now is a black box, it is fundamental to explicitly translate the inputs and outputs into the robot's language.

4.1 Model Input: Sensors Data

Until now, the inputs of the model presented in Figure 4.2 are the different use case scenarios defined. In this section, the main goal is to translate these concept input into a language that the robot (Pioneer-3DX) can understand.

In Figure 4.3, it is shown that these concept scenarios are non other than sensors data, extracted from different types of sensors and processed in the Perception block. These sensors are divided as follows:

- *Localization sensors:* which define the position of the robot in a local (wheel encoders) and global (GNSS) frame.
- *Range and vision sensors:* the eyes of the robot; they give an understanding about the surroundings of the robot, for detecting obstacles (sonar) and defining the type of each obstacle detected (camera).

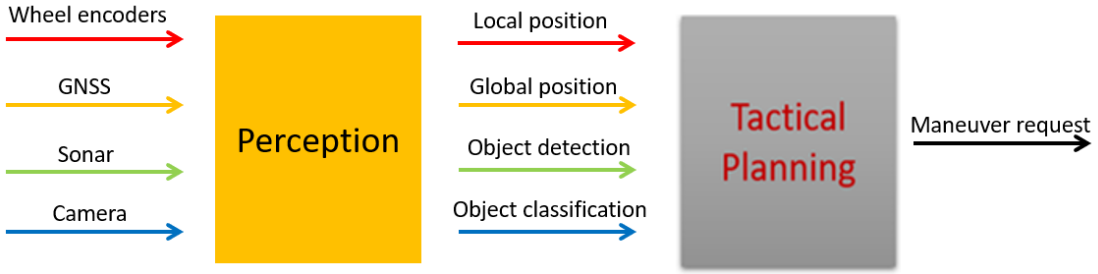


FIGURE 4.3: Block diagram of the tactical planning layer inputs.

The Table 4.1, explains in detail the sensors data input.

NB the camera can also be used as an obstacle detection sensor but it is not in the purpose of this thesis.

Sensors	Task	Data interpretation
Wheel encoders	Local localization (odometry)	x, y, θ
GNSS	Global localization	longitudinal and lateral position
Sonar	Object detection	ranges
Camera	Object classification (and object detection)	object type (and ranges)

TABLE 4.1: Sensors data.

In this thesis, a solution approach to scenario translation into sensor data for environmental recognition was set up, answering the question: *how the robot knows each scenario?* This solution is presented as follows:

- for pedestrian area and house entrances the robot will use sonar sensor to detect the pedestrians via a set of ranges (or distances to the obstacles).
- in crossroads, crosswalks, parkings , garages, slopes and narrow areas defining a set of conflict waypoints (critical waypoints) where these scenarios can occur and feeding them as inputs to the robot so it is always alerted to these kind of situations using wheel encoders to know the position of the robot in local frame and GNSS to compare the local position to a global one for more precision.

This solution approach was adopted depending on the sensors set up at hand. Other solutions are always preferable for better precision and less data load.

4.2 Model Output: Maneuvers

Same solution strategy as last section 4.1, will be applied in this section. First step will be determining the different maneuvers as selected in Chapter 3 section 3.1.4 and defined as three main maneuvers:

- *Collision Avoidance:* is a complex maneuver which can be decomposed into a set of simple maneuvers that can be performed by the robot: Full Stop and Free Drive.
- *Obstacle Avoidance:* by applying a set of variations to the linear and angular velocity.
- *Slow Down:* reducing the vehicle's velocity.

The main controller of this thesis will be the pure-pursuit controller which is a a path tracking algorithm that computes the angular velocity command, which moves the robot

from its current position to a look-ahead point in front of the robot. When the robot reaches that point, the algorithm then moves the look-ahead point on the path based on the current position of the robot. This process repeats until the robot reaches the last point of the path. Thus, the robot is constantly chasing a point in front of it. The algorithm assumes a constant linear velocity, which can be changed at any point.[13]

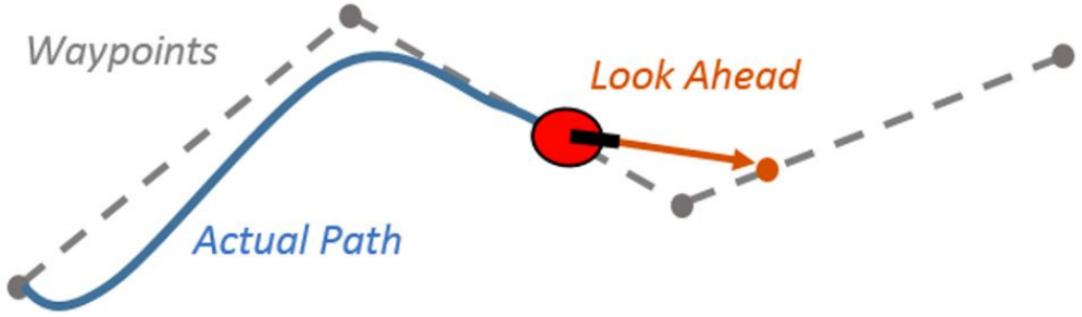


FIGURE 4.4: Path tracking with pure pursuit controller.[13]

This controller outputs the linear and angular velocity planned for the robot. So, the solution idea is to manipulate these outputs logically to be able to define and perform each maneuver cited above as shown in Figure 4.5.

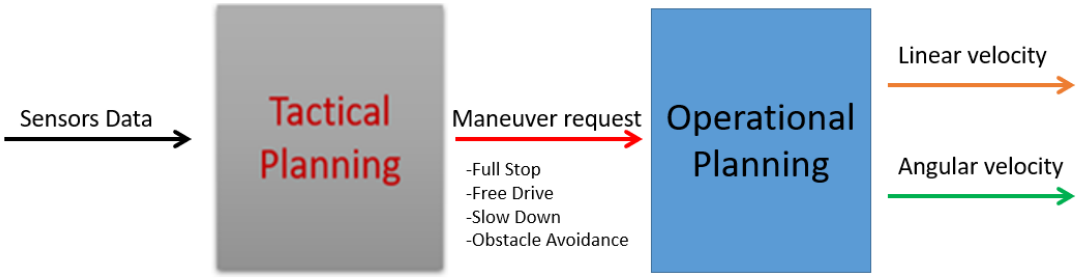


FIGURE 4.5: Block diagram of the tactical planning layer outputs.

The Table 4.2, explains in detail these Maneuvers.

So, the output of this block responsible for performing maneuvers (operational block) will be different linear and angular velocities which will be calculated and fed as information to the robot. With V_{pp} is the linear and angular velocity applied by the pure pursuit controller, the main idea is to apply a set of gains which will have a range of [0,1] to perform the maneuvers requested from the tactical planning layer. The choice of these

Maneuvers	Decomposition	Translation
Collision Avoidance	Full Stop (gain = 0)	V_{new}=V_{pp}.0
	Free Drive (gain = 1)	V_{new}=V_{pp}.1
Obstacle Avoidance		differential linear (dv) and angular (dω) velocities
Slow Down		V_{new}=$V_{pp}.gain_{slow}$

TABLE 4.2: Maneuvers.

gains will be explained in details later on in the next Chapter.

Important remarks

- There exist more optimized path following controllers than the Pure Pursuit that are exploited in the automated driving field like Model predictive control (MPC) which is is a feedback control algorithm that uses a model to make predictions about future outputs of a process.
- The difference between this approach and the one presented in Chapter 3 section 3.3 is that for the old model (Figure 3.11) only one maneuver at a time can be performed. On the other hand, this model approach allows the system to manipulate more parameters because of the allocation of tasks in different blocks which will optimize the data transfer and delete any redundancies. The integration of the tactical planning block in the model also allows more flexibility for the model to include different maneuvers. This can be seen later on in the next section of this Chapter.

4.3 Concept Solution

From Figure 4.2 and from the sensors data set up mentioned in last section 4.1, the concept solution idea comes in two main cases shown respectively in Figure 4.6 and Figure 4.7:

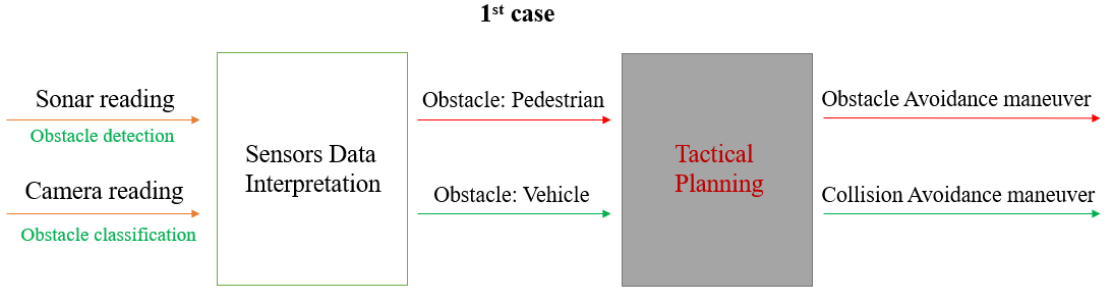


FIGURE 4.6: First concept solution case using range and vision sensor.

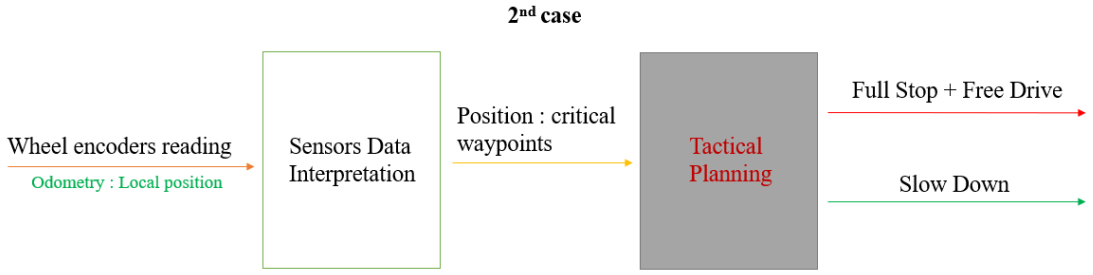


FIGURE 4.7: Second concept solution case using odometry.

In case 1, the sonar sensor will detect the obstacle and the camera will determine the obstacle type. These information are important to decide which maneuver to take depending on the obstacles type : if it's a pedestrian then robot has to perform an obstacle avoidance maneuver, on the other hand, if the obstacle is a vehicle then the robot has to perform a collision avoidance maneuver.

In case 2, the wheel encoders will give information about the position of the robot which will be compared to the critical waypoints explained in section 4.1 and the maneuvers will be selected depending on which set of waypoint the robot is close to : if the set of waypoints define the position of a slope then the robot has to slow down, or if it defines other scenarios (crossroads, crosswalks, garages, parkings) then the robot has to perform a full stop ; wait ; then free drive maneuver.

NB In case 1 a camera can be used directly for both obstacle detection and classification of object as shown in figure 4.8, but this concept solution is not relevant for this thesis.

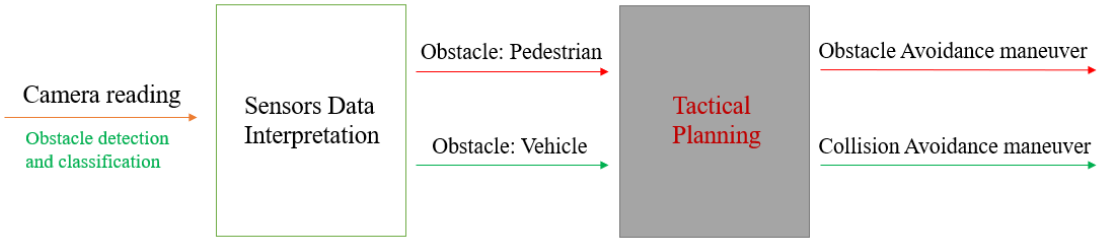


FIGURE 4.8: Extra case for using camera.

Now that it is clear what are the inputs and outputs of the tactical planning block as well as the relation established among these data it is the right time to decipher what is this black box called tactical planning block.

In this section, three methods are described to define the tactical planning block.

4.3.1 Rule-based Solution Approach

As defined in Chapter 2, a rule-based approach uses rules as the knowledge representation. Hence, in the following, a rule-based solution for the tactical planning block will be described using : If-Else solution and Finite State Machine solution.

4.3.1.1 If-Else Solution

An If-Else statement in programming is a conditional statement that runs a different set of statements depending on whether an expression is true or false. It is a very basic programming logic which is helpful with manipulating different conditions of a specific system. The syntax of this logic is shown in Figure 4.9.

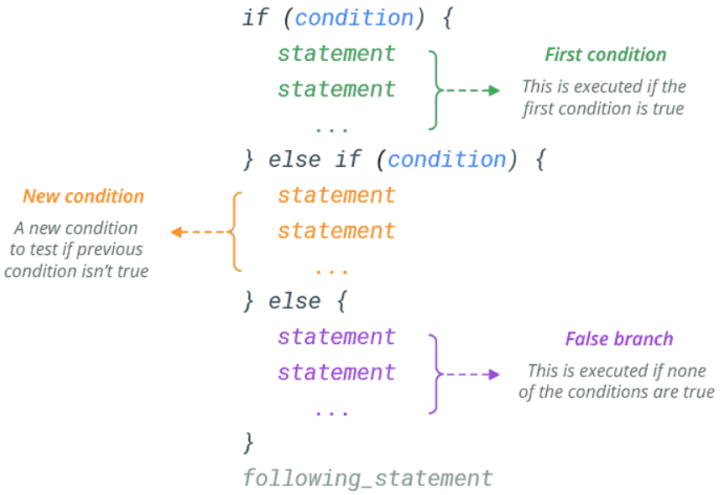


FIGURE 4.9: If-Else statement syntax.[14]

Following this syntax, the first concept solution for this thesis tactical planning was proposed in Figure 4.10, which represent *Nested if* statements.

First, the inputs and outputs for this logic will be defined as follows:

- *Inputs:* or the conditions which represent the sensors data : sonar (range sensing data), camera (vision data: classification) wheel encoders (localization reading data) readings.

- *Outputs:* or the statements which represent the simple and complex maneuvers selected in each condition : obstacle avoidance, collision avoidance, slow down , full stop and free drive.

```
if position == not_close_to_goal then
 if range == obstacle_detected then
  if obstacle == human then
     Maneuver_selection = Obstacle_Avoidance;
  elseif obstacle == vehicle then
     Maneuver_selection = Collision_Avoidance;
  else
     Maneuver_selection = Full_Stop;
     delay t;
     Maneuver_selection = Free_Drive;
 elseif range == obstacle_detected_critical then
    Maneuver_selection = Full_Stop;
    delay t1;
    Maneuver_selection = Free_Drive;
 else
    Maneuver_selection = Free_Drive;
else
  Maneuver_selection = Slow_Down;
  delay t2;
  Maneuver_selection = Full_Stop;
```

FIGURE 4.10: If-else concept solution.

This concept solution was developed theoretically and a part of it will be tested in the validation part which will be explained in detail later.

As shown in Figure 4.10:

- The conditions depend on the sensors data explained in detail in Table 4.3.
- The statements depend on the maneuvers which are divided into two types: 1- *complex maneuvers:* e.g. obstacle avoidance (for low speed obstacles: human) and collision avoidance (for high speed obstacles: vehicle) ; these maneuvers can be developed following pre-existing algorithms e.g vector field histogram algorithm (VFH) for obstacle avoidance. 2- *simple maneuvers:* full stop, free drive and slow down; which can be developed manually only from manipulating the linear velocity defined in the path planner (e.g. pure pursuit).

Sensors	Data type	Condition 1	Condition 2	Condition 3
Sonar	Range reading	Obstacle is detected	Obstacle is detected in critical situation (the robot is very close to the obstacle)	Obstacle is not detected
Camera	Classification	Obstacle is human	Obstacle is vehicle (car)	Obstacle is not human nor vehicle (unknown obstacle)
Wheel encoders	Localization reading	Robot is close to the goal location selected	Robot is not close to goal location	

TABLE 4.3: Sensor data conditions.

NB The delays mentioned in the concept solution depend mainly on the linear velocities: t is the delay performed while waiting for the unknown obstacle (e.g pets etc...) that will have a specific linear velocity or speed so as to not collide with it; *t1* is performed after the abrupt stop of the robot to give it time to run again; *t2* is performed between two maneuvers to be able to perform them sequentially.

This if-else solution is only useful when there isn't much sensor data to process. It will be more complicated when other sensors are added to the list (LIDAR, Radar, GPS,etc...) which means more conditions, means more lines. Then, it will be complicated to manipulate the conditions and statements as to perform the wanted tasks. The solution for that is to switch to states and transitions, which is the main idea of the next subsection.

4.3.1.2 Finite State Machine (State Flow) Solution

As discussed before in Chapter 2, finite state machine (FSM) is a method of modeling a system comprised of a limited number of modes depending on which mode it's in, the machine will behave in one manner or another. It is a method mostly used for systems with abrupt changes (discontinuous) with complex software design process. It is a graphical approach for modeling systems using state-transition diagrams. Nevertheless, why use finite state machine?

In case of few inputs to the system the previous approach cited in the last section (if-else approach) is convenient. However, when multiple inputs are included it causes the possibility of having many conditions and situations emerging in the system. Trying to handle all possible situations using if-else approach will make the system end up with a long list of nested If-Elseif-Else statements in its code each one checking internal and global variables in order to generate an output of a decision making system. This

approach will be challenging because of the various possible combinations of variable which should be processed which will lead to a missing or flawed logic because of the difficulty of this approach in this kind of situation.

One way to drastically simplify this, is to instead thing about the system's software as a comprised of a fixed number of states each state can be conceived of as its own little universe enabling the programmers to more clearly define rules and desired behavior under a particular set of circumstances. So, rather than continue with every possible outcome when the sensor data is inputed to the system, it is possible to manage these sensor data in a set of states each one for a particular function depending on the sensor data selected. Hence, by compartmentalizing the problem into smaller segments (states) it is easier to satisfy the systems goals.

Now that it is clear when to use FSM, what is the best way of expressing them? One approach is to use state-transition diagrams which display the system visually and make the connections between the states more apparent. Once these diagram semantic is understood by the users, these are indeed worth a thousand words.

In this thesis a deterministic FSM logic was used to describe the tactical planning system and defined in Figure 4.11:

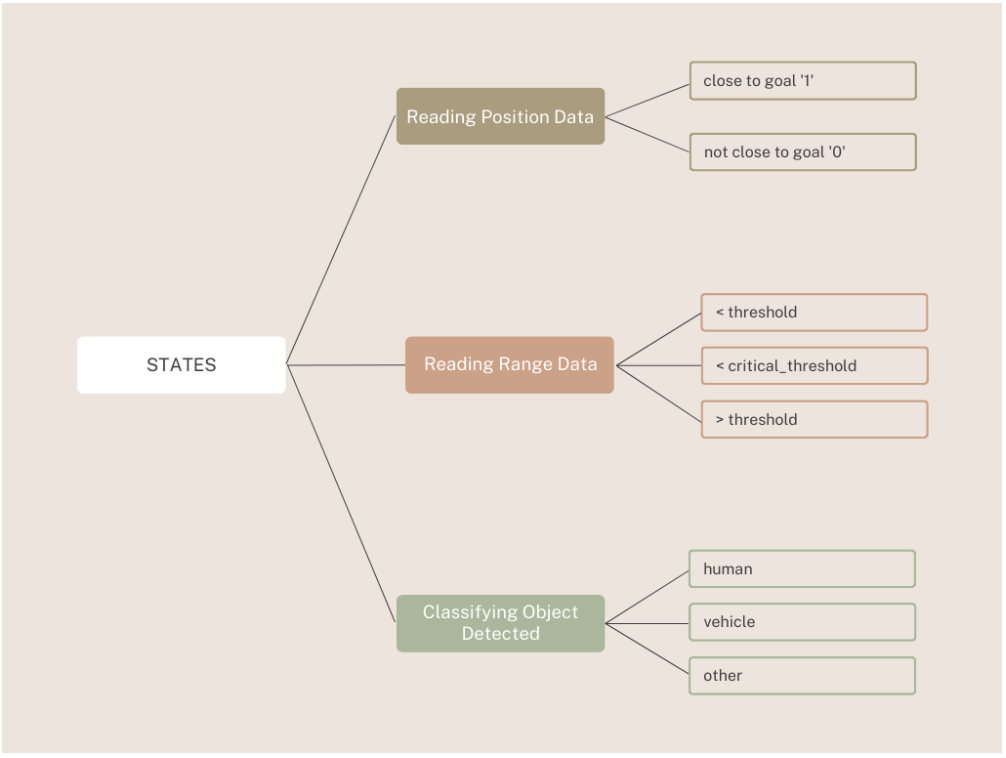


FIGURE 4.11: Tactical Planner States.

The *transitions* of the tactical planner are obviously the conditions performed on the sensors data.

As of the variables of the tactical planner can be defined as follows:

- *Input:* Sensors Data = {Position, Range, classification}.
- *Output:* Maneuver Selection= {Obstacle Avoidance, Collision Avoidance, Full Stop, Free Drive, Slow Down} .

The computing tool for designing the state transition diagram used for this system is *State Flow* which is a graphical programming environment based on finite state machines consisting of states, transitions, and data and works within Simulink model. State Flow models logic patterns such as decision trees and iterative loops. Flow charts represent combinatorial logic in which one result does not depend on prior results. Flow charts can be built by combining only connective junctions and transitions. The junctions provide decision branches between different transition paths. Executing a flow chart begins at a default transition and ends at a terminating junction, which is a junction that has no outgoing transitions.

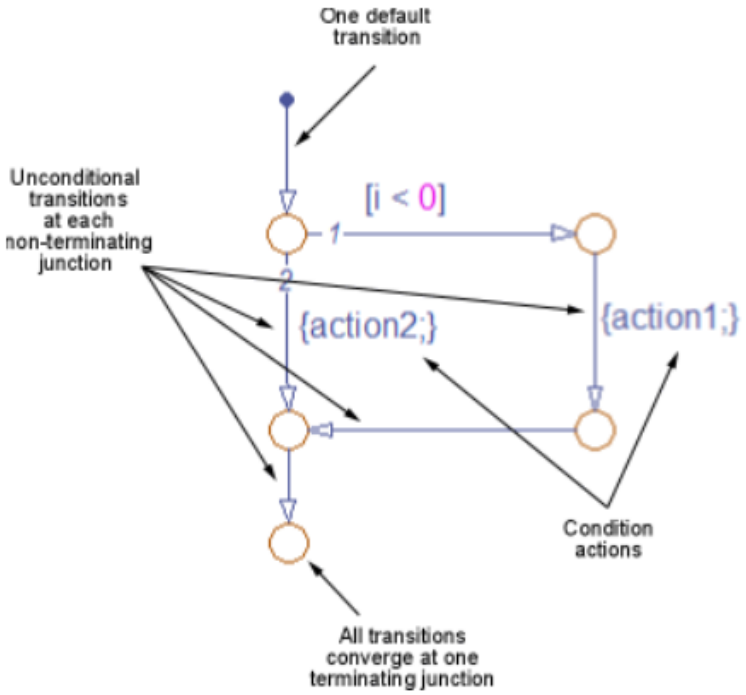


FIGURE 4.12: Practices for reating Flow Charts.[15]

The flow chart designed for this thesis concept is a nested If-Elseif-Else. The syntax of this type of flow chart format is shown in Figure 4.13.

The final flow chart concept solution of this thesis is represented in Figure 4.14. 4.11:

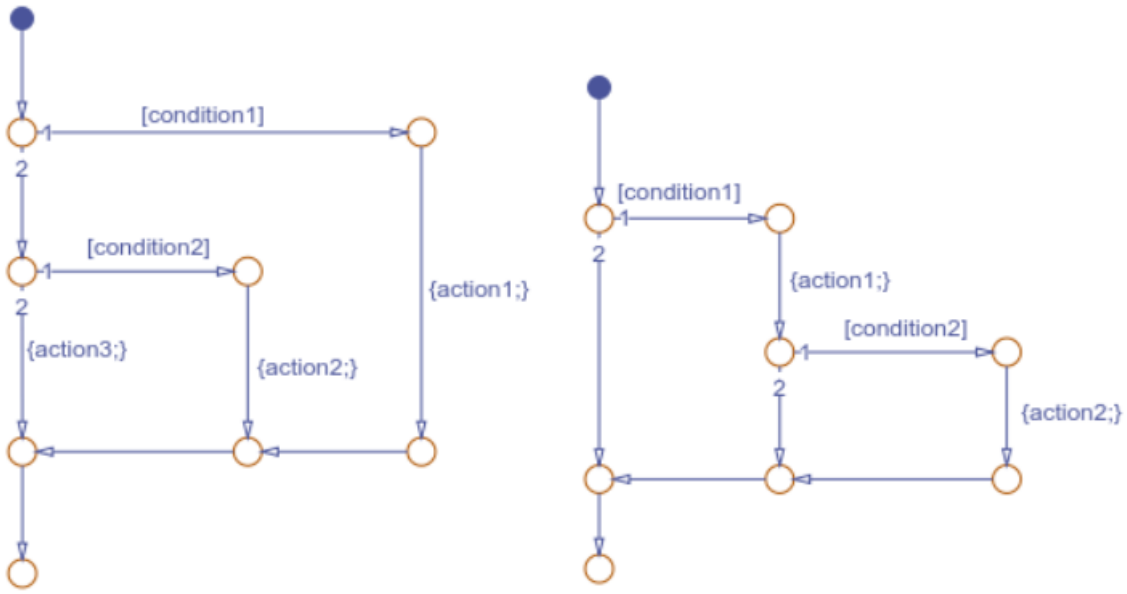


FIGURE 4.13: Left: Syntax of a If-Elseif-Else model. Right: Syntax of a Nested If.[15]

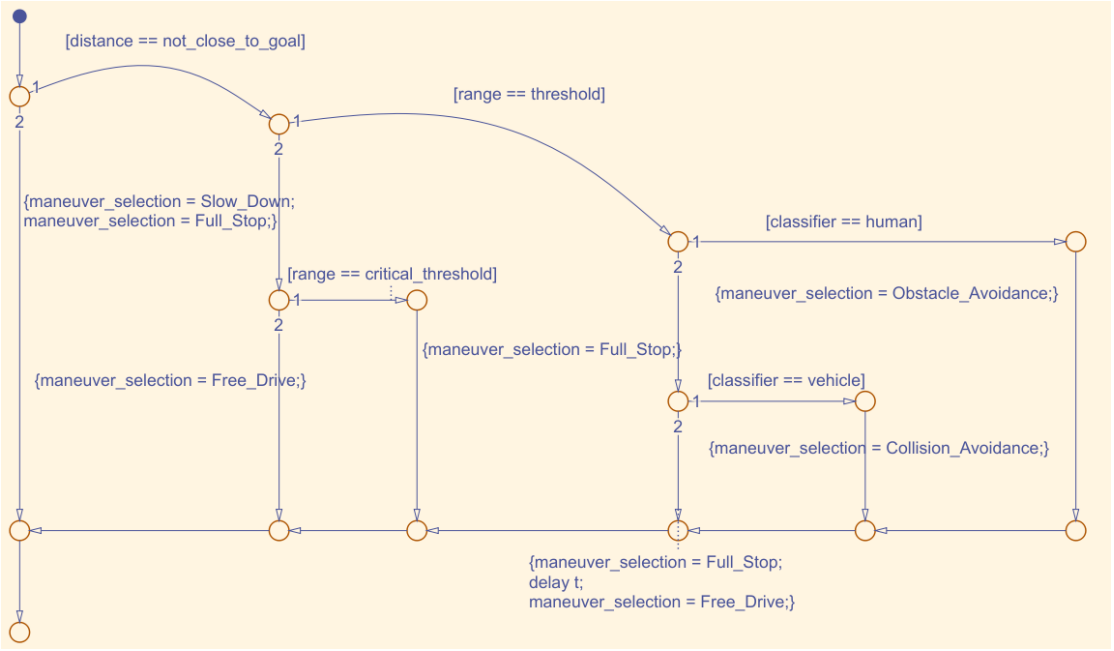


FIGURE 4.14: State Flow Concept Solution .

Concludingly, FSM or State Flow are important when having a rather large number of inputs or states. It is also user friendly and can be useful for algorithm optimization and troubleshooting since all the errors can be visualized and easier to predict in contrary to the written form of the algorithm (If-Else). Also, Simulink is a good interface for creating these types of models.

Nevertheless, some of the limitations of FSM-based studies are that they are fully reliant on knowledge certainty and cannot be generalized to unknown scenarios. They are unable to manage large complex networks, resulting in state and transition explosions. To have a completely reactive system; each state must be able to transit to every other state making it a fully connected graph. Therefore, the modification of FSM is labor-intensive making them difficult; to collaborate as they expand.[16]

This is where the rule-based approach becomes useless to these type of unpredicted scenarios and machine learning approach is the most convenient solution approach.

4.3.2 Machine Learning Solution Approach

In this section, the main goal is to define what can be the methodology of a ML solution approach depending in the learning task of the machine learning model. So, a new wording will be defining the tactical planning model using a ML approach; *intelligent tactical planning*. For that, it is important to answer the following questions: what is the learning task of the ML model? And what are the inputs and outputs of this model?

Previously, a set of scenarios where predefined to explain the tasks of the tactical planner.

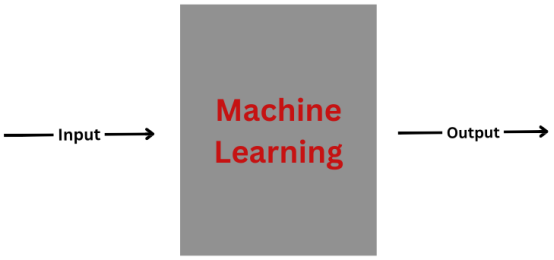


FIGURE 4.15: Machine Learning Solution Model.

But, real word scenarios are unpredictable and it can be quite hard for a developer to include all possible cases. These tasks can be given to an active system which can learn from past experience and can be able to predict future behaviour of other traffic participants.

The most used machine learning method for decision making in automated driving is: *Reinforcement Learning.*

As mentioned previously in Chapter 2, Reinforcement Learning (RL) is a branch of machine learning that deals with the issue of automatic learning and the best choices over time. RL enables an *intelligent agent* to learn from its mistakes and experiences. The RL agent acts in the environment to receive rewards where the goal of the agent is to choose the action that will maximize the expected cumulative reward over time. An RL agent can be characterized as a Markov Decision Process in the following way: the agent interacts with the environment by taking actions and getting feedback and rewards. As shown in Figure 4.16 at each time step t the agent obtains a representation

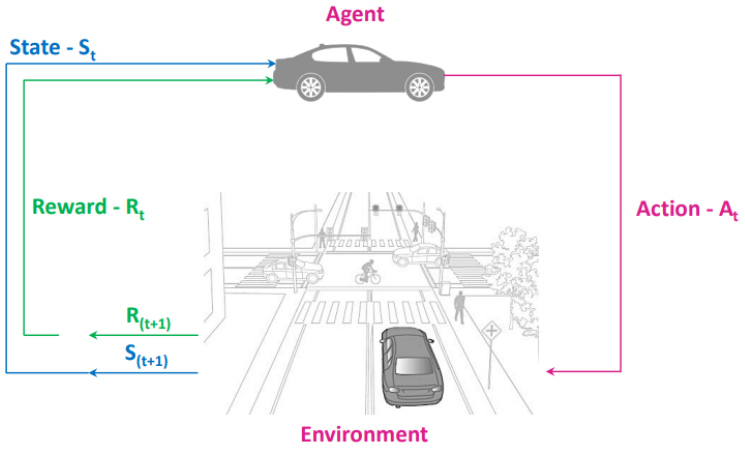


FIGURE 4.16: Reinforcement Learning Approach.[16]

of the environment state $s_t \in S$. Based on this state the agent decides on a course of action $a_t \in A$. Any action is chosen depending on the agent's behavior, often known as the *policy* which instructs the agent on the best course of action for every potential state. As a result of each action, the agent is rewarded with a reward $r_t \in R$ and observes the next state $s_{t+1} \in S$. The process of receiving a reward can be described as an arbitrary function, f . At each time t, we have:

$$f(s_t, a_t) = r_t$$

[16]

Reinforcement Learning plays an imperative role in the decision-making process of automated driving system, which enables automated vehicles to acquire an ideal driving strategy through constant interaction with the environment. It is simply a feedback approach that makes the vehicle, in this case called agent, interact with the environment in a real time manner and decide on actions depending on these interactions.

So, answering the questions mentioned in the beginning of this section: the learning task of the tactical planner is to decide on the most optimal behaviour (maneuver) in a specific state using the cumulative reward and depending on the environment representation in that state; the input and output of the system are as shown in Figure 4.17.

There are different RL methods for solving decision making problems for autonomous

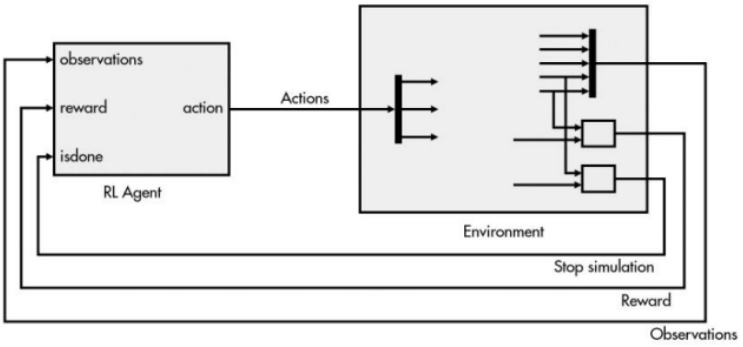


FIGURE 4.17: Diagram of a Simulink Environment for a RL agent.[17]

vehicles in urban environments which are defined in Figure 4.18.

But, since in this thesis, all the tests were done in a predefined, predictable structured

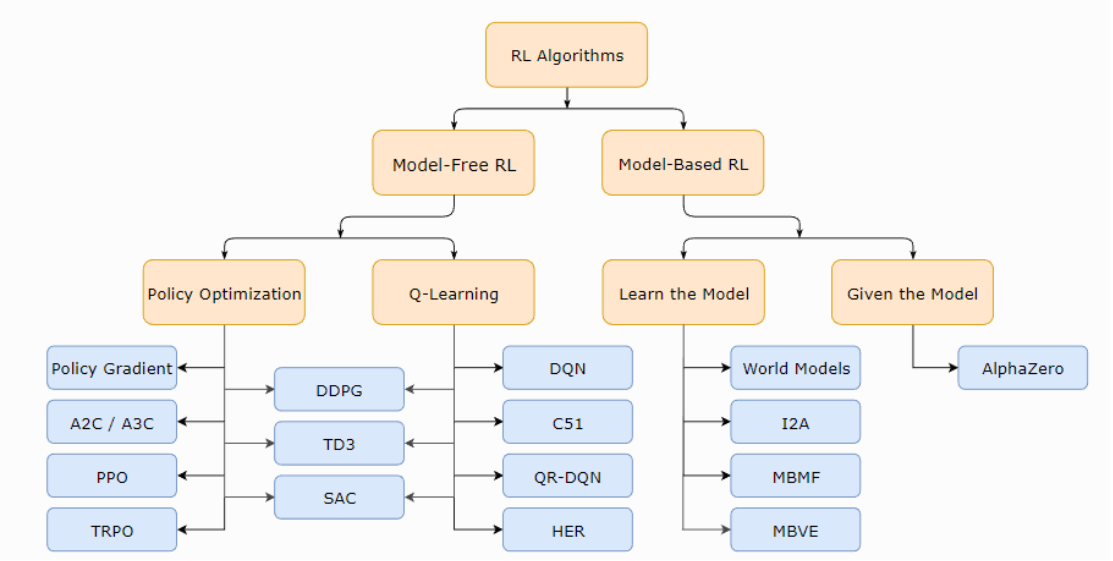


FIGURE 4.18: Reinforcement Learning taxonomy.[18]

environment (Laboratory). the use of AI will be useless in this matter. However, the use case represents an urban environment where it is important to implement AI methods to make the robot more interactif with its environment. This will be among the future work of this project. For more information about how to use AI methods for

decision making, see the following reference of my bachelor thesis were it was tackled and explained in details [36].

4.3.3 Methods Analysis

To sum up this chapter, it is important to mention the advantages and disadvantages to each method selected above, in order for the reader to have a glimpse on which method is best to use depending on the use case or test environment.

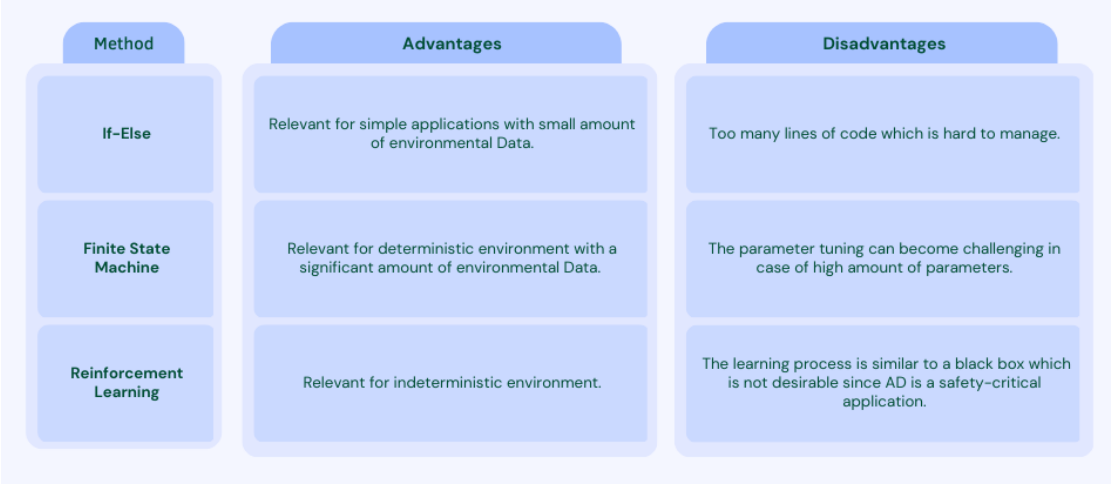


FIGURE 4.19: Defining the advantages and disadvantages of each solution method.

Chapter 5

Validation

In this Chapter, a validation of the concept model discussed previously will take place to prove the accuracy of the model for future usage.

For that, two tests will be done using the Pioneer 3-DX as the test vehicle and ROS as connection interface between the model on Simulink and the test vehicle.

In the following, the models development will be discussed in detail and the results will be shown. A detailed description of the ROS commands used to run the robot and implement the model to the robot will be discussed in appendix.

5.1 First Test : Testing the Logic Part

The Simulink model for the tactical planner will be tested using the robot's sonar readings as input and the output of the model is shown on the Simulink interface using a dashboard for visualising the results (Figure 5.1).

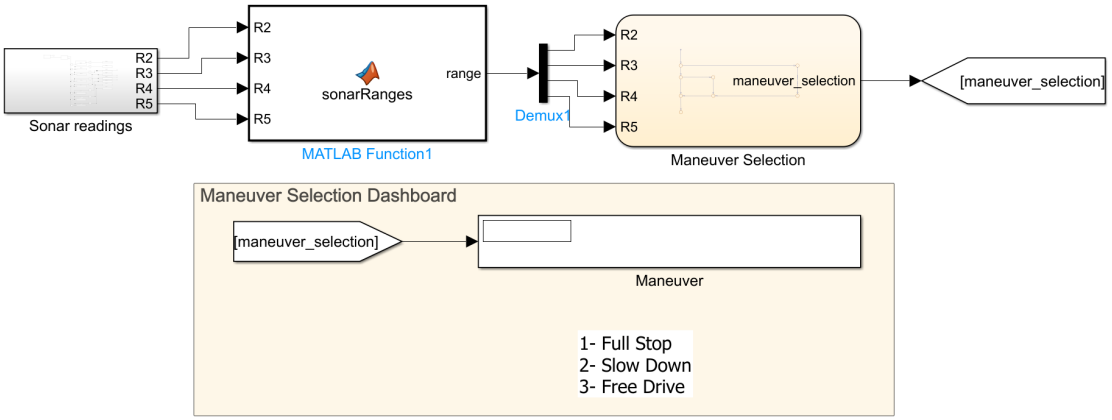


FIGURE 5.1: Model Logic test.

The Simulink model in Figure 5.1, will test the logic of the tactical planning model developed, it is divided into two main parts: the perception and data interpretation for sensor data preparation and the tactical planning part for sensor data processing and maneuver selection.

Perception and Data interpretation

The sensor used for this test is the Pioneer 3-DX sonar sensor which is responsible for providing object detection and range information for collision avoidance.

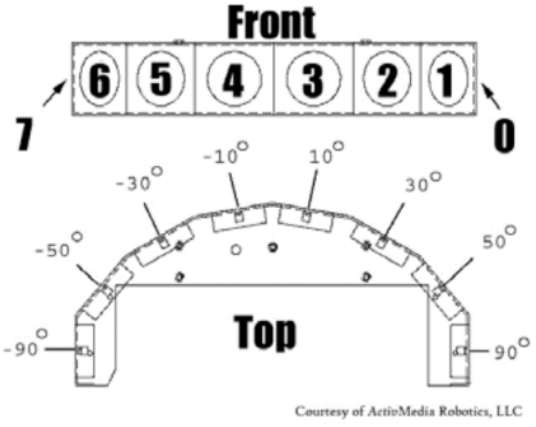


FIGURE 5.2: Front Sonar Arrangement.[19]

For this test, the frontal sonar are used for obstacle detection. The steps for extracting the sonar range data are as follow:

1. Extract the sonar data from the robot using the *Subsribe* block from Simulink ROS package which receive the sonar messages from ROS network in form of point cloud as raw data.

2. The messages are transferred to the *Read Point Cloud* block which extract point cloud data from ROS point cloud message into XYZ cartesian coordinates. Specifying the Maximum point cloud size expected as [height, width] is also registered in this block as of [8,1] for the robot because of the 8 sonar readings it gives.

3. Using a Bus Selector block to extract signals which are needed, in this case X and Y coordinates.

4. Calculating the ranges using the Math function $x^2 + y^2$ for each sonar sensor.

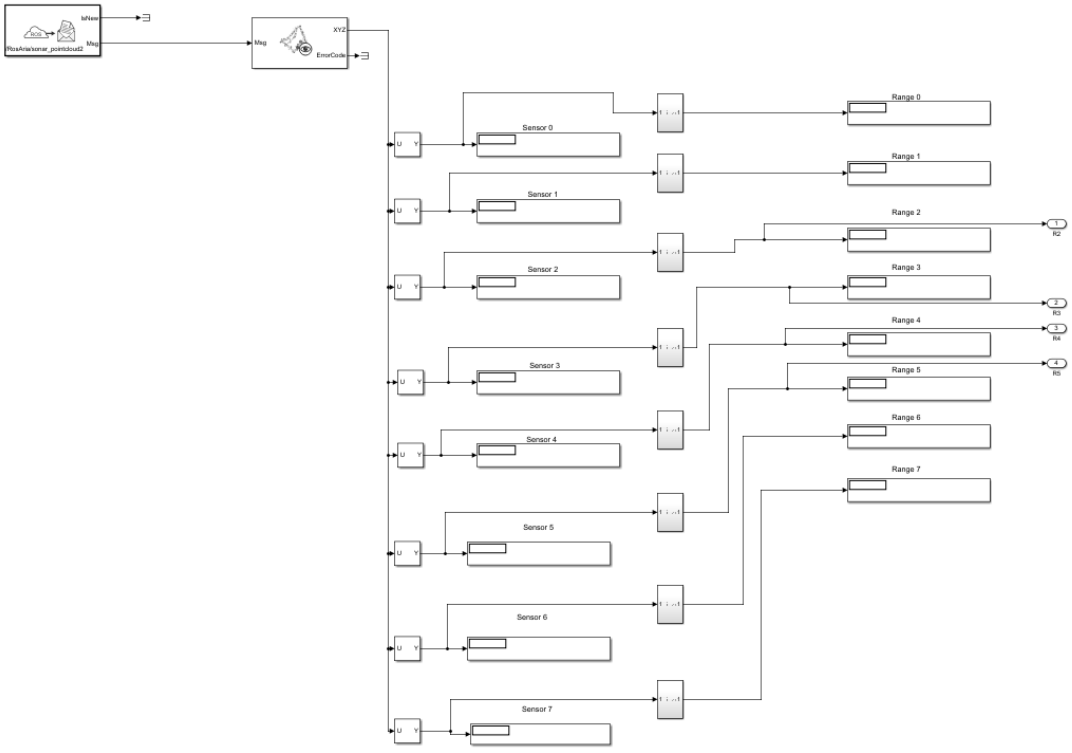


FIGURE 5.3: Sonar Reading Setup.

Since the front obstacles are mostly tested in this case, only 4 sonar outputs are enough, sonar: 2, 3, 4 and 5.

The MATLAB function in Figure 5.1 defines these ranges extracted into variables to manipulate them later on depending on the need.

```
function range = sonarRanges(R2,R3,R4,R5)
range = [R2 R3 R4 R5];
end
```

FIGURE 5.4: Matlab Function for defining ranges.

These ranges are then transferred to the tactical planning block to create conditions depending on the scenarios.

Tactical Planning

The tactical planning block is represented in form of Flow Charts developed in Stateflow environment. It inputs sonar ranges and outputs the maneuvers selected.

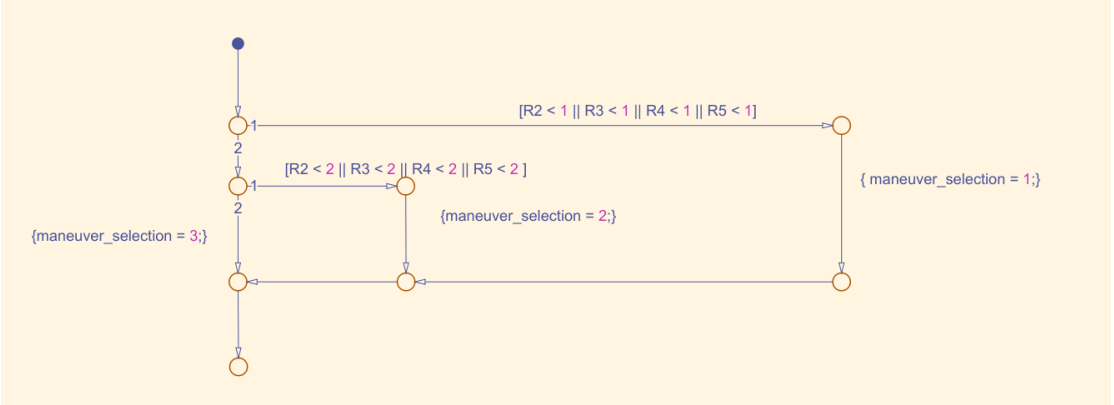


FIGURE 5.5: Tactical Planning Flow Chart.

The maneuver selected are :

1. Full Stop.
2. Slow Down.
3. Free Drive.

As explained in Figure 5.1 dashboard.

The sonar range sensing is from 0.1 m til 4 m. So, depending on that two conditions where selected : one for critical condition in case the robot detects an obstacle in a range of 1 m (very close) the a full stop maneuver should be performed; the other condition is set in case the obstacle is far but still can be the cause of a collision, the robot detects that obstacle in a range of a 2 m so that it slows down (reduces the velocity) until the obstacle gets out of the robot range otherwise it should stop once the 1 m range is

reached as defined in the first condition. In case both conditions aren't satisfied, then the robot can perform a free drive maneuver following the path planning algorithm.

Results

The result of each range condition tested by using a test obstacle in each range and each maneuver selected which will be displayed in the dashboard to check the accuracy of the model are shown in the following figures.

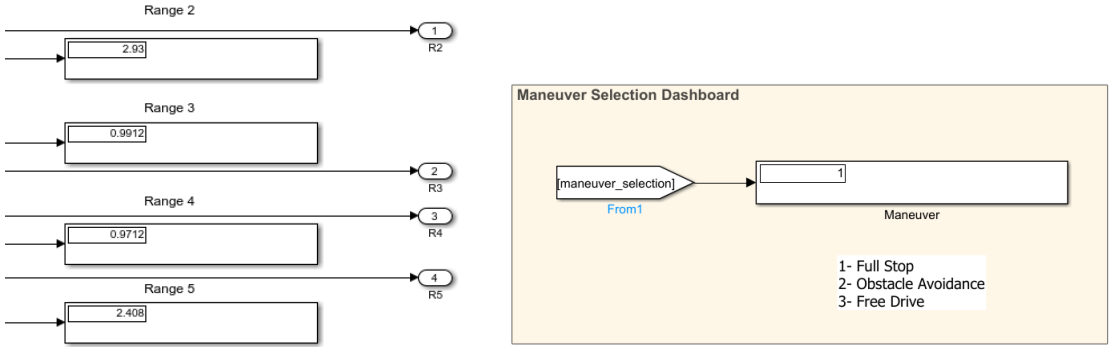


FIGURE 5.6: Left: Sonar range reading in first case. Right: Maneuver selected for the first case (Full Stop).

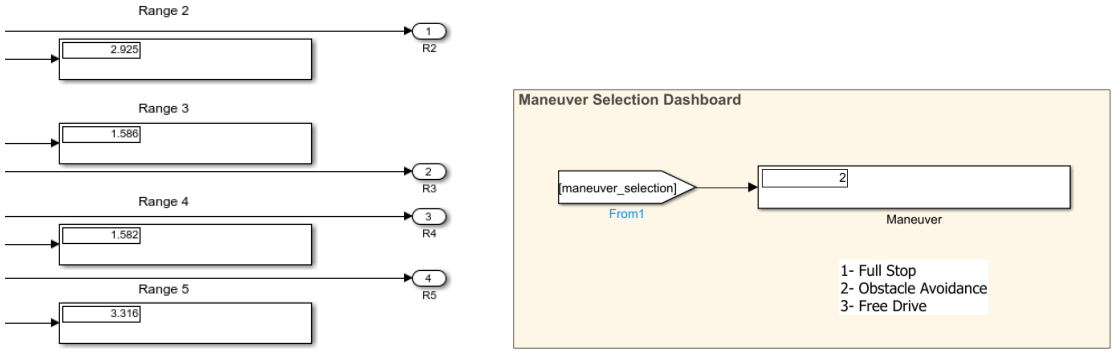


FIGURE 5.7: Left: Sonar range reading in second case. Right: Maneuver selected for the second case (Slow Down).

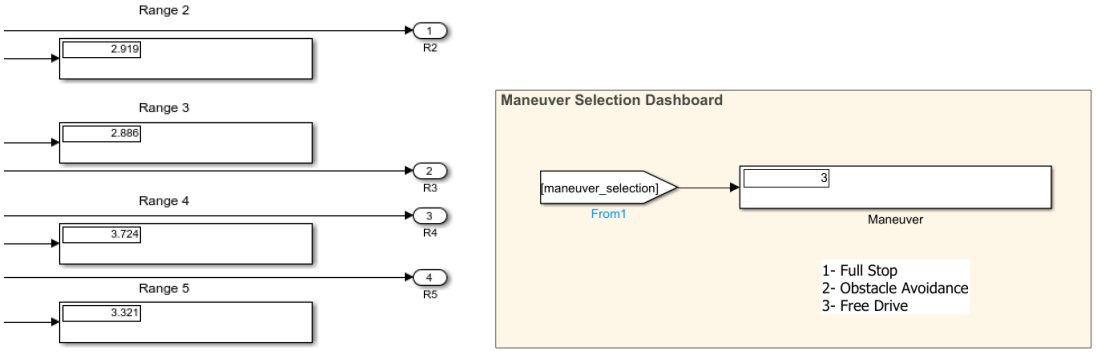


FIGURE 5.8: Left: Sonar range reading in third case. Right: Maneuver selected for the third case (Free Drive).

5.2 Second Test : Testing the Logic and Control Part

In this test, the main goal is to test the model's maneuvers using real sensor readings and velocity parameters. The results are recorded in a video and the Simulink in Figure 5.9 will be the main model which represents the functional architecture mentioned in Chapter 2 (Figure 2.7).

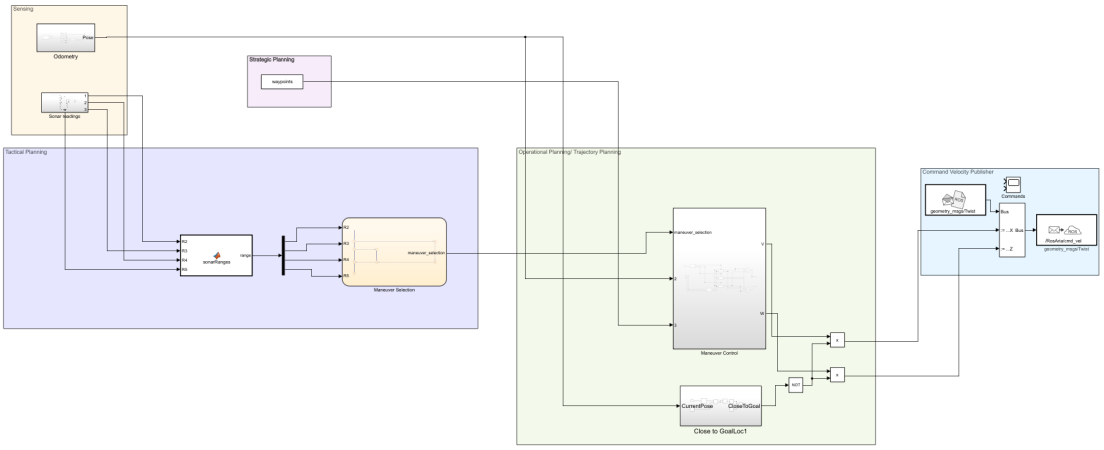


FIGURE 5.9: Model Logic and Control test.

In *the perception or sensing layer*, range and position information are calculated using respectively robot sonars and wheel encoders. The range reading is calculated as described in the previous section. The robot pose is extracted from wheel encoders of the pioneer robot by using a coordinate transformation method. The pose information received from the robot are Quaternion data type that are difficult to deal with. For that reason, they are transformed into Euler angles ZYX representation inside the extracted robot pose sub-block as shown in Figure 5.10. Fortunately, Simulink offer a ready coordinate transformation block in the library browser. Inside this block, the type of inputs coordinates and the desired output types can be specified. This block will take

the signal of the robot's pose after using the bus selector block in order to specify the needed signals inside the bus and then transform into Euler representation, then using MUX block to get the desired pose vector $[xy\theta]^T$.

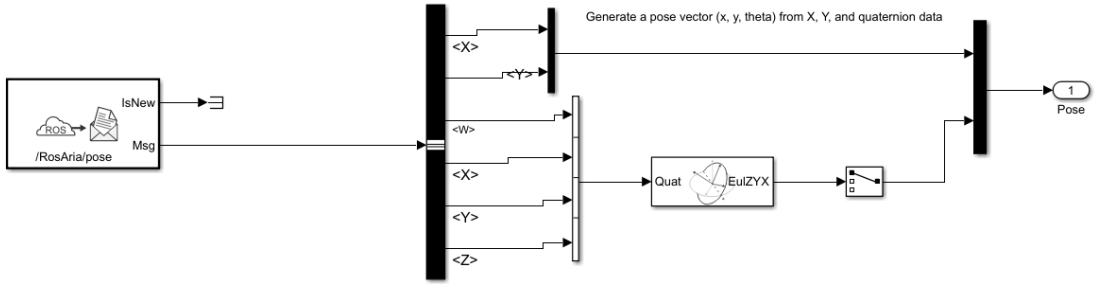


FIGURE 5.10: Coordinate Transformation inside the Extract Robot Pose Block.

In *the tactical planning layer* it is as described is last section only using the If-Else logic and tuning the real parameters into the algorithm as shown in Figure 5.11.

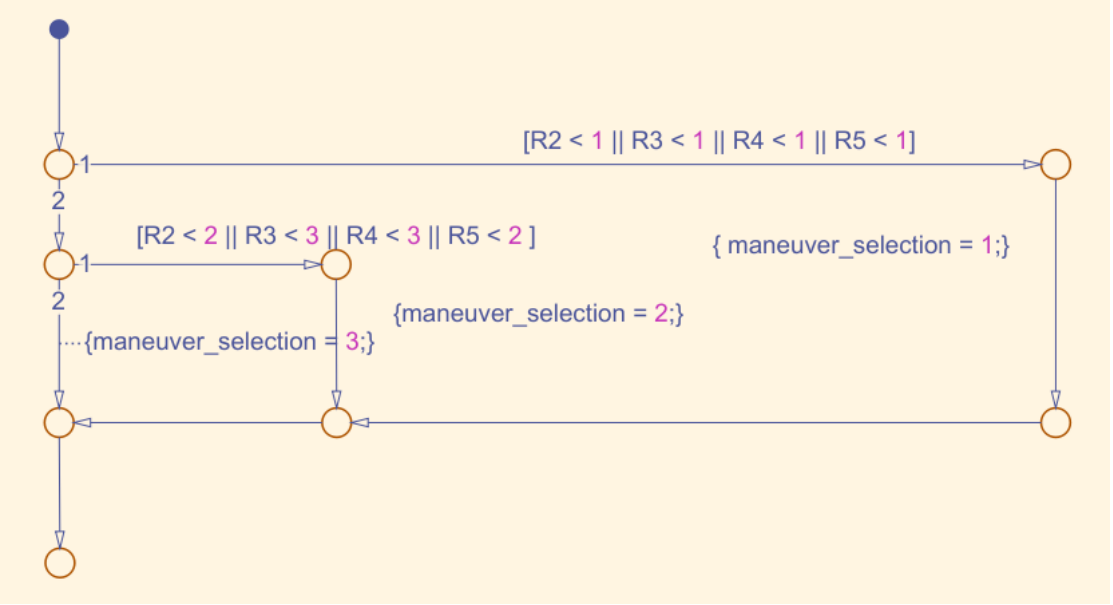


FIGURE 5.11: Parameter tuning of the sensors data.

In *the strategic planning layer* a path is defined for the robot to follow. It is represented as a set of waypoints which were recorded from the Robotics Research Lab main Corridor using ROS environment. It is a vector of X and Y coordinates (Figure 5.12).

0	0
0.0890	0
0.4680	-1.0000e-03
1.1370	-0.0020
1.8870	-0.0030
2.6360	-0.0050
3.3850	-0.0060
4.1280	0.0060
4.8650	0.0500
5.6080	0.1030
6.3450	0.1860
7.0890	0.2700
7.8340	0.3530
8.5780	0.4360
9.3230	0.5190
10.0670	0.6030
10.8040	0.6990
11.5420	0.8110
12.2810	0.9310
13.0210	1.0510
13.6940	1.1600
14.0870	1.2240
14.1910	1.2410

FIGURE 5.12: Waypoints of the Robotics Research Lab main Corridor.

In *the operational planning layer* which is the main layer of this section, the maneuvers selected in the tactical planning layer will be translated into linear and angular velocities as in the *Maneuver Control* block (Figure 5.13).

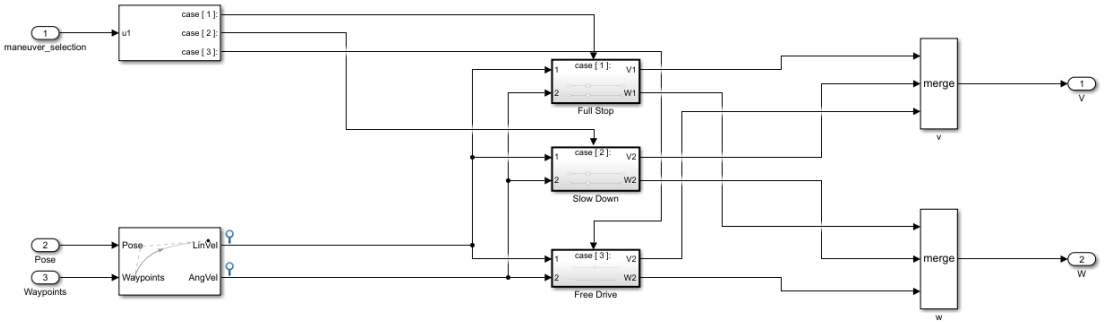


FIGURE 5.13: Maneuver Control Block.

The Switch case block has a single input which is the maneuver selected from the tactical planner. To select a case, the input value is defined using the Case conditions parameter. The cases are evaluated top down starting with the first case. Each case is associated with an output port that is attached to a Switch Case Action Subsystem block which represents the maneuvers in term of linear and angular velocities. When a case is selected, the associated output port sends an action signal to execute the subsystem. These Action Subsystems are connected to the pure pursuit controller which defines

the robot's linear and angular velocities. The controller parameters for this test was set to 0.8 m/s for desired linear velocity, 0.2 rad/s for maximum heading rate and the look ahead distance factor was set to 0.8 m/s. The purpose is to modify the v and ω depending on the maneuver. A set of gain blocks will be applied on the v and ω.

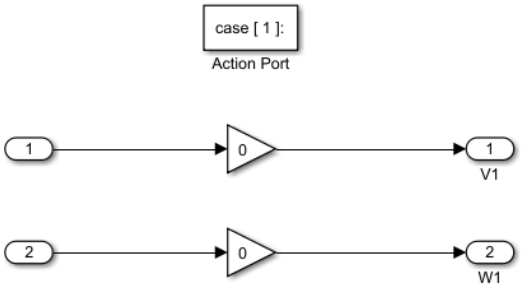


FIGURE 5.14: Full Stop Maneuver Block.

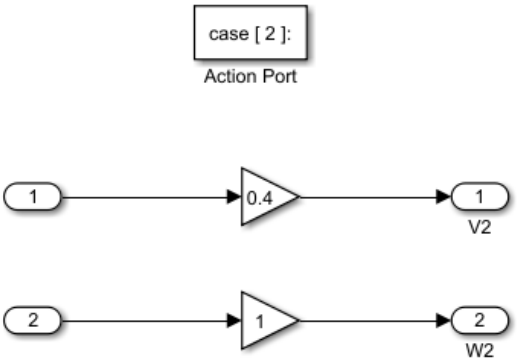


FIGURE 5.15: Slow Down Maneuver Block.

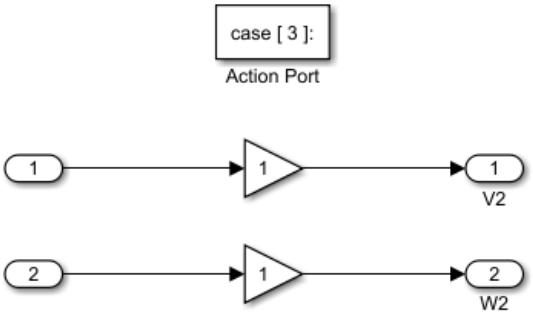


FIGURE 5.16: Free Drive Maneuver Block.

A Merge Block is to select the v and ω which is defined from a specific case to output it.

The speed-throttling block (Figure 5.17) will need the actual pose and goal location in

order to track the remaining distance from arrival. This block will allow the robot to slow down before reaching its destination at predefined range and to stop completely when reaching the target at predefined proximity. The output of this block is a constant gain value that will be multiplied by v and ω for the desired motion control.

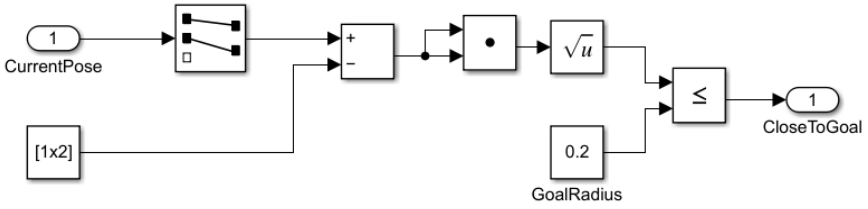


FIGURE 5.17: The speed-throttling function block.

After that, the continuously computed linear velocity and heading rate values will be forwarded to the final block (Command Velocity Publisher) for controlling the P3-DX robot. In order to send velocity commands from Simulink to the robot over ROS. In that block, a bus signal is created and the desired signals are as signed to it as in Figure 5.18. Then, the created bus signal is sent to the target ROS hardware by using a Publisher ROS block.

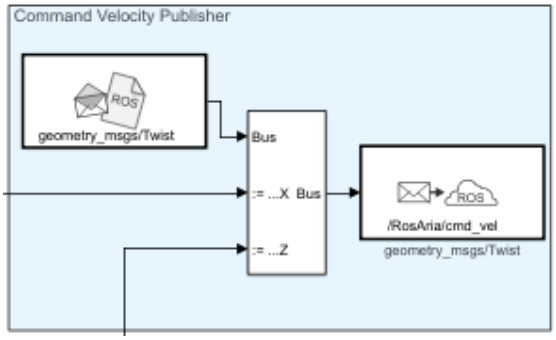


FIGURE 5.18: Command Velocity Publisher block.

Results

As mentioned before, the results of this test are more apparent in a video which will be attached to this thesis. The robot runs at a speed of 0.8 m/s in normal conditions (means with no obstacle detected) once there's a obstacle at front the robot reduces it's speed by 0.4 m/s at a range of 3m and stops once the range between the robot and the obstacle reach 1m.

The responding times of the robot between detecting an obstacle and performing a maneuver is of some few seconds which made the reaction slow in comparison with real life situations. That's why it is crucial to select a rather wide range for sonar readings to give time to the robot to process the data in an optimal way.

Parameter tuning (desired linear and angular speed and value of gain for speed reduction) is very important depending on the environment and the mechanics of the test vehicle.

Conclusion

As a conclusion, the concept model which combines tactical and operational planning is more effective in comparison with the model in Figure 3.11, since it is more flexible to different scenarios because it can manipulate more parameters in an organised way by specifying blocks for same data type: tactical planning for sensors data and operational planning for linear and angular velocity control. On the other model it was impossible to add more than one maneuver to be performed by the robot; it was more like a one maneuver at a time which made it difficult to manipulate the parameters depending on the scenarios.

The fact that 3 maneuvers were performed using this model (full stop, slow down , free drive) explains the accuracy of this model. Later on, the main objectif will be to add more maneuvers to the operational planning and more sensor data from different sensors into the tactical planning block.

NB a list of ROS commands which were used during the test is presented in the appendix. This reference [37] also tackles the ROS framework in detail.

Chapter 6

Conclusion

The task of this thesis was to develop and test a tactical planning model. This resulted in the use of different rule-based solution approach: If-Else method and Finite State Machine method. The following paragraphs give a detailed reading about the contributions of the thesis and the possible future work needed to improve these contributions. In this research two parts were taken into consideration in order to accentuate the general purpose of this thesis. The first part was a theoretical concept study of the important use of rule-based tactical planning methods and develop the first model inspired by the main functional architecture of an AD system. The second part was to make a comparison of each method from the two main approach : rule-based and AI-based approach.

The method suggested in this thesis is rule-based approach since the test environment is simple and predictable. For that, two methods were proposed : starting from a naive approach with is using the If-Else logic with writing lines of codes including all conditions from sensors reading data and the statements which represent the maneuvers selected to each scenario described from the sensor readings; the next method of this context approach is a more sophisticated method and hugely exploited among developers who tackle decision-making problems for automated driving systems, which is the Finite state Machine method which represents a set of states and transitions to select actions. This method allows the use of more input data (sensor data) which makes the system more practical to wider range of situations.

A small introduction into the other main approach for decision-making problems which is the AI-based approach, followed by a comparison of each solution method to give a general perspective to the reader.

In this work, the following tasks were done:

- Study analysis of the use case and extraction of possible scenarios and maneuvers.
- Concept design of the tactical planning model.
- Realization of the concept using different rule-based methods.
- Test and validation of the logic and control models created.

However, most importantly finding the right method to apply for an autonomous driving problem and making it into practice to mark the feasibility of the algorithms used. In addition to that, choosing the right Software framework that can handle the computational calculation of such methods plays a major role in solving such problems. Finally, comes the simulation part using the adequate simulator in order to visualize our results.

To conclude this work, the last point to be brought to light is the future work concerning this project. Therefore, it is crucial to focus on the major ongoing parts in this project which are:

- Adding more sensors data and maneuvers to the model to operate in more complex situations.
- Using more sophisticated control methods for more flexibility (e.g MPC).
- Designing an AI concept for the tactical planning and implementing it later on.

Appendix

Pioneer 3-DX robot is a small mobile robot to develop autonomous solutions. Connect robot with ROS on linux based system using the RosAria library which was developed by the manufacturer. The robot is running on the principle of differential drive system. It has 16 SONAR sensors, 8 in the front and 8 in the back. They all are mounted in such a way that they are getting data from all possible angels to reduce the blind zone for the robot. Installing Linux based environment on the host computer using the VMWare virtual machine. It is a software which allows to use different operating systems on a host computer without dividing the main hard drive. After downloading Linux on a Virtual Machine, installing AriaLibrary which is the package specialised and responsible for all Pioneer 3-DX functionalities on ROS environment.

.1 Connection with ROS

The following steps explain the bridging of the ROS environment with the robot:

1. Turn on the robot and connect it with your host machine using a USB cable.
2. One window will open on the virtual machine interface. Click on it to connect it with the virtual machine.
3. Give permission to that specific USB port: First, check if it is connected to the machine using:

   ls -l /dev— grep ttyUSB (Remember the port number)

   then,

   sudo chmod -R 777 /dev/ttyUSB(port_number)
4. Start ROS master using:

roscore

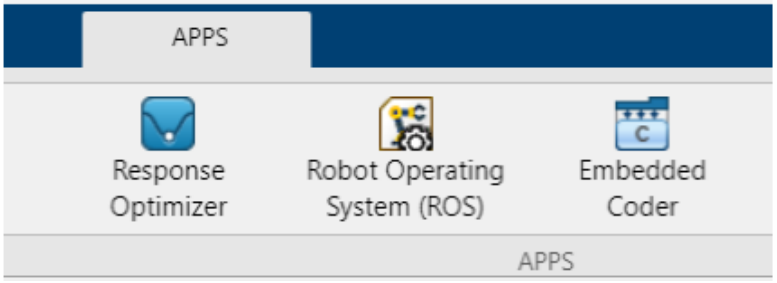
5. Start Aria Client to use the robot:

   roslaunch rosaria_client rosaria_client_launch.launch

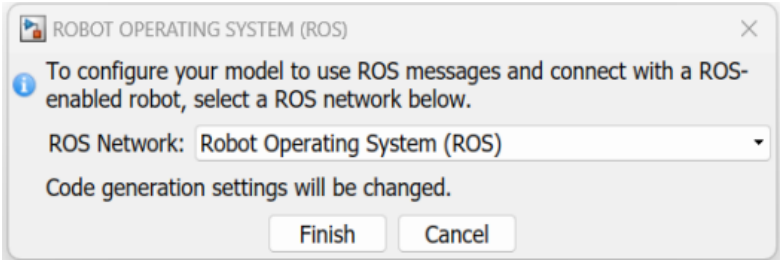
6. You will hear a beep sound from the robot.

.2 Bridge ROS and Simulink

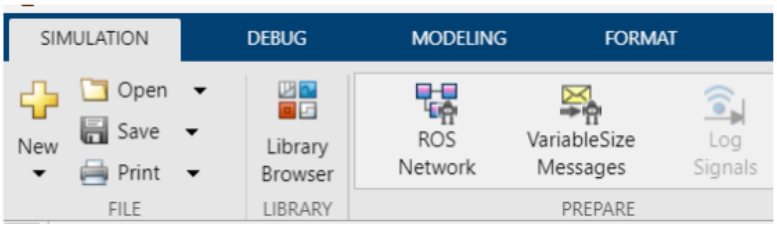
1. Open Simulink model, go to the app section of the top and select 'ROS'.



2. Select 'Robotic Operating System 1 (ROS1)'.



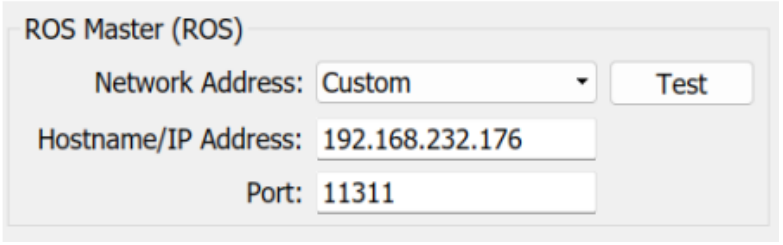
3. Go to the simulation section, select the 'ROS network' and write down the IP address of the ROS master machine.

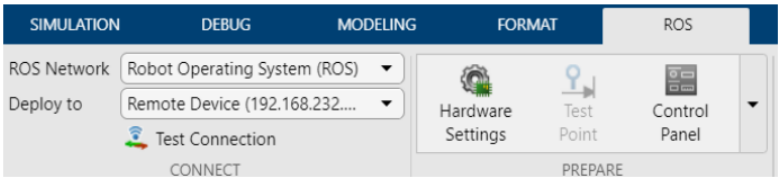


4. Test if it reached master then press OK.

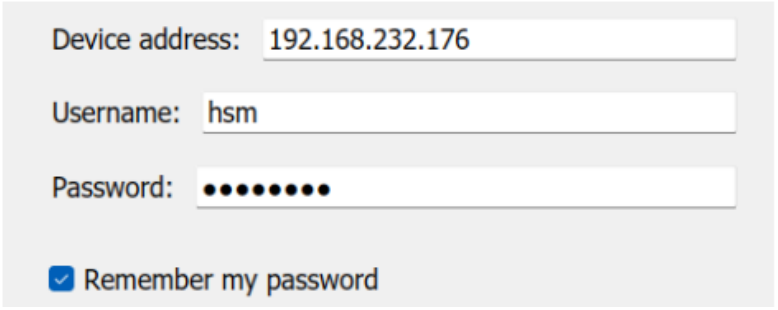
5. Go to the ROS section.

6. Select 'deploy to' then manage remote device.

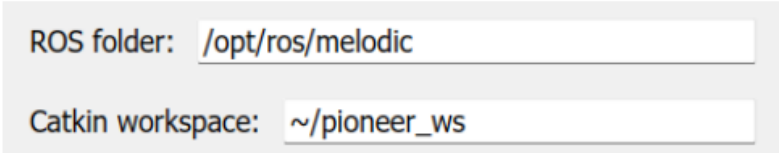




7. Write down the IP address of the ROS master machine, and the master machine's username and password.

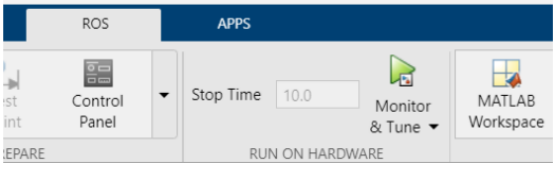


8. Look for the ROS folder section if you are using different ROS versions and catkin workspace if you are using different workspace.



9. Press OK.

10. Now run this model pressing 'Monitor and Tune'. It will create a separate folder



on the specified catkin workspace on step 8 and will start the Simulink model on the ROS interface to test the algorithms and visualize the sensor and pose topics.

These are the main steps that we followed to implement the Simulink model on the robot to test it.

References

[1] Delivers.ai. https://delivers.ai/, October 2022.

[2] Thermal imaging to advance reliability in self-driving cars. https://www.flir.com/news-center/corporate-news/thermal-imaging-to-advance-reliability-in-self-driving-cars/, January 2018.

[3] Laser light paves the way for autonomous driving. https://www.osram.com/os/news-and-events/spotlights/laser-light-paves-the-way-for-autonomous-driving.jsp, October 2021.

[4] 3 under-the-radar driverless car companies. https://www.fool.com/investing/2018/12/09/3-under-the-radar-driverless-car-companies.aspx, December 2018.

[5] Free flir starter thermal dataset for autonomous vehicle testing. https://www.flir.com/news-center/camera-cores--components/flir-open-source-starter-thermal-dataset-for-autonomous-vehicle-testing/, June 2018.

[6] Cheaper infrared cameras for self-driving cars, phones in the offing. https://auto.economictimes.indiatimes.com/news/auto-technology/cheaper-infrared-cameras-for-self-driving-cars-phones-in-the-offing/68340483?redirect=1, March 2019.

[7] Alexander Jungmann Rick Voßwinkel, Maximilian Gerwien and Frank Schrödel. Intelligent decision-making and motion planning for automated vehicles in urban environments. 2021.

[8] Mathworks: Developing autonomous mobile robots using matlab and simulink [online]. https://www.mathworks.com/campaigns/offers/next/autonomous-mobile-robots.html.

[9] Ai approaches compared: Rule-based testing vs. learning. https://www.tricentis.com/learn/artificial-intelligence-software-testing/ai-approaches-rule-based-testing-vs-learning.

[10] Finite state machines. https://brilliant.org/wiki/finite-state-machines/.

[11] Koustava Goswami Dr. Enda Howley, Dr. Patrick Mannion. Decision making for autonomous car driving using deep reinforcement learning(drl). August 2019.

[12] Clarissa Hoffmann. Bachelorarbeit: Einstaz mobiliter roboter für neuartige logistikanwendungen in urbanen gebieten, am beispiel einer automatisierten lebensmittelzustellung in gera-lusan. March 2022.

[13] Developing autonomous mobile robots using matlab and simulink. https://www.mathworks.com/campaigns/offers/next/autonomous-mobile-robots/path-planning.html.

[14] R if else elseif statement. https://www.learnbyexample.org/r-if-else-elseif-statement/.

[15] Create flow charts in stateflow. https://www.mathworks.com/help/stateflow/ug/flow-charts-in-stateflow.html.

[16] Hesham El-Sayed Jalal Khan Sumbal Malik, Manzoor Ahmed Khan and Obaid Ullah. How do autonomous vehicles decide? 2023.

[17] Mathworks, reinforcement learning onramp. https://matlabacademy.mathworks.com/details/reinforcement-learning-onramp/reinforcementlearning.

[18] Part 2: Kinds of rl algorithms. https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html, 2018.

[19] Omron adept mobile robots: Pioneer 3 operations manual. 2017.

[20] Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. A review of mobile robots: Concepts, methods, theoretical framework, and applications. *International Journal of Advanced Robotic Systems*, 16(2):1729881419839596, 2019.

[21] Delivery robot. https://en.wikipedia.org/wiki/Delivery_robot, October 2022.

[22] Dr James Jeffs. Mobile robotics in logistics, warehousing and delivery 2022-2042 agvs, grid-based agcs, amrs, mobile case-picking robots, mobile manipulators, heavy-duty autonomous level-4 trucks, autonomous last mile delivery vans, robots and drones, technologies, markets, and forecasts. https://www.idtechex.com/en/research-report/

mobile-robotics-in-logistics-warehousing-and-delivery-2022-2042/855, January 2022.

[23] Cesareo Contreras. 10 delivery robot companies to watch in 2022. https://www.robotics247.com/article/10_delivery_robot_companies_worth_watching_2022, March 2022.

[24] Lucas G[a] Alcalde and Nathan Rennolds. An autonomous delivery robot can climb stairs, and may be the next big innovation in last-mile delivery. https://www.businessinsider.com/experts-say-an-autonomous-delivery-robot-next-big-thing-2022-2, February 2022.

[25] Last mile (transportation),. https://en.wikipedia.org/wiki/Last_mile_(transportation), June 2021.

[26] Gordon Feller. Autonomous last-mile delivery robots. https://www.assemblymag.com/articles/97531-autonomous-last-mile-delivery-robots, December 2022.

[27] Alfredo Pastor Tella. Autonomous last mile delivery robots cut the highest cost of shipment. https://www.agvnetwork.com/last-mile-delivery-robots, July 2021.

[28] The top sensors used in autonomous delivery robots. https://www.arrow.com/en/research-and-events/articles/the-top-sensors-used-in-autonomous-delivery-robots, June 2022.

[29] The top sensors used in autonomous delivery robots/gps receivers. https://www.arrow.com/en/categories/rf-and-microwave/gps/gps-receivers, June 2022.

[30] Britannica, artificial intelligence. https://www.britannica.com/technology/artificialintelligence.

[31] Stuart J. Russell and Peter Norvig. Artificial intelligence a modern approach. 1995.

[32] intellias, how machine learning algorithms make self-driving cars a reality. https://www.intellias.com/how-machine-learning-algorithms-make-self-driving-cars-a-reality/, 2018.

[33] towards data science, a beginners guide to q-learning. https://towardsdatascience.com/a0beginners-guide-to-q-learning-c3e2a30a653c, 2019.

[34] Analytics vidhya, getting to grips with reinforcement learning via markov decision process. https://www.analyticsvidhya.com/blog/2020/11/reinforcement-learning-markov-decision-process/, 2020.

[35] Montse Cordova. Understanding use cases, use case scenarios, user stories, flow charts. https://www.techtarget.com/searchsoftwarequality/definition/use-case, November 2020.

[36] Hajar EL FAKIR. Bachelor's thesis: Intelligent decision making for automated driving systems. 2021.

[37] Joe Francis. Project work report: Autonomous logistic mobile robot in the framework of matlab/simulink ros. 2022.