Universidade do Minho
Escola de Engenharia

António Manuel Moreira Pereira

Adaptive shared-control of a robotic walker to improve human-robot cooperation in gait biomechanical rehabilitation

january 2021

Adaptive shared-control of a robotic walker to improve human-robot cooperation in gait biomechanical rehabilitation

António Manuel Moreira Pereira

UMINHO | 2021

Universidade do Minho
Escola de Engenharia

António Manuel Moreira Pereira

Adaptive shared-control of a robotic walker to improve human-robot cooperation in gait biomechanical rehabilitation

Master Dissertation
Master Degree in Biomedical Engineering
Medical Electronics

Dissertation supervised by:
Professor doctor Cristina P. Santos

january 2021

Direitos de Autor e Condições de Utilização do Trabalho por Terceiros

Acknowledgments

The worked developed along the past year involved a wide network of people, without whom it would be impossible to achieve the goals set in this dissertation.

I would like to begin by thanking my adviser Prof. Cristina P. Santos, who motivated me to work in the WALKit project, always showing great availability to explore novel ideas. The support, dedication and advice given along the development of this dissertation were invaluable.

I would also like to thank the full WALKit development team, composed by the members João André, João Lopes, Pedro Magalhães, Manuel Palermo, Prof. Paulo Carvalhal and Prof. Cristina Santos, whose advice was deeply important to solving multiple issues at hand. Also, I would like to thank all the members from the BiRD Lab (Biomedical Robotic Devices Laboratory), on which the WALKit team is included, for offering me a great work environment.

Special thanks are due to my friend Manuel Palermo for showing incredible availability in the discussion of multiple themes of this dissertation. His extensive knowledge and understanding of specific subjects functioned as a benchmark for the decisions made along development.

The warmest thank you goes out to my girlfriend and best friend, Marta, for her unconditional support, patience, care, deep affection and love shown in the last year especially. She not only helped me as a girlfriend but also as a work colleague, showing incredible availability for discussing details about the dissertation, and for that I am deeply thankful.

My deepest thanks go to my mother Margarida, my father Adriano and my brother Adriano, whose unconditional support and love shown since the beginning of my life, proved to be the main contributor for all my personal achievements.

Statement of Integrity

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowleighd the Code of Ethical Conduct of the University of Minho.

Resumo

Sessões de reabilitação de pacientes com deficiências na marcha é importante para que a qualidade de vida dos mesmos seja recuperada. Quando auxiliadas por andarilhos robóticos inteligentes as sessões têm mostrado melhorias significativas, face aos resultados obtidos por métodos clássicos. O andarilho WALKit é um dos dispositivos mencionados e permite ser conduzido por parte do paciente enquanto um especialista supervisiona todo o processo de forma a evitar colisões e quedas. Este processo de supervisão é moroso e requer constante presença de um especialista para cada paciente.

Nesta dissertação é proposto um controlador autónomo e inteligente capaz de partilhar a condução do andarilho pelo paciente e pelo supervisor evitando colisões com obstáculos.

Para remover a necessidade constante do médico supervisor, um módulo de condução autónoma foi desenvolvido. O modo autónomo proposto usa um sensor Light Detection and Ranging e o algoritmo de Simultaneous Localization and Mapping (Cartographer) para obter mapas e a localização do andarilho. Seguidamente, os planeadores global e local , A* e Dynamic Window Approach respetivamente, traçam caminhos válidos para o destino, interpretáveis pelo andarilho.

Usando o modo autónomo como especialista e as intenções do paciente, o controlador partilhado usa o algoritmo Proximal Policy Optimization, aprendendo o comportamento pretendido através de um processo de tentiva e erro, maximizando a recompensa recebida através de uma função pré-estabelecida. Uma rede neuronal com camadas convolucionais e lineares é capaz de inferir o risco enfrentado pelo sistema paciente-WALKit e determinar se o modo autónomo deve assumir controlo de forma a neutralizar o risco mencionado.

Globalmente foram detetados erros inferiores a 38 cm no sistema de mapeamento e localização. Quer nos cenários de testagem do controlador autónomo, quer nos do controlador partilhado, nenhuma colisão foi registada garantindo em todas as tentativas a chegada ao destino escolhido.

O modo autónomo, apesar de evitar obstáculos, não foi capaz de alcançar certos destinos não contemplados em ambientes de reabilitação. O modo partilhado mostrou também certas transições bruscas entre modo autónomo e intenção que podem comprometer a segurança do paciente.

É necessário, como trabalho futuro, estabelecer métricas de validação objetivas e testar o controlador com pacientes de forma a corretamente estimar o desempenho.

Palavras-chave: controlo compartilhado, reabilitação, andarilho inteligente, aprendizagem por reforço, navegação autônoma

Abstract

Rehabilitation sessions of patients with gait disabilities is important to restore quality of life. When aided by intelligent robotic walkers the sessions have shown significant improvements when compared to the results obtained by classical methods. The WALKit walker is one of the devices mentioned and allows the patient to drive it while a medical expert supervises the entire process in order to avoid collisions and falls. This supervision process takes time and requires constant presence of a medical expert for each patient.

This dissertation proposes an intelligent controller capable of sharing the walker's drivability by the patient and the supervisor, avoiding collisions with obstacles.

To remove the constant need of a supervisor, an autonomous driving module was developed. The proposed autonomous mode uses a Light Detection and Ranging sensor and the Simultaneous Localization and Mapping (*Cartographer*) algorithm to obtain maps and the location of the walker. Then, the global and local planners, A * and Dynamic Window Approach respectively, draw valid paths to the destination, interpretable by the walker.

Using the autonomous mode as a expert and the patient's intentions, the SC uses the Proximal Policy Optimization algorithm, learning the intended behavior through a trial and error process, maximizing the reward received through a pre-established function. One neural network with convolutional and linear layers is able to infer the risk faced by the patient-WALKit system and determine whether the autonomous mode should take control in order to neutralize the mentioned risk.

Globally, errors smaller than 38 cm were detected in the mapping and localization system. In the testing scenarios of the autonomous controller and in the SC no collisions were recorded guaranteeing the arrival at the chosen destination in all attempts.

The autonomous mode, despite avoiding obstacles, was not able to reach certain destinations not covered in rehabilitation environments. The shared mode has also shown certain sudden transitions between autonomous mode and intention that could compromise patient safety.

It is necessary, as future work, to establish objective validation metrics and testing the controller with patients is necessary in order to correctly estimate performance.

Keywords: shared-controller, rehabilitation, smart walker, reinforcement learning, autonomous navigation

Contents

List of Figures

List of Tables

List of Acronyms

ADS Autonomous Driving System. x, 7–9, 11–13, 18, 25, 53, 87, 90–93

CAD Computer-aided design. 49

DNN Deep Neural Network. xi, 9, 18, 32, 40, 42, 43, 52, 63, 68, 69, 87, 89, 90, 93

DP Dynamic Programming. viii, 37–39

DRL Deep Reinforcement Learning. 9, 10, 32, 71

GD Gait Disorders. 2, 4, 16, 25, 47, 72, 93

GPS Global Positioning System. 14–16

IMU Inertial Measurement Unit. 14, 15, 54, 56

LiDAR Light Detection And Ranging. ix–xii, 8, 14, 15, 27, 49, 53, 55, 56, 59, 60, 67, 68, 74, 83–86, 90, 92, 94

MC Monte Carlo. viii, 37–39

MDP Markov Decision Process. xi, 34, 35, 39

PPO Proximal Policy Optimization. viii, xi, 9, 10, 33, 42–44, 63, 66, 68, 69, 71, 89, 90, 92

RaDAR Radio Detection And Ranging. 14, 15

RL Reinforcement Learning. x, 9, 10, 30, 32, 33, 36, 41, 63, 89, 90, 92, 94

ROS Robot Operative System. 46, 47, 49, 53, 57, 63, 94

SC Shared-controller. vi, x, xii, xiii, 2–11, 25–31, 45, 46, 63–66, 72, 78–80, 82, 84, 87, 88, 90–94

SLAM Simultaneous Localization and Mapping. vii–xi, 8, 16–18, 27, 53, 56, 59, 60, 68, 74, 83–85, 90, 92

SoNAR Sound Navigation and Ranging. 14, 15

TD Temporal-Difference. viii, 37, 39

Chapter 1

Introduction

This dissertation was developed during the last year within the scope of the Integrated Master's in Biomedical Engineering at the Biomedical Robotic Devices Laboratory included in the Center for Micro-Electro-Mechanical Systems (CMEMS), a research center of the Department of Industrial Electronics (DEI) in University of Minho.

The focus of this dissertation was to implement an intelligent artificial supervisor capable of sharing the drivability of the WALKit walker between an autonomous driving algorithm and patient input, within gait rehabilitation scenarios. The Shared-controller (SC) of the walker aims to prevent possible dangerous situations, while also forcing the walker to be compliant to human commands.

1.1 Motivation

Gait Disorders (GD) lead to a poor quality of life due to the lack of mobility capabilities needed to perform daily tasks [8]. Mahlknecht et al. [8] reported a prevalence of 32.2% (95% confidence interval) of impaired gait on a population study of people aged 60-97 years, due to neurological, non-neurological and combined types of disorders (sensory ataxia, Parkinson, etc.), correlating the data with low indexes of physical health, psychological health and social relationships.

Robotic-aided rehabilitation of patients with GD is not a new concept [9], but recent developments in the field show a reduction in the time required and improvements in quality of gait achieved [10, 11], while diversifying the types of devices available - exoskeletons [12], orthosis [13], walkers [14, 15].

Robotic walkers are good options for rehabilitation scenarios due to lowering the gravitational load, allowing the medical expert not having to carry the patient and granting the user with the ability to "drive"

[14–16], creating a safe and practical environment for aided locomotion. These devices also allow an efficient use of medical personnel because one expert can "program" multiple walkers to perform specific actions of rehabilitation while attending to other patients.

The WALKit smart walker [17] (previously named ASBGo*) is a walker like the previously mentioned and it is the one used in the proceedings of this dissertation. When the ideal walker reaches the end of its development it should provide a personalized assistance according to the user needs which are decoded by Artificial Intelligence algorithms from the built-in sensorial data, performing a multitude of tasks for gait rehabilitation, namely: gait quality assessment, fall risk assessment and prevention, autonomous driving, rehabilitation biofeedback and full or semi-autonomous rehabilitation scenario programmer.

Each of these can induce different kinds of behaviors in the walker and these might not be balanced symbiotically with the patient. The following situation tries to explain the possible competition between the multiple subsystems. Let's assume the modules output the following information: the gait quality assessment module realizes that the patient has trouble initiating the gait so the rehabilitation scenario programmer tries to force the walker to move in constant accelerations and full-stops to force the patient to re-experience the initiation-gait-phase; the fall risk assessment and prevention determines that the patient has been unstable and advises the interruption of that rehabilitation scenario; the walker is too far away from the target so the autonomous driving module tries to go as fast as possible towards the destination. In this case, performing the actions advised by each module is not wise due to lowering the performance of all other modules.

The development of a SC that is able to manage the symbiotic relationship between all sub-controllers (single task controllers) and the patient is of high interest to the rehabilitation field. Recent research indicates that the main focus is given to the effectiveness of the robotic-aided rehabilitation with its biomechanical implications [16, 18], however, it is rare when research actually pays attention to the nuances of the proposed autonomy and how to actually deliver it as a functional feature of a walker.

1.2 Problem Statement

When using the WALKit in rehabilitation scenarios, the walker is able to be driven locally by commands issued by the patient and remotely by the medical expert [19]. The patient controls the walker on the majority of occasions, although the medical expert supervisor assumes control when any risk of fall is detected [19].

The fall risk associated to gait rehabilitation scenarios with robotic walkers is evaluated based on 3

main variables: **gait pattern quality**, **walker-patient interface compliance** and **rehabilitation setting complexity**. Moreira et al. [19] state that the prevalence of muscle fatigue, poor centralization/verticalization of the patient's trunk, among others, caused by the inherent GDs, produce low stability, low physical support and poor gait patterns. When commands issued by a remote-controller or autonomous algorithms greatly differ from the commands issued by the patient, the walker behaves in unexpected ways from the patient's point-of-view, lowering the stability of the patient even more [19]. Also, many patients suffer from cognitive impairments, compromising the ability to simultaneously perform the rehabilitation scenario, driving the walker, avoiding obstacles and reaching a certain destination [19]. The overwhelming nature of multi-tasking raises the probability of performing poor driving trajectories (collisions) and non-healthy gait patterns, which in turn cause unwanted falls[19].

The expert monitors the gait pattern quality of the patient and determines how much he needs to interfere. When patients are highly debilitated the interference is more frequent, forcing the patient to re-experience certain poor performance gait phases, while also keeping the walker close to the patient (great physical support)[19, 20]. Also, the supervisor is always aware of the patient's intention, never producing trajectories alienated from the patient's intention, inducing a sense of responsiveness and compliance, avoiding collisions patient-walker. The medical expert also possesses an acute sense of spatial awareness, *i.e.*, he is able to sense the properties of the environment - identifies walls, people, furniture, other patients, etc. -, assume unique characteristics about the objects detected - assume that furniture and walls are static, people are dynamic -, and locating the patient-walker system. This allows the supervisor to guide the patient to a certain destination and not collide with obstacles. All the actions performed by the supervisor diminish the risk faced by the patient, although the existence of multiple patients creates the need for dynamical behaviours from the supervisor that are based on the patient diagnosis which drastically increases the complexity of the risk management process .

Srivastava and Kao [21] state, according to the assist-as-needed principle, that allowing the patient to assume control is essential to achieve better results in rehabilitation, except when the user is putting himself in danger or when is executing rehabilitation behaviours too far from the target. Using the assist-as-needed principle in gait rehabilitation scenarios with a robotic walker forces the supervisor to allow the patient to drive the walker at all times, except when risk of fall is non-negligible [19].

In summary, the medical expert is able to not only evaluate risk and prevent it, but also determine when it is actually beneficial to override the patient's commands. The symbiotic shared control established between the patient, walker and medical expert is necessary to guarantee a safe and efficient gait rehabilitation scenario [19].

As already mentioned in section 1.1, synthesizing an autonomous agent capable of mimicking the decision making process of the medical expert is of high interest to the rehabilitation field. Although, when asking gait rehabilitation experts to describe the decisions made along a rehabilitation process, non-specific answers are given because the supervision done by medical experts is based on a lifetime of experience, so objectivity in the decision making process is hard to obtain.

Recent studies present SCs capable of replacing the supervisor with an autonomous agent which is capable of calculating gait quality and stability metrics [15, 19], sensing obstacles present [5, 6, 15, 20, 22–24], following a pre-defined path to a goal [20, 22, 23], avoiding obstacles [15, 22], while sharing the control with the patient following safety constraints [5, 6, 15, 20, 22–24]. Nevertheless, most approaches use simple heuristics which heavily constrain the behaviour of the SC, prioritizing excessively the supervisor over the patient.

To symbiotically merge the suggestions of each from the autonomous agent and patient, state-of-the-art approaches (chapter 2) suggest the use of a SC, where the system dynamics are perfectly described, each sub-controller is parameterized and hand-tuned (with help of medical experts) for the desired task/environment, low number of sub-controllers are used and the structure of the controller is sub-controller specific. These properties of the SCs pose many disadvantages, namely: not using sophisticated algorithms might not allow optimal behavior to be reached, describing system dynamics might be difficult to achieve and the controller parameter optimization might not be very intuitive [5, 6, 15, 20, 22–24].

1.3 Goals

The SC hypothesized needs to accomplish two tasks: evaluate the risk faced by the system patient-walker and determine how to counteract the risk from simultaneously trying to reach a destination, avoiding obstacles, and merging the patient's intention, like shown in Figure 1.1. Despite being mentioned in section 1.2 that gait quality of the patient impacts the overall risk, this was not considered in this dissertation due to the unavailability of testing with disabled patients.

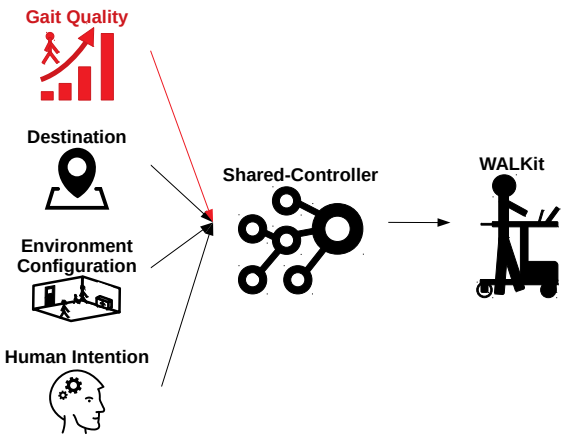


Figure 1.1: The conceptual SC proposed in this dissertation. This SC is responsible for interpreting each source of information available to the medical expert in a rehabilitation scenario - gait quality, destination, environment configuration and human intention - and symbiotically merging the inputs, producing WALKit commands that always guarantee a symbiotic relationship between walker and patient . Gait quality (red), although considered in normal gait rehabilitation scenarios, was not considered in this disseration due to the unavailability of testing with patients.

To build an intelligent SC able to replace or aid the medical expert, certain goals need to be achieved:

- **Goal 1: Present a review of the state-of-the-art approaches** to understand what are the advantages and disadvantages of each method in the literature, so that an informed decision can be made about how to actually deploy an efficient SC.
- **Goal 2: Determine the necessary sensors to effectively create an environment representation**, so that the autonomous agent is able to run risk assessment and prevention algorithms. The walker is used in multitude of medical installations, each with a different configuration, so **the sensing structure deployed cannot be environment-specific and should require no preparation by a technical expert**. To avoid this, the chosen sensors need to be included within the walker, with minimal structural and functional changes.
- **Goal 3: Acquire maps of the environment** so that the autonomous agent can determine within which regions it can travel and how to actually plan efficient paths to reach the destination. The map should not only be accurate enough to be used by an autonomous agent, but **should also be readable by a patient or a doctor**, due to the need to pick a destination for the rehabilitation scenario.
- **Goal 4: Assess the risk faced by the patient-walker system** given the environment's configuration, walker's limitations and inputs from the patient. All restrictions are going to be the

target of extensive examination in the following chapters.

- **Goal 5: Develop an autonomous agent responsible for generating trajectories that always prioritize patient's safety**, avoiding collisions with obstacles and reaching the destination.
- **Goal 6: Deploy a SC capable of symbiotically merging the suggestions of both autonomous driving agent and patient.** The controller needs to meet certain requirements:
  - **Sophisticated Intelligence** - Guaranteeing the possibility of optimal behavior.
  - **Scalable** - Support additional number of sub-controllers with minimal changes, in case other modules in the walker need to interfere with the drivability, like the example described in section 1.1;
  - **Generic** - Not sub-controller specific.
  - **Human-like tunability** - The medical expert should be able to easily understand and tune the controller parameters, parameterization which needs to be close to what actually the expert ponders when performing gait rehabilitation. This tries to remove the ambiguity in the description of optimal supervision by medical staff.
- **Goal 7: Validate the proposed framework within gait rehabilitation scenarios.**

The following **Research Questions (RQ)** are also expected to be answered:

- **RQ 1:** Can the medical expert be replaced by the autonomous agent developed?
- **RQ 2:** Which information is required to assess risk faced by the patient-WALKit system?
- **RQ 3:** Which metrics need to be measured to validate both the Autonomous Driving System (ADS) and SC?

1.4 Proposed Solution

This dissertation proposes a hierarchical control structure, as presented in Figure 1.2. On the base of the hierarchy, multiple sub-controllers are deployed, each responsible for monitoring the risk from one of the sources already mentioned - gait pattern quality, walker-patient interface compliance and rehabilitation setting complexity - and creating plans that diminish each. On the top of the hierarchy the prioritizer

evaluates the recommendations from the base-level commands and determines the relative contribution of each to the actual command supplied to the WALKit. The modularity of the hierarchy proposed allows for each sub-controller to be used separately, without the need for significant structural changes.

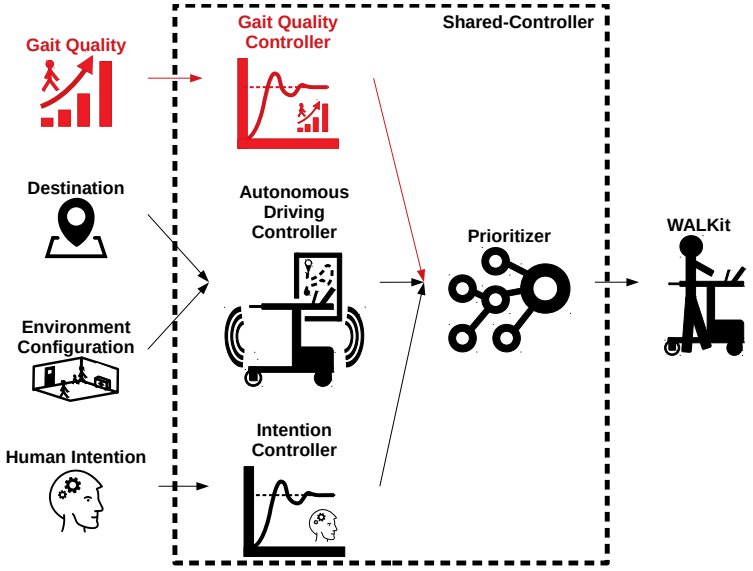


Figure 1.2: Representation of the hierarchy of the SC. Each sub-controller supplies the prioritizer with the commands to perform a specific task. The prioritizer ponders its commands received as inputs and determines the relative contribution of each sub-controller command, given the current circumstances of the rehabilitation scenario. Gait quality (red), although considered in normal gait rehabilitation scenarios, was not considered due to the unavailability of testing with patients.

An autonomous driving agent is developed and used as the expert in avoiding obstacles and reaching a certain destination, serving as a replacement for the supervisor. The intention controller proposed is responsible for expressing the patient intention into valid WALKit commands. If the walker faces a near collision situation the prioritizer resorts to autonomous driving agent and when the risk is neutralized the control falls back to the patient, following the assist-as-needed requirement.

It should be referenced that the implemented ADS is based on a heavily benchmarked approaches to guarantee the efficiency of the autonomous driving mode while performing the task, while the prioritizer component is the main focus of this dissertation.

The autonomous mode developed uses a Simultaneous Localization and Mapping (SLAM) system (Google's Cartographer [25]) that is able to map the environment and localize the walker simultaneously without the need for hardware to be placed in the environment, taking the scan of Light Detection And Ranging (LiDAR) system as input. The *A** [26] global planner is used to create a path from the location of the walker to the chosen destination and the local planner *Dynamic Window Approach* [4] algorithm is

used to translate the spatial path from A^* into velocity commands.

Reinforcement Learning (RL) is a field within Artificial Intelligence that specializes in learning the behaviour needed to perform a certain task based on the capitalization of a pre-established reward function [27]. The SC proposed uses this kind of algorithms, enabling a medical expert to indicate a highly abstracted reward function and allowing the controller to try to match the behaviour inferred, by running multiple scenarios and learning from experimenting with the environment.

Proximal Policy Optimization (PPO) is an algorithm that belongs to the Deep Reinforcement Learning (DRL) field, which carries the advantages of Deep Learning by using non-linear function approximators like Deep Neural Network (DNN) as generic controllers [27] and uses as input the raw mapping of the environment and commands from both autonomous mode and patient intention, being able to perform obstacle detection, collision avoidance and sub-controller prioritization all within one structure, without the need for heavy data pre-processing and creation of a set heuristics and assumptions to simplify the problem.

1.5 Solution Requirements

To prove the possible use of the proposed solution as an efficient rehabilitation tool, the WALKit walker needs to achieve the goals set not only in terms of functionality but also work in a time-efficient manner.

The mapping and localization algorithm should be able to create maps and localize the walker with the lowest error possible. Also, the mapping and localization, global planner, local planner and shared controller should be able to function in at highest frequency possible. Due to the computation power limitations this is not possible, so conservative targets were used to solve the tradeoff between efficiency and optimality. Presenting errors in mapping and localization below 0.5 m were deemed sufficient to track accurately the walker due to the value attributed being negligible when comparing to the size of the walker (1.2x1m approximately). The target for the update rate was chosen to be 10 Hz based on the capacity for the walker to break and guarantee safety. With a maximum velocity and acceleration of $1\ m/s$ and $2\ m/s^2$, the WALKit can transition from full-velocity to full-stop in 0.5 seconds, then, following Nyquist's sampling theorem, the controller should work at least at 4 Hz, however, for establishing a margin of error, we set the goal at 10 Hz.

To validate the ADS the relative frequency of collisions and goals reached should be used metrics, as evidenced in literature [15, 20, 23].

The SC, similar to the ADS, is validated using the relative frequency of collisions and goals reached

in the benchmark scenarios while also measuring the restrictiveness of the paths as described in the state-of-the-art [5, 22].

It was deemed acceptable for the SC and ADS not being able to reach a certain goal, however it is unacceptable that the walker collides with an obstacle, so that patient safety is guaranteed.

1.6 Contributions

The work developed in this dissertation offers an insight to a multitude of technological features used in multiple fields of research. The main contributions are:

- A system capable of creating high-quality maps with localization system allowing for a portable and multi-functional walker, without the need for external setup in plug-and-use fashion;
- An autonomous mode capable of driving the walker without the need for human supervision and able to avoid obstacles;
- A SC able to calibrate the gait rehabilitation session to the patient's needs.

The developed work has lead to one conference paper, one conference abstract and one book chapter, all accepted and awaiting publication.

- **A. Pereira**, J. Lopes, J. Afonso, L. Costa, J. Figueiredo, and C. P. Santos, " Adaptive SC of a robotic walker to improve human-robot cooperation in gait biomechanical rehabilitation." in 9th National Conference of Biomechanics. Porto: Taylor & Francis Group, 2021. (accepted)
- **A. Pereira**, J. Lopes, J. Afonso, L. Costa, J. Figueiredo, and C. P. Santos, " Adaptive SC of a robotic walker to improve human-robot cooperation in gait biomechanical rehabilitation." (Abstract) in 9th National Conference of Biomechanics. Porto, 2021. (accepted)
- João M. Lopes, João André, **António Pereira**, Manuel Palermo, Nuno Ribeiro, João Cerqueira and Cristina P. Santos, "ASBGo: A smart walker for ataxic gait and posture assessment, monitoring, and rehabilitation" In: Gupta D., Sharma M., Chaudhary V., Khanna A. (eds) Robotic Technologies in Biomedical and Healthcare Engineering. CRC Press, Taylor & Francis Group, USA., 2021. (accepted)

1.7 Dissertation Outline

This dissertation contains 9 chapters.

The current Chapter 1 offers a presentation to the motivation of this dissertation, along with a summary and relevant information about this dissertation.

Chapter 2 and chapter 3 contains the a review about the state-of-the-art of ADSs and SCs, respectively. The discussion of advantages and disadvantages is relevant to offer a foundation for discussion of the proposed controller in this dissertation.

Chapter 4 offers some insight on theoretical concepts about: basic concepts of RL, DRL and a full explanation of PPO (the algorithm used in this dissertation).

Chapter 5 explains in detail the functionalities of the WALKit walker, the robotic device target of this dissertation, and presents an overview of the SC proposed.

In Chapter 6 the full implementation of the ADS is described, along with considerations and validation protocols.

In Chapter 7, following the same structure from Chapter 6, the full implementation of the SC is described, along with considerations and validation protocols.

In Chapter 8, the results obtained from the validation protocols are shown along with some particular experiments showing interesting behaviour cases.

Chapter 9 the full discussion of efficiency of the proposed framework and description of the inherent limitations is presented.

Chapter 10 concludes the dissertation with a short summary of the results and discussion, presenting some future improvements to the system.

Chapter 2

Research on Autonomous Control

In this chapter a review of autonomous of ADSs is presented. The following review was built around the search of scientific information available, using the certain keywords as filter, such as: autonomous driving, robotics, smart walkers, slam, global planner, local planner, collision avoidance, path planning and autonomous driving sensors. Only benchmarked ADSs will be used, so this review will only include already validated algorithms which are already implemented in software packages.

2.1 Introduction

The main objective of the ADS is to fully guide the system walker-patient to a certain destination, while avoiding obstacles. The development of this type of system allows for a medical expert/user to plan an adequate destination and departure points for a rehabilitation scenario to be performed, without the need for constant supervision. In a review paper done by Yurtsever et al. [1], it is referenced that the "Core functions of a modular ADS can be summarized as: localization and mapping, perception, assessment, planning and decision making, vehicle control, and human-machine interface.".

Localization and Mapping is the component responsible for recreating an accurate description of the surroundings of the walker to allow for further planning. Perception and Assessment are the blocks that identify obstacles and infer the risk that the autonomous agent is subjected to. The Planning and Decision making components allow for the robot to define, within the risk constraint space, to draw a set of trajectories and translate those into commands that can be interpreted by the hardware (wheel motors and driving software). The pipeline mentioned before, with a schema in Figure 2.1, is widely accepted due to it is modularity. Yurtsever et al. [1] state an advantage of that modularity which consists in allowing for

the study of the problems related to each block of the pipeline to be independent, but it has downsides when information cannot be interpreted by blocks which are not adjacent, *e.g.*, the decision making block cannot directly access the sensor data because it does not know how to interpret it.

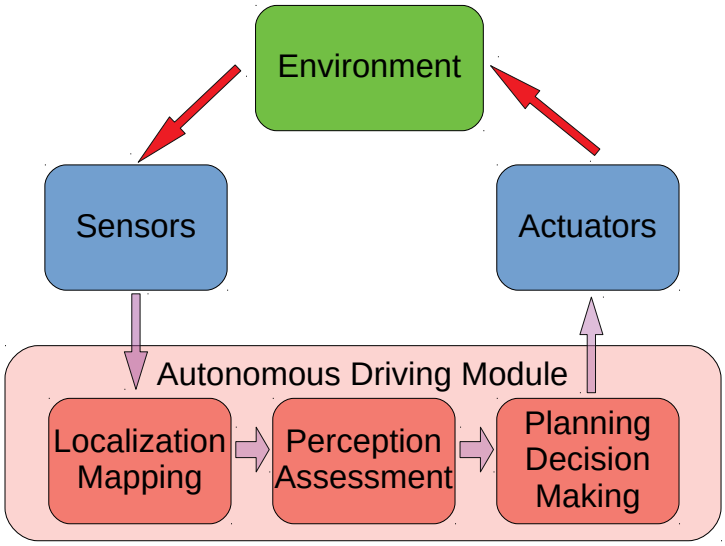


Figure 2.1: Pipeline of a modular ADS [1].

The analysis of each individual module is presented in the following sections. Sensors and actuators are also analyzed as part of the ADS though they are not part of the autonomous driving system *per se*. Instead, they can be considered as limitations or specifications of the system being discussed.

Most of the knowledge from autonomous driving comes from generic robotics and automobile industry, which is a completely different field from the studied in this dissertation but serves as a good basis for the autonomy component.

2.2 Localization and Mapping

To travel to any destination, two components are needed: the mapping of the environment and the current position [1]. Only when a broad answer is known, it can be possible to plan a trajectory to a certain destination. Firstly let us assume that acquiring the mapping of the surroundings and locating a reference point within it are trivial tasks, and let us focus on what type of data can be acquired from the environment.

2.2.1 Sensors

As source of information, typically there are four types of devices: global navigation satellites, range sensors (LiDAR, Radio Detection And Ranging (RaDAR), Sound Navigation and Ranging (SoNAR)), vision

sensors(video cameras) and motion sensors (Inertial Measurement Unit (IMU), encoders) [28]. Each of these is used in specific localization methods, all with advantages and disadvantages.

Range sensors are able to measure the distance of multiple points across space in reference to the sensor. LiDAR, RaDAR and SoNAR systems determine the distance to a certain obstacle by measuring the time difference between the emission of waves and the reception of their reflection [28]. These active sensors emit waves with different properties obtaining a different range of results, each more adequate for different conditions. LiDAR systems use laser and are the widely used due to their high range and angular resolution [28], with an example shown in Figure 2.2. They suffer from not working with same performance on low reflective surfaces and harsh climate conditions (rain, fog). Because rehabilitation scenarios are performed indoors, a lot of the disadvantages considered in automobile autonomous driving are not a factor in this dissertation. RaDAR systems emit radio waves and have lower angular resolution[29], but are commonly used in cars because they are impervious to most climate conditions. SoNAR systems use mechanical waves (sound), typically deployed in short range devices, where high data rate and low cost are mandatory requirements [30].

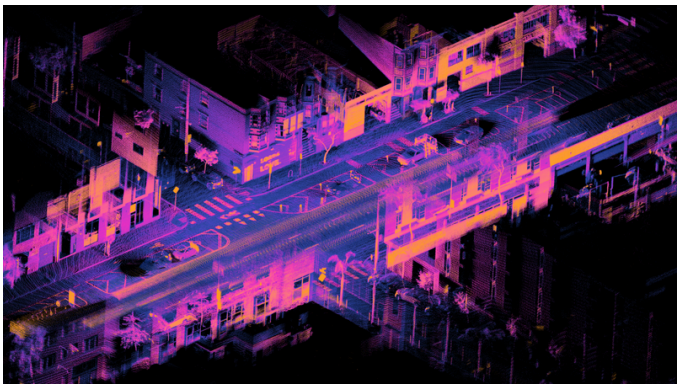


Figure 2.2: A scan of LiDAR system.

Note. By Daniel L. Lu - Own work, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=84598949

The Global Positioning System (GPS) is able to triangulate the position of any receiver around the globe [28] based on the captured electromagnetic signals originated from 3 different source satellites. Low precision and availability in indoor settings are usually the cause for not using this approach in the scenarios related to this dissertation [31]. Mirowski et al. [32] demonstrated a similar approach, in which instead of using GPS, Wi-Fi and the respective routers were used as signals and sources, serving a replacement for the satellites.

Video sensors (video cameras) are a great source of information, because they offer a full colorized

picture of the environment in multiple scenarios, similarly to human vision [28]. When interfaced with computer vision algorithms, the images are subjected to feature detectors which originate high levels of useful metadata beyond the raw data already presented by the physical sensor [28, 33]. They possess the big disadvantage of requiring larger bandwidth to transmit data and needing visible light, which poses a challenge for poorly lit environments.

Motion sensors like IMUs are able to inform the robot of accelerations and changes in heading, through its components like accelerometers and gyroscopes. The double integral of acceleration allows to calculate the position of the robot, but the errors are accumulated since the beginning, degrading the quality of their use as positioning sensors over time [29]. Wheel encoders can count wheel revolutions and calculate robot displacement, but suffer from the same disadvantages as IMUs.

The following Table 2.1 summarizes the main features, advantages and disadvantages of each controller.

Table 2.1: Summary of the sensors used in autonomous driving.

Sensor	Advantages	Disadvantages
LiDAR	+ High angular resolution + High range	- Poor detection of high light absorbent materials - Expensive
RaDAR	+ High range + Not affected by environmental factors	- Low angular resolution
SoNAR	+ Cheap + Fast scans	- Short range
GPS	+ Accurate in exteriors	- Low availability indoors - Low precision indoors
Video Cameras	+ Very rich raw data + Possibility of extracting useful metadata	- Increased data rate - Dependent on good lighting conditions
Motion Sensors	—	- Suffer from error accumulation

2.2.2 SLAM

When a source of environment data is available, mapping and localization is the logical next step. Many types of algorithms can be implemented using global/local "fixed" references like GPS [32, 34] or other similar techniques. Due to autonomous driving being explored more consistently in automobile autonomy, it is acceptable to use global reference approaches because they possess properties like resolution which are compatible with the nature of the problem. However, when talking about GDs and walker aided rehabilitation, this is not acceptable. In the context of this dissertation the localization algorithms need to identify the position in a centimeter scale and use mappings of environments with enough resolution to detect people, hospital beds, etc. Also, it is of high interest to use data from sensors that can be placed in the walker, avoiding connectivity issues to Wi-Fi routers, bluetooth devices or even GPS satellites.

Considering the restriction of sensors being implemented within the robot structure, SLAM algorithms have been developed, because the localization and mapping processes go hand-in-hand, *i.e.*, the location of the walker is determined from the definition of the environment, but the environment's mapping requires the knowledge of the walkers position in order for the sensor data to be correlated [2].

Cadena et al. [2] state that a SLAM algorithm is a *maximum a posteriori* estimation problem where, if X is the set of trajectories performed by the walker and Z is the set of measurements of the environment properties acquired by the used sensors, then Equation 2.1 translates the problem

$$X^{\star} \doteq \underset{X}{\operatorname{argmax}}\, P\left(X|Z\right) \tag{2.1}$$

that falls under the Bayes' theorem. So, to solve a SLAM problem, the algorithm performs a constant optimization of a set of trajectories over time with new data from the sensors [35], like shown in Figure 2.3.

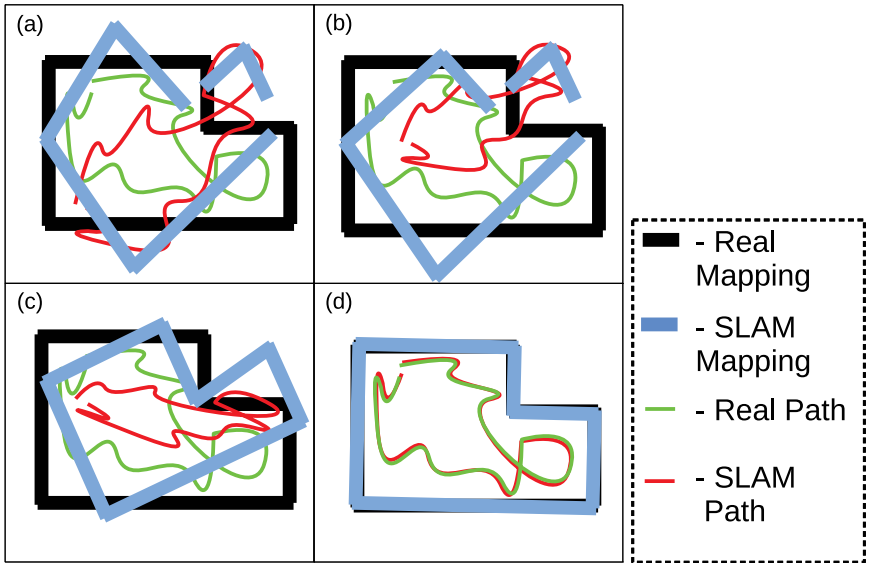


Figure 2.3: Conceptual results of a SLAM algorithm. The mapping of both path and environment mapping optimize from (a) to (b), (c) and (d) sequentially [2].

It is seen in Figure 2.3 that in the beginning of the SLAM process, the mapping and trajectories don't quite make sense, however, through optimization techniques the results get closer to the ground-truth like in Figure 2.3 (d), finally obtaining a good approximation of both mapping and pose. The quality depends highly on the general performance of the algorithm, the quality of the tuning, the properties of the environment, quality of the sensors and computational power.

There are 3 main types of SLAM systems - **grid-based**, **graph-based** and **feature-based**.

Grid-based (also named as particle-filter-based) approaches [36] try to create occupancy grid maps by creating individual maps from both scanner (measurements) and odometry data (pose) and trying to find the best distribution for the whole set of particles (measurement-pose pair). These algorithms have the disadvantage requiring a lot of data and in consequence a **huge amount of memory**, constituting a **poorly scalable** strategy.

Graph-based algorithms, as the ones developed by Sunderhauf and Protzel [37] and Mu et al. [38], work by creating nodes of a graph - which consist of pairs of measurements and poses - and connecting them by either soft or hard mathematical constraints. Comparing to grid-based, graph-based methods are more **sample efficient** because they learn the relationships between the samples acquired from odometry and scanners/cameras/other sources, instead of saving every pair and attributing an importance factor to each, avoiding large memory requirements.

When unique characteristics can be detected in an environment, the use of feature-based approaches increases the efficiency of SLAM algorithms, *e.g.*, when mapping an outdoor environment, trees can

be used to serve as a relevant landmark to improve the quality of mapping and trajectories[39, 40]. These algorithms need feature detection algorithms, which sometimes can be "tricky" to implement due to different detection algorithms producing different results. The extensive research of DNN allied with video camera feed, allowed for the improvement of feature detection algorithms on computer vision which opens doors for the field of Visual SLAM [39]. With this it is possible to use, for example, a mainly static object like a lamp post to be identified as such and used as a strong reference point [39].

2.3 Perception and Assessment

Yurtsever et al. [1] state that an automobile "ADS should constantly evaluate the overall risk level of the situation and predict the intentions of surrounding human drivers and pedestrians". For generic applications, the Perception and Assessment enables the ADS to extract meta-data from the environment so that safe and efficient trajectories can be planned.

Yurtsever et al. [3] try to solve the problem of risk estimation of the driving patterns of surrounding humand drivers. Performing segmentation of individual video frames captured by a monocular camera with DNNs, as seen in Figure 2.4, proved to be capable of inferring the risk created by external factors.

Figure 2.4: Two lane change samples and classification outputs of the system proposed by Yurtsever et al. [3].

The Perception and Assessment component is an important part of all ADSs, however, in rehabilitation scenarios it is always assumed that external factors do not actively try to induce disturbances in the safety of the patient [19], *e.g.*, people walking close to the walker do not actively try to collide with it. For this reason this portion of the ADS was not explored.

2.4 Planning and Decision Making

After obtaining a map of the environment and the current position of the robot, a plan needs to be established to reach the destination. Yurtsever et al. [1] state that "Planning can be divided into two sub-tasks: global route planning and local path planning". Each component is going to be analyzed in the following sections.

2.4.1 Global Planner

The goal of the global planner is to find the best route from the start position to the destination, taking into consideration the mapping of the environment, only considering spatial constraints [1].

The state-of-the-art contains multiple kinds of approaches for the Global Planner although only Dijkstra's[41] and A* [26] algorithms were considered. The reason for only analyzing the two algorithms mentioned is going to be analyzed in chapter 5. Dijkstra's [41] and A* [26] are solvers of the shortest path problem between 2 nodes in a graph.

2.4.1.1 Dijkstra's

When using Dijkstra's algorithm [41], the objective is to choose the sequence of vertices that minimizes the cumulative distance between start and destination of a graph. To determine the optimal path, the steps from Figure 2.5 are performed.

Dijkstra's algorithm

- Input: Graph G, source node s
- N is the set of neighboring vertices of each vertex $v \in V$ which corresponds to the vertex set of graph G. L corresponds to the length between each to vertices
- Create set of visited vertices Q initially equal to V
- for each $v \in V$:
    - $d(v) = \infty, p(v) =$ undefined where d is the accumulated distance and p is the ordered set of vertices from vertex s of the shortest path.
- $d(s) = 0$
- while Q is not an empty set:
    - $c \leftarrow$ Element in Q with minimum $d(v)$
    - Exclude c from Q
    - for each $v \in N(c)$:
        - $d_{temp}(c) = d(c) + L(c, v)$
        - if $d_{temp}(c) < d(v)$:
            - $d(v) = d_{temp}(c)$
            - $p(v) = c$
- return d, p

Figure 2.5: Dijkstra's algorithm [41].

Each node in a graph is connected to others with a certain length. The original Dijkstra algorithm calculates every possible path, starting from the initial vertex, and searching all neighboring vertices opting for the one with the shortest path. At every step, the next vertex to be analyzed is the one which accumulated distance from the initial position is the smallest. This guarantees that the more promising paths (shortest accumulated distanced) are first searched, this way establishing a priority queue.

In Figure 2.6, an example solving a simple graph is shown. Even though there are multiple possible ways of getting to node 3, going directly to 3 from the start is better, even though from the starter node perspective it is apparently shorter to go to node 2 first, due to the shorter distance, as evidenced by the shortest distance after the solving of the full graph in Figure 2.6e.

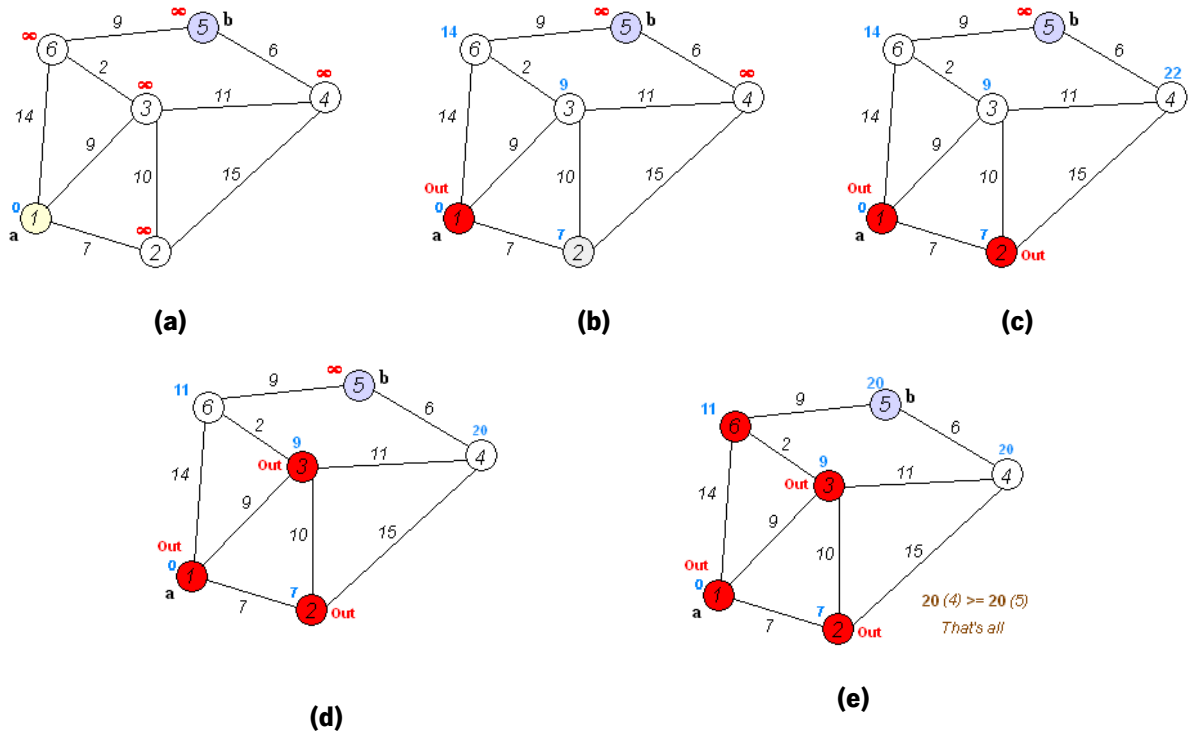


Figure 2.6: Dijkstra's algorithm to find the shortest path between a and b, sequentially from 2.6a to 2.6e Note. By Ibmua - Work by uploader., Public Domain, https://commons.wikimedia.org/w/index.php?curid=6282617 (edited).

For graphs with bigger complexity - bigger vertex space and increased amount of connections - this algorithm does not perform well [26]. The prioritization of the new vertices searched is based on the accumulated distance, raising the probability of exploring local minimums with the disregard for long-term effects [26]. The priority in searching new vertices is simple and guaranteed to converge but it is inefficient, because it lies on the assumption that the current accumulated length is a good indicator of path optimality which is not a good assumption due to the local minimums already mentioned [26].

2.4.1.2 A*

The A* (A star) algorithm is a generalization of the Dijkstra's algorithm and it tries to solve the scalability issue. In most contexts, when trying to find the optimal path between two nodes, certain heuristics can be used to prioritize the search [26]. If we consider traveling from New York to Los Angeles, it can be noticed that the euclidean distance from New York to Los Angeles (3,936 km) is higher when comparing to the distance from Las Vegas to Los Angeles (360 km) so probably going to Las Vegas helps to get to the destination. Using euclidean distance as a heuristic that suggests the optimality of the path being drawn, can hugely improve the quality and sample efficiency of the algorithm [26]. The use of heuristics

allows, in the context of traveling from New York to Los Angeles, not having to search if it is possible to pass through cities in Europe, Asia, South America, because those distances are significantly bigger than travelling through any city in the United States of America.

The algorithm finds the best path by establishing a priority queue like in Dijkstra's, but it calculates the accumulated length $d_{temp}(c)$ using the expression

$$d_{temp}(c) = d(c) + L(c, v) + h(c) \tag{2.2}$$

and h is the heuristic cost estimation [26]. A* is a generalization of Dijkstra's because if $h(v) = 0$ for all v, then $f(v)$ is the simple cumulative length function described in section 2.4.1.1.

In Figure 2.7, a comparison between Dijkstra's and A* is presented, and it shows that A* is more efficient because it searches for the most promising nodes according to the sum of the heuristic and the pure accumulated length.

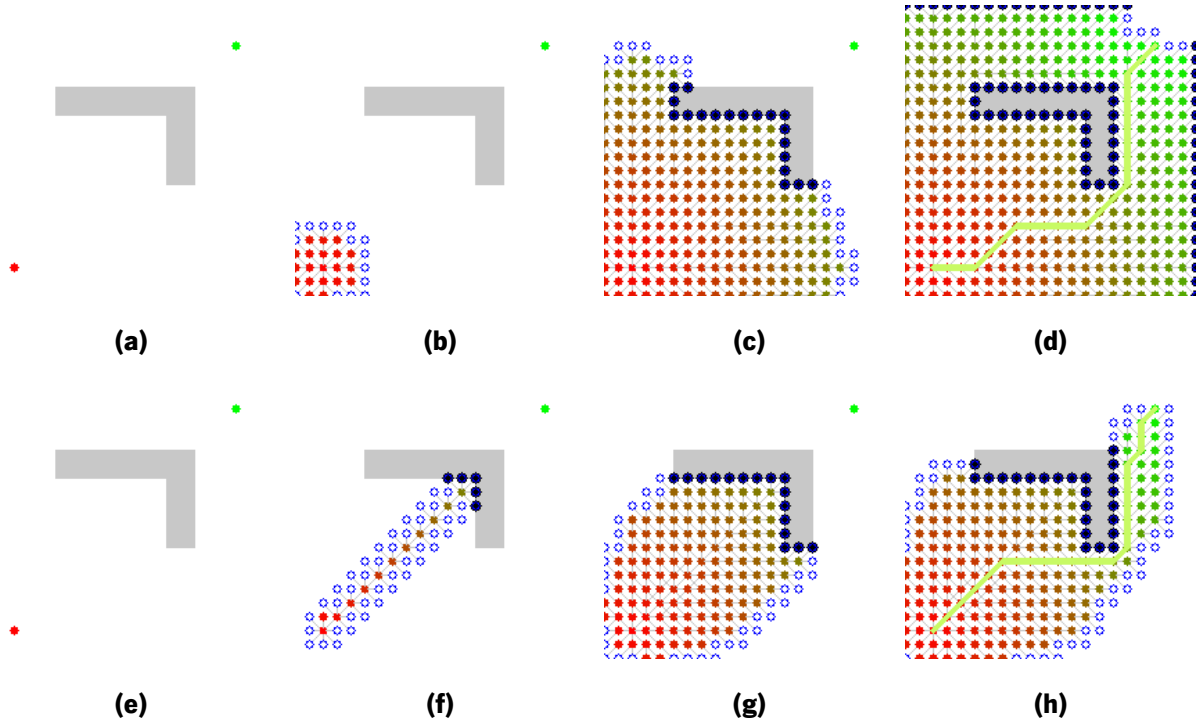


Figure 2.7: Comparison between Dijkstra's (2.7a - 2.7d) and A* (2.7e - 2.7h) algorithms.

Note. By Subh83 - Own work, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=14916867 (adapted).

2.4.2 Local Planner

The local planner is an algorithm that uses as input the path from the global planner, and translates it to velocity commands used by the walker. Trajectory Rollout [4] and Dynamic Window Approach [42] were the two considered local planners for reasons which are going to be discussed also in chapter 5.

These algorithms ponder the outcomes of performing certain movements within a constrained environment. Global Planners, as mentioned in section 2.4.1, solve a spatial problem where no information *w.r.t.* system dynamics is available. The driving patterns of different robots can be quite different because each holds different specifications, *e.g.*, one robot can have bigger acceleration limits and lower maximum velocities in comparison to others. Spatially, the path followed by different robots might be the same, however, when considering the time domain these paths might differ.

Before planning any velocity commands these algorithms need some constraints to be defined - shape of the robot , maximum and minimum velocities and accelerations. Then, a map of the surroundings of the robot (usually a sub map of the map of the global planner only including the immediate surroundings) is given as input [4, 42].

When every input is ready, the algorithms run as shown in Figure 2.8 with a specific example shown in Figure 2.9.

Trajectory Rollout

Sample a number N of velocities to perform in the following step within the space, velocity and acceleration constraints;

Simulate/predict the configuration(position and velocity) of all sampled velocities (Figure 2.9);

Evaluate the quality of the movement of each sampled velocity resorting to a predefined score function;

Choose the best evaluated trajectory;

Perform the chosen trajectory;

Repeat the process.

Figure 2.8: Trajectory Rollout algorithm [4].

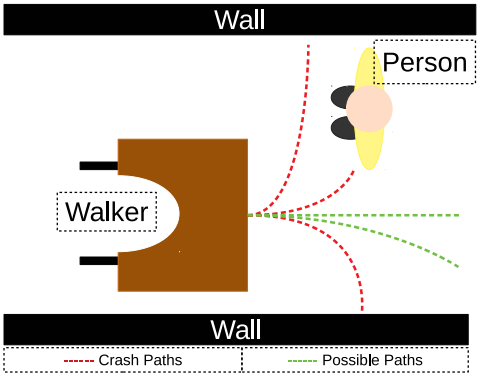


Figure 2.9: Schematic of the simulated trajectories.

As mentioned in Figure 2.9, the trajectories are evaluated by a parameterized score function, that indicates what the walker should or should not do. The score is influenced by three variables: proximity to the goal, proximity to obstacles and proximity to the predetermined global path. The last can be tuned by weighing each component, and adapting the score function to each desired behavior and setting.

Various algorithms try to first create paths that can be followed, and then evaluate them with no regard of the robot's limitations [4]. If the robot has a certain velocity and does not have enough acceleration to reduce its velocity and avoid a collision with an obstacle in front of it, the local planner should not even consider a trajectory that stops in front of the obstacle. Instead of just considering spatial constraints, like presence of obstacles, robot constraints are also considered at every time step because they efficiently decrease the amount of possible outcomes that need to be calculated - this is called a dynamic window [4].

The Trajectory Rollout samples velocities from the achievable velocity space thought a sequence of accelerations, while the Dynamic Window Approach only considers one constant acceleration [4]. The Dynamic Window Approach is computationally efficient, but might be less effective in certain scenarios, generating trajectories that take uncessary paths wasting more time than necessary [42].

Chapter 3

Research on Shared Control

After understanding how an ADS works, it is necessary to understand how a SC framework should function in smart-walkers. This section will describe innovations from a scientific point-of-view, since all the innovation and contributions of this dissertation is related to the SC.

The SC is responsible for sharing the control between patient and supervisor allowing the system walker-patient to not collide with walls and reach the destination, as shown in Figure 1.2. Smart walkers can be used for a multitude of GDs, so the requirements, specifications and limitations of each walker are correlated to the nature of the disability incurred by the patient.

The overwhelming majority of research on the field splits the problem of building a fully autonomous walker into 3 sections: patient intention detection, autonomous supervisor and prioritizer [6, 15, 20, 22–24], similar to block diagram of Figure 3.1. Similar to ADSs, splitting the controller into multiple layers turns the system modular, allowing each module to function independently.

In this chapter, a review of each module is presented, with also a section dedicated to the explanation of how the SCs were validated.

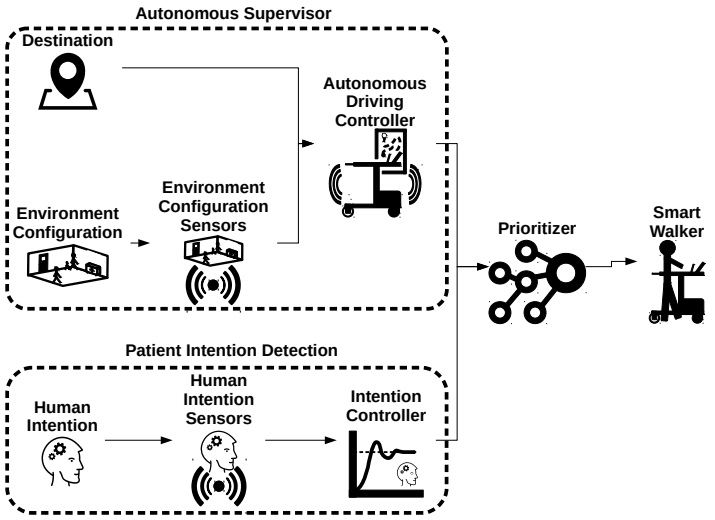


Figure 3.1: Block diagram of the common structure of SCs in state-of-the-art.

3.1 Patient Intention Detection

The patient intention detection module is responsible for, as the name suggest, detecting the intention of the patient. The transmission of intention from the patient to the walker must not overwhelm the user cognitively, so that the patient can always drive the walker without losing focus from other tasks - avoiding obstacles while performing the gait rehabilitation patterns advised by the supervisor. Also, the sensing structure must not compromise the stability of the user [19, 20].

Morris et al. [20] detected the intention of the patient by reading the force exerted in force sensors implemented in two horizontal bars of the improvised walker built on top of a *Nomad XR4000* mobile robot platform. The handles can be displaced forward and backwards, so that the user can push and pull both handlebars equally producing forwards and backwards intention commands, respectively [20]. If the handles are differentially displaced, the commands produced indicate a rotation intention [20]. This approach avoids the use of joysticks and buttons, which require significant hand movements, which in turn compromise physical support and stability of the patient [20].

Jiménez et al. [6], Sierra et al. [15], Spenko et al. [22], Graf and Schraft [23] apply similar structures to the one suggested by Morris et al. [20] in the walkers *AGoRA*, *UFES*, *PAMM* and *Care-O-bot II*, respectively, however the handles do not move.

Huang et al. [5] improve on the intention detector proposed by Morris et al. [20], processing the forces and torques registered by force sensors with a "dynamic model (...) comprised of a set of coupled nonlinear differential equations derived from the Lagrangian and the Lagrange multipliers"[5] of the walker *COOL Aide*, in a way of not only registering the immediate intention but also predicting the actual long-term

intention of patient. This dynamical model is not only capable of detecting the long-term intention but also differentiates inputs from actually intended commands from commands produced when the patient is unstable and grabs the handles.

3.2 Autonomous Supervisor

The autonomous supervisor module is the entity responsible for always diminishing risks of falls, always keeping the patient safe. Some of the controllers referenced in this section do not try to meet the same objectives as the ones set for this dissertation, *e.g.*, do not try to reach a destination or try to follow a path.

Morris et al. [20] implement in the *Nomad* walker an ADS similar to the methods presented in chapter 2. A SLAM algorithm produces a grid map, with the obstacles registered through the use of a LiDAR sensor, followed by planning using global and local planners, included in the robot native navigation modules [20]. Most approaches suggest the use of local planners like Morris et al. [20], similar to the local planners mentioned in chapter 2 (*Dynamic Window Approach* and *Trajectory Rollout*), although it is not specified which are used and how they behave [6, 20, 22]. Sierra et al. [15] mention using the *Dynamic Window Approach* algorithm as local planner, however the global planner is not specified.

The autonomous component of the SC proposed by Huang et al. [5] does not consider a destination, focusing only in obstacle avoidance. The *COOL Aide* is a mostly passive robot, only being able to control the steering angle [5]. This way, it only uses a local planner built on the Virtual Force Field framework [43], imposing virtual moments on the *COOL Aide* walker to avoid obstacles registered by a infrared sensor, scaling the moment with the relative velocity [5]. McLachlan et al. [24] uses the same approach as Huang et al. [5] and improve it by including the global path information in the Virtual Force Field, taking into account the information of the destination.

3.3 Prioritizer

The prioritizer uses the commands supplied by the Patient Intention Detection and Autonomous Supervisor, choosing which the most adequate based on the risk faced by the system.

Morris et al. [20] implemented a simple thresholding technique. The angle between the autonomous mode trajectory and user's intention trajectory is measured and if it is bigger than a certain predefined value, the walker becomes autonomous. Despite being a functional technique it is not optimal, because the decision making process does not take into account the presence of obstacles directly. It is fair to

say that the global planners used in this approach take into account the presence of obstacles, so if the SC uses as input the outputs of the global planner, then it indirectly considers the obstacle component. This might lead to results where the SC always forces the walker to follow closely the path planned, for fear of colliding with obstacles, which is not necessary. It also does not allow for a diversity of behaviours performed, specifically it only allows 2 behaviours to be "coded in" - below threshold and above threshold behaviours - and transitions between modes with high accelerations, possibly creating instabilities in the patient.

Graf and Schraft [23] created a full local planner with SC embedded within the same system. The structure proposed can use 1 of 4 behaviours, namely: direct control - patient commands are directly fed into the walker control -, re-plan path - find other path due to obstacle presence -, docking - full stop of the walker - and wall following - moves the walker without colliding parallel to obstacles. To prioritize each behaviour, the walker uses a angle thresholding technique similar to Morris et al. [20], but instead of using the described angle between autonomous and user intention paths, it uses the angles between the orientation of the walker and the obstacle tangent. Using the direct obstacle information obtained from the sensors, unlike Morris et al. [20], enables the proposed controller by Graf and Schraft [23] to directly determine when crashes are likely to happen and interfere when needed, showing a improved variability in the paths registered and increased relative control by the patient.

In the framework proposed by Morris et al. [20], only the current angle described by intention and autonomous paths is used, meaning that if, for any reason, one of the path's orientation changes drastically the walker will always prioritize the autonomous mode because only the immediate orientation is considered. Sierra et al. [15] instead of executing a simple thresholding with the immediate orientations, created a window considering also the "future" orientations of the ideal path (Figure 3.2).

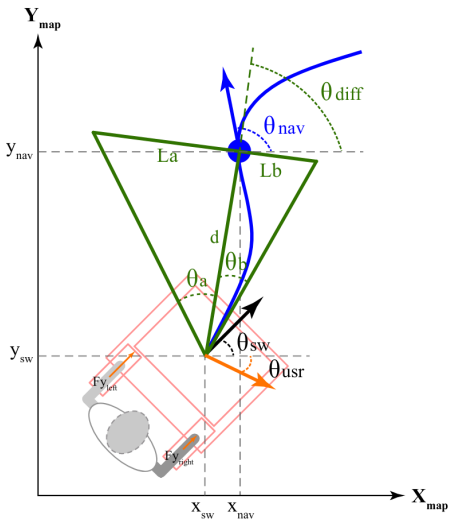


Figure 3.2: Triangular window for admissibility of following user's intention [15].

If the direction of the intention of the user is within green triangle window from Figure 3.2, the user intention is a valid one and it assumes control. In the opposite case the autonomous component assumes control. The size of the window is pre-parameterized, turning the walker more trustful of the patient commands by increasing the size of window, and vice-versa [15]. The walker uses a discretized threshold, inheriting the disadvantages from the mentioned approach, but is able to plan on the long-term, becoming complacent to the user's instructions.

Jiménez et al. [6] suggest a solution without the previous discrete (thresholding) approaches and implement a gradual transition from autonomous to intention control. The attraction and path velocity vectors are defined as the velocity vectors needed to get closer and to follow the path, respectively, together composing the "desired" autonomous velocity vector. The angle between the "desired" autonomous and user intention velocity vectors is measured and used to determine the damping parameter of the impedance controller used as SC [6]. Two curves are created to determine how the damping parameters for both linear and angular velocity vary according to the angle registered. When angle values are close to 0 the damping parameters are at a minimum which allows for the patient to drive the walker. When angle values increase, the damping parameters start rising toward the maximum value, turning the walker gradually autonomous. The curves can be dilated or constricted and maximum values can be increased so that the walker behaviour changes [6], turning the walker more or less trustful of the patient's commands. Opposite to the state-of-the-art approaches mentioned before, the present one allows for the removal of the abrupt change behaviour making the walker more predictable and comfortable [6]. This controller, however, does not avoid obstacle, instead, it gradually decreases both linear and angular according to the measured distance to obstacles [6]. The hyper-parameterization of the impedance controller requires

some understanding of the underlying functionality, which turns this approach hard to understand by a medical point-of-view.

Using fuzzy logic, McLachlan et al. [24] provide a smooth transition between autonomous and intention modes. Five discretized commands of driving (Large Negative, Small Negative, Zero, Small Positive and Large Positive) for both linear and angular velocities are available and consequently subjected to the fuzzy rules established. The rules can be changed from patient to patient allowing for optimal behaviour [24], evidencing patient adaptability. Finally the results are applied to a membership function, like in Figure 3.3, obtaining the actual applied velocity commands. McLachlan et al. [24] state "The premise behind fuzzy logic is that precise outputs can be obtained from imprecise or vague inputs", which represents a relevant advantage of this method over the already mentioned. However, this controller also infers the position of obstacles through the global path, inheriting the disadvantage of needing to stay close to the path, compromising the patient driving freedom.

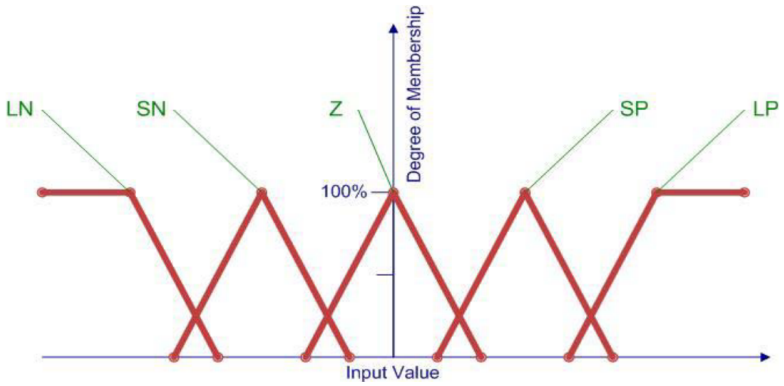


Figure 3.3: Triangle Fuzzy Logic Membership Function [24].

Huang et al. [5] use a similar impedance controller to the one from Jiménez et al. [6], but tries to implement the missing obstacle avoidance feature. It does so by acquiring a map and classifying a map as obstacle and non-obstacle while also measuring distance and relative velocity to each. When distances to obstacles are small and relative velocities of obstacles are high, the impedance controller parameters turn the walker more autonomous, producing virtual moments which deviate the walker from a collision trajectory to a safe one [5]. Despite tackling the obstacle avoidance problem it does not offer any method to perform any kind of autonomous driving converging to a goal.

Spenko et al. [22] merge both concepts from Huang et al. [5] and Jiménez et al. [6], creating a impedance controller that implements obstacle avoidance and destination convergence. Spenko et al. [22] propose a metric which consists of a quadratic function that uses as input the differences between actual and ideal positions, velocities and accelerations and patient stability. The introduction of a stability

criteria read from the patient is a significant advantage of the proposed SC that guarantees directly the safety of the patient, however, following the goals already mentioned in section 1.3, the gait quality metrics are not taken into consideration of performance. To merge the commands from the global planner and patient, the controller tunes the two parameters $K_{computer}$, K_{human} both constrained by the following equation

$$K_{computer} + K_{human} = 1 \tag{3.1}$$

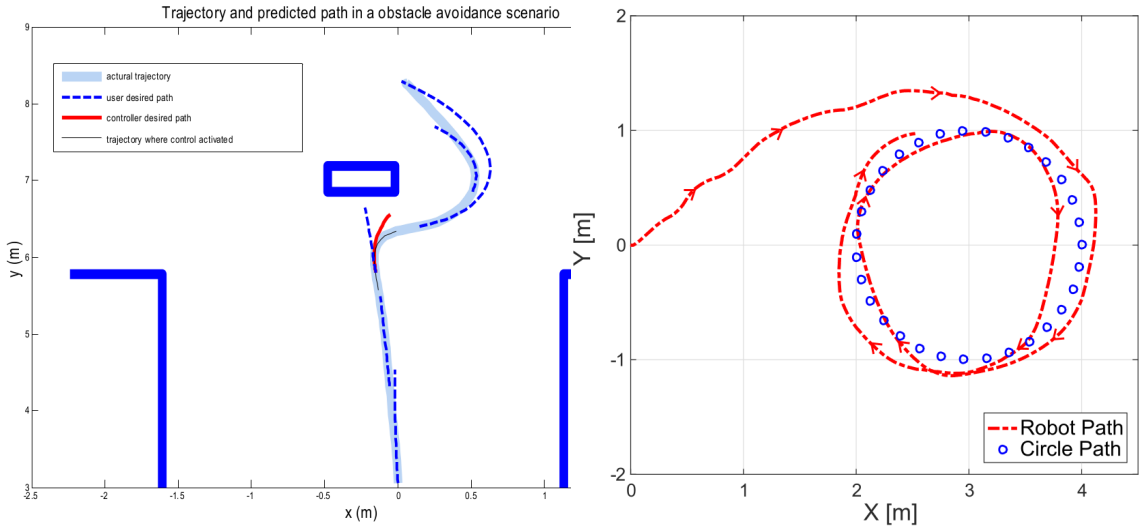
. It is not clear if Spenko et al. [22] pre-established functions describing the correlation between the performance metric and control parameters or if optimization algorithms were used to find the combination which produced the best metrics. If optimization algorithms are used, the controller falls under the policy-based RL category, which is going to be explained in following chapters. Using RL approaches allows the controller to be learned from experimentation, acquiring a human-like tunability (one goal of this dissertation), advantage which is going to be fully described in the following chapter 5.

3.4 Validation Metrics

To validate the SC, 3 main methods are used: case-by-case analysis, qualitative metrics and quantitative metrics.

Case-by-case analysis is used when specific rehabilitation scenarios are performed and the driving paths are compared. Huang et al. [5] tests 4 scenarios by comparing the paths of the intended and autonomous controllers with the actual walker *COOL Aide* path. The scenarios described try to evaluate how the SC behaves when: the intention commands initially try to cause collision, although, after some time, the user complies with the autonomous recommendations (shown in Figure 3.4 (a)), the user commands consistently try to cause a collision, the commands of intention and autonomous mode are in agreement with a low velocity, and the walker and user passing through a narrow door [5]. In all scenarios where the user commands pointed to collision, the SC was able to prioritize the autonomous controller, and after the obstacle was circumvented the walker returned the control to the user. Jiménez et al. [6] established also 2 test scenarios where it is studied how the SC behaves when: the reference path is composed by straight paths and the walker starts on top to the path, and the walker is not on top of a circular reference path (Figure 3.4 (b)). Jiménez et al. [6] proved that the walker is always able to follow the path closely while allowing smooth trajectories to be performed aligning with the intention of the patient. Many other studies evaluate the system by performing scenarios similar to the ones mentioned before [15, 20, 22–24], proving that each offers a certain level of freedom for the user to manipulate the

walker, however, establishing a comparison between all controllers and scenarios is complicated due to the subjective nature of the results and the diversity of scenarios.



(a) **(b)**

Figure 3.4: Examples of case-by-case analysis of the SC in the state-of-the-art controllers. In (a), Huang et al. [5] compare the trajectories performed by the actual with the trajectories predicted to be performed by the intention and autonomous controllers. In (b), Jiménez et al. [6] intended to evaluate the pattern of the SC when the *UFES* walker and patient were positioned outside the predefined path.

Qualitative metrics are used by Jiménez et al. [6] and Sierra et al. [15] through the use functionality questionnaires. Sierra et al. [15] use question such as: "I think the robotic device makes me feel safe?", "I think the device guides me well", "In this session, I felt that the device had the control of the path to be followed", among others, while Jiménez et al. [6] ask "I felt that the smart walker was guiding me" and "I felt an intuitive interaction with the smart walker". Answers to both sets of questions lead to the conclusion that both SCs are able to reach the set objectives [6, 15]. The question used are useful from an ergonomic standpoint, however, they do not necessarily indicate safety and efficiency in rehabilitation or guided assistance, and do not help in the comparison between literature controllers.

Quantitative metrics try to solve the subjectivity issue from the case-by-case analysis and quantitative metrics. Jiménez et al. [6] use the *Kinematic Estimation Error* which compares spatially the global path and the actual path of the *UFES* walker. The *Kinematic Estimation Error (KTE)* is expressed by the following expression

$$\text{KTE}(x_{\text{actual}}, x_{\text{global path}}) = \sqrt{\left[\mathbb{E}\left[x_{\text{actual}} - x_{\text{global path}}\right]\right]^2 + \text{Var}\left[x_{\text{actual}} - x_{\text{global path}}\right]} \tag{3.2}$$

where x_{actual} and $x_{\text{global path}}$ are the actual and global path coordinates described in the rehabilitation scenario. The highest KTE values registered by Jiménez et al. [6] were 0.3 ± 0.1337 which might infer that the walker is able to diverge from the initial planned global path. Graf and Schraft [23] and Sierra et al. [15] measure the relative ammount of time that the SC prioritized the autonomous and intention modes throughout the episodes. Sierra et al. [15] register an average of user control of 66.71% over 7 subjects replaying 10 episodes each, while Graf and Schraft [23] registered 31%. Sierra et al. [15] also measured the average velocity of $0.3m/s$ which is within normal gait velocities - important to avoid falls. Lastly, Spenko et al. [22] compares the distance to obstacles registered in scenarios with SC and free-driving, reaching the conclusion that the controller is able to always maintain a distance to obstacles with increased average (around 0.17) and diminished variance. Following the conclusions referenced by the authors, the controllers proposed indicate themselves capable of solving the issue of this dissertation. The quantitative metrics present are indeed more objective, although the lack of standardization compromises, again, the comparison between the different controllers, *e.g.*, the 66.71% and 31% of scenario time given to the patient registered by Sierra et al. [15] and Graf and Schraft [23], respectively, can lead the reader to think that the controller proposed Sierra et al. [15] is better than the one from Graf and Schraft [23], however, the comparison is not valid due to neither the environment configuration nor the paths described being the same.

Overall, the metrics presented by literature vary from author to author, compromising the validation process of each controller developed. Case-by-case scenario analysis, despite validating with an inherent subjectivity and only being capable of validating the SC for use in scenarios similar to the ones tested (not being able to extrapolate to general scenarios), are the ones that are most capable of understanding directly the behaviour of the walker, because they only try to explain certain constrained scenarios. Qualitative metrics and Quantitative metrics are based on simple and indirect heuristics that might be able to indicate the efficient and safe use of the device, however most metrics might lead to a false sense of validity for general scenarios of aided locomotion.

Summary In the following Table 3.1, a summary of the SC frameworks that exist in the state-of-the-art are presented, with the respective properties and functionality bullet points.

Table 3.1: Summary of SCs available in the state-of-the-art.

Authors (Walker)	Path Following	Obstacle Avoidance	Shared Controller	Bullet Points
Morris et al. [20] (*Normad XR4000*)	✔	✖	Angle Threshold	- Angle between ideal and walker paths
Graf and Schraft [23] (*Care-O-bot II*)	✔	✔	Angle Threshold	- Angle between obstacle tangent and walker path
Sierra et al. [15] (*AGoRA*)	✔	✖	Angle Threshold	- Angle between ideal and walker paths - Windowed long term approach
Jiménez et al. [6] (*UFES*)	✔	✖	Impedance Controller	- Angle between atractor-path and intention velocites
McLachlan et al. [24] (*UTS*)	✔	✖	Fuzzy Controller	- Fuzzy Rules - Membership Function
Huang et al. [5] (*COOL Aide*)	✖	✔	Impedance Controller	- Relative Distance and Velocity to obstacles
Spenko et al. [22]	✔	✔	Impedance Controller	- Difference in position, velocity and acceleration between ideal and actual path; - Stability Criteria

Chapter 4

Review of Reinforcement Learning

Go is a board game, created in China 3000 years ago, which puts 2 players competing against each other. The basis of the game is placing stones (either white or black dependent on the player) on the board creating territory boundaries. The player that covers the most area in the board is the winner [44]. The number of available configurations in the board is not trivial to calculate because not all permutations of board configurations are valid, so Tromp and Farnebäck [45] proposed a method that calculates in a board of 19x19 (standard board size) there are $3^{n^2} = 3^{19^2} = 1.74 \times 10^{172}$ valid and non valid board configurations and 1.2% of those are actual legal positions. Each player has to filter the valid from the invalid plays, which might be difficult in some circumstances, and within each valid calculate which presents itself as the most promising play.

In March 2016, the algorithm AlphaGo [46] developed by DeepMind "competed against legendary Go player Mr Lee Sedol, the winner of 18 world titles, who is widely considered the greatest player of the past decade" with a 4 - 1 victory in AlphaGo's favor [44].

AlphaGo uses DRL as its functionality basis, by tuning a DNN that determines what are the best actions for each board configuration and another that determines how favorable a choice is to make. It tunes both DNNs by capitalizing on a reward function which represents the autonomous agent wins [46].

The use of RL is explored within the context of this dissertation, mainly for two reasons: these algorithms use an abstract reward capable of being specified by a medical expert with ease, and are capable of learning highly complex behaviours, while avoiding the use of simple heuristics to perform any task.

In this chapter a review of theoretical concepts about RL is presented. There is a wide range of algorithms within the field of RL but this is review is pointing towards explaining DRL, more specifically

PPO because this is the used approach in this dissertation.

4.1 Basics

RL is a scientific field dedicated to understanding how an agent is able to learn how to perform a certain task by trying to maximize a reward [27]. It does so by interacting with the task's environment and figuring out which actions obtain the greater rewards, towards possibly converging to a behaviour that achieves task completion. The simple pipeline of any RL algorithm is shown in Figure 4.1 and 4.2.

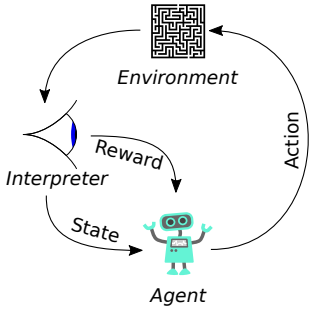


Figure 4.1: Schematic showing the main components of an RL algorithm.

Note. By Megajuice - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=57895741

Reinforcement Learning

Choose an action a according to the agent's behaviour;

Perform the chosen action a on the environment;

The information needed for the decision making process is registered (state s) once the effects of the action a have taken place;

A reward r is attributed to the transition between states and the respective action a performed.

The agent uses the information from the transition and reward to tune its behaviour.

The cycle repeats with the new environment configuration and the updated behaviour.

The algorithm ends when a certain task performance criteria is met.

Figure 4.2: Reinforcement Learning generic algorithm [27].

The agent's behaviour is always tuned so that the maximum amount of reward is earned in future interactions, meaning that if a certain action obtained a good reward, the agent chooses to perform that action in that environment configuration and vice-versa. It can be stated that the agent learns by making good decision but also by making mistakes [27].

4.1.1 Markov Decision Process

Markov Decision Process (MDP) is a mathematical framework that models the decision making process. It can be described by relationships between 4 vector spaces - state space S, action space A, probability space P and reward space R. As explained before, the agent interacts with the environment at each time step selecting an adequate action a for each state s, registering also the "next state" or "future state" s' and reward r [47].

A possible configuration of a certain MDP is shown in Figure 4.3, which contains multiple states (blue square) and actions (red circles). The state-action pairs are connected to another state with a reward and a probability associated. The reward represents the "quality" of each transition, in terms of task performance.

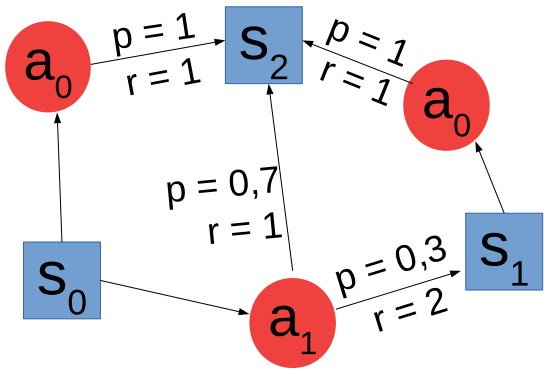


Figure 4.3: Graph of a possible MDP.

MDPs are stochastic frameworks so there is a uncertainty associated with each transition such that [47]:

$$\sum_{s' \in S} \sum_{r \in R} p(s', r|s, a) = 1, \text{for all } s \in S, a \in A(s) \tag{4.1}$$

where p is the probability of a transition. In the example given above, it can be seen that if the agent is situated in s_0 and executes action a_1, it has a chance of ending up in s_1 and s_2 of $0{,}3$ and $0{,}7$, respectively.

With each transition - triplet (s, a, s') - there is an associated reward, but due to the uncertainty of

each transition it is useful to determine the expected reward of each state-action (s,a) pair [27]. The expected reward obtained when performing a at s is calculated as suggested in the following expression.

$$r(s,a) = \sum_{r \in R} r \sum_{s' \in S} p(s', r|s, a) \tag{4.2}$$

In the case of Figure 4.3, the expected reward of performing a_1 on s_0 is:

$$\begin{aligned} r(s = s_0, a = a_1) &= (r_{a_1,s_2} \times p_{a_1,s_2}) + (r_{a_1,s_1} \times p_{a_1,s_1}) \\ &= (1 \times 0{,}7) + (2 \times 0.3) = 1{,}3 \end{aligned} \tag{4.3}$$

A simple example of an MDP is the mathematical formulation of playing in a slot-machine. The state space is filled by the possible configurations of the slots and the action space is filled with only the "play" or "not play" options. Each configuration of the slot when played has a chance of ending up in any state, and the reward corresponds to the amount of money it outputs with such transition.

Up until this point, the reward of each transition is given as a known component, but in a real context it is up to the user to describe a reward function that specifies the optimal behaviour of the agent on a certain task. On simple tasks like the slot-machine context, it is simple to understand that the reward should be the amount earned with a certain transition. On complicated tasks like winning a game of Go, it is hard to quantify individually each transition between board configurations, because one only wins when the game is over [27]. A "simple" solution to the problem, for the game of Go, is considering that the sequence of board configurations (state) and piece placement (action) from start to the end of the game group together as one super state and action, respectively. The agent is rewarded in case the state-action pair originates a win, otherwise no reward is attributed. This solution poses serious issues from an optimization point-of-view, despite being theoretically possible. The shaping of the reward in any MDP needs to describe well the task at hand, considering also the demands forced by the certain reward function upon the capacity of an agent to learn from highly-abstracted rewards, like the one mentioned before for game of Go [27].

4.1.2 Policy and Value Functions

With MDPs as a formulation of the environment, it is time to define how the reward is maximized by the learning agent.

Up until this point, it was defined that the agent tries to maximize the reward at every step. This is not necessarily true because, in most tasks, if the reward is maximized at every step, the overall reward

accumulation on the long-term might be compromised. When considering investing in any business, the businessman tries to maximize his earnings, but there might be some short-term losses that allow for long-term gains. Then, instead of maximizing rewards at every step, the cumulative reward is proposed as maximization target

$$G_t = r_{t+1} + r_{t+2} + ... + r_{t+T} \tag{4.4}$$

where G is the return, t is the time step index and T is the number of time steps to the task's episode [27].

An episode is defined by the interval between the start and end of the task. An episode might be finite or infinite. On both types of episode, maximizing the total return might pose some problems like obtaining infinite return on infinite-type episodes when the optimal behaviour is considered. Then, to learn the optimal behaviour, one has to consider a certain limited horizon that allows the agent to track long-term performance while paying attention to short-term actions. For that, the concept of discounted return is introduced:

$$G_t = \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^T r_{t+T} = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{4.5}$$

where γ is the *discount rate* between 0 and 1. This means that, at each time step the importance of each reward falls by a factor of γ. With a quick analysis it can be seen that Equation 4.5 is a geometric series and so convergence is always guaranteed for $0 \leq \gamma < 1$. It also can be noticed that when γ is 0 only the immediate reward is considered, and when is 1 (despite convergence not being guaranteed) the full episode is considered [27].

Now, with a clear idea of what actually the agent is trying to maximize, the concept of policy is introduced. Policy is the entity responsible for describing the behaviour of a certain agent because it constitutes the mapping between states and the probability associated with the execution of each action

$$\pi : S \rightarrow A \tag{4.6}$$

.

The "quality" of such policy is given by its associated value functions, making RL the algorithm responsible for finding the optimal policy π^*, which is the policy (or policies) that maximize the respective value functions. There are two types of value functions: the state-value function v and the action-value function q [27]. The state-value function outputs the expected return when a policy π is always followed

from state s thereon, as showed by the following expression.

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right] \tag{4.7}$$

The action-value function outputs the expected return when a certain action a is executed at state s and a policy π is followed thereafter:

$$q_\pi(s,a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] \tag{4.8}$$

.Each of the value functions has specific uses in the learning process, because both track the performance of the policy, but have the capacity to be more specific or more generic, with respect to the action-value and state-value functions, respectively. Equations 4.7 and 4.8 are correlated by

$$v_\pi(s) = q_\pi(s,\pi(s)) \tag{4.9}$$

, so the state-value function can be described as the action-value function when the actions analyzed are always drawn from the policy π. The reverse correlation is also determined as the following expression 4.10.

$$q_\pi(s, a, s') = r(s, a) + v_\pi(s') \tag{4.10}$$

4.1.3 Learning

The learning process can be performed in different ways, some more time efficient but more memory-consuming and vice-versa. Dynamic Programming (DP) and Monte Carlo (MC) methods are the ones classically used as learning processes but an amalgamation of the two called Temporal-Difference (TD) is also used. In this sections all methods are going to be discussed.

4.1.3.1 DP

In dynamic programming the learning process occurs by following the Generalized Policy Iteration method. This method contains 2 essential elements: the policy evaluation and the policy improvement. The Policy Evaluation refers to how the value functions are estimated based on the policy behaviour and the Policy Improvement step refers to how the policy is changed *w.r.t.* the new value function [27].

Policy Evaluation is done by recursively, through Bellman equations, calculating the values of each

state based on the value of the next state:

$$v^{\pi}(s) = r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s,\pi(s)) v^{\pi}(s') \tag{4.11}$$

Policy Improvement is done by finding the best actions for each state:

$$\pi'(s) = \arg\max_{a} \sum_{s',r} p(s',r|s,a) \left[r + \gamma v_{\pi}(s') \right] \tag{4.12}$$

With a continuous cycle of Evaluation and Improvement, shown in Figure 4.4, both policy and value functions are guaranteed to converge to an optimal state, where the task is performed perfectly [27].

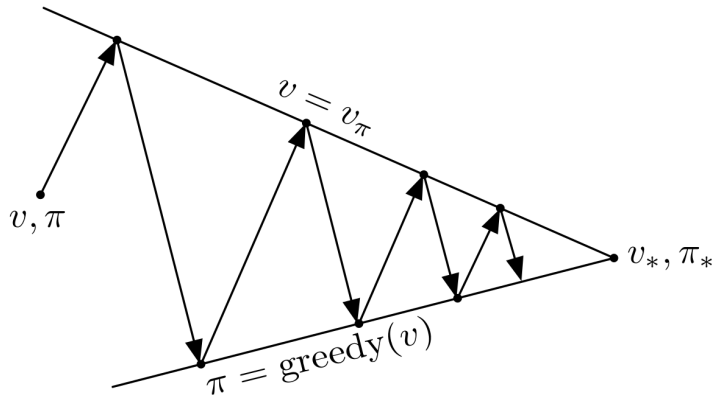


Figure 4.4: Generalized Policy Iteration Schematic [27].

From Equations 4.11 and 4.12, one can determine that a model of the environment is needed - to know the transition probability p one requires a model to determine which future states s' are reachable from current state s. Dynamic Programming approaches allow for solving model-based problems - a problem where the full transition distribution is known - while model-free problems are impossible to solve. Creating models of games like Tic-Tac-Toe is easy, but in the context of autonomous driving the task is not trivial. Also, DP does not scale well due to the curse of dimensionality, caused by the recursiveness in the equations [27].

4.1.3.2 MC Methods

MC methods use a similar framework to the Generalized Policy Iteration except the way that policy evaluation is done, because they try to remove the model required by the DP algorithms. Due to a model not being present in MC, an approximation of the value's distribution is needed. The approximation is

done by averaging multiple returns G from episodes starting from state s. Two problems arise from the last requirement:

- Because no model is present, multiple trajectories need to be rolled out so that the agent understands the dynamics of the environment at hand. So, to infer the value of a state at least one trajectory needs to include that same state, otherwise no information about its viability is available due to the unavailability of a model. Due to the stochasticity of the MDP if only one sample of each state is available, the estimation will have low chances of representing the actual average value of the state.

- When creating an estimation of the value function, the state-action pairs included in the episode-sample-dataset must be representative of the state and action spaces. In certain cases, the dataset acquired might only contain samples from a subset of both the state and action spaces, due to bad exploration. If optimal behaviour lies outside of one of these subsets, then the agent will not be aware of such fact and will never try to converge to such behaviour.

MC methods solve the first problem by rolling out numerous episodes, so that the expectation values are the most accurate possible and slowly update the estimation with the following expression

$$v(s_t) \leftarrow v(s_t) + \alpha \left[G_t - v(s_t) \right] \tag{4.13}$$

with α being a step-size for smooth the updates [27]. With multiple samples available and with small enough α the value converges to an accurate estimate.

The second problem is solved with use of on- and off-policy methods. On-policy methods try to find the best policy by using just one policy as expected. This means that the training process will always be executed with trajectories (episode samples) performed by the policy itself. Off-policy methods find the best policy by sampling trajectories with a different policy, creating the need for a target and a behaviour policy. The target policy is the actual policy that the agent is trying to learn, while the behaviour policy translates how the agent searches the state-action space [27].

4.1.3.3 TD

TD methods work as an alternative solving the issues of previous methods which are: DP is model-based and MC does not bootstrap (update estimates based on other estimates). TD fixes these issues by dropping the need to rollout full trajectories using the bootstrapping of DP through the estimation

of v, consequently dropping the need for a environment's model [27]. So, the equations 4.11 and 4.13 can be merged into the Temporal-Difference value update with the following Equation 4.14 [27].

$$v(s_t) \leftarrow v(s_t) + \alpha \left[r_{t+1} + \gamma v(s_{t+1}) - v(s_t) \right] \tag{4.14}$$

4.2 Deep Reinforcement Learning

4.2.1 Deep Learning

Deep Learning and DNNs have achieved revolutionary results on speech recognition, computer vision, pattern recognition, and many others [48]. DNNs are able to, for example, classify images of animals and their breeds just from "feeding" it examples of what is and what is not a certain type of animal [49]. These DNNs are included within the group of the function approximators, because they transform a certain dataset into a parameterized model [49]. Then, one only needs to know the parameter values and model to fully represent the initial dataset distribution, reducing enormously the amount of memory and computation time to estimate the output of a certain distribution [27].

These networks, like the one in Figure 4.5, are composed of various layers each with a specific function. The layers connect to each other sequentially and their multiple connections from the input to the output and activation functions are able to model highly abstracted non-linear models [49].

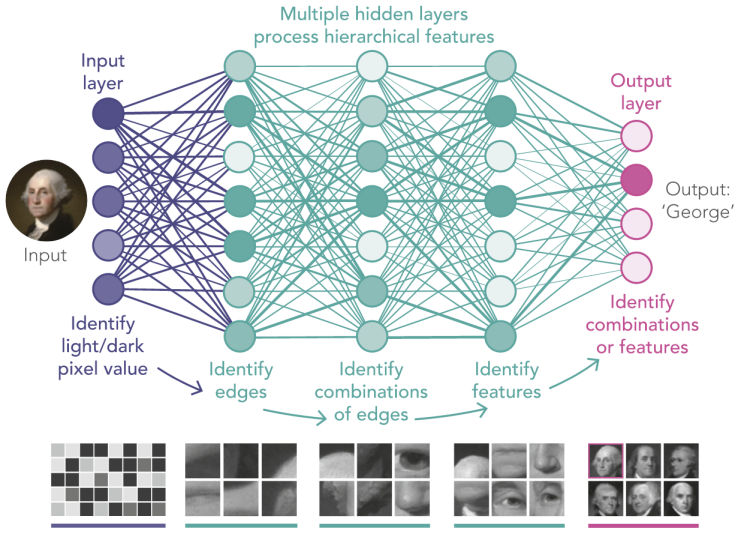


Figure 4.5: Example of a DNN. Adapted from [50].

There are two ways of training such networks: supervised learning and unsupervised learning [49, 51]. Supervised learning tunes the network by minimizing the error between the output of the network and the

set of ideal outputs when the network's input is subjected to a certain vector, meaning that is going to try to mimic a pre-built dataset [49]. Unsupervised learning on the other hand is going to find the best network parameters by tuning them by finding structure and patterns within the data [51]. Algorithms like gradient descent and its variations for non-differentiable equations are used to adjust the network parameters minimizing the output errors mentioned before - either by using external metrics as in supervised learning [49] or using internal metrics in unsupervised learning [51].

Different kinds of layers originate different behaviours. Fully-connected layers are the ones responsible for creating generic non-linear classifiers or regressors in the discrete and continuous domains, respectively [52]. Convolutional layers are mainly responsible for the success of computer vision because they detect local features, taking the context of the input into consideration. In Figure 4.5, the first layer is convolutional and is able to detect edges within the images [52]. The symbiosis within the full structure of the neural-network allows for the creation of a useful and efficient non-linear model.

4.2.2 Deep Actor-Critic

RL algorithms can be split into two sections: policy-based and value-based. Policy-based algorithms, upon each update, create a new estimation of the value function based on the trajectories sampled and update the previously saved policy [27]. Value-based algorithms, on the other hand, update the saved estimation of the value function and create a new policy every time [27]. Policy-based algorithms have faster convergence and are good with continuous and stochastic environments, while value-based are more sample efficient and stable [27].

When using either policy-based or value-based algorithms, some structure will need to function as an approximator for the component saved (policy or value). In the Basics section of this Chapter, the information shown refers to discrete state and action spaces. When considering a continuous environment the properties and algorithms can still roughly apply but the dimensionality of the problem increases drastically. In simple problems, *e.g.*, the game of tic-tac-toe, there is 255.168 possible configurations, so by today's computer memory capacity, a table or a table-like structure can be used to store the optimal action for each environment state. When considering cases where either the environment is not discretizable (autonomous driving scenario) or the dimensionality is enormous (game of Go), saving the optimal policy π^* in a table-like structure is not feasible for two reasons: there is not enough memory to store the information, and even if this was not the case it would take a lot of time just to find the position of the table where a certain state is positioned.

Deep Actor-Critic algorithms try to solve the issues originated by the curse of dimensionality and by

policy/value-based algorithms. Actor-Critic algorithms consist of a mix of the policy- and value-based kinds of methods, by keeping an estimation of value and policy, avoiding the need always build new estimates from scratch [27]. Figure 4.6 depicts a diagram of the actor-critic pipeline.

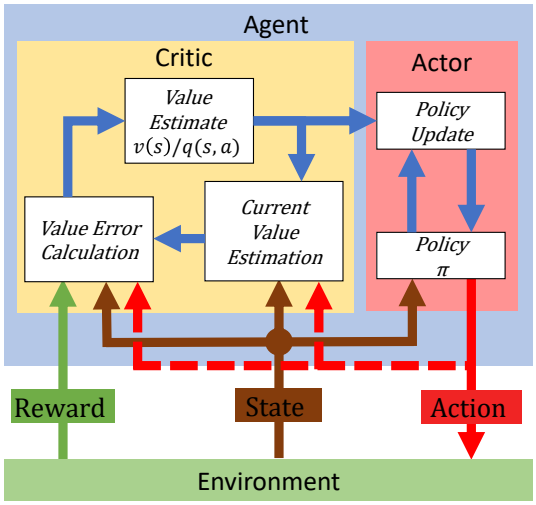


Figure 4.6: Diagram of the Actor-Critic algorithms.

The Actor (Policy) depicts how the agent behaves while the Critic "criticizes" the actions of the actor based on previous knowledge of the state (v) / action (q) values. Both Actor and Critic always keep an estimation of both policy and value function, respectively, removing the necessity for creating a new estimation of each at every cycle.

Deep Actor-Critic, inherit the structure of Actor-Critic, trying to solve the scalability issue, using DNNs as estimators of the value function and policy.

4.3 PPO

PPO is a Deep Actor-Critic algorithm and it is going to be detailedly analyzed due to being used in the context of this dissertation. It was first proposed by Schulman et al. [53] in 2017 with the support of the OpenAI team. PPO focuses on achieving 3 objectives:

1. Improving the agent performance - This is done by a similar policy gradient approach using the advantage function A part of gradient estimator.

2. Stabilizing the training process - If the agent, coincidently, evaluates a group of trajectories as "extremely good", it has a big incentive to update its policy drastically and might compromise the whole training process converging to a local instead of a global maximum.

3. Acting in a conservative manner - If no evidence exists proving that a certain policy is good, then the behaviour should stay generic allowing for exploration.

The loss function defines the performance of the current state of the DNN allowing optimizers - algorithms responsible for running gradient descent algorithms [54] - to tune parameters towards reducing the loss [27]. To translate the mentioned goals of PPO, the policy is tuned with the following loss function

$$L(\theta) = \mathbb{E}\left[L_{clip}(\theta) - aL_v(\theta) + bS[\pi_\theta(s)]\right] \tag{4.15}$$

, where L is the global, L_{clip} the clipped surrogate, L_v the value and S the entropy loss functions, and θ the network parameters.

The clipped surrogate loss emphasizes the necessity of improving the agent's performance, but not allowing the policy change its behaviour drastically. It uses the Advantage metric A which measures how much better it is to perform a certain action over the action advised by the current policy in state s, *i.e.*, what is the relation between $q_\pi(s,a)$ and $v_\pi(s)$. The following Equation 4.16 expresses the advantage A formally.

$$A_\pi(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t) = R_\pi(s_t, a_t) + V_\pi(s_{t+1}) - V_\pi(s_t) \tag{4.16}$$

The clipped surrogate loss also uses the probability ratio pr between old and new policies to quantify the change caused by the update. Given equation

$$pr_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{4.17}$$

, the certainty of performing a given s and π_θ is higher, lower and equal if pr_t is higher, lower and equal to 1, respectively, all relative to the previous policy $\pi_{\theta_{old}}$. Finally all the previous components merge into one equation

$$L_{clip}(\theta) = \mathbb{E}_\pi\left[\sum_{t=0}^{T} min\left(pr_t(\theta)A_t, clip(pr_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right] \tag{4.18}$$

where ϵ is the clip-factor responsible for not allowing the policy to diverge - this value usually is around 0.2 Schulman et al. [53]. The first member of the minimization function in J allows the policy to be optimized by maximizing both Advantage and certainty. The second member is responsible for setting the limits for each update of the policy. The minimization is used so that the more conservative choice is always prioritized.

Analyzing PPO from a Deep Actor-Critc point of view, it can be assumed that the actor and critic networks must be essentially similar, due to the inputs of both networks being equal - for both policy and state-value function the input is the state vector - and the networks trying to generalize essentially the same data. The difference between the networks is usually located in the output sections - the policy outputs the action vector and the state-value function outputs a scalar value. The value loss L_v component includes this similarity in the networks, motivating the policy to progressing to where the actual state-values are equal to the estimations.

The last component of the loss function is the entropy S, motivating its increase, allowing the networks to not become overconfident promoting exploration.

With the global loss L fully composed, a gradient-descent algorithm is run on the policy network parameters, converging π to π^*. The state-value network is tuned similarly, but instead of the loss function described for the policy, the mean-squared error loss between the outputs of estimation and true state-values is used. PPO runs episodically like demonstrated in the algorithm of Figure 4.7.

PPO

for iteration = 1, 2, ... do

  for iteration = 1, 2, ..., N do

    Run policy π_{old} in environment for T time steps

    Compute advantage estimates A_1, ..., A_T

  end for

  Optimize L *w.r.t.* θ, with K epochs and mini-batch size $M \leq NT$

  $\theta_{old} \leftarrow \theta$

end for

Figure 4.7: PPO's algorithm. Adapted from Schulman et al. [53].

Chapter 5

WALKit and Shared-Controller Architecture Overview

The current chapter offers a general review of the WALKit walker and the SC architecture implemented in the walker. This chapter tries to expose the walker's features and hardware/software components with and without the implementation of the SC proposed. The WALKit system and environments were also implemented in the simulator *Gazebo* [55] and a description of the simulation environment is also presented in this chapter.

5.1 WALKit Walker

The WALKit is a complex walker composed of multiple systems like gait and posture assessment, local and remote driving modules. In Figure 5.1 a complete view of the walker is shown with its multiple components.

Two motorized wheels in the rear (Figure 5.1b (**F**)) are controlled to steer the walker in any direction, but software restrictions do not allow the walker to drive backwards to avoid collisions with the patient. The walker is fully electric and the motors are supplied by the batteries in the low-level hardware (Figure 5.1b(**I**)). The low-level hardware is also responsible for controlling the motors and relaying the sensor data from wheel encoders, sonars, and others to the high-level computer.

The patient controls the motorized wheels by moving the handles (Figure 5.1c (**J**)) to the desired direction (front, left, right) or leaving it in neutral (no movement). The walker can also be controlled remotely, but this is not included in this dissertation because the remote control is only used by experts.

(a) **(b)** **(c)**

Figure 5.1: WALKit overview. (a) Front, (b) rear, and (c) top views are presented.This figure shows the RGB-D cameras for postural and gait analysis ((**A**) and (**C**) respectively), (**B**) touch screen, the (**D**) lifting columns, (**E**) high-level computer, (**F**) motors and motorized wheels, (**G**) sonar array, (**H**) infrared sensor, (**I**) low-level hardware, (**J**) patient driving handles, (**K**) forearm support with force sensors, (**L**) emergency button, (**M**) wood table, (**N**) harness (removable), and (**O**) horizontal handles.

A touch screen is included to interface the machine and human so that either the patient or the medical expert can tune maximum velocities, engage the multiple functioning modes (local, remote, autonomous, SCs), perform rehabilitation assessments and record data, and receive feedback from the machine in a human readable way.

Also the walker has additional features important to tasks unrelated to this dissertation like: video-depth cameras associated to gait and posture assessment algorithms; lifting columns, table and removable harness for increased comfort and support; and other sensors that offer a more complete depiction of the system patient-walker-world.

In relation to the architecture of the software, represented in Figure 5.2, the walker splits into two parts: high-level - responsible for running highly abstracted and computationally heavy algorithms like graphical interfaces, operative systems, gait and quality assessment (computer vision) - and low-level - responsible for interfacing the physical components like sensors and actuators and also running some low level software and firmware such as proportional–integral–derivative controllers. All the software in the high-level needs to always communicate with other components in an efficient and organized way, and for that the software Robot Operative System (ROS) is used [56]. ROS requires all communications between modules to be

centralized, *e.g.*, if the autonomous mode is driving the walker, the message from the sensors is acquired and structured by the low-level components, sent to the high-level ROS core, interpreted by the autonomous controller, and a command finally sent back to the ROS core which redirects to the low-level consequently to the wheels. In the sections 2.4.1 and 2.4.2 it was mentioned that only some planners, either global or local would be analyzed. This restriction was based on the requirement that the algorithm chosen needed to be included within *ROS Navigation Stack*[1], guaranteeing the robustness of the packages.

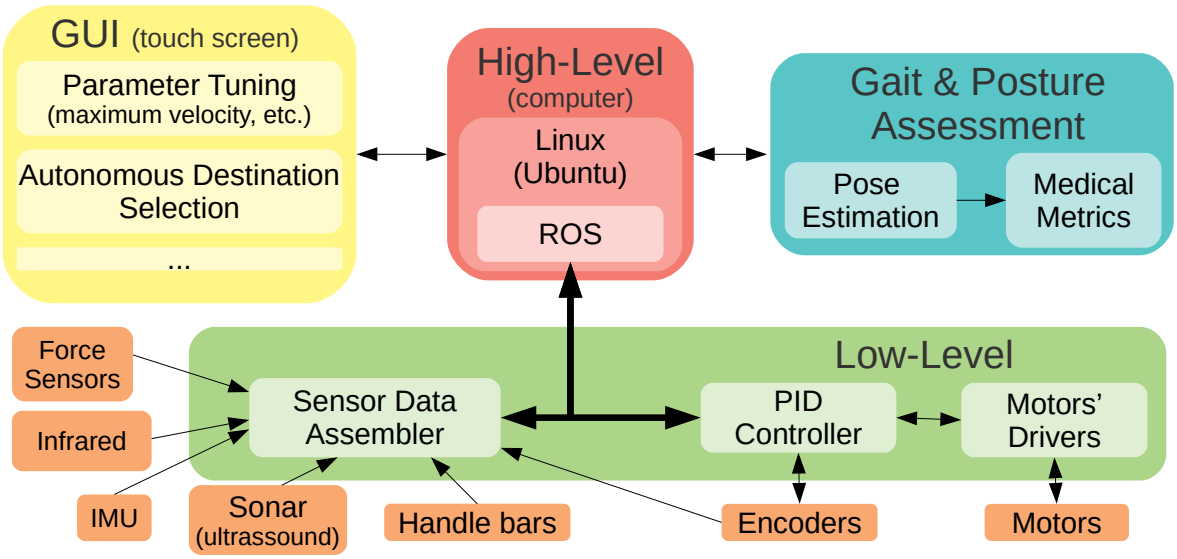


Figure 5.2: An overview of the hardware and software deployed in the WALKIt walker.

The walker is designed with maximum linear and angular velocities of $1\ m/s$ and $0.39\ rad/s$ so that patients with GD can keep up with the walker's movement, and it should always guarantee smooth movement so accelerations are always limited to a maximum of $2.0\ m/s^2$.

For further reference in this dissertation, a origin reference frame will have to be established as "root" for the whole walker, and it will be mentioned solely as "origin" or "origin point/frame". The origin is positioned at the point with equal and lowest distance from both back wheels, as shown in Figure 5.3.

[1] http://wiki.ros.org/navigation

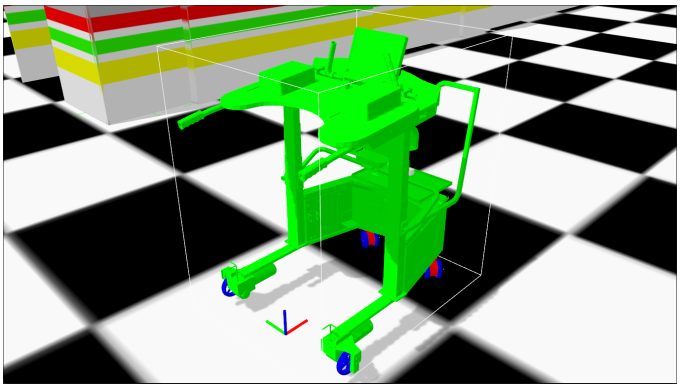


Figure 5.3: Origin reference frame of the WALKit walker. Also the XYZ axis are colored in red, green and blue respectively.

When a certain algorithm publishes velocity commands it does so by describing the linear and angular velocities. The linear velocity is a vector aligned with the X axis of the origin reference frame, and the angular velocity is expressed around the Z axis. These commands are translated into angular wheel velocity commands by the expressions

$$\begin{aligned} w_{left}\left(v_{walker}, w_{walker}\right) &= \frac{1}{r_{wheel}}\left(v_{walker} - w_{walker}\frac{d}{2}\right) \\ w_{right}\left(v_{walker}, w_{walker}\right) &= \frac{1}{r_{wheel}}\left(v_{walker} + w_{walker}\frac{d}{2}\right) \end{aligned} \tag{5.1}$$

where v_{walker} is the linear velocity, w_{walker} the angular velocity, r_{wheel} the rear wheel radius, d the distance between the rear wheels, and w_{left} and w_{right} the angular velocities of the left and right wheels, respectively.

When driving the walker it is essential that the walker does not collide with the patient. The origin reference frame was placed at the position which the patient is located, so that the walker always turns using the patient vertical axis as turning axis.

5.1.1 Simulator

Developing and debugging all the components introduced in the walker is a time-consuming process so a simulator was constructed to aid in this phase. The simulator must replicate the walker-world physics interactions, the hardware present and the software restrictions.

Koenig and Howard [55] developed a robotics simulator named *Gazebo*, that evolved into a highly benchmarked framework capable of being interfaced with ROS [57] while having a diverse library of modules that simulate real world hardware components like LiDAR, ultrasound, motor controllers, etc.

The WALKit mechanical design was created using the Computer-aided design (CAD) software *SolidWorks* (Dassault Systèmes SE, Vélizy-Villacoublay, France). To export the design to *Gazebo*, ROS offers the library *solidworks_urdf_exporter*[2] that creates an Unified Robot Description Format file with the correspondent STL meshes files for each part. The walker CAD design contains multiple parts and subparts, so it was deemed acceptable to export the static components as one part and the movable separated from each other. Then, the chassis, touch screen, handles were exported as one part (Figure 5.4 (green)), while the rotating pieces from the front wheels (Figure 5.4 (red)) and wheels (Figure 5.4 (blue)) are independent.

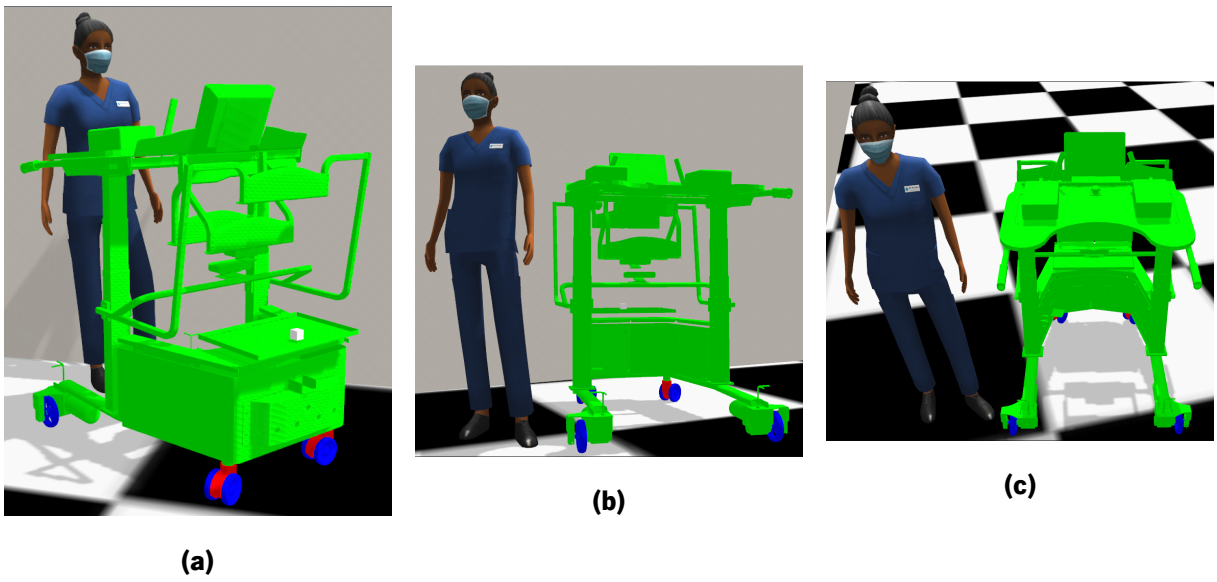


Figure 5.4: WALKit Gazebo model. (a) Front, (b) rear, and (c) top views are presented. A model of a nurse is included for spatial scale reference. The chassis is green, the rear and front wheels are blue, and the rotating front pieces are red.

In the real walker, to send a velocity command to the wheels and to determine the actual rotational velocity of the wheels the data sent and received is converted through multiple modules, like shown in Figure 5.5 but, when using Gazebo, multiple pre-developed and tested packages like *gazebo_ros_control*[3] exist and allow to fully interface with ROS. Ideally, it would be better to use exactly the same packages in both simulated and real environments, but due to some constraints in the real walker requirements this was not possible. A workaround to this problem is the existence of an interface, so that high-level information can adequately transform depending if the real or simulated environments is used.

[2]Source code: https://github.com/ros/solidworks_urdf_exporter
[3]Source code: https://github.com/ros-simulation/gazebo_ros_pkgs

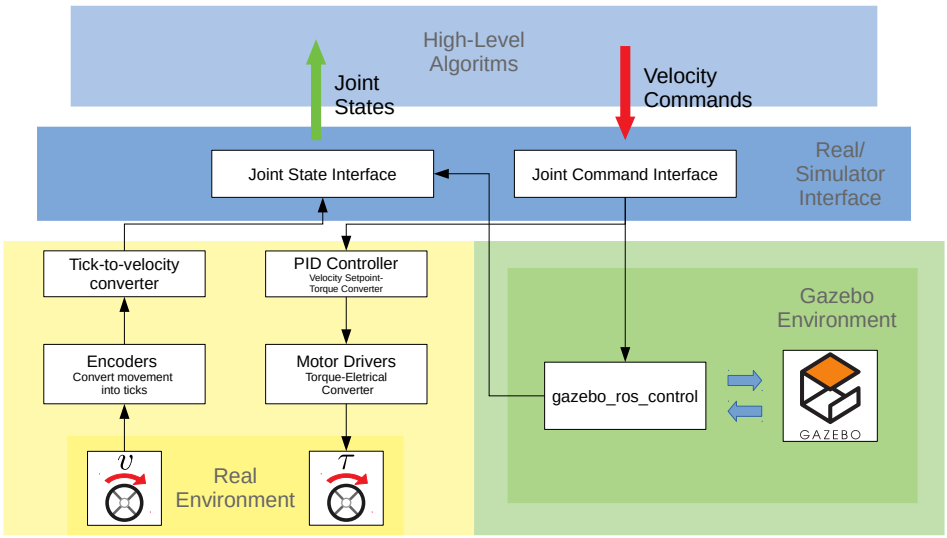
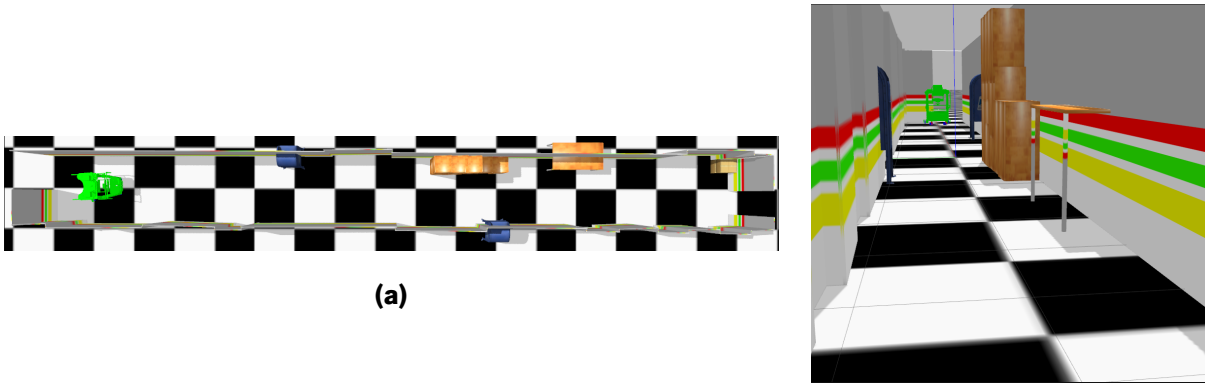


Figure 5.5: Control framework schematic for real and simulated environment.

In Figure 5.5, the framework shown only represents the control portion of the global framework and a similar structure is used for the sensor system.

Two *Gazebo* worlds were used in the simulation, with very different properties:

- Corridor - This corridor, shown in Figure 5.6, was created from scratch and it is generally used to benchmark the walker performance under very strict circumstances due to being narrow and full of obstacles. Usually these environments do not exist in gait rehabilitation scenarios with the walker, but it is useful to stress-test the machine. This world is fully static and only contains obstacles next to the walls.

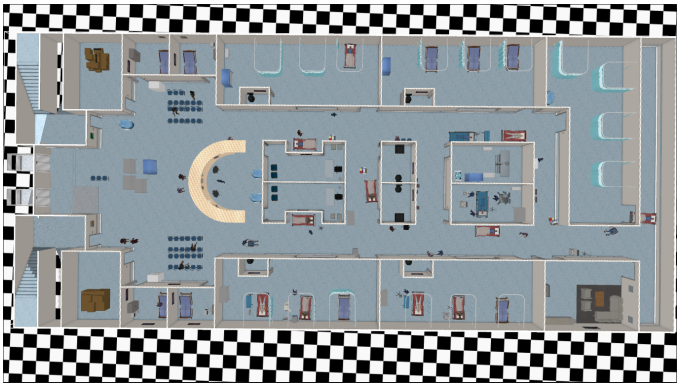


(a) **(b)**

Figure 5.6: Corridor world in the *Gazebo* Simulation. A top view a and perspective view b are presented.

- Hospital - This world, show in Figure 5.7, was adapted from an online repository

aws-robomaker-hospital-world[4]. This world is an accurate depiction of a real hospital, so it is the ideal environment for the development process. Unlike the Corridor world, the Hospital world is full of obstacles, not just close to walls, but also in the middle of corridors which prohibits the walker's passage. The version of the hospital in this dissertation is different from the one in the repository, because more static obstacles were added in, so that corridors and rooms would be more varied. Also, models of nurses move around the hospital with predefined and diverse trajectories so that the dynamic nature of this kind of scenarios is accurately portrayed.



(a)

(b)

(c)

Figure 5.7: Hospital world in the *Gazebo* simulation. A top view a and prespective views b and c are presented.

5.2 Shared-controller Architecture

The purpose of this section is to offer a generic overview of the SC developed in this dissertation merging the different concepts referenced in literature with ours. Nevertheless, each decision is going to be detailedly described in the following chapters 6 and 3.

The SC proposed, shown in Figure 5.8 splits into 3 sections as suggested by most literature approaches: patient intention detector, autonomous supervisor and prioritizer, as mentioned in chapter 7.

[4]Source code: https://github.com/aws-robotics/aws-robomaker-hospital-world

This structured showed an overwhelming acceptance Jiménez et al. [6], Sierra et al. [15], Spenko et al. [22], Graf and Schraft [23] due to inheriting the benefits of modular architectures.

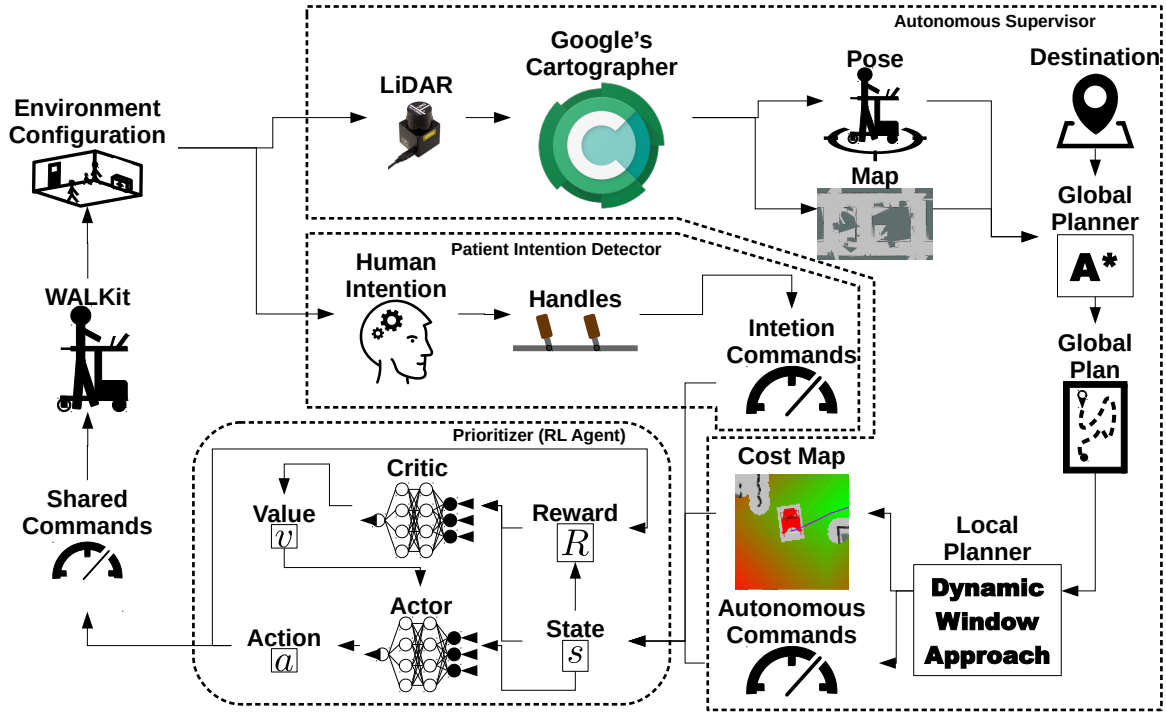


Figure 5.8: Overview diagram of the SC proposed in this dissertation.

To detect the intention of the patient, we resort to the movable handles implemented in the WALKit (see Figure 5.4 (c) (**J**)). As already mentioned in section 3.1, Morris et al. [20] reference that movable handles are not ideal due to risking the well being of the patient. Then, using the WALKit's movable handles might not be safe. However, the table from Figure 5.4 (c) (**M**) is the actual structure that offers physical support, so, the movable handles implemented the WALKit are safe to use in this context, while possibly increasing the intuitiveness of use. The input of the handles is then translated to velocity commands (Equation 5.1).

To replace the supervisor, a SLAM system with global and local planners to locate, map and autonomously control the walker for collision avoidance. The *Cartographer* SLAM algorithm [25] and the LiDAR sensor Hokuyo's scanning rangefinder *URG-04LX-UG01* (Hokuyo Automatic Co., Ltd., Japan) were implemented. *Cartographer* performs its task by splitting the problem into a grid-based method for local small maps and using a graph-based approach to merge all local small maps into one coherent map [25], which enables the WALKit walker to map huge environments with high-precision efficiently, advantages which were already mentioned in section 2.2.2. Also, *Cartographer* was already implemented in a ROS package, facilitating the implementation process. Despite being expensive, Hokuyo's LiDAR system was available for use and, following literature reviews, these sensors are the most promising in

terms of functionality, as referenced in Table 2.1.

To drive the walker autonomously to a certain pre-chosen goal, the *A** global planner was used. The *A** algorithm was chosen due to its increased efficiency over Dijkstra's. *A** converts the pose acquired and map by *Cartographer* into a path capable of reaching the destination. To convert the spatial path to velocity commands (Equation 5.1) readable by the walker, the local planner *Dynamic Window Approach* algorithm was used on the basis that is theoretically more efficient and faster [42]. The *Dynamic Window Approach* ROS package allows for a point cloud representing the cost of the surrounding of the WALKit walker *w.r.t.* the score function used in the local planning process to be published.

To prioritize either the autonomous supervisor or the patient, the PPO algorithm with its Actor-Critic structure is used. The intention and autonomous velocity commands and the local planner's cost map is used as a state space. The velocity inputs allow the RL agent to be aware what the patient and autonomous supervisor recommend. The cost map allows for the DNNs to be aware of obstacles and viability of certain paths relative to the convergence to the goal. The action obtained in the output of the policy (Actor), named admittance value α, determines the relative contribution of each input command to the overall shared velocity. If α is close to 1 the walker is fully autonomous and if close to 0 the walker is fully drivable by the patient. The critic evaluates the performance of the actor network based on a reward function, which motivates the WALKit walker to become autonomous when close to walls (< 0.6 m) and completely drivable by the patient when far from walls. Choosing RL as a framework to learn how to prioritize each source of velocity commands was due to learning the intended behaviour through the use of a highly abstract reward function [27], trying to remove the subjectivity in rehabilitation scenarios performed by human supervisors. The use of PPO was related to the use of DNN as non-linear, generic and scalable function approximators towards achieving the goals set for the controller referenced in chapter 1. If in the future we want to implement the discarded Gait Quality fall risk variable, only minimal changes are possibly needed.

5.2.1 Computation Hardware and Updated Hardware/Software Structure

Following the rationale presented in section 5.1 and Figure 5.2, the full SC is implemented in the high-level portion of the WALKit architecture, due to requiring increased computational power. The changes to the system architecture are presented in the Figure 5.9, showing the addition of the SC and ADS modules.

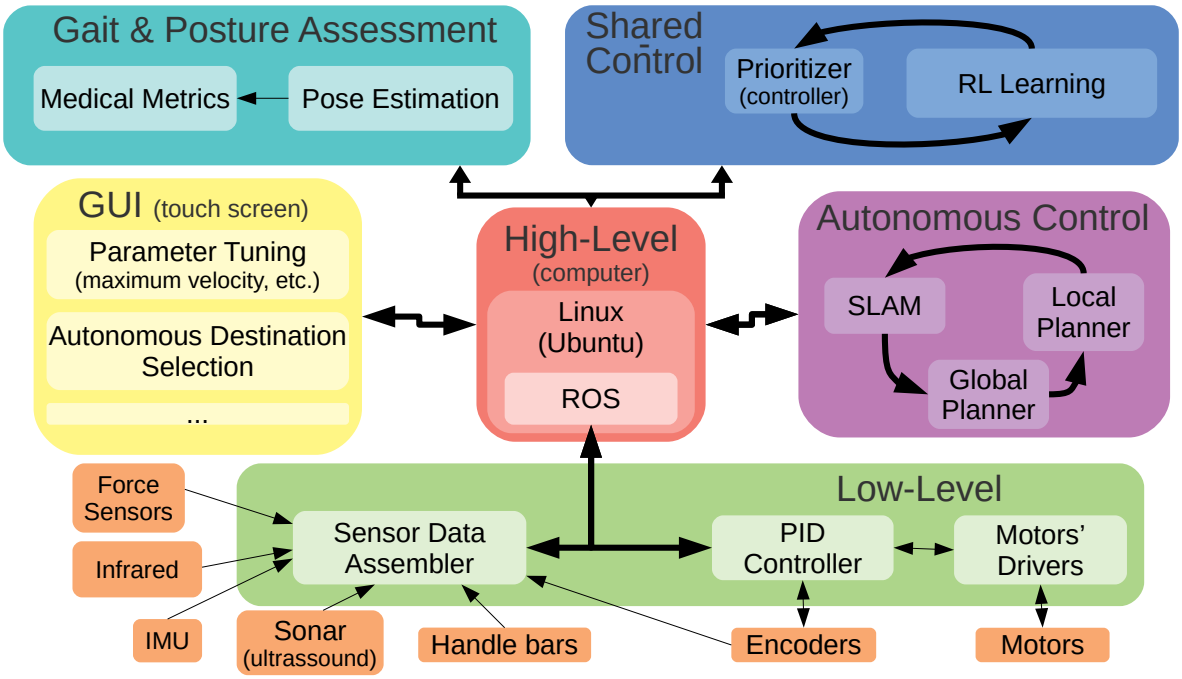


Figure 5.9: An overview of the updated hardware and software deployed in the WALKIt walker.

The following Table 5.1 presents the computational resources available in the WALKit walker and in the computer used for simulation of *Gazebo* environments.

Table 5.1: Computation Resources Available

Property	WALKit Intel NUC-6i7KYK	*Gazebo* PC
Processor Model	Intel Core i7-6770HQ	Intel Core i7-8550U
Processor Frequency	2.6 - 3.5 GHz	1.8 - 4 GHz
No. of Cores	4	4
No. of Threads	8	8
RAM Capacity	8 GB	8 GB
GPU Model	None	None

The computer implemented in the WALKit walker simply runs the algorithms developed, while the *Gazebo* PC not only runs the algorithms developed but also runs simulation and trains DNNs used in the PPO framework, training which will be described in chapter 7. Training DNNs is a very demanding process in terms of computational power, and shows significant better results when using GPUs comparing to the use of CPUs [58]. For that reason, the used computation hardware used in this dissertation is not the most suitable for the task at hand, however certain design choices were made to filter unnecessary data improving efficiency.

Chapter 6

Autonomous Driving

6.1 Introduction

In chapter 2 a description of all the individual components of the developed ADS was presented. In this chapter the implemented ADS will be fully discussed along with the consequences of the various choices along development, with the details about the validation protocols of the system.

As stated before, the main objective of the ADS is to create an accurate mapping of the environment, drive the walker autonomously to the goal and avoid obstacles. These 3 objectives have a high range of possibilities and outcomes considering the high variability of environments.

In summary, a LiDAR device scans the environment's obstacles profile and a SLAM algorithm creates a map and estimates the position within the map. Once a destination is chosen, the global planner creates a path from origin to destination and the local planner translates a portion of the global plan into velocity commands, avoiding obstacles.

The implemented packages which are going to be mentioned in this section are fully interfaced with ROS and are developed in C++.

6.2 Sensors

The ADS needs relevant data to deduce where it locates itself and how to avoid obstacles. The sensors chosen must feed the data to the algorithms while meeting some important criteria. The sensors chosen should be relatively cheap so that the WALKit walker can, in the future, become a useful and necessary tool for rehabilitation while being able to be mass-produced. Also, the sensors should not change significantly

the structure of the walker. The walker was specifically designed to work within certain constraints, and changing the specifications could compromise the whole walker as a rehabilitation tool, so drastic changes on weight or area occupied are not acceptable.

The walker already contains an array of ultrasounds disposed in the front of the chassis, shown in Figure 5.1 (a) (**G**). These sensors were not considered due to their low range, low angular resolution, and occurrences of crosstalk.

The implementation of motion sensors like IMUs was deemed unviable due to the "noisy" signals produced by the nonexistence of ground shock absorbers making the whole walker shake on minimally irregular surfaces.

To use the wheel encoders as sources of odometry, the roll of the wheels should be close to pure, meaning that the wheel should not slide, and so the distance traveled can be calculated using the perimeter and the number of rotations of the wheel. In sudden acceleration and breaking situations the wheel has some difficulty to get a good grip with the floor, producing wheel slide, making the wheel revolutions counted by the encoder not match the distance traveled. It should be referenced that the grip registered in a scenario is dependent of both floor and wheel physical properties. To guarantee equal functionality in all scenarios, using wheel encoders as sources of odometry is not advised due to dependency to the floor properties.

The Hokuyo's scanning rangefinder URG-04LX-UG01 (Hokuyo Automatic Co., Ltd., Japan) was available, and was determined to be a good fit for the established requirements. Its specifications are presented in Table 6.1. It is small and light allowing it to be placed anywhere without compromising the structure of the walker, it has good angular resolution, and requires low power. At maximum range, the laser can register two neighboring points as close as 4.28 cm, resolution which is able to provide a good depiction of the environment (Figure 6.1). The only disadvantage of this laser is its short range (5 m) and consequent failure to detect walls or any obstacles that the walker should be able to detect in a hospital environment, which are beyond the maximum range. Also, the laser scans in 2D, so the autonomous

Table 6.1: Hokuyo's URG-04LX-UG01 Specifications

Property	Value
Range	5 m
Angular Resolution	0,352°
Accuracy	30 mm
Power Consumption	2,5 W
Weight	0,16 kg
Size	5x5x7 cm

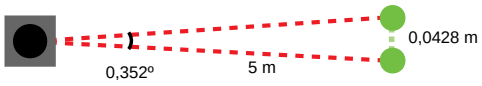


Figure 6.1: Representation of the distance between two points of the a scan obtained by the Hokuyo's URG-04LX-UG01 LiDAR.

driving will only be achievable in a 2D space, so no significant changes in vertical position should exist, which usually is not a problem due to rehabilitation scenarios not being performed in environments with elevations. The 2D nature of the LiDAR system forces an assumption to be made, which is that the environments used must be fully scannable by the rangefinder, *i.e.*, all obstacles must be observable by the sensor point-of-view, so, for example, furniture smaller than the height of the sensor must not be placed in the environment. The best location in the walker for the LiDAR (Figure 6.2) was determined to be on top of the chassis due to being one of the only positions where the field of view is not obstructed by the walker itself nor the patient.

Figure 6.2: Photo of the Hokuyo's scanning rangefinder URG-04LX-UG01 and its positioning in the walker.

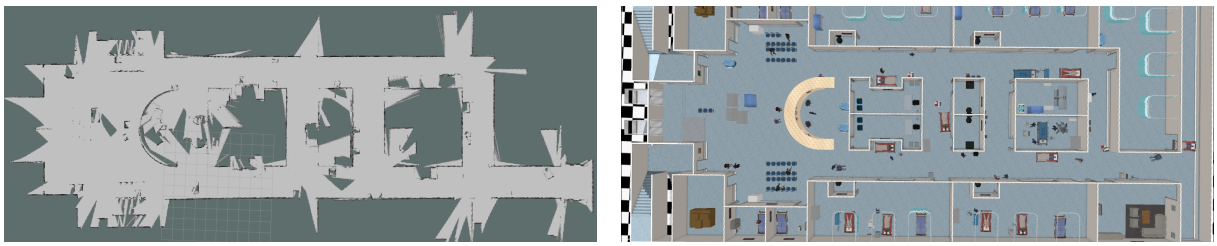
When thinking about the consequences of 240° as field of view, one might conclude that the walker cannot be aware of obstacles behind it. This is irrelevant because in rehabilitation scenarios the walker is not supposed to move backwards so that no collision with the patient are registered guaranteeing patient safety.

6.3 SLAM

With relation to the SLAM algorithm many already existing packages were considered that are widely used: HectorSLAM [59], Gmapping [36], KartoSLAM [60] and Google's *Cartographer* [25].

When guessing the robot's position, many SLAM algorithms need an odometry input, so that an initial guess of the robot's position can be used to bootstrap the optimization process. This requires the development and implementation of odometry computation algorithms, which becomes unviable when noisy sensors (IMU, encoders, etc.) are used. Google's *Cartographer* was specially considered to be the best algorithm because it offers support for a non-odometry-input mode. The scan-matcher - algorithm responsible for calculating the most likely position of the robot given a certain map - of the *Cartographer* software is good enough to infer estimates of initial odometry, without the need for additional sensors and algorithms.

Google's *Cartographer* [25] splits into 2 two parts: the local SLAM - a grid-based approach that creates a small local map (submap) - and the global SLAM - a graph method that establishes constraints between submaps, making sure that the submaps interlace, allowing loop closures. Each scan from the LiDAR is fed as an input and the map, shown in Figure 6.3, and pose serve as outputs of the SLAM algorithm.



(a) **(b)**

Figure 6.3: Comparison between map created by *Cartographer* (a) and the real Hospital simulation environment (b).

6.4 Global Planner

When a destination is selected by the user, the global planner creates a path using the map and pose obtained by the *Cartographer* algorithm.

Cartographer outputs a map in the Occupancy Grid ROS message format which consist of a matrix with values between -1 and 100. The higher the matrix entry value the more likely it is to be an obstacle, and the value -1 represents a entry with no information. Usually path planners do not take the shape of the robot into account and consider that every robot is an infinitesimal point in space. These solvers come from graph problems and do not necessarily comply with the requirements from robotics. In this walker, the infinitesimal point is the origin. If the robot's pose is translated in an entry too close to an obstacle, the walker is probably (or even certainly) in collision, so, some points on the map should not be considered

as viable options for the path planning.

The package *costmap_2d* [1] is a package that creates a map with multiple layers, each with a specific function. The layers are: the static layer - map of the environment -, the obstacle layer - tracks obstacles directly from the sensors -, and inflation layer - pads the area next to obstacles in the other layers. The static layer registers obstacles that are included in the map, so mostly keeps track of mapped static obstacles, while the obstacle layer is usually used for obstacles that are not present in the map like dynamic obstacles. The two parameters, *inflation_radius* and *cost_scaling_factor* from the package need to be tuned carefully so that optimal behaviour can be achieved. The *inflation_radius* is responsible for the size of the padding around the obstacles, while the *cost_scaling_factor* determines how steep the values of padding should decrease along the center of the padding outwards. In this implementation the values used were for 0.6 for the *inflation_radius* and 10 for the *cost_scaling_factor*, with an example of the results in Figure 6.4, tuned by a trial and error. The choice of the parameter *inflation_radius* was based on the area occupied by the walker, in order to pad the obstacles so that the walker does not collide. The *cost_scaling_factor* was chosen based on offering a smooth gradient of cost values to the padding.

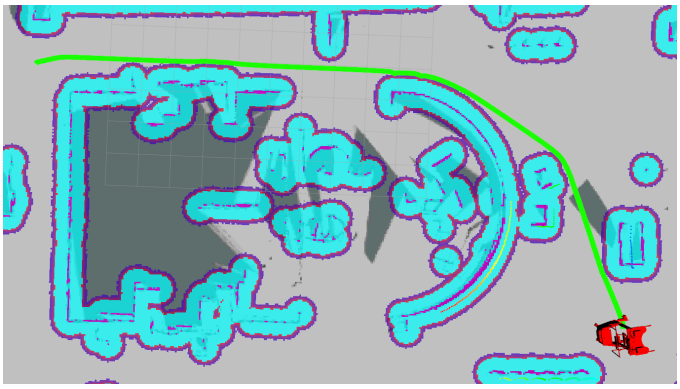


Figure 6.4: Global Path (green line) created by the A* algorithm in a walker scenario. Around the obstacles, a padding gradient(light-blue - red - purple) was created by the *costmap_2d* package.

Now, with the obstacles registered and fully padded, a path can be created. The *global_planner*[2] package contains implementations of both Dijkstra and A*. Only A* (Figure 6.4) was used, due to its computational efficiency (section 2.4.1.2). The parameters *lethal_cost, neutral_cost, and cost_factor* regulate how close the global path gets to obstacle, turning the walker very conservative when these are high. The global planner does not have, theoretically, any temporal constraints, due to only being used at the goal selection (beginning of the scenario). When the patient selects the goal, the path planner is called outputting a path, and unless the walker deviates too much from the initial trajectory the global planner

[1] http://wiki.ros.org/costmap_2d
[2] http://wiki.ros.org/global_planner

is not called again. The time used by the global planner is not shown in the Results chapter 8 because it always planned all the paths adequately in all validation scenarios below 1 second, due to the simplicity of the environment graphs considered.

6.5 Local Planner

With a global plan like in Figure 6.4, the path is then translated into velocity commands. The conversion is done by the *base_local_planner*[3] package. The local planner uses the a portion (5x5 m) of the global map (Figure 6.4) surrounding the walker and the portion of the path included within it and *Dynamic Window Approach* algorithm (section 2.4.2) is run.

Unlike the global planner, the local planner must function at a higher rate because it needs to, at every single time step, simulate possible trajectories and determine which is the best trajectory to follow. So, most parameters that regulate the local planner's behaviour are related to the tradeoff between number scenarios simulated and time used for the calculations. In Table 6.2 the most essential parameters are mentioned, which were tuned by trial and error.

Table 6.2: base_local_planner parameters.

base_local_planner parameters		
Parameter	Description	Value
holonomic_robot	Robot is holonomic	False
sim_time	Trajectory simulation time limit	1 s
sim_granularity	Step-size of the simulation	0.2 s
vx_samples	Number of linear velocities to sample	5
vtheta_samples	Number of angular velocities to sample	5
pdist_scale	Weight of the score given to motivate staying close to the path	0.6
gdist_scale	Weight of the score given to motivate staying close to theighal	0.8
occdist_scale	Weight of the score given to avoiding obstacles	0.01

The output of this algorithm is a vector of both linear and angular velocities compatible with the already accepted velocity commands of the walker (Equation 5.1).

The *base_local_planner* package allows the publishing of the cost cloud created by the local planner while calculating the velocity commands. The cost cloud is a set of points in the space close to the walker that represent the cost to travel through a location, forcing the walker to minimize the cost of the overall path. In Figure 6.5 an example of a cost cloud is presented, where it is possible to observe that with

[3] http://wiki.ros.org/base_local_planner

this "map" the walker can be aware of obstacles, the position within the environment and infer if it is approaching the goal.

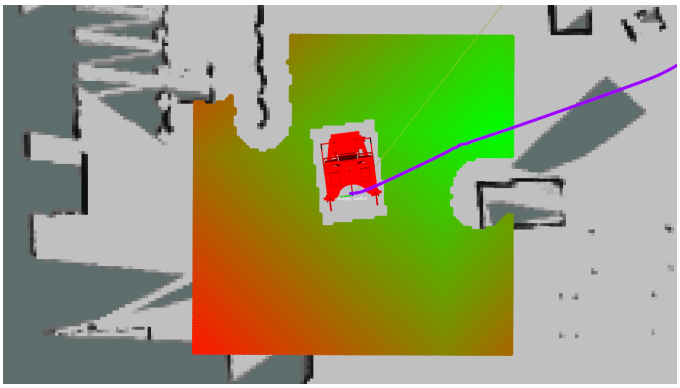


Figure 6.5: Example of the cost cloud published by the local planner. The map is square with a predefined size (5 meters) around the walker, showing low (green) and high (red) cost values, whose values are attributed by the parameterization of the local planner. The empty space in the square map are due to the presence of obstacles, so their cost is not marked.

6.6 Validation Protocols

6.6.1 LiDAR-SLAM

The goal of the described protocol in this section is to determine if the LiDAR and SLAM systems are creating quality environment maps and drawing coherent trajectories performed by the walker. The following validation protocol is based on the approach presented by Steder et al. [61].

If one is able to acquire a ground-truth map of the testing environment and deploy a static positioning system with low error to obtain the real position of the walker in the world then, both the quality of the map and position can be calculated by comparison between ground-truth and walker-registered data, after the reference frames from all systems are aligned. Acquiring accurate maps and deploying a good positioning system is not easy, for the following reasons: difficult access to positioning tools with low-error widely accepted by the scientific community [61]; scarce availability of maps containing the full description of the environment - usually the ones available are architectural building plans which do not include obstacles like furniture, minor wall alterations, etc. [61].

Instead of performing direct comparison between the outputs of the proposed and ground-truth systems, some metrics can be tracked that indirectly quantify the obtained results. The following case is presented in an attempt to explain the concept of indirect metrics. In each environment there are

metrics, *e.g.*, the distance between 3 distinct objects, that are exclusive to that environment. If the metrics used can be tracked by the walker and a simpler ground-truth system, and are specific enough to the environment's configuration, then the assumption that those metrics can be used as validation metrics of the LiDAR-SLAM system is reasonable [61].

The package OpenCV contains a module called *Aruco*[4] capable of tracking QR code markers (Figure 6.6) with video feed and calculate the relative position to the camera. If a video camera is implemented in the WALKit, and if multiple of these markers can be placed throughout each environment, then they can be used as environment specific and easily detectable features, whose euclidean distances between each other can be measured with a ruler, serving as a ground-truth metric.

Figure 6.6: QR code marker used in the SLAM validation process.

Following the concept of indirect metrics the following procedure was developed. Ten *Aruco* markers were dispersed throughout 2 different environments: the real environment similar to the corridor world in Gazebo (Figure 6.7), and a $100m^2$ atrium. To detect the *Aruco* markers, a video camera Orbbec Astra (Orbbec 3D Technology International Inc., United States) was placed on the walker. The euclidean distances between the markers were measured with a ruler (precision 0.05 m). The mean squared error between the ground-truth and *Aruco*-LiDAR-SLAM system measurements is presented as a validation metric for the mapping and localization component, with each environment being scanned 3 times. In Figure 6.7 an example of the placement of the markers and *Aruco* marker detection in the real corridor world.

[4]Source Code: https://docs.opencv.org/master/d5/dae/tutorial_aruco_detection.html

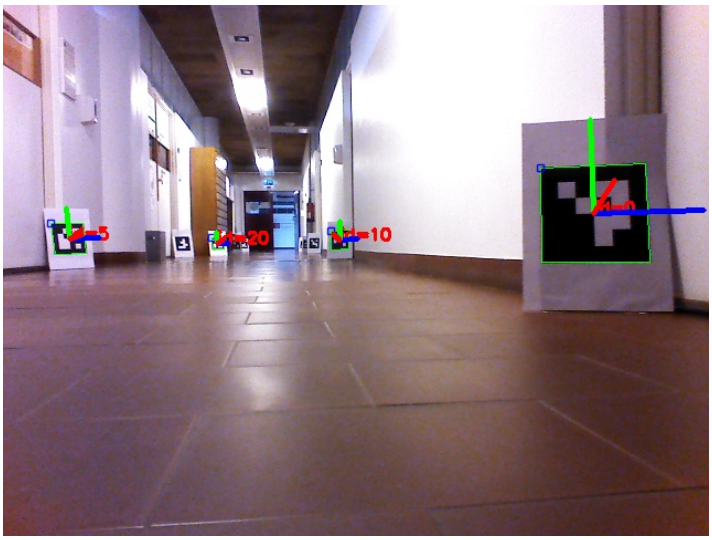


Figure 6.7: QR code marker detection by the *Aruco* module in the real corridor world.

While measuring the marker pair distance error, we can calculate the walker's positioning error, but nothing can be said about the quality of the map. However, for the LiDAR-SLAM to actually infer that the walker moved to a certain position and for the inference to be correct, then the created map must be accurate, due to the localization process needing an accurate depiction of the environment in the first place. So, when the SLAM algorithm outputs small positioning errors it does so by building a good map, otherwise it would not. This means that the evaluation of mapping and localization precision are one and the same. It should be noticed that this statement is only true if the results are accurate, but if not, the source of the error cannot be determined.

The *Aruco* marker detection algorithm showed errors of localization that scale with the true distance to the marker. For markers with distance beyond 5 m the error standard-deviation was 0.3 cm. For markers position at a distance of 3 below meters the error is reduced to 0.05. Because *Aruco* is supposed to be a low-error sensor, only marker position measurements with values below 3 cm were considered correct measurements.

6.6.2 Global-Local Planner

The validation of both Global and Local planners had a main focus on functionality, instead of how optimal the found solution is. This means that special emphasis was put on determining if the walker can actually avoid obstacles and reach the destination, instead of quantifying how efficient the established plans are, possibly serving as a safe option to replace the human supervisor.

Five scenarios were used to test the WALKit autonomous driving capabilities, those being shown in

Table 6.3. The autonomous supervisor testing scenarios were chosen based on the fact that these are similar to the rehabilitation scenarios with disabled patients [19]. Each of the scenarios was recorded with 3 trials. Each scenarios tries to represent different possible situations faced by the walker, namely: navigating in free-space, performing 180°turns, avoid static obstacles and avoid dynamic obstacles not previously scanned and included in the map created by the LiDAR-SLAM system.

Table 6.3: Global-Local Planner validation scenarios.

Scenario No.	Goal*	Obstacle Type	Description
0	5 m straight ahead	None	Testing unimpeded driving.
1	5 m straight back	None	Testing capability of $180°$ turns.
2	8 m to the right	$90°$ corner (static) to the right	Testing if walls can be avoided while turning to the right.
3	8 m to the left	$90°$ corner (static) to the left	Testing if walls can be avoided while turning to the left.
4	8 m straight ahead	Person walking (dynamic) perpendicular to the walker's trajectory	Testing if dynamic obstacles not present in original map can be avoided and if walker is able to reach the goal nevertheless.

* Goal with relation to the initial position of the walker.

6.6.2.1 Metrics

The performance of the autonomous supervisor (Global-Local Planner) is evaluated using 5 metrics suggested by literature.

Number of collisions is used to determine if the WALKit walker, with its autonomous planners, is capable of detecting and avoiding collisions, either static or dynamic (dependent on the scenario), and consequently come to a conclusion if the current controller is capable of replacing the human supervisor *w.r.t.* reducing the risk created by possible collisions.

Number of goals reached is calculated to evaluate if the hypothesized autonomous supervisor is capable of directing the patient-walker system towards the destinations, complying with the predefined gait rehabilitation scenario.

As already referenced in the solution requirements, it is unacceptable to register collisions, due to the fall-risk associated, however, it is acceptable if the autonomous supervisor fails to reach the destination in a minority of trials.

KTE values, already mentioned in chapter 3, are also measured to determine how close the walker's actual described path is to the path planned by the global planner A* [6]. If values are orders of magnitude below the length of the path, it is suggested that the local controller is following the global plan accurately.

Update Frequency of the local planner commands are evaluated to determine if the autonomous supervisor is capable of simulating a set of trajectories in time for safe driving scenarios. It was already established that the update rate should meet the 10 Hz mark, however, 4 Hz is the minimum safe threshold for the walker.

Case-by-case analysis is the most used technique for validation in literature due to the abstract nature of the task. Therefore, a trial from each scenario is presented, so that a comparison between scenarios can be made. Multiple snapshots dynamic obstacle scenario (scenario 4) are included to evaluate the behaviour of the walker along the scenario and its interactions with the obstacle.

Chapter 7

Shared Control

7.1 Introduction

When a medical expert is present in a rehabilitation scenario, he or she interferes with the system walker-patient when a certain situation is deemed dangerous. In this chapter, the prioritizer block mentioned in section 5.2 is detailedly described, block which is responsible for determining whether the control should be given to the patient or to the autonomous supervisor, given the environment's configuration.

In state-of-the-art approaches it is usually implied that the methods by which the SC is done, translate, in a way, what the medical expert is doing. This assumption is many times misleading because the controllers only express an approximation of what the expert is actually doing, *e.g.*, the medical assistant does not interfere when a threshold is met, instead it acts gradually. To avoid this, the controller should follow a more abstract notion of what the expert is actually doing. RL allows the walker, through trial and error, to learn how to capitalize on a highly-abstracted reward function. This allows the walker to not be built around biases about how the controller should behave. Instead it expresses the behaviour implied by the reward function built.

PPO trains both the actor and critic DNN following the steps in Figure 4.7. D'Eramo et al. [62] built a RL library, named *MushroomRL*[1] and the full implementation of PPO and the walker's environment was developed considering the library and ROS requirements. The *PyTorch*[2] library was used to implement the DNNs used in PPO. The full implementation was done using the programming language *Python 3.8*[3] due

[1] https://github.com/MushroomRL/mushroom-rl
[2] https://pytorch.org/
[3] https://www.python.org/

to the need for interfacing with *MushroomRL* (a *Python* package).

7.2 State and Action Space

Before any kind of controller can be built, the state and action spaces need to be defined so that the controller can create a policy. The state space needs to have a full description of the components needed to make a decision, so it was hypothesized that it should be composed of 5 elements: current position of the walker (Dimension: 2) current velocity of the walker (Dimension: 2), patient intention target velocity (Dimension: 2), autonomous target velocity (Dimension: 2), map and obstacle map (Dimension: size of the map) and goal.

To optimize the training time required to train a controller with computational resource limitations it is necessary to strict and optimize the state space to its limit. Theoretically, the current velocity of the walker is required to define the present state. However, if accelerations, either positive or negative, are not limiting, then the walker's velocity can change instantaneously, removing the need for the prioritizer to take into account the current velocity. This is not usually the case for most controllers due to the existence of a settling time. As already mentioned in the requirements, the controller will function at 10 Hz with and the maximum acceleration of $1\ m/s^2$, which means that the walker can transition from maximum velocity ($1\ m/s$) to a full-stop within 1 second. No significant changes in driving patterns were detected with the addition of the current velocity, due to the short time required to change velocity, so this data was eventually discarded from the state-space.

For the SC to actually share the control, it needs input from intention detection and autonomous supervisor modules, otherwise the agent does not know the choices that can be made, rendering a possible optimization impossible *w.r.t.* both the inputs mentioned.

Both map and obstacle map are the main problem to the dimensionality issue of the state space, because they both consist of matrices with a size up to the thousands or tens of thousands of entries - the map of 5x5 m with resolution 0.1 m produce a matrix of 2500 entries. Ideally, the SC should only give the priority to the autonomous mode if obstacles are nearby or if significant divergence from the goal is registered. To avoid obstacles the controller only needs local information but, to not diverge from the goal, some global information is theoretically needed. This can be circumvented by assuming that the global path is a good metric for goal convergence, and if the local portion of the global path is being followed then the walker is performing well. Then, if only local information is needed then global position becomes redundant. The cost cloud published by the local planner from Figure 6.5 allows the walker to infer all the

components mentioned before - goal and obstacles - within the same map.

So, instead of using a full state-space mentioned in the beginning of this section - current position and velocity, patient and supervisor intention, map and obstacle map and goal - only the target velocities and cost map are needed for the state description, resulting in a vector like,

$$s = (v_{auto}, w_{auto}, v_{int}, w_{int}, costcloud_0, ...costcloud_N) \tag{7.1}$$

where v_{auto} and w_{auto} are the linear and angular velocities of the autonomous mode, v_{int} and w_{int} are the linear and angular velocities of the intention mode, and $costcloud_i$ is the cost of point i.

In terms of action space, the actions need to express the partial contributions of the autonomous and intention modes. A one dimensional vector is used as action

$$a = (\alpha) \tag{7.2}$$

where α is the admittance value. The admittance value is then transformed into velocity commands by the expression

$$(v_{shared}, w_{shared}) = \alpha(v_{auto}, w_{auto}) \times (1 - \alpha)(v_{int}, w_{int}) \tag{7.3}$$

where v_{shared} and w_{shared} are the linear and angular velocities calculated by the SC. α is continuous variable that can assume values in the $[0,1]$ interval, turning the walker completely autonomous when α is closer to 1 and completely drivable by the patient when closer to 0.

7.3 Reward

The reward chosen for the training needs to translate 3 components: the risk of falls associated to collisions with obstacles, the necessity of giving control (following the assist-as-needed concept) to the patient and converging to the goal.

As already defined in the action space, the admittance value α regulates the contribution of both autonomous and intention modes. A simple reward function is

$$r(s,a) = (1 - \alpha) \tag{7.4}$$

, where the reward motivates the autonomous supervisor to produce low α, turning the walker less autonomous and rewarding being driven by the patient. Even though this reward expresses the requirement

of following the patient's intention, it does not necessarily translates the desire to stay away from obstacles. The obstacle penalty is expressed when the walker collides and the episode ends, stopping the agent from accumulating more reward. The penalty received by colliding ($r = 0$), at this moment, is the same as following the autonomous mode when close to obstacles ($\alpha = 1 \Rightarrow r = 0$), which is problematic due to the walker not "feeling" motivated enough to engage the autonomous mode. However, with a reward of -1 when the walker collides with obstacles, the SC is motivated to behave autonomously (0 reward) when close to obstacles.

Because we want the walker to actually finish the task, if no reward is given at the end of the episode, the walker will always give priority to not end the task and keep earning reward from the prioritization. The reward given at the end cannot be 1 because there is not enough incentive, but a reward like 10 is sufficient.

Assembling every component produces an expression like

$$r(s,a) = \begin{cases} -1 & \text{if } s \in \text{collision} \\ 10 & \text{if } s \in \text{goal} \\ 1 - \alpha & \text{otherwise} \end{cases} \tag{7.5}$$

. With some study of the Equation 7.5, it can be stated that to actually learn when to prioritize the autonomous mode while avoiding obstacles, the walker will have to crash into the obstacles and get a penalization. So, when rolling out N episodes with PPO, the walker will only use the last sample (representing the collision) of each episode to optimize itself in terms of obstacle avoidance. The scarce number of samples *w.r.t.* obstacle avoidance will translate into a very slow learning of this behaviour. To overcome this, a heuristic was created to express more gradually the presence of obstacles in the reward function with the expression

$$\begin{aligned} dist_{factor} &= \max\left(\min\left(2.5 \min\left(obst_{distances}\right) - 1.5, 1\right), 0\right) \\ r(s,a) &= \begin{cases} -1 & \text{if } s \in \text{collision} \\ 10 & \text{if } s \in \text{goal} \\ -\left(dist_{factor} + \alpha - 1\right) & \text{otherwise} \end{cases} \end{aligned} \tag{7.6}$$

where $obst_{distances}$ is the set of distances from obstacles acquired by the LiDAR. With the help of Figure 7.1, one can see that when the distance between the walker and an obstacle is inferior than 0.6 it should choose an α closer to 1 (autonomous) and vice-versa. A region of transition is seen in the middle and the steepness of this portion of the surface determines how smoothly the walker transitions between intention

and autonomous modes. This reward simply represents the danger faced by the walker and the patient when the full system is at a distance at least 0.6 meters.

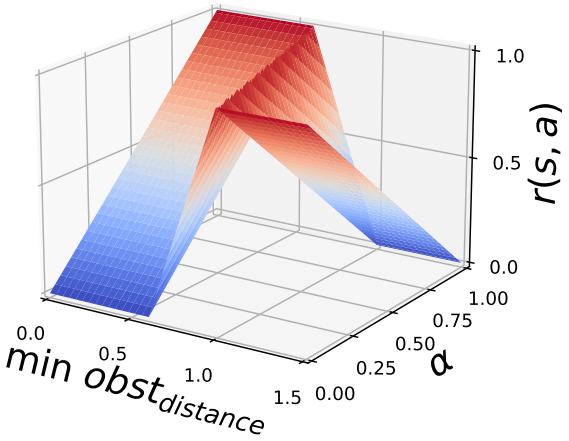


Figure 7.1: Reward surface created by the reward function, in Equation 7.6. Red - > 0.5 reward, Blue - < 0.5 reward.

Simple reward functions like Equation 7.5 should always be used, because they include the minimal amount of human behaviour biases, but the trade-off between simplicity and computational requirements must be always considered when building an intelligent agent, as already mentioned in section 4.1.1.

7.4 Discount Rate

When training, the policy needs the discount rate γ to track the reward on the long and short-term. This time "window" needs to be big enough in order for the agent to be able to do a correct assessment and stop in time to not collide with an obstacle. Because of that, γ is highly dependent on the walker's driving properties - maximum accelerations and velocities.

When driving the WALKit walker, it is always assumed that other dynamic obstacles do not try to purposely crash with the walker. This assumption is fair considering that the walker is built to work in hospital and environments where the patient's well being is a priority. Still, there are no guarantees that an unknown entity will not suddenly appear in the walker's range and crash with it. This is a possible situation but unavoidable, because the sensors do not detect such a range necessary and actuators cannot make sudden movements so that the patient does not fall. Because the SLAM system works at 10 Hz and the range of the LiDAR system is 5 m, the sensor is able to detect one sample of any movable obstacle with

a velocity of 50 m/s (180 km/h), value which is well above the normal speed of any movable entity in day-to-day activities.

The only restriction for γ is the effective maximum acceleration of walker, *i.e.*, taking into account wheel sliding and break inefficiency.

The discount factor was established in the time domain, instead of time-step domain, allowing for the automatic change of γ when the time-step is changed. A γ_{time} (discount factor in the time domain) of 0.3 was chosen which expresses a behaviour that discards a future beyond the $2-3$ seconds from the decision making process at every step, as shown in Figure 7.2, allowing the walker to ponder the necessity to break, accelerate or even change trajectory within that period. The controller will function at 10 Hz, therefore, the actual γ is $\gamma = \gamma_{time}^{\frac{1}{f}} = 0.3^{\frac{1}{10}} \simeq 0.8865$.

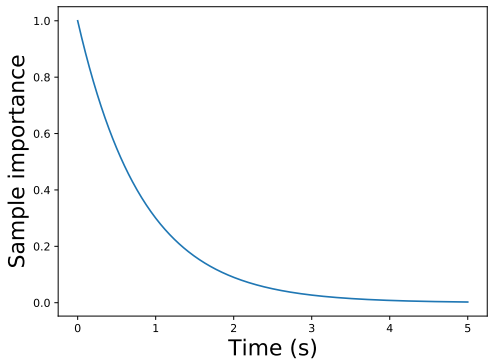


Figure 7.2: Sample importance decay with γ equal to 0.3.

7.5 Policy and State-Value Networks

As seen in the PPO section 4.3 of this dissertation, two DNNs are needed to work as function approximators. Because PPO only estimates, in terms of value functions, the value of each state, then the input of both actor and critic network is the state-vector. This means that the structure of the networks is practically the same except for the outputs - the actor outputs the action to perform on the environment and the critic output the value of the state. Because both the outputs are of dimension 1, the structure of the networks is coincidentally similar.

The network used, with a representation in Figure 7.3, can be separated into two parts: the cost map feature detection and shared-command regression.

The cost map is fed into 3 consecutive convolutional layers that are responsible for extracting the relevant information from the map, constituting the part of cost map feature detection. The output of each convolutional layer is subjected to a ReLU activation function and produces 10 channels, like suggested

by Li and Yuan [63]. The first convolutional layer uses a kernel of size 7 and stride 2, and the second and third used a kernel size of 10 and stride 1.

Three dense layers compose the shared-command regression and are responsible for learning the regression required for the prioritization process. The first dense layer connects to the flattened output from the convolutional layers and to the velocity commands of both intention and autonomous mode, leaving the other two to connect sequentially with sizes 256.

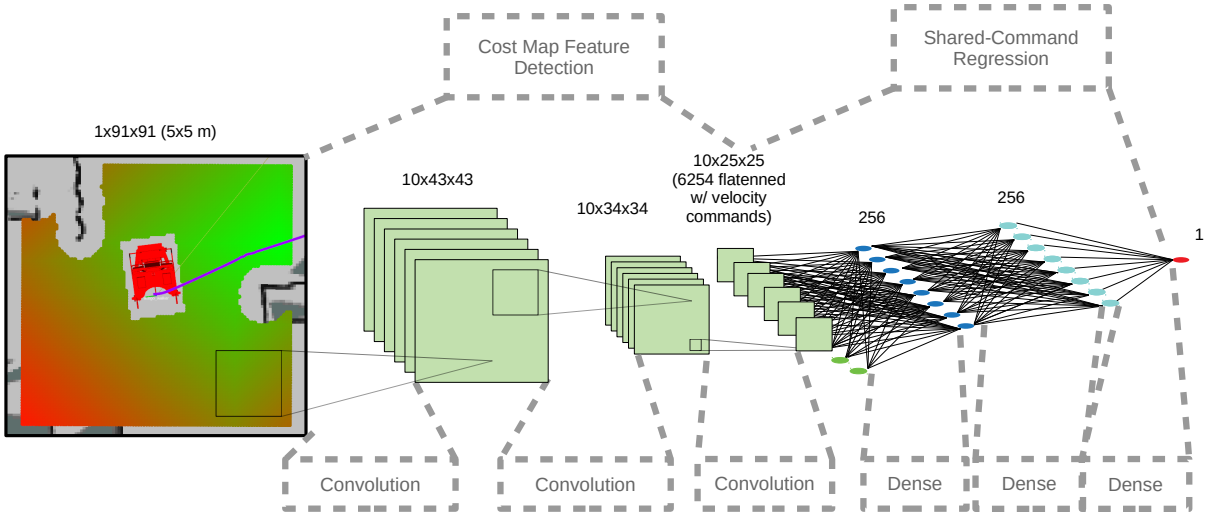


Figure 7.3: A representation of the DNNs used as policy and value estimators. The number of entries (inputs/outputs) in this figure is not representative of the actual number present in the network.

7.6 Policy Structure

PPO is an on-policy method, so, the policy that is used for sampling trajectories is the same on which optimization is run on. So the policy must be capable of achieving optimal behavior, but still be able to explore the possible actions. Because DNNs model are deterministic, this means that a certain policy will always execute the same actions given a certain state, so no exploration is done. Also, PPO optimizes stochastic policies, so the network must be wrapped in a way that confers stochasticity to the policy and allows it to explore.

The *MushroomRL* library allows the network to be wrapped by a Gaussian Policy. This Gaussian Policy adds to the output of the network a random value drawn from a Gaussian distribution with a standard-deviation σ_π and mean 0, like the expression

$$a(s) = \pi(s) = NN(s) + GD(\sigma_\pi) \tag{7.7}$$

where $NN(s)$ is the neural network function GD is the Gaussian distribution sampler. To initiate the policy, a initial standard-deviation $\sigma_{\pi_{initial}}$ of 0.4 for the Gaussian noise was chosen. The initial noise needs to be high enough for decent exploration of the action and state space, but also small enough so the actions performed are not random. If σ_π remains always constant it cannot be expected for the policy to converge to a decent policy, so the parameter σ_π is included with the network parameters in the optimization process allowing it to change noise intensities according to the loss function.

7.7 Data Normalization

LeCun et al. [64] state that "convergence is usually faster if the average of each input variable over the training set is close to zero (...) but also if they are scaled so that all have about the same covariance". All network inputs (states) were normalized using the domain limits, according to Table 7.1.

Table 7.1: State domain limits and standardization parameters.

State	Minimum	Maximum	M_{norm}	Δ_{norm}
Cost map	0	100	50	100
$v_{auto}, v_{intention}$	0	1	0.5	1
$w_{auto}, w_{intention}$	-0.39	0.39	0	0.78

The normalization of the states is done as follows,

$$\bar{s} = \frac{s - M_{norm}}{\Delta_{norm}} \tag{7.8}$$

where M_{norm} is the vector of the domain intermediate values and Δ_{norm} the domain range.

The inverse process of the normalization of the states should be applied to the actions so that they are scaled adequately. In the section 7.2, it was mentioned that the parameter α is in the interval $[0, 1]$, however, the outputs of the policy are within the range of $]-\infty, \infty[$. The logistic function, shown in Figure 7.4, scales the unbounded outputs from the network to the α domain, with the equation

$$l(x) = \frac{1}{1 + e^{-x}} \tag{7.9}$$

, where l is the logistic function and x is the unscaled output of the policy.

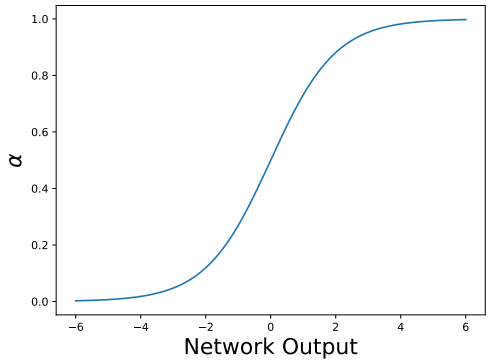


Figure 7.4: Logistic Function used in the adjustment of network outputs to environment actions.

7.8 Training Pipeline

To train the controller both the *Gazebo* environments (Corridor and Hospital) were used, and no training in real environments was done, leaving only testing for the real environments.

Because DRL , in most cases, needs millions of samples to converge to a decent policy, training the controller in real-time becomes very limiting due to the constant need of human feedback in the intention component. To avoid this, a simulator of human intention was developed. The intention simulator cannot output just random velocity commands because most humans drive the walker predominantly well, otherwise the actions would not be representative of the actual human distribution. So, the human intention simulator copies 70% of the time the output of the autonomous commands and 30% left outputs random commands. The proportion of each source of intention commands were chosen based on trial and error, and also because they offer diverse enough behaviour to train.

The process begins by placing the walker in the initial position and picking a goal from a pool choice of size 50, which was found to be sufficient number of destinations to force the walker to experience sufficient obstacles configurations. All destinations should be reached within 30 seconds but each episode is limited to 90 seconds just in case the walker gets stuck.

PPO evaluates the policy based on 2048 samples (3 min and 24 s), which is a decent size of samples to create a good estimation of the walker's behaviour. The Advantage (Equation 4.16) of each state-action pair is compared to the state-value estimation using Generalized Advantage Estimation [65] the λ factor with value 0.95 and the γ of 0.8865. The new state-value estimates are calculated using the old estimation of the state-value function from the critic network and the new reward is obtained from the trajectories.

In the update step of PPO, the 2048 samples (now with also the advantages) are randomly separated into mini-batches of 256 samples and policy is updated by calculating the mean score *w.r.t.* the loss

functions in Equations 4.15 and 4.18 with ϵ as 0.95 , a as 0 and b as 0.01. The value error component L_v was not used ($a = 0$) because *MushroomRL* library does not support parameter sharing. Each mini-batch is used to update the policy (actor network) running the stochastic gradient descent algorithm, repeating the same process 4 times so that the process is accelerated. The critic network is also updated by fitting the samples 10 times with the using the new value estimates, but this time using backpropagation with the mean squared error between new and old state-value estimates.

The parameters from both networks are optimized using Adam optimizers [54], with learning rate per update of $1e-4$.

7.9 Validation Protocols

In this validation protocol we want to test the ability for the walker to prioritize the autonomous supervisor when a danger is faced while also maximizing the contributions of the patient to the control of the WALKit.

To validate the SC, the same scenarios from the autonomous mode described in Table 6.3 were used. In this validation protocol, because the SC is being put to the test, it is important that the velocity commands from the patient are adequate for the SC to prioritize the actions from the patient and vice-versa. Due to the impossibility of testing the SC with gait-disabled patients, the healthy drivers were told to try to mimic possible actions from the patients, meaning that they should try to crash with obstacles, and try to increase the distance to the goal allowing to simulate dangerous inputs, but also to drive adequately to simulate correct inputs. A precise protocol of driving was not forced upon the drivers to simulate the unpredictability of the patients in this context. Each scenario was performed 3 times by 3 different users.

7.9.1 Metrics

All the metrics evaluated in the Global-Local Planner section of the autonomous supervisor are also included in the validation of the SC, with the addition of one other specific to the SC's task - the patient relative control time.

The **patient relative control time** is the percentage of time that the patient controls the walker when compared to the total amount of time in the trial, which is represented by the following equation

$$\text{patient relative control time} = 100\frac{1}{N}\sum_{n=0}^{N}\alpha \tag{7.10}$$

To validate the training of the SC, the same was performed 10 times to guarantee uniformity in the results. At the end of the training procedure, when the walker is performing at theoretical maximum performance, the **discounted reward**, referenced in Equation 4.5, should be

$$\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} = \frac{1}{1-\gamma} = \frac{1}{1-0.883} = 8.54 \tag{7.11}$$

due to the convergent nature of the discounted reward series [27].

Chapter 8

Results

The current chapter presents the results obtained from all validation procedures mentioned in chapters 6 and 7, regarding the Mapping and Localization system (LiDAR-SLAM), autonomous supervisor (Global-Local Planner) and prioritizer (Shared-Control).

8.1 Autonomous Driving

The results from the current section regard only the task of the autonomous supervisor with its mapping, localization, planning and control capabilities.

8.1.1 LiDAR-SLAM

For each of the validation environment (atrium and corridor) mentioned in section 6.6.1, the QR code markers were positioned like referenced in the validation protocol. The following maps will be specific to the corridor world, but the metric values regard both environments. The euclidean distances between the QR code markers positioned in the corridor world, used as ground-truth measurements, are shown in Table 8.1. The distance between QR code markers were calculated using the *Aruco*-LiDAR-SLAM system, and the mean squared error in relation to the ground-truth measurements is show in Figure 8.2, through the respective box plot. With the presented errors of the LiDAR-SLAM system, it is possible acquiring maps similar to the shown in Figure 8.1.

Table 8.1: *Aruco* marker ground-truth euclidean distances in the corridor environment.

Marker IDs	Euclidean Distance (m)
0 - 5	3.70 ± 0.05
5 - 10	3.05 ± 0.05
10 - 15	3.05 ± 0.05
15 - 20	1.22 ± 0.05
20 - 25	2.50 ± 0.05
25 - 30	2.73 ± 0.05
30 - 35	1.90 ± 0.05
35 - 40	2.37 ± 0.05
40 - 45	2.65 ± 0.05

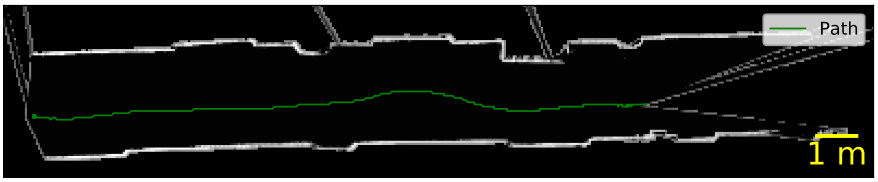


Figure 8.1: An example of the map obtained and the trajectory(green path) in one trial from the corridor environment.

Regarding the spatial errors, the box plot from Figure 8.2 shows that all 100th percentiles are below 0.21 m and a maximum outlier of 0.38 m, which represents errors of 58% and 25% for the mapping and localization process below the set requirement (0.5 m) for the WALKIt walker, respectively.

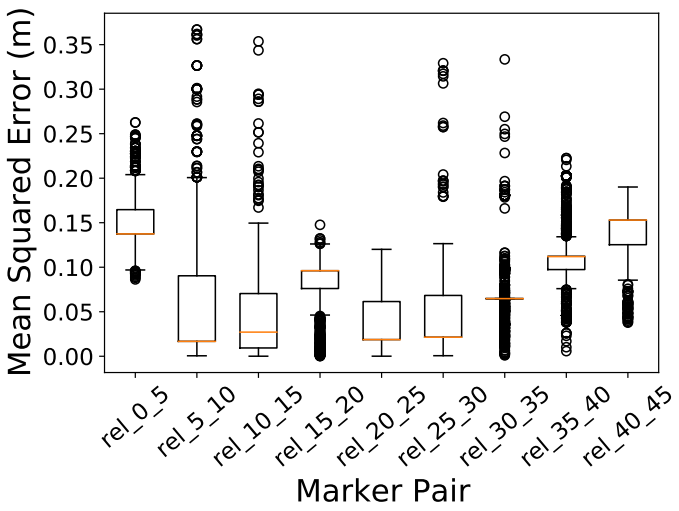


Figure 8.2: Box plot of the mean squared error between the distances of the *Aruco* markers measured by the LiDAR-SLAM system and the ground-truth. rel_x_y represents the error relative to the relation between marker x and y, from Table 8.1.

Considering the timing performance, the *Cartographer* SLAM system was able to update the map and position with the frequencies shown in the box plot from Figure 8.3. Results from Figure 8.3 show a 0th, 25th, 75th and 100th percentiles at 7 Hz, 8.9 Hz, 10 Hz and 10 Hz respectively, suggesting that the update rate of map and localization is below the established requirement for the full controller of 10 Hz. Also, the worst case outlier was registered to be located at 0.5 Hz, however, it should be referenced that these delays in computation were detected when the SLAM system worked with full SC active with all CPU cores of the WALKit's computer being used at maximum capacity and while scanning the environment for the first time.

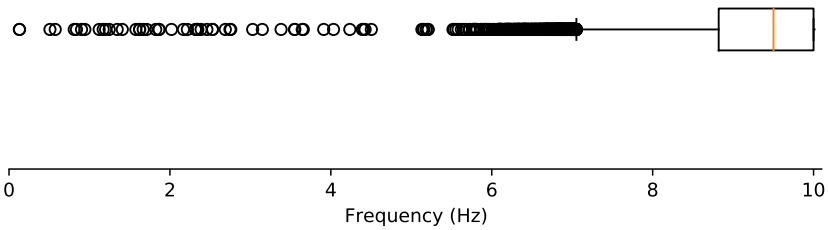


Figure 8.3: Box plot of the map/pose publishing frequencies along the scenarios.

Figure 8.4 shows a specific map acquired by the developed framework, evidencing the obstacles present in the environment (walls and furniture) and some unexpected artifacts. The noise from Figure 8.4

is included in the map due to rare wrong measurements of the LiDAR sensor or calculations by the SLAM system. While Cartographer maps the environment and converges to a solution close to ground-truth, the sections with inaccurate mapping are neither deleted nor remapped, originating maps with the mentioned noisy artifacts.

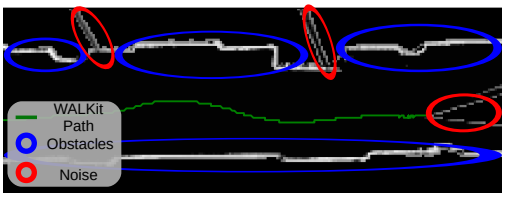


Figure 8.4: Example of uncertainty of interpretation of the created map.

8.1.2 Global-Local Planner

To test how the Global Planner created ideal paths and how the Local Planner controlled the walker's velocity, the scenarios explained in the validation procedures from section 6.6.2 were followed. In Table 8.2 the amount of collisions and goals reached is presented. No collision were detected and the WALKit was always able to converge to the destination.

Table 8.2: Number of collisions and goals reached in each scenario of validation of the global and local planners.

Scenario No.	No. of Goals Reached	No. of Collisions
0	3 (100 %)	0 (0 %)
1	3 (100 %)	0 (0 %)
2	3 (100 %)	0 (0 %)
3	3 (100 %)	0 (0 %)
4	3 (100 %)	0 (0 %)

Also, the KTE values were measured to compare the difference between the actual and planned paths from the autonomous mode, shown in Table 8.3, registering an average over all scenarios and trials of 0.3101 m. Considering the length of the scenario paths performed ($5 - 8$ m), the registered KTE values are about at least 16 times below the length of the overall path ($5/0.3101 = 16.12$), which respect the set requirement that the walker path deviations should be orders of magnitude smaller than the overall path. The values registered in scenario 1 (180°turn) were higher (average 0.9383 m) when comparing to the rest of the scenarios, even when comparing to scenario 4 where the walker tries to avoid dynamic

obstacles (average 0.2328 m), which was expected to produce the highest KTEs caused by the deviations forced by the obstacle.

Figure 8.5 presents one example of path rolled out by the walker in each scenario, so that a case-by-case analysis can be performed. In the scenarios where the walker had to only move forward or make slight deviations - scenarios 0, 2 and 3 (Figures 8.5a, 8.5d, 8.5c) - the walker followed the path planner accurately as also evidenced by the lowest KTE values. Regarding scenario of turning 180° (Figure 8.5b), the walker started by performing a pure turn (no linear velocity), but after $1 \sim 2$ seconds it started performing a wide curve. The expected behaviour would make the walker turn purely until fully aligned with the path, followed by traveling forward until reaching the destination. This unexpected wide curve causes a significant deviation from the initial path, as shown in Figure 8.5b, producing exceptionally high KTEs.

Table 8.3: KTE values registered in the ADS validation scenarios. Units: meters.

Scenario No.	Trial 0	Trial 1	Trial 2	Average	Standard Deviation
0	0.0636	0.3000	0.1973	0.1870	0.1185
1	1.0401	0.4116	0.8856	0.7791	0.3275
2	0.2158	0.0598	0.2251	0.1669	0.0929
3	0.0532	0.2150	0.2192	0.1625	0.0947
4	0.2813	0.3857	0.0976	0.2549	0.1459

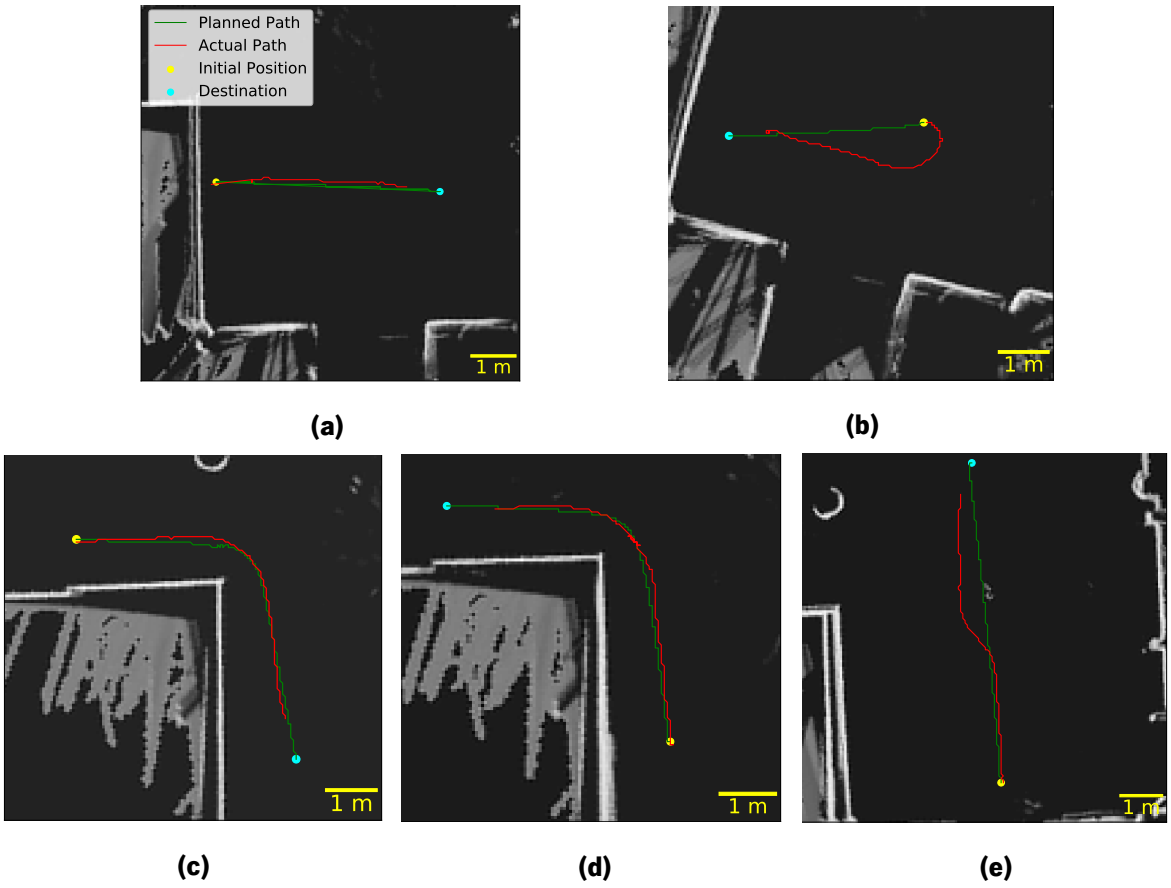


Figure 8.5: Walker paths performed in the benchmarking scenarios of the Global-Local planner system. An example of each scenario is show: driving straight ahead ((a) - Scenario 0), turning $180°$((b) - Scenario 1), right corner ((c) - Scenario 2), left corner ((d) - Scenario 3), and presence of a dynamic obstacle ((e) - Scenario 4))

In the scenario 4, where dynamical obstacles are present, a temporal analysis is required to undestand the decision making process of the ADS. Figure 8.6 offers multiple snapshots of the walker and environment along the scenario 4, also showing the plots of distance to the closest obstacle and velocity commands. The walker initiates the scenario by detecting some obstacles behind, which do not impede travel (Figure 8.6a). After it initiates movement, the walker starts: detecting an obstacle to the right (Figure 8.6b), reducing its linear velocity and changing the angular velocity, as the distance to the obstacle decreases (7.5 seconds Figure 8.6e). After deviating course (Figure 8.6c), the WALKit heads once again to the destination finishing its trajectory (Figure 8.6d). The velocity of the human interacting with the walker as an obstacle was not constant in all trials, because it was deemed important to test the autonomous controller in environments where humans walk without restrictions. So, in certain trials the walker performed like Figure 8.5, but when human-obstacles moved at higher velocity, the walker was forced, in one case, to stop and wait for the obstacle to move. From Figure 8.6 we can detect that, as expected, the walker stops and changes course by reducing its linear velocity - trying to avoid collision - and raising the absolute value of the

angular velocity - trying to head towards alternative directions with no obstacles - when obstacles become too close. In scenario 4, despite the walker deviating from the original path due to the presence of the dynamic obstacle, it still arrived at the destination. The KTE values of this scenario 4 were not as high as in scenario 1 because in the first half of the episode the path is followed accurately, only increasing the values once the obstacle forces the walker to deviate.

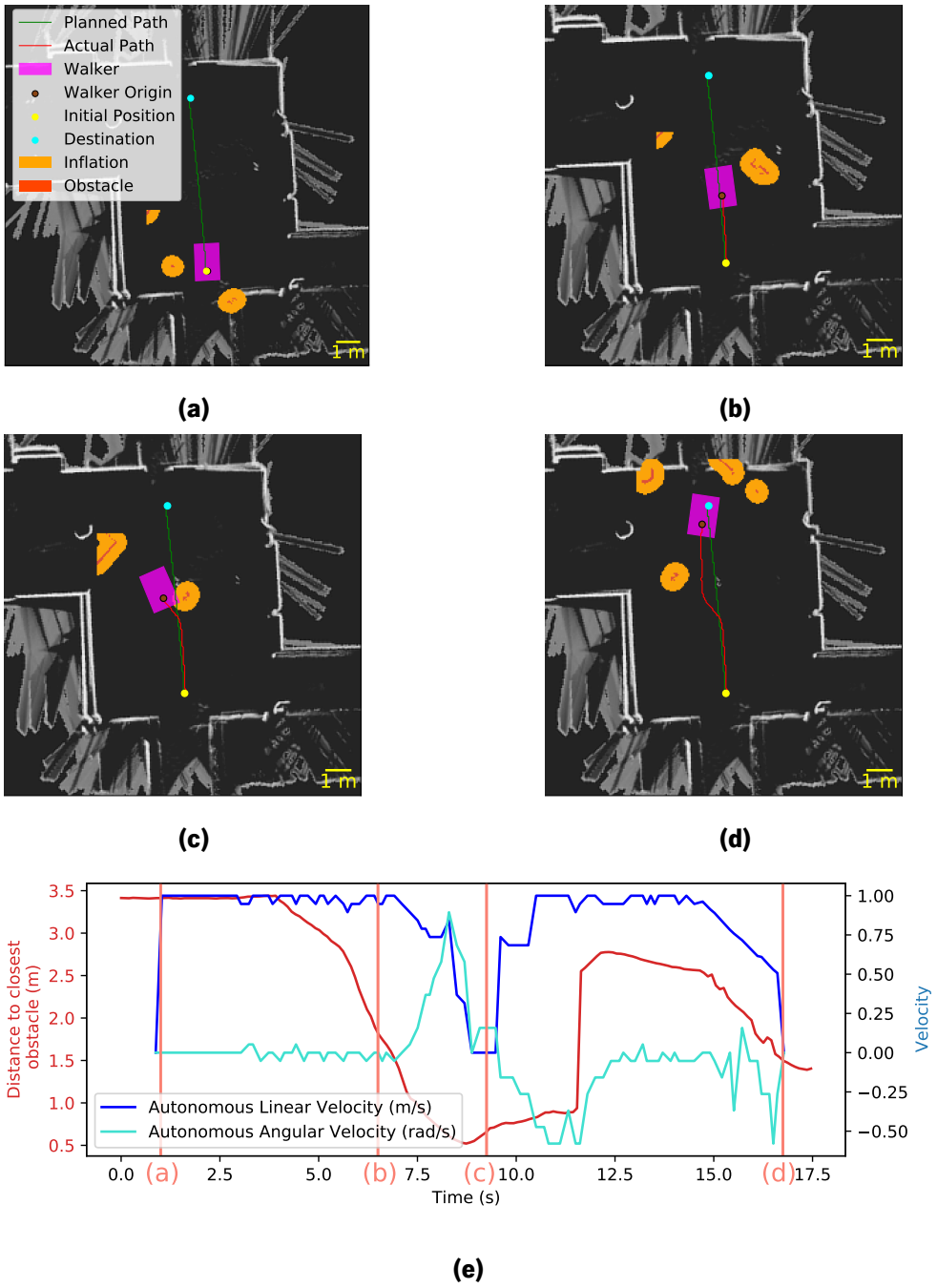


Figure 8.6: Environment and walker while benchmarking with scenario with dynamical obstacles. The walker starts by initiating its trajectory (a), after a while detects an obstacle (b), avoids collision (c), and arrives at the destination (d). A plot of the linear and angular velocities and distance to the closest obstacle (e).

A analysis of the update rate of the autonomous commands was done to determine how fast the local planner is able to plan, and the results are show in Figure 8.7. All the frequency quartiles are located closely to the 10 Hz mark, which respects the established requirement of 10 Hz.

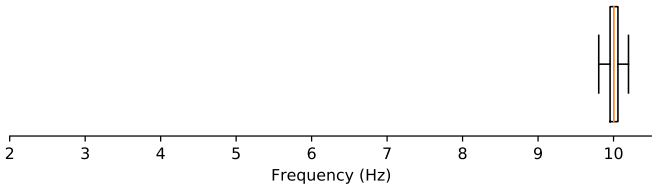


Figure 8.7: Box plot representing the rate at which the local planner *Dynamic Window Approach* was able to calculate each velocity command over all scenarios and trials.

Some testing was done in environments which are unlikely to be used in gait rehabilitation scenarios. In the scenario from Figure 8.8, the walker reached a position where it had to go the opposite way it was headed, but surrounded by close obstacles (different from the tested scenario 1). The walker started to turn continuously (Figure 8.8a), but at a certain moment started slowing down (Figure 8.8b), until reaching a distance of 10 cm from the wall to the chassis (Figure 8.8c). The behaviour performed was not adequate to reach the destination resulting in the robot not being able to figure out how to solve the situation.

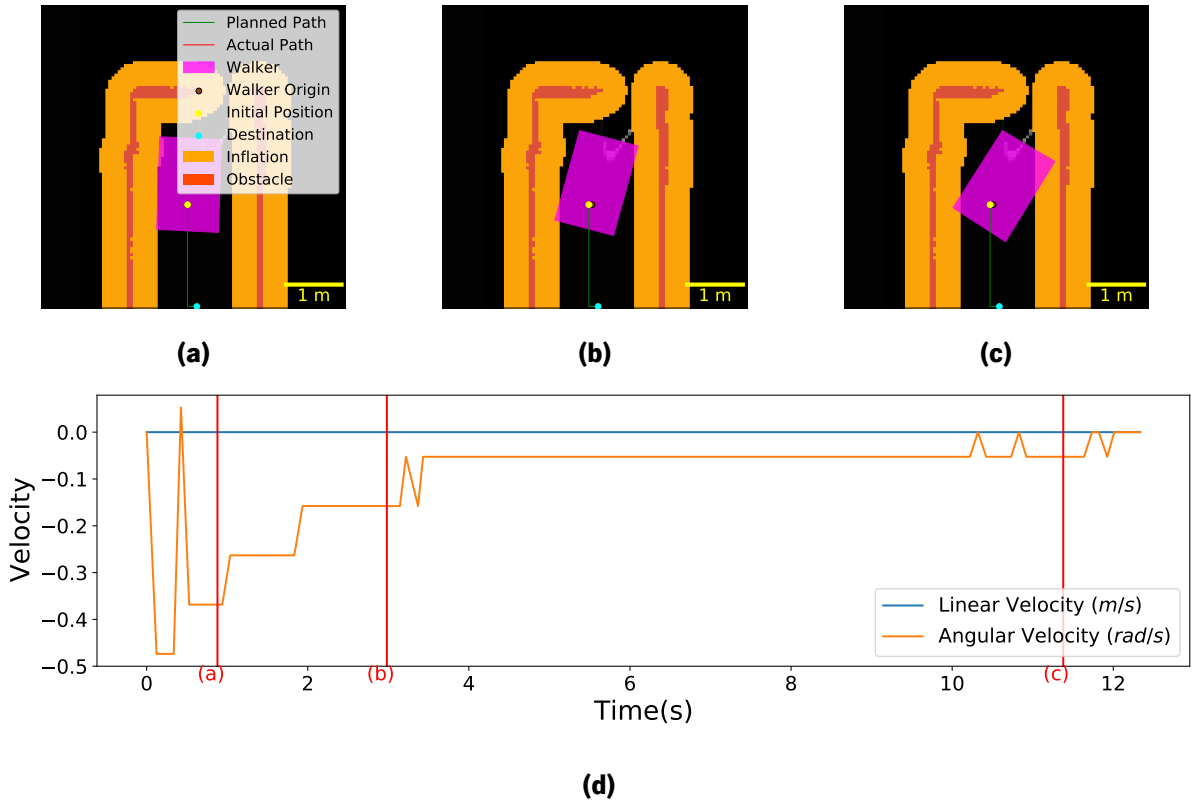


Figure 8.8: Walker unable to autonomously turn $180°$ when close obstacles are present. The walker initiates the turning (a), approximating continuously its chassis to the obstacle (b) and finally being unable to continue (c). A plot of the linear and angular velocities (d) through the episode

.

8.2 Shared-Control

The current section presents the results related to the prioritizer and its ability to keep the patient safe while following patient's intention.

8.2.1 Controller Behaviour

The scenarios in the SC validation protocols were performed, as described in section 3.4, with the amount of collisions and goals reached registered in Table 8.4. No collisions were detected, however, the destination was not reached in all trials - despite not crashing, in one trial in scenario 2 the walker did not reach the destination.

Table 8.4: Number of collisions and goals reached in each scenario of validation of the SC.

Scenario No.	No. of Goals Reached	No. of Collisions
0	9 (100 %)	0 (0 %)
1	9 (100 %)	0 (0 %)
2	8 (88.89 %)	0 (0 %)
3	9 (100 %)	0 (0 %)
4	9 (100 %)	0 (0 %)

The KTE values and relative patient control time registered in the validation scenarios are shown in Table 8.5 and Table 8.6, respectively. The values from Table 8.6, exposing the relative control time of the patient, suggest that the patient is able to control on average 70.72% of the time, resulting in KTE values around 0.6343 meters on average over all trials and scenarios performed.

Similar to the validation of the Global-Local Planner, a trial of each scenario was included to present the trajectories described by the SC (Figure 8.9), so that a case-by-case analysis is presented to study the correlation between α and the path of the walker.

Table 8.5: KTE values registered in the SC validation scenarios. Units: meters.

Scenario No.	Trial 0	Trial 1	Trial 2	Average	Standard Deviation
0	0.4216	0.1912	0.8249	0.4792	0.3208
1	1.1832	1.3266	0.4495	0.9864	0.4705
2	1.2495	0.3845	0.9090	0.8477	0.4357
3	0.5664	0.7854	0.7155	0.6891	0.1119
4	0.1840	0.2600	0.0632	0.1690	0.0992

Table 8.6: Relative patient control time values registered in the SC validation scenarios. Units: Percentage.

Scenario No.	Trial 0	Trial 1	Trial 2	Average	Standard Deviation
0	84.65	91.11	79.04	84.93	6.04
1	70.43	71.08	78.31	73.27	4.37
2	63.98	62.91	76.10	67.66	7.32
3	70.43	55.18	62.53	62.71	7.63
4	70.78	56.88	67.52	65.06	7.26

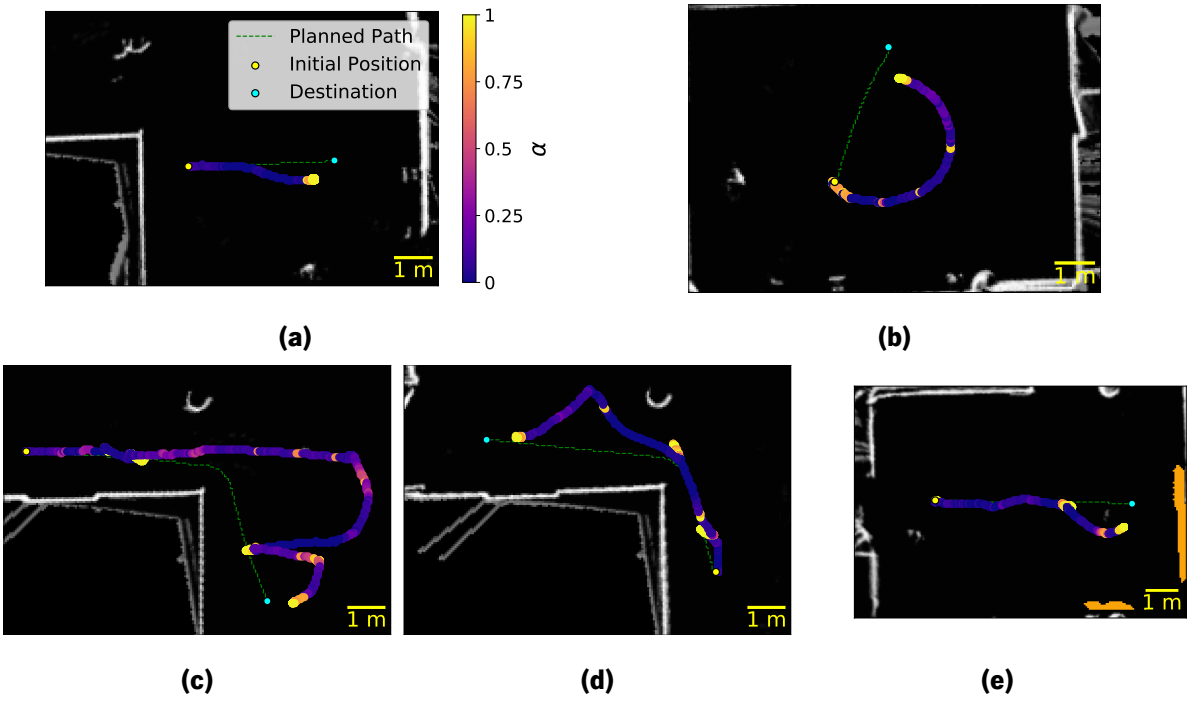


Figure 8.9: Walker paths performed in the benchmarking scenarios of the Shared-Controller system. An example of each scenario is show: driving straight ahead ((a) - Scenario 0), turning $180°$((b) - Scenario 1), right corner ((c) - Scenario 2), left corner ((d) - Scenario 3) and presence of a dynamic obstacle ((e) - Scenario 4).

In scenario 0 from Figure 8.9a where no obstacles are present and the destination is 5 meter straight in front of the walker, the relative control time was the highest over all scenarios, registering uniformly low α values ($0 \sim 0.25$).

In scenario 1 from Figure 8.9b where no obstacles are present and the destination is 5 meters straight behind the walker, the patient control time is the second highest and the KTE values are also exceptionally high, as registered in the Global-Local Planner results. Opposite to all other scenarios, in scenario 1 the α values are exceptionally high within the first meters of the scenario, around 0.75.

In Figures 8.9c, 8.9d and 8.9e, a trial from scenario 2, 3 and 4, respectively, present portions of the path whose α values are superior to the overall path (> 0.5). These "spikes" in α are registered when the

walker's distance to an obstacle is smaller than 1 meter. These spikes are going to be detailedly analyzed when the scenario 4 is examined temporally. In scenario 2 from Figure 8.9c, the value of α also increases gradually, surpassing the 0.5 threshold as the walker goes beyond the 2 meters of distance relative to the global path, and also gradually reducing as the walker gets closer to the path. Also, the scenarios 2, 3 and 4, which represent the scenarios where obstacles are present, possess the lowest relative patient control time, shown in Table 8.6.

In every trial from every scenario (Figure 8.9), the α consistently rise gradually from any value to 1 when the walker is located at less than a meter from the destination.

Similar to the Global-Local planner results section 8.1.2, the scenario 4 (presence of a dynamic obstacle) was analyzed with detail so that each decision along the scenario could be analyzed. In the context of the SC the mentioned detailed analysis is included in Figure 8.10, so that a future comparison can be made between autonomous and shared modes. When the walker starts its trajectory (Figure 8.10a) with no obstacles present, the α value is 0, allowing the patient to express his intention. When the walker-patient system finds an obstacle, despite the command from the patient inciting a collision, the SC prioritized the autonomous control ($\alpha = 1$) which was informing the walker to stop, avoiding the obstacle (Figure 8.10b). Still, the patient's commands kept trying to collide, even when the obstacle was moving, but the prioritizer produced α equal to 1, keeping the control on the autonomous supervisor (Figure 8.10c). The patient and autonomous supervisor suggest valid commands after the dynamical obstacle moves, clearing the path between walker and the destination. As expected, α drops to 0, returning the control to the patient (Figure 8.10d) and finally finishing the scenario (Figure 8.10e).

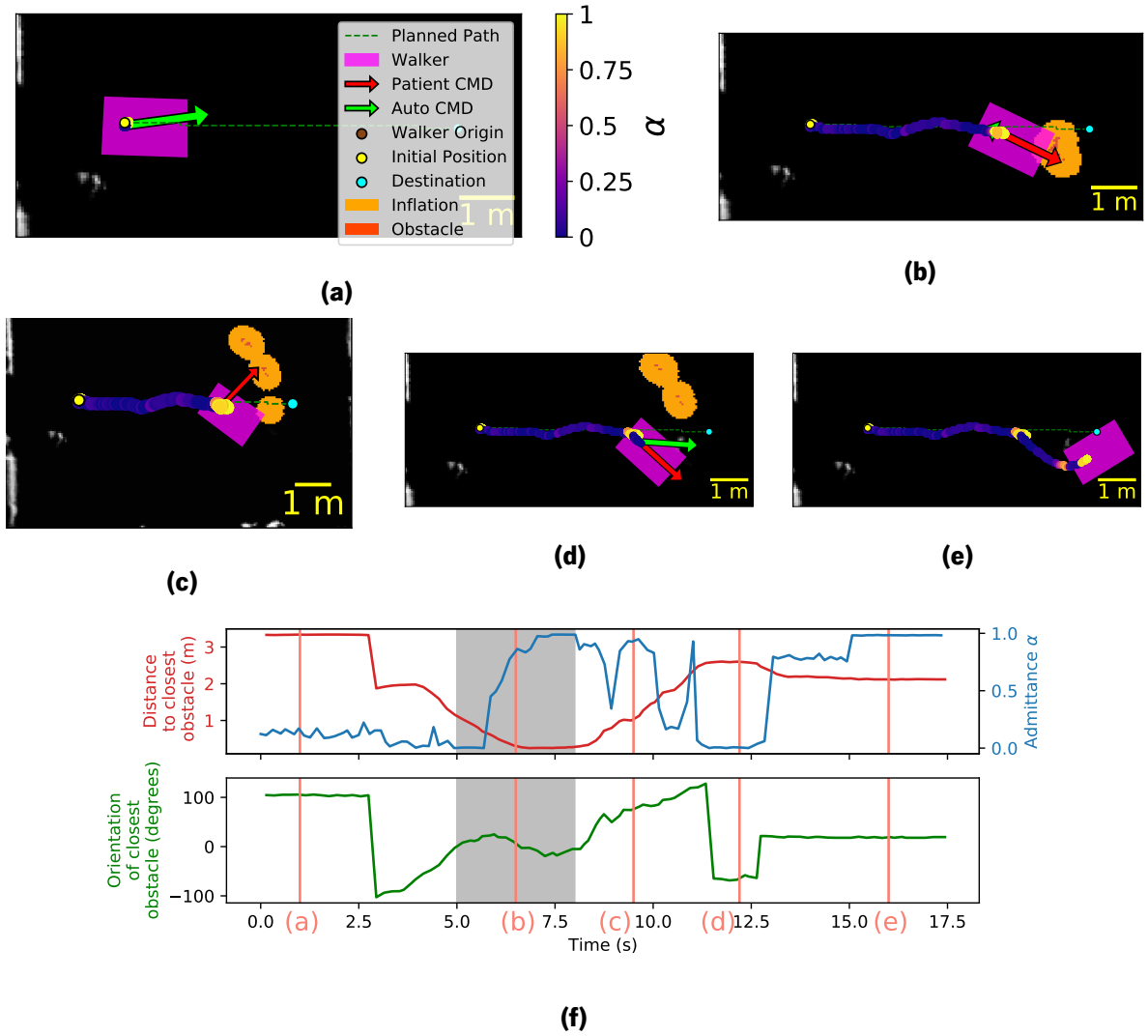


Figure 8.10: Snapshots of the WALKit's interpretation of the environment in the scenario with dynamical obstacles while validating the SC. The walker starts the scenario by presenting low admittance ($\alpha = 0$) (a), finds an obstacle and becomes autonomous (b) staying still until the obstacle leaves (c). The scenario ends by lowering the α value to 0 (d), allowing the patient to drive, reaching the destination (e). The plots of the distance and orientation of the closest obstacle along the plot of the admittance value α throughout the entire trial are shown (f), highlighting regions where the distance to obstacle (I) and the orientation to the obstacle (II) influenced the values of α, respectively.

Also, in the trial from Figure 8.10f, we can notice that when the orientation of the walker relative the obstacle is zero (in front) and the distance is below 1 meter, the prioritizer raises the α value directing the control towards the autonomous supervisor. However, in another trial from scenario 4, when distance is below 1 meter but the closest obstacle is not directly in front of the walker, as shown in Figure 8.11, the admittance level does not transition to 1 due to the relative orientation of the walker.

A real photograph showing the WALKit when a user is trying to collide with a dynamic obstacle (person in front of the WALKit) is shown in Figure 8.12, similar to the situation shown in Figure 8.10b.

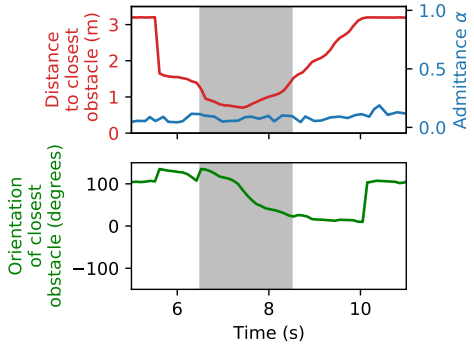
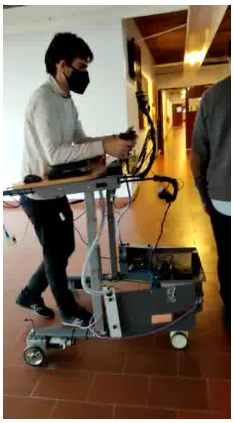
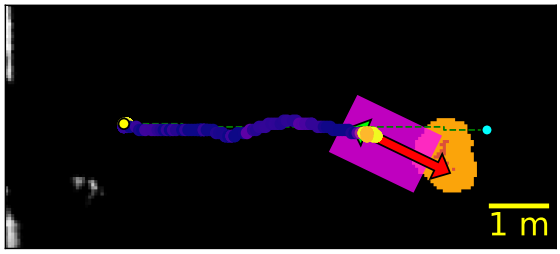


Figure 8.11: Plot of admittance, obstacle distance and orientation relative to the walker throughout one trial of scenario 4. As shown in the gray highlighted are, the α value does not change to 1 despite the distance inferior to 1 meter, as evidenced in Figure 8.10.



(a) Real Environment



(b) WALKit's interpretation of the environment

Figure 8.12: WALKit autonomously stopping in front of obstacle despite user's commands inciting the collision.

Regarding the update frequency of the SC, whose box plot is presented in Figure 8.13, suggest that the recommended value of 10 Hz was not met, however, the minimum update rate (4 Hz) was. These low update rates are created by the implementation of the full SC using Python 3.8, allied to all the other software already being executed, such as: Cartographer and *Dynamic Window Approach*.

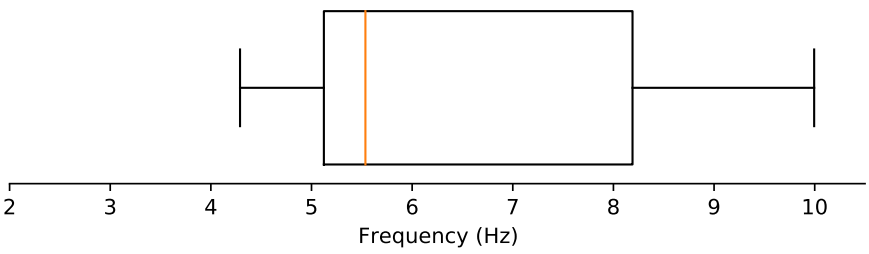


Figure 8.13: Box plot of the SC update frequency.

8.2.2 Training

To study the controller training procedure, the training was executed 10 times and the results were analyzed with a confidence interval of 95%, like shown in Figure 8.14.

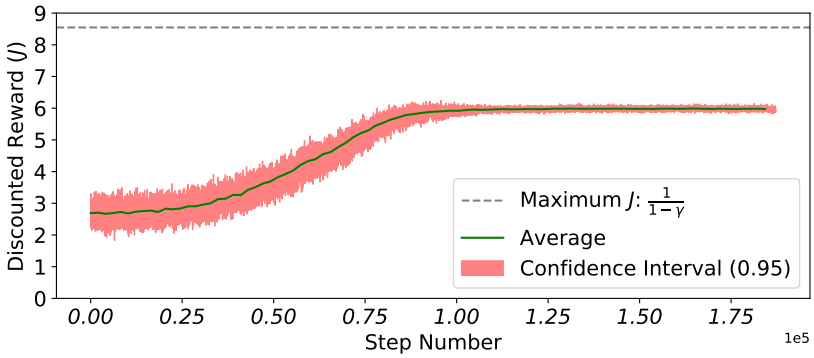


Figure 8.14: Plot of the Discounted Reward J throughout training.

The controller was trained until J stabilized around 1.25×10^5 steps, which corresponds to 3 hours and 28 minutes of gait rehabilitation time. Due to the training being performed in a low performance PC, the *Gazebo* simulation speed was limited to 30% of the real time so that the simulation steps would not interfere with the performance of other algorithms (SLAM, Local Planner, Gait Analysis, etc.), increasing the time of training to 11 hours and 34 minutes.

Chapter 9

Discussion

9.1 Localization and Maps

The spatial errors detected in the LiDAR-SLAM are below the set threshold, as already mentioned in section 8.1.1, suggesting that the present system is capable of creating accurate maps, while also correctly identifying the position and orientation of the walker within it. These errors are below the requirements (25% in the worst case scenario), however, these might be even inferior than registered due to the ground-truth measurements performed by *Aruco* and the ruler inheriting errors of 0.05 m, which are non-negligible.

The *Cartographer* package associated with the Hokuyo's LiDAR works as intended in terms of functionality with low spatial errors, but in a temporal analysis the results present a higher variance, not always being able to reach the goal update rate. As already mentioned the majority of delays in the SLAM computations take place when the environments are being scanned for the first time. To avoid these delays, a pre-driving scan was performed by an expert so that Cartographer does not need to create a full map from scratch while performing gait rehabilitation. The pre-driving scan improved the performance of the system significantly allowing for elimination of the frequency outliers registered in Figure 8.3. The mentioned pre-driving scan is not only advantageous for efficiency of the mapping algorithm but also to avoid performing a new mapping for every session of rehabilitation. Due to the fact that these pre-scans are done using the same system as the one used in gait rehabilitation scenarios it is completely valid to assume that they can be performed in hospital environments, offering no additional effort for medical staff.

The maps created however presented some unexpected artifacts, as shown in Figure 8.4, possibly introducing some ambiguity in the map interpretation for patients and doctors, hampering the use of such

maps by inexperienced users. Some post-processing of the scanned maps could remove the ambiguity of interpretation of the noisy maps. Lavrenov et al. [7] propose a filtering process with a non-linear median filter obtaining 3D models as show in Figure 9.1. The 3D models can be uploaded to the *Gazebo* simulator or just a renderer, possibly helping with the interpretation of the maps created.

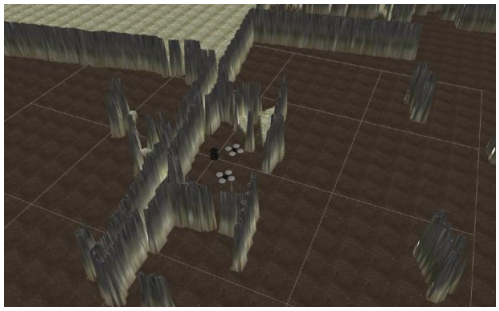


Figure 9.1: Refined map obtained by Lavrenov et al. [7].

Despite the current system being capable of mapping all the tested scenarios, the same is not guaranteed for environments that lie slightly outside the gait rehabilitation environment set. It should be referenced that the current framework was developed only having certain specific scenarios in mind, but building a robust walker that works in most should be the target. Nevertheless, the characteristics of the current framework, especially the LiDAR, cause some limitations in system. The LiDAR *Hokuyo URG-04LX-UG01*, due to its short range and incomplete field-of-view, is incapable of building a complete surrounding obstacle profile in environments like big halls. In multiple scenarios, some shown in Figure 9.2, the current system might not be able to infer the correct position, and in result not be able to determine the velocity which poses a problem from safety standpoint. Also, the fact that the LiDAR sensor does not scan in 3D, makes it impossible for the walker to detect downward stairs and features that could be used for the SLAM algorithm like roof lamps. Using a SLAM approach similar to the long-range 3D sensor mentioned in the work developed by Moosmann and Stiller [66] could greatly improve the robustness of the system.

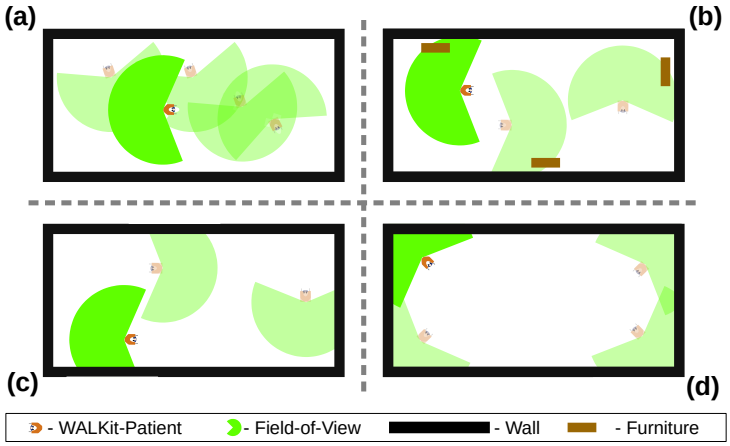


Figure 9.2: Examples of the uncertainty in localization caused by the short range LiDAR in uniform environments. The walkers and fields-of-view without transparency represent the true position of the walker. The ones with transparency represent some possible outputs of the position determined by the LiDAR-SLAM system. If the walker is not detecting any obstacle (a) it might output a multitude of positions. If it detects a wall (c), the system only can position the walker close to a portion of the map which contains a wall. If the system detects a corner (d) it cannot determine in which corner is the correspondent. Finally, even when it detect obstacles (b), if the profile is uniform, the results may vary.

9.2 Autonomous Mode

Executing rehabilitation scenarios related to moving forward and performing a 180°turn (Scenario 0 and 1), proved the ability for the global planner to draw straight paths from start to the destinations and the local planner to drive in uniform paths close the global path without taking unnecessary turns, when no obstacles and no complex environments are present. However, in scenario 1 from Figure 8.5b, the walker did not perform the expected behaviour by performing a pure turn of 180°at the start of the scenario, instead performing the wide-turn reported. The reason why it does not perform the mentioned behaviour is closely related to what happens in the non-standard scenario from Figure 8.8. The *Dynamic Window Approach* local planner from the *base_local_planner* package is optimized to uniformly shaped robot footprints - the area occupied by the walker is highly symmetrical and centered in the rotation axis. When the robot's footprint is uniformly shaped, the local planner does not have to predict the changes of area occupied by the robot while performing pure turns because these do not occur. However, the WALKit's rotation axis is not positioned at the center of the footprint, as evidenced in Figure 8.8, so pure turns produce changes in the area occupied by the walker forcing for a more rigorous planning with longer prediction horizons [67]. For this reason, the walker produces certain behaviours which do not fully comply

with the expected, suggesting that the chosen local planner is not the best for the task at hand. Demeester et al. [67] presented an alternative for non-holonomic arbitrarily shaped robots which builds a local path in 2 phases: a first phase similar to the *Dynamic Window Approach* simulates multiple trajectories considering kynematic constraints and the cost of each path (as described in 2.4.2) and a second phase that, instead of only using the robot origin point, plans using the robot's geometry and kinodynamic constraints. The newly mentioned algorithm could possibly solve the limitations of the local planner currently being used in the WALKit.

The trials performed in validation scenarios 2 and 3 (turning right and left corners) prove that, given certain static obstacles scanned and mapped by the LiDAR-SLAM system, the A* global planner is capable of delineating a path which takes the environment into consideration by never crossing any obstacle. The local planner *Dynamic Window Approach* proved to able to detect obstacles and perform collision avoidance of obstacle previously scanned and mapped, due to no collisions occurring and the destination always being reached.

In scenario 4, the A* global planner was able to plan a direct path from start to destination identical to scenario 0, but, instead of strictly following the global path, the local planner was able to detect the dynamic obstacle not present in the pre-scanned map, deviating from global path and avoiding the movable obstacle. In this scenario the local planner proved not only capable of following the global path closely (low KTEs), but also, when necessary, to completely ignore the global path and avoid the obstacle.

In addition to the ability of the local planner always being capable of directing the walker autonomously to the destination, the low values of KTE suggest that the paths described are indeed close to the path suggested by the global planner A*, which turns the trajectories predictable when the global plan is known. Furthermore, based on the fact that no collisions occurred and the destinations were always reached, it can be stated that the planners were able to adequately drive the walker autonomously.

In some occasions the local planner showed some indecisiveness due to desynchronization between the map and pose and the real position of the WALKit, which translated into full stops. These desynchronizations are rare and usually happen after loading the map obtained from pre-driving scan described in the previous section 9.1. To avoid this, it is advised that, when using previously scanned maps from another session, before engaging the autonomous mode, synchronization "laps" are performed by an expert so that WALKit can start the autonomous mode with correct pose and map measurements. The probability of the desynchronization effect decreases significantly after the first synchronization, however there is low probability of occurring at any stage. Despite this effect, the walker is always able to transition to a full-stop, guaranteeing patient's safety, due to the local planner always keeping track of obstacles using

data from the LiDAR system that is not subjected to any kind of processing.

The ability to create global paths taking into consideration obstacles present in the maps in a negligible time, and the ability to propose velocity commands which avoid either static or dynamic obstacles, while keeping close to the global path suggest that the use of a global and local strucutre suggested by the ADS [3] and walker SC [6, 15] literatures can perform collision risk assessment and prevention, while also performing a trajectory, possibly serving as autonomous supervisors replacing human ones.

9.3 Shared-Control

By performing scenario 0, the low and uniform α values, the highest patient control time percentage (relative to other scenarios) and higher values of KTE when comparing to the same scenario in autonomous driving validation, suggest that the SC proposed is capable of mainly prioritizing the control patient when no obstacles are present and the path is straight. Moreover, the prioritizer does not allow the patient to fully control the walker, as evidenced by the patient control time not being 100%, pointing towards the fact that the SC is capable of taking into account the advice from the autonomous supervisor and converging to the destination.

In scenario 1, where the walker is supposed to perform an 180°turn, the KTE values were exceptionally high, which contradict the expected lower values when comparing to collision avoidance scenarios, however, these results are influenced by the exceptionally high KTEs registered in the autonomous supervisor module. As already discussed, the local planner in the autonomous mode does not perform the expected behaviour, so, the wide-turn reported "spills" over to the SC. Changing the used local planner could remove the wide-turn behaviour, reducing the KTEs, removing the unexpected paths.

Also, in scenario 1, the exceptional high α at the start evidences that, the walker was capable of inferring that its orientation at the beginning was opposite to the direction of the destination and that the probability of reaching the goal with that orientation is low, which caused the autonomous supervisor to assume control and gradually transition the control back to the patient upon pointing towards the destination. Therefore, the use of the local cost map as a component of the state-space proved to supply the information necessary for the walker to understand if it is approaching the destination.

In scenario 2, 3 and 4, the registered α close to 1 caused by proximity to obstacles suggests that the controller is capable of detecting obstacles and determine that the autonomous supervisor is the entity most capable of dealing with the dangerous situation, which complies with the expected behaviour, producing the lowest relative patient control times. Scenario 2 and 3, specifically, show that the WALKit

is capable of handling pre-mapped and static obstacles, while scenario 4 demonstrates the ability of dealing with ob not pre-mapped and dynamic. Due to raising the α value to 1, the patient's input did not influence the walker's control when close to obstacles, which is expected because, as it was defined in the SC structure, the autonomous mode would function as an expert whose commands are followed until the risk is neutralized. Therefore, the policy DNN output does not change despite of the different velocity commands supplied by the patient. This can be dangerous for the well being of the user for the reason that the walker starts moving in an unexpected way - prevalence of a sense of unresponsiveness. To avoid this, a communication system should be implemented in the walker so that the patient is aware of what both the SC and ADS intend to do. A feedback system with human-in-the-loop, as shown in Figure 9.3, with the objective of exerting torques and forces through actuators implemented in the handles so that the position and stiffness of handles themselves possibly give the necessary information for the patient to "feel" the autonomous instructions. If the patient perceives the handles being stiff and moving autonomously he can understand where the walker wants to drive to and prepare for the upcoming movement. Otherwise, the handles become soft and stop moving autonomously allowing the patient to drive. The mentioned interface follows a similar structure as proposed by Steele and Gillespie [68], however the cited work is applied to a haptic steering wheel to aid in land vehicle guidance.

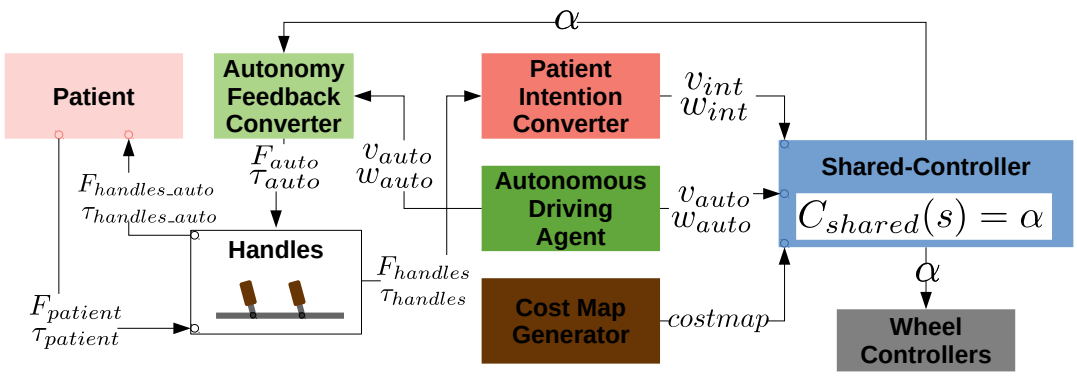


Figure 9.3: Proposed feedback and human-in-the-loop shared control system. F - Linear Force, τ - Torque.

The increased values of KTE in the SC validation, when compared to the same regarding the autonomous supervisor suggest that the walker did not strictly follow the path recommended by the global planner, showing a possible improvement over the strategies proposed by the state-of-the-art that always try to force the walker to stay close to the original path [15, 20].

The orientation of the walker seems to affect the prioritization of the actions, as shown in Figures 8.10f and 8.11, *i.e.*, the walker seems to have higher α (autonomous) when obstacle are directly on the front of the walker ($0°$), which is expected due to the chances of collision being higher when the walker is facing the

obstacle. This property shows that the policy network structure - feed-forward, convolutional and linear - is not only able to detect the presence of obstacles, but also determine what are the consequences of their position to the driving patterns, property which advantageous when comparing to the SC not embedded with obstacle avoidance features as Morris et al. [20] and Jiménez et al. [6].

When approaching the goal, the admittance values always converged to 1, turning the walker autonomous. The walker is always more motivated to converge to the goal when close to it than following patient's instructions. Despite in certain circumstances no obstacles being present and the patient's intention not causing any dangers, when close to the goal, the walker prefers to not consider the patient's intention and finish the episode, so that situations where the walker drives close to the goal and the scenario is not finished are avoided. This pattern is seen in all finishing steps of every scenario due to the reward function, in Equation 7.6, supplying the agent with a bigger reward when the walker finishes the episode successfully. The probability that the patient commands are adequate to reach the destination is inferior to the commands from the autonomous supervisor, so the agent learns to ignore the patient completely. The disregard for patient commands might cause some problems similar to the problems of engagement of the autonomous mode for obstacles avoidance mentioned above, creating one extra reason for the development of the mentioned active handles with autonomous mode feedback and human-in-the-loop.

In scenarios where the walker is approaching the obstacle, the time the SC takes to change α from 0 to 1 is within the established time-window created ($1 \sim 2$ seconds) by the γ factor (0.883), as evidenced by the transition highlighted in gray from Figure 8.10f. However, in certain circumstances, the α transitions do not occur smoothly, as also evidenced in the plot from Figure 8.10f, transitions which can be too abrupt risking the stability of the patient [19]. Certain changes could avoid the problematic behaviours: implementing penalizations included in the reward function for the high velocity derivative values, increasing γ and decreasing the rate of change of the $dist_{factor}$ from Equation 7.6. Penalizing the derivative of velocity directly turns the agent unmotivated to capitalize on such reward. Increasing the γ forces the Critic network to estimate the state-value based on a bigger horizon and so the walker can become more conservative, transitioning gradually. Also, reducing the rate of change of the $dist_{factor}$ changes the reward function, motivating the walker to become gradually autonomous when the obstacles are farther away. It should be referenced that increasing γ causes the optimization problem to become significantly more difficult to solve because a wider time window (Figure 7.2) needs to be considered, as already mentioned in chapter 4, due to the need for studying the influence of each action in each state for a longer period, requiring more training and time [27]. Through experimentation it was determined that the chosen value for γ offers the best tradeoff between risk management performance and computation

time, considering the limitations in computational power mentioned in section 5.2.1.

If values of α constantly changed smoothly, the full action-space range established $[0,1]$ would be used, and the continuity of the action space would prove to be as important as firstly hypothesized. Nevertheless, the continuity of the action space is always an advantage because it allows for γ to be increased in the future, creating smooth α signals without the need to change the policy structure.

Using the cost map and velocity commands from the patient and autonomous mode as state-space with the exclusion of dynamic information to reduce the state-space dimensionality proved to be sufficient to prioritize control, due to no collisions being registered. However, in one trial from scenario 2, the walker was not able to reach the destination because the patient driver tried, consistently, to collide with a specific obstacle. The walker avoided the obstacle faced in scenario 2, returned the control to the patient, but the patient always returned to the same near-collision position. Some controllers, as the one proposed by Huang et al. [5], perform a temporal analysis, in the specific case of predicting where the obstacle might move in the next steps. Performing this analysis in our controller could avoid the registered abrupt changes in α, and also allow the controller to understand that the patient is constantly trying to avoid goal convergence, becoming gradually more autonomous if the user persists. Adapting the policy and state-value DNNs structure to a recurrent one with Long Short-Term Memory layers, as suggested by Gers et al. [69], without adding the previously mentioned dynamic information to the state-space, could enable the agent to perform temporal analysis on multiple static frames. The augmentation of the DNN structure would increase the number of parameters to be tuned and increase training times, although the controller framework would be more generic (one goal of this dissertation).

In terms of training procedure, the algorithm PPO was able to maximize the discounted reward J but not up to the maximum theoretical value, as evidenced in Figure 8.14. The RL algorithm should always find the optimal policy π^* which maximizes the discounted reward with the assumption that the reward function represents the optimal behaviour. In our training scenarios, the chosen reward was capable of producing the behaviour shown in Figure 8.9, which is close to the behaviour implied by the reward. Still, J does not achieve its maximum because the indication to become autonomous when the distance is inferior to 0.6 m, implied by the reward function, was proven to be too aggressive, producing the already reported sudden changes in the admittance. Then, the agent decided to converge to a policy that gets penalized when choosing to become autonomous for distances to obstacles higher than 0.6 m, and avoid the high probability of getting closer to the obstacle, colliding, and receiving the worst reward, consequently choosing to become a better agent on the long-term. Using a new reward that influences the walker to engage the autonomous mode at a distance above 0.6 m represents more accurately the behaviour trying

to be reproduced, allowing the J values to reach the maximum.

The SC presents the different results as the ADS, mentioned in section 9.2, with regard to the update rate of the SC. The SC was not able to keep up with the pre-established 10 Hz, when every functionality is running - SC, ADS, LiDAR-SLAM - and all computer resources are used, leaving no additional resources for other WALKit features, like gait and posture analysis, however, it still respected the 4 Hz mark. The reason for the complete computer power use is the execution of the RL and SC with an interpreted language (*Python*). If these software packages were written in a compiled language like C++ the execution time could possibly decrease drastically leaving more time for other algorithms to compute data. This translation between languages was not done due to the need for interfacing with packages like *PyTorch* and *MushroomRL*.

Data obtained from the validation protocols supports the fact that the SC developed, follows all the set objectives in terms of functionality due to never colliding, which is the ultimate goal. The use of a RL algorithm (PPO) with a trial and error training and optimization through the gradient descent algorithm allows the agent to express a behaviour that considers a full range of possibilities, instead of using simple heuristics with hard constraints to solve the problem as in the state-of-the-art, evidencing **sophisticated intelligence**.

If some feature or sub-controller needs to be added to the shared-controller hierarchy, one only needs to change the DNN structure, state and action space, followed by a re-train of the policy. Despite the need for training a new policy every time the structure is changed, it should be faster to re-train than to create and validate a set of heuristic which will probably behave in a non-optimal way. The **scalabe** and **generic** nature of DNNs used in Deep Actor-Critc RL algorithms enable the structure to be changed without the need for a complete rework, as suggested with the implementation of recurrent layers.

The reward function proved to be easily parameterizable without compromising functionality. One simple change was already mentioned in this discussion to improve the behaviour of the walker, improvement which is abstract in terms of long-term consequences, but the through a trial and error process, the walker is able to converge to the broad behaviour implied by the reward. No tests were performed with medical experts but due to the simple nature of the reward function it is assumed that the controller possesses **human-like tunability**.

In the validation protocols delineated in this dissertation, the used metrics were chosen due to their use in literature, such as: number of collisions [5, 6, 15, 20, 23], number of destinations reached [5, 6, 15, 20, 23], KTE values [6], relative patient control time [15, 23] and case-by-case analysis [5, 20, 23]. In results from the SC presented in this dissertation, we obtain an average relative patient control time of

70.70%, which is a higher value when compared to the 66.71% og Sierra et al. [15], suggesting that our controller is better in terms of patient prioritization. The previous conclusion might be incorrect because the variety of scenarios in literature and scarcity of standards imposing restrictions and rules for validation, force the evaluation process to be subjective and comparisons between methods to be misleading. Also, certain metrics, such as KTE, do not guarantee that the controller is working at maximum performance, because the patient might deviate from the global path and the SC task of diminishing fall-risk might not be compromised, due to its dependence to obstacles presence. The only two objective metrics that do not depend on the validation protocol performed are the number of collisions and number of destinations reached. For the context of this dissertation, it is usually assumed (as referenced in the Problem Statement section 1.2) that if the walker behaves in unpredictable ways and collides with obstacles, falls and poor gait metrics are likely to occur. This assumption is generally true, although it does not include the full range of causes for poor performance. Nevertheless, to completely replace the human supervisor, it must be guaranteed that autonomous gait rehabilitation scenarios are performed in such a way that fall-risk is diminished and gait quality improvement is, at least, similar to the achieved by the human supervisor, if not better. So, to validate the system, we propose the use fall-risk and gait quality assessors to measure directly the metrics instead of using rough assumptions. Also, patient/medical expert reviews should also be used for evaluating these SCs in regards to functionality and ergonomy. However, it was impossible to assess such metrics in this dissertation due to the unavailability of patients and medical personnel testing, and for the great amount of time needed to deploy gait quality and fall-risk assessment features.

Chapter 10

Conclusions

The SC proposed is able to correctly evaluate the risk faced by the patient-WALKit system, and symbiotically prioritize which entity should control the walker (either autonomous supervisor or patient intention), based on information about the surroundings of the walker.

An association of a LiDAR sensor with the Google's Cartographer SLAM software enabled the acquisition of the environment map and localization of the walker, system which was exposed to a thorough validation process by the detection of QR Code markers as position references. The validation process led to conclusion that both positioning and mapping presented errors maximum errors of 0.38 m, 25% below the set requirement. The Cartographer software updated the map (7 Hz) slightly below the target update frequency (10 Hz), however a pre-driving scan was implemented to improve mapping efficiency.

An ADS was implemented in the walker, which is responsible for supplying autonomous supervision, so that when the walker is in a dangerous situation, it can resort the autonomous commands reducing the risk faced. The autonomous mode uses the global planner *A**, responsible for drawing a spatial path from the current position to the goal selected by either the patient or the medical expert. *A** showed always being able to plan from start to the destination positions. The local planner *Dynamic Window Approach* responsible for creating velocity commands to converge to the goal while avoiding collisions, showed the capacity to autonomously drive the walker safely, possibly being capable of replacing the human supervisor. Despite the promising results within the validation environments, the same was not evidenced in environments where gait rehabilitation are unlikely to occur.

In this dissertation, the proposed SC developed uses a RL algorithm named PPO, which learns the intended behaviour by capitalizing on a medically defined reward function, through a process of trial and error. The DNNs used as policy and as state-value estimator were able to accurately perform collision-risk

assessment and avoidance. Overall, the controller proposed was capable of delegating the control of the WALKit 70.70% of the time to the patient with no collisions detected, in this way respecting the assist-as-needed concept [21]. Due to the short discounting factor γ and the detection of a movable obstacles, sudden breaking patterns were induced in the walker caused by the fast transitions between intention to autonomous control. These transitions might cause, in certain circumstances, some danger for the patient.

Results indicate that the SC proposed is a promising solution in the validation scenarios with healthy patients, although further testing with patients with GD is needed.

All goals set for this dissertation were achieved with exception for Goal 7 (Validate the proposed framework within gait rehabilitation scenarios). The completion of this dissertation enables to answer the raised Research Questions (RQs):

- **RQ 1: Can the medical expert be replaced by the autonomous agent developed?** Results show that the ADS developed is able to produce velocity commands capable of avoiding obstacles and reach the goal in gait rehabilitation environments, keeping the patient safe. In environments where gait rehabilitation scenarios are unlike to occur, the walker might not converge to goal, requesting the help of medical personnel as a last resort. However, the autonomous mode guarantees patient safety at all times.

- **RQ 2: Which information is required to assess risk faced by the patient-WALKit system?** In the system proposed, the walker was able to assess risk with just static information about obstacles, path and velocity commands from the interfering parties (autonomous and patient). However, results show that this state space can only work within certain assumptions where the walker is able to fully break in a short time frame. Expanding the state space to accommodate information about the dynamic nature of the environment or recursion networks could greatly improve the smoothness of the behaviour, guaranteeing the safety of the patient.

- **RQ 3: Which metrics need to be measured to validate both the ADS and SC?** State-of-the-art approaches trying to solve the issue of SC in smart robotic walkers show a lack of standardization and objectivity for key performance indicators. Metrics for evaluation should be stipulated in close work to medical experts and patients. We present frequency of collisions, frequency of reaching the destination, KTE and relative patient control time as objective metrics for evaluation as the most objective metrics to validate the current system. However, these metrics are indirect risk assessors that rely on diverse amount of assumptions, therefore, creating metrics that

directly evaluate fall-risk and gait quality should be studied.

10.1 Future Work

As already discussed along this dissertation, multiple changes could improve robustness of the SC.

Functionality reviews by patients and medical experts should be performed, but more importantly key performance indicators should be defined as to standardize the validation process for future implementations of this controller. This would allow an increased objectiveness to be induced in the work flow, and clear metrics would possibly induce relevant improvements for SC of smart walkers.

The introduction of a novel autonomous driving algorithm capable of performing long-term planning, instead of just being capable of performing simple movements of rotation and translation, would be able to solve the cases where the current local planner *Dynamic Window Approach* failed to do so, turning the walker fully independent from the medical supervisor.

Implementing an active controller in the drivable handles that induces haptic sensation on the patient, would help the last to infer where the walker is trying to go, removing the feeling of unresponsiveness when autonomous agent assumes control. This could greatly improve complacency of the patient to the walker, making the user feel more confident while using the rehabilitation device.

Training the same model with increased discounting factor γ, in a variety of worlds (not only hospital and corridor simulation environments) and with higher computational power could greatly improve on the trajectory smoothness issue, reducing the chances of dangerous events for the patient.

When mapping environments with increased size (outside the normal values for rehabilitation) the use of a LiDAR system capable of tracking a higher range and even in 3D could highly improve both the quality of mapping and interpretation of the same map for patient and medical expert.

Finally, the RL packages developed in a compiled language like C++ could greatly improve the time required for training and execution time of the algorithms, and even interfacing with ROS, allowing other features of the walker to operate simultaneously.

Bibliography

[1] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A Survey of Autonomous Driving: Common Practices and Emerging Technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, jun 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9046805/

[2] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, dec 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7747236/

[3] E. Yurtsever, Y. Liu, J. Lambert, C. Miyajima, E. Takeuchi, K. Takeda, and J. H. Hansen, "Risky Action Recognition in Lane Change Video Clips using Deep Spatiotemporal Networks with Segmentation Mask Transfer," *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019*, pp. 3100–3107, 2019. [Online]. Available: http://dx.doi.org/10.1109/ITSC.2019.8917362

[4] B. P. Gerkey and K. Konolige, "Planning and Control in Unstructured Terrain," in *ICRA Workshop on Path Planning on Costmaps*, 2008. [Online]. Available: http://pub1.willowgarage.com/apubdb{_}html/files{_}upload/8.pdf

[5] C. Huang, G. Wasson, M. Alwan, P. Sheth, and A. Ledoux, "Shared Navigational Control and User Intent Detection in an Intelligent Walker," Tech. Rep.

[6] M. F. Jiménez, M. Monllor, A. Frizera, T. Bastos, F. Roberti, and R. Carelli, "Admittance Controller with Spatial Modulation for Assisted Locomotion using a Smart Walker," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 94, no. 3-4, pp. 621–637, jun 2019. [Online]. Available: http://link.springer.com/10.1007/s10846-018-0854-0

[7] R. Lavrenov, A. Zakiev, and E. Magid, "Automatic mapping and filtering tool: From a sensor-based occupancy grid to a 3D Gazebo octomap," in *2017 International Conference on*

Mechanical, System and Control Engineering, ICMSC 2017, 2017, pp. 190–195. [Online]. Available: https://www.researchgate.net/publication/318036867

[8] P. Mahlknecht, S. Kiechl, B. R. Bloem, J. Willeit, C. Scherfler, A. Gasperi, G. Rungger, W. Poewe, and K. Seppi, "Prevalence and Burden of Gait Disorders in Elderly Men and Women Aged 60-97 Years: A Population-Based Study," *PLoS ONE*, vol. 8, no. 7, jul 2013. [Online]. Available: /pmc/articles/PMC3722115/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC3722115/

[9] C.-Y. Lee and J.-J. Lee, "Walking-support robot system for walking rehabilitation: design and control," *Artificial Life and Robotics*, vol. 4, no. 4, pp. 206–211, dec 2000.

[10] H. Barbeau and M. Visintin, "Optimal outcomes obtained with body-Weight support combined with treadmill training in stroke subjects11No commercial party having a direct financial interest in the results of the research supporting this article has or will confer a benefit upon the author(s) or upon any organization with which the author(s) is/are associated." *Archives of Physical Medicine and Rehabilitation*, vol. 84, no. 10, pp. 1458–1465, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0003999303003617

[11] R. Riener, L. Lünenburger, S. Jezernik, M. Anderschitz, G. Colombo, and V. Dietz, "Patient-cooperative strategies for robot-aided treadmill training: First experimental results," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 13, no. 3, pp. 380–394, sep 2005.

[12] J. F. Veneman, R. Kruidhof, E. E. Hekman, R. Ekkelenkamp, E. H. Van Asseldonk, and H. Van Der Kooij, "Design and evaluation of the LOPES exoskeleton robot for interactive gait rehabilitation," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 15, no. 3, pp. 379–386, sep 2007.

[13] C. Pinheiro, J. M. Lopes, J. Figueiredo, L. M. Goncalves, and C. P. Santos, "Design and technical validation of a wearable biofeedback system for robotic gait rehabilitation," in *2020 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2020*. Institute of Electrical and Electronics Engineers Inc., apr 2020, pp. 16–21.

[14] C. D. Lim, C. M. Wang, C. Y. Cheng, Y. Chao, S. H. Tseng, and L. C. Fu, "Sensory cues guided rehabilitation robotic walker realized by depth image-based gait analysis," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 1, pp. 171–180, 2016.

[15] S. D. Sierra, M. F. Jiménez, M. C. Múnera, A. Frizera-Neto, and C. A. Cifuentes, "Remote-Operated Multimodal Interface for Therapists during Walker-Assisted Gait Rehabilitation: A Preliminary Assessment," in *ACM/IEEE International Conference on Human-Robot Interaction*, vol. 2019-March. IEEE Computer Society, mar 2019, pp. 528–529.

[16] K. R. Mun, S. B. Lim, Z. Guo, and H. Yu, "Biomechanical effects of body weight support with a novel robotic walker for over-ground gait rehabilitation," *Medical and Biological Engineering and Computing*, vol. 55, no. 2, pp. 315–326, feb 2017. [Online]. Available: http://link.springer.com/10.1007/s11517-016-1515-8

[17] M. Martins, C. Santos, A. Frizera, and R. Ceres, "Real time control of the ASBGo walker through a physical human-robot interface," *Measurement: Journal of the International Measurement Confederation*, vol. 48, no. 1, pp. 77–86, feb 2014.

[18] P. Li, Y. Yamada, X. Wan, Y. Uchiyama, W. Sato, K. Yamada, and M. Yokoya, "Gait-phase-dependent control using a smart walker for physical training," in *IEEE International Conference on Rehabilitation Robotics*, vol. 2019-June. IEEE Computer Society, jun 2019, pp. 843–848. [Online]. Available: https://ieeexplore.ieee.org/document/8779563/

[19] R. Moreira, J. Alves, A. Matias, and C. Santos, "Smart and Assistive Walker - ASBGo: Rehabilitation Robotics: A Smart-Walker to Assist Ataxic Patients." *Advances in experimental medicine and biology*, vol. 1170, pp. 37–68, 2019. [Online]. Available: http://link.springer.com/10.1007/978-3-030-24230-5{_}2

[20] A. Morris, R. Donamukkala, A. Kapuria, A. Steinfeld, J. T. Matthews, J. Dunbar-Jacob, and S. Thrun, "A robotic walker that provides guidance," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 1, 2003, pp. 25–30.

[21] S. Srivastava and P. C. Kao, "Robotic Assist-As-Needed as an Alternative to Therapist-Assisted Gait Rehabilitation," *International Journal of Physical Medicine & Rehabilitation*, vol. 4, no. 5, 2016. [Online]. Available: /pmc/articles/PMC5450822/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC5450822/

[22] M. Spenko, H. Yu, and S. Dubowsky, "Robotic personal aids for mobility and monitoring for the elderly," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 14, no. 3, pp. 344–351, sep 2006. [Online]. Available: http://ieeexplore.ieee.org/document/1703566/

[23] B. Graf and R. D. Schraft, "Behavior-based path modification for shared control of robotic walking aids," in *2007 IEEE 10th International Conference on Rehabilitation Robotics, ICORR'07*. IEEE, jun 2007, pp. 317–322. [Online]. Available: http://ieeexplore.ieee.org/document/4428444/

[24] S. McLachlan, J. Arblaster, D. K. Liu, J. V. Miro, and L. Chenoweth, "A multi-stage shared control method for an intelligent mobility assistant," in *Proceedings of the 2005 IEEE 9th International Conference on Rehabilitation Robotics*, vol. 2005. IEEE, 2005, pp. 426–429. [Online]. Available: http://ieeexplore.ieee.org/document/1501134/

[25] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1271–1278.

[26] F. Duchon, A. Babinec, M. Kajan, P. Beno, M. Florek, T. Fico, and L. Jurišica, "Path planning with modified A star algorithm for a mobile robot," in *Procedia Engineering*, vol. 96. Elsevier Ltd, 2014, pp. 59–69.

[27] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[28] A. Woo, B. Fidan, and W. W. Melek, "Localization for Autonomous Driving," in *Handbook of Position Location*. Wiley, jan 2019, pp. 1051–1087. [Online]. Available: http://doi.wiley.com/10.1002/9781119434610.ch29

[29] C. Debeunne and D. Vivet, "A Review of Visual-LiDAR Fusion based Simultaneous Localization and Mapping," *Sensors*, vol. 20, no. 7, p. 2068, apr 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/7/2068

[30] X. Yu and M. Marinov, "sustainability A Study on Recent Developments and Issues with Obstacle Detection Systems for Automated Vehicles." [Online]. Available: www.mdpi.com/journal/sustainability

[31] I. Brilakis, M. W. Park, and G. Jog, "Automated vision tracking of project related entities," *Advanced Engineering Informatics*, vol. 25, no. 4, pp. 713–724, oct 2011. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1474034611000048

[32] P. Mirowski, T. K. Ho, S. Yi, and M. MacDonald, "SignalSLAM: Simultaneous localization and mapping with mixed WiFi, bluetooth, LTE and magnetic signals," in *2013 International Conference*

on Indoor Positioning and Indoor Navigation, IPIN 2013. IEEE Computer Society, oct 2013, pp. 1–10. [Online]. Available: http://ieeexplore.ieee.org/document/6817853/

[33] S. Sumikura, M. Shibuya, and K. Sakurada, "OpenVSLAM: A Versatile Visual SLAM Framework," 2019. [Online]. Available: https://doi.org/10.1145/3343031.3350539

[34] K. Jo, K. Chu, and M. Sunwoo, "GPS-bias correction for precise localization of autonomous vehicles," in *IEEE Intelligent Vehicles Symposium, Proceedings*. IEEE, jun 2013, pp. 636–641. [Online]. Available: http://ieeexplore.ieee.org/document/6629538/

[35] U. Frese, R. Wagner, and T. Röfer, "A SLAM Overview from a User's Perspective," Tech. Rep. [Online]. Available: www.informatik.uni-bremen.de/agebv/en/

[36] G. Grisetti and C. Stachniss, "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling," Tech. Rep.

[37] N. Sunderhauf and P. Protzel, "Switchable constraints for robust pose graph SLAM," in *IEEE International Conference on Intelligent Robots and Systems*, 2012, pp. 1879–1884.

[38] B. Mu, S. Y. Liu, L. Paull, J. Leonard, and J. P. How, "SLAM with objects using a nonparametric pose graph," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 2016-November. Institute of Electrical and Electronics Engineers Inc., nov 2016, pp. 4602–4609. [Online]. Available: http://ieeexplore.ieee.org/document/7759677/

[39] M. R. Walter, R. M. Eustice, and J. J. Leonard, "Exactly sparse extended information filters for feature-based SLAM," *International Journal of Robotics Research*, vol. 26, no. 4, pp. 335–359, apr 2007. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0278364906075026

[40] C. Kerl, J. Sturm, and D. Cremers, "Dense visual SLAM for RGB-D cameras," in *IEEE International Conference on Intelligent Robots and Systems*, 2013, pp. 2100–2106.

[41] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, dec 1959. [Online]. Available: https://link.springer.com/article/10.1007/BF01386390

[42] *The dynamic window approach to collision avoidance*, vol. 4, no. 1, 1997.

[43] J. Borenstein and Y. Koren, "The Vector Field Histogram—Fast Obstacle Avoidance for Mobile Robots," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, 1991.

[44] Google, "AlphaGo | DeepMind," 2018. [Online]. Available: https://deepmind.com/research/case-studies/alphago-the-story-so-far

[45] J. Tromp and G. Farnebäck, "Combinatorics of Go," Tech. Rep., 2016.

[46] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, jan 2016. [Online]. Available: http://www.nature.com/articles/nature16961

[47] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[48] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, apr 2017.

[49] H. Nguyen, S. J. Maclagan, T. D. Nguyen, T. Nguyen, P. Flemons, K. Andrews, E. G. Ritchie, and D. Phung, "Animal recognition and identification with deep convolutional neural networks for automated wildlife monitoring," in *Proceedings - 2017 International Conference on Data Science and Advanced Analytics, DSAA 2017*, vol. 2018-January. Institute of Electrical and Electronics Engineers Inc., jul 2017, pp. 40–49. [Online]. Available: https://ieeexplore.ieee.org/document/8259762/

[50] M. Mitchell Waldrop, "What are the limits of deep learning?" *Proceedings of the National Academy of Sciences of the United States of America*, vol. 116, no. 4, pp. 1074–1077, jan 2019. [Online]. Available: http://www.pnas.org/lookup/doi/10.1073/pnas.1821594116

[51] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2nd Edition)*. USA: Wiley-Interscience, 2000.

[52] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," Tech. Rep., 2015. [Online]. Available: http://arxiv.org/abs/1511.08458

[53] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[54] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[55] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, Sep 2004, pp. 2149–2154.

[56] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[57] W. Qian, Z. Xia, J. Xiong, Y. Gan, Y. Guo, S. Weng, H. Deng, Y. Hu, and J. Zhang, "Manipulation task simulation using ros and gazebo," in *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, 2014, pp. 2594–2598.

[58] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," in *Proceedings - 2016 7th International Conference on Cloud Computing and Big Data, CCBD 2016*. Institute of Electrical and Electronics Engineers Inc., jul 2017, pp. 99–104.

[59] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.

[60] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent, "Efficient sparse pose adjustment for 2d mapping," 10 2010, pp. 22–29.

[61] B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner, "On Measuring the Accuracy of SLAM Algorithms," Tech. Rep.

[62] C. D'Eramo, D. Tateo, A. Bonarini, M. Restelli, and J. Peters, "Mushroomrl: Simplifying reinforcement learning research," https://github.com/MushroomRL/mushroom-rl, 2020.

[63] Y. Li and Y. Yuan, "Convergence Analysis of Two-layer Neural Networks with ReLU Activation," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017, pp. 597–607. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/a96b65a721e561e1e3de768ac819ffbb-Paper.pdf

[64] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp." Springer, Berlin, Heidelberg, 2012, pp. 9–48. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-35289-8{_}3

[65] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016. [Online]. Available: https://sites.google.com/site/gaepapersupp.

[66] F. Moosmann and C. Stiller, "Velodyne SLAM," in *IEEE Intelligent Vehicles Symposium, Proceedings*. IEEE, jun 2011, pp. 393–398. [Online]. Available: http://ieeexplore.ieee.org/document/5940396/

[67] E. Demeester, M. Nuttin, D. Vanhooydonck, G. Vanacker, and H. V. Brussel, "Global dynamic window approach for holonomic and non-holonomic mobile robots with arbitrary cross-section," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2005, pp. 2357–2362.

[68] M. Steele and R. B. Gillespie, "Shared control between human and machine: Using a haptic steering wheel to aid in land vehicle guidance," in *Proceedings of the Human Factors and Ergonomics Society*, vol. 45, no. 23. Human Factors and Ergonomics Society Inc., oct 2001, pp. 1671–1675. [Online]. Available: http://journals.sagepub.com/doi/10.1177/154193120104502323

[69] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, oct 2000. [Online]. Available: https://doi.org/10.1162/089976600300015015