Chandler, Christopher (2025) *Model checking for trustworthy reasoning in autonomous driving.* PhD thesis.

https://theses.gla.ac.uk/85397/

Copyright and moral rights for this work are retained by the author

A copy can be downloaded for personal non-commercial research or study, without prior permission or charge

This work cannot be reproduced or quoted extensively from without first obtaining permission from the author

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given

Enlighten: Theses
https://theses.gla.ac.uk/
research-enlighten@glasgow.ac.uk

Model Checking for Trustworthy Reasoning in Autonomous Driving

Christopher Chandler

Submitted in fulfilment of the requirements for the
Degree of Doctor of Philosophy

School of Engineering
College of Science and Engineering
University of Glasgow

March 2025

Abstract

This thesis proposes model checking for trustworthy reasoning in autonomous driving. We combine closed-loop input control and model checking to plan sequences of closed loop controllers for obstacle avoidance in real-time—the former for its superior reactivity to immediate stimuli, and the latter for its strength in modal reasoning. In addition, as both closed-loop control and model checking are transparent, decisions can in principle be explained. To demonstrate our approach, we implement a bespoke model checker on low-powered autonomous robot, an additional measure to further promote resource efficient planning. We conduct two case studies. In the first we focus simply on avoiding obstacles to investigate whether using model checking for real-time planning and obstacle avoidance is reliable on our platform. In the second we extend our approach to support goal-directed obstacle avoidance an again characterise its reliability. We show that it is possible to use model checking to develop an approach to real-time obstacle avoidance which is light on resources, transparent, and suitable for online decision-making.

Contents

List of Tables

List of Figures

Acknowledgements

First I would like to thank Prof. Alice Miller for her guidance throughout the PhD. It has been a long and winding road, and I would not have made it this far without her support.

I would like to thank Dr. Bernd Porr for his unwavering enthusiasm for all things robotics and the many things I have learned from being in his mere vicinity.

I would like thank Prof. Frank Pollick for his swift and noble deeds in my darkest hour.

Of course I would not be here at all without the support of my family.

And finally I would like to thank my cat for all the lovely cuddles.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Chapter 1

Introduction

1.1 Motivation

Autonomous vehicles (AVs), also known as self-driving cars, have been touted as having the potential to bring many benefits to society, such as reduced emissions through optimized route planning and braking, decreased fatality rates through the elimination of human error, and increased agency and autonomy for individuals with poor mobility [151]. Indeed, it has been estimated that 94% of road accidents in the U.S. alone are due to human error [180]. Other benefits include improved productivity and increased trade. In the U.K., for example, it has been forecast that the annual economic benefit of AVs could grow to £51 billion by 2030 [111].

While the potential benefits of autonomous driving to society are wide ranging, they ultimately cannot be realised if the public is unwilling to use the technology. This has prompted research into strategies which promote the widespread adoption of AVs. Consequently, there has been significant research into factors which determine public acceptance (e.g., [41, 113, 174, 210]), though the precise variables which influence acceptance and public perceptions of autonomous driving technology remain ambiguous and poorly understood [152]. It is widely believed, however, that safety and trust are important determinants of adoption [41, 206, 210].

While trust plays a key role in the adoption of AVs, it is also a safety issue in its own right. As trust mediates decision-making in collaborations with automation [87, 118, 155, 156], it is crucial for optimal human-machine performance. In the specific context of AVs, if users do not trust the automation, then they simply will not use it, however trusting the technology too much could lead to over-reliance in critical situations and possibly result in fatal accidents. It is therefore necessary to *calibrate* user trust to appropriately reflect the automation capability and maintain user interest in the technology (e.g., repairing trust after an error to prevent disuse).

One approach generally considered important for calibrating user trust towards AVs is the provision of explanations for vehicle actions [79, 112, 149, 199]. However, users are not the only stakeholders involved in autonomous driving with a need for explanations. Developers, technicians, insurers, and regulators all require insight into AV decisions at different levels of

granularity to support deployment and governance [112, 149]. As state-of-the-art AV navigation systems are often based on deep learning, however, this is a non-trivial task. Deep learning models are inherently opaque "black-box" systems which resist straightforward explanation. Indeed, recent progress in the development black-box artificial intelligence (AI) methods has lead regulating bodies to insist on guarantees of transparency to ensure the proliferation of trustworthy AI systems. In 2024, for example, the European Parliament signed the Artificial Intelligence Act [46] into law, the first legislation globally aimed at regulating the use of AI-enabled technologies.

Regulatory demands for trustworthy AI systems naturally apply to autonomous driving and transparency is a crucial aspect. However, this is problematic for many state-of-the-art AV systems. For example, one recent trend in AV navigation is the use of deep reinforcement learning (DRL) for end-to-end planning and control [9, 38, 126] which is fundamentally a black-box system. The main advantage of end-to-end DRL compared to explainable methods is performance, however the lack of transparency makes oversight and validation of the system difficult. This suggests that achieving equivalent performance using explainable methods is preferable.

While black-box AI methods often demonstrate superior performance in autonomous driving tasks, they nonetheless have important limitations. For example, end-to-end DRL for AVs is based on learning rewards for state/action pairs using trial and error, which for obvious safety reasons is not suitable for online decision-making. Consequently, DRL models are often pretrained in simulated environments, which may provide a significant number of cases for training, but ultimately results in decisions based on rigid generalisations. This ignores important structural information from the immediate environment vital for avoiding imminent collisions. In addition, black-box AI methods are often compute intensive, requiring expensive specialist hardware to ensure the real-time performance of deployed algorithms, which in turn increases the overall energy budget of the system and inflates the cost of AV development [125, 177].

It would be ideal then to develop an approach to AV navigation which is light on compute resources, explainable and suitable for online decision-making, as this is would be optimal for promoting the development of trustworthy AV systems which are cheap to manufacture. In this thesis, we take first steps in this direction. We focus on developing an approach to obstacle avoidance which relies on sensor data for planning to ensure reactivity to imminent situations. This can be achieved using closed-loop input control, which depends on active feedback from the environment and real-time disturbance processing. As a measure to ensure our approach is computationally lightweight, we develop our planning method on a low-powered robotic system.

Closed-loop control, however, can only react to an immediate stimulus, hence it does not have the ability to generate complex plans. This requires the ability to acquire and interpret distal sensor information and reason about possible future outcomes. Model checking [13] is a widely used computer-aided technique for automatically verifying reactive systems, based on a precise mathematical and unambiguous model of possible system behaviour. To verify that the model

meets specified requirements (usually expressed in temporal logic) all possible executions in the model are checked systematically. While typically used for verification of system properties, it has also been proposed as a means of solving AI planning problems [11, 71]. In Pagojus et al. [154], for example, the SPIN [90] model checker was used to plan overtaking manoeuvrers for a simulated autonomous vehicle on a traffic heavy road. We draw inspiration from this approach to develop a real-time planner for generating sequences of closed-loop controllers. As model checking is a symbolic technique, it has the advantage of being inherently explainable.

1.2 Thesis Statement and Contributions

Thesis Statement *Model checking can be used to develop a trustworthy real-time planner for generating sequences of closed-loop controllers for obstacle avoidance which is light on computational resources, inherently explainable, and suitable for online decision-making. We will demonstrate this by developing our approach on a low-powered robot.*

We identified the following contributions:

- The advocacy of model checking, specifically techniques used in SPIN, as an explainable approach for real-time planning in autonomous ground navigation;
- Development of a real-time planning method using model checking for collision avoidance which relies on input control so makes decisions about the immediate driving context;
- Development of a bespoke real-time model checker for planning on a low-powered device.

1.3 Research Questions

We focus on two main research questions with sub-questions derived from the objectives of the research effort, as per the thesis statement in Section 1.2. As model checking is inherently explainable, our aim is to investigate whether model checking can be used to reliably plan sequences of closed-loop controllers for obstacle avoidance in manner suitable for online decision-making on a low-powered robot. Specifically, we focus on addressing three main objectives: (i) to evaluate the reliability of model checking for obstacle avoidance, (ii) to assess the reliability of model checking for real-time planning, and (iii) to describe the memory usage of planning using model checking. Sub-questions for the main research questions are derived from these.

1.3.1 Case Study I: Evasion

In Chapter 4, we conduct an initial investigation into whether model checking can be used for online reasoning on a low-powered robot. Consequently, we address the research question:

> **RQ1** How can we use a bespoke model checker to plan sequences of discrete closed-loop tasks (which have an attentional focus on disturbances) for collision avoidance on a low-powered robotic system in real-time?

The following sub-questions are derived from **RQ1** and the research objectives:

> **RQ2** Does planning using model checking reliably generate more efficient collision avoidance behaviours compared to a baseline method?
>
> **RQ3** Does planning using model checking reliably meet the hard real-time deadline of 100 milliseconds for autonomous driving applications?
>
> **RQ4** What is the memory usage of planning using model checking?

1.3.2 Case Study II: Obstacle Avoidance

In Chapter 5, we extend our initial approach using model checking for real-time obstacle avoidance to accommodate goal-directed behaviour and therefore address the research question:

> **RQ5** How can we extend our bespoke implementation of model checking to plan sequences of avoidance and attraction closed-loop tasks for goal-directed obstacle avoidance on a low-powered robotic system in real-time?

The following sub-questions are derived from **RQ5** and the research objectives:

> **RQ6** Does goal-directed planning using model checking reliabily generate efficient collision avoidance behaviour compared to a baseline method?
>
> **RQ7** Does goal-directed planning using model checking reliably meet the real-time deadline of 100 milliseconds for autonomous driving applications?
>
> **RQ8** What is the memory usage of goal-directed planning using model checking?

1.4 Thesis Structure

We now provide an overview of the remaining chapters:

- **Background** We provide background details relevant to our approach. First we discuss social and regulatory considerations for the development of trustworthy AI, highlighting

how this transfers to the autonomous driving domain. We also provide details on real-time constraints, discuss challenges for achieving this in practice, and highlight the relative merits of common approaches to AV navigation. Subsequently, we provide a discussion of common barriers to AV adoption and model checking.

- **Preliminaries** We introduce definitions and concepts underpinning the use of model checking for real-time planning and obstacle avoidance. We describe our low-powered robotic research platform and motivate definition of an embodied agent able to formulate complex, multi-step plans. Next we introduce the notion of a temporary closed-loop controller as a motion primitive with discrete outcomes which we call tasks and discuss our approach to modelling the physical environment. Last we provide necessary conceptual background on the model checking as planning paradigm.

- **Real-Time Obstacle Avoidance** We describe our initial investigation into model checking for real-time planning and obstacle avoidance by generating sequences of closed-loop tasks. First we explain the details of our methodology. We then provide details on the implementation of our method and summarise preliminary results. Last we provide details on a case study of our approach and draw conclusions on its merits and limitations.

- **Real-Time Goal-Directed Obstacle Avoidance** We describe our investigation into extending our initial approach to support goal-directed obstacle avoidance, in which we use model checking and recursive forward simulation to plan/replan sequences of closed-loop tasks in real-time. We first explain the details of our methodology. We then provide details on the implementation of our method. Last we explain our case study and draw conclusions on the merits and limitations of our approach.

- **Conclusion** We summarise the significance of this thesis. We discuss the value of our approach, analyse limitations/assumptions and describe feasible avenues for future work.

Note that a glossary of variables introduced in this thesis can be found in Appendix B.

Research Artefacts

The following papers were written during my PhD. I am lead author on two of these papers contributor on one. Research from two of these papers is included in this thesis.

- **Chandler, C.**, Porr, B., Miller, A., Lafratta, G. (2023) Model Checking for Closed-Loop Robot Reactive Planning, in *Electronic Proceedings in Theoretical Computer Science*. 395, 77–94. https://doi.org/10.4204/EPTCS.395.6. Introduces real-time model checking for planning sequences of closed-loop tasks for obstacle avoidance. I was responsible for generating the idea, implementing the method, collecting/analysing data,

and writing the paper. Chapter 4 up to and excluding Section 4.5 is derived from this paper. Section 4.5 is directly from this paper.

- Lafratta, G. , Porr, B. , **Chandler, C.** and Miller, A. (2025) Closed-loop multi-step planning, in *Neural Computation* (Accepted for Publication). Introduces physics simulation using a lightweight gaming engine for real-time planning of closed-loop tasks with some learning aspects. Giulia's research aligns closely with my own. I was involved in solidifying formal aspects in the early drafts.

- **Chandler, C.**, Porr, B., Lafratta, G., Miller, A. (2025) Real-Time Model Checking for Closed-Loop Robot Reactive Planning, in *Science of Computer Programming* (Submitted). Extends real-time model checking for planning and obstacle avoidance. I was responsible extending the analysis, collecting/analysing data, and writing the paper. Chapter 4 up to Section 4.5 is based on this paper. So is the evaluation in Section 4.6.

The following links provide access to code and data produced during the course of this thesis:

- Dataset available at https://doi.org/10.5525/gla.researchdata.2012
- Code available at https://doi.org/10.5281/zenodo.15830809
- Videos available at https://www.youtube.com/@ChrisDChandler

Chapter 2

Background

In this chapter, we highlight and discuss literature relevant to the thesis. In Section 2.1, we discuss social and regulatory considerations for the development of trustworthy AI, highlighting how this transfers to the autonomous driving domain. In particular, we focus on the transparency, safety and reliability of AI algorithms. Section 2.2 focuses on real-time constraints for decision-making in autonomous driving and challenges related to achieving this while reducing manufacturing costs. We discuss the relative merits of common approaches to AV navigation. In Section 2.3, we highlight three major barriers to AV adoption related to trustworthiness. Specifically, we discuss the literature on public acceptance, explainability methods and approaches to testing and validation of AVs. We then discuss the relevance of formal methods for trustworthy reasoning in Section 3.5, specifically model checking applied to autonomous robotic systems.

2.1 Trustworthy AI

Recent developments in AI and machine learning promise to bring significant benefits to society and humanity. However, the prospect of human reliance on AI has stimulated concerns over the trustworthiness of AI systems, especially as they are given greater autonomy to make high stakes decisions. As a consequence, there has been a general trend towards promoting the development of trustworthy AI systems from governance bodies, such as the U.S. National Institute of Standards and Technology (NIST) [188]. Perhaps the most robust initiative in this regard has been the publishing of a set of guidelines and principles for the development of trustworthy AI systems from the AI High Level Expert Group (HLEG) of the European Commission [1]. Following on from this, in 2024 the European Parliament signed the Artificial Intelligence Act [46] into law, the first legislation globally aimed at regulating the use of AI-enabled technologies.

The notion of "trustworthiness" in this context departs significantly from common usage of the term. The Oxford English Dictionary defines trust as: *firm belief in the reliability, truth, or ability of someone or something; confidence or faith in a person or thing, or in an attribute of a person or thing* [153]. Trust is therefore an attitude towards a person or artefact, whereas the

related notion of "trustworthiness" denotes a property or set of properties which promote the attitude of trust. Conventionally, we call the trusting agent the *trustor* and the trusted person or artefact the *trustee*. However, in the context of AI systems a more holistic, multi-dimensional definition of trust is often favoured, incorporating specific ethical, social and legal concerns over the potential risks associated with AI decision-making, especially in safety-critical domains.

Following Kuznietsov et al. [112], two broad categories of AI systems can be identified: *symbolic* and *sub-symbolic*. Symbolic approaches rely on encoding information using well-defined mathematical representations, manipulating symbols to draw conclusions based on formal logic or program induction. Historically, this approach was responsible for the first wave of successful AI applications in the form of expert systems [31, 173]. One advantage is that a causal chain of reasoning can be extracted, making expert systems intrinsically understandable by human actors. However, while the manipulation of symbolic representations *prima facie* mimics how human beings reason about the relations between concepts, in practice symbolic AI methods typically require significant human expertise to build domain-relevant knowledge bases.

In contrast, sub-symbolic AI approaches, such as methods involving Deep Neural Networks (DNNs), transform input data to outputs mathematically via the learning of millions of model parameters from large batches of training data [65]. The advantage of this approach is that it is possible to learn complex non-linear functions and achieve human-level performance or better on a variety of tasks (e.g., [30, 179]). However, unlike symbolic methods, the sheer complexity of the relationships between model parameters makes it impossible to understand how DNN-based methods arrive at decisions [112]. Nonetheless, advances in DNNs have been at the forefront of recent progress in AI. As greater autonomy is given to AI to make critical decisions, it is crucial that frameworks for assessing its trustworthiness are in place.

The European Commission AI HLEG identified four ethical principles and seven requirements for the development of trustworthy AI [1]. The principles are: (i) respect for human autonomy; (ii) prevention of harm; (iii) fairness; and (iv) explicability. From these principles the following requirements were derived: (i) human agency and oversight; (ii) technical robustness and safety; (iii) privacy and data governence; (iv) transparency; (v) diversity, non-discrimination and fairness; (vi) societal and environmental well-being; and (vii) accountability. The recent NIST risk management framework for AI [188] shows a similar range of concerns. Figure 2.1 provides an overview of the framework suggested in the AI HLEG ethical guidelines.

Trustworthy AI from the perspective of governance agencies is therefore socio-technical, based on ethical, social and legal norms. However, while interaction with human actors is generally covered, none of these guidelines elaborate on the specific factors influencing trust in AI systems at the level of individual human beings. As trust mediates decision-making in collaborations with automated systems [87, 118, 156], it is crucial for both safe interactions with automation/AI and optimal human-machine performance. In the literature, this is known as the

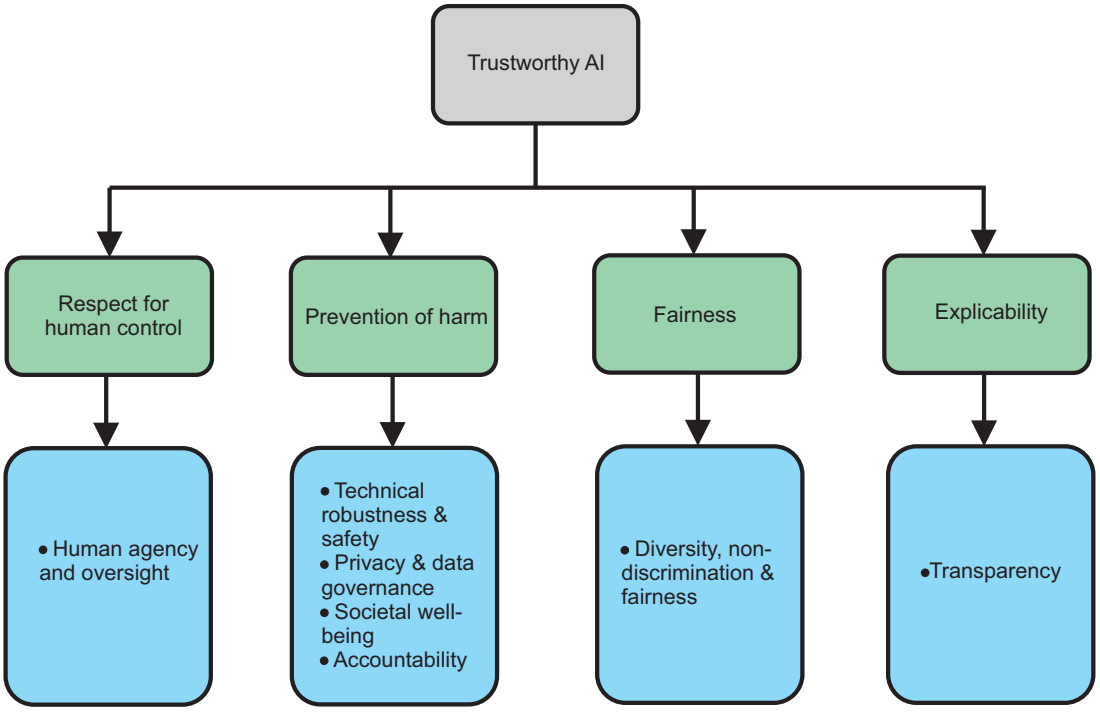

Figure 2.1: Overview of AI HLEG guidelines [1, 105]. Four guiding ethical principles from which seven requirements for the development of trustworthy AI systems are derived.

problem of trust calibration. If users trust an automated system too much or too little, then this can lead to misuse or disuse of the system, i.e., because user expectations (and emergent reliance behaviours) do not match the capability of the automation [155]. In the best case scenario, this can result in sub-optimal human-machine performance, however in safety-critical applications (e.g., autonomous driving) poor trust calibration can have severe (possibly fatal) consequences.

2.1.1 Human Factors

While there has been significant research in the human factors literature on trust in automated systems, the notion of trust (and related notion of trustworthiness) has been notoriously difficult to define. In the early 2000s, Mayer et al. [137] examined the antecedents and outcomes of organizational trust, proposing the following definition: *the willingness of a party to be vulnerable to the actions of another party based on the expectation that the other will perform a particular action important to the trustor, irrespective of the ability to monitor or control that party*.

The model in [137] highlighted three antecedents of perceived trustworthiness: ability, benevolence and integrity. Ability refers to the trustee's competence for a task in a given domain, benevolence refers to the trustee's benign motives, and integrity involves the trustor's perception that the trustee adheres to a set of acceptable principles of conduct [137]. However, while influential in human factors research, the model developed by Mayer et al. was focused on human-human trust in organisations, not specifically on human trust in automated systems.

Around the same time, Lee and See [118] supplied a definition popular in human factors

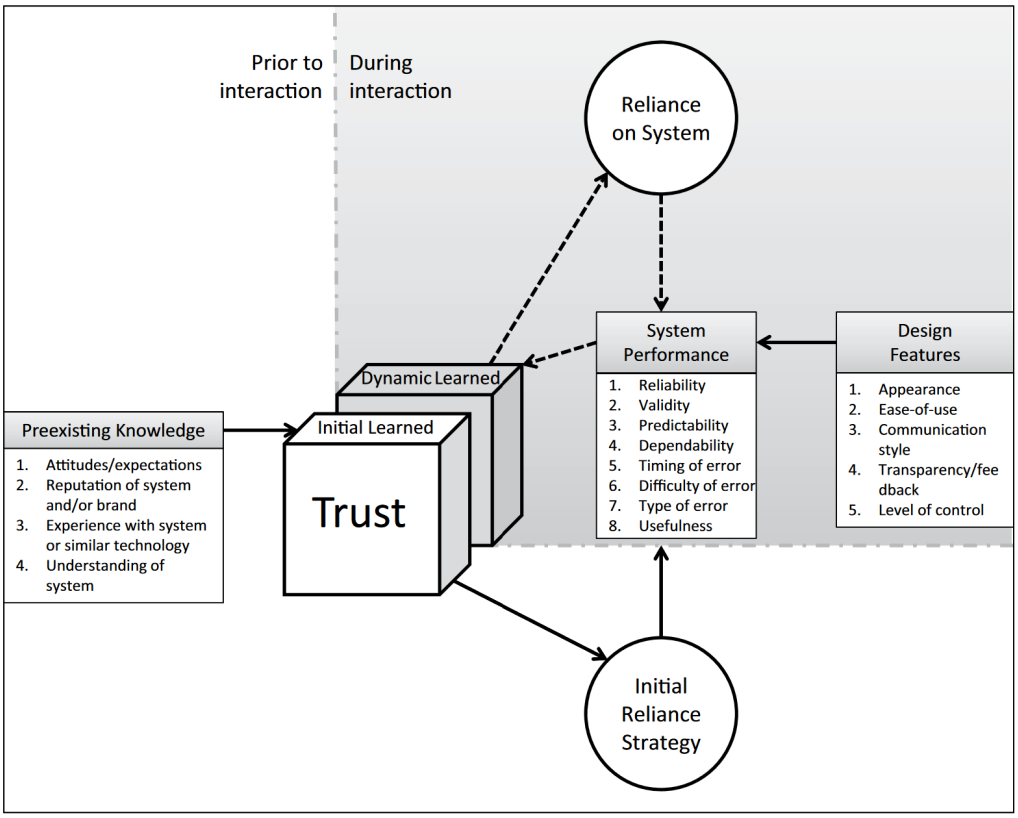

Figure 2.2: Factors influencing learned trust. Dotted arrows represent dynamic aspects which can change during a single interaction. Reproduced from [87].

research characterising trust as *the attitude that an agent will help achieve an individual's goals in a situation characterised by uncertainty and vulnerability*. Here the focus was on how to design human-machine interfaces for appropriate reliance (i.e., the emergent behaviour of trust) on a system in supervisory control interactions, i.e., human-in-the-loop oversight of automated system performance. However, the definition in [118] is too general to capture the multi-dimensional character of trust and guide empirical research efforts. Nonetheless, it has been influential, forming the basis of an oft-cited framework for investigation of trust in machines.

Hoff and Bashir [87] conducted a large meta-analysis and review of empirical research into trust in automation, decomposing the concept of trust into dispositional, situational and learned aspects. *Dispositional* trust denotes static attributes of individuals which vary little between situations (e.g., gender, age, culture, etc.), *situational* trust refers to environmental and cognitive factors, and *learned* trust comprises both background knowledge of systems and processes for dynamic changes in trust from experience of machine performance (see Figure 2.2).

While the Hoff and Bashir model has been criticised for its lack of empirical validation, its characterisation of the multi-dimensional nature of trust in automated systems (e.g., AI systems) has been fruitful. In the human factors literature, significant research has made use of this framework. For example, Ingram et al. [95] investigated the effects of task difficulty on

dynamic learned trust in an image classification task. Daronnat et al. [48] created multiple linear regression models to determine which combinations of factors led to the best predictions of users' trust ratings using the model in [87] to generate inclusion criteria. In a recent review arguing for a combined empirical and ethnographic approach to trust in AVs, Raats et al. [161] used the model to organise which aspects of trust empirical research has investigated.

Although the model in [87] captures the multi-dimensional character of trust, it is focused on human operators as supervisors of automation and therefore much of the research it has inspired manipulates machine level antecedents of trust to promote better calibration assuming human oversight. However, technological advances are in general leading to new interaction paradigms, with human operators in more peer-like relationships with automation, suggesting a focus on bidirectional human-machine collaboration would be appropriate. In a recent review of research into the perception of machines as teammates, Rix [168] argued along similar lines, concluding that the creation of a team setting and social entity, along with configuration of machine collaborative behaviours, are the main drivers of successful human-machine teams.

The Hoff and Bashir model was not aimed specifically at autonomous driving, but it has been frequently cited in studies investigating human trust in the AV domain (e.g., [79, 83, 86, 161, 210]). Its relevance, however, has been challenged. In a recent study, Walker et al. [199] argued that there is a lack of clarity regarding the relationship between situational and learned aspects, which could lead to inconsistent use between researchers and cause confusion. To address this, Walker et al. proposed a simpler framework with revised terminology, distinguishing between *expectations* (dispositional and initial learned trust) and *calibration* (situational and dynamically learned trust—all experiences related to interacting with the system across various situations). According to this framework, trust calibration is influenced by prior knowledge and system characteristics (e.g., transparency/feedback), but varies in a feedback cycle with experience of AV behaviour, dynamically adjusting the user's mental model of the AV system's reliability.

2.1.2 Transparency

As should be clear, the notion of "trustworthiness" has no precise unilateral definition. Its intended meaning varies according to the context of use and application. Furthermore, as evidenced by varying perspectives in human factors research, within a given domain uncertainty over its accurate characterisation remains a persistent challenge. From the perspective of regulating bodies, such as NIST, the notion of trustworthiness is normative, hence a stipulative definition may be sufficient. However, for empirical research into trust in automated/AI systems which aims at objectivity, confusion over the correct notion of trustworthiness is problematic. Nonetheless, common themes about system and performance characteristics essential for promoting the trustworthiness of automated/AI systems emerge, such as system transparency.

As mentioned above, transparency is one of the key requirements of the AI HLEG guidelines for trustworthy AI, which implements the core ethical principle of explicability. This is

conceived as crucial for building and maintaining users' trust in AI systems [1]. Specifically, the transparency requirement comprises three aspects: traceability, explainability and communication. *Traceability* requires developers of AI systems to document the data sets and processes used in development of the AI to enable identification of reasons for possible erroneous decisions. *Explainability* requires that decisions made by an AI system can be understood and traced by human beings, hence there is some overlap with the traceability requirement. *Communication* requires AI systems to make it clear to users that they are not interacting with a human.

While the concerns mentioned here are normative, encompassing general development and deployment practices as well as AI system features, both the Hoff and Bashir model and suggested simplification by Walker et al. stress the importance of system transparency and system feedback as one factor important for the calibration of subjective human trust [87, 199]. The design of methods able to achieve this is generally the purview of the field of eXplainable AI (XAI), though the notion of transparency endorsed is typically narrower than the transparency requirement outlined in the AI HLEG guidelines. Indeed, in lieu of the upcoming EU Artificial Intelligence Act, Gyevnar et al. [77] argue that, while XAI can address many of the transparency concerns raised in the legislation, there is a "transparency gap" between legal conceptions of AI transparency and current XAI approaches which risks leaving important aspects unaddressed.

Nonetheless, the provision of explanations for system decisions is considered important for trust in automated/AI systems, such as AVs [112, 149]. In a recent study, for example, Ha et al. [79] showed that explanation type significantly affects trust in AVs and that the perceived risk of driving situations significantly moderates the effects of explanation type. In an earlier study, Koo et al. [108] found that "how" explanations (e.g, "The car is braking") lead to poor driving performance, whereas "why" information describing the reasons for actions (e.g, "Obstacle ahead") was preferred by drivers and lead to better driving performance. "How and why" explanations resulted in safest performance but increased negative affect towards the vehicle.

However, end-users are not the only stakeholders with a need for transparency in autonomous driving [149]. AI decisions in this context should in principle be explainable to all stakeholders involved in the deployment of AV technology. For example, end-users who lack domain expertise may be satisfied with a simple explanation that requires less background knowledge to interpret, but developers, regulators and insurers need a more detailed explanation with technical terms that support a deeper understanding and analysis of AI decision-making [112, 149].

Consider for example the Molly problem[1] described by the International Telecommunication Union (ITU), an adaptation of the Trolly problem where a young girl Molly is hit by an unoccupied AV without any witnesses present. The accident investigation process can be aided by explanations containing the vehicle's observations, the road rules and traffic signs it acted on. Hence explanations are considered necessary for both calibrating end-user trust and auditing of vehicle actions. Vehicle decision-making should therefore be transparent in principle to

[1] https://www.itu.int/en/ITU-T/focusgroups/ai4ad/Pages/MollyProblem.aspx

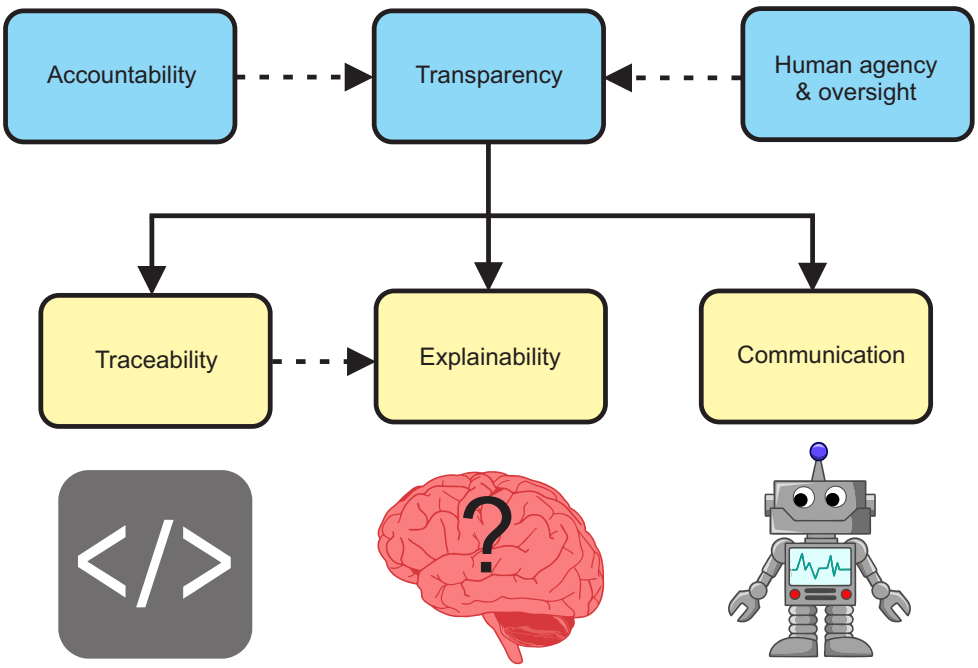

Figure 2.3: Transparency sub-requirements from AI HLEG guidelines. Traceability includes documentation of datasets and processes used in development of the AI and the ability to explain decisions post-hoc to satisfy related accountability requirement (e.g., for accident analysis).

promote accountability and governance in the AV ecosystem. This is reflected in the AI HLEG requirement for traceability which enables the related requirement of accountability. Figure 2.3 illustrates sub-requirements for transparency as detailed by the AI HLEG ethical guidelines.

2.1.3 Safety and Reliability

Transparency is therefore a broad concept which aims to promote governance and oversight of AI-enabled technologies, generally considered important for the trustworthy integration of AI into the fabric of modern society. This is particularly important for safety-critical application domains where erroneous decisions could lead to physical or psychological harm. Indeed, the desire to mitigate potentially harmful aspects of AI is reflected in the AI HLEG guidelines via the core ethical principle of prevention of harm, implemented in part by the requirement for technical robustness and safety [1]. Specifically, four sub-requirements are promoted: resilience to attack and security, fallback plan and general safety, accuracy, reliability and reproducibility.

Figure 2.4 provides an overview of the sub-requirements. *Resilience to attack and security* is concerned with protecting the AI system against vulnerabilities which could be exploited by malicious actors to influence decision-making. *Fallback plan and general safety* aims at (i) mitigating harmful consequences in the event of AI system failures, and (ii) ensuring the AI functions as intended without causing harm to human beings or the environment. This includes processes for identifying and assessing risks associated with use of the AI. *Accuracy* concerns the ability of an AI to make correct decisions based on data or models. *Reliability and repro-*

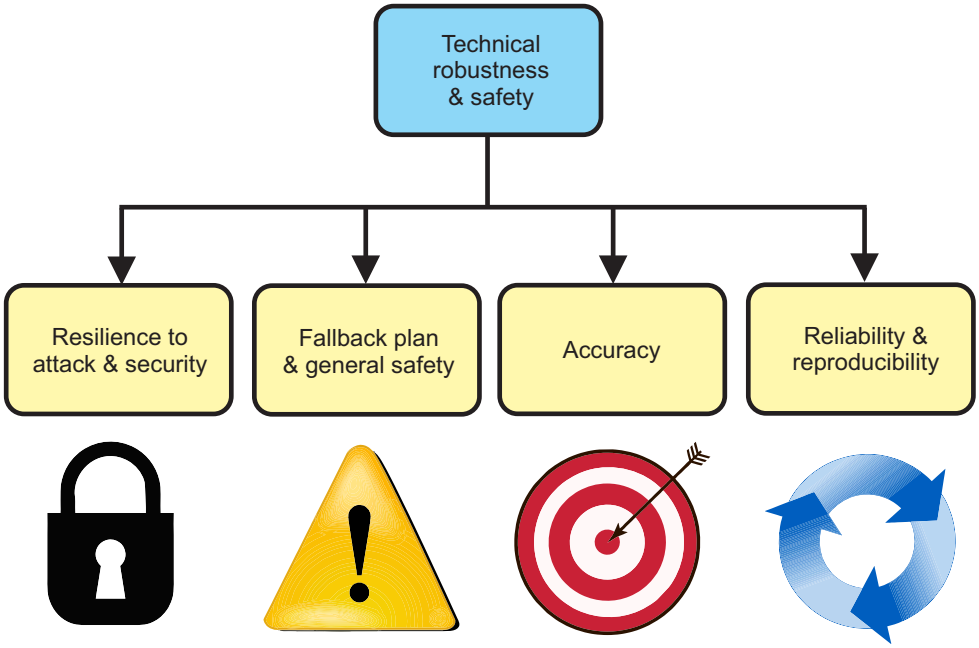

Figure 2.4: Technical robustness and safety sub-requirements from AI HLEG guidelines.

ducibility concerns (i) continuity of decision-making for a range of inputs and situations, and (ii) whether the AI exhibits the same behaviour when repeated under the same conditions.

Aspects of this requirement are echoed in human factors research into trust in automation/AI. As indicated in Figure 2.2, the top performance factor influencing the formation of dynamic learned trust in Hoff and Bashir [87], based on a large-scale meta-analysis of empirical research into trust in automation, is reliability of the system. Indeed, the primacy of reliability in the formation of human-machine trust is well documented (e.g., [57, 155, 198]), with findings generally suggesting that higher reliability of the automated system is associated with higher levels of trust in the automation. However, this is not observed in all instances, mediated by factors such as the nature of the task and level of autonomy [15]. In safety-critical domains, such as autonomous driving, characterising the reliability of the system and how this affects human-machine trust formation is important for developing safe and trustworthy AI systems.

The technical robustness and safety requirement advanced by the AI HLEG guidelines, however, focuses more on assurance of AI-enabled systems. This is often a non-trivial task, especially for AI approaches which make inherently opaque decisions, mediated by the characteristics of the use case. For example, Kalra and Paddock [98] demonstrated the infeasibility of empirical validation of AI-enabled systems in autonomous driving, arguing that an AV would have to drive hundreds of millions of miles to achieve statistical confidence in its reliability, owing to the relative rarity of collisions over the typical lifetime of a vehicle. Other validation methods are therefore necessary for AVs. A holistic approach to validating reliability (which in the context of autonomous driving is a safety-critical issue) is typically advocated by governance agencies [1, 188], potentially comprising a variety of methods, such as simulation,

scenario-based testing and empirical validation. These aspects can then be organised into some kind of workflow which aims to accumulate multi-dimensional evidence of the vehicle's safety.

According to the NIST risk management framework for AI [188], validity and reliability are necessary conditions for the trustworthiness of AI systems. Major contributing factors to validity are accuracy and robustness, where accuracy concerns the closeness of obtained results to ground truth and robustness is effectively defined as generalisability, i.e., the ability of an AI system to maintain adequate performance across a variety of scenarios. Reliability is conceived as: *a goal for overall correctness of AI system operation under the conditions of expected use and over a given period of time, including the entire lifetime of the system* [188]. In the context of autonomous driving, this includes the repeated accuracy of the AI system in specific tasks (e.g., collision avoidance), but also the operating milieu of the system, which in addition to accuracy includes hardware, computational and task-specific constraints. For example, an AI-enabled system for AV navigation should be able to respond reliably to imminent events in high-speed driving but also achieve this with minimal energy consumption to drive down costs [103, 126].

2.2 Autonomous Driving

Autonomous driving has been touted has having the potential to bring many benefits to society, such as increased safety due to the elimination of human error, better mobility for disabled individuals, and reduced emissions through optimized route planning and braking [151]. To completely realise these benefits, it is necessary is to achieve full autonomy, however there are many safety-critical challenges to be overcome. Consequently, manufacturers have focused on the incremental introduction of automated driving functions to better manage the risk. In recognition of this, SAE International [176] classified automated driving systems (ADS) into six levels of autonomy, broadly referred to in the AV domain and adopted by regulatory agencies such as the U.S. National Highway Traffic Safety Administration (NHTSA). Figure 2.5 provides an overview of the automation levels and how they relate to the dynamic driving task (DDT).

The most advanced ADS commercially available at the moment is DRIVE PILOT[2] installed on the Mercedez-Benz S-Class and EQS Sedans. The technology is marketed as L3 conditional automation, where the driver may disengage from the DDT and divert attention to a non-driving related task, but should be ready to take over control of the vehicle when prompted by the system, within a restricted operational driving domain (ODD). However, L5 full automation has not yet been achieved at a scale sufficient for commercialisation. One reason is the challenge of meeting the requirement for real-time performance while keeping manufacturing costs low.

Following Lin et al. [125] on performance constraints, the reaction time of an ADS is determined by two factors: *frame rate* and *processing latency*. Frame rate determines how fast the real-time sensor data can be fed into the process engine, while the processing latency of rec-

[2] https://www.mbusa.com/en/owners/manuals/drive-pilot

Level	Name	Definition	DDT		DDT fallback	ODD
			Sustained lateral and longitudinal motion control	OEDR		
L0	No Driving Automation	Driver performs the entire DDT, even when enhanced by active safety systems (e.g., emergency braking).	Driver	Driver	Driver	n/a
L1	Driver Assistance	Sustained and ODD-specific execution by an ADS of either the lateral or the longitudinal vehicle motion control subtask of the DDT (but not both simultaneously) with the expectation that the driver performs the remainder of the DDT.	Driver and System	Driver	Driver	Limited
L2	Partial Driving Automation	Sustained and ODD-specific execution by an ADS of both the lateral and longitudinal vehicle motion control subtasks of the DDT with the expectation that the driver completes the OEDR subtask and supervises the ADS	System	Driver	Driver	Limited
L3	Conditional Driving Automation	Sustained and ODD-specific performance by an ADS of the entire DDT with the exception that the DDT fallback-ready user is receptive to requests to intervene, as well as to DDT performance-relevant system failures in other vehicle systems, and will respond appropriately.	System	System	Fallback-ready user (becomes the driver during fallback)	Limited
L4	High Driving Automation	Sustained and ODD-specific performance by an ADS of the entire DDT a DDT fallback without any expectation that a user will need to intervene.	System	System	System	Limited
L5	Full Driving Automation	Sustained and unconditional (i.e., not ODD-specific) performance by an ADS of the entire DDT and DDT fallback without any expectation that a user will need to intervene.	System	System	System	Unlimited

Figure 2.5: SAE levels of automation.

ognizing scenes and making operational decisions determines how fast the system can react to the captured sensor data. The fastest recorded action by a human driver takes 100-150 milliseconds [148, 192] which suggests a processing latency of less than 100 milliseconds for better than human performance. In addition, to react quickly to changing traffic conditions in real-time, the system should update its scene understanding at a suitably high frequency, suggesting a frame rate of 100 milliseconds as the traffic situation could change drastically between frames. However, real-time performance in practice often requires significant compute resources.

2.2.1 Computing Systems

There are two main approaches to the development of an ADS based on the AI learning architecture: modular and end-to-end pipelines [177]. Modular pipelines execute operations in sequence from module-to-module to arrive at a control command, whereas end-to-end pipelines generate control commands directly from the raw sensor data. Each approach has its advantages and disadvantages with respect to the safety and reliability of decision-making for AV systems.

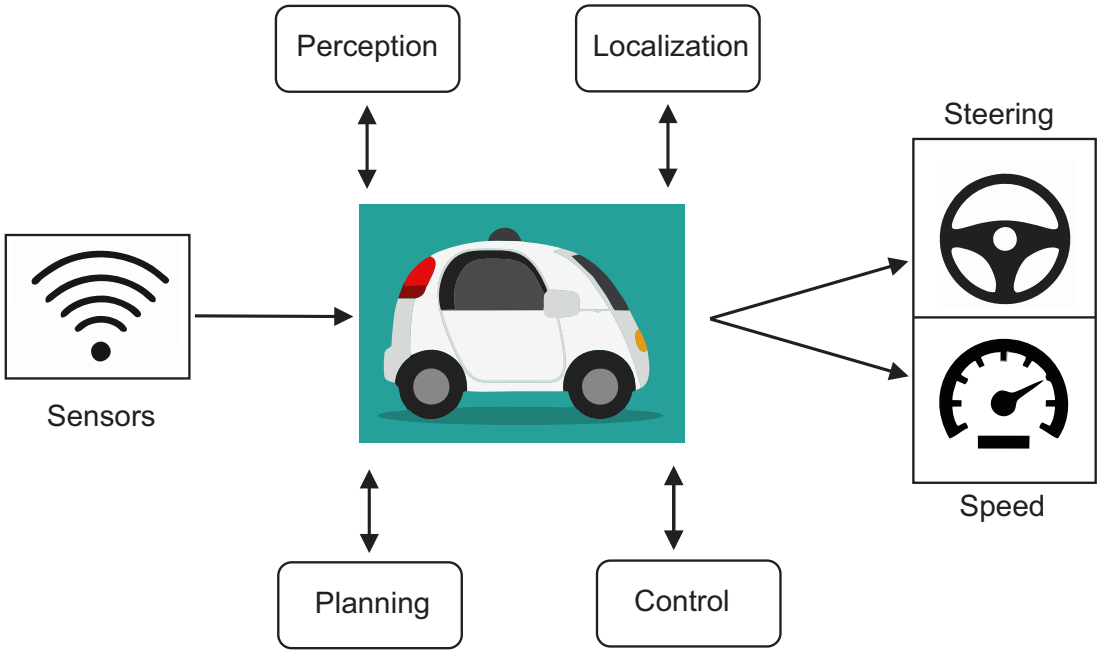

Figure 2.6: Typical pipeline for modular ADS.

Figure 2.6 shows the typical architecture of a modular pipeline. Modular design decouples sensing, perception, localization, planning, etc., which has the advantage of making it easier for teams of developers with different areas of expertise to work together and debug systems [104, 177, 208]. In addition, new functionality can be integrated with existing components without modifying their inner workings, e.g., the implementation of hard-coded safety constraints on top of the planning module. However, there are typical bottlenecks in modular AV systems (e.g., perception [123, 125]) which are a challenge for building real-time systems. Modular pipelines are also prone to error propagation [138] which can lead to unexpected behaviours.

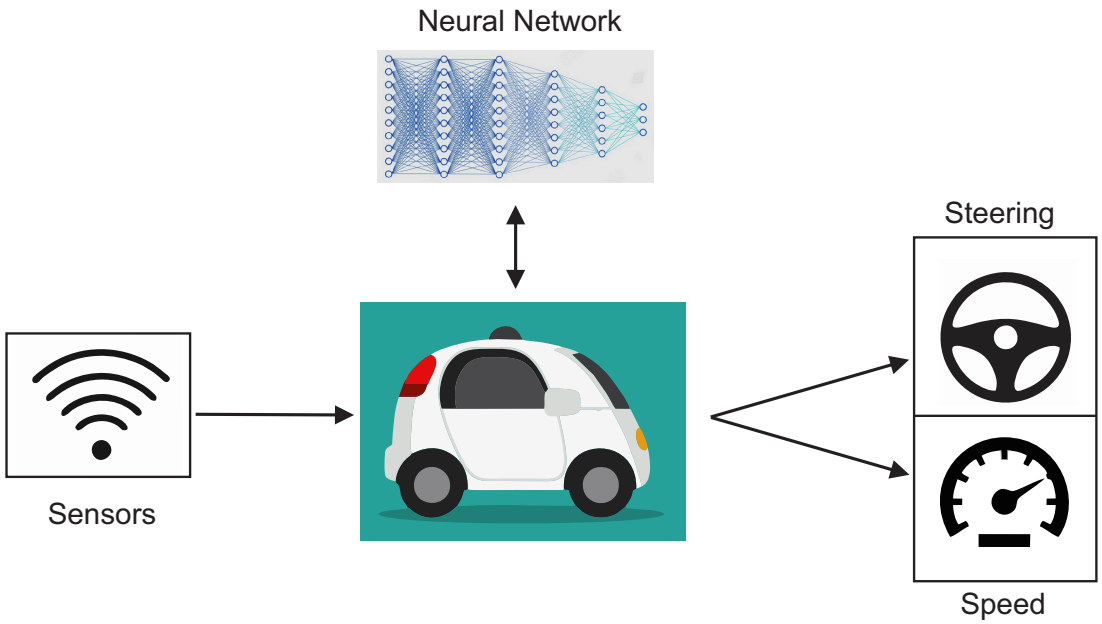

Figure 2.7: Typical pipeline for end-to-end ADS.

In contrast, end-to-end pipelines do not require interaction between system components, so error propagation is not a similar concern. Figure 2.7 shows the basic architecture of an end-to-end pipeline. In the modular case, AI algorithms may be involved in specific tasks, such as object detection in the perception module [126], however in end-to-end systems they are applied to generate control commands directly from raw sensor data. In comparison to the modular approach, the development of end-to-end pipelines requires specific expertise in machine learning and the AV domain. However, once an AI model is trained and deployed to an AV, it can respond within real-time constraints and has the potential to reach human-level accuracy or better [40]. In general, the end-to-end approach streamlines data processing by eliminating redundant calculations, thereby increasing efficiency and robustness.

2.2.2 Real-Time Challenges

Specialist hardware is often required for real-time performance using either approach [125]. One reason is the sheer volume of real-time data generated from a suite of sensors (e.g, LiDAR, radar, camera). It has been estimated that an AV could generate 20-40 terabytes of data in a single day, putting significant demand on the vehicle computing system. For example, one key sensing technology implemented on AV systems is LiDAR, which can generate 10-70 MB data per second, a massive amount of data for the computing platform to process in real-time [126, 177]. In practice, however, each sensor modality has its limitations, hence fusing data from various sensors is often necessary, increasing the computational burden yet further [19, 136].

Another reason specialist hardware is often required is AI algorithms [125]. To accelerate the performance of popular DNN-based approaches, either as components of a modular system

(e.g., object detection) or in end-to-end systems, GPU-accelerated hardware is often necessary. However, these computation units often consume a lot of power and generate significant heat, consequently putting extra demand on the heat dissipation system. For example, one of the top GPU-accelerated computing units for AVs is the NVIDIA PX2 which consumes a sizeable 80 Watts of power [19]. For electric AV systems in particular, this can mean a significant reduction in driving capacity, which has lead some researchers to develop more efficient algorithms for AV tasks. For example, Malawade et al. [136] developed an energy-aware adaptive sensor fusion method, which uses approximately 60% less energy and achieves 58% lower processing latency on the industry-standard NVIDIA PX2 platform compared to other fusion approaches.

The energy consumption of AVs is further impacted by storage requirements. While consumption by the computing unit is dependent on the computing hardware architecture (e.g., CPU, GPU, etc.), energy demands for storage are predictable. According to Lin et al. [125], a typical storage system consumes around 8 Watts to store every 3 terabytes of data. As GPS technology cannot locate a vehicle with sufficient precision, state-of-the-art AV systems typically use an offline map, which for safety and reliability needs to be stored locally, as internet connectivity cannot be guaranteed at all times. It has been estimated that a typical map of the U.S. requires 41 terabytes of storage [125], putting energy consumption at around 110 Watts. In addition, requirements for auditability from regulators may require the storage of large amounts of structured and unstructured real-time data for post-hoc analysis in the event of accidents.

Real-time performance constraints therefore have a direct impact on the computing hardware architecture and energy-efficiency of an ADS, which ultimately drives up manufacturing costs. According to Liu et al. [177], a typical non-luxury vehicle in the U.S. usually costs around $30,000 to manufacture, whereas the cost of a prototype AV has been estimated at around $250,000, largely due to the additional hardware requirements for AV systems. In addition, commercial deployment may require onboard backup systems to ensure system reliability in the event of failures. Maintenance costs (e.g., insuance, repair) are also likely to be high [177], which could make AVs unattractive for consumers. It is therefore necessary to achieve real-time performance while minimising energy consumption to drive down costs, otherwise widespread commercial availability is economically infeasible for manufacturers. This will require innovation in efficient hardware architectures, and efficient computational methods for AV tasks [127].

2.2.3 Navigation Methods

The most complex task is real-time planning with obstacle avoidance [165, 208] as this requires the seamless integration of all aspects of the system. In addition to constraints on processing latency and energy efficiency, there are a number of characteristics of the driving environment which add to the complexity. For example, from the perspective of an embodied agent (e.g., an AV), the environment is partially observable, so it is impossible to have complete knowledge of obstacles using data from sensors alone. This is especially problematic in complex urban

environments, where vehicles and other road users could be occluded by visible objects. In addition, the environment consists of dynamic objects (i.e., vehicles and other road users), hence the ability to predict future actions becomes an essential component of avoiding collisions. Safe and reliable path planning for AVs is therefore a non-trivial task with many specific challenges.

Authors often distinguish between global and local planning in the autonomous driving literature [165, 191]. Global planning typically focuses on calculating a high-level optimal route from a start location to some destination on an offline map to achieve long-term navigation goals, whereas the focus of local planning is on real-time navigation and obstacle avoidance, i.e., responding reactively to obstacles while making progress towards a goal. Our focus in this thesis is on local path planning and obstacle avoidance. Many approaches have been taken to address the local planning and obstacle avoidance problem for autonomous driving (for a comprehensive survey see [165]). These can be grouped into traditional planning and learning-based methods.

Traditional Planning

Some traditional methods for local path planning in autonomous driving are illustrated in Figure 2.8. One popular approach is graph-based methods, which normally partition a prior map of the environment into a discrete representation to form a connected graph. An optimal or near-optimal path to a goal location can then be extracted while avoiding obstacles. Dijkstra [56] and A* [82] are two well-known graph algorithms (see Figures 2.8A and 2.8B). However, the real-time performance of graph-based methods is limited, as there is a trade-off between granularity of the discrete representation and real-time performance due explosion of the state space. In addition, without modification, graph-based planners produce jerky trajectories [165], which cannot be executed comfortably by a vehicle with non-holonomic constraints. Several modifications of graph-based planners have been proposed in the literature to address this, often combining algorithms such as A* with reactive obstacle avoidance (e.g., the dynamic window approach [214]) or post-hoc trajectory optimization (e.g., conjugate gradiant smoothing [143]).

Sampling-based methods, such as Rapidly-exploring Random Trees (RRT) [116], are another popular approach to local planning and obstacle avoidance. The main idea behind sampling-based methods is to construct connections between randomly sampled points in the obstacle-free configuration space until the goal is reached (see Figure 2.8C). As the approach is probabilistically complete, given enough time, a path is guaranteed to be found [116, 150, 165]. For example, Feraco et al. [62] used RRT to plan local paths for a prototype racing car, generating reliable paths for unknown environments. However, due to randomness, results may not be replicable, and optimal solutions may not be found [45]. This has been address through extensions such as RRT*, however optimality is dependent on the time the planner has for sampling the environment. While typically fast to converge on a solution, similar to graph-based methods, the path is discrete, resulting in jerky trajectories [72]. As for graph-based approaches, several RRT modifications have been suggested to smooth generated trajectories (e.g, [92, 196, 212]).

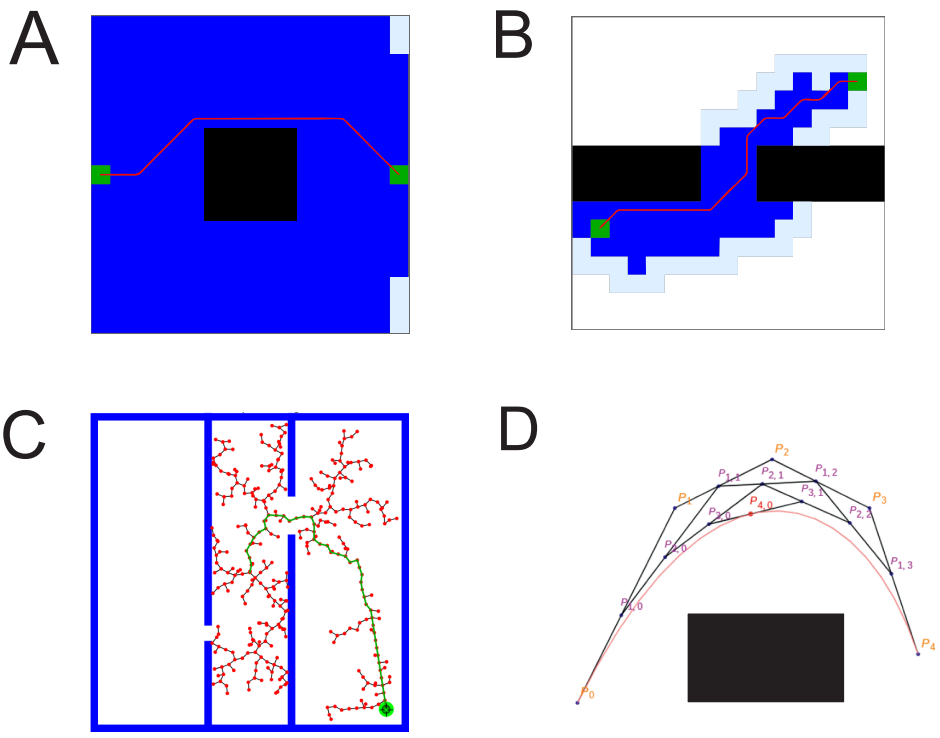

Figure 2.8: A: Dijkstra graph algorithm avoiding a single obstacle [172]. B: A* graph algorithm traversing a narrow gap [172]. C: RRT algorithm negotiating a narrow gap [20]. D: Bézier curve interpolation using de Casteljau's algorithm [190] to calculate a smooth trajectory.

Interpolating curve planners are often used for smoothing the trajectory between discrete states on a planned path, taking into consideration paramaters and constraints such as feasibility, comfort and vehicle dynamics [72]. There are various possible techniques, each with specific advantages and disadvantages. Figure 2.8D shows one example of a fourth degree Bézier curve using de Casteljau's algorithm, which generally has low computational cost, though this increases with the curve degree [72]. Other methods include polynomial and clothoid curves, however in general interpolating curve planners depend on a prior planned path and can be time consuming when faced with immediate obstacles in real-time. Numerical optimization, which aims to minimize or maximize a function subject to different constrained variables, is an alternative approach for generating smooth trajectories. The main advantage is that road and vehicle constraints can be easily incorporated, as well as other road users. For example, control barrier functions have been used to guarantee that generated trajectories are within a safe set [181]. However, again it relies on a prior planned path. In addition, as optimization takes place at each continuous motion state, achieving energy-efficient real-time performance is challenging.

Learning-Based Planning

One benefit of traditional approaches is that they are easy to understand by human actors. However, to address issues with performance, in recent years there has been an increasing interest in machine learning approaches based on DNNs, which in comparison are opaque. Traditional

planning methods are typically deployed using a modular pipeline, however learning-based methods can be used with either architecture. In particular, there has been surging interest in machine learning for end-to-end AV navigation systems [165]. There are two basic approaches to the use of machine learning: imitation learning (IL) and reinforcement learning (RL).

IL is a supervised learning approach which learns from expert demonstrations of driving behaviour to generate a driving policy; this requires a dataset containing trajectories collected under the expert's driving policy, where each trajectory is a sequence of state-action pairs [38, 40]. One approach is behavioural cloning [14], which, to minimise imitation loss with the expert policy, assumes that state-action pairs are independently and identically distributed (i.i.d.), as is typical in supervised learning. However, as outputs of the model interact with the environment, there may be correlations between sequential predictions, violating the i.i.d. assumption and thus limiting generalisability. Direct policy learning (e.g., [170]) is one alternative, which aims to directly learn a driving policy, mapping inputs to actions instead of control commands. The main drawback is that continuous access by an expert is needed during training which is expensive and inefficient [40]. Inverse reinforcement learning aims to address this by deducing expert driving behaviours through a reward function, extrapolating from collected example behaviours to find the underlying reward. However, there are many drawbacks to this approach: e.g., models are expensive to train, often unstable, and computationally demanding at runtime [38, 40].

One major issue with IL is handling cases that are out-of-distribution, given its dependence on offline datasets of expert behaviour, which cannot cover all possible scenarios. For this reason, in recent years there has been an increased interest in RL-based methods, trained on the basis of interactions with simulated vehicle environments [40]. The goal of an RL agent is to maximize its cumulative numerical reward through trial and error interactions with the environment [186], often represented using a discrete grid. This is a closed-loop feedback cycle, where feedback about an action is provided in retrospect after a training episode. RL can be formulated as a Markov Decision Process, but many recent approaches in autonomous navigation for AVs have either augmented RL with DNNs or taken a model-free approach (e.g., [35–37]). However, while increased accuracy has been achieved, there is a "reality gap" between models trained in simulated environments and real world applications [38, 50, 91]. Ultimately, RL models learn rigid generalisations, based on simple reward structures [38], which may lead to risky behaviours. In addition, as mentioned above, approaches based on DNNs require accelerated hardware for real-time performance, which increases the energy demand and cost of the system.

2.3 Barriers to Adoption

There is therefore a need for planning and obstacle avoidance methods which are resource efficient, as this will help drive down the cost of implementation and contribute towards making fully autonomous driving commercially feasible. However, there are further barriers to the adop-

tion of fully autonomous driving related to the trustworthy AI issues discussed in Section 2.1.

2.3.1 Public Acceptance

One issue is public acceptance of the technology. As mentioned above, autonomous driving has been touted has having the potential to bring many benefits to society, however the benefits of the technology can only be realised if users are willing to adopt AVs. While there has been significant research investigating the factors determining acceptance (e.g., [110, 142, 189]), the precise drivers underlying the willingness of individuals to adopt AVs remain ambiguous [2]. Nonetheless, common themes have emerged regarding public opinion on autonomous driving.

Kyriakidis et al. [113], for example, surveyed 5000 respondents using an online questionnaire across 109 countries, which has been well-cited in the AV adoption literature. In general, the results were mixed on opinions towards AVs, with some respondents viewing autonomous driving as positive, and others not considering it an enjoyable experience compared to manual driving. Respondents were primarily concerned by security, safety, legal aspects and automatic data transmission—high income countries in particular were negative towards automatic transmission of data to insurers and tax authorities [113]. Big Five [97, 163] personality factors did not have much influence, though respondents who scored high on neuroticism were slightly more concerned about automatic data transmission, and those who scored high on agreeableness less.

Schoettle and Sivak [174] studied public opinion on AVs in China, India, Japan, the USA, the UK, and Australia. A survey was conducted in each country with approximately 500 respondents for each survey. The majority of respondents had heard about AVs and had a positive general opinion, except in Japan which was majority neutral (50.3%). The most positive opinions about AVs were observed in China (87.2%) and India (84.2%) and the most negative opinions in the U.K. (13.7%) and U.S. (16.4%). However, the major concern across all countries was related to the safety of fully autonomous vehicles, as summarised in Table 2.1. In particular, survey participants were concerned about using AVs where no driver controls are available at all, especially in the U.S. where 60.1% of respondents were “very concerned” [174].

Response	**China**	**India**	**Japan**	**U.S.**	**U.K.**	**Australia**
Very concerned	68.0	59.0	31.1	51.1	44.8	44.4
Moderately concerned	27.7	27.5	45.8	30.7	36.8	34.3
Slightly concerned	3.8	10.8	19.5	14.6	14.6	17.4
Not at all concerned	0.5	2.7	3.6	3.6	3.8	4.0

Table 2.1: Responses to concern over safety consequences of equipment/system failure for fully autonomous vehicles according to country, reproduced from Schoettle and Sivak [174].

Indeed, concerns over the safety of fully autonomous vehicles have been a dominant theme in the results of numerous survey studies on public attitudes towards AVs [93, 113, 174]. Further-

more, several survey studies have identified males and young adults as potential early adopters of autonomous driving [4, 93, 113, 144, 174], owing to a greater appetite for risk in driving and pedestrian behaviour [89, 169, 194]. It has also been found that respondents with higher education and income living in densely populated urban areas are more likely to have a positive view of AV technology [16]. However, the evidence from survey studies is correlational, hence it provides no insight into the causal factors driving public acceptance of AVs, which remain poorly understood. Nonetheless, it is widely believed that perceived safety and trust are important determinants of AV adoption [105], which has been investigated using explanatory models.

Various models of technology adoption have been proposed, however two prominent in the literature are the Technology Acceptance Model (TAM) [49] and Unified Theory of Acceptance and Use of Technology (UTAUT) [197]. These models are based on the notion that subjective belief and user perception shape technology acceptance, with behavioural intention (BI) and actual usage taken as measures of acceptance [210]. In particular, the TAM approach has inspired several studies (e.g., [41, 110, 210]). Figure 2.9 shows the original TAM model by Davis, Bagozzi, and Warshaw [49], which proposes that perceived ease of use (PEOU), perceived usefulness (PU) and attitude towards using (ATT) are antecedents of behavioural intention. BI is determined by a user's attitude towards acceptance which has two predictors, PEOU and PU.

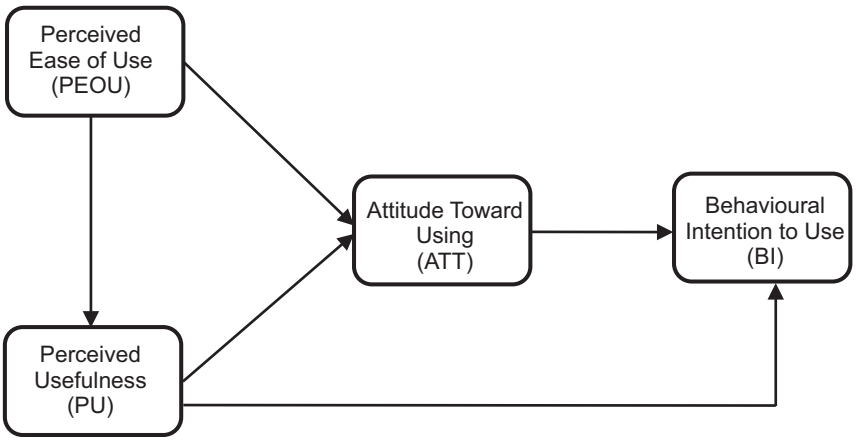

Figure 2.9: Original TAM proposed by Davis, Bagozzi, and Warshaw [49].

Choi and Ji [41] adapted the original TAM, extending it by incorporating two subjective attitudinal factors, perceived risk and trust, and two driving related traits, locus of control and sensation seeking. Based on the human factors literature, trust was decomposed into three determinants: system transparency, technical competence, and situation management. An illustration of the model is shown in Figure 2.10. To validate the model, a survey of 552 respondents was conducted and the results were analysed using partial least squares. Results showed that PU and trust are major determinants of the behavioural intention to use AVs, and that the identified sub-components have a positive influence on trust formation. In addition, trust was shown to have a negative effect on perceived risk. A similar model was proposed by Xu et al. [206] which

also showed that perceived safety and trust were steady and direct predictors of intention to use.

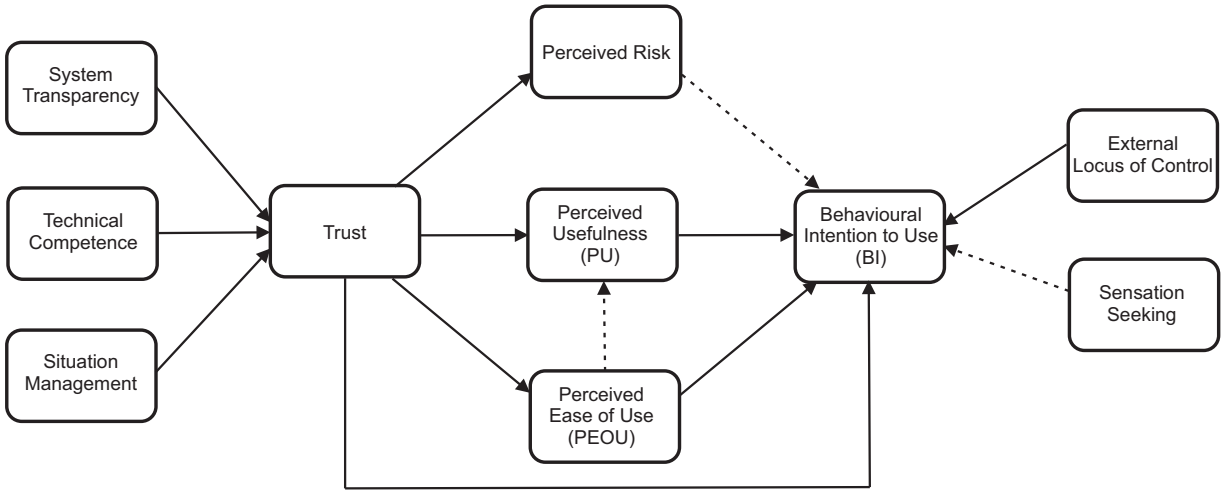

Figure 2.10: Adapted TAM proposed by Choi and Ji [41]. Solid lines represent significant relationships between measures and dashed lines represent insignificant relationships ($p < .05$).

In a recent study, Zhang et al. [210] incorporated initial trust [87] and perceived risk (specifically, perceived safety risk [PSR] and perceived privacy risk [PRR]) into TAM to investigate the effect on public acceptance of autonomous vehicles. Figure 2.11 shows an illustration of the extended TAM approach. Using structure equation modelling, Zhang et al. found strong support for the model, with initial trust identified as the strongest predictor of user attitudes towards autonomous driving. In addition, the strongest determinant of initial trust was perceived safety risk, suggesting a possible intervention strategy for AV adoption. For example, manufacturers could explain the safety features activated to protect users when systems fail, and increase transparency into AV algorithms and the data used in training. Indeed, this is reflected in the AI HLEG ethical guidelines [1] in both the requirements for transparency and accountability. However, while trust-building efforts by manufacturers may be useful for promoting adoption prior to use, strategies for maintaining trust during operation (i.e., the calibration of dynamic learned trust in the Hoff and Bashir [87] model) are needed to ensure safe and continued use.

2.3.2 Explainability

As mentioned in Section 2.1.2, the provision of explanations for vehicle actions is one strategy considered important for maintaining trust during the operation of AVs [112, 149]. In addition, the transparency afforded by explanations is important for ensuring sufficient insight into the reliability of the system for a range of stakeholders involved in AV deployment (e.g., users, regulators, insurers, developers, etc.) who all have different requirements for the level of detail provided by explanations. Explanations are therefore important for accountability, human oversight and the robustness of AV systems. While several taxonomies of XAI methods have been proposed, the field is complex and the focus varies which has resulted in a general lack of

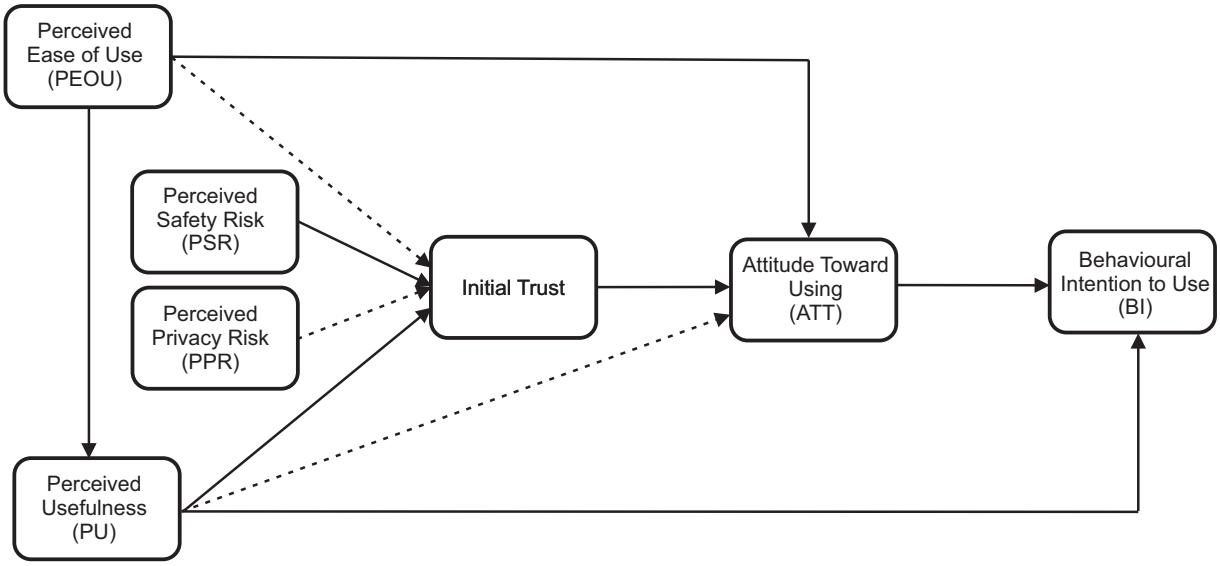

Figure 2.11: Adapted TAM by Zhang et al. [210]. Solid lines represent significant relationships between measures and dashed lines represent insignificant relationships ($p < .05$).

consensus [77, 183]. Nonetheless, common themes in XAI taxonomies can be derived [183].

One common distinction is between AI models which are inherently explainable and the explanation of opaque models after they are trained. The former has been referred to as *ante-hoc explainability* and the latter *post-hoc explainability* [112, 183]. As indicated in Section 2.1, AI approaches which rely on well-defined symbolic representations and logical manipulation are often ante-hoc explainable, as a causal chain of reasoning between represented concepts can be formally established. Some sub-symbolic approaches can also be considered ante-hoc explainable, e.g., linear regression models, decision trees, rule-based learners [21]. However, if these models have many rules or parameters, it may become impossibile to comprehend the decisions they make, hence post-hoc explainability methods may still be necessary [183].

Typically the choice of whether to use sub-symbolic ante-hoc models depends on the problem, as these models often require additional manual effort in feature engineering and often do not achieve an accuracy comparable to opaque (i.e., "black-box") models, such as DNNs [183]. This is widely known as the performance-explainability trade-off, which can lead machine learning practitioners to assume that an ante-hoc explainable model is unable to achieve sufficient performance on a problem [171, 183]. As a consequence, opaque models are frequently used by system developers, which are only amenable to post-hoc explainability methods. Post-hoc methods can be further classified into *model-agnostic* and *model-specific*. Model-agnostic post-hoc methods work for any type of model, whereas model-specific methods are only applicable to specific model types, e.g., DNNs, support vector machines (SVMs) [112, 183]. One final common distinction for post-hoc methods is between *local* and *global* explanations, where the former concerns explaining the output of a model and the latter the model as a whole.

Beyond these distinctions, taxonomies of XAI methods tend to vary depending on their spe-

cific focus. Relevant to the present thesis, in a recent review Kuznietsov et al. [112] built on these common notions to develop a taxonomy of XAI methods focused on regulatory compliance for the development of safe and trustworthy autonomous driving (see Figure 2.12). Based on [183], they categorised XAI methods according to representation, stage, mode, scope and medium.

Representation, stage and scope have already been discussed, however in [112] representation and scope are extended. In addition to symbolic and sub-symbolic *representation*, a mixed representation was included to account for AI approaches which combine both aspects (e.g., XAI methods for neuro-symbolic AI models [211]). *Scope* was also extended to include cohort based explanations, which describes explanations for the output of a model for a group of inputs. The remaining two categories are medium and mode. *Medium* concerns the method of delivery (e.g., textual, visual, etc.) but the *mode* category is more complex and requires some discussion.

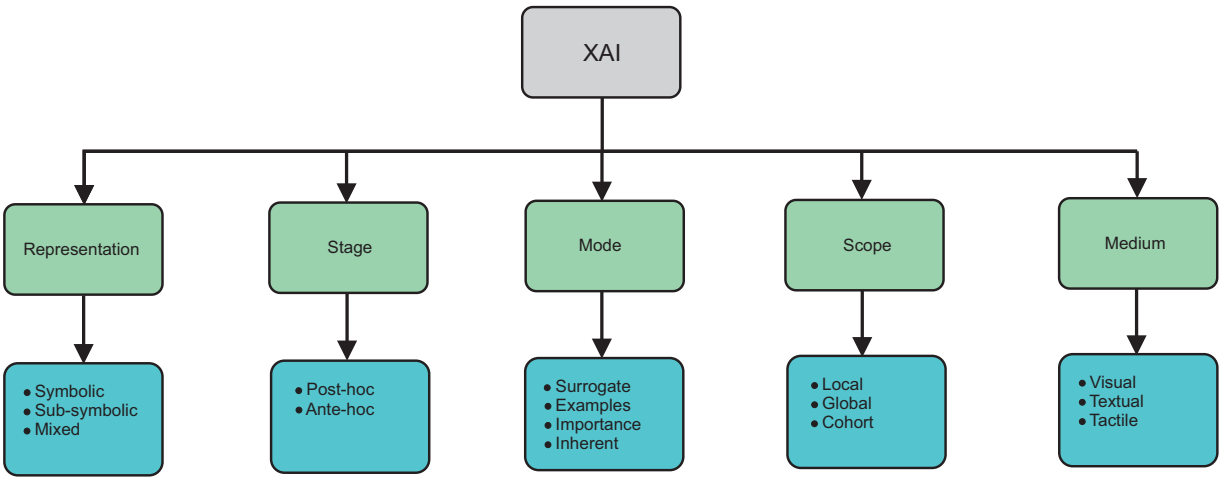

Figure 2.12: Taxonomy of XAI methods for safe and trustworthy autonomous driving [112].

According to Kuznietsov et al. [112], mode describes the syntactic and semantic form of an explanation, which has four possibilities: surrogate models, example-based methods, importance-based methods and inherent explainability. *Surrogate* models distil a more complex model (e.g., DNN, SVM, etc.) into a interpretable model. For example, Mishra et al. [141] used a decision tree to explain the decisions of an RL agent based on states and corresponding actions determined by the optimal policy, using a visual explanation interface which was evaluated on users. Gyevnar et al. [78] proposed CEMA based on a cognitive model of how people select causes for actions, using simulation based on a probabilistic planner to emulate counterfactual reasoning by an AV agent. They generated natural language explanations about causal relationships which were validated in an online study for 200 participants against a baseline of hand-written explanations. However, it is typically unclear how representative explanations via surrogate models are of the original decision due to inconsistencies in feature attribution methods [112].

Example-based methods are another post-hoc approach which use representative and coun-

terfactual examples to provide insight into some aspects of model decisions, identifying minimal alterations to the input of a model which can change its output [112]. For example, Li et al. [121] used Shapely values [205] to rank the most influential features in decisions by a DRL model in an autonomous driving task to generate representative counterfactual explanations, validated with a user study which indicated an improvement in understanding model predictions. However, this approach assumes that the user is able to understand the examples correctly.

Importance-based methods focus on highlighting which input features are most relevant to a decision from a black-box model. For example, Cui et al. [47] combined SHapely Additive exPlanation (SHAP) and random forests to increase the post-hoc explainability of a deep reinforcement learning (DRL) model for a vehicle following task. SHAP was used to determine the important features used by the DRL algorithm in decision-making on which a random forest was then trained to explain the decisions of the original model. Ma et al. [135] used a mean impact value method to rank the importance of features for a neural network model in a lane change task. Closely related features were then used as input to a binary logistic regression model for intention prediction. While importance methods can provide transparency into opaque models, however, care should be taken in interpretation, as provided explanations are correlational [112].

The final category is *inherent* explanations, provided by models which are ante-hoc explainable by construction. In other words, they are furnished by interpretable models which reveal explicit causal relationships between their inputs and outputs by virtue of their design [112]. While some mixed representation methods are inherently explainable (e.g., [8]), models in this category are often symbolic. For example, Brewitt et al. [29] proposed Goal Recognition with Interpretable Trees (GRIT) to infer the goals of other vehicles and predict trajectories for safe interaction. They aimed to satisfy four core objectives for AV methods of being fast, accurate, interpretable and verifiable using decision trees trained on vehicle trajectory data. Hanna et al. [81] introduced the interpretable Goal and Occluded Factor Inference (GOFI) to jointly infer a probabilistic belief over goals and potential occluded factors, demonstrating how it could be integrated into Monte Carlo Tree Search (MCTS) action selection. However, inherent explanations can require significant AI expertise which may not be available to all stakeholders [112]. Nonetheless, for satisfying accountability, transparency and safety requirements (e.g., as detailed in the AI HLEG ethical guidelines [1]), symbolic approaches can provide a rigorous characterisation of causal relationships, which may benefit future safety certification [58, 149].

2.3.3 Testing and Validation

In the automotive industry, the standard ISO 26262 (*Road vehicles—Functional safety*) [185] defines a staged V-model process for validating the functionality and safety of a road vehicle's intended use [18, 162] (see Figure 2.13). The left side of the V-model represents the design phase, whereas the right side represents the testing and validation phase. However, while such an approach may be feasible for hardware/software in vehicles without autonomous functionality,

the incorporation of the DDT into the system (especially in fully autonomous driving) requires validation of decision-making and control across a range of scenarios—possibly infinite due to high variability in driving conditions [18]. There are three main strategies in the literature: field operational tests (FOTs), and simulation-based X-in-the-Loop and scenario-based testing.

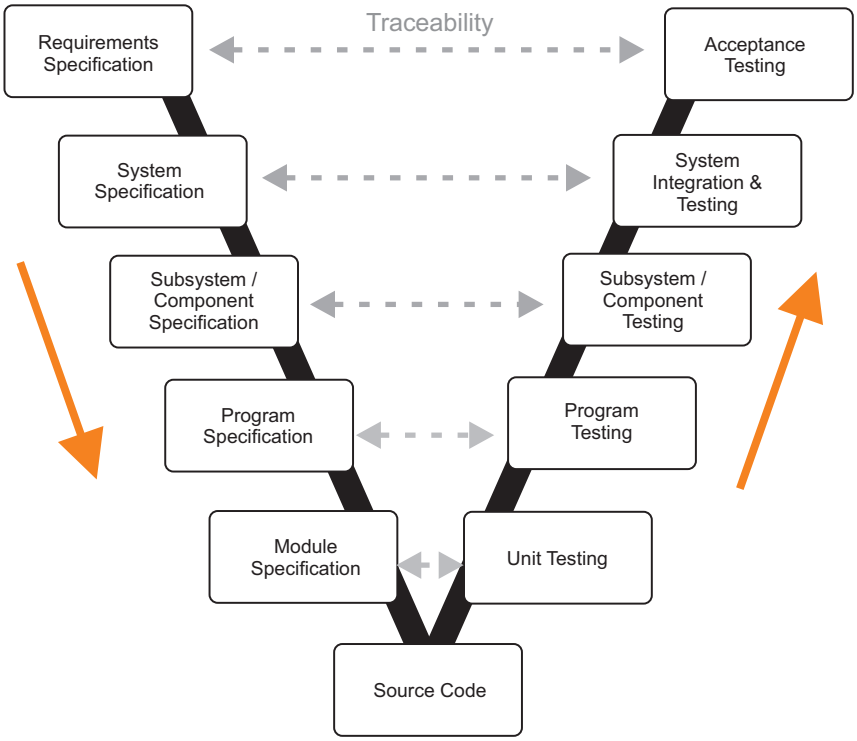

Figure 2.13: V-model in ISO 26262 [162, 185]

FOTs are conducted in realistic driving conditions on public roads. One clear advantage is the ability to test the operation of AV systems (especially decision-making and planning, which is ultimately responsible for turning system inputs into actionable outputs) in an ecologically valid driving environment. As mentioned in Section 2.1.3, however, demonstrating the reliability of AVs through empirical road testing alone is insufficient [98] and moreover replicating dangerous situations on real roads is infeasible for obvious safety reasons [42]. Furthermore, there is little control over the scenarios encountered, so the replicability and completeness of real world testing is problematic. This has prompted researchers to develop alternative approaches for acquiring evidence of the safety and reliability of AV systems using simulation.

X-in-the-Loop (XiL) methods incorporate simulated aspects at varying levels of abstraction from the actual system (see Figure 2.14 for a summary of XiL methods). The most abstract method is Model-in-the-Loop (MiL) which can be used at design time to validate system functional requirements prior to the availability of concrete system artefacts. Both Software-in-the-Loop (SiL) and MiL are time agnostic, meaning that simulations can be run repeatedly without being limited by real-time constraints, however the results in general lack ecological validity [98]. This typically increases as the XiL method becomes more concrete. The most concrete

XiL method is Vehicle-in-the-Loop (ViL), which involves testing a complete AV system with simulated environmental inputs. However, while this has the benefit of achieving greater ecological validity, hardware aspects make it necessary to test the system in real-time, introducing issues with scalability [18]. One approach to address these limitations is scenario-based testing.

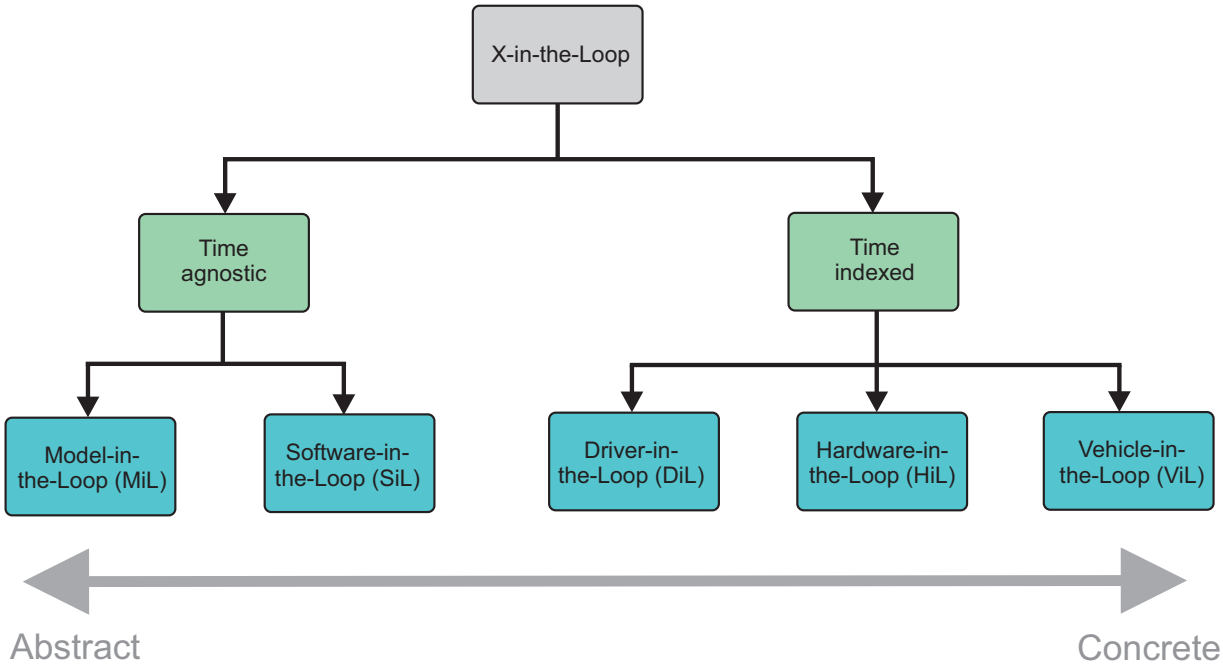

Figure 2.14: X-in-the-Loop methods, adapted from [18].

Due to the complexity of the driving environment and the high variability of possible scenarios, exhaustive validation of AV safety and reliability is impractical. Scenario-based testing aims to address this through techniques which ensure comprehensive coverage of safety-related cases which are relatively inexpensive and feasible within a finite period of time. One of the main challenges is how to optimize scenario generation and improve efficiency [187]. Some approaches focus on achieving sufficient coverage of the ODD for an ADS task to systematically demonstrate safety and reliability with minimal resources (e.g., [12, 69, 96]), others target unsafe scenarios for focused validation of specific issues (e.g., [68, 187]), and some methods generate scenarios using naturalistic distributions (e.g., [51]). The ultimate aim is to provide sufficient evidence for the certification of an ADS. However, one issue is quantifying the amount of samples required to guarantee sufficient coverage for a scenario [213]. This is particularly important for black-box AI components, such as DNNs, due to the inductive nature of learning systems.

2.4 Mobile Robotics and Obstacle Avoidance

The barriers to adoption discussed in Section 2.3 are important challenges for the development of fully autonomous vehicles which are trustworthy, however efficient methods for local planning and obstacle avoidance are also necessary, as mentioned in Section 2.2.3. Obstacle avoidance has

a long history in mobile robotics, an area of robotics concerned with autonomous ground, ariel and underwater vehicles [146]. While this can be planned ahead in static environments where obstacles are indicated on a prior map, this is not the case for dynamic or unknown environments. It is therefore desirable for autonomous mobile robots to detect and avoid obstacles in real-time.

Several methods have been suggested in the literature. One family of approaches to reactive obstacle avoidance is bug algorithms (BAs) [133], an efficient obstacle avoidance method suitable for autonomous navigation in unknown environments. This is particularly useful for small robots with limited resources in indoor environments where accurate localization via GPS is unavailable and simultaneous localization and mapping (SLAM) techniques are too computationally intensive. While there are numerous variants, the canonical BA (e.g., Bug1 and Bug2 [133]) navigates the contours of an object at a safe distance after it comes into contact until it is possible to resume its original path. More recent BA variants have incorporated ranged sensor data. Kamon et al. [100] introduced TangentBug based on a detectable obstacle field around the robot, allowing it to take action before coming into direct contact with an obstacle, resulting in more efficient trajectories. However, it has been argued that BAs tend to rely on perfect position estimation [139] which is problematic for environments where GPS is unavailable or unreliable. Consequently, mobile robots using a BA must estimate position using odometry, acquired via noisy on-board sensors, which can lead to rapid degradation of obstacle avoidance performance.

Artificial potential field (APF) [106] methods characterise a mobile robot as a particle suspended in a potential field with attractive forces generated by the target and repulsive forces generated by obstacles in the environment. The total force applied to the robot is the sum of repulsive and attractive forces, compelling the robot to move in a relevant direction [106, 146]. One of the advantages of this approach is the simplicity of the basic idea making it straightforward to implement. It is also suitable for online decision-making based on immediate sensor data. However, there are known issues with the standard formulation. Perhaps the most well-known is the tendency for APF methods to get stuck in local minima (e.g., when negotiating U-shaped obstacles) [109]. Other issues include difficulty in passing between objects that are close together and unstable motion in narrow passages due to the influence of repulsive forces in opposite directions. To reduce uncertainty due to noise from sonar sensors, an extension to APF called a virtual force field (VFF) was introduced in [24] which used certainty grids for robust sensing and heuristics to address issues with local minima. However, certain obstacle configurations remain problematic, such as narrow hallways with repulsive forces in opposite directions, due to the sudden reduction of the certainty grid to a force vector in a single step [146].

The vector field histogram (VFH) [25] method addresses this by breaking the problem into two steps. First a two-dimensional certainty grid is reduced to a one-dimensional polar histogram indicating the density of obstacles for a set of sectors around the robot's momentary location. A histogram valley below a critical threshold closest to the direction of the goal is then chosen as a candidate safe direction, the centre of which determines relevant control inputs for

steering the robot. The oscillations observed in narrow hallways for APF and VFF methods are not a problem, and the VFH method is robust to sensor noise due to the use of certainty grids, making it suitable for use with sensor fusion techniques [25]. However, the robot can still get trapped in local minima for certain obstacle configurations (e.g., U-shaped obstacles). As with the VFF method, this can be addressed using heuristics to detect when the robot is trapped in cyclic behaviour and generate a strategy to counteract the situation. Other issues with VFH are trajectory oscillations for certain configurations of obstacles and possible collisions due to it being agnostic to vehicle geometry, kinematics and dynamics. These issues were later addressed in an extension to the method VFH+ [195] though it could still get stuck in dead-ends.

One obstacle avoidance method which incorporates both vehicle kinematics and dynamics is the dynamic window approach (DWA) [66]. This method searches in the velocity space consisting of possible pairs of linear and angular velocities. DWA focuses only on the next steering command and approximates trajectories using a circular curvature to reduce the search space to two dimensions [66]. This is reduced to admissible velocities that: (i) are within the maximum velocities of the vehicle, (ii) generate safe trajectories, and (iii) are reachable in the next time step given vehicle accelerations [178]. An objective function which compromises between velocities which make progress to the goal, maximise distance from obstacle and high speed is then used to select velocities from the remaining candidates (i.e., the dynamic window). As the approach incorporates vehicle dynamics and optimizes for high speed, it is particularly useful for applications which require fast autonomous navigation. However, the approach was intended for static environments and consequently struggles with dynamic obstacles.

The ability to handle dynamic obstacles is generally a strength of heuristic methods for reactive obstacle avoidance, such as fuzzy logic [61, 166, 167], deep reinforcement learning [39, 59, 140] and convolutional neural networks [33, 145], though learning based methods are computationally demanding and fuzzy logic is subject to classical problems: oscillations and local minima. In recent years there has been increased interest in hybrid planning and control methods. This combines the power of discrete methods for long horizon task planning and low level control for execution of the continuous motion. In [114], for example, a motion-planning framework for a hybrid system was proposed with partial safety guarantees, and in [134] a sampling-based approach using temporal logic and RRT* was introduced. The advantage of the hybrid approach is that low level state/action pairs do not have to be repeatedly planned from scratch as the high level planner provides concrete input for the low level motion [75]. However, these hybrid methods are often computationally demanding due to the increased internal complexity. Often the discrete aspects utilise approaches that have been developed in formal methods.

2.5 Formal Methods

The discipline of "formal methods" describes a set of mathematical techniques for analysing the properties of systems, typically applied to the analysis of software and hardware components. They provide frameworks to formally specify and verify systems in a systematic manner and achieve assurances or guarantees that a system specification satisfies desired properties [5, 204]. One common application is the formal verification of software systems, especially safety-critical systems which could lead to fatal consequences [107]. Given specification of the desired behaviour of a system as a formal property, a formal proof of its correctness can be derived, or alternatively proof that the specification is not met [5]. This can be done by constructing a model of the system or through other methods, e.g., theorem proving [10, 147]. Our focus in this thesis is one type of model based formal verification, an automated technique called *model checking*.

2.5.1 Model Checking

Model checking [13] is a widely used computer-aided technique for automatically verifying reactive systems, based on a precise mathematical and unambiguous model of possible system behaviour. To verify that the model meets specified requirements (usually expressed in temporal logic), all possible executions are checked systematically. If an execution failing the specification is found, the execution path which caused the violation is returned, which can inform refinement of the model or be taken as evidence that the system violates the specification. It has been applied to ensure the safety and reliability of safety-critical systems [84, 130, 201] and applied to many aspects of software verification, e.g., to industrial systems and operating systems [17, 193]. The typical workflow of verification using model checking is shown in Figure 2.15.

System models for verification using model checking comprise all variables relevant for establishing whether a system property holds. For the most part, they are expressed using *finite-state automata*, which consists of a finite set of states and a set of transitions [13]. States typically comprise information about the values of system variables, whereas transitions describe how the system evolves from one state to another over time. A system model therefore provides semantics for a domain (e.g., a reactive system) which grounds the truth of formal properties.

One common challenge is that the state space of a model can grow exponentially with the number of variables or events, known as *state-space explosion* [13]. As model checking involves exhaustively checking all possible executions of the modelled system, the exponential growth in the size of the state space can make verification of the model computationally infeasible. While there are various techniques to address this, such as symmetry breaking [43] and partial order reduction [44], one common approach is to construct an approximate model that abstracts away unnecessary detail yet is sufficient to prove/disprove the target property [175]. Indeed, this is often necessary when verifying continuous aspects of a reactive system (e.g., a robot interacting

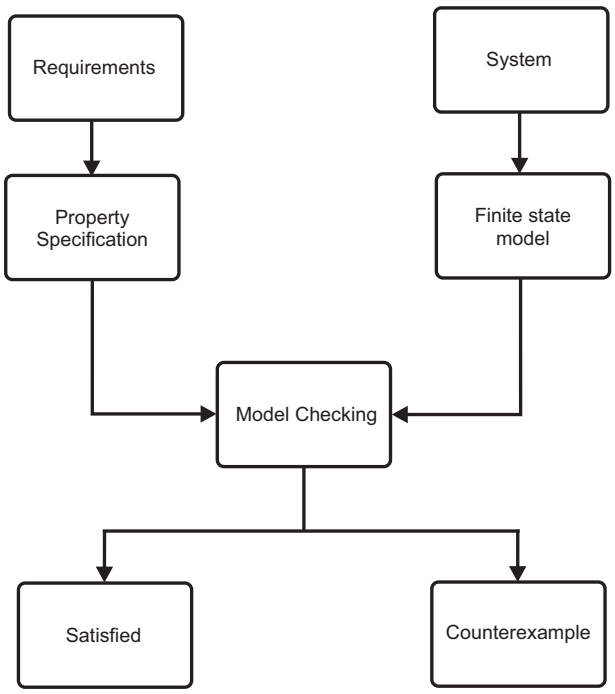

Figure 2.15: Schematic of typical model checking workflow.

with the physical environment) as model checking is not effectively computable for infinite-state systems. As verification involves exhaustive search of the state space, a finite and discrete representation is required which is fine-grained enough to ensure a decision about satisfaction of a property but coarse enough to ensure that the reachable state space can be searched within an acceptable period of time. It is therefore desirable to keep the model both small and relevant.

As mentioned above, property specification is normally provided in some well-defined formal logic, usually temporal logic as we are often interested in how the system evolves over time. One major advantage of model checking is that it relies on symbolic representation of domain properties, hence the approach is fundamentally ante-hoc explainable. It is therefore possible to establish a causal chain of reasoning sufficient for a formal proof of system correctness. In particular, as all possible execution paths in the model are considered, it is possible to reason about complete *sequences* of state-action pairs. From the perspective of trustworthy AI, this is clearly beneficial in terms of accountability, traceability, explainability, and reliability [1, 58].

While model checking is typically applied as an offline technique for the analysis of reactive systems, it has also been proposed as a means of solving AI task-planning problems [11, 71], due to similarities between planning and model checking problems. Indeed, experimental results have shown that the performance of SPIN [90] and SMV [22] model checkers are comparable to state-of-the-art planners [23]. Hoffman et al. [88] have argued that model checking languages based on automata networks (e.g., Promela and SPIN [90]) lend themselves naturally to an agent-based view of the world, a useful abstraction for decision-making in autonomous systems.

Autonomous robotic systems, such as AVs, are hybrid and cyber-physical. In other words,

they combine discrete decision-making with continuous control and interact with a continuous physical environment, which is often partially observed and unstructured [64, 131]. This presents unique challenges for the use of model checking for verifying robotic systems.

One approach is to use hybrid automata [85], which combines discrete decision-making and modelling of continuous variables. However, while a relevant abstraction, reachability is only decidable for restricted classes of hybrid automata [7] and the modelling and verification of discrete decision-making tends to scale poorly as the complexity of the model increases [129]. Hence some research has focused on a modular approach to the verification of autonomous systems. For example, Kamali et al. [99] represented vehicle platooning as a multi-agent system. Discrete decision-making was verified using GWENDOLEN [52] and the Agent Java PathFinder (AJPF) model checker [53], while real-time requirements were verified in UPAAL [115].

There are several examples of model checkers being used to generate plans for autonomous agents. Gu et al. [73] synthesised and verified plans for autonomous agents in complex road conditions using a GUI tool (MALTA) integrated with the UPPAAL model checker. Guo et al. [76] investigated the use of online model checking for real-time revision of motion plans for a robotic agent using specifications in linear temporal logic for partially observed environments. In the context of autonomous driving, the SPIN model checker was combined with simulation in Unity for online planning of collision-free paths for successive overtaking manoeuvrers [154].

One advantage of the approach in [154] was the ability to make discrete decisions about actions based on immediate sensor data. From a safety and reliability perspective, this is ideal as up-to-date information about the environment is necessary in high-speed driving contexts to avoid imminent collisions. In addition, the decisions are explainable. In comparison, popular end-to-end planning methods using deep reinforcement learning are opaque and pre-trained in simulated environments. While simulation produces significant amounts of data for training, ultimately model outputs are not informed by the immediate context, which could lead to risky and unexpected behaviours. However, while [154] was a promising step towards reliable planning for AVs, the compiltion time due to SPIN made the approach infeasible for real-time applications (approx. 3 seconds, even though the algorithm latency was only 20 milliseconds).

Standard model checkers are not designed for real-time tasks and are generally too slow for online verification in a dynamic continuous environment. For this reason, models are typically verified offline prior to implementation [203, 215] for real-time applications. In addition, most research using model checking for autonomous robots has focused on validating results using simulation (e.g., [63, 67, 80, 85, 124, 128, 129, 157]), which cannot replicate real world conditions precisely. Real world robotic systems are prone to error (e.g., sensor failures, errors in actuation) and tolerance to this error is an important aspect of safety and reliability. While simulation environments can be useful for testing a navigation method with respect to a number of scenarios quickly, the idealised nature of simulations means that there is no guarantee that the assumptions of the method are robust to the uncertainty present in real world robotic systems.

To address issues with uncertainty we work directly with error in the system to develop robust obstacle avoidance using model checking, hence we adapt the approach in [154] to a real autonomous robot operating under hard real-time constraints. Consequently, our approach is derived from techniques in the SPIN model checker. Note that while the presence of real-time constraints may suggest using timed automata [6], our focus is on generating safe sequences of motion primitives where non-determinism in the choice of possible future actions is resolved by concurrent obstacles in the immediate environment. We are therefore not interested in modelling real-time aspects of the system directly in this thesis, only on ensuring that the processing latency of the implementation is less than 100 milliseconds. In the interest of planning and obstacle avoidance which is in addition low cost and energy efficient, we develop our approach on a low-powered robotic platform. As model checking is symbolic, our is approach is ante-hoc explainable. We validate our approach using physical experiments instead of simulations to understand the performance of collision avoidance in the real world despite error in the system.

Chapter 3

Preliminaries

In this chapter, we introduce definitions and concepts underpinning the use of model checking for real-time planning and obstacle avoidance. In Section 3.1, we describe our low-powered robotic research platform, conceptual model of the sensed data and motivate our restriction to an egocentric point of view. Section 3.2 motivates and describes our closed-loop definition of an embodied agent able to formulate complex, multi-step plans. In Section 4.3.1, we introduce the notion of a temporary closed-loop controller as a motion primitive with discrete outcomes which we call tasks. We describe how these can be chained together to form plans. We briefly discuss our action-based approach to modelling the physical environment in Section 3.4. In Section 3.5, we provide necessary conceptual background on the model checking as planning paradigm.

3.1 Robotic Platform

As a measure to promote the development of energy-efficient real-time planning and obstacle avoidance, our approach was developed on the low-powered robot shown in Figure 3.1, adapted from a widely available mobile robot development platform, AlphaBot by Waveshare[1]. For sensing the environment, we equipped the robot with a low cost 360 degree 2D laser scanner, RPLiDAR A1M8 by Slamtec[2], and for actuation we used two continuous rotation servos by Parallax[3]. The hardware programming interface for the robot was a Raspberry Pi 3 Model B[4] running a Quad Core 1.2GHz Broadcom 64bit CPU with 1GB RAM and wireless LAN.

3.1.1 Sensor Data

The LiDAR generates a 360 degrees 2D point cloud at a rate of ≈ 5 Hz, with each scan generating 8192 data points. We use a 2D vector space to model the point cloud with the origin

[1] https://www.waveshare.com/alphabot-robot.htm
[2] https://www.slamtec.com/en/LiDAR/A1/
[3] https://www.parallax.com/product/parallax-continuous-rotation-servo/
[4] https://www.raspberrypi.com/products/raspberry-pi-3-model-b/

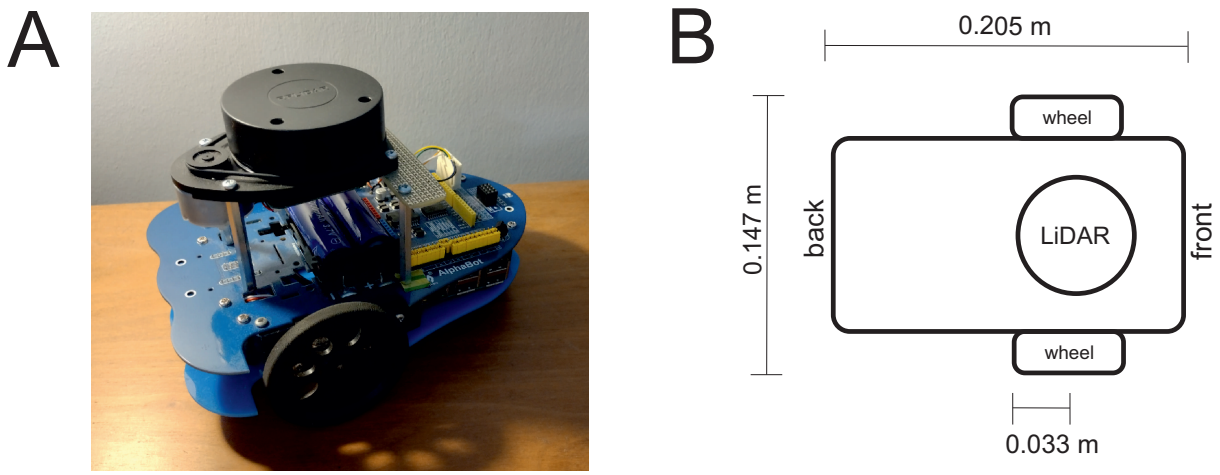

Figure 3.1: A: our robotic platform. B: schematic showing robot dimensions.

representing a point at the centre of the LiDAR. The robot is oriented towards positive x by convention which corresponds to the heading 0 radians, as shown in Figure 3.2. Rotating 90 degrees left or right is equivalent to headings $\frac{1}{2}\pi$ and $-\frac{1}{2}\pi$ radians (as illustrated in Figures 3.2A and 3.2B, respectively). The maximum heading is π in the left direction and $-\pi$ in the right direction, each of which represents a 180 degrees rotation relative to a local frame of reference.

Note that in this initial investigation we restrict to rotating $\pm\frac{1}{2}\pi$ radians as this simplifies our approach for obstacle avoidance. If we used rotations of $\pm\frac{1}{4}\pi$ radians, for example, it is more likely that a single rotation will result in facing the same obstacle, even if it is convex. However, if we rotate perpendicular to the obstacle, then only concave obstacles remain problematic (e.g., U-shaped obstacles). Rotating $\pm\frac{1}{2}\pi$ radians also makes it possible to use symmetry on the axes of the vector space for perception which simplifies the implementation of our approach.

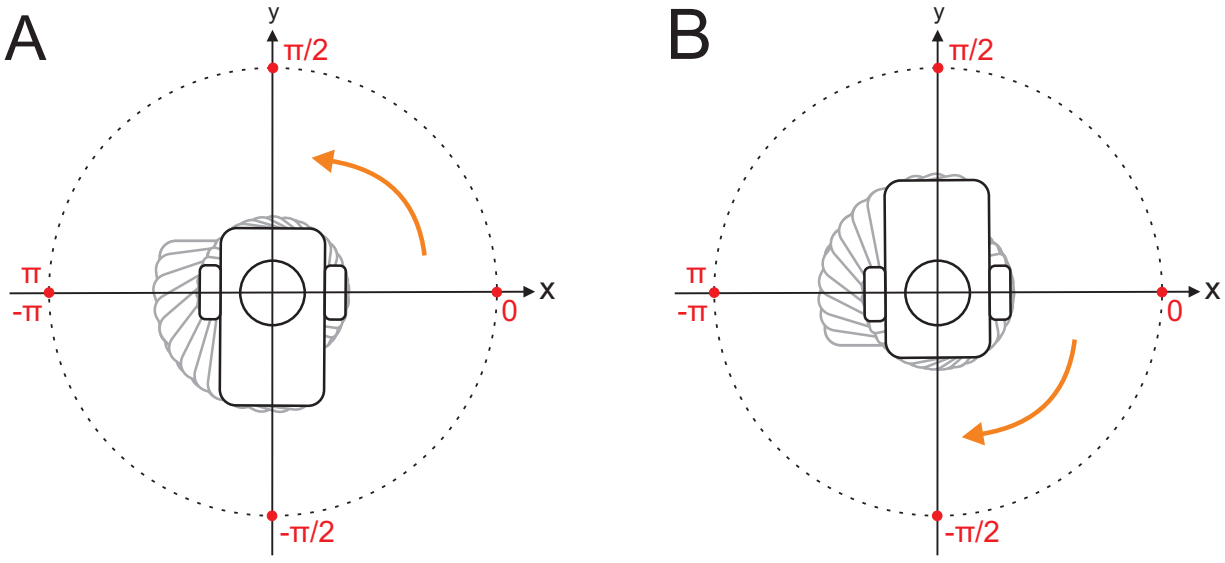

Figure 3.2: A: rotating left to heading $\frac{1}{2}\pi$ radians. B: rotating right to heading $-\frac{1}{2}\pi$ radians.

When viewed from the local frame of the robot, actions change the environment state but the robot itself remains static. Figure 3.3A shows an example of state changes from the local perspective as the robot rotates left ≈ 30 degrees. The local frame is indexed at time t and so describes the current state of the environment sensed by the LiDAR, hence $t-1$, $t-2$ and $t-3$ represent historical environment states as a consequence of previous actions. However, it is also possible to simulate actions in an environment and predict states to reason about possible future outcomes. Figure 3.3B shows simulation of the robot rotating left ≈ 30 degrees. Again the robot is indexed at the current time t, however in this case indices $t+1$, $t+2$ and $t+3$ represent future environment states as a consequence of simulated actions. Actions are therefore *temporally extended*, hence reasoning about robot actions requires either information about the past or simulations of the future. The current state of the environment alone is insufficient.

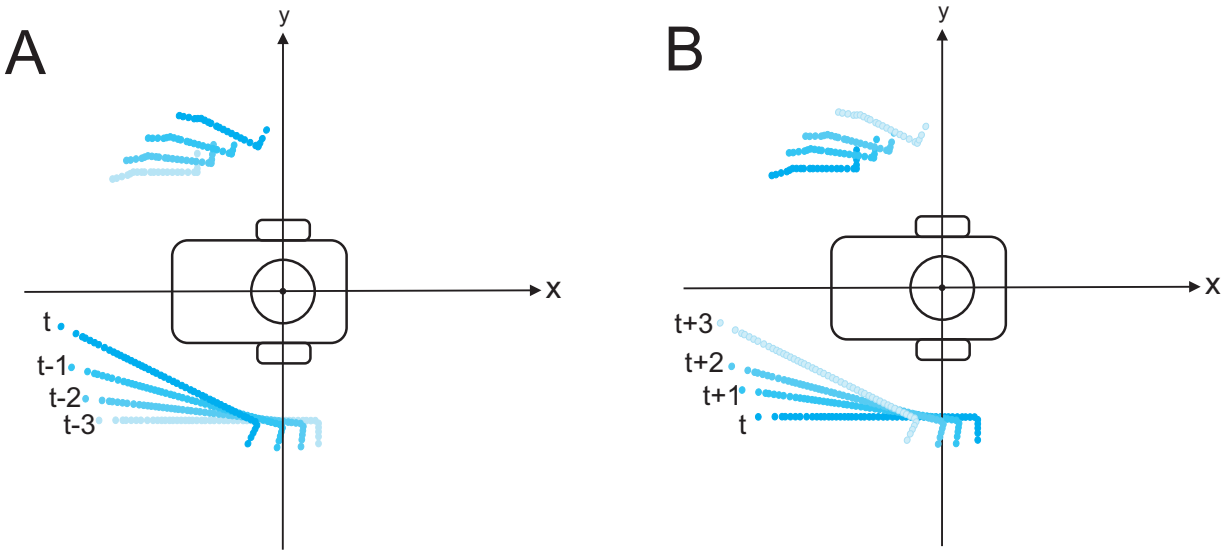

Figure 3.3: A: historical state changes from the robot perspective for the action of rotating left ≈ 30 degrees. B: simulated actions and predicted state changes.

3.1.2 Frame of Reference

As should be clear, our robot is an embodied agent which necessarily has a self-referential perspective on its surroundings—observations of the environment are indexical to a local frame of reference and entirely dependent on built-in sensors. This means that the environment is *partially observed* and all knowledge of objects in the environment is relative to an egocentric point of view. As explained in Chapter 2, an offline HD map is often used to provide a global perspective for AV navigation systems and improve observability, however the storage and maintenance of an offline HD map is resource-intensive. In addition, it is non-trivial to precisely tack the position of an embodied agent on a global map based on data from a local frame of reference [119, 182] and difficult to achieve in real-time, especially with limited compute. We therefore restrict our approach to an egocentric perspective and keep the environment partially observable.

3.2 Embodied Agent Definition

In addition to practical considerations, our restriction to an egocentric point of view is biologically realistic, as even simple organisms can respond in real-time to previously unseen situations. The notion of a Braitenberg vehicle [28] encapsulates the simplest form of response an embodied agent can have to its environment: a reflex. One possible configuration is shown in Figure 3.4. Here proximal sensors are connected directly to actuators, invoking an additive forward motion reflex in response to an external stimulus. Upon sensing an input (see Figure 3.4A), the agent performs an action to change its state, which is in turn sensed by the agent, and the loop repeats until the sensor is inactive (see Figure 3.4B). This can be conceived as sensing a disturbance to the "resting" state of the vehicle, which is returned to when the disturbance is eliminated.

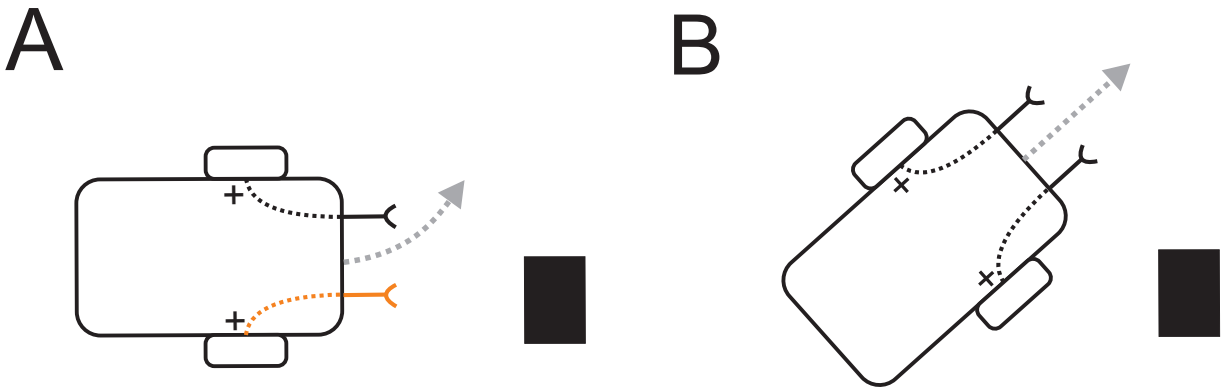

Figure 3.4: Simple Braitenberg vehicle executing an avoidance response to an external stimulus (i.e., an obstacle). A: right proximal sensor senses obstacle which increases forward actuation of the right wheel. B: actuation of right wheel returns to original speed (i.e., its "resting" state).

Complex biological systems are naturally capable of more sophisticated manoeuvrers, such as the prediction of many consecutive obstacles (i.e., disturbances) and the generation of plans to counteract them. As a reflex can only respond to the current state of the environment, simple closed-loop behaviour, as exemplified by a Braitenberg vehicle, is insufficient for the generation of complex multi-step plans. To achieve this, it is necessary for an embodied agent to access and interpret distal sensor information; this makes it possible to reason about the future outcomes of actions. Indeed, there is cross-disciplinary evidence that biological agents have "core" knowledge of physics and causality, which allows organisms to reason about their environment and organise their behaviours in accordance with predicted outcomes [117, 184]. In other words, embodied agents have a *mental model* of the world, allowing them to simulate and evaluate future actions. Our working definition of an embodied agent reflects this capacity for modal reasoning.

Figure 3.5 illustrates the general closed-loop architecture of our embodied agent. As indicated, we use LiDAR to provide distal sensor information about the environment. The controller

reacts to the sensor information and if necessary sends the sensor information to an artificial mental model of the world, which simulates actions and uses modal reasoning to generate a complex multi-step plan, subsequently returned to the controller for execution. The output from the controller is a signal to the actuators appropriate for the current action. The agent feedback loop repeats at a frequency of ≈ 5 Hz, i.e., timing is dependent on the speed of the LiDAR.

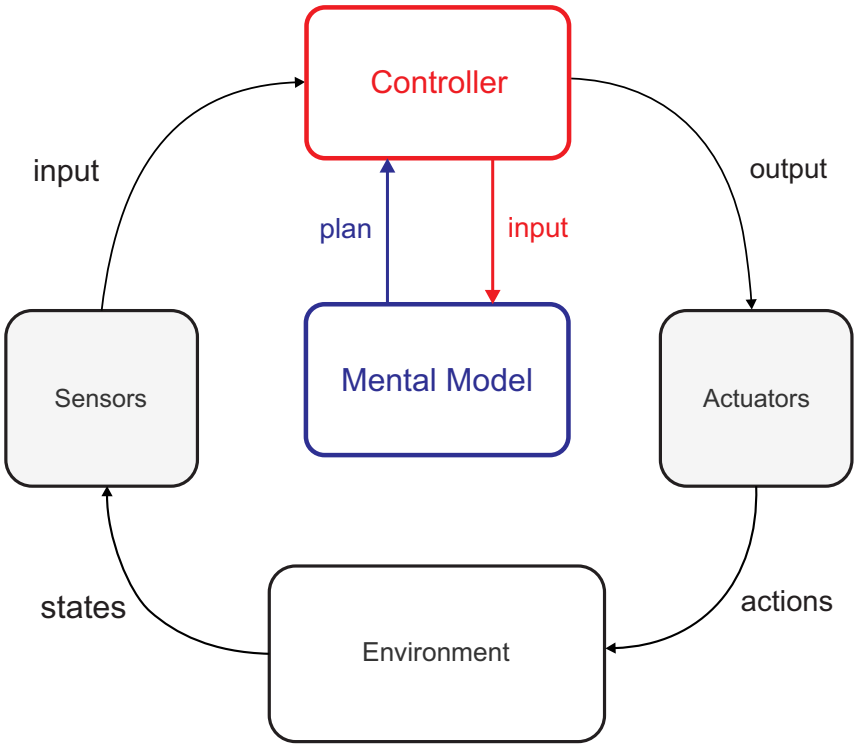

Figure 3.5: General closed-loop architecture of embodied agent.

From a safety and reliability perspective, one of the benefits of a closed-loop agent architecture is reactivity. Similar to [154], our general approach aims to make transparent hard decisions about actions based on immediate information about the environment, which is beneficial for avoiding imminent collisions in a high speed driving environment. To achieve this, it is necessary to make online decisions in real-time based on input control. While popular RL approaches to AV navigation are also closed-loop, they are based on output control trial and error, i.e., they attempt an action and the outcome updates the behavioural policy of the model, which leads to better performance *after the fact*. For obvious safety reasons, RL approaches are therefore not suitable for online decision-making based on information about the immediate environment. Hence RL models are typically pre-trained in simulated environments, which introduces further safety and reliability issues associated with the "reality gap", as discussed in Section 2.2.3.

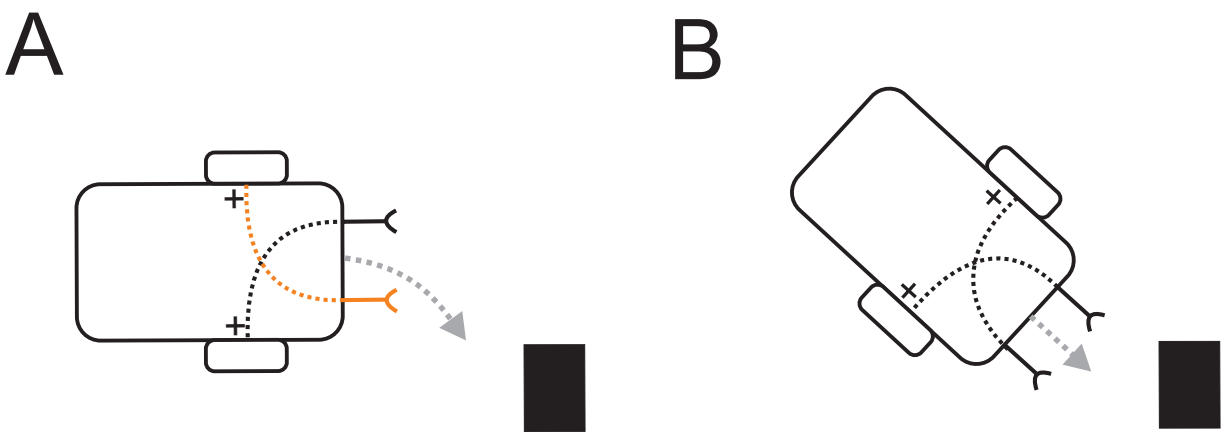

Figure 3.6: Simple Braitenberg vehicle executing an attraction response to an external stimulus (i.e., an obstacle). A: right proximal sensor senses obstacle which increases forward actuation of the left wheel. B: actuation of left wheel returns to original speed (i.e., its "resting" state).

3.3 Closed-Loop Tasks

While reactivity is an advantage of closed-loop input control, a single closed feedback loop cannot form complex plans, as discussed in the previous section. The Braitenberg vehicle example in Figure 3.4 represents the simplest notion of a closed-loop controller, where proximal sensors are hard-wired directly to actuators, generating one behaviour in response to an external stimulus, in this case *avoidance*. While the agent can avoid collisions, it cannot simultaneously make progress towards a goal, as it always executes the same strategy. An alternative configuration of the agent is shown in Figure 3.6, which in this case generates an *attraction* behaviour in response to an external stimulus. Again it always executes the same strategy. In practice, successful obstacle avoidance requires both avoiding obstacles and making progress towards a goal, so a single closed-loop controller, which can only execute a single behaviour, is insufficient.

From the self-referential perspective of an embodied agent, however, an object in the environment is only temporarily in view, hence a single permanent control-loop for obstacle avoidance is not required. Figure 3.7A, for example, shows a simple Braitenberg-style vehicle which by default drives in a straight line and can avoid objects, in this case perceived using LiDAR data. As each data point has distance information, it is possible to collect data points into subsets of observations, based on predefined geometric dimensions, which ultimately serve to restrict the attentional focus of the robot. Each subset of data points (i.e., subsets L and R in Figure 3.7) indicates the relative direction of objects from the robot. It is straightforward to determine if a subset is empty or not and map this outcome onto a closed-loop action, rotate left or right.

In Figure 3.7A, an object comes into view in subset R, so it is non-empty, meaning that there is a discrete disturbance D which needs to be eliminated. A negative feedback loop is then spawned (see Figure 3.7B) which eliminates the disturbance by sending a motor signal to the actuators (i.e., rotate left) until subset R is empty, as shown in Figure 3.7A. The obstacle (i.e., disturbance) is now no longer in view so the control loop has no reason to exist. It can therefore

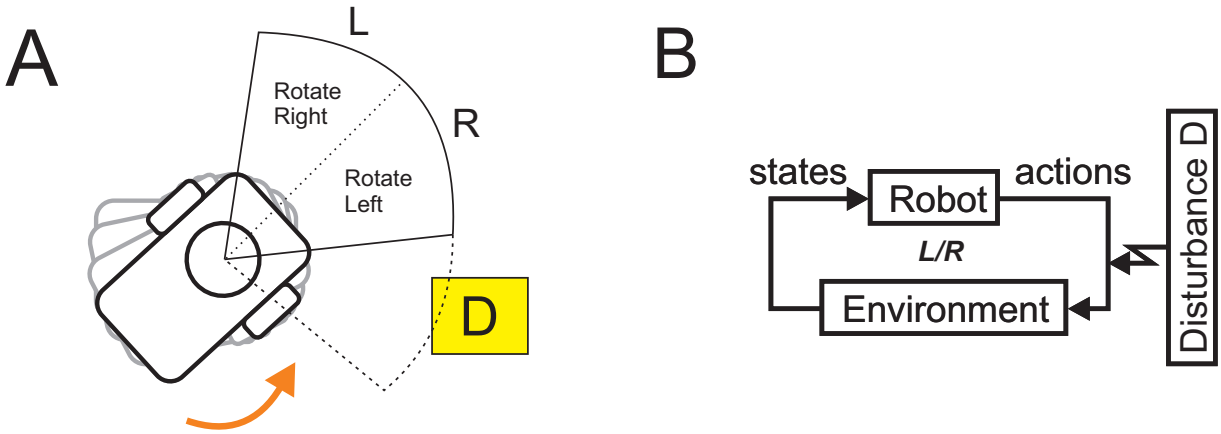

Figure 3.7: A: a Braitenberg-style vehicle perceives a disturbance D in partition R of the environment and spawns a temporary control system to rotate left. B: temporary closed-loop controller with an attentional focus on perceived disturbance which is destroyed when D is out of view.

be destroyed and the robot can return to the default behaviour of driving straight. Hence objects are transient from an egocentric perspective, so a permanent control-loop is unnecessary. In this thesis, we call such a temporary control system a *task*. As a task has a finite lifetime, it has discrete outcomes. This makes it possible to reason about task execution using discrete methods.

However, the agent in Figure 3.7A can only reason about task execution based on current information about the environment (i.e., reflexively). To enable modal reasoning, in this case about the future, it is necessary to reason about *predicted* task outcomes. For complex biological organisms, such as human beings, this might involve imagining the execution of an action. As a proxy for imagination, our approach is to use a combination of static geometric relationships as a proxy for temporal aspects of task execution in the hybrid system and forward simulation to predict the discrete outcomes of tasks in the perceived environment. This makes it possible to ground the truth of statements about the future and reason about outcomes using discrete logic.

3.3.1 Chaining Tasks

The notion of a task as a temporary control system also makes it possible to construct sequences of closed-loop actions for complex manoeuvres based on input control. The tree in Figure 3.8 illustrates the space of possible sequences for the Braitenberg-style example in Figure 3.7 for two steps ahead. The task T_0 is the default task of driving in a straight line, whereas the tasks T_L and T_R are closed-loop avoidance behaviours for eliminating disturbances (i.e., encountered objects in the environment). Using simulation of task execution to predict task outcomes, our aim is to reason about sequences of tasks which avoid obstacles and make progress towards a goal, where a sequence of tasks selected for execution is a *plan* satisfying temporal constraints.

As can be seen in Figure 3.8, however, there are possible sequences which lead to the robot doubling back on itself. This is ultimately because the abstraction used for perception of the environment by the agent in Figure 3.7A (and mapping to closed-loop tasks) is naive. In a

corner, for example, the robot could get trapped. In addition, the robot can only reason about avoiding obstacles, so progress towards a goal is impossible. The semantics of task outcomes therefore depends on the environment model, which is tightly coupled to the definition of tasks. This in turn influences the sequences of closed-loop tasks available for formulating plans.

3.4 Action-Based Mental Models

As a hybrid system, an embodied robotic agent combines discrete decision-making with continuous control and feedback from a continuous physical environment [132]. By focusing on closed-loop tasks, which execute based on continuous feedback from the environment, we effectively offload responsibility for continuous control to the task level. This allows us to focus on the discrete decision-making aspect to form complex plans. Distal sensor information is provided by LiDAR, however some form of perception is required to interpret the raw sensor data and reason about task outcomes. Perception, however, must be appropriate for the problem.

Consider the example agent in Figure 3.7A. The perception model provides information on the relative location of objects which determines the creation of a closed-loop task. In this case, the selected action is fully determined, however if the robot approaches a disturbance approximately straight on this could result in undesired behaviour. Figure 3.9A shows the robot detecting a disturbance D almost directly ahead in partition L. From the perspective of an external observer, it is clear that rotating right will result in facing another object. However, from the egocentric perspective of the robot, there is no knowledge of this second object. It is therefore not possible for the robot to determine that efficiently avoiding the corner requires rotating left.

Sophisticated manoeuvres, such as avoiding a corner, require the ability to perceive multiple disturbances simultaneously (i.e., for situational analysis) and reason about consecutive actions,

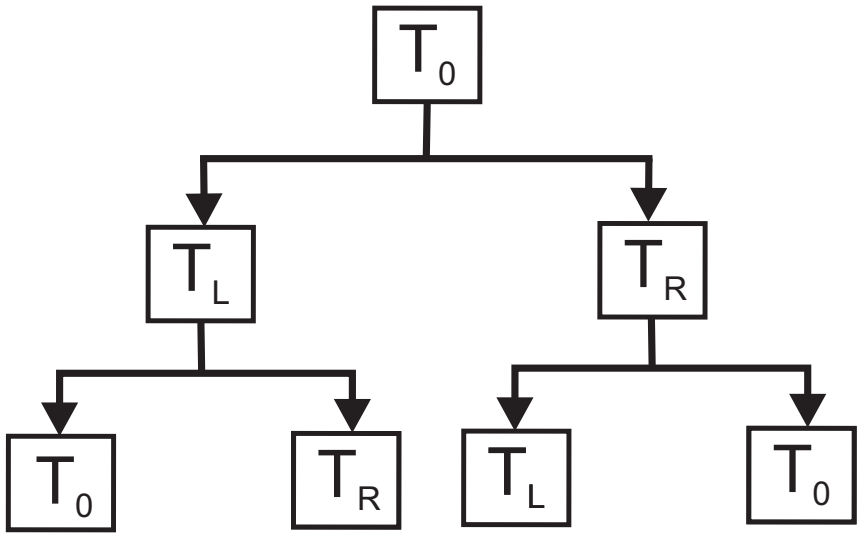

Figure 3.8: Possible sequences of tasks two steps ahead for the agent in Figure 3.7A.

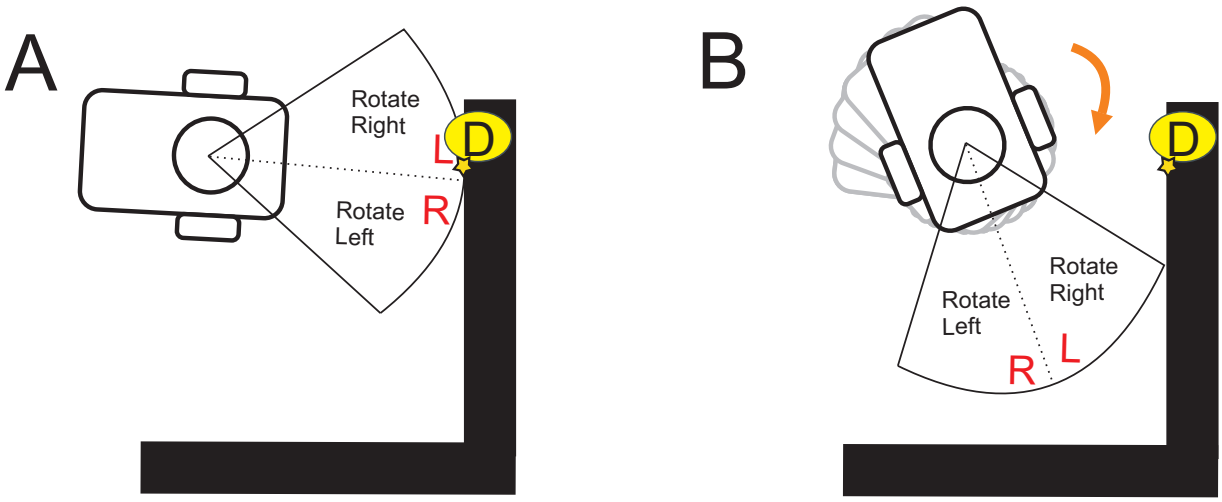

Figure 3.9: A: robot detects a disturbance *D* almost directly ahead in partition *L*. B: robot rotates right in response to disturbance and now has to negotiate a second obstacle.

as illustrated in Figure 3.10. In other words, we need an action-based mental model of the environment to reason effectively about predicted task outcomes. By doing this we can generate efficient sequences of closed-loop tasks (i.e., plans) for obstacle avoidance and ensure unwanted sequential behaviours do not arise, such as the robot doubling back on itself repeatedly. As we restrict our approach to rotations of $\pm\frac{1}{2}\pi$ radians in this thesis, then to negotiate two disturbances in one plan, a depth of at least four is required for the tree in Figure 3.8, which represents the Braitenberg-style example illustrated in Figure 3.7A. The robot must execute a rotation, drive straight and then repeat this for the second disturbance, generating a plan consisting of four tasks. We limit the size of the task tree represented by our method to four tasks as we want to keep our model as small as possible to promote real-time performance.

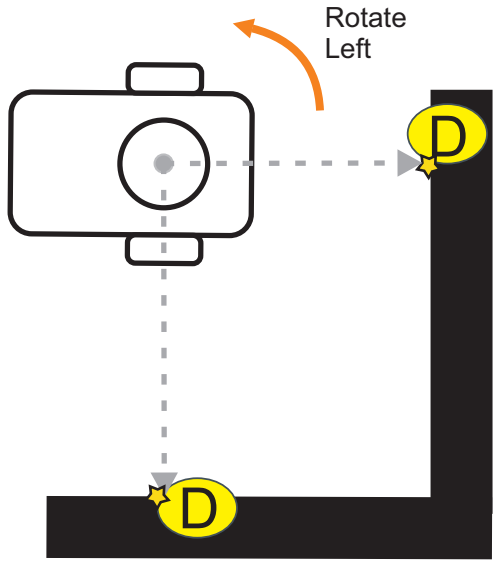

Figure 3.10: Egocentric situational analysis, reasoning about multiple disturbances.

Model-based reasoning methods are well-suited for the development of an action-based mental model which supports decision-making. Indeed, this is often the case in RL planning, where a discrete model (e.g., Markov Decision Process) is used for learning a reward function based on discretization of possible agent actions, typically represented by a regular grid. However, abstraction of the environment is often from a global rather than local perspective.

As mentioned in Section 2.5.1, model checking is a model-based verification approach which has been applied to planning problems. One useful feature of the approach is the ability to develop models which abstract away unnecessary details then perform exhaustive reasoning on a discrete model, typically about possible future sequences of states. Abstraction is often used to manage state-space explosion, which is a common issue for discrete planning methods. This suggests that model checking and abstraction could be used to develop appropriate action-based mental models which ignore irrelevant details of the environment to efficiently generate plans.

3.5 Model Checking as Planning

We now demonstrate how an off-the-shelf model checker can be used for planning. Our examples in this section use the SPIN model checker and are inspired by Pagojus et al. [154] (discussed in Section 2.5.1) in which SPIN was used to plan successive overtaking manoeuvres for an autonomous vehicle on a traffic heavy road. Note that we do not use SPIN for planning in this thesis as it is not optimised for real-time performance on a low-powered device (details of our approach are provided in Chapters 4 and 5). Instead, we use a purpose-built model checker *in situ* on our robot which is derived from model checking techniques supported by SPIN and research in [154]. In this section, we introduce only the bare essentials of the SPIN model checker and its associated modelling language Promela [70] required to explain our examples.

3.5.1 Promela basics

SPIN is a widely used open-source software verification tool. It can be used for the formal verification of multi-threaded software applications and supports the high-level state-based description language Promela (PROcess MEta LAnguage) which is loosely based on Dijkstra's guarded command language [55]. The core aspects of Promela are:

- Promela specifications consist of process templates called *proctypes*, global message channels and other global variables.
- Inline functions define a replacement text for a symbolic name, possibly with parameters. It does not define a new variable scope. The body of an inline function is directly pasted into the body of a proctype at each point of invocation.

- Each component type is declared within a `proctype` structure which contains local variables and statements.
- Each proctype instantiation represents the behaviour of an individual component.
- Guards (statements which may block, like boolean statements or channel operations) and choice statements control execution flow.
- Inter-component communication takes place via shared variables and channels. Our examples do not use this feature.
- Two choice constructs (`if...fi` and `do...od` statements) allow us to express nondeterminism in our model.
- Variable types include `bit`, `byte`, `short`, `int`, `array` and an enumerated type `mtype`.
- Initialisation information: i.e., how many and which instantiations of each proctype to create when the program is run. Processes are either initiated via a defined `init` process or via the `active` keyword.

Consider the simple Promela example shown in Figure 3.11. There are two processes, defined as instances of proctypes A and B via the `init` process. The `atomic` clause here ensures that they both start executing at the same time. We shall refer to the processes themselves as A and B for simplicity. There is a single (global) variable, `variable_1` which is incremented by process A and decremented by process B. Even this very simple example has several associated behaviours. Both processes are enabled in the initial state by guard `( variable_1 == 1 )` as the value of `variable_1` is 1. If process A executes two steps in succession (evaluating the guard and then incrementing `variable_1`), process B is then blocked as its guard no longer holds. Similarly process B can execute both of its steps and cause A to be blocked. Finally A and B can each execute their initial steps (in either order) and then they can each execute their remaining steps (in either order). There are thus six paths resulting from this specification.

3.5.2 SPIN verification

SPIN supports the specification and verification of properties in Linear Temporal Logic (LTL) [159]. LTL properties consist of a finite set of atomic propositions, boolean connectives, and temporal operators, e.g., $\Diamond$ ("eventually"), $\Box$ ("always") and **U** ("until"). However, we do not provide further details here, except to describe any property that we use in this section.

For the simple example in Figure 3.11 we might (incorrectly) assume that, for all paths, whenever `variable_1` is equal to 2 then it will eventually become equal to 1 again. This property (which we will refer to as ϕ) is stated as $\Box(r \rightarrow \Diamond t)$ where `r` and `t` are defined as

```
byte variable_1 = 1;
proctype A(){
    (variable_1 == 1) -> variable_1 = variable_1 + 1
}
proctype B(){
    (variable_1 == 1) -> variable_1 = variable_1 - 1
}
init{
    atomic{run A(); run B()}
}
```

Figure 3.11: Simple Promela example, two processes and one global variable

`variable_1==2` and `variable_1==1`, respectively. Note that universal quantification (i.e., "for all paths" ϕ) is implicit for LTL properties. It is therefore not possible to directly express existential properties (i.e., "for some path" ϕ) which identify specific paths in LTL.

When a Promela specification is compiled and run with SPIN, a global automaton consisting of all the initiated components is constructed. This is referred to as the *state-space* of the model. By searching the state-space, SPIN can capture any executions of the model that violate the property. In practice, an automaton combining the state-space and an automaton representing the property is checked for violating *cycles*, but we do not go into details of that process here.

For the simple Promela example shown in Figure 3.11 and the property ϕ described above, SPIN would indicate an error and return the first path for which the property was violated. The path where A executes its two steps and B is blocked would constitute such a violating path.

3.5.3 SPIN as a planner

To demonstrate how SPIN can be used as a planner, we refer to a popular puzzle known as the *River Crossing Puzzle*, which can be stated as follows:

> A farmer has a FOX, a GOAT, and a CABBAGE. He wants to cross a river, but his boat will hold only one item except himself. He cannot leave the FOX with the GOAT, or the GOAT with the CABBAGE. How can he get all THREE across?

This problem can be modelled in Promela using `do...od` statements to represent the choice of the farmer at each step in the puzzle. To describe our Promela specification, we first list the global variables, an `mtype` declaration and an inline function, as illustrated in Figure 3.12. Note that the `d_step` construct ensures that the statements contained within it are executed in such a way as to appear to occur simultaneously. Descriptions are provided in the comments.

An outline of the main proctype is shown in Figure 3.13 (the full proctype specification is provided in Appendix A). For the first two choices shown in Figure 3.13 we list the whole

```
/* Global variables */
mtype = {NULL,FOX,GOAT,CABBAGE}
mtype passenger=NULL; /*identity of passenger*/
byte foxP=0; /*position of fox, 0==left_bank, 1==on boat, 2==right_bank*/
byte goatP=0;
byte cabbageP=0;

byte temp;
bool goal = false; /* set to true if fox, goat and cabbage*/
/*are all at right_bank */

/*inline statement*/
inline swap(pos1,pos2){/*swap the values of variables pos1 and pos2*/
    temp = pos2;
    d_step{
    pos2=pos1;
    pos1=temp;
    }
}
```

Figure 3.12: Global variables for River Crossing puzzle specification

statement (i.e., the guard, the updates to variables, and a descriptive comment). However, for the remaining choices we include only the comment and omit the rest of the statement.

The conditions under which the fox cannot be left alone with the goat, or the goat left alone with the cabbage, is not encoded in the model. SPIN will therefore explore all executions in which the fox, goat and cabbage are carried back and forth across the river and finish when all three items are on the right bank. This will of course include illegitimate paths, however we can filter the search to include legitimate paths only using our LTL property.

For us to plan a sequence of actions to find a (legitimate) solution, we define propositions `badFriends1`, `badFriends2` and `safe`. We then combine the propositions into the LTL formula `property1`. The definition of propositions and the property is shown in Figure 3.14.

A solution path is a path in which the goal is reached (i.e., the fox, goat and cabbage are all on the right bank) and neither the fox and the goat, or the goat and the cabbage, have been left alone together on either the left or right bank. In other words, a path in which proposition `safe` is true in every state up to the point where `goal` is true. A solution path is therefore one in which `safe U goal` is true. We can find a path satisfying this property by finding a counterexample to the LTL property `!(safe U goal)`. By using Spin's *iterative search for short trail* option we can in addition find a solution path that is as short as possible. By running a guided simulation we can then observe the trail. Indeed, by judicious embedding of print statements in our Promela model (see Appendix A) we can easily extract the actions.

The solution to the puzzle is:

Pick up goat from left_bank
Deposit goat at right_bank
Pick up fox from left_bank
Deposit fox to right_bank and pick up goat

```
active proctype puzzle()
{

left_bank:
do
:: passenger==NULL && foxP==0 -> passenger=FOX; foxP=1;
   goto right_bank;
   /*pick up fox from left_bank*/
:: passenger==NULL && goatP==0 -> passenger=GOAT; goatP=1;
   goto right_bank;
   /*pick up goat from left_bank*/
:: ... /*pick up cabbage from left_bank*/
:: ... /*return fox to left_bank, pick up goat */
:: ... /*return fox to left_bank, pick up cabbage*/
:: ... /*return fox to left bank*/
:: ... /*return goat to left_bank and pick up fox*/
:: ... /*return goat to left_bank and pick up cabbage*/
:: ... /*return goat to left_bank*/
:: ... /*return cabbage to left_bank and pick up fox*/
:: ... /*return cabbage to left_bank and pick up goat*/
:: ... /*return cabbage to left_bank*/
od;


right_bank:
do
:: ... /*pick up fox from right_bank*/
:: ... /*pick up goat from right_bank*/
:: ... /*pick up cabbage from right_bank*/
:: ... /*deposit fox at right_bank  (not finished)*/
:: ... /*deposit goat at right_bank  (not finished)*/
:: ... /*deposit cabbage at right_bank  (not finished)*/
:: ... /*deposit fox to right_bank and pick up goat*/
:: ... /*deposit fox to right_bank and pick up cabbage*/
:: ... /*deposit goat to right_bank and pick up fox*/
:: ... /*deposit goat to right_bank and pick up cabbage*/
:: ... /*deposit cabbage to right_bank and pick up fox*/
:: ... /*deposit cabbage to right_bank and pick up goat*/
:: ... /*deposit fox at right_bank, go to finish*/
:: ... /*deposit goat at right_bank, go to finish*/
:: ... /*deposit cabbage at right_bank, go to finish*/
od;

finish: printf("\nFINISH\n"); goal=true

}
```

Figure 3.13: Main proctype for River Crossing puzzle specification

```
#define badFriends1 (foxP==goatP && !(cabbageP==foxP))
#define badFriends2 (goatP==cabbageP &&  !(foxP==goatP))
#define safe (!(badFriends1 || badFriends2))

ltl property1 { ! ( safe U goal) }
```

Figure 3.14: Propositions and LTL property for River Crossing Puzzle

Return goat to left_bank and pick up cabbage
Deposit cabbage at right_bank
Pick up goat from left_bank
Deposit goat at right_bank

Chapter 4

Real-Time Obstacle Avoidance

In this chapter, we describe our initial investigation into model checking for real-time planning and obstacle avoidance by generating sequences of closed-loop tasks. In Section 4.1, we motivate the use of a bespoke lightweight model checker implemented *in situ* on a real robot. We provide a high level conceptual overview of our approach in Section 4.2. In Section 4.3, we explain the details of our methodology. In particular, we explain how we abstract temporal aspects of task execution into static geometric relationships to reason about discrete task outcomes (from an egocentric perspective) and generate plans using model checking. Section 4.4 provides details on the implementation of our method. We summarise preliminary results for a cul-de-sac scenario in Section 4.5. In Section 4.6, we provide details on an in-depth case study of our approach. We draw conclusions on the merits and limitations of our method in Section 4.7. A glossary of variable names introduced in this chapter is available in Appendix B.

4.1 Motivation

The obstacle avoidance problem for an embodied robotic agent requires both avoiding obstacles and making progress towards a goal. However, achieving this using model checking in real-time is a non-trivial problem. As mentioned in Section 2.5.1, while there have been some planning applications of model checking validated using simulation, it is normally applied offline for system analysis, hence standard model checking tools are not optimised for real-time tasks.

Although standard model checkers may not be designed for real-time, there is reason to believe that this is feasible. As mentioned in Section 2.5.1, the SPIN model checker was used to plan overtaking manoeuvrers for simulation of an autonomous vehicle on rural roads in [154]. While the time delay was unacceptable (approx. 3 seconds), this was due to model compilation in SPIN and communication between the model checker and Unity gaming engine. The model checking algorithm itself only took around 20 milliseconds to generate plans, which is well within the 100 milliseconds deadline for AV decision-making discussed in Section 2.2.

This suggests that putting a model checker directly on a robot and eliminating the compila-

tion step may result in sufficient real-time planning performance. Although it is possible to put a pre-built model checker, such as SPIN, on a robot this would still require a model compilation step and moreover include additional functionality which is superfluous to need, consuming limited compute resources. One way to address this is to build a stripped-down model checker which avoids the compilation step by construction. The easiest way to do this, while optimising for real-time, is to integrate the model checking procedure into the code from scratch (i.e., build the model checker and model for planning directly into the robot code).

We therefore investigate the use of bespoke model checking for planning and obstacle avoidance, implemented *in situ* on a real robot. Specifically, we investigate the use of model checking to form sequences of closed-loop tasks by reasoning about their discrete outcomes in the context of hybrid planning and control. To simplify the problem, we restrict attention to avoiding static objects without making progress towards a goal. Hence we address the research question:

> **RQ1** *How can we use a bespoke implementation of model checking to plan sequences of discrete closed-loop tasks (which have an attentional focus on disturbances) for collision avoidance on a low-powered robotic system in real-time?*

4.2 Approach

Figure 4.1 shows the emergent behaviour of an agent which can only spawn a single closed-loop task in response to obstacles for a corner scenario. Initially, the robot is driving in a straight line in default task T_0 until it eventually encounters an obstacle in the environment, the disturbance D. During each iteration of the control loop, the robot scans the horizon for obstacles and continues driving straight if none are found. This generates an error signal when sensed by the robot.

To counteract the disturbance, the agent subsequently outputs a reflex motor action to change its state. This changes the state of the environment, which is in turn sensed by the robot and the loop repeats until the disturbance D is no longer in view. Hence the disturbance is eliminated. From a control theory point of view, this represents a negative feedback loop (see Section 3.3 for details). In Figure 4.1, this is the closed-loop task T_R, which has an attentional focus on disturbance D. Once eliminated, T_R is destroyed and the robot falls back on the default task T_0.

The task feedback loop is a simple mechanism where only information about the disturbance *at the present moment* can be encoded or retained. This means that a task is (i) memoryless and (ii) produces stereotyped behaviour. As discussed in Section 3.3, a control task has no information about whether other obstacles are present in the environment and can only generate a simple reflex in response to obstacles it encounters. This can lead to inefficient agent behaviour.

Figure 4.2 illustrates one example of a scenario where closed-loop obstacle avoidance on its own may produce undesirable robot motion. In both Figures 4.2A and 4.2B, the robot starts by executing the default task T_0. Through distal sensors, the robot senses that the space ahead

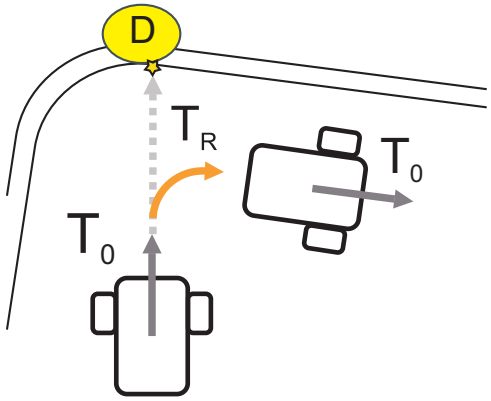

Figure 4.1: Robot encountering an obstacle, the disturbance D, in its environment.

contains an obstacle (i.e., the disturbance D_1) which it now must counteract by performing a control action. It can either rotate left or right, represented by tasks T_L and T_R, respectively. In this case, the task T_L is defined as rotating $\frac{1}{2}\pi$ radians, and task T_R as rotating $-\frac{1}{2}\pi$ radians.

There is no preference for either task, as each can achieve the control goal of eliminating the disturbance D_1. In Figure 4.2A, the robot initiates task T_R to rotate right until the obstacle, the disturbance D_1, is out of sight, which can then be safely followed by the default task T_0. However, the robot could have equally chosen to initiate task T_L and rotate left, as shown in Figure 4.2B. In this case, it would immediately face yet another obstacle D_2 after returning to the de-

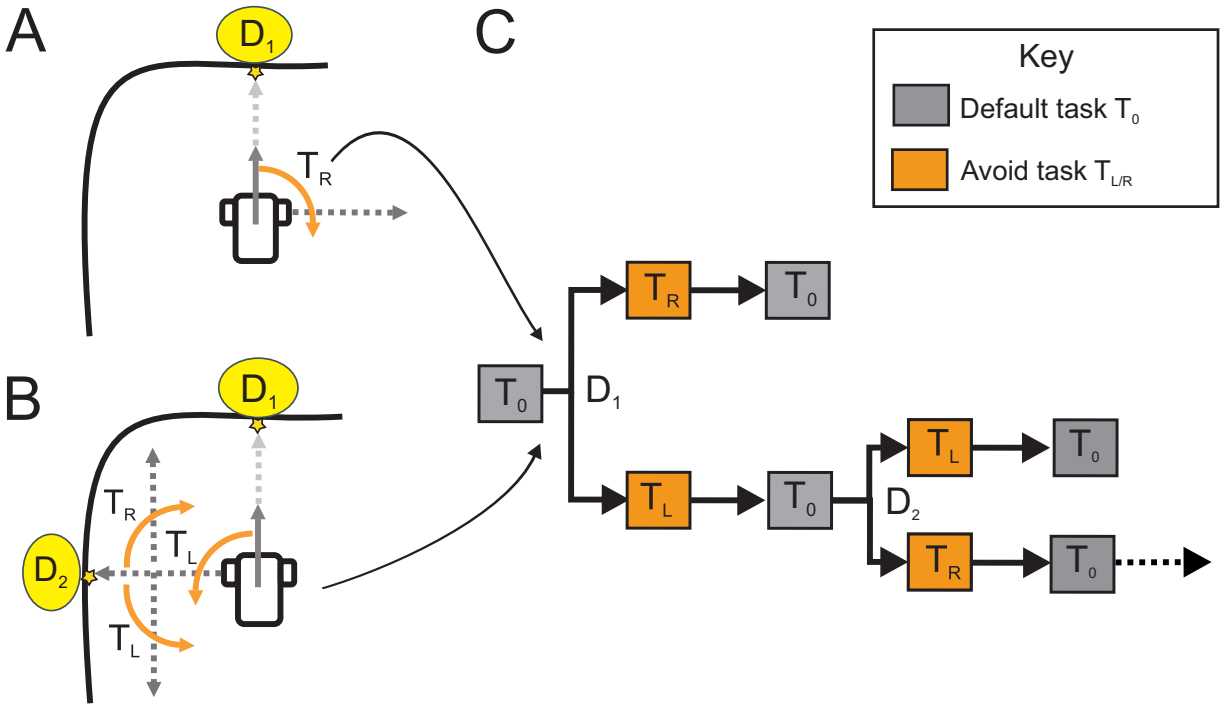

Figure 4.2: Uncertainty in corner situation from the perspective of a closed task feedback loop. A and B: valid options for eliminating the disturbance D_1 with varying consequences. C: behaviour tree for reasoning about multiple disturbances to form chains of closed-loop tasks.

fault task T_0. Consequently, the robot must again perform an avoidance manoeuvre. If the robot then initiates task T_L, it would escape the corner and continue in the default task T_0 undisturbed. If the robot initiates task T_R, however, yet another obstacle would be encountered. Hence there is a chance that the robot could get trapped in the corner, which is unwanted behaviour.

It is obvious to an external observer that the behaviour depicted in Figure 4.2A is more efficient and therefore desirable. As a closed feedback loop only has information about the present disturbance, reliably choosing the most efficient option for obstacle avoidance therefore requires extending the scope of environment observation beyond the present task. In this initial investigation, we use model checking to explore *future* system states resulting from the execution of task sequences which satisfy a constraint for obstacle avoidance. Our aim is to reason about possible chains of closed-loop tasks, as illustrated by the behaviour tree shown in Figure 4.2C.

4.3 Methodology

In this section, we explain the details of our approach for real-time planning using model checking. We define the action space and set of control tasks for our approach in Section 4.3.1. In Section 4.3.2 we explain how we construct a safe zone around the robot and the semantics of task outcomes based on disturbances. In Section 4.3.3 we describe how we abstract closed-loop sequences using spatial relationships. We describe our transition system in Section 4.3.4. In Section 4.3.5, we explain the LTL property and how our method generates plans at runtime.

4.3.1 Action Space

We formally define the action space for our approach as the set of control tasks:

$$Act = \{T_0, T_S, T_L, T_R\} \tag{4.1}$$

introduced in Section 4.1. We define a second straight task T_S because there are possible situations where a straight task between two avoid tasks might be desirable for tactical manoeuvrers. This differs from the default task T_0 in that the robot expects it can only drive for a finite period. When in the default task T_0, the robot has the expectation that it can continue driving in a straight line for a potentially infinite period of time, i.e., until a disturbance is encountered in its path.

4.3.2 Discrete Task Outcomes

Tasks have discrete outcomes which are used for agent reasoning and planning using model checking, as explained in Section 3.3. The outcomes of tasks provide information on the success or failure of the real or predicted execution, which simplifies discrete reasoning while grounding our approach in information about the continuous evolution of the hybrid system. This

approach to discretization makes it possible to reason about spatial abstractions representing task execution using distal sensor information and simple geometric notions. In the remainder of this section, we ground the semantics of discrete task outcomes by defining partitions of the environment which witness the satisfaction of existential conditions in first order logic.

Avoid Tasks $T_{L/R}$

We first define a safe zone, an invariant subset of the state space surrounding the robot constructed to respect spatial requirements for the evolution of continuous rotations over time. The safe zone is indicated in Figure 4.3A by a dotted circle and is defined by the radius r, a distance from the origin of the point cloud to the furthest point on the robot plus some tolerance. No object should ever be in the safe zone so the robot can always rotate safely around the origin.

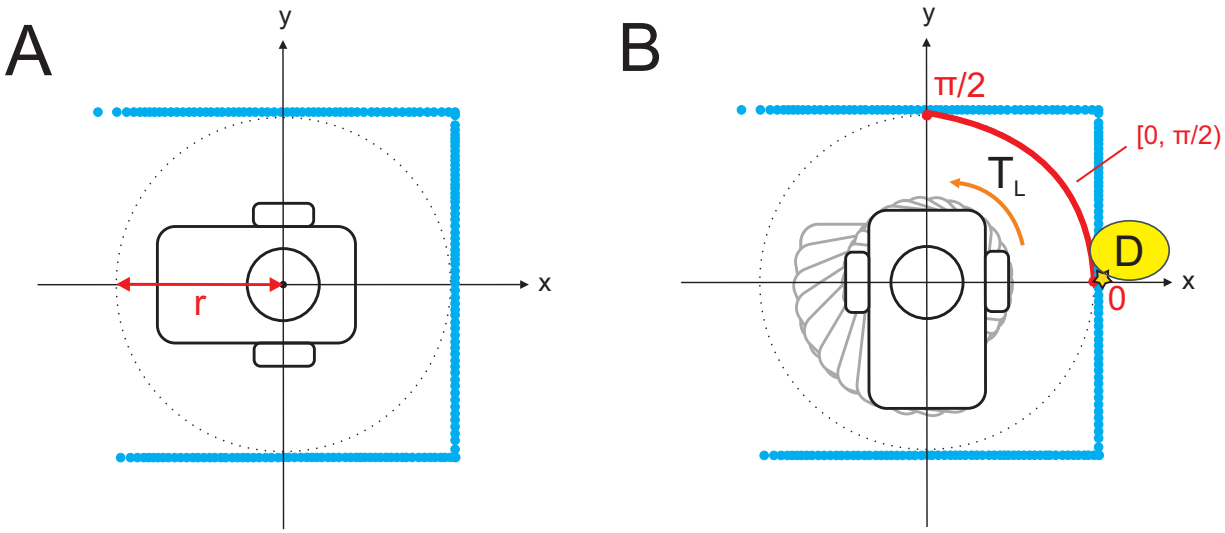

Figure 4.3: Construction of safe zone. A: the radius r defines dimensions of the safe zone (indicated by dotted circle). B: example of the robot executing task T_L until the absolute difference between the current and initial angle of the disturbance D is not in the partition P_{avoid}.

As indicated in Figure 4.3B, we define success for an avoid task $T_{L/R}$ as rotating until the absolute difference between the current and initial angle of the disturbance is *not* in the partition:

$$P_{avoid} = \{D \mid |D_\theta^{curr} - D_\theta^{init}| < \frac{1}{2}\pi\} \tag{4.2}$$

where D_θ^{curr} is the current relative angle to the disturbance and D_θ^{init} is the initial relative angle to the disturbance. Hence the task is a failure during execution until the empty set is a witness to the elimination of disturbance D. Table 4.1 summarises the possible outcomes for the partition. Hence success and failure can be expressed as satisfaction of the first order existential condition:

$$P_{avoid} \models \exists D(|D_\theta^{curr} - D_\theta^{init}| < \frac{1}{2}\pi) \tag{4.3}$$

Success	Failure
$P_{avoid} = \emptyset$	$P_{avoid} \neq \emptyset$

Table 4.1: Possible outcomes for avoid partition P_{avoid}.

Straight Task T_S

We next define success and failure criteria for the straight task T_S which has an attentional focus on *proximal* disturbances to trigger the next task in a plan. Figure 4.4 shows the construction of a partition for witnessing disturbances which ensures that the safe zone is always respected:

$$P_{shield} = \{o \in O \mid r < min(o_x) \leq r + d_{shield} \wedge |o_y| \leq \frac{1}{2}(L + tol)\} \tag{4.4}$$

where O is the set of observations for a scan event, r and $r + d_{shield}$ are lower and upper bounds, respectively, on the longitudinal dimension of observations, and the bound $\frac{1}{2}(L + tol)$ ensures that the lateral dimension of the partition respects the width of the robot plus some tolerance. Note that an observation o has two dimensions, where o_x denotes the x-coordinate and o_y the y-coordinate. We use this convention for observation subscripts throughout the thesis.

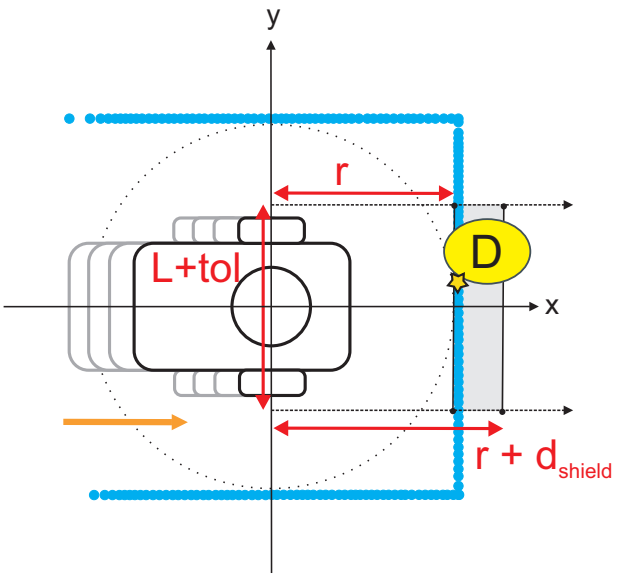

Figure 4.4: Construction of partition P_{shield} for witnessing proximal disturbances.

During each scan event, we iterate the set of observations O and select the observation with the nearest x-coordinate as our disturbance D. Table 4.2 summarises the possible outcomes.

Success	Failure
$P_{shield} = \emptyset$	$P_{shield} \neq \emptyset$

Table 4.2: Possible outcomes for shield partition P_{shield}.

Again success and failure can be expressed as satisfaction of a first order existential condition:

$$P_{shield} \models \exists D(r < D_x \leq r + d_{shield} \wedge |D_y| \leq \frac{1}{2}(L + tol)) \tag{4.5}$$

Default Task T_0

We can now define success and failure criteria for the default task T_0 which has an attentional focus on *distal* disturbances to trigger the generation of plans by model checking. Figure 4.4 shows construction of a look ahead partition for witnessing distal disturbances:

$$P_{look} = \{o \in O \mid r + d_{shield} < min(o_x) \leq d_{look} \wedge |o_y| \leq \frac{1}{2}(L + tol)\} \tag{4.6}$$

where O is the set of observations for a scan event, $r + d_{shield}$ and d_{look} are lower and upper bounds, respectively, on the longitudinal dimension of observations, and again the bound $\frac{1}{2}(L + tol)$ ensures that the lateral dimension respects the width of the robot plus some tolerance.

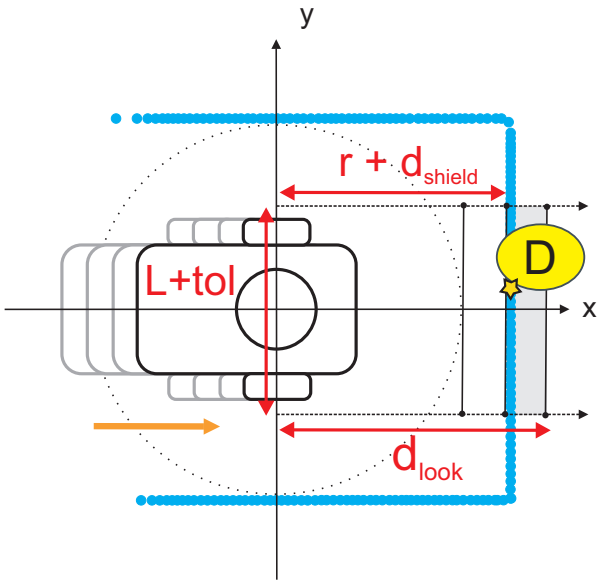

Figure 4.5: Construction of partition P_{look} for witnessing distal disturbances.

During each scan event, we iterate the set of observations O and select the observation with the nearest x-coordinate as our disturbance D. Table 4.3 summarises the possible outcomes.

Success	Failure
$P_{look} = \emptyset$	$P_{look} \neq \emptyset$

Table 4.3: Possible outcomes for look ahead partition P_{look}.

Success and failure can therefore be expressed as satisfaction of the existential condition:

$$P_{look} \models \exists D(r + d_{shield} < D_x \leq d_{look} \wedge |D_y| \leq \frac{1}{2}(L + tol)) \tag{4.7}$$

The partition P_{look} provides distal information on the disturbance D in the direction of travel. Failure of the default task T_0 therefore implies that a proximal disturbance will eventually be witnessed, which can be expressed as satisfaction of the temporal existential condition:

$$P_{shield} \models \Diamond \exists D(r < D_x \leq r + d_{shield} \wedge |D_y| \leq \frac{1}{2}(L + tol)) \tag{4.8}$$

As a consequence, the future path is finite, so the task now becomes the finite straight task T_S.

4.3.3 Abstract Closed-Loop Sequences

The semantics developed in the previous section make it possible to reason about task outcomes for a single disturbance during task execution. To also reason about multiple disturbances and *future* task outcomes, distal information about possible future disturbances is required. We achieve this by abstracting away the timed aspects of closed-loop tasks to represent their continuous evolution with spatial constructions, defined in relation to predicted future disturbances. We in addition require that the robot stays in the default task T_0 for as long as possible, a bias which we operationalise by ensuring that the robot prefers selecting the shortest safe sequence.

In the remainder of this section, we explain how we reason about the safety of task execution for two, three, and four step sequences of closed-loop tasks using geometric spatial abstractions. It is necessary to reason about the safety of task sequences in order of increasing length, as this allows us to limit the amount of computation required. If it is reasoned that there is a safe two step sequence, for example, there is no need to reason about sequences of greater length, as we have biased the robot towards selecting the shortest. Hence reasoning about safety is *cumulative*. Note that we restrict our approach to a maximum of four steps to keep the model as small as possible for efficient computation and yet expressive enough to avoid trap situations common for many local obstacle avoidance methods, in particular cyclic behaviour on U-shaped obstacles.

Two Step Sequences

The shortest possible sequence of tasks is two steps. This represents a situation where the robot encounters a disturbance, executes a single avoid task $T_{L/R}$ then returns immediately to the default task T_0 because no additional disturbances have been detected in the lateral dimension. An example of the robot executing the sequence $T_L \rightarrow T_0$ after encountering a longitudinal disturbance D_x is shown Figure 4.6A. As the safe zone represents the closed-loop behaviour of rotations, we can assume that rotations are safe, so it only remains to find a spatial representation for the discrete outcome (i.e., success or failure) of driving straight in the lateral dimension.

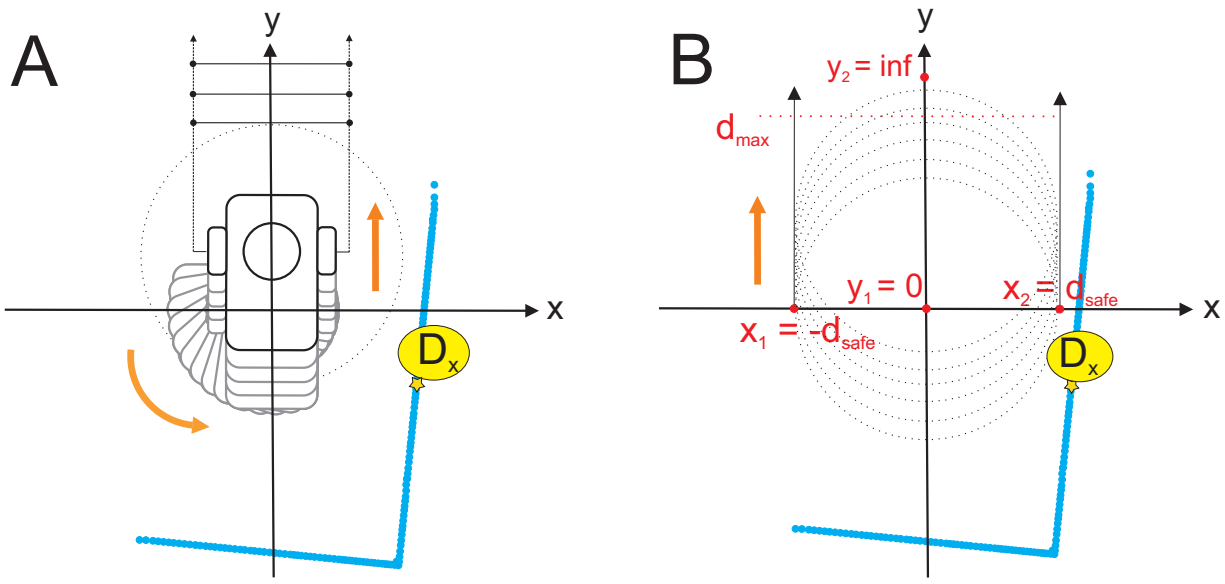

Figure 4.6: A: closed-loop execution of the two step sequence $T_L \rightarrow T_0$. B: spatial abstraction of the closed-loop sequence in A which in this case witnesses no disturbances.

To do this, we construct lateral partitions of the state space, finite over-approximations of future robot trajectories which represent driving straight in the lateral dimension, one for the positive (i.e., left) and one for the negative (i.e, right) direction. If a lateral partition contains no disturbance, we assume that the robot can drive straight in the associated direction for a (potentially) infinite period of time, hence the robot can return to the default task T_o

Prior to constructing a partition, we iterate over the set of observations O and subtract a longitudinal offset $\Delta_x = D_x - d_{safe}$ from the x-coordinates to simulate witnessing D_x as a proximal disturbance in the future. We can then iterate over the set of simulated observations and construct a lateral partition. First we construct a partition for the positive direction:

$$P_y^+ = \{o \in O \mid |o_x| \leq d_{safe} \wedge 0 < min(o_y) \leq d_{max}\} \tag{4.9}$$

where d_{safe} is a bound on the longitudinal dimension to respect the combined safe zone and outer shield $r + d_{shield}$, 0 is a lower bound on the lateral dimension, and d_{max} is an upper bound indicating a maximum lateral look ahead distance for witnessing distal disturbances. We use the parameter d_{safe} instead of the radius r to approximately model the future proximal detection of disturbances during execution of straight tasks, as this event triggers the next task in a plan. As shown in Figure 4.6B, the partition P_y^+ is empty, so returning to the default task T_0 is possible. Hence executing sequence $T_L \rightarrow T_0$ in response to the disturbance D_x is considered a success.

However, there could have been a second disturbance preventing the robot from returning to the default task, as illustrated in Figure 4.7A where the robot instead executes the sequence of tasks $T_R \rightarrow T_0$ in response to the longitudinal disturbance D_x. In this case, the robot encounters the negative lateral disturbance D_y^- so now needs to spawn another avoid task. As the geometry of task execution is symmetrical to the previous sequence, we can iterate over the set of

observations and construct a partition which is the negative lateral reflection of Equation 4.9

$$P_y^- = \{o \in O \mid |o_x| \leq d_{safe} \wedge -d_{max} \leq max(o_y) < 0\} \tag{4.10}$$

where d_{safe} is a bound on the longitudinal dimension respecting the combined safe zone and outer shield, 0 is an upper bound on the lateral dimension, and $-d_{max}$ is a lower bound indicating a maximum look ahead distance for witnessing distal disturbances in the negative direction. As shown in Figure 4.7B, the partition P_y^- is not empty, meaning that the y-coordinate of a disturbance in the negative lateral direction is greater than the parameter $-d_{max}$. It is therefore not possible to return to the default task T_0, so the sequence $T_R \rightarrow T_0$ is considered a failure.

Table 4.5 summarises the possible future outcomes for two step sequences in terms of the positive and negative lateral partitions defined in Equations 4.9 and 4.10, respectively.

Sequence	Success	Failure
$T_L \rightarrow T_0$	$P_y^+ = \emptyset$	$P_y^+ \neq \emptyset$
$T_R \rightarrow T_0$	$P_y^- = \emptyset$	$P_y^- \neq \emptyset$

Table 4.4: Possible outcomes for two step sequences.

As in the previous section, success or failure for either sequence (i.e., whether the robot can return to T_0) can be expressed as satisfaction of an existential condition over the relevant partition:

$$P_y^+ \models \exists D(|D_x| \leq d_{safe} \wedge 0 < D_y \leq d_{max}) \tag{4.11}$$

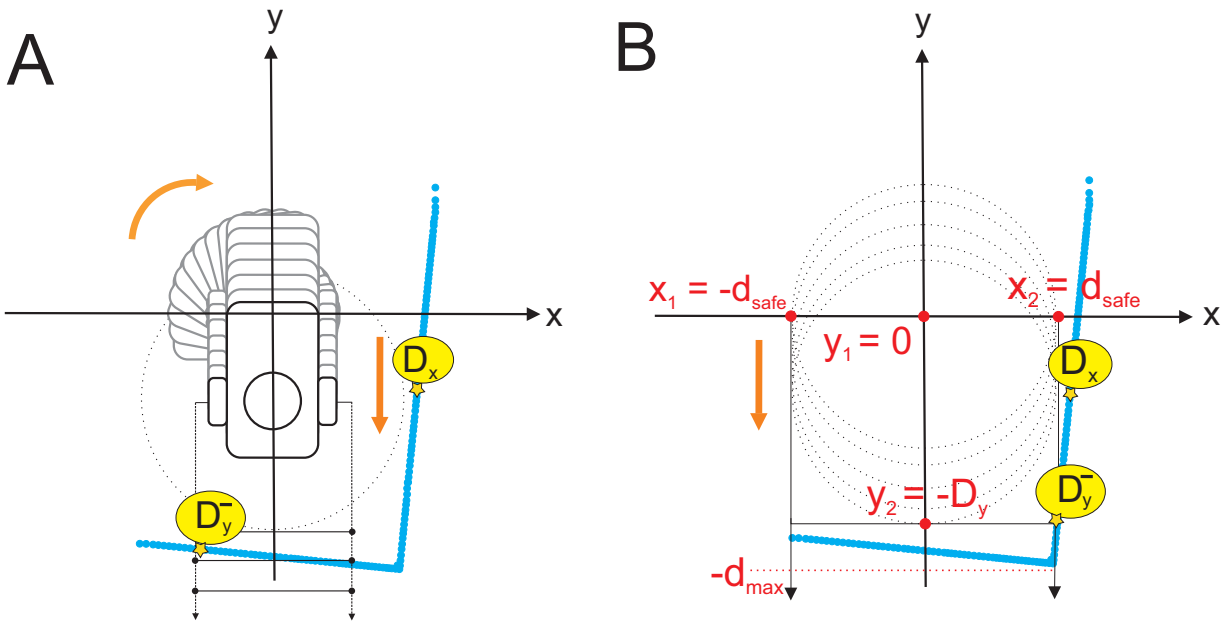

Figure 4.7: A: closed-loop execution of the two step sequence $T_R \rightarrow T_0$. B: spatial abstraction of the closed-loop sequence in A which in this case witnesses disturbance D_y^-.

$$P_y^- \models \exists D(|D_x| \leq d_{safe} \wedge -d_{max} \leq D_y < 0) \tag{4.12}$$

The lateral partitions provide distal information on disturbances in the lateral direction. Failure of a two step sequence therefore implies that a proximal disturbance will eventually be witnessed, which can be expressed as satisfaction of the existential condition in Equation 4.8. As the future path is finite in the relevant lateral direction, the appropriate straight task is therefore T_S. The procedure for constructing the lateral partitions at runtime is provided in Algorithm 1.

Algorithm 1: Construct Lateral Partition

Input : Simulated observations O, parameter d_{safe}, parameter d_{max}, and boolean flag *positive* indicating whether the partition should be negative or positive.

Output: The positive or negative lateral partition P.

```
1  Procedure ConstructPY(O, d_safe, d_max, positive)
2      P ← ∅
3      for o ∈ O do
4          if |o_x| ≤ d_safe then
5              if positive ∧ 0 < o_y ≤ d_max then
                   /* positive partition                    */
6                  add o to P
7              else if ¬positive ∧ −d_max ≤ o_y < 0 then
                   /* negative partition                    */
8                  add o to P
9      P ← FilterNearest(P, d_max)
10     return P
11 Procedure FilterNearest(P, d_max)
12     P′ ← ∅
13     D ← ∅
14     min ← d_max
15     for o ∈ P do
16         if |o_y| < min then
17             min ← |o_y|
18             D ← o
19     add D to P′
20     return P′
```

Three Step Sequences

The next possible sequence length is three steps, representing occasions where the robot could become boxed in. Figure 4.8A shows the robot executing the sequence $T_L \rightarrow T_L \rightarrow T_0$ to turn around and go back the way it came, because it has reasoned that it has no room to manoeuvre in the lateral dimension. As the robot has just come from this direction, and we restrict attention

to static environments, we can assume returning to the default task after turning around is safe.

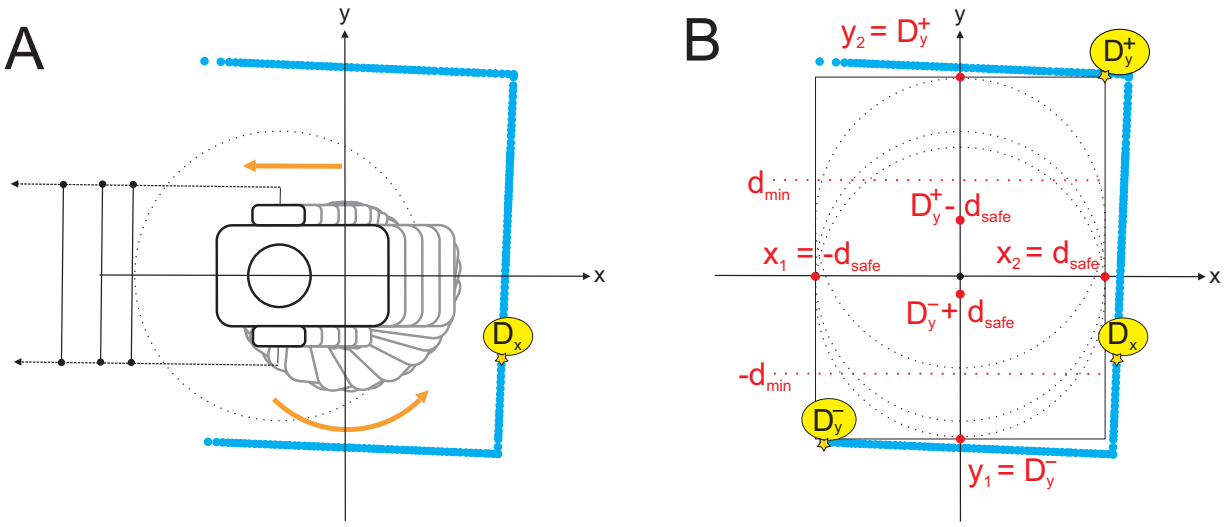

Figure 4.8: Simulation of three step sequence. A: closed-loop execution of the sequence $T_L \rightarrow T_L \rightarrow T_0$. B: the union of positive and negative lateral partitions.

Figure 4.8B shows abstraction of the closed-loop behaviour into geometric spatial relationships. In this case, we use the lateral dimension to reason about whether the robot can move distance d_{min} in either direction. As Equations 4.11 and 4.12 are both satisfied for this scenario, the union $P_y^+ \cup P_y^-$ models the conjunction. Hence Equations 4.11 and 4.12 together form independently necessary but jointly sufficient conditions for reasoning about whether the robot has enough room to manoeuvre. In conjunction with a further condition on the union:

$$P_y^+ \cup P_y^- \models \exists D(0 < D_y - d_{safe} < d_{min}) \wedge \exists D(-d_{min} < D_y + d_{safe} < 0) \tag{4.13}$$

it is then possible to reason if the robot can move in some lateral direction. Equation 4.13 checks whether the robot can move at least distance d_{min} in both directions while respecting the robot safe zone. In our example, the condition is satisfied, so we can reason that it is unsafe to drive in either lateral direction. Hence a sequence of length three is generated to turn around and evade.

Four Step Sequences

The final possible sequence length is four steps, representing a situation where there are lateral disturbances in both directions but Equation 4.13 is not satisfied. Hence the robot can drive safely in some lateral direction for a while but then must counteract a second disturbance. Figure 4.9 shows the robot executing the four step sequence $T_L \rightarrow T_S \rightarrow T_L \rightarrow T_0$ which includes an intermediate straight task. As we have reasoned that the robot can drive safely in the lateral direction, we now only need to consider whether the final subsequence $T_L \rightarrow T_0$ is possible.

We first simulate the lateral displacement of the robot, i.e., as if it had executed the initial subsequence $T_L \rightarrow T_S$. We calculate the lateral offset $\Delta_y^+ = D_y^+ - d_{safe}$ to simulate the expected

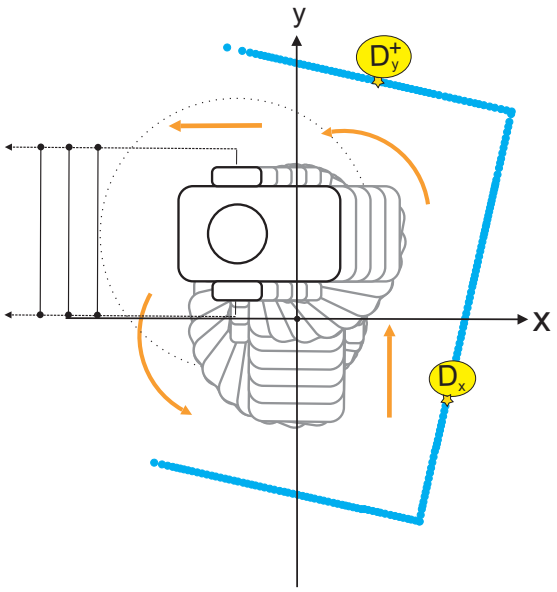

Figure 4.9: Robot executing the four step sequence $T_L \rightarrow T_S \rightarrow T_L \rightarrow T_0$.

state change while respecting the safe zone and outer shield, as shown in Figure 4.10A. As we are reasoning about spatial abstractions, lateral translation of the state space is sufficient, so it is unnecessary to simulate rotations. It is now possible to reason about the final subsequence $T_L \rightarrow T_0$ in the longitudinal dimension by constructing longitudinal partitions. The procedure is the same whether we have simulated driving straight in either the positive or negative direction.

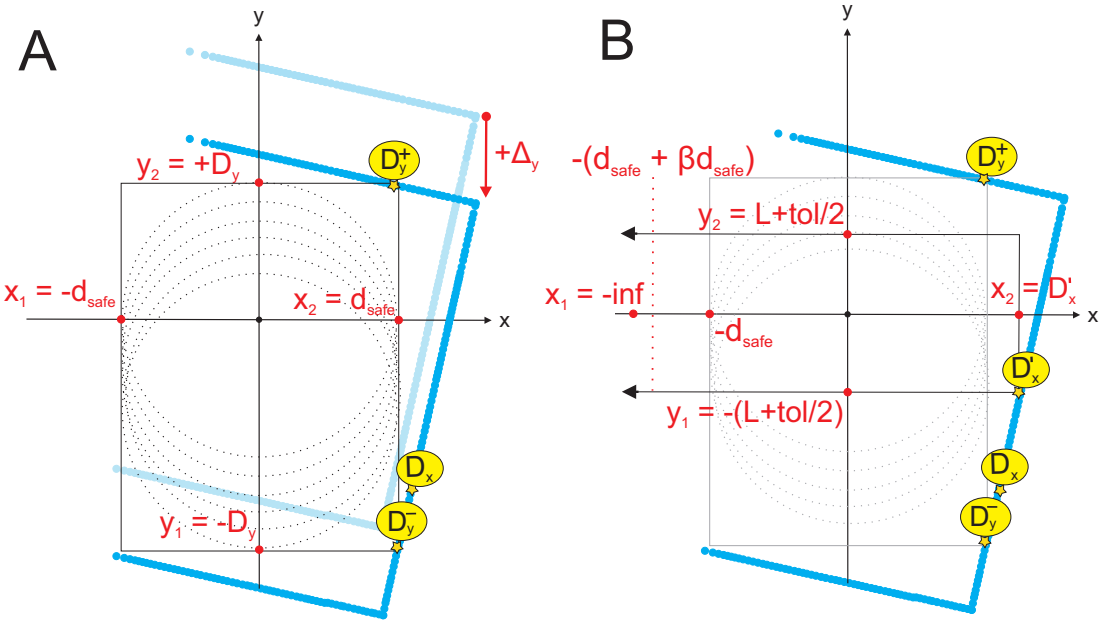

Figure 4.10: A: simulation if lateral displacement for the subsequence $T_L \rightarrow T_S$. B: abstraction for longitudinal movements to check if the final subsequence $T_L \rightarrow T_0$ is possible.

We iterate over the set of observations and construct the positive partition

$$P_x^+ = \{o \in O \mid d_{safe} < o_x \leq d_{safe} + \beta d_{safe} \wedge |o_y| \leq \frac{1}{2}(L + tol)\} \tag{4.14}$$

where d_{safe} is a lower bound on the longitudinal dimension which excludes the lateral partition, $d_{safe} + \beta d_{safe}$ is an upper bound with a tunable coefficient β, and $\frac{1}{2}(L + tol)$ is a bound on the lateral dimension to respect the width of the robot plus some tolerance. As indicated in Figure 4.10B, the partition P_x^+ witnesses a new disturbance D'_x, so we can conclude that driving in the positive direction has a finite lifetime, hence we cannot return to the default task T_0.

We then iterate over the set of observations and construct the negative partition

$$P_x^- = \{o \in O \mid -(d_{safe} + \beta d_{safe}) \leq o_x < -d_{safe} \wedge |o_y| \leq \frac{1}{2}(L + tol)\} \tag{4.15}$$

where $-(d_{safe} + \beta d_{safe})$ is a lower bound on the longitudinal dimension with the tunable coefficient β, $-d_{safe}$ is an upper bound which again excludes the lateral partition, and $\frac{1}{2}(L + tol)$ is again a bound on the lateral dimension to respect the width of the robot. As shown in Figure 4.9D, the negative longitudinal partition P_x^- is empty, so the robot can conclude that driving straight in the negative longitudinal direction has a (potentially) infinite lifetime. Hence it is possible for the robot to return to the default task T_0 which is the desired outcome.

Table 4.5 summarises the possible future outcomes for final subsequences in terms of the positive and negative longitudinal partitions defined in Equations 4.14 and 4.15, respectively. While our example has focused on the positive lateral direction, both directions are assessed for completeness. The same procedure applies to the negative direction by a symmetrical argument. Note that we refer to longitudinal partitions in the positive lateral direction as $P1$ and $P2$ in the negative lateral direction, however this distinction is omitted where directionality is irrelevant.

Lateral Offset	**Subsequence**	**Success**	**Failure**
$\Delta_y^+ = D_y^+ - d_{safe}$	$T_R \to T_0$	$P1_x^+ = \emptyset$	$P1_x^+ \neq \emptyset$
	$T_L \to T_0$	$P1_x^- = \emptyset$	$P1_x^- \neq \emptyset$
$\Delta_y^- = D_y^- + d_{safe}$	$T_R \to T_0$	$P2_x^- = \emptyset$	$P2_x^- \neq \emptyset$
	$T_L \to T_0$	$P2_x^+ = \emptyset$	$P2_x^+ \neq \emptyset$

Table 4.5: Possible outcomes for final subsequence of four step sequences

As for the lateral dimension, success for either subsequence can be expressed as satisfaction of an existential condition in first order logic over the relevant longitudinal partition:

$$P_x^+ \models \exists D(d_{safe} < D_x \leq d_{safe} + \beta d_{safe} \wedge |D_y| \leq \frac{1}{2}(L + tol)) \tag{4.16}$$

$$P_x^- \models \exists D(-(d_{safe} + \beta d_{safe}) \leq D_x < -d_{safe} \wedge |D_y| \leq \frac{1}{2}(L + tol)) \tag{4.17}$$

The runtime procedure for constructing longitudinal partitions is provided in Algorithm 2.

Algorithm 2: Construct Longitudinal Partition

Input : Simulated observations O, lateral offset Δ_y, parameter d_{safe}, width of the robot plus some tolerance $L+tol$, boolean flag *positive* indicating sign of partition.

Output: The positive or negative longitudinal partition P.

```
1  Procedure ConstructPX(O, Δ_y, d_safe, L+tol, positive)
2      P ← ∅
3      for o ∈ O do
4          o_y ← o_y − Δ_y                          // simulate lateral displacement
5          if |o_y| ≤ ½(L+tol) then
6              if positive ∧ d_safe < o_x ≤ d_safe + β d_safe then
                   /* positive partition                          */
7                  add o to P
8              else if ¬positive ∧ −(d_safe + β d_safe) ≤ o_x < −d_safe then
                   /* negative partition                          */
9                  add o to P
10     return P
```

4.3.4 Disturbance-Focused Transition System

The abstraction approach discussed in the previous section motivates the construction of our transition system. A state in our model has the from $s = [x, y, \theta]$ where x represents the longitudinal dimension, y represents the lateral dimension, and θ represents the orientation of the robot in radians. Figure 4.12 shows relevant states for planning two step sequences. To ensure the combined safe zone and outer shield is respected, we offset all states in the transition system distance d_{safe} from either the origin or a lateral disturbance. Hence we have one initial state in our model $s_0 = [d_{safe}, 0, 0]$ which is offset distance d_{safe} from the origin of the vector space. As our approach is egocentric, the orientation 0 radians is indexical to the direction the robot is facing, hence all paths in our model start from the initial state s_0. The states $s_1 = [0, d_{safe}, \frac{1}{2}\pi]$ and $s_2 = [0, -d_{safe}, -\frac{1}{2}\pi]$ represent transitions T_L and T_R, respectively, from s_0 and are fixed. See Figure 4.11 for an illustration of the distribution of states s_0, s_1 and s_2 relative to the robot.

States s_3 and s_4 represent driving straight in the lateral direction from states s_1 and s_2, respectively, but their valuation varies depending on the location of lateral disturbances. As shown in Figure 4.12A, if there are none, then as the lateral partitions are empty, we assume that $s_3 = [0, \infty, \frac{1}{2}\pi]$ and $s_4 = [0, -\infty, -\frac{1}{2}\pi]$, meaning that it is possible for the robot to drive in either direction for a (potentially) infinite period of time, so the relevant transition to s_3 or s_4 is the default task T_0. Hence both Equations 4.11 and 4.12 are not satisfied, meaning that a two step sequence in either direction is possible. However, if there are lateral disturbances, both states

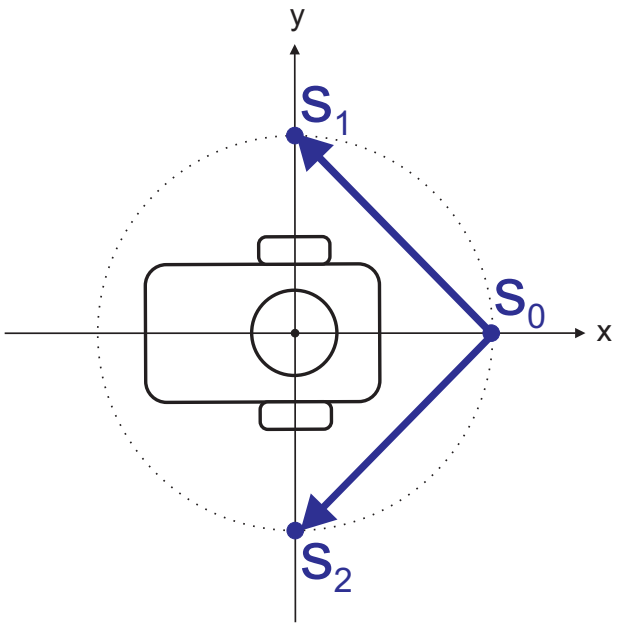

Figure 4.11: Distribution of states s_0, s_1 and s_2 relative to the robot.

would have a finite valuation such that $s_3 = [0, D_y^+, \frac{1}{2}\pi]$ and $s_4 = [0, D_y^-, -\frac{1}{2}\pi]$, as shown in Figure 4.12B. As the lateral partitions are not empty, the robot can only drive for a finite amount of time in either direction, so the relevant transition is the task T_S. In this case, both Equations 4.11 and 4.12 are satisfied, meaning that a two step sequence in either direction is *not* possible.

If this is the case, then we consider the possibility of a three step sequence. Figure 4.13 shows the addition of two additional states for this purpose. State $s_{13} = [-d_{safe}, 0, \pi]$ is fixed in our model and represents the execution of a second avoid task $T_{L/R}$ from either state s_1 or s_2.

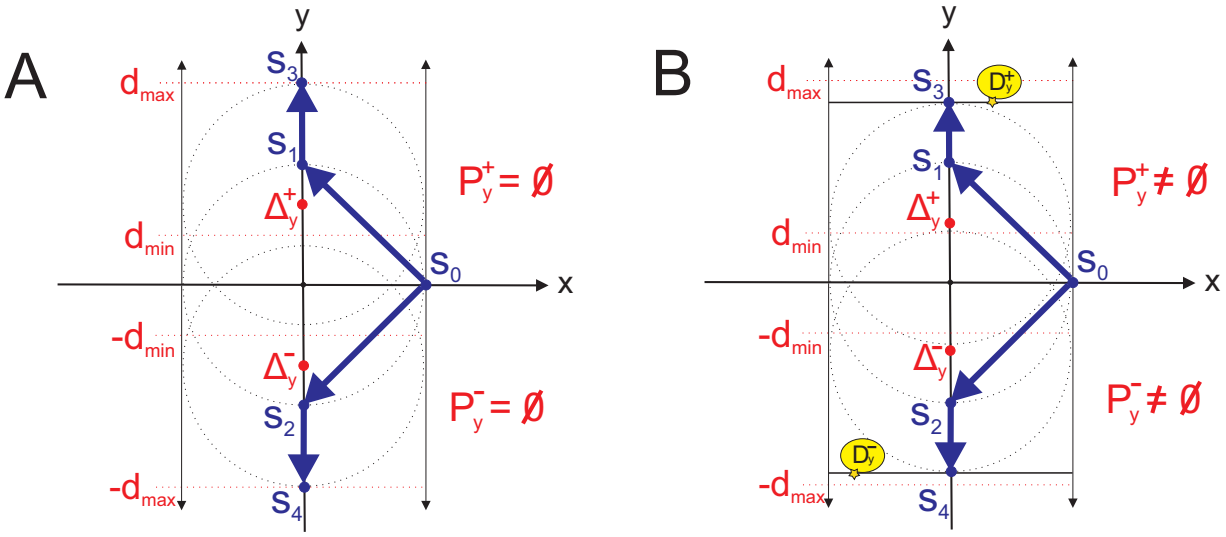

Figure 4.12: States for two step sequences. A: lateral partitions with no distal disturbances detected. Hence two step sequences are possible. B: lateral partitions with distal disturbances detected, resulting in a finite valuation for states. Hence two step sequences are not possible.

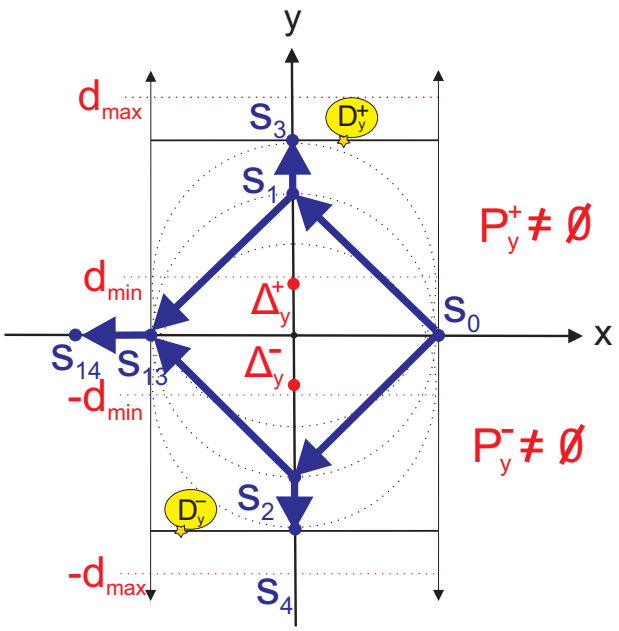

Figure 4.13: States for three step sequences.

The state $s_{14} = [-(d_{safe} + \beta d_{safe}), 0, \pi]$ represents the final transition T_0 which is assumed safe in our approach, as the robot should be able to go back the way it came in static environments without encountering a disturbance. Starting from the initial state s_0, this would result in either the sequence $T_L \rightarrow T_L \rightarrow T_0$ or $T_R \rightarrow T_R \rightarrow T_0$, which is chosen non-deterministically, as either sequence would result in the robot going back the way it came. If Equation 4.13 is satisfied, then a three step sequence is possible, otherwise we move on to consider a four step sequence.

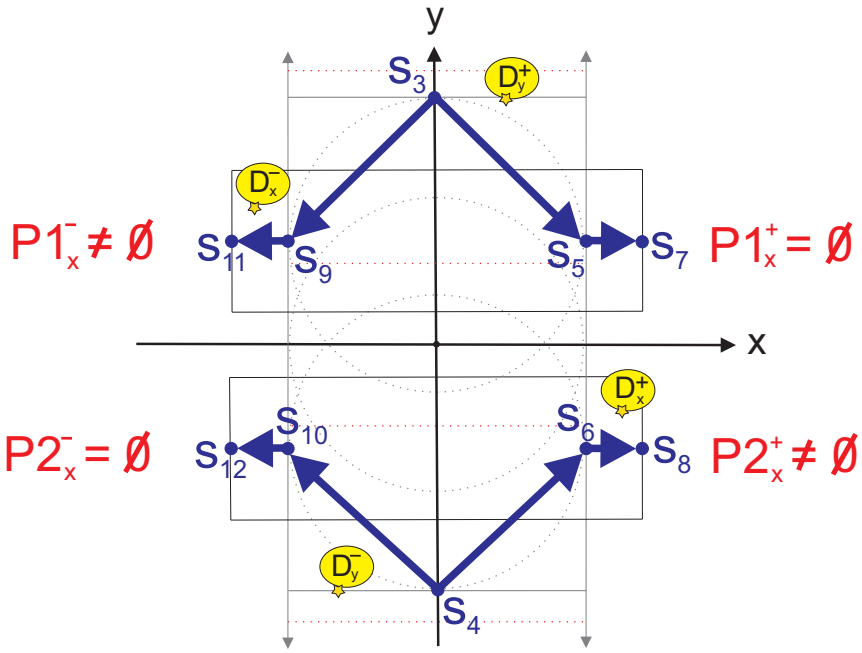

Figure 4.14: States for final subsequence of four step sequence.

Figure 4.14 shows the distribution of states for assessing longitudinal movements, representing the final subsequence in four step sequences, as discussed in Section 20. States $s_9 = [-d_{safe}, D_y^+ - d_{safe}, \pi]$ and $s_5 = [d_{safe}, D_y^+ - d_{safe}, 0]$ represent the transitions T_L and T_R, respectively, from state s_3 in the positive lateral direction. The longitudinal dimension is fixed but the lateral dimension varies relative to the location of the positive lateral disturbance D_y^+. If the longitudinal partitions are empty then $s_{11} = [-\infty, D_y^+ - d_{safe}, \pi]$ and $s_7 = [\infty, D_y^+ + d_{safe}, 0]$, otherwise $s_{11} = [-(d_{safe} + \beta d_{safe}), D_y^+ - d_{safe}, \pi]$ and $s_7 = [d_{safe} + \beta d_{safe}, D_y^+ - d_{safe}, 0]$. In the former case, Equations 4.16 and 4.17 are satisfied, so we can assume the robot can drive in either longitudinal direction for a (potentially) infinite period of time, otherwise they are not satisfied, so driving straight in either direction is not possible. The same holds for the set of states $\{s_4, s_6, s_8, s_{10}, s_{12}\}$ in the negative lateral direction by a symmetrical argument.

The resulting transition system is depicted in Figure 4.15. Note that our transition system is a tree structure rather than a graph as states represent unique points on the plane in relation to the egocentric perspective of the robot. If a leaf state in the tree has an infinite dimension, then this means that future states are unknown, as the robot then returns to driving straight for a (potentially) infinite period of time in the default task T_0. The leaves of the tree represent a final check to ensure that the robot can do this for *at least* a finite period so it has time to replan should a disturbance be encountered thereafter. Our model only exists to reason about discrete disturbances within a finite distance from the robot. Hence we call our model for planning using model checking a *disturbance-focused* finite transition system which we define as follows.

Definition 1 (Disturbance-Focused Transition System) *A disturbance-focused transition system DTS is a tuple* $(S, Act, \rightarrow, I, AP, L)$ *where*

- $S = \{s_0, s_1, ..., s_{14}\}$ *is a set of states where the orientation of the robot is fixed, the longitudinal and lateral dimensions fixed for states* $\{s_0, s_1, s_2, s_{13}, s_{14}\}$*, and for the remaining states the longitudinal or lateral dimensions vary in relation to disturbances;*
- $Act = \{T_0, T_S, T_L, T_R\}$ *is the set of tasks defined in Section 4.3.1;*
- $\rightarrow \subseteq S \times Act \times S$ *is a transition relation where* $s \rightarrow s'$ *is admissible if and only if a task can evolve the model from state* s *to* s'*;*
- $I = \{s_0 = [d_{safe}, 0, 0]\}$ *is the initial state;*
- AP *is a set of atomic propositions;*
- $L : S \rightarrow 2^{AP}$ *is a labelling function which determines which elements of* AP *are true at each state* s*.*

As should be clear from above, states in our model receive a valuation (labelling) at runtime using the abstraction approach described in Section 4.3.3, based on the location of disturbances.

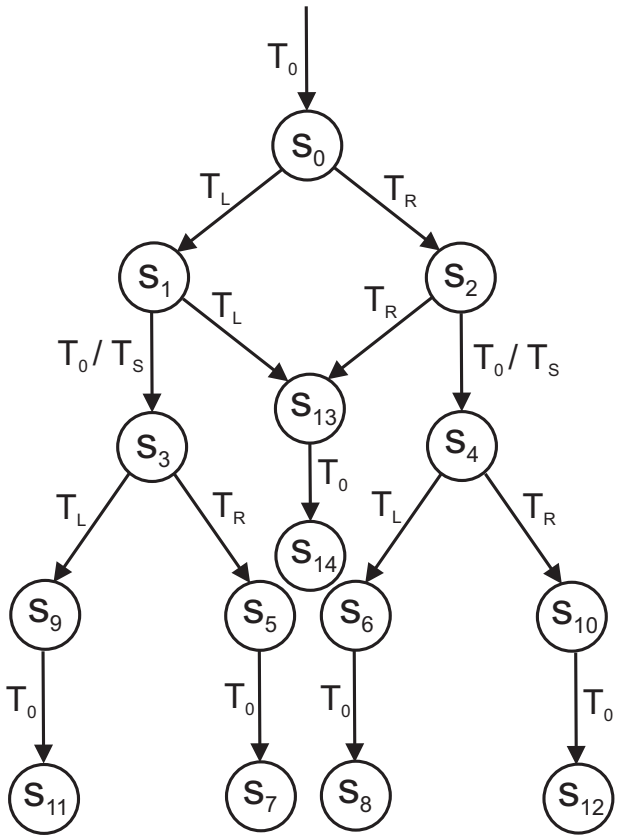

Figure 4.15: Disturbance-focused transition system. Note that transitions to potential final states $\{s_1, s_2, s_{11}, s_7, s_8, s_{12}\}$ represent a finite period of driving in the default task T_0 to ensure that the robot has time to replan if another disturbance is encountered. However, subsequent states in the default task T_0 are not represented as they are unknown (i.e., the future path is uncertain).

4.3.5 Planning Using Model Checking

We use the *disturbance-focused* transition system described in the previous section to generate plans for negotiating multiple disturbances using distal sensor information. To do this we define an LTL property, construct a non-deterministic finite automaton (NFA) and form the product transition system and NFA for a valuation of state properties for the sensed situation.

We implement the underlying directed graph for the finite transition system shown in Figure 4.15 and compute the product at runtime when a new planning sequence is initiated. Planning using model checking can be reduced to a simple reachability analysis on the product, which amounts to simply deciding which states in the graph are terminal. We do this in order of increasing sequence length to ensure reasoning about safety accumulates for each plan length.

LTL Plan Specification

We define a regular property in LTL which is stated in terms of a set of two state properties, *horizon* and $safe$. Before introducing the LTL property, we define these state properties.

Definition 2 *Property horizon is true at state s if and only if the valuation of either the longitudinal or lateral dimension is infinite.*

In Definition 2 an infinite dimension in the valuation of a state means that the robot can drive in a straight line for a (potentially) infinite period of time, so it can return to the default task T_0. If the future path of the robot seems infinite, then there is no known disturbance ahead, so by definition it is deemed safe. Our definition of the state property $safe$ is less straightforward.

Definition 3 *Property $safe$ is true at state s if one of the following conditions hold: (i) the absolute value of one finite dimension at s is d_{safe}; (ii) s has a finite lateral dimension such that the absolute offset from a lateral disturbance (i.e., $|D_y^+ - d_{safe}|$ or $|D_y^- + d_{safe}|$) is greater than parameter d_{min} and there is no longitudinal dimension; or (iii) proposition horizon is true at s.*

In Definition 3 condition (i) ensures rotations are safe due to construction of the safe zone and outer shield for witnessing proximal disturbances. Relevant states around the origin have one dimension equal to d_{safe}, and this uniquely identifies them. Condition (ii) ensures that the robot has space to move in the lateral direction while respecting the safe zone. The final condition (iii) guarantees that a state is safe when the future path seems infinite. In our approach, we always start in the initial state s_0 and assess each state for a possible path in order of preference, hence for any final horizon state we always know that the path leading to state s is finite.

Based on the explained semantics, we use an LTL property φ for planning, where

$$\varphi = \neg(safe\ \mathbf{U}\ (safe \wedge horizon)) \tag{4.18}$$

which states that $\neg(safe\ \mathbf{U}\ (safe \wedge horizon))$ holds for every path. Note that a finite path satisfies $safe\ \mathbf{U}\ (safe \wedge horizon)$ if both $safe$ and $horizon$ are true at some state s in the path, and $safe$ is true for all states in the path leading to s. As our interest is in *solution paths* (see Section 3.5 for details), the property φ negates the desired outcome, so that what would normally be the set of counterexamples for a run of the system, in our case becomes a set of safe *future discrete task outcomes*. Consequently, the set of solution paths is the set of paths which satisfy $safe\ \mathbf{U}\ (safe \wedge horizon)$. Our property and modelling approach is intentionally simple, primarily to ensure fast computation and minimal resource usage for reliability on a low-powered device.

Checking the Product Transition System

We are interested in the set of paths that satisfy the property $safe\ \mathbf{U}\ (safe \wedge horizon)$. The set of counterexamples for φ constitute a language of finite words which can be recognised by an NFA. We therefore construct the NFA $\mathscr{A}_{\neg\varphi} = (Q, \Sigma, \delta, Q_0, F)$ where $Q = \{q_0, q_1\}$ is the set of states, $\Sigma = 2^{AP}$ is a finite alphabet, $\delta : Q \times \Sigma \to 2^Q$ is a transition relation, $Q_0 = \{q_0\}$ is the initial state, and $F = \{q_1\}$ is the accepting state. Figure 4.16 shows the NFA $\mathscr{A}_{\neg\varphi}$.

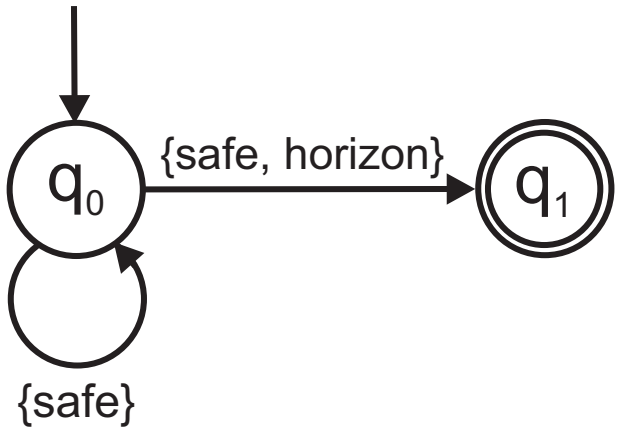

Figure 4.16: NFA for the property $safe\ \mathbf{U}\ (safe \wedge horizon)$.

When a new disturbance is encountered by the robot, we use the abstraction approach explained in Section 4.3.3 to assign a valuation to the set AP. We then calculate the automaton formed as the product of TS and $A_{\neg\varphi}$ and generate a (finite) accepting path non-deterministically using f-DFS. From the path we can then extract the transitions to form a plan for the scenario. Figure 4.17A shows an instance of our transition system with valuation of the elements of $\{horizon, safe\}$ and Figure 4.17B the product automaton. Note that accepting (solution) paths are those for which the NFA part of the state in Figure 4.17B is the accepting state q_1.

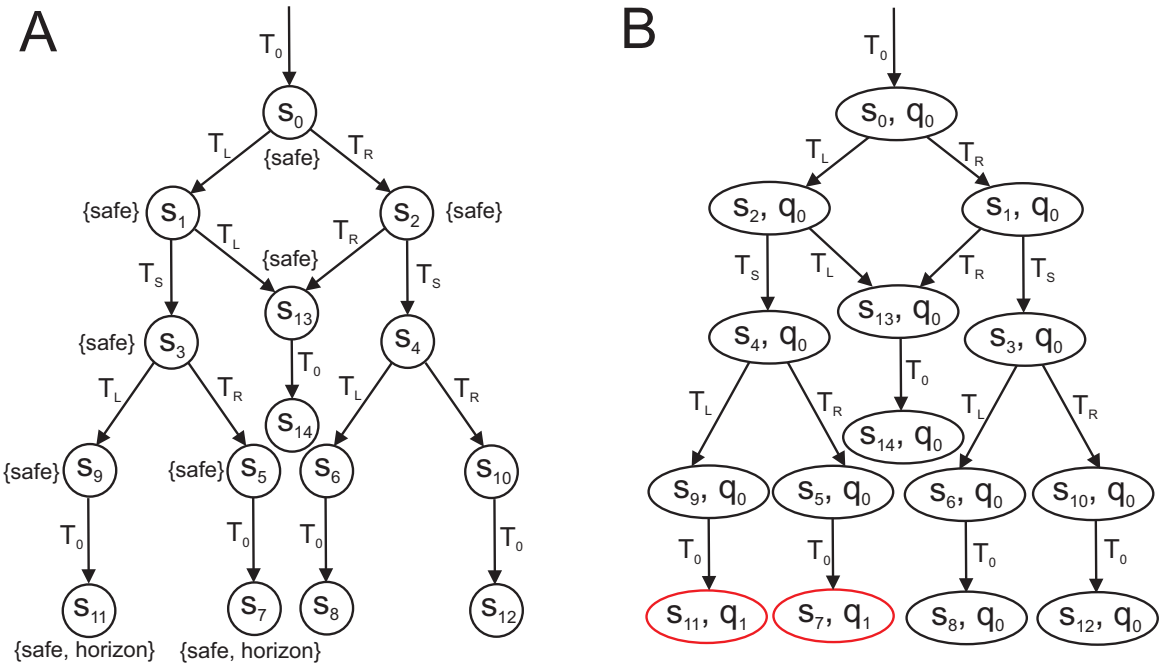

Figure 4.17: A: an instance of our transition system showing a valuation of the elements of $\{horizon, safe\}$. B: product transition system and NFA resulting from the valuation in A.

Runtime Generation of Plans

Algorithm 3 provides details on the runtime procedure for computing our abstraction and generating plans. As mentioned above, we assess possible sequences in order of length to determine a valuation of our transition system. Note that the subroutine *generatePlan()* finds a solution path by f-DFS on the product transition system and NFA. Starting from the initial state s_0, it then extracts the transitions between states form a plan consisting of a sequence of associated tasks. By default f-DFS does not return the accepting state of the product, so the final transition is not returned. However, as the final transition in a plan is always the default task T_0 this is sufficient.

4.4 Implemetation

Our approach was implemented on the low-powered robotic platform described in Section 3.1 following an event-driven paradigm using callbacks. Planning using model checking is used to reason about future task outcomes to form plans consisting of sequences of closed-loop tasks. We now provide an overview of the agent architecture and explain agent reasoning for current task outcomes during the real execution, which underpins the abstraction of closed-loop tasks.

4.4.1 Agent Architecture

Figure 4.18 shows the overall agent architecture. Our focus is on planning sequences of closed-loop control tasks in response to disturbances. The robotic agent mostly reasons reactively about the sensed environment state while planning using model checking reasons about predicted states using distal sensor information. This separation of concerns is reflected the architecture shown in Figure 4.18. Our agent is implemented in C++ using an event-driven paradigm.

4.4.2 Reactive Agent Reasoning

Figure 4.19 illustrates control flow for the task controller, which is closed-loop and comprises two functional components. We motive this with an example. Suppose our robot is executing the default task T_0 driving towards disturbance D which it has not seen. A new scan event from the LiDAR calls the agent event handler at a sample rate of ≈ 200 milliseconds, which immediately passes the set of observations to the task event handler, as indicated in Figure 4.19A.

The default task T_0 sends a signal to the motors and looks ahead for disturbances. As the disturbance D is currently out of range, it seems that the robot can continue driving straight for a (potentially) infinite period of time, so the existential condition in Equation 4.7 is not satisfied and the loop repeats. This continues until the look ahead partition witnesses a distal disturbance at which point the task is a failure, i.e., the existential condition in Equation 4.7 is satisfied. Reactive reasoning then takes place by the mechanism shown in Figure 4.19B.

Algorithm 3: Generate Plan Using Model Checking

Input : Set of observations O and disturbance D.
Output: Sequence of discrete control tasks *plan*.

Procedure `GenerateNewPlan(`O, D`)`
- $term \leftarrow \emptyset$ `// set of terminal states for product`
- $\Delta_x \leftarrow 0$ `// simulate proximal detection of` D
- **if** $\Delta_x > d_{safe}$ **then**
  - $\Delta_x = D_x - d_{safe}$
- $O' \leftarrow \emptyset$
- **for** $o \in O$ **do**
  - $o_x \leftarrow o_x - \Delta_x$
  - add o to O'
- $P_y^+ \leftarrow$ `ConstructPY(`O', d_{safe}, d_{max}, ***true***`)` `// asses two step sequences`
- $P_y^- \leftarrow$ `ConstructPY(`O', d_{safe}, d_{max}, ***false***`)`
- **if** $P_y^+ = \emptyset$ **then**
  - add s_3 to *term*
- **if** $P_y^- = \emptyset$ **then**
  - add s_4 to *term*
- **if** $P_y^+ = \emptyset \vee P_y^- = \emptyset$ **then**
  - $plan \leftarrow generatePlan(term)$
  - **return** *plan* `// assess three step sequences`
- **if** $D_y^+ \in P_y^+ < d_{min} \wedge D_y^- \in P_y^- > -d_{min}$ **then**
  - add s_{14} to *term*
  - $plan \leftarrow generatePlan(term)$
  - **return** *plan*
- $\Delta_y^+ \leftarrow D_y^+ - d_{safe}$ `// assess four step sequences`
- $\Delta_y^- \leftarrow D_y^- + d_{safe}$
- $P1_x^+ \leftarrow$ `ConstructPX(`O', Δ_y^+, d_{safe}, $L+tol$, ***true***`)`
- $P1_x^- \leftarrow$ `ConstructPX(`O', Δ_y^+, d_{safe}, $L+tol$, ***false***`)`
- $P2_x^+ \leftarrow$ `ConstructPX(`O', Δ_y^-, d_{safe}, $L+tol$, ***true***`)`
- $P2_x^- \leftarrow$ `ConstructPX(`O', Δ_y^-, d_{safe}, $L+tol$, ***false***`)`
- **if** $P1_x^+ = \emptyset$ **then**
  - add s_7 to *term*
- **if** $P1_x^- = \emptyset$ **then**
  - add s_{11} to *term*
- **if** $P2_x^+ = \emptyset$ **then**
  - add s_8 to *term*
- **if** $P2_x^- = \emptyset$ **then**
  - add s_{12} to *term*
- $plan \leftarrow generatePlan(term)$
- **return** *plan*

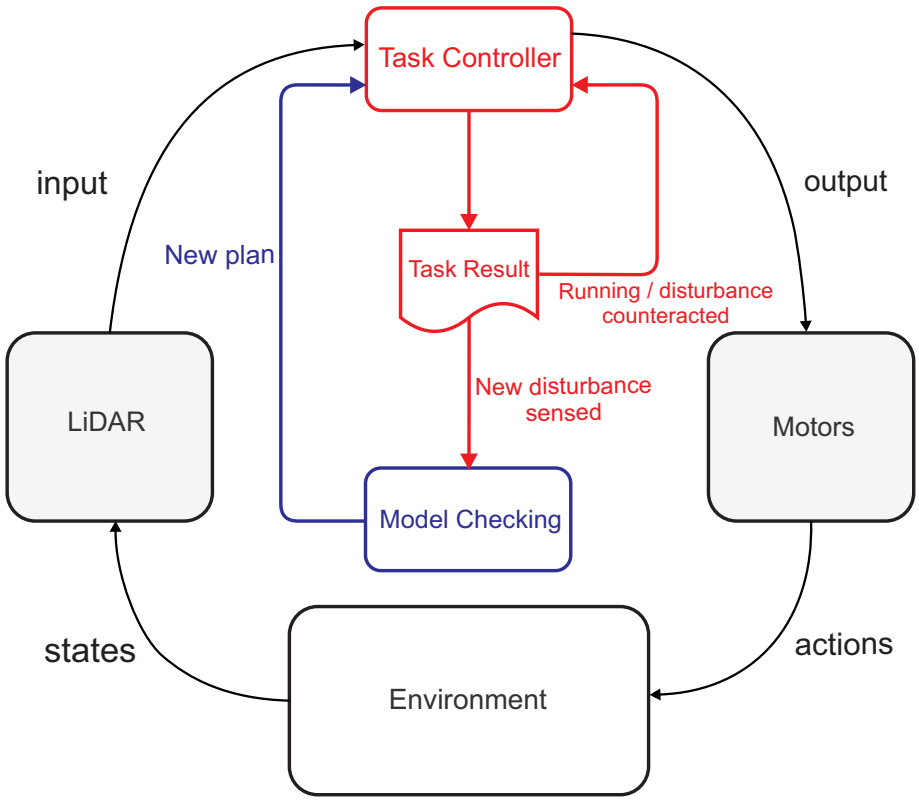

Figure 4.18: Agent architecture. Red shows reasoning based on the actual environment state. Blue shows reasoning by model checking about discrete outcomes based on predicted states.

Once the default task T_0 has failed, a plan is generated using model checking. The plan is then stored and execution of the current task continues. At this point, the robot has a finite expectation for the future path, so the look ahead partition is ignored until the safe zone shield partition witnesses a proximal disturbance, i.e., the existential condition in Equation 4.5 is satisfied. The first task in the plan is then initiated, which in our case is the avoid task T_L.

Each iteration of the control loop the task outcome interface shown in Figure 4.19A sends a signal to the motors and checks whether the existential condition in Equation 5.3 is *not* satisfied. When it evaluates true, the avoid task T_L is considered a success, at which point the mechanism in Figure 4.19B initiates the next task, or returns to the default task T_0 if the plan is empty.

4.5 Preliminary Results

Our research goal was to answer **RQ1**. For preliminary investigations into the feasibility of our approach, we compared our method for real-time planning using model checking to a baseline with a single closed feedback loop, resulting in stereotyped behaviour, as discussed in Section 4.1. We made two comparisons for a cul-de-sac scenario, in which we compared the resulting trajectories and recorded the real-time performance of planning using model checking. We

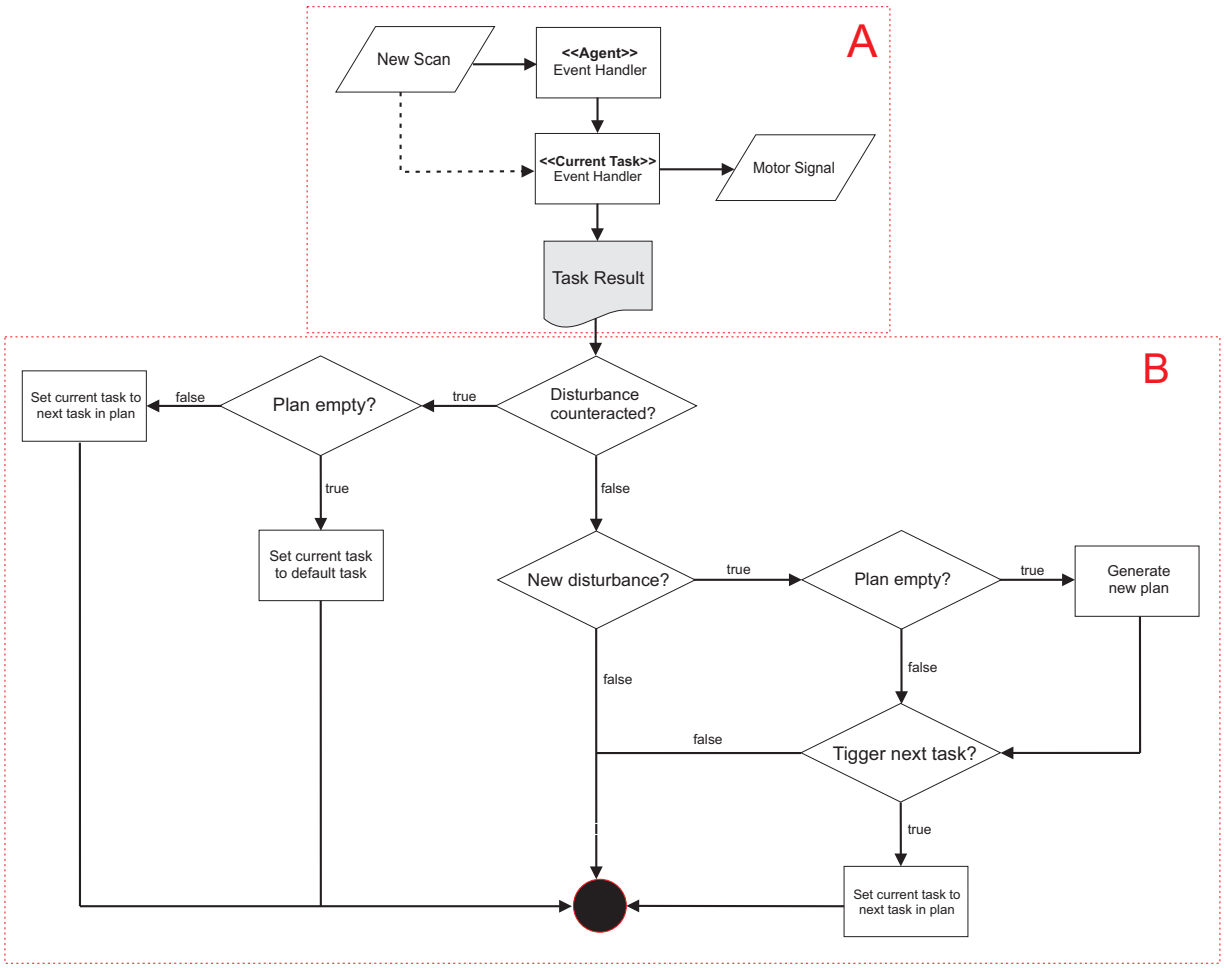

Figure 4.19: Reactive agent reasoning. A: interface for discrete task outcomes. B: reactive reasoning mechanism for initiating tasks or generating a new plan based on task outcomes.

extracted the emergent trajectories from video using the optical flow method in OpenCV[1] for comparison. In addition, we recorded processing latency for planning using model checking. Our aim was to meet real-time constraints, defined specifically as less than 100 milliseconds.

Comparison 1 Results are summarised in Figure 4.20. In Figure 4.20A, the robot approaches the cul-de-sac, senses a disturbance in the environment (i.e., a point on the wall to the left) and generates the four step plan $T_R \rightarrow T_S \rightarrow T_R \rightarrow T_0$ using model checking, illustrated in Figure 4.20B. Algorithm 3 determines that $P_y^+ \neq \emptyset$ and $P_y^- \neq \emptyset$, so a two step plan is not possible. Furthermore, the robot can drive at least $d_{min} = 0.5$ meters in either lateral direction so a three step plan is not necessary. Hence Equations 4.11 and 4.12 are satisfied and Equation 4.13 is not.

Consequently, it is necessary to construct longitudinal partitions and check if the final subsequence is safe for each lateral direction. In this case, Algorithm 3 determines that $P1_x^+ \neq \emptyset$ and $P1_x^- \neq \emptyset$, so the robot cannot return to the default task T_0 after initially driving in the positive lateral direction. Equations 4.16 and 4.17 are therefore satisfied. However, in the negative lateral

[1] https://opencv.org/

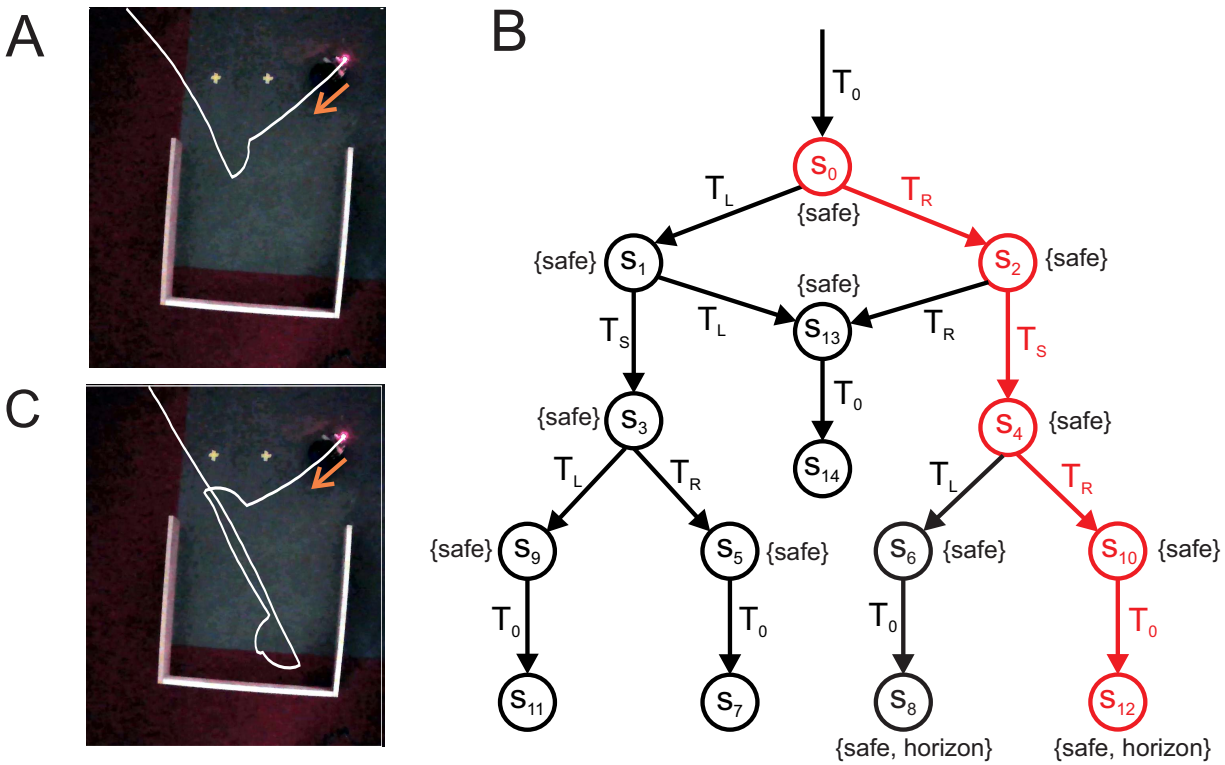

Figure 4.20: Comparison 1. A: trajectory for planning using model checking (last couple of steps cropped). B: the transition system and generated plan. C: trajectory of baseline method.

direction $P2_x^+ = \emptyset$ and $P2_x^- = \emptyset$, hence Equations 4.16 and 4.17 are not satisfied, so returning to the default task T_0 is possible. Both states s_8 and s_{12} are therefore marked as terminal.

A path to one of the states is chosen non-deterministically using model checking, in this case state s_{12}. Execution time for planning using in situ model checking was 9.22 milliseconds. Figure 4.20C shows the baseline, which manages to navigate out of the cul-de-sac by chance, however the trajectory is less efficient as the robot enters the cul-de-sac first. In comparison, the model checking approach does not enter the cul-de-sac at all, as shown in Figure 4.20A.

Comparison 2 Results are summarised in Figure 4.21. In Figure 4.21A, the robot approaches the bottom right corner of the cul-de-sac, senses a disturbance and infers that it is boxed in and generates the three step plan $T_L \rightarrow T_L \rightarrow T_0$ using model checking, illustrated in Figure 4.21B. Algorithm 3 has determined that $P_y^+ \neq \emptyset$ and $P_y^- \neq \emptyset$, however the robot cannot drive $d_{min} = 0.5$ meters in either direction. Hence Equations 4.11, 4.12 and 4.13 are all satisfied.

Consequently, a three step plan is necessary. State s_{14} is therefore assumed safe and one of the two possible three step sequences generated non-deterministically using model checking. Execution time for planning using in situ model checking was 10.87 milliseconds. Figure 4.21C shows the baseline, which eventually navigates out of the cul-de-sac by chance, but gets trapped between two walls for a significant period and collides with one of them. Results suggested that model checking generates more efficient trajectories and can successfully avoid a trap situation.

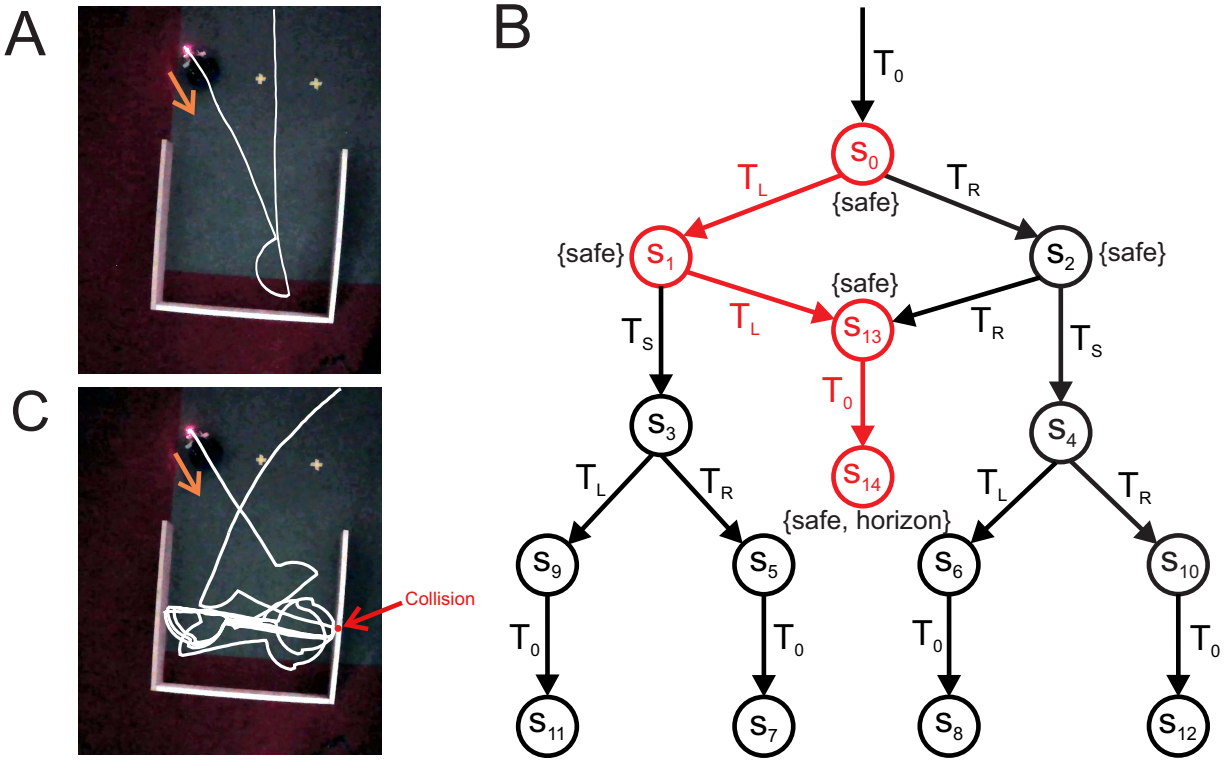

Figure 4.21: Comparison 2. A: trajectory for planning using model checking. B: the transition system and generated plan. C: trajectory of baseline method showing an eventual collision.

4.5.1 Summary

Our method was implemented on the low-powered robotic system described in Section 3.1 and combined discrete decision-making with continuous control (i.e., a hybrid system). The continuous control aspect was handled by closed-loop tasks, or motion primitives, which have an attentional focus on disturbances (i.e., obstacles which interrupt the resting state of the robot, the default task T_0). However, the tasks themselves have a finite lifetime and discrete outcomes, which we modelled using abstraction of timed aspects of tasks into static geometric relationships. To generate sequences of tasks, we implemented a bespoke lightweight model checker which meets the hard real-time deadline of 100 milliseconds. As mentioned in Section 4.1, we achieved this by integrating the model checking procedure with the robot code, as this was the most straightforward way to optimise model checking for real-time performance. No collisions were observed for planning using model checking and trajectories were more efficient.

We presented these results at the Formal Methods for Autonomous Systems (FMAS) workshop in November 2023 [34]. However, the evidence we acquired for our method was anecdotal, so the conclusions we could draw were limited owing to the design of experiments. The sample size was too small to make robust general claims about the reliability of planning using model checking for obstacle avoidance in comparison to the baseline, nor did we have robust insight into the reliability of real-time performance. In addition, we collected no data on memory usage so could not draw any conclusions about the memory footprint of our method on the robot.

4.6 Case Study I: Evasion

We now explain the design of a case study involving two scenarios to address shortcomings identified in the preliminary results. Specifically, we wanted to investigate whether planning using model checking reliably generated more efficient obstacle avoidance behaviours compared to the baseline method, as was indicated by results in the previous section. In addition, we wanted to gain greater insight into the reliability of real-time performance with respect to the deadline. We also wanted to investigate the efficiency of model checking in terms of memory usage. These specific objectives identified three additional research questions for our case study:

RQ2 *Does planning using model checking reliably generate more efficient collision avoidance behaviours compared to the baseline method?*

RQ3 *Does planning using model checking reliably meet the hard real-time deadline of 100 milliseconds for autonomous driving applications?*

RQ4 *What is the memory usage of planning using model checking?*

In Section 4.6.2, we conduct a close study of a cul-de-sac scenario, which often results in an agent becoming trapped when using reactive methods with a single control loop for obstacle avoidance. In Section 4.6.3, we investigate a playground scenario in which we let the robot roam free in a closed environment with one free-standing static object and a cul-de-sac. This scenario is intended to demonstrate the versatility and generalisability of planning using model checking. We discuss threats to validity in Section 4.6.4 and summarise our results in Section 4.6.5.

4.6.1 Artefacts

The baseline method is a simple Braitenberg-style machine [28] which can only spawn a single control task at a time, as described at a high level in Section 4.2. When in the default task T_0 and a disturbance D is encountered, the robot uses a simple deterministic rule to decide which avoid task to generate, either task T_L or task T_R. If the angle of the disturbance is greater than the heading 0 radians then the robot spawns the avoid task T_R, otherwise the task T_L is generated. Once the disturbance is counteracted, it then returns to the default task T_0. The baseline can therefore not reason about the consequences of its actions, which can lead to inefficient behaviour. As mentioned in Section 4.2, this can result in the robot getting trapped in a corner scenario. In this case study, we compare the baseline method to our planning approach described in Section 4.3.

4.6.2 Scenario 1 - Cul-de-sac

We addressed all research questions for the cul-de-sac scenario. To answer **RQ2** we conducted a statistical comparison of generated trajectories between planning using model checking and the baseline method and compared the number of collisions. We collected data on the real-time performance of model checking to answer **RQ3** and profiled memory usage to answer **RQ4**.

Environment Definition

We focused our attention on a cul-de-sac scenario due to the prevalence of corners and parallel walls, as reactive controllers for obstacle avoidance based on a single control loop can easily get trapped in such a situation. Our intention here was to recreate difficulties for the baseline method and see if planning using model checking showed any improvement in behaviour.

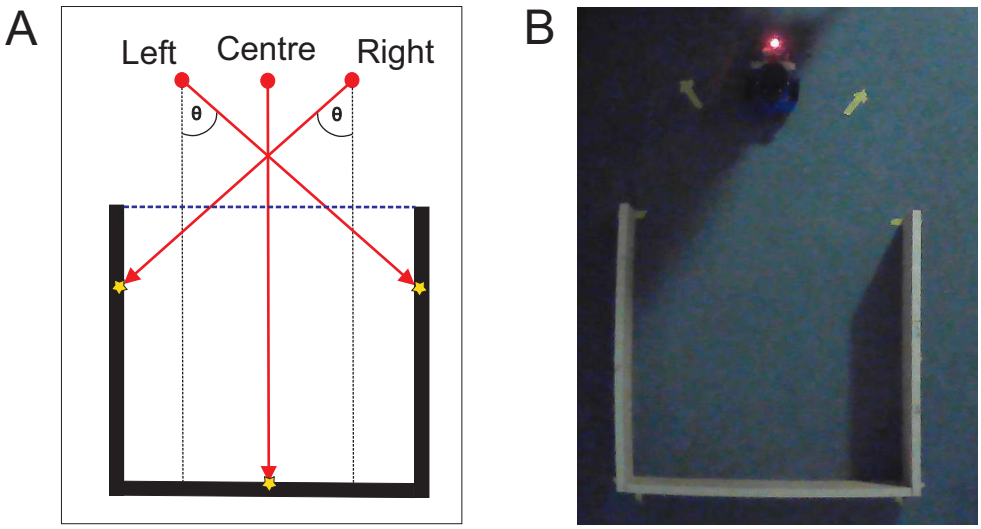

Figure 4.22: A: idealised schematic of cul-de-sac scenario. B: actual cul-de-sac scenario.

Figure 4.22A shows three starting positions (“Left”, “Centre”, “Right”) and the orientation of the robot relative to the bottom wall. For the centre position, the orientation of the x-axis is offset 0 degrees relative to the bottom wall, but for the left and right starting positions it is offset 45 degrees. There were two main reasons for this setup: (i) we wanted to maximise the chances of the baseline method leading the robot to get trapped, hence we used multiple starting positions and orientations; and (ii) while from a model checking perspective planning from the left or right position is symmetrical, pilot experiments revealed that the robot had a strong right veer when driving straight, so we decided to balance our study to accommodate. The dashed blue line in Figure 4.22A represents a cut-off used to prepare trajectories for analysis. We collected 15 runs for each method and starting position, resulting in a dataset consisting of 90 runs in total.

Analysis of RQ2

We focused on the construct of "reliability" to answer **RQ2** which is concerned with efficient obstacle avoidance behaviour. For the statistical analysis, we wanted to compare the amount of time the agent took to navigate out of the cul-de-sac. We operationalised this measure using trajectory length as a proxy for elapsed time, focusing only on the portion of the trajectory in the cul-de-sac (i.e., below the trajectory cut-off in Figure 4.22A). In addition, we also compared the number of collisions for each method. As collisions are rare events, there was a chance none would be observed, hence we focused on both metrics to ensure we had sufficient data.

Design of Analysis Our study had a 2×3 between-factors design for the statistical analysis of trajectory lengths and reporting of collisions. The method variable had two levels, "Single" and "Multi", representing the baseline and planning using model checking, respectively. The starting position variable had three levels ("Left", "Centre", "Right") as indicated in Figure 4.22A.

Data Collection and Preparation During data collection, we used markers to indicate the correct placement of the robot. Runs were terminated when the robot was observed to pass the threshold of the cul-de-sac by a visibly clear margin. We also recorded the number of collisions.

In practice, the variation in terminating runs was large, so it made sense to focus only on the portion of the trajectory in the cul-de-sac for the analysis. For each run, we extracted the trajectory from video using the optical flow method in OpenCV and calculated the length in pixels. Trajectories were truncated at the entrance to the cul-de-sac as shown in Figure 4.23B.

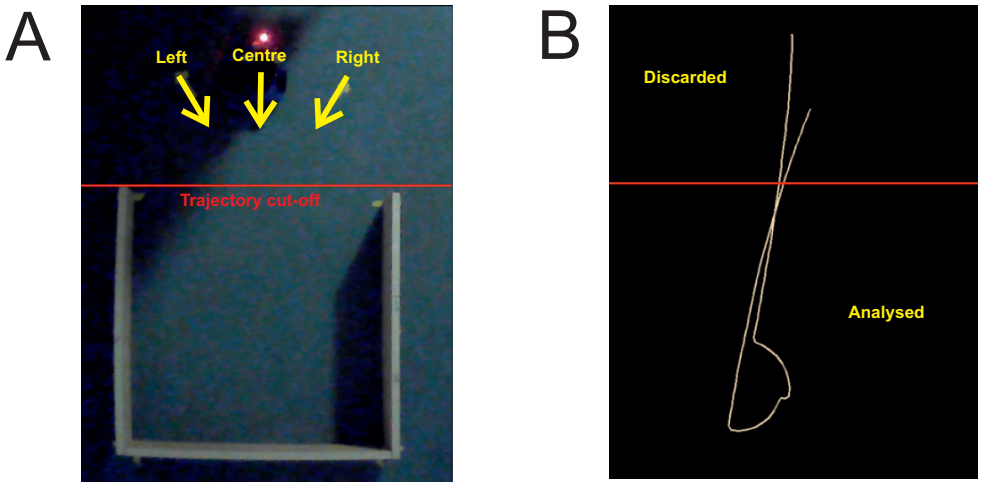

Figure 4.23: The cul-de-sac experimental setup. A: the three starting positions and placement of the trajectory cut-off. B: example trajectory and truncation according to the cut-off.

Results First we compared each method while ignoring starting position. A non-parametric Mann-Whitney U test was performed. As the shape and dispersion of the distributions was

different (see Figure 4.24 for boxplots), the null hypothesis was that the mean ranks for each method are identical, and the alternative hypothesis was that it is lower for the model checking group, $\alpha = .025$ (one-tailed). The test revealed that the mean rank for model checking was significantly lower, $U = 251$, $n_1/n_2 = 45$, $p < .001$, with a medium effect size, $r = .37$.

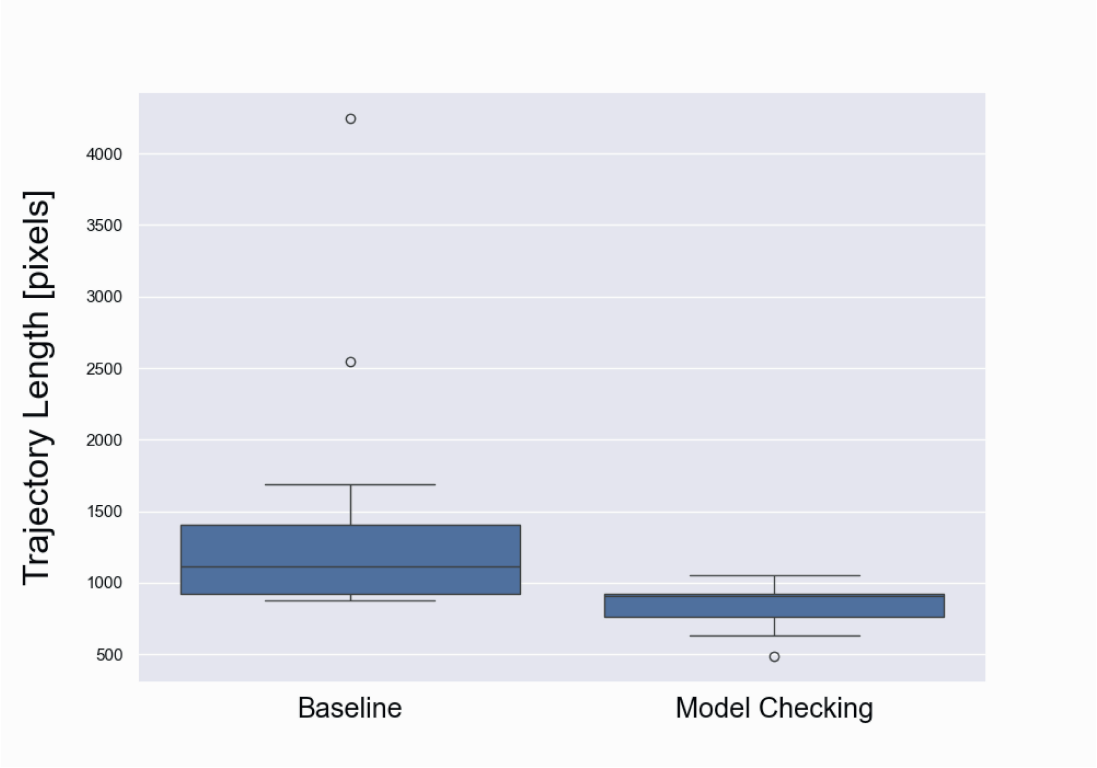

Figure 4.24: Trajectory length grouped by method.

Next a comparison was made between starting positions. It was reasoned that this could have an influence on the generated trajectory lengths due to differences in starting pose, specifically angle of entry into the cul-de-sac. We performed a non-parametric Kruskal-Wallis test on starting position and trajectory length for each method individually. For the baseline, the test revealed that there was a significant effect of starting position on trajectory length $H(2) = 12.17, p = .002$. Post-hoc analysis of pairwise comparisons using Dunn's test (Bonferroni correction applied) revealed that there was a significant difference between the centre starting position and both the left ($p = .026$) and right ($p = .002$) starting positions, while the left starting position was not significantly different from the right ($p = 1.$)[2]. The results suggest the single control task produces similar trajectory lengths when entering the cul-de-sac at an angle of ≈ 45 degrees. See Figure 4.25A for a comparison of the distributions for each starting position.

For the model checking approach, the test also revealed that there was significant effect of starting position $H(2) = 30.2, p < .001$. In this case, post-hoc analysis of pairwise comparisons revealed that there was a significant difference between the centre starting position and the right

[2]Rounded down from $> 1.$ as a secondary effect of the Bonferroni correction.

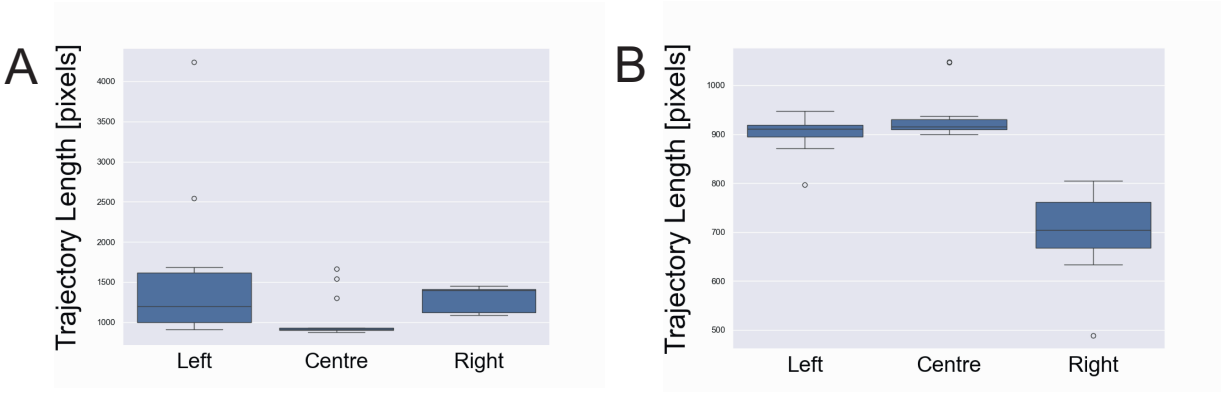

Figure 4.25: Trajectory length grouped by starting position. A: baseline, B: model checking.

($p < .001$) but not the left ($p = .872$), while the left starting position for this method was significantly different from the right ($p < .001$). The results suggest model checking generates different trajectory lengths when entering the cul-de-sac at an angle of ≈ 45 degrees from different starting positions, whereas the centre and left starting positions generate similar trajectory lengths. Figure 4.25B shows the distributions of trajectory lengths for each starting position. Trajectory lengths for the right starting position were significantly shorter because the robot had a right veer, so it would barely enter the cul-de-sac before generating a three step plan.

The number of collisions was also observed for each run. For the baseline method, 3 collisions were observed and 0 collisions were observed for planning using model checking. Each collision was observed for a different run of the baseline method (i.e., 3 runs in total). Table 4.6 shows collisions for the baseline grouped according to starting position.

Starting position	**Collisions**
Centre	1
Left	2
Right	0

Table 4.6: Distribution of collisions for baseline.

Analysis of RQ3

We again focused on the construct of "reliability" to answer **RQ3** which is concerned with meeting the hard real-time deadline of 100 milliseconds. We operationalised this by using processing latency as a metric, defined as the time delta from initiation of planning to completion. We aimed to describe the mean latency and estimate confidence bounds for planning using model checking for insight into whether our approach can consistently meet real-time constraints.

Design of Analysis We calculated the minimum, maximum and mean processing latency for all model checking runs and estimated 95% confidence intervals using the bootstrap method.

Data Collection and Preparation Processing latency was collected for each executed planning sequence at runtime using an in-built time and date package in C++.

Results The minimum processing latency for planning using model checking was 5.58 milliseconds and the maximum was 18.76 milliseconds. The mean processing latency for all runs was 10.1 milliseconds, with an estimated 95% confidence interval of $[9.33, 10.96]$.

Analysis of RQ4

We focused on profiling memory usage to answer **RQ4** in an exploratory manner. While memory usage contributes to the overall efficiency of our approach, we had no specific target to achieve, hence no construct was identified. We therefore focused on *describing* the memory footprint. Memory usage was estimated for both model checking and the entire process. We also estimated process memory usage for the baseline to indicate relative performance, however no direct comparison could be made due to uncertainty in the data.

Design of Analysis We estimated the runtime memory usage of both model checking and the entire program. Model checking memory usage was estimated by summing the compile time memory of the stack, set and adjacency list data structures used by f-DFS and their worst case usage at runtime, based on a state size of 4 bytes as each state is represented by an integer.

Data Collection and Preparation Process memory usage was collected using the Linux top utility at a sampling rate of 1 millisecond. For runtime memory usage of model checking, we instrumented the code to record the growth of the stack and set of visited states during f-DFS execution and dumped the data to disk.

Results Maximum memory usage for model checking at runtime was estimated at 2.9 kilobytes, while the maximum memory usage of the process was 4.31 megabytes. However, this includes the amount of shared memory available to a process, not all of which is typically resident to the program (i.e., includes memory which could potentially be shared with other processes). The maximum amount of shared memory was 3.34 megabytes, hence a lower bound on the maximum was calculated at 992 kilobytes. Maximum model checking usage was therefore estimated to comprise 0.06% to 0.29% of maximum process memory usage.

In comparison, maximum process memory usage for baseline was 4.3 megabytes with a maximum shared memory component of 3.28 megabytes. In this case, a lower bound on the maximum was calculated at 1.02 megabytes. While there is uncertainty, the data suggests that memory consumption for model checking is approximately equivalent to the baseline.

4.6.3 Scenario 2 - Playground

We again addressed all three research questions for the playground scenario. To answer **RQ2** we conducted two comparisons between the baseline and planning using model checking during general operation, focusing on evasion of a cul-de-sac aspect and collisions. We again collected data on real-time performance to answer **RQ3** and profiled memory usage to answer **RQ4**.

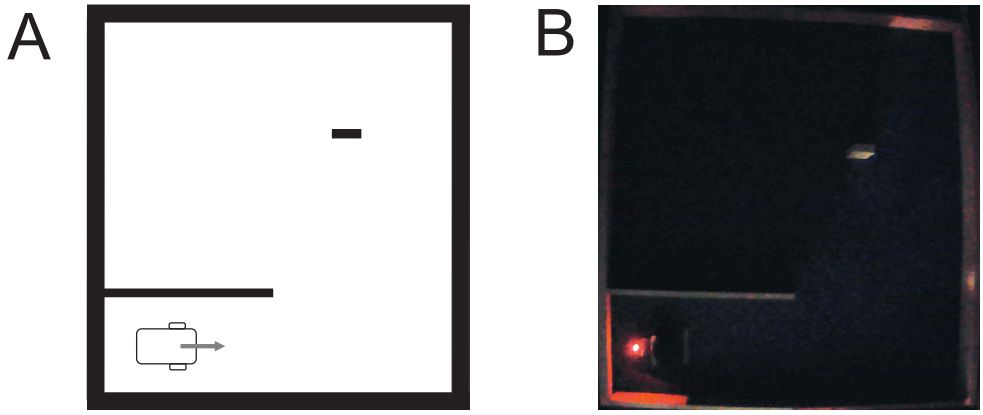

Figure 4.26: A: idealised schematic of playground scenario. B: actual playground.

Environment Definition

We let the robot roam free in a closed environment with one free-standing static object and a cul-de-sac, as illustrated in Figure 4.26. Our aim was to study the emergent behaviour of the robot during general operation where the robot has to counteract disturbances multiple times. We included a cul-de-sac because we were interested in comparing the emergent behaviour for each method when negotiating this situation in the context of normal operation. For each comparison (two in total), we let the robot roam free in the environment for 5 minutes using each method.

Analysis of RQ2

We again used the number of collisions as a measure of reliability to answer **RQ2**. In addition, we used time elapsed inside the cul-de-sac during exploration as a metric.

Design of Analysis For each run, we calculated the number of times the cul-de-sac was visited, the time elapsed inside the cul-de-sac and summed the number of collisions.

Data Collection and Preparation We made two comparisons between the baseline and planing using model checking. For each run, we let the robot explore the playground environment for 5 minutes and recorded the behaviour on video. We then extracted the trajectories using optical flow in OpenCV and inspected the difference visually. Following this, we calculated the number of times the robot visited the cul-de-sac and time elapsed inside the cul-de-sac, using the

light on the robot as a point of reference for consistency with analysis of the previous scenario. We also considered the cul-de-sac visited if the robot manoeuvred to evade. Following the same protocol of the previous scenario, we recorded the number of collisions for each run.

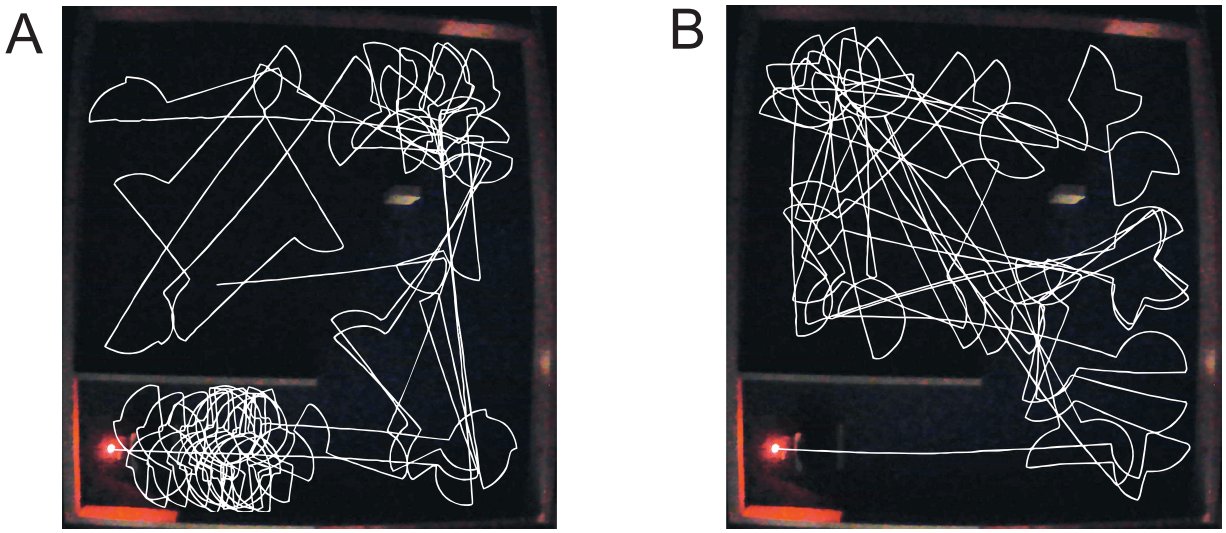

Figure 4.27: Comparison 1. A: baseline method. B: planning using model checking.

Results Figure 4.27 shows extracted trajectories for the first comparison. It can be seen in Figure 4.27A (baseline) that the robot spends significant time in the cul-de-sac, whereas in Figure 4.27B (planning using model checking) the robot spends no time in the cul-de-sac. The baseline method visited the cul-de-sac once during the run at which point it got trapped for 89 seconds. There were 4 collisions for the baseline and 0 collisions for model checking.

In Figure 4.28 trajectories for the second comparison are shown. Figure 4.27A shows that the baseline method spends significant time in the cul-de-sac. However, in this case, the robot visited the cul-de-sac on three separate occasions for a total elapsed time of 79 seconds. For this comparison, planning using model checking also visited the cul-de-sac on three separate occasions but evaded rather than enter the cul-de-sac. The total time elapsed for evasion was 12 seconds. Evasion manoeuvres are indicated by the yellow box in Figure 4.28B. On one occasion, the robot had to execute three consecutive two step plans before a successful evasion of the cul-de-sac, indicating the possibility that there may be scenarios where the robot can still get trapped. There were 2 collisions for the baseline and 0 collisions for model checking.

Analysis of RQ3

We again used processing latency as a metric for reliability to answer **RQ3**. We aimed to characterise the relationship between processing latency and plans of different lengths as it was found in the previous scenario that only two step plans were generated, introducing bias to the results.

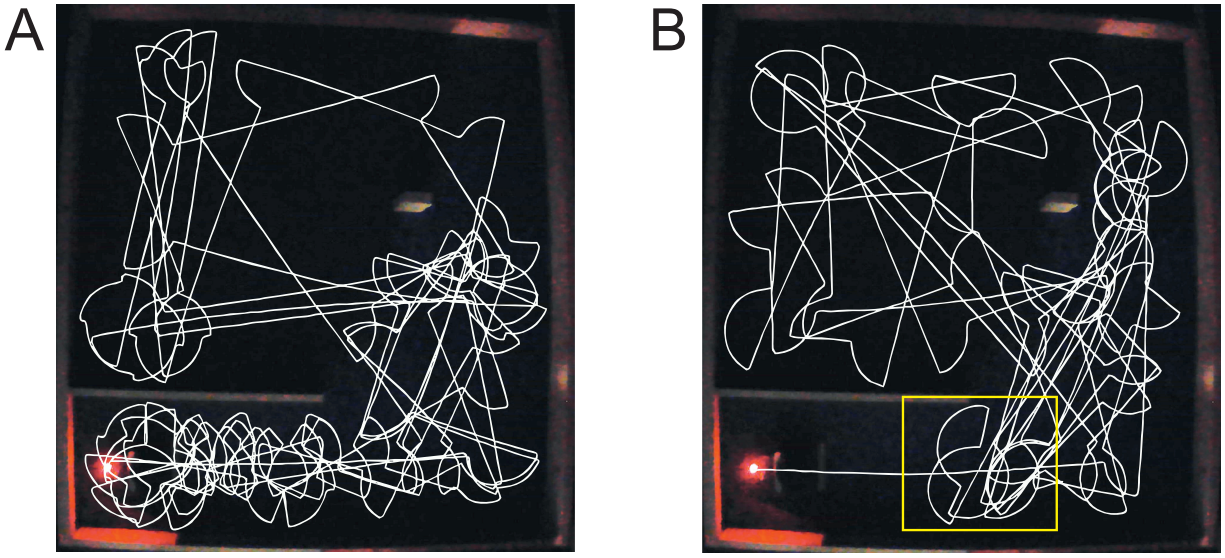

Figure 4.28: Comparison 2. A: baseline method. B: planning using model checking (yellow box indicates evasion manoeuvres at entrance to the cul-de-sac).

Design of Analysis We grouped processing latency by the number of steps in the plan and calculated descriptive statistics, estimating 95% confidence intervals for each plan length.

Data Collection and Preparation Data collection and preparation was the same as in the previous scenario (see Section 4.6.2). We also extracted the number of steps for each plan.

Results Table 4.7 shows results for processing latency across both model checking runs grouped according to the number of steps in plans. There were 42 two step plans, 32 three step plans and 18 four step plans. Processing latency appears to increase as the length of the plan increases.

Steps(n)	**Min.**	**Max.**	**Mean**	**CI (95%)**
2	2.5	14.09	6.64	[5.92, 7.44]
3	5.12	12.15	7.66	[7.11, 8.35]
4	5.45	21.61	11.49	[9.6, 13.83]

Table 4.7: Processing latency in milliseconds for plans consisting of 2, 3 and 4 steps.

Analysis of RQ4

We again profiled memory usage to answer **RQ4** in an exploratory manner. Memory usage was estimated for both model checking and the entire process. We again estimated process memory usage for the baseline method to provide an indicative comparison of relative resource usage.

Design of Analysis Same as analysis in the previous scenario.

Data Collection and Preparation Same as data collection in the previous scenario.

Results Maximum memory usage by model checking at runtime was estimated at 2.91 kilobytes and maximum memory usage of the process at 4.09 megabytes. Again this includes the amount of shared memory available to a process, which could potentially be shared with other processes. The maximum amount of shared memory was 3.03 megabytes, hence a lower bound on the maximum was calculated at 1088 kilobytes. Maximum model checking memory usage was estimated to comprise 0.06% to 0.26% of maximum process memory usage.

In comparison, maximum process memory usage for the baseline was 3.83 megabytes with a maximum shared memory component of 2.95 megabytes. A lower bound on the maximum was calculated at 900 kilobytes. As in the previous scenario, the data suggests that process memory consumption is approximately equivalent for each method.

4.6.4 Threats to Validity

The construct of "reliability" was motivated by the requirement for technical robustness and safety in the AI HLEG guidelines, specifically the sub-requirement for reliability and reproducibility (see Section 2.1.3). As a symbolic ante-hoc explainable AI approach for planning, we were interested in characterising (i) the continuity of decision-making using model checking for a range of inputs and situations, and (ii) whether model checking exhibits the same behaviour when repeated under the same conditions. Hence we focused on investigating the continuity of emergent behaviour for evasion of a cul-de-sac under a range of different starting conditions and included statistical analysis of generated trajectories for repeated trials. In particular, we focused on the real-time performance of our method with respect to a deadline relevant for the autonomous driving (see Section 2.2) and estimated confidence bounds on the population mean.

Our exploration of memory usage was not included under the construct of "reliability" as the data collection method (i.e., Linux top utility) did not support an accurate measure due to uncertainty. Hence our investigation of memory usage was exploratory. As mentioned in the analysis, some of the process memory is shared, meaning it could used by other processes which may vary between runs, so there was no way to know precisely how much memory was used by the process alone. Hence we estimated the memory usage of model checking and characterised this as a percentage of process memory using minimum and maximum bounds. In addition, there was no reliable way to truncate the generated data according to the time spent in the cul-de-sac as we did with trajectory lengths for Scenario 1. As there was a high degree of variability in stopping runs after exiting the cul-de-sac, the memory data generated was not comparable.

Another threat to internal validity comes from the trajectory analysis. Trajectory length was used as a proxy for elapsed time inside the cul-de-sac on the basis that time is directly proportional to distance, however this was ultimately dependent on the accuracy of the optical flow method in OpenCV. For example, we used the light on the robot to focus tracking on the bright-

est object in the frame, but in practice there was some variation. This was in part due to varying light conditions during data collection, which ultimately influenced the relative brightness of the environment. Sometimes two points were picked up. We addressed this in data preparation by visually inspecting each trajectory and manually preparing it for analysis. However, the trajectory analysis does not generalise to general operation, limiting external validity.

One threat to the external validity of our analysis of processing latency was that only three step plans were generated for Scenario 1, hence the results in the analysis were biased. This was a side effect of the cul-de-sac setup. We addressed this in Scenario 2 which provided sufficient data for all plan lengths for comparative analysis of estimated population means.

There are two major threats to the external validity of collisions as a measure of reliability. First we relied on human observation which is prone to error. The definition of a collision for this study was that a discrete sound had to be made by the robot colliding, at which point the collision was recorded for the run, however inaudible collisions may have been missed, especially if there was background noise during data collection. Second, as mentioned above, collisions are rare events, so not a reliable form of evidence alone for the construct, and provide little information on emergent behaviour. Indeed, a similar point was argued by Karla and Paddock [98] for the validation of autonomous driving systems (see Section 2.1.3 for a discussion).

4.6.5 Summary

In Scenario 1 we conducted a close study of a cul-de-sac. We compared trajectory lengths and collisions for a baseline and planning using model checking to answer **RQ2**. We found evidence that model checking reliably produces more efficient trajectories for avoiding a cul-de-sac. There were 3 collisions for the baseline method and 0 for model checking, however we had not data on emergent behaviour under general operation of the robot. To answer **RQ3**, we collected data on the processing latency. Processing latency was well within the 100 milliseconds real-time deadline for decision-making in autonomous driving, however we found that the nature of the cul-de-sac setup had side effects which restricted the generality of our conclusions—only three step plans were generated, so we had no data on the range of plan lengths. We profiled memory usage to answer **RQ4**, however there was significant uncertainty in the data.

In Scenario 2 we let the robot roam free in a playground environment. We compared time spent in the cul-de-sac and the number of collisions for a baseline method and planning using model checking to answer **RQ2**. In both comparisons, the baseline spent a significant amount of time in the cul-de-sac, whereas model checking spent none. There were 6 collisions for the baseline method and 0 for model checking. We collected data on processing latency and grouped it according to the plan length to answer **RQ3**. We observed an increase in mean processing latency associated with an increase in plan length, however the maximum latency was 21.61 milliseconds, again well within the 100 milliseconds deadline. To answer **RQ4** we profiled memory usage which was similar to the first scenario. Hence RQs have been answered.

Response to RQ2 *Our evidence suggests that planning using model checking reliably generates more efficient obstacle avoidance behaviour with respect to a cul-de-sac scenario compared to the baseline method. Planning using model checking tends to spend less time in the cul-de-sac and successfully avoids collisions.*

Response to RQ3 *Our data suggests that planning using model checking consistently meets the hard real-time deadline of 100 millisecond—regardless of the length of the plan, though it does have an impact. Our analysis suggests this holds for the population.*

Response to RQ4 *Our data suggests that maximum model checking memory usage is approximately 2.9 kilobytes and that maximum process memory usage ranges between* 992 *kilobytes and* 4.31 *megabytes. For comparison maximum process memory usage for the baseline method ranged between 900 kilobytes and 3.83 megabytes.*

4.7 Discussion

In this initial investigation, we have shown that it is possible to do real-time hybrid planning and control using model checking on a low-powered robot by chaining closed-loop tasks which have an attentional focus on disturbances. Abstraction of the temporal evolution of the hybrid system allows us to focus only on information relevant to collision avoidance, thereby keeping the model small which limits the amount of computation. We have presented evidence that compared to obstacle avoidance using a single control loop model checking is reliable with respect to real-time constraints and obstacle avoidance behaviour. Hence **RQ1** is answered.

Response to RQ1 *We can can use abstraction to model the evolution of closed-loop task sequences in terms of static geometric relationships relative to the egocentric perspective of the robot. It is then possible to reason about the predicted discrete outcomes of sequences using a single LTL property and generate plans for collision avoidance in real-time. Abstraction makes it possible to focus only on disturbances which keeps the model small. Thus issues with state-space explosion are avoided and resource usage minimised.*

Online methods using symbolic representations, such as our approach using model checking, can potentially offer a route to flexible general reasoning for partially observed, unstructured environments which is safe, reliable and explainable. However, real-time performance is crucial, particularly for high speed driving scenarios. As discussed in Section 2.2, it has been argued that even a lag of 100 milliseconds is unacceptable [123, 125]. While real-time obstacle avoidance for a robot in partially observed and unstructured environments was investigated

in [27], the approach can only reason about the current trajectory to ensure the robot is at rest for imminent collisions, hence no planning is involved. Real-time performance was reported at 49 milliseconds via simulation on an average laptop. In comparison, our approach achieves real-time planning with a processing latency less than 22 milliseconds on a low-powered robot.

Planning algorithms often assume that a robot moves in a global coordinate system in which trajectories are generated as continuous or piecewise continuous curves. In this context, model checking can be performed offline to synthesise a runtime monitor for the trajectory following error and signal fatal deviations [124], effectively performing classical line following. However, converting a local coordinate system into global coordinates from unreliable sensor data is non-trivial [119, 182] and hard to achieve in real-time without precise offline map data. In [157], for example, a cleaned and labelled global map is used to check generated trajectories for potential violations, and in [128] the locations of pedestrians in global coordinates are simulated. In contrast, we take a constructivist [160] approach and keep the environment partially observable by only working with sensory inputs that are accessible from an egocentric perspective, i.e., the relative position of disturbances. We do not need a global perspective as we do not perform precise line following along a globally defined trajectory. Instead, closed-loop tasks have an attentional focus on a specific disturbance which they counteract locally in a reactive manner.

Classical real-time obstacle avoidance methods, as discussed in Section 2.4, also tend to operate on local sensor data, however they do not involve discrete planning. Our approach uses LTL to specify the desired property of a sequence of discrete control tasks, demarcated spatially in the continuous environment by the location of disturbances (i.e., obstacles) using our abstraction. One advantage of this approach is the ability to express non-Markovian properties, such as temporally extended goals [122]. In comparison, traditional reactive obstacle avoidance methods (e.g., APF [106] and DWA [66]) rely on predicting the next action based on the current state, and as a consequence can often get trapped in local minima. Indeed, this was observed in our case study for the baseline. On our approach, however, plan specification in LTL made it possible to reason strategically about obstacles and generate complex multi-step plans. While our method could still get stuck in cyclic behaviour, this was not observed for U-shaped obstacles.

The use of model checking for hybrid planning and control has received increased attention in recent years, due to the ability to specify complex long horizon tasks using temporal logic and the potential to generate correct-by-construction controllers. For example, in [76] motion plans were revised using reactive synthesis for a mobile robot in real-time under specifications for a finite fragment of LTL in partially observable workspaces, however validation of the method was achieved through simulation and the real-time performance was not reported. In [114] iterative planning with partial guarantees for cosafe and safe components of LTL was proposed, and in [134] an abstraction-free sampling-based LTL algorithm based on RRT* was proposed which is probabilistically complete and asymptotically optimal. Again experimental validation was achieved through simulation. In comparison, we developed our approach on a real robot where

uncertainty in the system is unavoidable due to inaccurate sensing and actuation, a key issue in transferring idealised guarantees to the real world [200]. Consequently, our planning approach over-approximates the future path of the robot to guarantee that a set of possible trajectories satisfies our LTL specification and are therefore safe. We therefore relax the typical bisimulation property [60] requirement for guarantees in the continuous system to a simulation relation [13].

One advantage of hybrid planning using temporal logic specifications is the potential to provide causal explanations for robotic behaviour in a human interpretable form [102]. While we do not focus on generating explanations in this thesis, our use of LTL for plan specification and the characterisation of disturbances as the cause of a task ensures that this is possible. In [102] an algorithmic approach for generating factual and "why" explanations from LTL specifications over a deterministic Markov Decision Process (MDP) was proposed, and in [32] a method for generating counterfactual explanations using LTL constraints was developed. Although this is a non-trivial task, the interpretable nature of temporal logic specifications makes causal explanations at least feasible. In contrast, methods involving opaque models, such as DNNs, are less amenable, which undermines the trustworthiness of AI-enabled systems. As mentioned in Section 2.1, transparency has been emphasised in guidelines from AI regulators [1, 188], especially for the governance of AI in safety-critical domains such as autonomous driving [112, 149].

An obvious limitation of our approach in this chapter is that obstacle avoidance is not target-driven. However, any realistic task involving autonomous navigation, such as overtaking for AVs, will require target-driven behaviour. To achieve this, some form of localisation in the environment is required, usually through SLAM [164] which is computationally expensive—performance of the localisation operation alone can impose a time lag in the order of hundreds of milliseconds [26, 120]. Localisation is important for a complete robotic system, however our focus in this chapter was restricted to flexible decision making. Hence we have omitted localisation from our initial investigation into the use of model checking for real-time obstacle avoidance. We incorporate this in the next chapter to accommodate target-driven behaviour.

Chapter 5

Real-Time Goal-Directed Obstacle Avoidance

In this chapter, we describe our investigation into extending the approach in Chapter 4 to support goal-directed obstacle avoidance. In Section 5.1, we motivate the combined use of abstraction and recursive forward simulation to plan/replan sequences of closed-loop tasks using real-time model checking. We provide a high level conceptual overview of our approach in Section 5.2. In Section 5.3, we explain the details of our methodology. Specifically, we explain how we use a layered abstraction approach to characterise "situations" and simulate relevant short sequences of tasks. We introduce two interrelated transitions systems for this purpose. In Section 5.4, we provide details on the implementation of our method. We explain our case study in Section 5.5. In Section 5.6, we draw conclusions on the merits and limitations of our approach. See Appendix B for a glossary of variable names introduced in this chapter.

5.1 Motivation

In Chapter 4 we showed that it is possible to plan sequences of closed-loop tasks in real-time on a low-powered device using model checking. However, as mentioned in the conclusion, the main limitation was that the approach was not goal-directed. Rather the focus was on reasoning about multiple disturbances to facilitate *situational analysis*, the major benefit of which is the ability to overcome common issues with closed-loop input control, in particular avoiding trap situations. Hence we focused on a cul-de-sac scenario due to the prevalence of corners which often lead to stereotyped behaviour from closed-loop controllers.

In this subsequent investigation into real-time model checking for planning and obstacle avoidance, we extend our approach and focus on avoiding objects while making progress towards a goal, i.e., the complete obstacle avoidance problem. This requires the strategic combination of both avoidance and attraction closed-loop behaviours to generate effective goal-directed plans. As the emergent behaviour of closed-loop tasks varies, it is therefore necessary to rethink

the approach to modelling the physical environment (which is tightly coupled to the definition of tasks) to ensure a relevant interpretation of distal sensor information for decision-making. Consequently, the bespoke implementation of model checking requires adjustment to ensure a relevant distinction between closed-loop tasks which prioritise avoidance or attraction.

While our approach in Chapter 4 was capable of operating in partially observed environments, the restricted nature of the avoidance task made this somewhat irrelevant. As the only aim was to not hit objects, information about the dimensionality of objects was unnecessary to plan sequences of closed-loop tasks. For goal-directed behaviour, however, it becomes more of an issue. For example, if the goal is on the other side of an object, the 2D point-cloud will only perceive the nearest edge, so the length of the object is unknown. However, to plan a sequence of tasks to reach a goal, prior knowledge of object dimensions is necessary, or alternatively the method should be able to replan as new information about the environment becomes available.

Previously we developed an egocentric model of the environment which abstracted away timed aspects of task execution into static geometric relationships in Chapter 4. This made it possible to predict the outcomes of tasks based on the location of disturbances in the surrounding environment. However, the additional requirement to get to a goal (which may be obscured from view) makes a similar approach infeasible. Static geometric relationships, while a relevant proxy for the evolution of timed aspects in the hybrid system, are too inflexible and complex to accommodate goal-directed obstacle avoidance. A more promising approach is to combine a model of spatial relationships with recursive forward simulation for flexible replanning.

We therefore investigate the use of bespoke model checking for planning and obstacle avoidance which can (i) combine closed-loop avoidance and attraction behaviours to form goal-directed sequences of tasks, and (ii) replan based on new information about the environment. We aim to do this in the context of hybrid planning and control. To simplify the problem, we continue to restrict attention to static environments and focus on a scenario similar to overtaking but without the complication of oncoming vehicles. We aim to do this in real-time, specifically within the real-time deadline of 100 milliseconds. Hence we address the research question:

RQ5 *How can we extend our bespoke implementation of model checking to plan sequences of avoidance and attraction closed-loop tasks for goal-directed obstacle avoidance on a low-powered robotic system in real-time?*

5.2 Approach

In this section, we first extend the baseline in Chapter 4 to describe a minimal solution which has the ability reach a goal (i.e., it can spawn avoidance or attraction tasks), but as before it can only reason about a single disturbance at a time. However, it does have the ability to forward plan using simulation and replan if necessary. This involves simulating execution of the straight

task T_S (driving straight for some distance d) or rotating left or right, i.e., simulating task T_L or task T_R. Rotations are defined the same as in Chapter 4. In addition to avoiding objects (i.e., disturbances represented by D), the robot also needs to eliminate error to the goal, represented as the disturbance $D^G = [x, y]$ where x is the longitudinal error and y is the lateral error in meters.

Figure 5.1 shows a static obstacle avoidance scenario similar to an overtaking task for AVs. Initially, the goal $D^G = [1, 0]$ is straight ahead. The agent checks if there is a disturbance D ahead for the current state of the environment and if the way is clear subtracts distance d from the x-coordinates of observations in O to simulate execution of the task T_S. This process is repeated until the disturbance D_1 is encountered for some future predicted state of the environment.

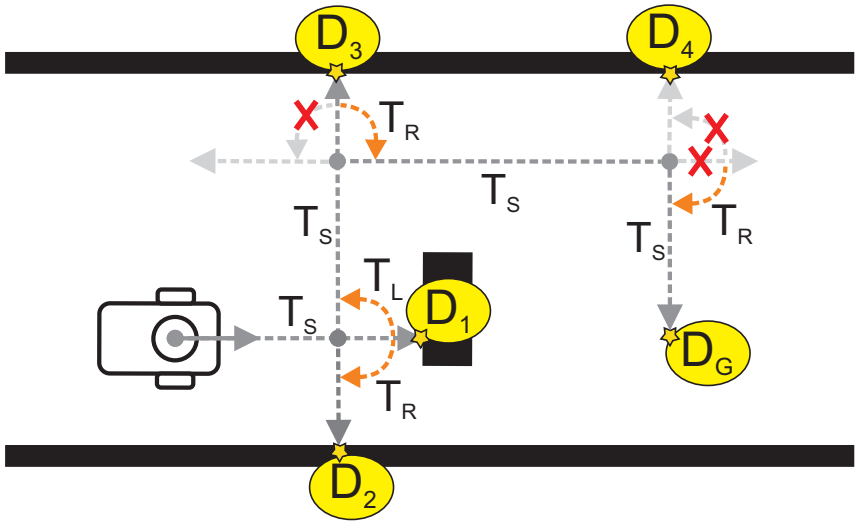

Figure 5.1: Minimal extension to baseline in Chapter 4 able to complete an overtake manoeuvrer. The default task T_0 continues executing until a disturbance is encountered or error to the goal D^G in some dimension is eliminated. Relative direction of the goal determines valid rotations.

The robot can then simulate task sequence $T_L \rightarrow T_S$ or $T_R \rightarrow T_S$ to counteract the disturbance by rotating the set of observations clockwise or anti-clockwise, respectively, then simulating the straight task. Note that either sequence is valid because the goal is straight ahead. If the goal was to the right or left of the robot, then rotating away from the goal would not be allowed as a measure to promote the progressive elimination of D^G. For this reason, we include the final straight task to prevent the robot from immediately doubling back on itself. When the goal is straight ahead, rotating left or right would put the goal on the right or left, respectively, making it possible for an infinite loop to occur. Hence the planning procedure would not terminate.

From the perspective of an external observer, it is obvious that simulating $T_R \rightarrow T_S$ will result in encountering a second disturbance D_2. However, the robot can only perceive a finite portion of the environment directly ahead, so this is unknown from the egocentric perspective of the robot. We do not allow the longitudinal error to be increased to ensure progress towards the goal, so the only valid sequence in response to D_2 is $T_L \rightarrow T_S$. It should be clear that the robot could get trapped here eliminating D_1 and D_2 repeatedly. Hence there is still the potential for

the planning procedure to get stuck in an infinite loop, which is undesirable. This is a limitation of only reasoning about a single disturbance at a time, however this is not inevitable.

An infinite loop can be avoided if sequence $T_L \rightarrow T_S$ is eventually simulated to counteract disturbance D_1. The robot then repeatedly simulates the task T_S until the disturbance D_3 is encountered, which necessarily increases lateral error to the goal. As the goal is to the right, the only valid sequence is $T_R \rightarrow T_S$. Again the robot repeatedly simulates task T_S, this time until the longitudinal error to the goal is eliminated $D^G = [0, y]$. The goal is again to the right, so the only valid sequence is $T_R \rightarrow T_S$. The robot then simulates the repeated execution of T_S until the lateral error is eliminated. Hence $D^G = [0, 0]$ meaning that the robot has now arrived at the goal.

The approach in Figure 5.1 represents a simple obstacle avoidance approach similar to the baseline described in Chapter 4. As explained, however, this does not prevent the simulation of tasks from getting stuck in trap situations when first encountering the obstacle to be overtaken, which could potentially result in an infinite loop. In addition, it should be clear from Figure 5.1 that only spawning tasks in response to disturbances or the elimination of error does not generate efficient trajectories, nor does it result in the correct orientation of the robot when it reaches the goal. Furthermore, this representation of the approach is a best-case idealisation. In practice, error in the real execution can lead to encountering unexpected disturbances, which requires replanning. This can also lead to random trajectories which fail to get to the goal (e.g., because the robot never rotates exactly $\pm\frac{1}{2}\pi$ radians) so the approach is unreliable.

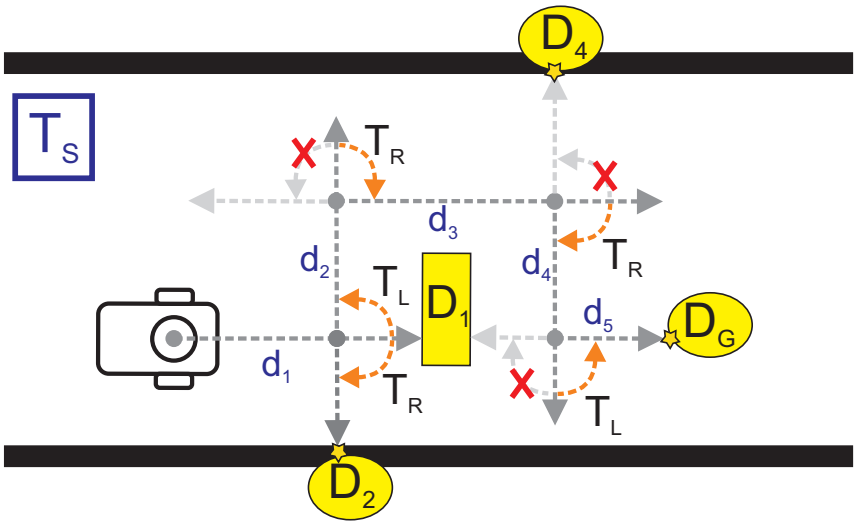

Figure 5.2: General overview of approach. The straight task T_S executes in a closed-loop fashion until the control goal is achieved. This is a distance d determined by either the dimension of the object we want to overtake or relative error to the goal D^G to minimally deform the path of the robot. As with the minimal solution, relative direction of the goal determines valid rotations.

Our method therefore aims to reliably generate more efficient trajectories by reasoning about distances for straight tasks to minimally deform the path of the robot in relation to the obstacle. Figure 5.2 provides an overview of our general approach. In contrast to the minimal solution

above, the straight task T_S constitutes a negative feedback loop with the control goal of driving straight for a finite distance, determined either by the dimensions of the object we want to overtake, or relative error to the goal D^G. To do this and avoid trap situations requires distal sensor information and the ability to reason about multiple disturbances concurrently.

To eliminate all error to the goal D^G, it is necessary to forward plan, which can be achieved using recursive simulation of task execution. Figure 5.3 illustrates an example plan for the overtake situation illustrated in Figure 5.2. As the environment is partially observable, without an accurate offline map it is not possible to plan an accurate path to the goal from a local frame of reference because the dimensionality of the object we want to overtake is unknown. Hence we further require that our approach has the ability to replan the sequence of tasks if it transpires that the appropriate task for a situation deviates from the expected next task in the planned sequence.

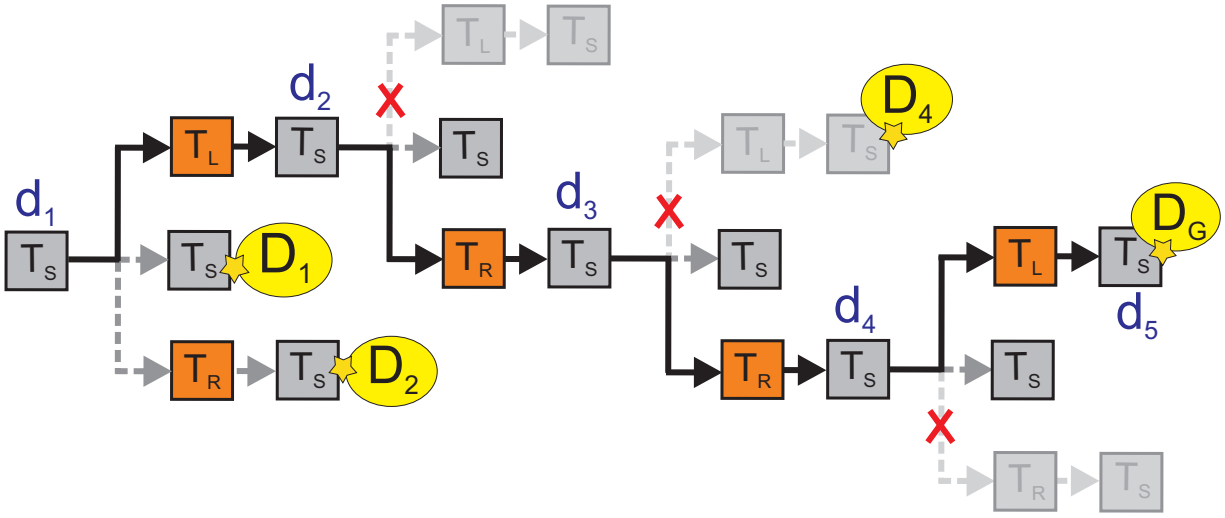

Figure 5.3: Example plan for the overtake situation depicted in Figure 5.1 which eliminates all error to the goal D^G prior to execution using simulation of task execution to forward plan.

5.3 Methodology

In this section, we provide the details of our methodology. In Section 5.3.1, we explain the action space, which again comprises a set of closed-loop control tasks. Specifically, we give details on how the control goal for a closed-loop task reflects a different attentional focus which respects the two dimensional geometry of objects. Section 5.3.2 describes the discrete outcomes for tasks. In Section 5.3.3, we explain how we interpret the relative direction of the goal in a discrete manner, based on continuous tracking data. Section 5.3.4 describes an extension to the goal abstraction for reasoning about the relevant short sequence of tasks for a modelled situation. We provide the details of two transition systems used in our approach in Sections 5.3.5 and 5.3.6. In Section 5.3.7, we provide details on recursive model checking for goal-directed plans.

5.3.1 Action Space

We formally define the action space for our approach as the set of closed-loop tasks:

$$Act = \{T_S, T_L, T_R\} \tag{5.1}$$

introduced in Section 5.2. In contrast with the approach developed Chapter 4, we have one straight task T_S which has a finite lifetime and is immediately destroyed when the control goal of eliminating a disturbance (in this case, the continuous distance d) is achieved (see Figure 5.4A for an illustration). The tasks T_L and T_R are similar to the approach in Chapter 4, however we rotate $\pm\frac{1}{2}\pi$ radians to eliminate error between the current and desired orientation, i.e., the required orientation for execution of the subsequent straight task (illustrated in Figure 5.4B).

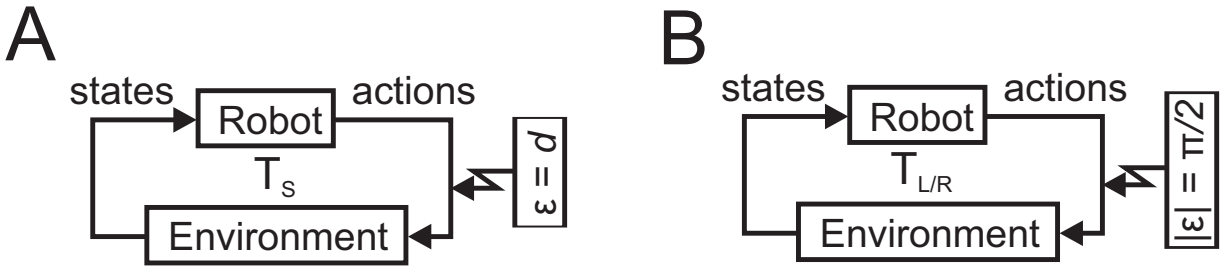

Figure 5.4: A: closed-loop task T_S, where the control goal is eliminating distance d. B: closed-loop tasks $T_{L/R}$, control goal is eliminating relative error to desired orientation which is $\pm\frac{1}{2}\pi$.

As we assume that the object to be avoided has dimensionality in the plane, rotating away from a disturbance is insufficient for elimination. Instead, this requires counteracting both the longitudinal and lateral dimension of an obstacle. Hence we refer to the task $T_{L/R}$ as a "rotate" instead of an "avoid" task. Whether a task is classified as avoidance/attraction is decided by the model checking procedure at runtime, instead of designated a priori as in Chapter 4. In this chapter, disturbance elimination is a spatially extended notion and requires *sequences* of tasks.

5.3.2 Discrete Task Outcomes

Consequently, discrete task outcomes have a different semantics for goal-directed obstacle avoidance. As in Chapter 4, the discrete outcomes of tasks provide information on the success or failure of the real execution, which grounds our approach in information about the continuous evolution of the hybrid system. In the remainder of this section, we explain the semantics of discrete success and failure criteria for the set of control tasks defined in the previous section.

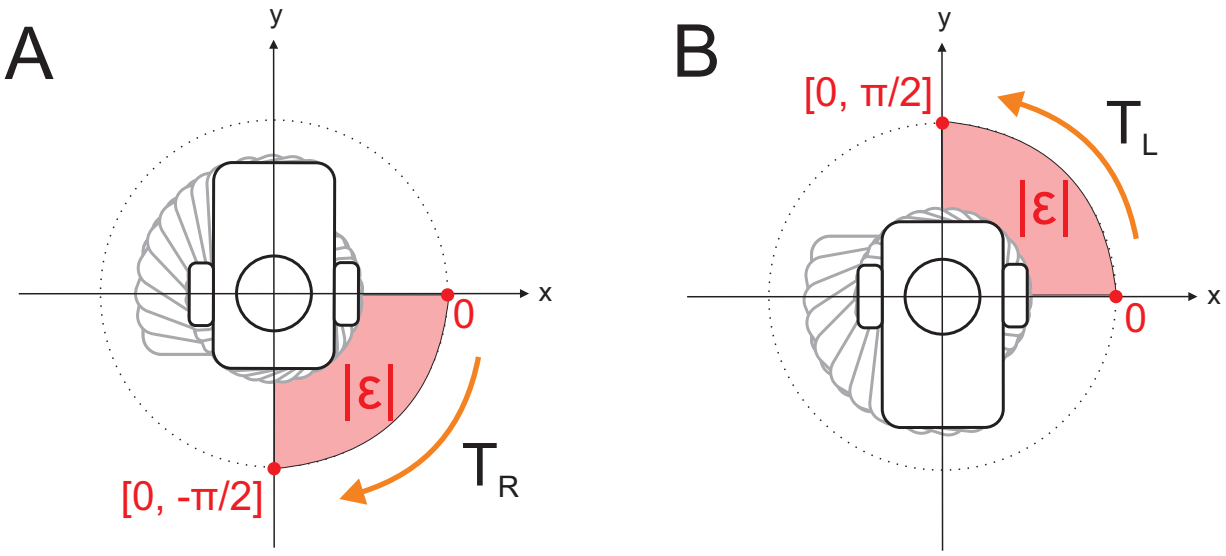

Figure 5.5: A: task T_R eliminating error $|\varepsilon|$. B: task T_L eliminating error $|\varepsilon|$.

Rotate Tasks $T_{L/R}$

Similar to the approach in Chapter 4, we define a safe zone around the robot to ensure rotations are safe (see Figure 5.5). We define success for a rotate task $T_{L/R}$ as rotating until the absolute difference between the current and initial orientation of the robot θ is *not* in the partition:

$$P_{rotate} = \{\theta \mid |\theta^{curr} - 0| < \frac{1}{2}\pi\} \tag{5.2}$$

where θ^{curr} is the current orientation of the robot and 0 is a constant. As shown in Figure 5.5, the initial orientation of the robot is always 0 radians and the desired orientation for our approach is always $\pm\frac{1}{2}\pi$ radians. Table 5.1 summarises the possible outcomes for the rotate partition.

Success	**Failure**
$P_{rotate} = \emptyset$	$P_{rotate} \neq \emptyset$

Table 5.1: Possible outcomes for rotate partition P_{rotate}.

Hence success and failure can be expressed as satisfaction of the first order existential condition:

$$P_{rotate} \models \exists\theta(|\theta^{curr} - 0| < \frac{1}{2}\pi) \tag{5.3}$$

From the perspective of an external observer, rotate tasks look the same as the avoid tasks described in Chapter 4. However, as shown in Figure 5.5, rotate tasks have a different attentional focus, i.e., eliminating $|\varepsilon|$ until $\theta^{curr} \geq \frac{1}{2}\pi$. The focus is on error elimination because disturbance elimination is a higher order concept requiring sequences of tasks. The sole purpose of a rotate task is to ensure that the robot is always in the correct orientation to counteract a spatially extended disturbance. This could be an object (i.e, a disturbance D) or error to the goal D^G.

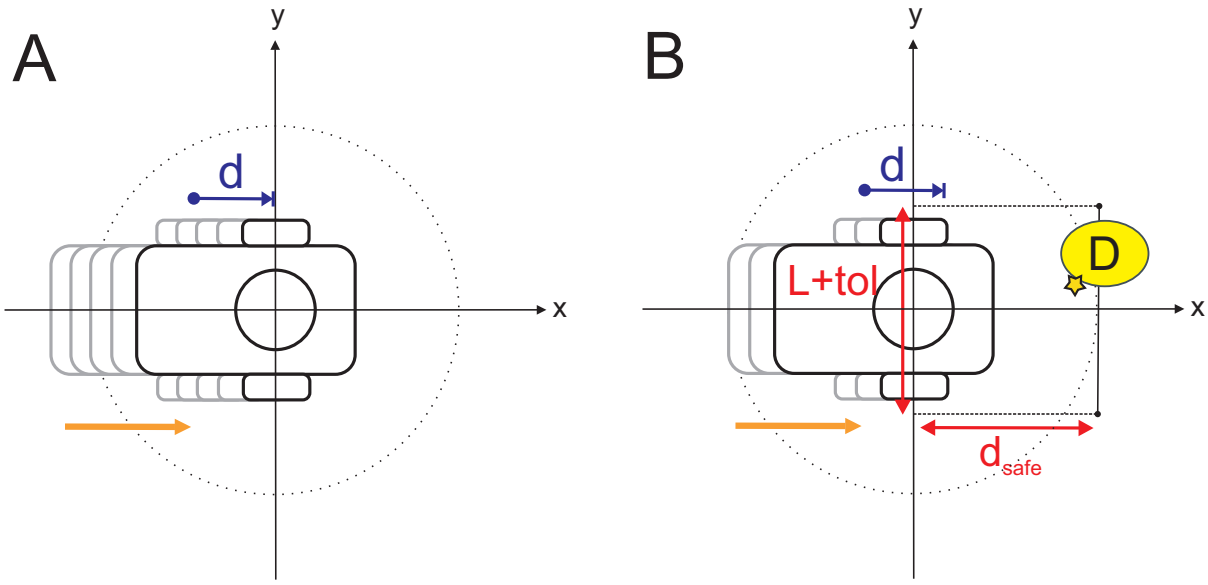

Figure 5.6: A: executing the task T_S to achieve the control goal of reaching distance d. B: witnessing disturbance D during T_S before achieving the control goal of reaching distance d.

Straight Task T_S

We next define success and failure criteria for the straight task T_S. Success is achieving the control goal of driving straight until a distance d^{target} has been eliminated (see Figure 5.6A) and failure is witnessing a proximal disturbance in the direction of travel during execution, meaning that achieving the control goal is impossible (see Figure 5.6B). This means that an additional outcome "running" is necessary to indicate that the control goal has not yet been achieved.

We define the control goal in terms of the following partition:

$$P_{dist} = \{d \mid d^{curr} < d^{target}\} \tag{5.4}$$

where d^{curr} is the current distance travelled by the robot and d^{target} is the target distance, determined by our model during the planning procedure at runtime. In addition, we define the witnessing of a proximal disturbance in terms of the partition, shown in Figure 5.6B:

$$P_{safe} = \{o \in O \mid 0 < o_x \leq d_{safe} \wedge |o_y| \leq \frac{1}{2}L + tol\} \tag{5.5}$$

where O is the set of observations for a scan event, d_{safe} is the radius of the safe zone and outer shield defined in Chapter 4, and $L + tol$ is the width of the robot plus some tolerance. Table 5.2 summarises possible outcomes for the straight task T_S in terms of a union of these partitions.

Running	Success	Failure
$P_{dist} \neq \emptyset \wedge P_{safe} = \emptyset$	$P_{dist} = \emptyset \wedge P_{safe} = \emptyset$	$P_{dist} \neq \emptyset \wedge P_{safe} \neq \emptyset$

Table 5.2: Possible outcomes for the union of partitions $P_{dist} \cup P_{safe}$.

Success or failure of the straight task T_S can therefore be expressed as the existential condition:

$$P_{dist} \cup P_{safe} \models \exists d^{curr} \exists D(d^{curr} < d^{target} \wedge (0 < D_x \leq d_{safe} \wedge |D_y| \leq \frac{1}{2}(L + tol))) \tag{5.6}$$

5.3.3 Tracking the Goal

To successfully plan/replan a sequence of tasks to reach a spatial goal, it is necessary for the robot to have up-to-date information on its coordinates. During the real execution, we update the relative coordinates of the goal D^G in a closed-loop fashion using dead reckoning, however this method is notoriously imprecise due to accumulated error in the system. In addition, the resulting data is continuous, hence it is necessary to find some way to interpret the information which supports discrete decision-making. We therefore construct an over-approximated abstraction which allows us to tolerate some degree of error in the continuous tracking and determine the relative direction of the goal D^G in a discrete manner. This is used to interpret the relative direction of D^G when initiating planning, but also during recursive simulation of task execution.

Our goal abstraction is informed by the restricted set of tasks defined in Section 5.3.1. As indicated in Section 5.2, from the self-referential perspective of the robot, only information about whether the goal is straight ahead, or generally in the left/right direction, is required to navigate to the goal (e.g., to support reasoning about relevant sequences of tasks). To achieve this, we construct egocentric, mutually exclusive partitions which witness the goal $D^G = [x, y]$.

Goal is Straight Ahead

We first define a partition for determining if the goal D^G is straight ahead, as illustrated in Figure 5.7. The partition is indicated by the grey area and has an infinite extension to the right. It is straightforward to decide if the partition witnesses the goal, which we define formally as:

$$P^G_{straight} = \{x, y \mid x > d_{safe} \wedge |y| \leq \lambda d_{safe}\} \tag{5.7}$$

where x and y are the relative coordinates of the goal, d_{safe} is a parameter defining a lower bound on the x-coordinate, consisting of the combined safe zone and outer shield, and λ is a coefficient for tuning the lateral dimension of the partition. There is no upper bound on the x-coordinate, as this would imply an upper bound on the plan length, making our approach too restrictive.

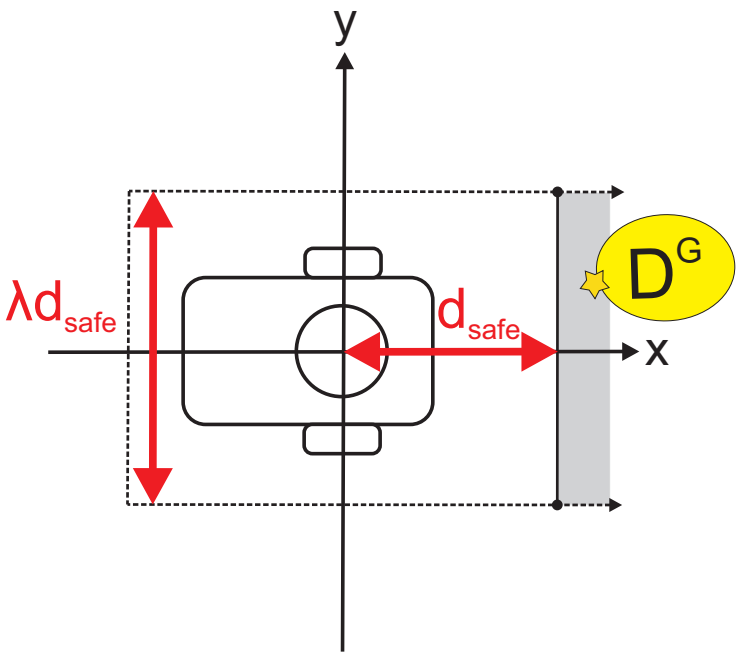

Figure 5.7: Partition for determining if the goal is straight ahead.

Goal is on the Left/Right

We next define partitions for determining if the goal is on the left or right, illustrated by Figures 5.8A and 5.8B, respectively. Again partitions are indicated by the shaded area in grey and are unbounded in the direction moving away from the robot. We define the left partition as:

$$P^G_{left} = \{x, y \mid (x < -d_{safe} \wedge y > 0) \vee y > \lambda d_{safe}\} \tag{5.8}$$

where $-d_{safe}$ is an upper bound on the x-coordinate of the goal to exclude the area immediately

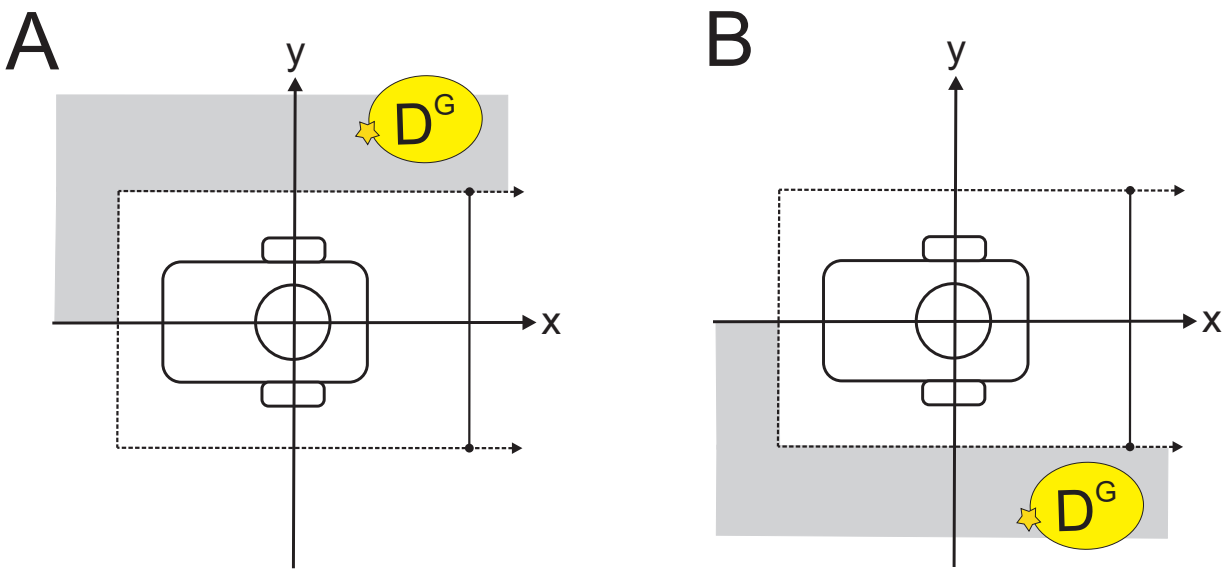

Figure 5.8: A: partition for left direction. B: partition for right direction.

behind the robot, 0 is a lower bound on the y-coordinate to indicate the left direction where the x-coordinate is less than $-d_{safe}$, and λd_{safe} is a lower bound on the y-coordinate where the x-coordinate is greater than $-d_{safe}$. Otherwise there is no bound on the coordinates in D^G.

We define the right partition for witnessing the goal as:

$$P^G_{right} = \{x, y \mid (x < -d_{safe} \wedge y < 0) \vee y < -(\lambda d_{safe})\} \tag{5.9}$$

The same interpretation of parameters holds for this partition as for P^G_{left} by a symmetrical argument. As should be clear, they are the negative reflection of those parameters.

For the partitions in Figure 5.8 we include the area immediately behind the robot to ensure appropriate decisions about the relative direction of the goal can be made if it is first of all necessary to increase the lateral error when navigating around an object, as illustrated in Figure 5.9. Note that from the egocentric perspective of the robot, the longitudinal error has been increased, hence the goal D^G is now behind the robot and on the right. By a symmetrical argument, the same would apply if the robot had instead rotated right to avoid the disturbance D. However, from the perspective of the robot the goal D^G would have then been on the left. The procedure for determining the relative direction of the goal is outlined in Algorithm 4. Note that it returns an initial state in our model which will be explained in later Section 5.3.6.

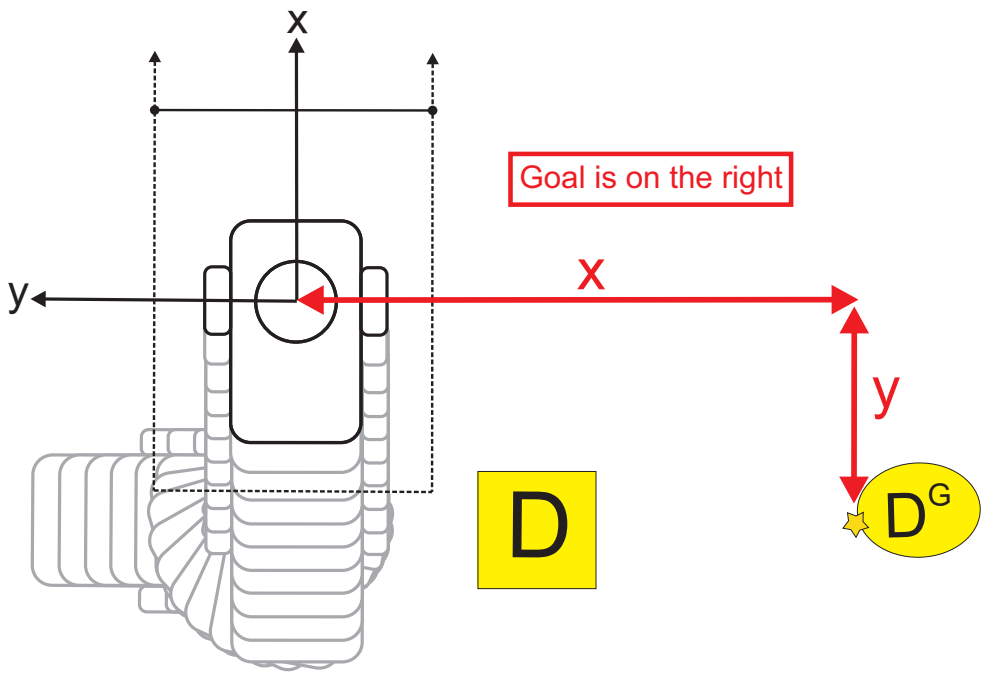

Figure 5.9: Increasing the lateral error to navigate around an object, the disturbance D. Red arrows indicate global perspective of an external observer. From the perspective of the robot, the longitudinal error has been increased and the goal D^G is on the right using our abstraction.

Algorithm 4: Determine Goal Direction

Input : Goal coordinates $D^G = [x, y]$, parameter d_{safe} and coefficient λ.
Output: Relevant initial state in model.

```
1 Procedure GoalDirection(D^G, d_safe, λ)
2     if D^G_x > d_safe ∧ |D^G_y| ≤ λ d_safe then
3         return s_5
4     else if D^G > λ d_safe ∨ (D^G_x < −d_safe ∧ D^G_y > 0) then
5         return s_3
6     else if D^G_y < −(λ d_safe) ∨ (D^G_x < −d_safe ∧ D^G_y < 0) then
7         return s_4
```

5.3.4 Abstract Situational Analysis

The abstraction defined in the previous section provides discrete information on the general location of the goal D^G. However, the environment may also contain obstacles (i.e., disturbances) which need to be avoided for safe navigation. The robot therefore requires the ability to interpret distal sensor information and predict whether sequences of closed-loop tasks will be successful.

Consequently, it is necessary to extend our abstraction to interpret and reason about disturbances. The extension defined in this section can be viewed as a separate layer which facilitates reasoning about obstacle avoidance when combined with the goal abstraction. In other words, the combination makes it possible to reason about sequences of tasks in response to *situations*.

A situation is formally defined as a tuple $\mathscr{S}_i = \langle O_i, D_i^G \rangle$ where O is the set of observations, D^G is error to the goal, and i is an arbitrary index indicating the order of the situation in a sequence. As will be explained in Section 5.3.7, a *plan* on our approach is ultimately the concatenation of short sequences of tasks based on real-time simulation of task execution which recursively generates a sequence of situations. *Situational analysis* is reasoning about the predicted outcomes of task sequences represented by our abstraction for a specific situation $\mathscr{S}_i$. This determines relevant inputs for the model checking procedure described in the next section.

Drive Straight

We first define a partition to predict if the robot can successfully execute the straight task T_S and drive distance d_{target} for a situation $\mathscr{S}_i$. Figure 5.10A shows construction of the partition (indicated by the grey area) in relation to the goal abstraction defined in the previous section. For a situation $\mathscr{S}_i$, we iterate the set of observations O_i and sort them into the partition:

$$P_{straight} = \{o \in O_i \mid d_{safe} < o_x \leq \alpha d_{safe} \wedge |o_y| \leq \frac{1}{2}(L + tol)\} \tag{5.10}$$

where d_{safe} is a lower bound on the x-coordinate of observations, αd_{safe} is an upper bound, and $\frac{1}{2}(L + tol)$ is a bound on the y-coordinate of an observation representing the width of the robot

plus some tolerance. The distance associated with task T_S is defined as $d^{target} = \alpha d_{safe} - d_{safe}$.

Table 5.3 summarises the possible future outcomes for the straight task T_S, which is determined by whether an observation (i.e., a disturbance D) is witnessed in the partition $P_{straight}$.

Success	**Failure**
$P_{straight} = \emptyset$	$P_{straight} \neq \emptyset$

Table 5.3: Possible outcomes for straight $P_{straight}$.

Success and failure can therefore be expressed in terms of the first order existential condition:

$$P_{straight} \models \exists D(d_{safe} < D_x \leq \alpha d_{safe} \wedge |D_y| \leq \frac{1}{2}(L+tol)) \tag{5.11}$$

Figure 5.10A shows the robot using the partition $P_{straight}$ to respond to a situation $\mathscr{S}_i$. In this case, the goal is in partition $P^G_{straight}$ and Equation 5.11 is satisfied, so it is predicted that the robot can drive straight distance d^{target} by executing the straight task T_S. The task is therefore simulated resulting in a new situation $\mathscr{S}_{i+1}$. However, while the goal is still in partition $P^G_{straight}$, Equation 5.11 is satisfied, so it is not possible to execute T_S and drive straight distance d^{target}. It is therefore necessary to increase the lateral distance to the goal to avoid the disturbance D.

Drive Left/Right

We next define partitions to predict whether the robot can successfully execute either of sequence $T_L \rightarrow T_S$ or $T_R \rightarrow T_S$. We generate short sequences of tasks for disturbance elimination because objects are spatially extended in two dimensions and error to the goal D^G is two dimensional. Hence it will always be necessary for a rotation task $T_{L/R}$ to be followed by a straight task T_S when attempting to reach a goal on the other side of a disturbance (see Figure 5.2 for example).

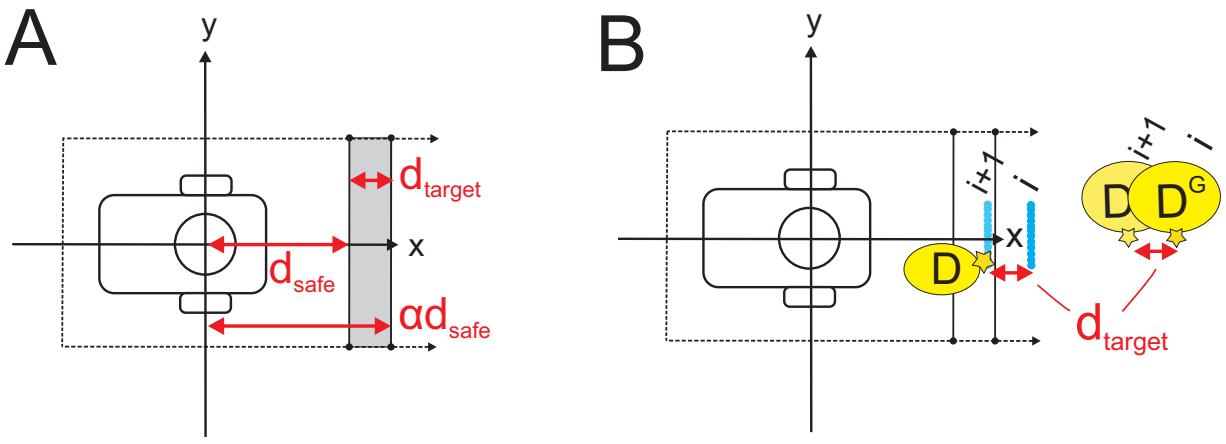

Figure 5.10: A: relevant parameters for partiton $P_{straight}$. B: example of our robot simulating the task T_S in response to situation $\mathscr{S}_i$ and witnessing a disturbance in $P_{straight}$ for situation $\mathscr{S}_{i+1}$.

Construction of partitions for predicting the discrete outcomes of each sequence is illustrated in Figure 5.11A, which also shows an example of the robot executing the sequence $T_L \rightarrow T_S$ in response to disturbance D. Formal definition of the partition for sequence $T_L \rightarrow T_S$ is the same as partition P_y^+ in Equation 4.9 and for sequence $T_R \rightarrow T_S$ it is the same as partition P_y^- in Equation 4.10. The corresponding existential conditions in Equations 4.11 and 4.12 therefore apply. In either case, the final control task T_S is associated with the target distance $d^{target} = d_{max} - d_{safe}$. Again the procedure for constructing lateral partitions at runtime is provided in Algorithm 1.

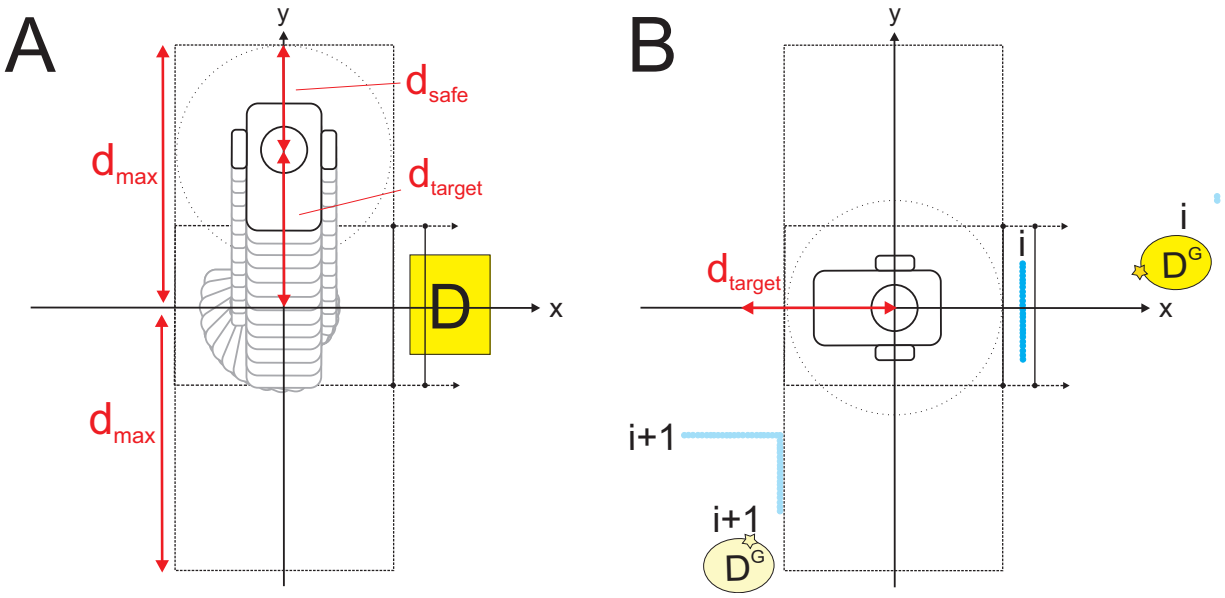

Figure 5.11: A: lateral partitions showing distance d_{target} and example of robot executing sequence $T_L \rightarrow T_S$. B: example of the robot simulating $T_L \rightarrow T_S$ in response to situation $\mathscr{S}_i$.

Figure 5.11B shows the egocentric perspective of the robot as it uses partition $P_{straight}$ to respond to disturbance D for situation $\mathscr{S}_i$. The goal is in partition $P^G_{straight}$, so ideally the robot would drive straight, however Equation 5.11 is satisfied so it is predicted that this is not possible. As Equation 4.11 is not satisfied, the robot selects the sequence $T_L \rightarrow T_S$ illustrated in Figure 5.11A (from the perspective of an external observer) to eliminate the lateral dimension of the disturbance D. The task is therefore simulated resulting in a new situation $\mathscr{S}_{i+1}$. The goal D^G is now in partition P^G_{right} and partition P_y^- is empty so Equation 4.12 is not satisfied. Consequently, it is possible to generate the sequence $T_R \rightarrow T_S$ to continue eliminating the disturbance D.

Simulation of Tasks

As indicated by our abstraction for situational analysis, a sequence of tasks generated for a situation $\mathscr{S}_i$ is a tuple $\mathscr{T}_i = \langle T_i, T_i \rangle$ where the final element could be null (i.e., only T_S is generated). The procedure for simulating a generated sequence of tasks $\mathscr{T}_i$ is based on simple coordinate geometry. While this may be imprecise, due to error in actuation and tracking, it is lightweight

and therefore suitable for real-time performance on a low-powered device. Our set of tasks is simple and the abstraction somewhat course which helps with imprecision, hence the simulation method is sufficient for our purposes. Our use of replanning also prevents it from becoming too problematic. The procedure for simulating sequences of tasks is outlined in Algorithm 5.

Algorithm 5: Simulate Tasks for Situation $\mathscr{S}_i$

Input : Situation $\mathscr{S}_i = \langle O_i, D_i^G \rangle$ and generated sequence of tasks $\mathscr{T}_i$
Output: Successor situation $\mathscr{S}_{i+1}$

```
1  Procedure SimulateTasks(S_i, T_i)
2      x ← 0
3      y ← 0
4      O_{i+1} ← ∅
5      D^G_{i+1} ← ∅
6      if T_i[0] = T_S then
7          x ← getDistance(T_i[0])
8      else
9          y ← getDistance(T_i[1])
10     for o ∈ S_i[0] do
11         o' ← Translate(o, x, y, T)
12         add o' to O_{i+1}
13     D^G_{i+1} ← Translate(S_i[1], x, y, T)
14     S_{i+1} = ⟨O_{i+1}, D^G_{i+1}⟩
15     return S_{i+1}
16 Procedure Translate(o, x, y, T)
17     o' ← ∅
18     if length(T) > 1 then
19         if T_0 = T_L then
               // T_L → T_S
20             o'_x ← o_y − y
21             o'_y ← −o_x
22         else
               // T_R → T_S
23             o'_x ← o_y − y
24             o'_y ← o_x
25     else
           // T_S
26         o'_x ← o_x − x
27         o'_y ← o_y
28     return o'
```

5.3.5 Task-Driven Transition System

The abstraction in the previous section underpins our transition system which models a short sequence of tasks $\mathscr{T}$. The distribution of states for our transition system is shown in Figure 5.12A. We have one initial state $s_0 = [d_{safe}, 0, 0]$ which is offset distance d_{safe} from the origin of the vector space. As the heading 0 is indexical to the direction the robot faces, all paths in our model start from the state s_0. As in Chapter 4, the states $s_1 = [0, d_{safe}, \frac{1}{2}\pi]$ and $s_2 = [0, -d_{safe}, -\frac{1}{2}\pi]$ represent transitions T_L and T_R, respectively, from state s_0 and are fixed. Note that as states s_0, s_1 and s_2 are offset distance d_{safe} from the origin, they represent a point on the plane in front of the robot on the edge of the safe zone. If the robot executes T_L or T_R from state s_0, then from the egocentric perspective of the robot, this point remains directly ahead, but from the perspective of an external observer it moves in an arc from state s_0 to either s_1 or s_2. The physical robot itself, however, remains at the origin and rotates left or right by $\frac{1}{2}\pi$ radians.

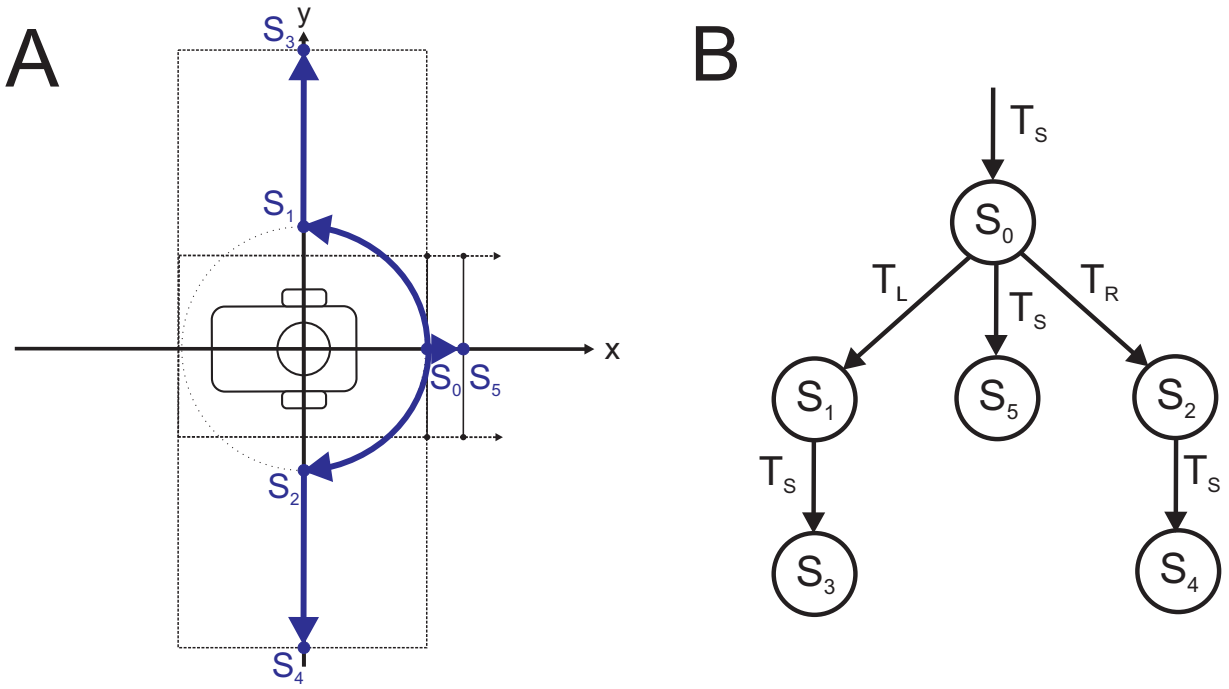

Figure 5.12: A: distribution of states for transition system. B: task-driven transition system. Note that the transition system is a tree structure because we are only interested in a finite portion of the state space relevant for a situation $\mathscr{S}_i$ and the subsequent state is unknown.

States s_3 and s_4 again represent driving straight in the lateral direction from states s_1 and s_2, respectively, and as in Chapter 4 their valuation varies according to the location of lateral disturbances. If there are no disturbances in the lateral partitions, then both Equations 4.11 and 4.12 are not satisfied, hence $s_3 = [0, d_{max}, \frac{1}{2}\pi]$ and $s_4 = [0, -d_{max}, -\frac{1}{2}\pi]$, which in this case means that the robot can drive straight distance $d_{target} = d_{max} - d_{safe}$ in either lateral direction. However, if there are lateral disturbances in each direction, both states would have a valuation such that $s_3 = [0, D_y^+, \frac{1}{2}\pi]$ and $s_4 = [0, D_y^-, -\frac{1}{2}\pi]$, hence both Equations 4.11 and 4.12 are satisfied,

meaning that it is *not* possible to drive distance $d_{target} = d_{max} - d_{safe}$ in either direction.

The final state s_5 represents driving straight in the longitudinal direction from the initial state s_0. If there is no disturbance in partition $P_{straight}$ then Equation 5.11 is satisfied, hence $s_5 = [\alpha d_{safe}, 0, 0]$, meaning that the robot cannot drive straight distance $d_{target} = \alpha d_{safe} - d_{safe}$. However, if there is a disturbance in partition $P_{straight}$ then Equation 5.11 is *not* satisfied, hence state $s_5 = [D_x, 0, 0]$, meaning that the robot can drive straight distance $d_{target} = \alpha d_{safe} - d_{safe}$.

The resulting transition system is depicted in Figure 5.12B. It is clearly simpler than the model developed in Chapter 4, however the planning procedure is more complex, as will be explained in the next section. While disturbances determine whether a sequence is possible in our model, they are not the only relevant consideration. Rather our focus is on generating a sequence of tasks $\mathscr{T}_i$ for a situation $\mathscr{S}_i$. We therefore call our model for goal-directed planning using model checking a *task-driven* finite transition system which we define as follows.

Definition 4 (Task-Driven Transition System) *A task-driven transition system TDTS is a tuple* $(S, Act, \rightarrow, I, AP, L)$ *where*

- $S = \{s_0, s_1, ..., s_5\}$ *is a fixed set of states;*
- $Act = \{T_S, T_L, T_R\}$ *is the set of tasks defined in Section 5.3.1;*
- $\rightarrow \subseteq S \times Act \times S$ *is a transition relation where* $s \rightarrow s'$ *is admissible if and only if a task can evolve the model from state s to* s'*;*
- $I = \{s_0\}$ *is the initial state;*
- AP *is a set of atomic propositions;*
- $L : S \rightarrow 2^{AP}$ *is a labelling function which determines which elements of AP are true at each state s.*

As should be clear from above, states in our model receive a valuation (labelling) at runtime using the abstraction approach described in Section 5.3.4, based on the location of disturbances.

5.3.6 Attract-Avoid Transition System

The $TDTS$ defined in the previous section is used to reason about the safety of a short sequence of tasks $\mathscr{T}_i$ for a situation $\mathscr{S}_i$. However, the transition system can only be used to reason about the *avoidance* of disturbances using model checking. We in addition want to prioritise *attraction* towards the goal as a bias to promote the minimal deformation of the robot path when negotiating disturbances. To do this, we require that the robot always prefers a task sequence which eliminates error to the goal D^G and that an appropriate avoid sequence is chosen when this is not

possible for a situation. We therefore need some way to resolve non-determinism in the $TDTS$ which promotes an attentional focus on the two dimensional geometry of a disturbance D.

We derive a secondary transition system from the $TDTS$ defined in Section 5.3.5 to further reason about the properties of task sequences, specifically to prioritise their order. We call our model an *attract-avoid* finite transition system which we define as follows.

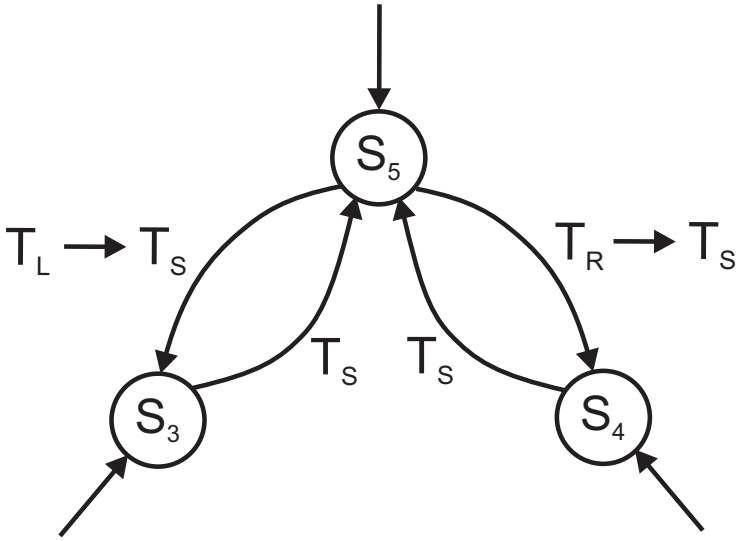

Figure 5.13: Attract-avoid transition system.

Definition 5 (Attract-Avoid Transition System) *An attract-avoid transition system $AATS$ is a tuple $(S', Act', \rightarrow', AP', I')$ derived from a TDTS where*

- $S' = \{s_5, s_3, s_4\}$ *is the set of final states in the TDTS for each possible sequence;*
- $Act' = \{T_S, T_L \rightarrow T_S, T_R \rightarrow T_S\}$ *is the set of possible sequences in the TDTS;*
- $\rightarrow' \subseteq S' \times Act' \times S'$ *is a transition relation where $s \rightarrow s'$ is admissible if and only if a sequence of tasks can evolve the TDTS from the initial state s_0 to a state in S';*
- $I' = \{s_3, s_3, s_4\}$ *is the set of initial states;*
- *AP is a set of atomic propositions.*

The $AATS$ is shown in Figure 5.13. The initial state for a run is determined by the relative direction of the goal D^G. For example, if the goal is straight ahead (i.e., partition $P^G_{straight}$ is not empty), then the initial state would be s_5. We explain how we use the $AATS$ in the next section.

5.3.7 Planning Using Model Checking

We use the *task-driven* transition system defined in Section 5.3.5 as an initial step to reason about the safety of possible task sequences for a situation $\mathscr{S}_i$ using the LTL property defined

in Equation 4.18. To then prioritise attraction and resolve non-determinism in the model, we use the *attract-avoid* transition system to impose a relevant ordering and select an appropriate sequence of tasks for the situation. We therefore use model checking as a discrete function $f : \mathscr{S}_i \mapsto \mathscr{T}_i$ which maps from a situation to a safe sequence of tasks. The task sequence $\mathscr{T}_i$ is then simulated using Algorithm 5 to produce a new situation $\mathscr{S}_{i+1}$ and the procedure repeats recursively until error to the goal D^G has been eliminated. Consequently, a *plan* is a sequence of task sequences $(\mathscr{T}_i)_{i\in\mathbb{N}}$. As our approach is best first, state-space explosion is minimised.

LTL Specification

As in Chapter 4, we use the property φ defined in Equation 4.18 which is stated in terms of two state properties, *horizon* and $safe$. However, the state properties have a different interpretation.

Definition 6 *Property horizon is true at state s if one of the following conditions hold: (i) the longitudinal dimension at s is* αd_{safe}*; (ii) the absolute value of the lateral dimension at s is* d_{max}*.*

In Definition 6 condition (i) ensures that states s_3 and s_4 are valid horizon states in the event that there are no lateral disturbances, and condition (ii) that state s_5 is a valid horizon state when there is no longitudinal disturbance, meaning that the robot can drive straight distance d_{target}.

Definition 7 *Property* $safe$ *is true at state s if one of the following conditions hold: (i) the absolute value of one dimension at s is* d_{safe}*; (ii) proposition horizon is true at s.*

In Definition 7 condition (i) ensures that rotations are safe due to construction of the safe zone defined by the parameter d_{safe}. Relevant states around the origin have one dimension equal to d_{safe}, and this uniquely identifies them. Condition (ii) guarantees that a state is safe when there are no disturbances preventing the robot from driving straight distance d_{target}.

Reasoning About Safety

For a situation $\mathscr{S}_i$ we first use the situational analysis approach described in Section 5.3.4 to arrive at a valuation of states and assign a valuation to the set AP. It is then possible to determine the set of paths which satisfy the negation of the property $\neg\varphi = safe\ \mathbf{U}\ (safe \wedge horizon)$ by calculating the automaton formed as the product of $TDTS$ and the NFA $\mathscr{A}_{\neg\varphi}$ (see Section 4.3.5). We again implement the underlying graph of the $TDTS$ and compute the product for a situation at runtime, which in practice simply amounts to identifying which states in the graph are terminal.

Figure 5.14A shows one example of a situation $\mathscr{S}_i$ resulting in the valuation of state properties for $TDTS$, illustrated in Figure 5.14B. The state property $safe$ is always true for states $\{s_0, s_1, s_2\}$ by condition (i) of Definition 7. As Equation 5.11 is not satisfied, state $s_5 = [\alpha d_{safe}, 0, 0]$, hence by condition (i) in Definition 6 state property *horizon* is true and state property $safe$

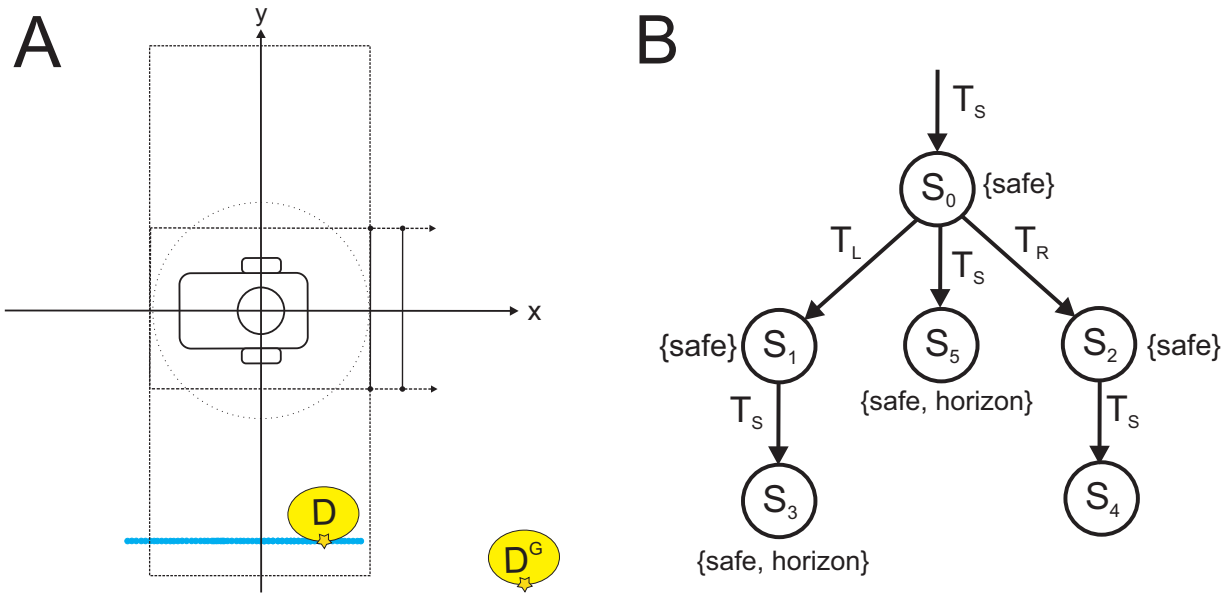

Figure 5.14: A: example situation $\mathscr{S}_i$. B: valuation of state properties for situation in A.

is true by condition (ii) in Definition 7. Equation 4.12 is satisfied, so the valuation of state $s_4 = [0, D_y^-, 0]$, hence by Definitions 6 and 7 both state properties are false. Finally, Equation 4.11 is not satisfied, hence the valuation of state $s_3 = [0, d_{max}, 0]$. By condition (ii) in Definition 7 the state property *horizon* is therefore true and by condition (ii) in Definition 7 property $safe$ is also true. The resulting product automaton $TDTS \otimes \mathscr{A}_{\neg\varphi}$ is illustrated in Figure 5.16.

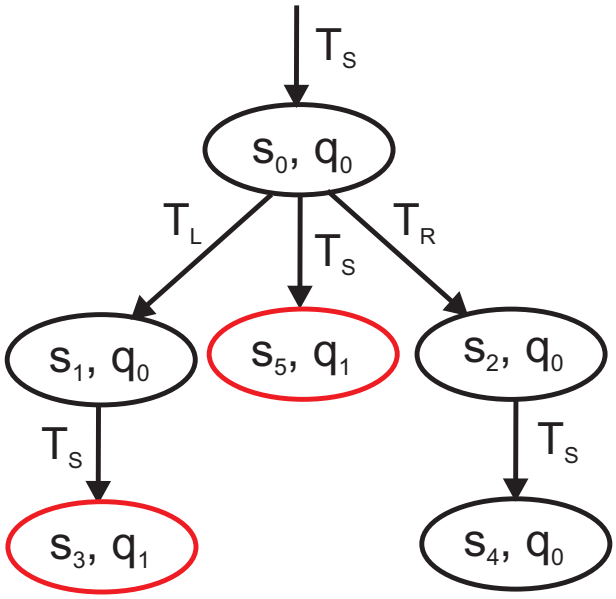

Figure 5.15: Product automaton $TDTS \otimes \mathscr{A}_{\neg\varphi}$ for example in Figure 5.14.

Two sequences are valid for the situation in Figure 5.14A. At this point, if we simply generated a sequence using f-DFS, as in Chapter 4, there is a possibility that the sequence $T_L \rightarrow T_S$ could be generated due to non-determinism. From the perspective of an external observer, it is

obvious that this is undesirable, as while the disturbance D would be avoided, the robot would then be driving away from the goal D^G. The clear option is to execute task T_S and continue doing so until it is possible to execute $T_R \rightarrow T_S$ and move towards the goal. For this we use the $AATS$.

Prioritising Attraction to the Goal

As the $AATS$ is formed from states in the $TDTS$ the same valuation of state properties applies. In our example, we have already reasoned about which of the possible final states in the $TDTS$ satisfy the property $safe$ **U** $(safe \wedge horizon)$ and calculated the product automaton $TDTS \otimes \mathscr{A}_{\neg\varphi}$. Hence for the states $S' = \{s_5, s_4, s_3\}$ it is known for which states the conjunction $safe \wedge horizon$ is true. The resulting valuation of state properties for the $AATS$ is shown in Figure 5.16.

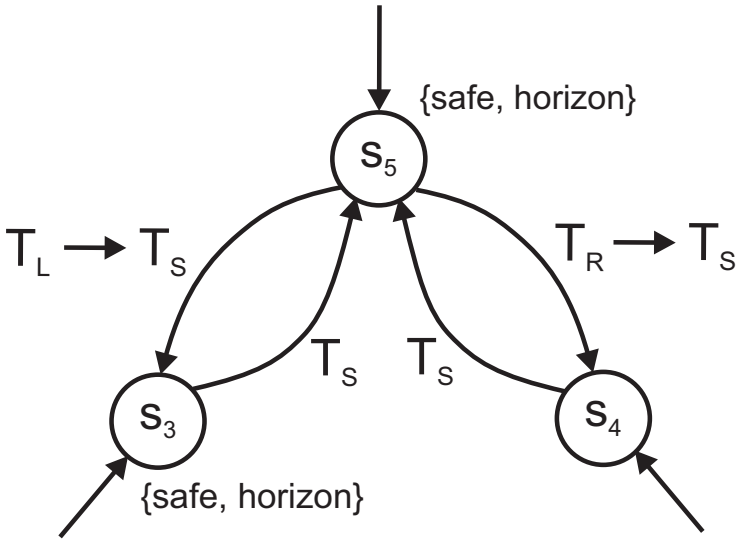

Figure 5.16: Valuation of state properties in $AATS$ for example in Figure 5.14.

We are interested in returning a state that satisfies the property $safe \wedge horizon$ as this corresponds to a terminal state for a safe sequence in $TDTS$. We define a second property $\Box\phi$ where

$$\phi = \neg(safe \wedge horizon) \tag{5.12}$$

which states that $\neg(safe \wedge horizon)$ is an invariant condition that holds for every state. The set of counterexamples for ϕ constitute a language of finite words which can be recognised by an NFA. We therefore construct another NFA $\mathscr{A}_{\neg\Box\phi} = (Q', \Sigma, \delta', Q'_0, F')$ where $Q' = \{q'_0, q'_1\}$ is the set of states, $\Sigma = 2^{AP}$ is a finite alphabet, $\delta' : Q' \times \Sigma \rightarrow 2^{Q'}$ is a transition relation, $Q'_0 = \{q'_0\}$ is the initial state, and $F' = \{q'_1\}$ is the accepting state. Figure 5.17 shows the NFA $\mathscr{A}_{\neg\Box\phi}$.

When a valuation has been assigned to the set AP for the $TDTS$, we calculate the product automaton for $AATS$ and $\mathscr{A}_{\neg\Box\phi}$ and generate a (finite) path using breadth-first search (BFS). The algorithm returns a path to the first state it encounters where the property $safe \wedge horizon$

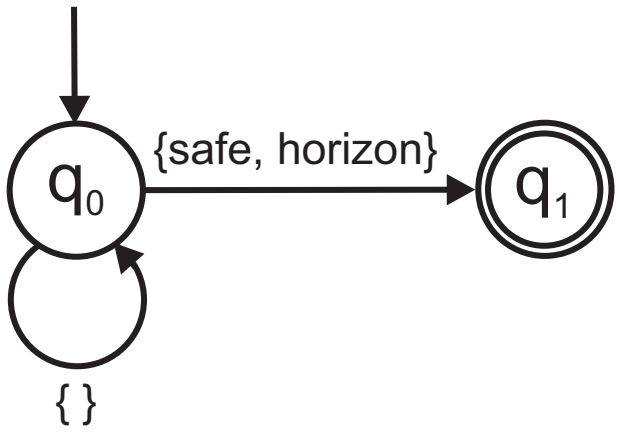

Figure 5.17: NFA for the property $safe \wedge horizon$.

is true. The initial state is determined by the relative location of the goal using the abstraction defined in Section 5.3.3. Consequently, if the property $safe \wedge horizon$ is true at the initial state, then a single state is returned constituting a path of length zero. However, this is sufficient as we are only interested in the final state to indicate the relevant sequence in the $TDTS$. If it is the initial state that is returned, then the selected sequence is driven by attraction to the goal D^G. Otherwise a path with a final state representing a relevant avoidance sequence is returned. We then use the final state from the returned path as the terminal state for $TDTS$ to generate the relevant sequence of tasks $\mathscr{T}_i$ for the situation using f-DFS. Note however that as the sequence is fully determined at this point, a simple if-then-else algorithm would be equally sufficient.

Recursive Forward Planning

The combined $TDTS/AATS$ approach makes it possible to reason about the relevant sequence of tasks $\mathscr{T}_i$ for a situation $\mathscr{S}_i$ in a way that promotes minimal deformation of the robot path when negotiating a two dimensional object. As mentioned above, a *plan* is a sequence $(\mathscr{T}_i)_{i\in\mathbb{N}}$ generated using forward simulation of task sequences via the procedure in Algorithm 5. Hence a plan corresponds to a sequence of simulated situations. Planning can therefore be viewed as a proxy for imagination in biological organisms, e.g., human beings (see Section 3.3 for details).

However, as the robot has an egocentric perspective on the environment, it only has partial information available when generating a plan, making it necessary to replan for accurate obstacle avoidance. When encountering an object (i.e., a disturbance D) in the path of the robot, the point cloud provides incomplete information on the dimensionality of the object as it is mostly hidden from view. In addition, due to error in actuation and tracking, the relevant sequence $\mathscr{T}_i$ for a situation during the real execution may depart from what has been planned. One possible strategy for replanning is to take a receding horizon approach and generate a new plan after the execution of each task sequence. However, to minimise computation, we focus on replanning only when the relevant sequence during execution deviates from the sequence that is expected.

The procedure for generating plans is outlined in Algorithm 6. When checking the relevant sequence for the current situation, only the abstraction is used so there is no recursion, indicated by the boolean parameter *reactive*. Otherwise we call the procedure recursively to eliminate all error to the goal, based on a partition for witnessing whether the error is within a bound on the x-coordinate and y-coordinate (we define this partition in Section 5.4.2). We also include the parameter *recursionTarget* as a limit on the depth of recursion for experimentation purposes, as will be explained in Section 5.5. At each step in the recursion, a sequence $\mathscr{T}_i$ is generated and added to a queue of tasks (i.e., a plan), which is then retrieved by the agent after the procedure completes to begin execution. The procedure *generateState()* uses the $AATS$ to generate an accepting state via BFS indicating the desired sequence, which is used by the *generateSequence()* procedure to generate a sequence of tasks $\mathscr{T}_i$ relevant for the situation $\mathscr{S}_i$ using f-DFS.

5.4 Implementation Details

As with the approach developed in Chapter 4, our method was implemented on the low-powered robotic platform described in Section 3.1. Planning using model checking is applied recursively using simulation to generate a path which eliminates error to the goal D^G. If an unexpected disturbance D is encountered during the real execution, or the robot determines that the best sequence of tasks $\mathscr{T}_i$ for a situation does not match what is planned, then the robot replans a path to the goal. We now outline the agent architecture and reactive replanning mechanism.

5.4.1 Agent Architecture

Figure 5.18 shows the agent architecture. As in Chapter 4, the agent mostly reasons reactively about the sensed environment state. The recursive model checking procedure described in the previous section is used to recursively plan a sequence of closed-loop tasks which negotiates any visible disturbance D and eliminates error to the goal D^G. The red aspects represent reasoning about task outcomes based on the current state of the environment and the blue aspects represent reasoning about the future using model checking. Reactive model checking (i.e., reasoning about the relevant sequence of tasks $\mathscr{T}$ for the current situation) is performed after the successful execution of a task, and if the first task in the sequence does not match, a new plan is generated.

5.4.2 Reactive Replanning

Figure 5.19 illustrates control flow for the task controller. When the robot initiates, it knows the location of the goal D^G but has no plan. A new scan event from the LiDAR calls the agent event handler and passes the set of observations O to the current task event handler, which by default is the straight task T_S, which returns a task result. As the straight task T_S has no control goal

Algorithm 6: Generate Plan Using Model Checking

Input : Observations O, error to goal D^G, boolean *reactive*, integer *recursionTarget*.
Output: Sequence of tasks $\mathscr{T}_i$.

Procedure `GeneratePlan(`O, D^G, D, *reactive*, *recursionTarget*`)`
  $\mathscr{T}_i \leftarrow \emptyset$
  $term \leftarrow \{s_5, s_3, s_4\}$
  $initAATS \leftarrow$ `GoalDirection(`D^G, d_{safe}, λ`)`
  $P_{straight} \leftarrow \emptyset$
  for $o \in O$ **do**
    if $o_x > 0 \wedge o_x \leq \alpha d_{safe} + d_{safe} \wedge |o_y| \leq \frac{1}{2}(L + tol)$ **then**
      add o to $P_{straight}$
  if $initAATS = s_5$ **then**
    if $P_{straight} \neq \emptyset \vee D \neq \emptyset$ **then**
      remove s_5 from *term*
    else
      $accept \leftarrow generateState(term, s_5)$
      $\mathscr{T}_i \leftarrow generateSequence(accept)$
  if $plan = \emptyset$ **then**
    $P_y^+ \leftarrow$ `ConstructPY(`O, d_{safe}, d_{max}, ***true***`)`
    $P_y^- \leftarrow$ `ConstructPY(`O, d_{safe}, d_{max}, ***false***`)`
    if $P_y^+ \neq \emptyset$ **then**
      remove s_3 from *term*
    if $P_y^- \neq \emptyset$ **then**
      remove s_4 from *term*
    $accept \leftarrow generateState(term, initAATS)$
    $\mathscr{T}_i \leftarrow generateSequence(accept)$
  if $\mathscr{T}_i[0] = T_S$ **then**
    add $d_{target} = \alpha d_{safe} - d_{safe}$ to $\mathscr{T}_i[0]$
  else
    add $d_{target} = d_{max} - d_{safe}$ to $\mathscr{T}_i[1]$
  if $\neg$ *reactive* **then**
    push $\mathscr{T}_i$ to queue
    $\mathscr{S}_{i+1} \leftarrow$ `SimulateTasks(`$\langle O, D^G \rangle$, $\mathscr{T}_i$`)`
    recursionDepth += 1
    if $recursionDepth < recursionTarget \wedge (|D^G[0]| > d_{safe} \wedge |D^G[1]| > d_{safe})$ **then**
      return `GeneratePlan(`$\mathscr{S}_{i+1}[0]$, $\mathscr{S}_{i+1}[1]$, $\langle 0, 0 \rangle$, ***false***, *recursionTarget*`)`
  $recursionDepth = 0$
  return $\mathscr{T}_i$

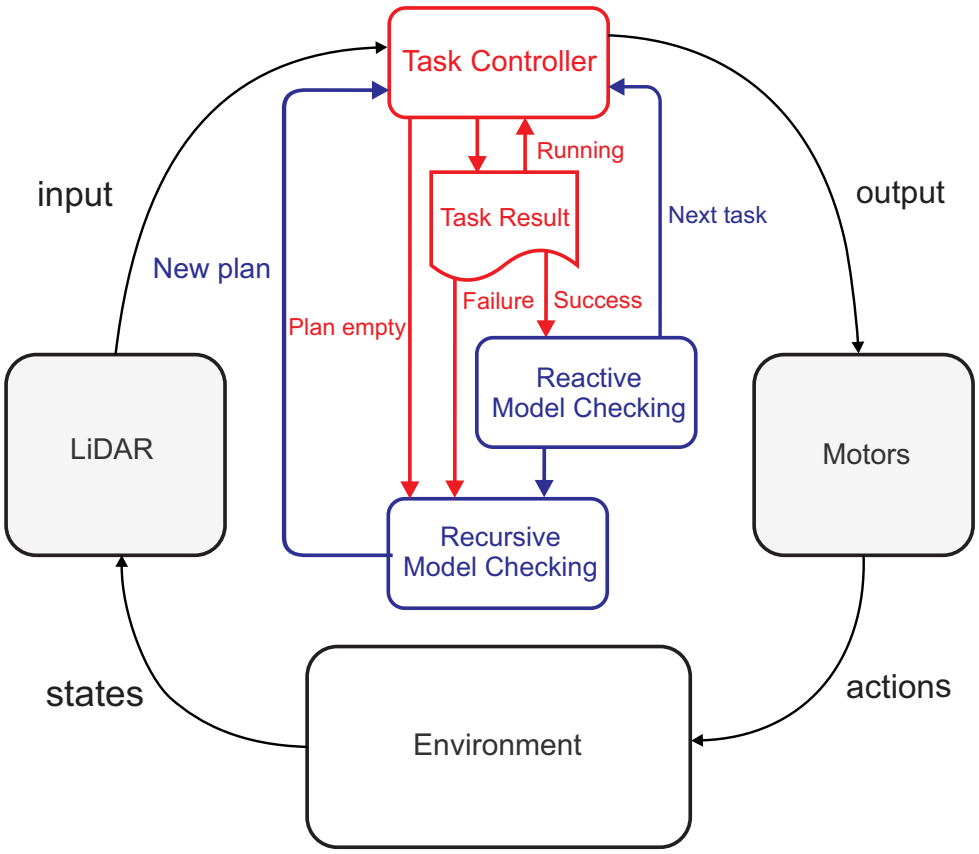

Figure 5.18: Agent architecture for goal-oriented obstacle avoidance.

at this point, because there is no plan, the task result is null. The location of the goal is then updated, which initially is the same as stipulated because the robot has not yet moved.

The robot then reasons whether it should go towards the goal. Due to imprecision in tracking and error in actuation, we define an egocentric partition around the robot which indicates whether the robot has (approximately) arrived at the goal $D^G = [x, y]$. This is defined as:

$$P_{goal} = \{x, y \mid x \leq d_{safe} \wedge y \leq d_{safe}\} \tag{5.13}$$

where the parameter d_{safe} defines an upper and lower bound on the coordinates of the goal D^G. If both coordinates are in the partition, then the robot has arrived so can stop. Otherwise, the robot reasons that it still needs to make progress towards the goal, as shown in Figure 5.19B.

Next the robot reasons whether the task was a failure, i.e., whether an unexpected disturbance has been detected. If the task is a failure, then a new plan is generated, otherwise the robot continues down the reasoning chain to consider if the plan is empty. When first initiated, the task is not a failure but the plan is empty, so it generates a new plan. As the robot has not started execution of the plan yet, it reasons that the next task in the plan should be triggered. The control flow loop then ends, resulting in another new scan event ≈ 200 milliseconds later. This process repeats until the task is a success, at which point the robot checks the relevant task sequence

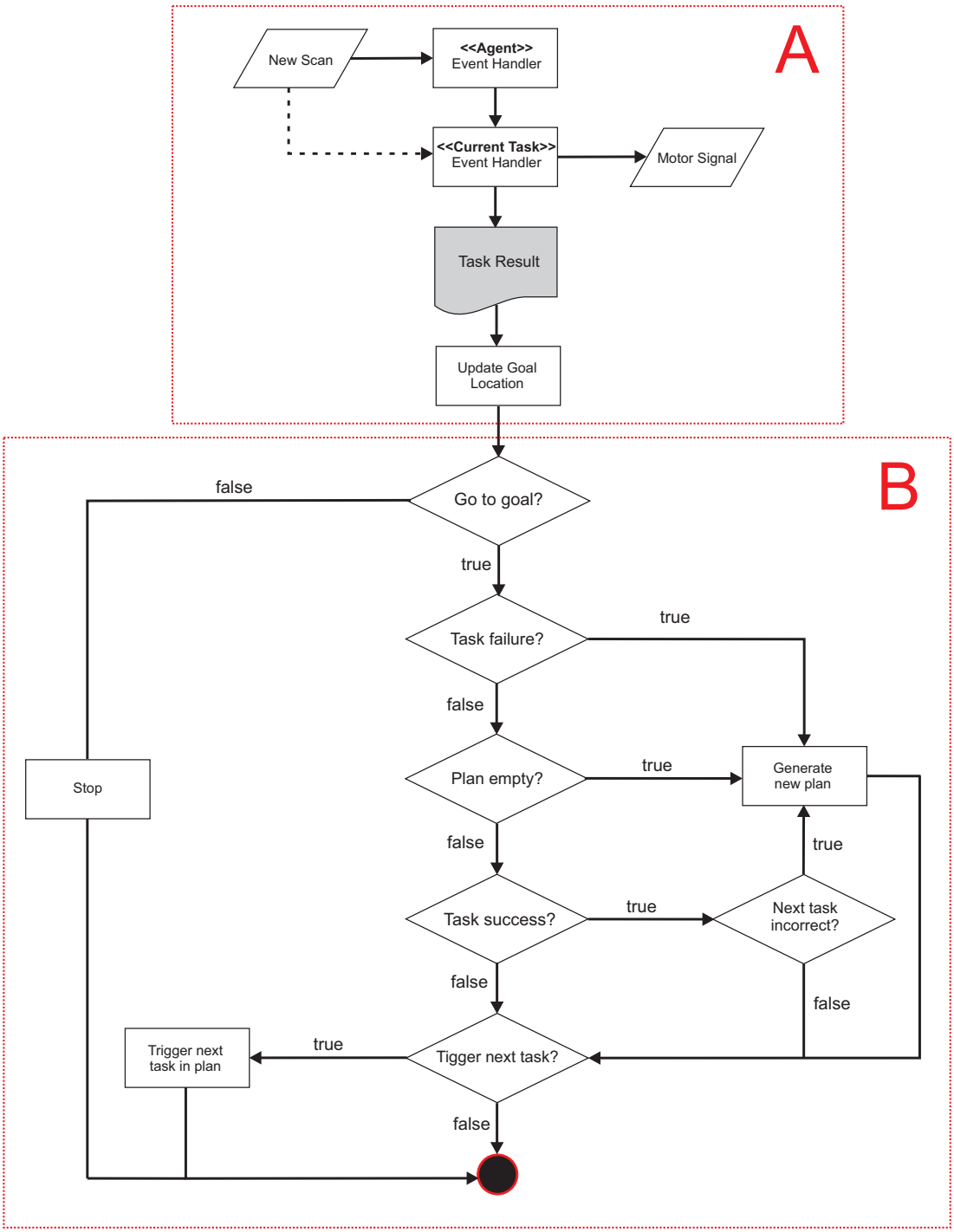

Figure 5.19: Control flow for reactive replanning. A: interface for discrete task outcomes and closed-loop update of goal location using dead reckoning. B: reactive reasoning chain.

for the current situation using reactive model checking (see Figure 5.18). If the first task in the sequence does not match the next task in the plan, then a new plan is generated. Otherwise the next task in the plan is triggered. This process continues until $P_{goal} \neq \emptyset$ and the robot stops.

5.5 Case Study II: Obstacle Avoidance

We describe the design of a case study used to validate our approach for an obstacle avoidance scenario similar to an overtaking task in autonomous driving. Following on from the case study in 4.6, we focused on reliability and memory usage. Specifically, we were interested in whether our approach reliably produces more efficient obstacle avoidance behaviour compared to the extended baseline described in Section 5.2. In addition, we were interested in whether our approach was able to reliably meet the real-time deadline. As in Section 4.6, we also wanted to profile the memory usage of both planning using model checking and the process for goal-directed obstacle avoidance. Hence we identified the following research questions for our study:

RQ6 *Does goal-direction planning using model checking reliabily generate more efficient collision avoidance behaviour compare to the baseline method?*

RQ7 *Does goal-directed planning using model checking reliably meet the real-time deadline of 100 milliseconds for autonomous driving applications?*

RQ8 *What is the memory usage of goal-directed planning using model checking?*

5.5.1 Environment Definition

We focus on an overtaking task in a static bounded environment. Our aim was to mimic overtaking in autonomous driving, though there are clear differences. Specifically, the environment is not dynamic and there are no oncoming vehicles which simplifies the task, however as our focus was on the efficiency of trajectories, this was considered sufficient. Figure 5.20 shows the environment setup. The environment has one object which the robot needs to pass to arrive at the goal $D^G = [1,0]$ (distance in meters). Note that we define arrival at the goal as stopping anywhere in the goal area, which reflects definition of the partition P_{goal} for witnessing the elimination of D^G. As the robot never rotates exactly $\pm\frac{1}{2}\pi$ radians, we balanced the study to control for any bias in the generated trajectories, with a large gap on the right for half of the runs (see Figure 5.20A) and the same size of gap on the left for the other half of runs (see Figure 5.20B).

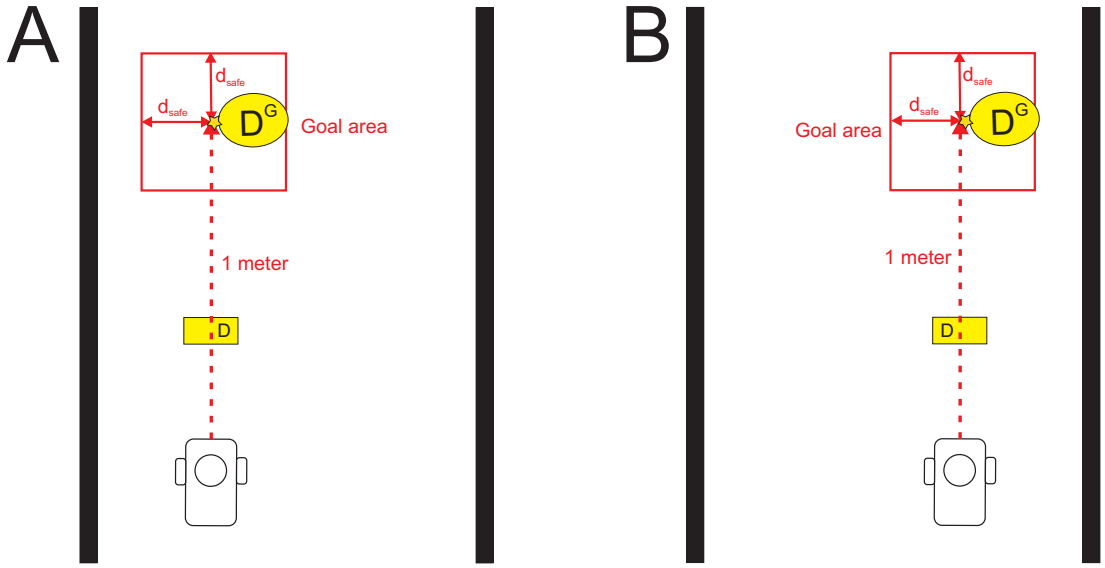

Figure 5.20: Environment setup. A: large gap on right for passing the disturbance D. B: large gap on left for passing the disturbance D. Red square represents goal area of same dimensions as the partition P_{goal}. The goal D^G is 1 meter away directly in front of the robot.

5.5.2 Artefacts

The baseline was described at a high level in Section 5.2. Similar to our method using model checking, the baseline method takes a situation $\mathscr{S}_i$ as input, generates a sequence of tasks $\mathscr{T}_i$ in response, simulates the outcome, and repeats the process recursively until the goal is reached. When the baseline method initiates planning to a goal, it first uses the goal abstraction in Section 5.3.3 to determine whether the goal is straight ahead, to the left, or to the right for the current situation and generates the corresponding control task. However, if the relevant task is T_S and the robot perceives a disturbance blocking its path, it selects either T_L or T_R, depending on whether the angle of the disturbance is greater or less, respectively, than 0 radians. As mentioned in Section 5.2, to prevent the robot from doubling back on itself, we in addition require that the robot executes a straight task immediately after a rotate task $T_{L/R}$, however the robot can only perceive disturbances directly ahead so it can only reason about a single disturbance at a time, unlike the model checking approach. Execution of the relevant tasks is then simulated and the process repeats until the goal is reached, at which point the plan is executed by the robot. As the environment is partially observed, the baseline can replan using the method in Section 5.3.7.

We compared the baseline method to goal-directed obstacle avoidance using model checking as described in Section 5.3. While our approach has the ability to eliminate all error to the goal, we were also interested in comparing the performance of our approach with a strict limit on recursion depth, as it seemed reasonable to expect this adjustment would lead to an improvement in processing latency, though generated plans would be incomplete until the goal is within range. Hence we in addition compared the performance of a second approach using model checking for planning and goal-directed obstacle avoidance where the recursion depth was limited to 5.

5.5.3 Analysis

We now provide details of our analysis. We collected data for 90 runs in total, 45 runs for the scenario shown in Figure 5.20A and 45 runs for the scenario in Figure 5.20B. For each scenario, we collected data for 15 runs from each method: the baseline planning method, model checking with no recursion limit, and model checking with a recursion limit of 5. Hence for each method there were 30 runs in total for our analysis of the research questions.

Analysis of RQ6

We focused on the construct of "reliability" to answer **RQ6**, specifically the comparative efficiency of trajectories between each method, as in Section 4.6. For the statistical analysis, we again used trajectory length as a proxy for time, conceived as a measure of minimal path deformation. This is based on the fact that a straight trajectory to the goal would be optimal, so a shorter trajectory represents a more efficient negotiation of the two dimensional disturbance D. As in Section 4.6, we also compared the number of collisions for each method. In addition, we compared the extent to which each method was successful in reaching the goal area.

Design of Analysis Our study had a 3×2 between-factors design for statistical analysis of trajectory lengths, reporting of collisions and goal attainment. The method variable had three levels: "Baseline" (baseline method), "No Limit" (model checking with no limit on recursion) and "Limit" (model checking with recursion limited to depth 5). The scenario variable had two levels: "Right" and "Left" as represented in Figures 5.20A and 5.20B, respectively.

Data Collection and Preparation During data collection, we used markers to indicate the correct placement of the robot. Runs were either terminated when the robot determined that it was at the goal, or if the robot drove out of the environment bounds. We also recorded the number of collisions, which we defined as the robot making a distinct sound when colliding with an object. In addition, we recorded whether the robot stopped in the goal area or not, marked off with tape. To account for cases where the robot might straddle the goal area, we defined success as the robot stopping with the centre of the LiDAR across the goal area threshold. As in Section 4.6, for each run we extracted the trajectory from video using the optical flow method in OpenCV and calculated the length of the trajectory in pixels prior to analysis.

Results First we compared trajectory lengths for each method while ignoring the scenario. A Kruskal-Wallis test was performed. It was found that there was a significant effect of method on trajectory length $H(2) = 29.4, p < .001$. Post-hoc analysis of pairwise comparison using Dunn's test (Bonferroni correction applied) revealed that there was a significant difference between the baseline method and both no limit model checking $(p < .001)$ and limit model checking $(p < .001)$, while there was no statistical difference between model checking methods. As

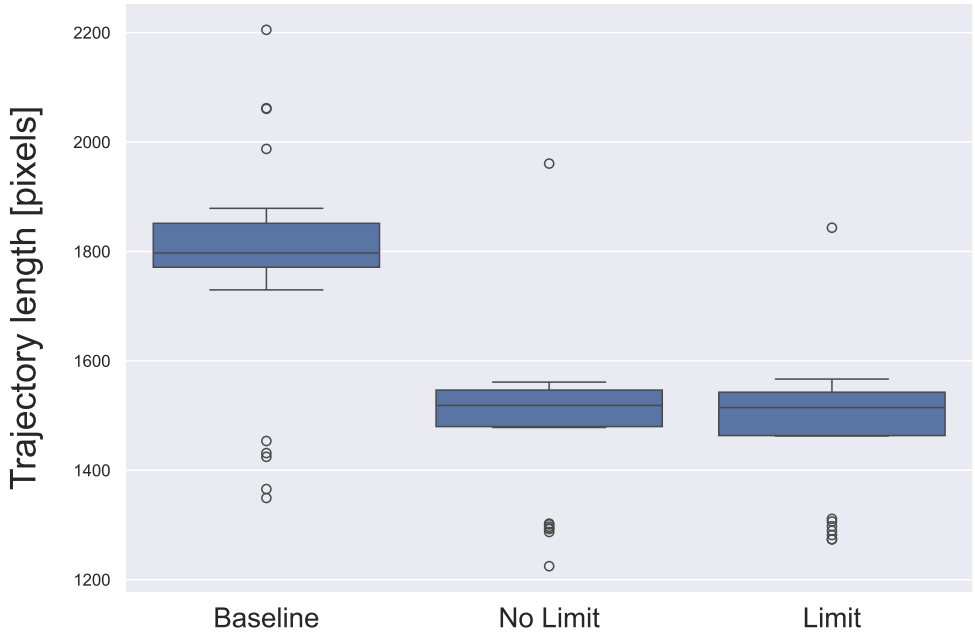

Figure 5.21: Box plot of trajectory lengths for each method across both scenarios.

indicated in Figure 5.21, the results suggest that the baseline method generates less efficient trajectories than both model checking approaches, but model checking approaches are the same.

Next we analysed the influence of the scenario on the generated trajectories to investigate the impact of asymmetries in the real execution. For each method, a non-parametric Mann-Whitney U test was performed $\alpha = .05$ (two-tailed). There was no statistical difference between the mean ranks for the baseline method. However, there was a statistical difference for no limit model checking $U = 225, n_1/n_2 = 15, p < .001$, with a medium effect size, $r = .49$. There was also a statistical difference for limit model checking $U = 241, n_1/n_2 = 15, p < .001$, again with a medium effect size, $r = 0.44$. The result suggest that the scenario has a significant effect on the generated trajectories for the model checking methods. The null result for the baseline method could be due to greater variation in the generated trajectories due to randomness in planning.

Figure 5.22A shows one example of a trajectory generated by the no limit model checking approach with a large gap for passing the disturbance on the left. In Figure 5.22B, a generated trajectory for the baseline is shown for the same scenario. In either case, the robot successful navigates to the goal, however it is clear that the baseline trajectory is longer. In comparison, the model checking approach takes a shorter route which minimally deforms the path of the robot.

The number of collisions was also observed for each run. For the baseline, 14 collisions were observed over 30 runs and no collisions were observed for either model checking method. We also recorded successful navigation to the goal. The baseline successfully navigated to the goal for 13 runs out of 30 and both model checking methods were successful for all 30 runs.

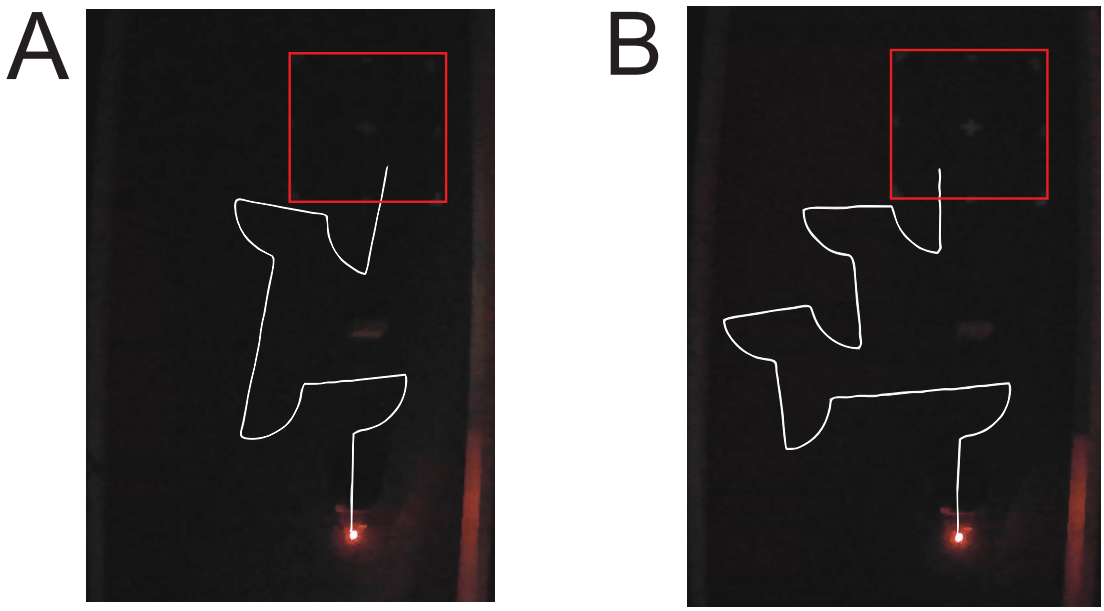

Figure 5.22: Example of trajectories. A: No limit model checking. B: baseline method.

Analysis of RQ7

We focused on the construct of "reliability" to answer **RQ7**, operationalised for this scenario using processing latency, as in Section 4.6. We aimed to describe the mean processing latency for planning using model checking and estimate confidence bounds to assess whether our approach can reliably meet the hard real-time deadline of 100 milliseconds.

Design of Analysis As in Section 4.6, we calculated the minimum, maximum and mean processing latency for all no limit and limit model checking runs (each method 30 runs in total) and estimated 95% confidence intervals using the bootstrap method. We also compared processing latency for each method using significance testing, as it was expected that the processing latency for limit model checking would be less than no limit model checking.

Data Collection and Preparation Processing latency and algorithm memory usage was collected for all no limit and limit model checking runs, 30 runs each. We used a built-in package in C++ to collect processing latency at runtime and dumped the data to a log.

Results The minimum processing latency for no limit model checking was 20.9 milliseconds and the maximum was 120.3 milliseconds. The mean was 57.4 milliseconds with an estimated 95% confidence interval of $[51.8, 63.7]$. For limit model checking, the minimum processing latency was 17.3 milliseconds and the maximum was 83.7 milliseconds. The mean was 35.5 milliseconds with an estimated 95% confidence interval of $[33.9, 37.6]$. We also compared each method using statistical testing. A Mann Whitney U test was performed. It was hypothesised that the mean rank of limit model checking would be lower than the mean rank of no limit model

checking, $\alpha = .025$ (one-tailed). The test revealed that the difference was indeed significant $U = 2533, n_1 = 98/n_2 = 90, p < .001$, with a small effect size, $r = .21$ (see Figure 5.23).

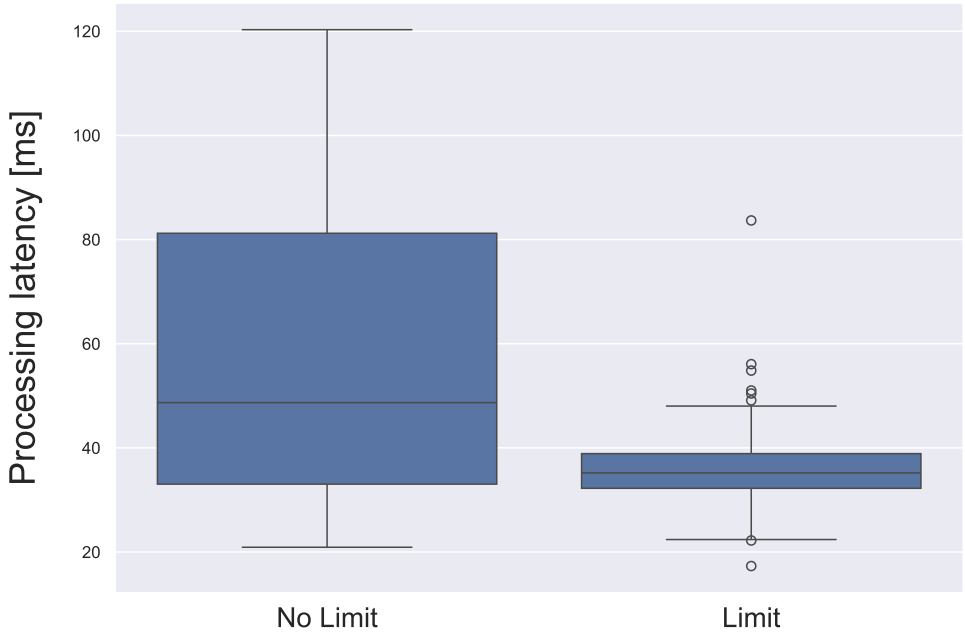

Figure 5.23: Comparison of processing latency for no limit and limit model checking methods.

Analysis of RQ8

We focused on describing memory usage to answer **RQ8**. As in Section 4.6, we identified no construct as our analysis was descriptive. We provide memory usage for all three methods.

Design of Analysis We estimated the runtime memory usage of both model checking and the entire program. We followed the same procedure as in Section 4.6 for estimating model checking memory usage for both no limit and limit model checking methods based on worst case usage at runtime and a state size of 4 bytes. We also estimated process memory usage for all methods.

Data Collection and Preparation Process memory usage was collected for all runs, collected using the Linux top utility at a sampling rate of 1 millisecond. We instrumented the code to record the growth of relevant data structure for f-DFS and BFS and dumped the data to disk. The data was then processed and cleaned in Python prior to analysis.

Results Maximum memory usage by no limit model checking at runtime was 2.34 kilobytes for BFS and 2.5 kilobytes for f-DFS. Maximum memory usage of the process was 5.02 kilobytes.

However, this includes the amount of shared memory available to a process, not all of which is typically resident to the program. The maximum amount of shared memory was 3.12 megabytes, hence a lower bound on the maximum was calculated at 1.94 megabytes. Maximum model checking usage was estimated to comprise 0.04% to 0.1% of maximum process memory usage.

For limit model checking maximum memory usage was also 2.34 kilobytes for BFS and 2.5 kilobytes for f-DFS. Maximum memory usage of the process was 4.4 kilobytes. The maximum amount of shared memory was 3.12 megabytes, hence a lower bound on the maximum was calculated at 1.26 megabytes. Maximum model checking usage was estimated to comprise 0.05% to 0.1% of maximum process memory usage.

In comparison, maximum process memory usage for baseline was 5.7 megabytes with a maximum shared memory component of 3.03 megabytes. In this case, a lower bound on the maximum was calculated at 2.7 megabytes. While there is uncertainty, the data suggests that memory consumption for model checking is less than consumption for the baseline.

5.5.4 Threats to Validity

Some of the threats to internal validity discussed in Section 4.6 apply. For example, the same issues with OpenCV are present, though experience from the previous study helped organise data collection to avoid some of the experienced pitfalls (e.g, inappropriate lighting conditions). However, some issues have not been over come, such as the uncertainty in metrics provided by the Linux top utility. Ultimately, a better method for profiling the memory usage would have been ideal. To address this shortcoming we made conservative estimates for memory usage.

There are a couple of new threats to the internal validity. One issue that became obvious after testing was a bug in the code from previous experimental development efforts which affected the smoothness of model checking trajectories. For some runs, this has the effect of making the robot perceive no lateral disturbance for some runs by adjusting the parameter of the lateral partition. While the robot is ultimately successful in reaching the goal, the side effect is that it executes an extra turn, notices a disturbance head on (i.e., the unseen lateral disturbance), then replans and corrects its course. While this affects the data, it does not ultimately undermine the result, which is achieved even though some runs produce longer trajectories than would have otherwise occurred. Once this is addressed, the trajectories should be yet more efficient.

Another issue with internal validity was manually deciding when the robot had arrived at the goal. There were some ambiguous cases where the centre of the LiDAR was on the edge of the goal area, primarily for the baseline method. If this was the case, we biased towards saying that the robot had arrived at the goal, though ultimately it was a judgement call. Ultimately, this was a consequence of the poor tracking using dead reckoning. One possible improvement would be to install an inertia measurement unit (IMU). Although this would not prevent error from accumulating in the system, it should be more reliable and allow for more flexible experimentation over greater distances. For example, if the tracking is sufficiently accurate, we could simply use

distance to the goal and some minimum threshold as a measure for success.

One issue with external validity is that our study focuses on one scenario (both scenarios are a reflection of each other). As we are interested in autonomous driving, a task similar to overtaking is appropriate for this thesis, however additional environmental setups would help demonstrate the generality of our approach, which in the present study is unclear. While our baseline is sufficient for initial research, comparing our method to other well-known algorithms for planning and obstacle avoidance, such as A*[82] or DWA[66], could strengthen our results.

5.5.5 Summary

We conducted a close study of trajectory lengths, observed collisions and recorded when the robot was successful in reaching the goal to answer **RQ6**. We found evidence that both no limit and limit model checking planning methods produce more efficient trajectories for goal-directed obstacle avoidance than the baseline planning method. However, there was no statistical difference between model checking planning methods. Over the course of 30 runs, 14 collision were observed for the baseline method but none were observed for either model checking method. Both model checking methods also reliably reached the goal for all 30 runs, whereas the baseline method only reached the goal 13 times. Hence planning using model checking had a 100% success rate for goal attainment while the baseline method success rate was only 43%.

To answer **RQ7** we collected data on processing latency. We found that mean processing latency (and confidence bounds on the population mean) for both no limit model checking and limit model checking were well within the 100 milliseconds deadline. We in also found a significantly lower processing latency in limit model checking method compared to no limit.

We collected data on memory usage to answer **RQ8**. While the data on memory usage is uncertain, we found some evidence that both model checking methods used less process memory than the baseline. In addition, mean processing latency for both no limit model checking and limit model checking was well within the 100 milliseconds deadline. RQs have been answered.

Response to RQ6 *Our evidence suggests that goal-directed planning using model checking reliably generates more efficient obstacle avoidance behaviour than the baseline method. Both model checking methods tend to spend less time completing the task (i.e., produce shorter trajectories), successfully avoid collisions, and have a 100% success rate in reaching the goal. Results also showed that there is no significant difference in trajectory lengths between each model checking method.*

Response to RQ7 *Our data indicates that both model checking methods meet the hard real-time deadline of 100 milliseconds. In addition, it was found that processing latency for limit model checking is significantly less than no limit model checking, though the effect size was weak. Our analysis suggests this holds for the population.*

Response to RQ8 *Our data suggests that maximum model checking memory usage for both methods was 2.35 kilobytes for BFS and 2.5 kilobytes for f-DFS. Maximum process memory usage for no limit model checking ranged between 1.94 and 5.02 megabytes, and for limit model checking it ranged between 1.26 and 4.4 megabytes. For comparison maximum baseline process memory usage ranged between 2.7 and 5.7 megabytes.*

5.6 Discussion

In this final investigation, we have extended the bespoke implementation of model checking in Chapter 4 to support the planning of avoidance and attraction closed-loop tasks. A layered abstraction was used to make discrete decisions about the relative location of the goal and reason about predicted outcomes for sequences of tasks for a situation, using a finite transition system and LTL property to reason about the safety of sequences. A second transition system built from a subset of states in the original was then defined to further reason about the properties of task sequences for a situation (i.e., whether the sequence was driven by attraction or avoidance). It was then possible to apply this recursively using simulation of task sequences to form a best-first approach for generating a plan to the goal. When the predicted sequence for a current situation was not expected, the replanning mechanism was triggered to handle partial observability.

Response to RQ4 *We can use a layered abstraction approach to reason about situations and define two interrelated transitions systems for reasoning about the types of sequences relevant for a situation. This makes it possible to reason about the properties of task sequences, specifically whether they are driven by attraction or avoidance. By applying this procedure recursively using simulation of task sequences, we can then forward plan to the goal. To in addition handle partial observability, we can replan when the predicted sequence for the current situation does not match the next task in the plan, or a disturbance is unexpectedly encountered. Hence we can replan only when necessary. This result in a best-first approach which minimises state-space explosion, promoting efficiency.*

In general, when updating plans specified using LTL to reflect environment changes in uncertain environments, two approaches are possible: reactive synthesis or iteratively updating the automaton. The former is computationally demanding, which is problematic for low-powered platforms like the robot used in this thesis. Similar to [114] we iteratively update the automaton, allowing us to replan as new information about the environment is received. While our approach in this chapter limits state space explosion using a best-first approach, the solution is somewhat convoluted due to the interaction of two automatons and untested on other scenarios. Indeed, the approach is likely coupled to the overtaking task as a consequence of extending the approach in Chapter 4 to accommodate goals, rather than viewing the goal-directed obstacle avoidance problem from a fresh perspective. It is therefore unlikely that the solution would be robust in cluttered environments. One possible alternative approach for goal-directed obstacle avoidance is to use the *iterative search for short trail* approach described briefly in Section 3.5.3. As this returns a path for successive model depths, it could be used to create a viable goal-directed obstacle avoidance method based on the shortest path which has a more robust general application.

As in Chapter 4, our solution guarantees that a generated plan is safe with respect to a set of possible continuous trajectories in the state space represented by our abstraction, and that a path to the goal will be found if there is one. However, there is no explicit guarantee that the robot will follow a continuous trajectory within a safe set, which we address by using over-approximation in our abstraction. Several attempts to provide robust safety guarantees for robotics systems have been proposed in the formal methods literature, however rigorously guaranteeing the safety of autonomous systems is in general a non-trivial task [132]. Indeed, one persistent issue is the "reality gap" which has lead to the use of runtime verification. In [54], for example, drones were simulated offline, plans generated by conventional means, and runtime monitors synthesised using model checking to ensure model properties are satisfied during execution. Both [157] and [128] synthesised a safety monitor as an extra layer between the motion planner and trajectory tracker to check if trajectories generated by the planner violated certain criteria. In [215] a monitor was synthesised based on verified models of a quadcopter flight controller to trigger corrective action (e.g., emergency landing). In contrast, we perform planning with the model

checker itself instead of checking if an existing algorithm leads to a dangerous situation [74]. We define a robot safe zone to ensure our approach does not lead to collisions during execution.

The main motivation for this is the complexity of modelling the physical environment [132] which is often partially observed and unstructured and therefore difficult to verify offline. While runtime verification methods can be used to check that runtime behaviour conforms to expectation and trigger remedial action, they can only reason about a single trace at a time so cannot be used for online planning. In comparison, our approach can reason actively about possible future paths in an online fashion based on sensor data, which is convenient for generating plans with guarantees of safety in unknown environments. If a plan is generated, it is *because* it is safe. While a runtime monitor could be synthesised to ensure the safe zone is never violated, this would only apply after a plan has been generated. If remedial action is then triggered, the extra processing could negatively impact the real-time performance of online planning.

Our approach is similar to [76] which uses a Büchi automaton (BA) to revise motion plans in real-time for uncertain environments, and in principle [101] which combines sampling-based point-to-point navigation with the deterministic μ-calculus to provide probabilistic guarantees. However, we do not generate the automaton for model checking on-the-fly, rather we maintain a static graph representing the structure of each transition system and update the relevant accepting state at runtime to form the product automaton and reflect sensed changes in the environment. This is possible because we work from an egocentric perspective using an abstraction with a fixed structure, however the consequence is that our method is coupled to the task domain so it will not scale. The upshot is that we do not have to compute the automaton which makes the implementation fast. In future work, we aim to use the iterative search for short trail option and automate construction of the automaton for more flexible plan specifications in LTL. Another option is to investigate timed automata for more explicit assurances on real-time performance.

Chapter 6

Conclusion

In this chapter we summarise the contributions of the thesis. First we discuss the extent to which the thesis statement has been addressed and research questions answered. We then discuss the limitations of the research in this thesis and provide a discussion of avenues for future work.

6.1 Answering the Research Questions

The research in this thesis was directed by the statement: *Model checking can be used to develop a trustworthy real-time planner for generating sequences of closed-loop controllers for obstacle avoidance which is light on computational resources, inherently explainable, and suitable for online decision-making*. We have addressed this by developing an online method for obstacle avoidance on a low-powered robot using model checking which is intrinsically explainable. Specifically, we focused on the following three objectives: (i) to evaluate the reliability of model checking for obstacle avoidance, (ii) to assess the reliability of model checking for real-time planning, and (iii) to describe the memory usage of planning using model checking. We therefore do not focus explicitly on generating explanations for the actions of our robot.

Our initial focus in Chapter 4 was on **RQ1**: *How can we use a bespoke model checker to plan sequences of discrete closed-loop tasks (which have an attentional focus on disturbances) for collision avoidance on a low-powered robotic system in real-time?* We addressed this by implementing a bespoke model checking algorithm integrated with the robot code which utilises an egocentric model of the local environment. This model abstracted away the temporal aspects of closed-loop tasks using the location of disturbances to determine discrete bounds on the low-level continuous behaviour. It was then possible to use a specification in LTL to reason in real-time about safe sequences of closed-loop tasks and generate temporally extended plans. From the primary research question in Chapter 4 and research objectives we derived three sub-questions which were addressed in the case study in Section 4.6. **RQ2** and **RQ3** concerned the reliability of model checking for planning and obstacle avoidance, as per research objectives (i) and (ii), and **RQ4** was concerned with memory usage as per objective (iii). Our results indicated

that our approach generated more reliable obstacle avoidance behaviour compared to a baseline within a hard deadline of 100 milliseconds and used minimal memory resources.

The approach in Chapter 4 showed that it was possible to implement model checking on a low-powered robot *in situ* for real-time planning. This was possible because we used abstraction to keep the state space small by only focusing on relevant information about disturbances, and because we integrated the mode checking procedure directly with the robot code using a static graph so did not have to compute the product automaton. However, the main limitation was that the approach was not goal-directed, as it could only spawn closed-loop tasks to avoid a disturbance, whereas goal-directed behaviour requires closed-loop tasks driven by attraction.

In Chapter 5 we therefore focused on **RQ5**: *How can we extend our bespoke implementation of model checking to plan sequences of avoidance and attraction closed-loop tasks for goal-directed obstacle avoidance on a low-powered robotic system in real-time?* We addressed this by using a version of our previous abstraction to reason about the best short task sequence for a local situation, which we then simulated. This procedure was repeated in a recursive fashion until the goal was reached. To handle uncertainty in the environment, the robot replanned if the local environment violated its expectation. As localisation was unreliable, we constructed an abstraction to determine the general direction of the goal, and to reason whether a task sequence is driven by attraction or avoidance we constructed a second automaton from the states of the base automaton. From the primary research question in Chapter 5 and research objectives we again derived three sub-questions which were addressed in the case study in Section 5.5. As per research objectives (i) and (ii), **RQ6** and **RQ7** were concerned with the reliability of our approach, and **RQ8** was concerned with memory usage as per research objective (iii). Our results indicated that our method generates more reliable obstacle avoidance behaviour compared to a baseline mostly within the hard deadline of 100 milliseconds and with a small memory footprint.

6.2 Limitations

The experimental work in this thesis lacks testing on a couple of typical edge cases known to be problematic for reactive obstacle avoidance methods in mobile robotics, so it is unclear how our approach generalises to other scenarios. For example, we have not tested our methods in either Chapter 4 or Chapter 5 in narrow passage ways or cluttered environments with many obstacles. While we have tested our approach in Chapter 4 on U-shaped obstacles, where many classical obstacle avoidance methods typically get trapped in local minima, we have not done this for the goal-directed method developed in Chapter 5. However, our data for real-time obstacle avoidance in Chapter 4 suggests that our approach may prove useful for avoiding getting trapped in corners, which could have applications in service robots (e.g., robot vacuum cleaners).

Another relevant issue in this regard is the scalability of our approach. As the method developed in Chapter 4 has a fixed maximum plan length, scalability is not problematic. We can

expect real-time performance to remain consistent regardless of the environment configuration as the maximum size of the state space is restricted. However, the generated plan length for our goal-directed method in Chapter 5 is dependent on the distance to the goal—the further away the goal, the deeper the level of recursion during the planning process. While most plans were generated within 100 milliseconds with a goal one metre away, this was not true in all cases unless the maximum depth of recursion was bounded. This indicates that our goal-directed method will not perform within real-time requirements when more steps are needed to arrive at a goal.

The implementation of our approach generally performs well regarding the real-time deadline of 100 milliseconds due to the direct integration of model checking into the robot code. However, the trade-off of this approach is a lack of flexibility when developing solutions for different navigation problems. While this was not too problematic for the obstacle avoidance method developed in Chapter 4, this is because it had a single objective—avoid obstacles. However, manually integrating the model checking procedure into the robot code soon became an issue for goal-directed avoidance where multiple objectives needed to be managed simultaneously. Ultimately, this resulted in a somewhat complicated solution. This may be because we attempted to extend our original solution to handle multiple objectives, so it is possible that a fresh perspective on the problem would have lead to a more straightforward (and scalable) solution. However, the sensible approach would be to separate the model checking code into an extensible library with a well-defined interface to compute relevant automata on-the-fly.

We also used a simple set of closed-loop tasks for this initial investigation, driving straight or rotating $\pm\frac{1}{2}\pi$ radians away from an obstacle. As mentioned in Section 3.1, this was a design choice to simplify the initial implementation of our approach, as our primary interest was in showing that it is possible to use model checking for planning sequences of closed-loop controllers (i.e., Braitenberg vehicles [28]) with an attentional focus on disturbances. One advantage of rotating $\pm\frac{1}{2}\pi$ radians to avoid obstacles is that it guarantees we will avoid convex obstacles, as the robot will then be perpendicular to the obstacle for subsequent straight movements. It also conveniently makes it possible to utilise symmetry on the axes of the vector space representing 2D LiDAR data for constructing relevant abstractions of the state space. However, this is not realistic for real-world applications. As the robot never rotates exactly $\pm\frac{1}{2}\pi$ radians, for example, navigation in cluttered environments with narrow gaps may not be feasible. However, more complex motion and abstraction methods could be incorporated into the basic approach.

This would require more precise localisation, which is a non-trivial task and indeed was a persistent issue even with the simple set of closed-loop tasks used in this thesis. Consequently, we had to construct an abstraction to indicate the general direction of the goal as error would quickly accumulate using dead reckoning. Without a prior map, the standard approach to localisation is to use some form of SLAM [164]. As mentioned in Section 4.7, however, this is computationally expensive—the localisation operation alone can impose a time lag in the order of hundreds of milliseconds [26, 120]. While sensor fusion techniques may achieve more pre-

cise localisation [209], experimental validation rarely measures the processing latency and often utilises more powerful processors than is available on a Raspberry Pi (e.g., [158, 202]).

6.3 Avenues for Future Work

There are several ways forward from the work in this thesis. Perhaps the most obvious avenue for further work is to investigate the use of other model checking approaches for online planning on a real robotic platform. As we focus on real-time performance, for example, we could investigate the use of timed automata [6] for planning. The research question would be whether we can generate online plans with guarantees on real-time performance using timed automata. Another question arising from this thesis is how to build a model checker optimised for real-time on a low-powered device in a stand alone library which computes automata on-the-fly. Ideally, this would be platform independent so it can work on a variety of robotic devices and support flexible adaptation of plan specifications for different domains, as discussed in Section 6.2. This would support investigation into applying hybrid planning and control techniques using model checking to different areas of mobile robotics, such as ariel and underwater robots. Automated techniques for discretizing the environment based on the properties of motion primitives would also be an interesting research direction to help speed up iteration and development, or alternatively abstraction-free approaches based on sampling methods (e.g., using RRT* [134]).

Although AVs promise to reduce road accidents through the elimination of human error, it is unreasonable to expect accidents will never happen. Consequently, the accident investigation process can be aided by explanations containing the vehicle's observations, the road rules and traffic signs it acted on, as mentioned in Section 2.1.2. While the legal context is still evolving [3, 94], the ability to audit vehicle actions is useful for determining liability in insurance claims and legal proceedings. It can also aide developers in the improvement of AV technology. Our model checking approach in this thesis uses temporal logic for plan specification which is explainable, however the generation of causal explanations is a non-trivial task [32, 102]. In addition, storing actions in a data recorder and recovering them with sufficient fidelity to reconstruct events is technically challenging [207]. In the interest of developing trustworthy methods for autonomous driving, one research direction related to this thesis is to investigate how to store and generate causal explanations for post-hoc analyses of plans generated using temporal logic.

We tested our approach in Chapter 4 extensively on a U-shaped obstacle, which is known to be problematic for classical real-time obstacle avoidance methods (e.g., AFP [106] and DWA [66]). As discussed in Section 4.7, this is because they typically predict the next action based on the current state so cannot reason about temporally extended goals. Indeed, our baseline method for the case study in Chapter 4 also lacked the ability to reason about extended temporal goals and consequently often got trapped on the U-shaped obstacles in our experiments. Our method using model checking, however, reliably did not do this which raises the question of

how our approach would compare to traditional obstacle avoidance methods. This would be worth pursuing in the interest of studying the role online planning using model checking can play in overcoming classical obstacle avoidance issues in mobile robotics through rich temporal specifications. As indicated in Section 6.2, this could have applications for small indoor service robots which cannot rely on GPS for localisation and do not have enough resources for SLAM. With some development, the approach may also be useful for rovers in unknown environments.

6.4 Summary

In this thesis we have shown that model checking and closed-loop control can be used for online planning and obstacle avoidance on a low-powered robot. As model checking uses a precise and unambiguous formalism, it is also explainable and therefore suitable for the development of trustworthy reasoning methods in autonomous driving. Our approach demonstrates that hybrid planning and control is feasible on a real autonomous robot with minimal resources under strict real-time constraints, a persistent challenge in the hybrid planning community. It also demonstrates that full model checking can be utilised as an online technique with respect to planning for autonomous systems, whereas the majority of research in this domain focuses on offline applications using established tools, or runtime verification for online monitoring. Most real-time obstacle avoidance approaches in the mobile robotics community are not able to plan for long horizons. Our approach in comparison does this well within hard real-time constraints. It is our hope that the work in this thesis is continued to benefit autonomous navigation for mobile robots in the short term and trustworthy reasoning for autonomous driving in the long term.

Appendix A

Promela Code for River Crossing Example

Figures A.1 and A.2 show listings which include full code for the `puzzle` proctype described in Section 3.5.3. The global variables, `mtype` declaration, inline function, propositions and LTL declaration are provided in Section 3.5.3. From the command line, to run SPIN and save the shortest path to an error file use `spin -search -i puzzle.pml`. To then run a guided simulation (and output associated actions) use `spin -t puzzle.pml`.

```
active proctype puzzle()
{

left_bank:
do
:: passenger==NULL && foxP==0 -> passenger=FOX; foxP=1;
   printf("\nPick␣up␣fox␣from␣left_bank\n");
   goto right_bank;
   /*pick up fox from left_bank*/
:: passenger==NULL && goatP==0 -> passenger=GOAT; goatP=1;
   printf("\nPick␣up␣goat␣from␣left_bank\n");
   goto right_bank;
   /*pick up goat from left_bank*/
:: passenger==NULL && cabbageP==0 -> passenger=CABBAGE; cabbageP=1;
   printf("\nPick␣up␣cabbage␣from␣left_bank\n");
   goto right_bank;
   /*pick up cabbage from left_bank*/
:: passenger==FOX && goatP==0 -> swap(foxP, goatP); passenger=GOAT;
   printf("\nReturn␣fox␣to␣left_bank␣and␣pick␣up␣goat\n");
   goto right_bank;
   /*return fox to left_bank, pick up goat */
:: passenger==FOX && cabbageP==0 -> swap(foxP, cabbageP);
   passenger=CABBAGE;
   printf("\nReturn␣fox␣to␣left_bank␣and␣pick␣up␣cabbage\n");
   goto right_bank
   /*return fox to left_bank, pick up cabbage*/
:: passenger==FOX -> foxP==0; passenger=NULL;
   printf("\nReturn␣fox␣to␣left_bank");
   goto right_bank
   /*return fox to left bank*/
:: passenger==GOAT && foxP==0 -> swap(goatP, foxP); passenger=FOX;
   printf("\nReturn␣goat␣to␣left_bank␣and␣pick␣up␣fox\n");
   goto right_bank
   /*return goat to left_bank and pick up fox*/
:: passenger==GOAT && cabbageP==0 -> swap(goatP, cabbageP);
   passenger=CABBAGE;
   printf("\nReturn␣goat␣to␣left_bank␣and␣pick␣up␣cabbage\n");
   goto right_bank
   /*return goat to left_bank and pick up cabbage*/
:: passenger==GOAT -> goatP=0; passenger=NULL;
   printf("\nReturn␣goat␣to␣left_bank");
   goto right_bank
   /*return goat to left_bank*/
:: passenger==CABBAGE && foxP==0 -> swap(cabbageP, foxP); passenger=FOX;
   printf("\nReturn␣cabbage␣to␣left_bank␣and␣pick␣up␣fox\n");
   goto right_bank
   /*return cabbage to left_bank and pick up fox*/
:: passenger==CABBAGE && goatP==0 -> swap(cabbageP, goatP);
   passenger=GOAT;
   printf("\nReturn␣cabbage␣to␣left_bank␣and␣pick␣up␣goat\n");
   goto right_bank
   /*return cabbage to left_bank and pick up goat*/
:: passenger==CABBAGE -> cabbageP=0; passenger=NULL;
   printf("\nReturn␣cabbage␣to␣left_bank");
   goto right_bank
   /*return cabbage to left_bank*/
od;
```

Figure A.1: Main Proctype for Crossing Puzzle

```
right_bank:
do
:: passenger==NULL && foxP==2 -> passenger=FOX; foxP=1;
   printf("\nPick␣up␣fox␣from␣right_bank\n");
   goto left_bank
   /*pick up fox from right_bank*/
:: passenger==NULL && goatP==2 -> passenger=GOAT; goatP=1;
   printf("\nPick␣up␣goat␣from␣right_bank\n");
   goto left_bank
   /*pick up goat from right_bank*/
:: passenger==NULL && cabbageP==2 -> passenger=CABBAGE; cabbageP=1;
   printf("\nPick␣up␣cabbage␣from␣right_bank\n");
   goto left_bank
   /*pick up cabbage from right_bank*/
:: passenger==FOX && (goatP+cabbageP<4) -> foxP=2; passenger=NULL;
   printf("\nDeposit␣fox␣at␣right_bank\n");
   goto left_bank
   /*deposit fox at right_bank*/
:: passenger==GOAT && (foxP+cabbageP<4) -> goatP=2; passenger=NULL;
   printf("\nDeposit␣goat␣at␣right_bank\n");
   goto left_bank
   /*deposit goat at right_bank*/
:: passenger==CABBAGE && (foxP+goatP<4) -> cabbageP=2; passenger=NULL;
   printf("\nDeposit␣cabbage␣at␣right_bank\n");
   goto left_bank
   /*deposit cabbage at right_bank*/
:: passenger==FOX && goatP==2 -> swap(foxP, goatP); passenger=GOAT;
   printf("\nDeposit␣fox␣to␣right_bank␣and␣pick␣up␣goat\n");
   goto left_bank
   /*deposit fox to right_bank and pick up goat*/
:: passenger==FOX && cabbageP==2 -> swap(foxP, cabbageP);
   passenger=CABBAGE;
   printf("\nDeposit␣fox␣to␣right_bank␣and␣pick␣up␣cabbage\n");
   goto left_bank
   /*deposit fox to right_bank and pick up cabbage*/
:: passenger==GOAT && foxP==2 -> swap(goatP, foxP);
   passenger=FOX;
   printf("\nDeposit␣goat␣to␣right_bank␣and␣pick␣up␣fox\n");
   goto left_bank
   /*deposit goat to right_bank and pick up fox*/
:: passenger==GOAT && cabbageP==2 -> swap(goatP, cabbageP);
   passenger=CABBAGE;
   printf("\nDeposit␣goat␣to␣right_bank␣and␣pick␣up␣cabbage\n");
   goto left_bank
   /*deposit goat to right_bank and pick up cabbage*/
:: passenger==CABBAGE && foxP==2 -> swap(cabbageP, foxP); passenger=FOX;
   printf("\nDeposit␣cabbage␣to␣right_bank␣and␣pick␣up␣fox\n");
   goto left_bank
   /*deposit cabbage to right_bank and pick up fox*/
:: passenger==CABBAGE && goatP==2 -> swap(cabbageP, goatP);
   passenger=GOAT;
   printf("\nDeposit␣cabbage␣to␣right_bank␣and␣pick␣up␣goat\n");
   goto left_bank
   /*deposit cabbage to right_bank and pick up goat*/
:: passenger==FOX && goatP==2 && cabbageP==2 -> foxP=2; passenger=NULL;
   printf("\nDeposit␣fox␣at␣right_bank\n");
   goto finish
   /*deposit fox at right_bank, go to finish*/
:: passenger==GOAT && foxP==2 && cabbageP==2 -> goatP=2; passenger=NULL;
   printf("\nDeposit␣goat␣at␣right_bank\n");
   goto finish
   /*deposit goat at right_bank, go to finish*/
:: passenger==CABBAGE && foxP==2 && goatP==2 -> cabbageP=2;
   passenger=NULL;
   printf("\nDeposit␣cabbage␣at␣right_bank\n");
   goto finish
   /*deposit cabbage at right_bank, go to finish*/
od;

finish: printf("\nFINISH\n"); goal=true

}
```

Figure A.2: Main Proctype for Crossing Puzzle continued.

Appendix B

Glossary

D Disturbance representing an obstacle. 53

T_0 Default straight task. 53

T_L Left closed-loop avoid task. 54

T_R Right closed-loop avoid task. 53

T_S Finite straight task. 55

r Radius of the safe zone. 56

$AATS$ Attract-Avoid Transition System. 110

DTS Disturbance-Focused Transition System. 69

D^G Disturbance representing two-dimensional error to the goal. 95

D_y^+ The y-coordinate of a disturbance in the positive lateral direction (m). 63

D_y^- The y-coordinate of a disturbance in the negative direction (m). 60

D_θ^{curr} The current angle to a disturbance. 56

D_θ^{init} The initial angle to a disturbance. 56

L The wheelbase of the robot (m). 57

O The set of observations generated by a LiDAR scan. 57

P_x^+ Longitudinal partition in front of the robot. 65

P_x^- Longitudinal partition behind the robot. 65

P_y^+ Lateral partition in the positive (left) direction. 60

P_y^- Lateral partition in the negative (right) direction. 61

P_{avoid} Difference between the current and initial angle to a disturbance. 56

P_{dist} Partition for distance travelled by straight task. 100

P_{left}^G Partition for determining whether goal is to the left. 102

P_{look} Triggers generation of a plan if a disturbance is witnessed. 58

P_{right}^G Partition for determining whether goal is to the right. 103

P_{rotate} Difference between the current and initial angle from the initial heading 0. 99

P_{safe} Partition for witnessing disturbances during execution of straight task.. 100

P_{shield} Triggers the next task in a plan if a disturbance is witnessed. 57

$P_{straight}^G$ Partition for determining whether goal is straight ahead. 101

$P_{straight}$ Partition for predicting if the robot can drive straight distance d^{target}. 104

$TDTS$ Task-Driven Transition System. 109

$T_{L/R}$ An arbitrary avoid task. 56

Δ_x The longitudinal offset for simulating a proximal disturbance. 60

Δ_y^+ An offset to simulate driving in the positive lateral direction (m). 63

Δ_y^- An offset to simulate driving in the negative lateral direction (m). 65

$\mathscr{S}_i$ Situation in a sequence comprising a set of observations and error to the goal. 104

$\mathscr{T}_i$ Sequence of tasks generated for a corresponding situation. 106

θ^{curr} Current orientation of the robot relative to initial heading 0. 99

d^{curr} Current distance travelled by robot when executing a straight task. 100

d^{target} Target distance for straight tasks. 100

d_{look} Upper bound on x-coordinate for witnessing disturbance in P_{look}. 58

d_{max} The maximum lateral distance of the abstraction (m) . 60

d_{min} Minimum distance robot can travel in either lateral direction (m). 63

d_{safe} The radius of the combined safe zone and outer shield (m). 60

d_{shield} Upper bound on the longitudinal dimension of disturbances. 57

tol Tolerance indicating over approximation for abstraction (m). 57

$|\varepsilon|$ Absolute error to be eliminated by rotate tasks. 99

Bibliography

[1] European Commission. High Level Expert Group on AI. *Ethics Guidelines for Trustworthy AI*. [Online]. 2019. URL: https://digital-strategy.ec.europa.eu/en/policies/expert-group-ai (visited on 01/13/2025).

[2] S. Al Mansoori, M. Al-Emran, and K. Shaalan. "Factors Affecting Autonomous Vehicles Adoption: A Systematic Review, Proposed Framework, and Future Roadmap". In: *International Journal of Human–Computer Interaction* 40.24 (2024), pp. 8397–8418. ISSN: 1044-7318, 1532-7590. DOI: 10.1080/10447318.2023.2286089.

[3] M. Alawadhi et al. "Review and analysis of the importance of autonomous vehicles liability: a systematic literature review". In: *International Journal of System Assurance Engineering and Management* 11.6 (Dec. 2020), pp. 1227–1249. ISSN: 0975-6809, 0976-4348. DOI: 10.1007/s13198-020-00978-9.

[4] A. Alessandrini et al. "Users' Preferences towards Automated Road Public Transport: Results from European Surveys". In: *Transportation Research Procedia* 3 (2014), pp. 139–144. ISSN: 23521465. DOI: 10.1016/j.trpro.2014.10.099.

[5] J. B. Almeida et al. "An Overview of Formal Methods Tools and Techniques". In: José Bacelar Almeida et al. *Rigorous Software Development*. Series Title: Undergraduate Topics in Computer Science. London: Springer London, 2011, pp. 15–44. ISBN: 978-0-85729-017-5 978-0-85729-018-2. DOI: 10.1007/978-0-85729-018-2_2.

[6] R. Alur and D. L. Dill. "A theory of timed automata". In: *Theoretical Computer Science* 126.2 (1994), pp. 183–235. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(94)90010-8.

[7] R. Alur, T. A. Henzinger, and P. Ho. "Automatic symbolic verification of embedded systems". In: *IEEE Transactions on Software Engineering* 22.3 (1996), pp. 181–201. DOI: 10.1109/32.489079.

[8] M. Antonello et al. "Flash: Fast and Light Motion Prediction for Autonomous Driving with Bayesian Inverse Planning and Learned Motion Profiles". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 9829–9836. DOI: 10.1109/IROS47612.2022.9981347.

[9] A. Anwar and A. Raychowdhury. "Autonomous Navigation via Deep Reinforcement Learning for Resource Constraint Edge Nodes using Transfer Learning". In: *IEEE Access* 8 (Oct. 2019), pp. 26549–26560. DOI: 10.1109/ACCESS.2020.2971172.

[10] M. Ayala-Rincón and C. A. Muñoz, eds. *Interactive Theorem Proving: 8th International Conference, ITP 2017, Brasília, Brazil, September 26–29, 2017, Proceedings*. Vol. 10499. Lecture Notes in Computer Science. Springer International Publishing, 2017. ISBN: 978-3-319-66106-3 978-3-319-66107-0. DOI: 10.1007/978-3-319-66107-0.

[11] F. Bacchus and F. Kabanza. "Planning for temporally extended goals". In: *Annals of Mathematics and Artificial Intelligence* 22 (1998), pp. 5–27. DOI: https://doi.org/10.1023/A:1018985923441.

[12] G. Bagschik, T. Menzel, and M. Maurer. "Ontology based Scene Creation for the Development of Automated Vehicles". In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. 2018, pp. 1813–1820. DOI: 10.1109/IVS.2018.8500632.

[13] C. Baier and J. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[14] M. Bain and C. Sammut. "A framework for behavioural cloning". In: K. Furukawa, D. Michie, and S. Muggleton. *Machine Intelligence 15*. Oxford University PressOxford, 2000, pp. 103–129. ISBN: 978-0-19-853867-7 978-1-383-02650-4. DOI: 10.1093/oso/9780198538677.003.0006.

[15] A. L. Baker et al. "Toward an Understanding of Trust Repair in Human-Robot Interaction: Current Research and Future Directions". In: *ACM Transactions on Interactive Intelligent Systems* 8.4 (2018), pp. 1–30. ISSN: 2160-6455, 2160-6463. DOI: 10.1145/3181671.

[16] P. Bansal, K. M. Kockelman, and A. Singh. "Assessing public opinions of and interest in new vehicle technologies: An Austin perspective". In: *Transportation Research Part C: Emerging Technologies* 67 (2016), pp. 1–14. ISSN: 0968090X. DOI: 10.1016/j.trc.2016.01.019.

[17] D. Basile, A. Fantechi, and I. Rosadi. "Formal Analysis of the UNISIG Safety Application Intermediate Sub-layer". In: *Formal Methods for Industrial Critical Systems*. Ed. by A. L. Lafuente and A. Mavridou. Springer International Publishing, 2021, pp. 174–190. ISBN: 978-3-030-85248-1. DOI: 10.1007/978-3-030-85248-1_11.

[18] F. Batsch et al. "A taxonomy of validation strategies to ensure the safe operation of highly automated vehicles". In: *Journal of Intelligent Transportation Systems* 26.1 (2022), pp. 14–33. ISSN: 1547-2450, 1547-2442. DOI: 10.1080/15472450.2020.1738231.

[19] J. A. Baxter et al. “Review of Electrical Architectures and Power Requirements for Automated Vehicles”. In: *2018 IEEE Transportation Electrification Conference and Expo (ITEC)*. Long Beach, CA: IEEE, 2018, pp. 944–949. ISBN: 978-1-5386-3048-8. DOI: 10.1109/ITEC.2018.8449961.

[20] A. Becker and L. Huang. *Rapidly Exploring Random Tree (RRT) and RRT**. [Online]. 2018. URL: https://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/ (visited on 03/02/2025).

[21] V. Belle and I. Papantonis. “Principles and Practice of Explainable Machine Learning”. In: *Frontiers in Big Data* 4 (2021), p. 688969. ISSN: 2624-909X. DOI: 10.3389/fdata.2021.688969.

[22] B. Bérard et al. “SMV — Symbolic Model Checking”. In: *Systems and Software Verification: Model-Checking Techniques and Tools*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 131–138. ISBN: 978-3-662-04558-9. DOI: 10.1007/978-3-662-04558-9_12.

[23] D. Berardi and G. De Giacomo. *Planning Via Model Checking: Some Experimental Results*. [Online]. 2000. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=4564925606b35693114d271d187de29a8d56a0c9 (visited on 02/09/2025).

[24] J. Borenstein and Y. Koren. “Real-time obstacle avoidance for fast mobile robots”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 19.5 (1989), pp. 1179–1187. DOI: 10.1109/21.44033.

[25] J. Borenstein and Y. Koren. “The vector field histogram-fast obstacle avoidance for mobile robots”. In: *IEEE Transactions on Robotics and Automation* 7.3 (1991), pp. 278–288. DOI: 10.1109/70.88137.

[26] S. Bouraine, A. Bougouffa, and O. Azouaoui. “Particle swarm optimization for solving a scan-matching problem based on the normal distributions transform”. In: *Evolutionary Intelligence* 15.1 (Mar. 2022), pp. 683–694. ISSN: 18645917. DOI: 10.1007/S12065-020-00545-Y/FIGURES/8.

[27] S. Bouraine, T. Fraichard, and H. Salhi. “Provably safe navigation for mobile robots with limited field-of-views in dynamic environments”. In: *Autonomous Robots* 32.3 (Apr. 2012), pp. 267–283. ISSN: 1573-7527. DOI: 10.1007/s10514-011-9258-8.

[28] V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. Cambridge, Massachussets: MIT Press, 1986.

[29] C. Brewitt et al. “GRIT: Fast, Interpretable, and Verifiable Goal Recognition with Learned Decision Trees for Autonomous Driving”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 1023–1030. DOI: 10.1109/IROS51168.2021.9636279.

[30] T. B. Brown et al. “Language Models are Few-Shot Learners”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. ISBN: 9781713829546. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[31] B. G. Buchanan and R. G. Smith. “Fundamentals of Expert Systems”. In: *Annual Review of Computer Science* 3.1 (1988), pp. 23–58. ISSN: 8756-7016, 8756-7016. DOI: 10.1146/annurev.cs.03.060188.000323.

[32] A. Buliga et al. “Generating counterfactual explanations under temporal constraints”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 15. 2025, pp. 15622–15631. DOI: 10.1609/icaps.v30i1.6740.

[33] P. Chakravarty et al. “CNN-based single image obstacle avoidance on a quadrotor”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 6369–6374. DOI: 10.1109/ICRA.2017.7989752.

[34] C. Chandler et al. “Model Checking for Closed-Loop Robot Reactive Planning”. In: *Electronic Proceedings in Theoretical Computer Science*. Vol. 395. 2023, pp. 77–94. DOI: 10.4204/EPTCS.395.6.

[35] J. Chen, S. E. Li, and M. Tomizuka. “Interpretable End-to-End Urban Autonomous Driving With Latent Deep Reinforcement Learning”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.6 (2022), pp. 5068–5078. DOI: 10.1109/TITS.2020.3046646.

[36] J. Chen, B. Yuan, and M. Tomizuka. “Model-free Deep Reinforcement Learning for Urban Autonomous Driving”. In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. 2019, pp. 2765–2771. DOI: 10.1109/ITSC.2019.8917306.

[37] L. Chen, X. Hu, et al. “Conditional DQN-Based Motion Planning With Fuzzy Logic for Autonomous Driving”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.4 (2022), pp. 2966–2977. ISSN: 1524-9050, 1558-0016. DOI: 10.1109/TITS.2020.3025671.

[38] L. Chen, P. Wu, et al. “End-to-End Autonomous Driving: Challenges and Frontiers”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46.12 (2024), pp. 10164–10183. ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2024.3435937.

[39] P. Chen, J. Pei, et al. “A deep reinforcement learning based method for real-time path planning and dynamic obstacle avoidance”. In: *Neurocomputing* 497 (2022), pp. 64–75. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2022.05.006.

[40] P. S. Chib and P. Singh. “Recent Advancements in End-to-End Autonomous Driving Using Deep Learning: A Survey”. In: *IEEE Transactions on Intelligent Vehicles* 9.1 (2024), pp. 103–118. DOI: 10.1109/TIV.2023.3318070.

[41] J. K. Choi and Y. G. Ji. “Investigating the Importance of Trust on Adopting an Autonomous Vehicle”. In: *International Journal of Human-Computer Interaction* 31.10 (2015), pp. 692–702. ISSN: 1044-7318, 1532-7590. DOI: 10.1080/10447318.2015.1070549.

[42] A. Chougule et al. “A Comprehensive Review on Limitations of Autonomous Driving and Its Impact on Accidents and Collisions”. In: *IEEE Open Journal of Vehicular Technology* 5 (2024), pp. 142–161. ISSN: 2644-1330. DOI: 10.1109/OJVT.2023.3335180.

[43] E. M. Clarke, E. A. Emerson, et al. “Symmetry reductions in model checking”. In: *Computer Aided Verification*. Ed. by A. J. Hu and M. Y. Vardi. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 147–158. ISBN: 978-3-540-69339-0.

[44] E. M. Clarke, T. A. Henzinger, et al., eds. *Handbook of Model Checking*. Springer International Publishing, 2018. ISBN: 978-3-319-10574-1 978-3-319-10575-8. DOI: 10.1007/978-3-319-10575-8.

[45] L. Claussmann et al. “A Review of Motion Planning for Highway Autonomous Driving”. In: *IEEE Transactions on Intelligent Transportation Systems* 21.5 (2020), pp. 1826–1848. ISSN: 1524-9050, 1558-0016. DOI: 10.1109/TITS.2019.2913998.

[46] European Commission. *Artificial Intelligence Act*. [Online]. 2024. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32024R1689 (visited on 01/14/2025).

[47] Z. Cui et al. “An interpretation framework for autonomous vehicles decision-making via SHAP and RF”. In: *2022 6th CAA International Conference on Vehicular Control and Intelligence (CVCI)*. IEEE, 2022, pp. 1–7. ISBN: 978-1-66545-374-5. DOI: 10.1109/CVCI56766.2022.9964561.

[48] S. Daronnat et al. “Inferring Trust From Users’ Behaviours; Agents’ Predictability Positively Affects Trust, Task Performance and Cognitive Load in Human-Agent Real-Time Collaboration”. In: *Frontiers in Robotics and AI* 8 (2021), p. 642201. ISSN: 2296-9144. DOI: 10.3389/frobt.2021.642201.

[49] F. D. Davis, R. P. Bagozzi, and P. R. Warshaw. "User acceptance of computer technology: A comparison of two theoretical models". In: *Management Science* 35.8 (1989), pp. 982–1003. DOI: 10.1287/mnsc.35.8.982.

[50] I. G. Daza et al. "Sim-to-real transfer and reality gap modeling in model predictive control for autonomous driving". In: *Applied Intelligence* 53.10 (2023), pp. 12719–12735. ISSN: 0924-669X, 1573-7497. DOI: 10.1007/s10489-022-04148-1.

[51] G. De Nicolao, A. Ferrara, and L. Giacomini. "Onboard Sensor-Based Collision Risk Assessment to Improve Pedestrians' Safety". In: *IEEE Transactions on Vehicular Technology* 56.5 (2007), pp. 2405–2413. ISSN: 0018-9545, 1939-9359. DOI: 10.1109/TVT.2007.899209.

[52] L. A. Dennis and B. Farwer. "Gwendolen: A BDI language for verifiable agents". In: *Proceedings of the AISB 2008 Symposium on Logic and the Simulation of Interaction and Reasoning, Society for the Study of Artificial Intelligence and Simulation of Behaviour*. University of Aberdeen, 2012, pp. 16–23. URL: https://personalpages.manchester.ac.uk/staff/louise.dennis/pubs/gwendolen.pdf.

[53] L. A. Dennis, M. Fisher, et al. "Model checking agent programming languages". In: *Automated Software Engineering* 19.1 (2012), pp. 5–63. ISSN: 0928-8910, 1573-7535. DOI: 10.1007/s10515-011-0088-x.

[54] A. Desai, T. Dreossi, and S. A. Seshia. "Combining Model Checking and Runtime Verification for Safe Robotics". In: *Runtime Verification*. Ed. by S. Lahiri and Giles Reger. Cham: Springer International Publishing, 2017, pp. 172–189. ISBN: 978-3-319-67531-2.

[55] E. W. Dijkstra. *A Discipline of Programming*. London: Prentice-Hall, 1976.

[56] E. W. Dijkstra. "A note on two problems in connexion with graphs". In: *Edsger Wybe Dijkstra: His Life, Work, and Legacy*. 2022, pp. 287–290. ISBN: 978-1-4503-9773-5. DOI: 10.1145/3544585.

[57] M. T. Dzindolet et al. "The role of trust in automation reliance". In: *International Journal of Human-Computer Studies* 58.6 (2003), pp. 697–718. ISSN: 10715819. DOI: 10.1016/S1071-5819(03)00038-7.

[58] European Commission. Joint Research Centre. *Trustworthy autonomous vehicles: assessment criteria for trustworthy AI in the autonomous driving domain*. EUR 30942 EN. LU: Publications Office, 2021. URL: https://data.europa.eu/doi/10.2760/120385 (visited on 11/12/2024).

[59] M. Everett, Y. F. Chen, and J. P. How. "Collision Avoidance in Pedestrian-Rich Environments With Deep Reinforcement Learning". In: *IEEE Access* 9 (2021), pp. 10357–10377. DOI: 10.1109/ACCESS.2021.3050338.

[60] G.E. Fainekos, H. Kress-Gazit, and G.J. Pappas. "Temporal Logic Motion Planning for Mobile Robots". In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. 2005, pp. 2020–2025. DOI: 10.1109/ROBOT.2005.1570410.

[61] M. Faisal et al. "Fuzzy Logic Navigation and Obstacle Avoidance by a Mobile Robot in an Unknown Dynamic Environment". In: *International Journal of Advanced Robotic Systems* 10.1 (2013), p. 37. DOI: 10.5772/54427.

[62] S. Feraco et al. "A local trajectory planning and control method for autonomous vehicles based on the RRT algorithm". In: *2020 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE)*. IEEE, 2020, pp. 1–6. ISBN: 978-88-87237-49-8. DOI: 10.23919/AEITAUTOMOTIVE50086.2020.9307439.

[63] A. Ferrando et al. "Verifying and Validating Autonomous Systems: Towards an Integrated Approach". In: *Runtime Verification*. Ed. by C. Colombo and M. Leucker. Springer International Publishing, 2018, pp. 263–281. ISBN: 978-3-030-03769-7. DOI: 10.1007/978-3-030-03769-7_15.

[64] M. Fisher, L. Dennis, and M. Webster. "Verifying autonomous systems". In: *Communications of the ACM* 56.9 (2013), pp. 84–93. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/2494558.

[65] F. Foukalas. "A Survey of Artificial Neural Network Computing Systems". In: *Cognitive Computation* 17.1 (2025), p. 4. ISSN: 1866-9956, 1866-9964. DOI: 10.1007/s12559-024-10383-0.

[66] D. Fox, W. Burgard, and S. Thrun. "The dynamic window approach to collision avoidance". In: *IEEE Robotics & Automation Magazine* 4.1 (1997), pp. 23–33. DOI: 10.1109/100.580977.

[67] D. Fraser et al. "Collaborative models for autonomous systems controller synthesis". In: *Formal Aspects of Computing* 32 (2020), pp. 157–186. DOI: 10.1109/TCST.2006.872519.

[68] A. Gambi, T. Huynh, and G. Fraser. "Generating effective test cases for self-driving cars from police reports". In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2019, pp. 257–267. ISBN: 978-1-4503-5572-8. DOI: 10.1145/3338906.3338942.

[69] F. Gao et al. “Automatic Virtual Test Technology for Intelligent Driving Systems Considering Both Coverage and Efficiency”. In: *IEEE Transactions on Vehicular Technology* 69.12 (2020), pp. 14365–14376. ISSN: 0018-9545, 1939-9359. DOI: 10.1109/TVT.2020.3033565.

[70] R. Gerth. *Concise Promela Reference*. [Online]. (visited on 08/01/2024). 1997. URL: https://spinroot.com/spin/Man/Quick.html (visited on 08/01/2025).

[71] F. Giunchiglia and P. Traverso. “Planning as Model Checking”. In: *Recent Advances in AI Planning*. Ed. by S. Biundo and M. Fox. Vol. 1809. Series Title: Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000, pp. 1–20. ISBN: 978-3-540-67866-3, 978-3-540-44657-6. DOI: 10.1007/10720246_1.

[72] D. Gonzalez et al. “A Review of Motion Planning Techniques for Automated Vehicles”. In: *IEEE Transactions on Intelligent Transportation Systems* 17.4 (2016), pp. 1135–1145. ISSN: 1524-9050, 1558-0016. DOI: 10.1109/TITS.2015.2498841.

[73] R. Gu, E. Baranov, et al. “Synthesis and Verification of Mission Plans for Multiple Autonomous Agents under Complex Road Conditions”. In: *ACM Transactions on Software Engineering and Methodology* 33.7 (2024), pp. 1–46. ISSN: 1049-331X, 1557-7392. DOI: 10.1145/3672445.

[74] R. Gu, C. Seceleanu, et al. “Model Checking Collision Avoidance of Nonlinear Autonomous Vehicles”. In: *Formal Methods*. Ed. by M. Huisman, C. Păsăreanu, and Naijun Zhan. Cham: Springer International Publishing, 2021, pp. 676–694. ISBN: 978-3-030-90870-6.

[75] H. Guo, F. Wu, et al. “Recent Trends in Task and Motion Planning for Robotics: A Survey”. In: *ACM Computing Surveys* 55.13 (Dec. 31, 2023), pp. 1–36. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3583136.

[76] M. Guo, K. H. Johansson, and D. V. Dimarogonas. “Revising Motion Planning Under Linear Temporal Logic Specifications in Partially Known Workspaces”. In: *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 5025–5032. ISBN: 978-1-4673-5643-5 978-1-4673-5641-1. DOI: 10.1109/ICRA.2013.6631295.

[77] B. Gyevnar, N. Ferguson, and B. Schafer. “Bridging the Transparency Gap: What Can Explainable AI Learn from the AI Act?” In: *Frontiers in Artificial Intelligence and Applications*. Ed. by K. Gal et al. IOS Press, 2023. ISBN: 978-1-64368-436-9 978-1-64368-437-6. DOI: 10.3233/FAIA230367.

[78] B. Gyevnar, C. Wang, et al. “Causal Explanations for Sequential Decision-Making in Multi-Agent Systems”. In: *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2024, pp. 771–779. ISBN: 9798400704864. DOI: 10.5555/3635637.3662930s.

[79] T. Ha et al. “Effects of explanation types and perceived risk on trust in autonomous vehicles”. In: *Transportation Research Part F: Traffic Psychology and Behaviour* 73 (2020), pp. 271–280. ISSN: 13698478. DOI: 10.1016/j.trf.2020.06.021.

[80] J. Hamilton et al. “Towards Adaptive Planning of Assistive-care Robot Tasks”. In: *Electronic Proceedings in Theoretical Computer Science*. 2022, pp. 175–183. DOI: 10.4204/EPTCS.371.12.

[81] J. P. Hanna et al. “Interpretable Goal Recognition in the Presence of Occluded Factors for Autonomous Vehicles”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 7044–7051. DOI: 10.1109/IROS51168.2021.9635903.

[82] P. Hart, N. Nilsson, and B. Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. ISSN: 0536-1567. DOI: 10.1109/TSSC.1968.300136.

[83] R. Häuslschmid et al. “SupportingTrust in Autonomous Driving”. In: *Proceedings of the 22nd International Conference on Intelligent User Interfaces*. ACM, 2017, pp. 319–329. ISBN: 978-1-4503-4348-0. DOI: 10.1145/3025171.3025198.

[84] K. Havelund, M. Lowry, and J. Penix. “Formal Analysis of a Space-Craft Controller Using SPIN.” In: *IEEE Transactions on Software Engineering* 27 (2001), pp. 749–765. DOI: 10.1109/32.940728.

[85] T. A. Henzinger, P. Ho, and H. Wong-Toi. “HyTech: A model checker for hybrid systems”. In: *Computer Aided Verification*. Ed. by O. Grumberg. Springer Berlin Heidelberg, 1997, pp. 460–463. ISBN: 978-3-540-69195-2, 978-3-540-63166-8. DOI: 10.1007/3-540-63166-6_48.

[86] S. Hergeth et al. “Keep Your Scanners Peeled: Gaze Behavior as a Measure of Automation Trust During Highly Automated Driving”. In: *Human Factors: The Journal of the Human Factors and Ergonomics Society* 58.3 (2016), pp. 509–519. ISSN: 0018-7208, 1547-8181. DOI: 10.1177/0018720815625744.

[87] K. A. Hoff and M. Bashir. “Trust in Automation: Integrating Empirical Evidence on Factors That Influence Trust”. In: *Human Factors: The Journal of the Human Factors and Ergonomics Society* 57.3 (2015), pp. 407–434. ISSN: 0018-7208, 1547-8181. DOI: 10.1177/0018720814547570.

[88] J. Hoffmann et al. “Let’s Learn Their Language? A Case for Planning with Automata-Network Languages from Model Checking”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.9 (2020), pp. 13569–13575. ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v34i09.7083.

[89] C. Holland and R. Hill. “The effect of age, gender and driver status on pedestrians’ intentions to cross the road in risky situations”. In: *Accident Analysis and Prevention* 39.2 (2007), pp. 224–237. ISSN: 0001-4575. DOI: 10.1016/j.aap.2006.07.003 .

[90] G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. 1st. Addison-Wesley Professional, 2011. ISBN: 978-0-321-77371-5.

[91] X. Hu et al. “How Simulation Helps Autonomous Driving: A Survey of Sim2real, Digital Twins, and Parallel Intelligence”. In: *IEEE Transactions on Intelligent Vehicles* 9.1 (2024), pp. 593–612. DOI: 10.1109/TIV.2023.3312777.

[92] C. Huang et al. “Personalized Trajectory Planning and Control of Lane-Change Maneuvers for Autonomous Driving”. In: *IEEE Transactions on Vehicular Technology* 70.6 (2021), pp. 5511–5523. ISSN: 0018-9545, 1939-9359. DOI: 10.1109/TVT.2021.3076473.

[93] L. M. Hulse, H. Xie, and E. R. Galea. “Perceptions of autonomous vehicles: Relationships with road users, risk, gender and age”. In: *Safety Science* 102 (2018), pp. 1–13. ISSN: 09257535. DOI: 10.1016/j.ssci.2017.10.001.

[94] V. Ilkova and A. Ilka. “Legal aspects of autonomous vehicles — An overview”. In: *2017 21st International Conference on Process Control (PC)*. Strbske Pleso, Slovakia: IEEE, June 2017, pp. 428–433. ISBN: 978-1-5386-4011-1. DOI: 10.1109/PC.2017.7976252.

[95] M. Ingram et al. “Calibrating trust toward an autonomous image classifier.” In: *Technology, Mind, and Behavior* 2.1 (2021). ISSN: 2689-0208. DOI: 10.1037/tmb0000032.

[96] S. Jesenski et al. “Generation of Scenes in Intersections for the Validation of Highly Automated Driving Functions”. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019, pp. 502–509. ISBN: 978-1-72810-560-4. DOI: 10.1109/IVS.2019.8813776.

[97] O. P. John, S. Srivastava, et al. “The Big-Five trait taxonomy: History, measurement, and theoretical perspectives”. In: *Handbook of Personality: Theory and Research*. Ed. by L. A. Pervin and O. P. John. 2nd. Guilford Press, 1999, pp. 102–138. ISBN: 9781462544950.

[98] N. Kalra and S. M. Paddock. "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" In: *Transportation Research Part A: Policy and Practice* 94 (2016), pp. 182–193. ISSN: 09658564. DOI: 10.1016/j.tra.2016.09.010.

[99] M. Kamali, S. Linker, and M. Fisher. "Modular Verification of Vehicle Platooning with Respect to Decisions, Space and Time". In: *Formal Techniques for Safety-Critical Systems*. Ed. by C. Artho and P. C. Ölveczky. Springer International Publishing, 2019, pp. 18–36. ISBN: 978-3-030-12988-0. DOI: 10.1007/978-3-030-12988-0_2.

[100] I. Kamon, E. Rivlin, and E. Rimon. "A new range-sensor based globally convergent navigation algorithm for mobile robots". In: *Proceedings of IEEE International Conference on Robotics and Automation*. Vol. 1. 1996, 429–435 vol.1. DOI: 10.1109/ROBOT.1996.503814.

[101] S. Karaman and E. Frazzoli. "Sampling-based motion planning with deterministic μ-calculus specifications". In: *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. 2009, pp. 2222–2229. DOI: 10.1109/CDC.2009.5400278.

[102] D. Kasenberg, R. Thielstrom, and M. Scheutz. "Generating explanations for temporal logic planner decisions". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 30. 2020, pp. 449–458. DOI: 10.1609/icaps.v30i1.6740.

[103] D. Katare et al. "A Survey on Approximate Edge AI for Energy Efficient Autonomous Driving Services". In: *IEEE Communications Surveys & Tutorials* 25.4 (2023), pp. 2714–2754. ISSN: 1553-877X, 2373-745X. DOI: 10.1109/COMST.2023.3302474.

[104] S. Kato et al. "An Open Approach to Autonomous Vehicles". In: *IEEE Micro* 35.6 (2015), pp. 60–68. DOI: 10.1109/MM.2015.133.

[105] K. Kaur and G. Rampersad. "Trust in driverless cars: Investigating key factors influencing the adoption of driverless cars". In: *Journal of Engineering and Technology Management* 48 (2018), pp. 87–96. ISSN: 09234748. DOI: 10.1016/j.jengtecman.2018.04.006.

[106] O. Khatib. "Real-time obstacle avoidance for manipulators and mobile robots". In: *Proceedings. 1985 IEEE International Conference on Robotics and Automation*. Vol. 2. 1985, pp. 500–505. DOI: 10.1109/ROBOT.1985.1087247.

[107] J. M. Kizza. "Software Issues: Risks and Liabilities". In: *Ethical and Secure Computing*. Series Title: Undergraduate Topics in Computer Science. Springer International Publishing, 2019, pp. 149–176. ISBN: 978-3-030-03936-3 978-3-030-03937-0. DOI: 10.1007/978-3-030-03937-0_7.

[108] J. Koo et al. “Why did my car just do that? Explaining semi-autonomous driving actions to improve driver understanding, trust, and performance”. In: *International Journal on Interactive Design and Manufacturing (IJIDeM)* 9.4 (2015), pp. 269–275. ISSN: 1955-2513, 1955-2505. DOI: 10.1007/s12008-014-0227-2.

[109] Y. Koren and J. Borenstein. “Potential field methods and their inherent limitations for mobile robot navigation”. In: *Proceedings. 1991 IEEE International Conference on Robotics and Automation*. 1991, 1398–1404 vol.2. DOI: 10.1109/ROBOT.1991.131810.

[110] S. Koul and A. Eydgahi. “Utilizing Technology Acceptance Model (TAM) for driverless car technology Adoption”. In: *Journal of technology management & innovation* 13.4 (2018), pp. 37–46. ISSN: 0718-2724. DOI: 10.4067/S0718-27242018000400037.

[111] KPMG. *Connected and Autonomous Vehicles – The UK Economic Opportunity*. [Online]. 2015. URL: https://assets.kpmg.com/content/dam/kpmg/pdf/2015/04/connected-and-autonomous-vehicles.pdf (visited on 02/19/2025).

[112] A. Kuznietsov et al. “Explainable AI for Safe and Trustworthy Autonomous Driving: A Systematic Review”. In: *IEEE Transactions on Intelligent Transportation Systems* 25.12 (2024), pp. 19342–19364. ISSN: 1524-9050, 1558-0016. DOI: 10.1109/TITS.2024.3474469.

[113] M. Kyriakidis, R. Happee, and J. De Winter. “Public opinion on automated driving: Results of an international questionnaire among 5000 respondents”. In: *Transportation Research Part F: Traffic Psychology and Behaviour* 32 (2015), pp. 127–140. ISSN: 13698478. DOI: 10.1016/j.trf.2015.04.014.

[114] M. Lahijanian et al. “Iterative Temporal Planning in Uncertain Environments With Partial Satisfaction Guarantees”. In: *IEEE Transactions on Robotics* 32.3 (2016), pp. 583–599. DOI: 10.1109/TRO.2016.2544339.

[115] K. G. Larsen, P. Pettersson, and W. Yi. “Uppaal in a nutshell”. In: *International Journal on Software Tools for Technology Transfer* 1.1 (1997), pp. 134–152. ISSN: 1433-2779. DOI: 10.1007/s100090050010.

[116] S. M. LaValle and J. J. Kuffner Jr. “Randomized Kinodynamic Planning”. In: *The International Journal of Robotics Research* 20.5 (2001), pp. 378–400. DOI: 10.1177/02783640122067453.

[117] Y. LeCun. *A Path Towards Autonomous Machine Intelligence | OpenReview*. Publication Title: OpenReview.net [Online]. 2022. URL: https://openreview.net/forum?id=BZ5a1r-kVsf (visited on 01/10/2025).

[118] J. D. Lee and K. A. See. “Trust in Automation: Designing for Appropriate Reliance”. In: *Human Factors* 46.1 (2004), pp. 50–80. DOI: 10.1518/hfes.46.1.50_30392.

[119] J. J. Leonard and H. F. Durrant-Whyte. “Simultaneous map building and localization for an autonomous mobile robot”. In: *Proceedings IROS ’91:IEEE/RSJ International Workshop on Intelligent Robots and Systems ’91*. 1991, pp. 1442–1447. DOI: 10.1109/IROS.1991.174711.

[120] B. Li, Y. Wang, et al. “GP-SLAM: laser-based SLAM approach based on regionalized Gaussian process map reconstruction”. In: *Autonomous Robots* 44.6 (July 2020), pp. 947–967. ISSN: 15737527. DOI: 10.1007/S10514-020-09906-Z/FIGURES/26.

[121] M. Li, H. Sun, Y. Huang, et al. “SVCE: Shapley Value Guided Counterfactual Explanation for Machine Learning-Based Autonomous Driving”. In: *IEEE Transactions on Intelligent Transportation Systems* 25.10 (2024), pp. 14905–14916. ISSN: 1524-9050, 1558-0016. DOI: 10.1109/TITS.2024.3393634.

[122] S. Li, D. Park, et al. “Reactive Task and Motion Planning under Temporal Logic Specifications”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 12618–12624. DOI: 10.1109/ICRA48506.2021.9561807.

[123] X. Li, Z. Sun, D. Cao, et al. “Real-time trajectory planning for autonomous urban driving: Framework, algorithms, and verifications”. In: *IEEE/ASME Transactions on Mechatronics* 21.2 (2016), pp. 740–753. DOI: 10.1109/TMECH.2015.2493980.

[124] Q. Lin, S. Mitsch, et al. “Safe and Resilient Practical Waypoint-Following for Autonomous Vehicles”. In: *IEEE Control Systems Letters* 6 (2022), pp. 1574–1579. DOI: 10.1109/LCSYS.2021.3125717.

[125] S. Lin, Y. Zhang, et al. “The Architectural Implications of Autonomous Driving: Constraints and Acceleration”. In: *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2018, pp. 751–766. ISBN: 978-1-4503-4911-6. DOI: 10.1145/3173162.3173191.

[126] L. Liu, S. Lu, et al. “Computing Systems for Autonomous Driving: State of the Art and Challenges”. In: *IEEE Internet of Things Journal* 8.8 (2021), pp. 6469–6486. DOI: 10.1109/JIOT.2020.3043716.

[127] S. Liu, L. Liu, et al. “Edge Computing for Autonomous Driving: Opportunities and Challenges”. In: *Proceedings of the IEEE* 107.8 (2019), pp. 1697–1716. ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2019.2915983.

[128] S. B. Liu, H. Roehm, et al. “Provably safe motion of mobile robots in human environments”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 1351–1357. DOI: 10.1109/IROS.2017.8202313.

[129] D. Louise et al. “Practical verification of decision-making in agent-based autonomous systems”. In: *Automated Software Engineering* 23.3 (2016), pp. 305–359. DOI: 10.1007/s10515-014-0168-9.

[130] Y. Lu et al. “Availability analysis of satellite positioning systems for avaiation using the Prism model checker”. In: *Proceedings of the 17th International Conference on Computational Science and Engineering (CSE 2014)*. 2014, pp. 704–713. DOI: 10.1109/CSE.2014.148.

[131] M. Luckcuck et al. “A Summary of Formal Specification and Verification of Autonomous Robotic Systems”. In: *Integrated Formal Methods*. Ed. by W. Ahrendt, T. Tarifa, and S. Lizeth. Springer International Publishing, 2019, pp. 538–541. ISBN: 978-3-030-34968-4. DOI: 10.1007/978-3-030-34968-4_33.

[132] M. Luckcuck et al. “Formal Specification and Verification of Autonomous Robotic Systems: A Survey”. In: *ACM Computing Surveys* 52.5 (Sept. 30, 2020), pp. 1–41. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3342355.

[133] V. Lumelsky and A. Stepanov. “Dynamic path planning for a mobile automaton with limited information on the environment”. In: *IEEE Transactions on Automatic Control* 31.11 (1986), pp. 1058–1063. DOI: 10.1109/TAC.1986.1104175.

[134] X. Luo, Y. Kantaros, and M. M. Zavlanos. “An Abstraction-Free Method for Multirobot Temporal Logic Optimal Control Synthesis”. In: *IEEE Transactions on Robotics* 37.5 (2021), pp. 1487–1507. DOI: 10.1109/TRO.2021.3061983.

[135] Y. Ma et al. “Lane Change Analysis and Prediction Using Mean Impact Value Method and Logistic Regression Model”. In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2021, pp. 1346–1352. ISBN: 978-1-72819-142-3. DOI: 10.1109/ITSC48978.2021.9564943.

[136] A. V. Malawade, T. Mortlock, and M. A. A. Faruque. “EcoFusion: energy-aware adaptive sensor fusion for efficient autonomous vehicle perception”. In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. DAC ’22. San Francisco, California: Association for Computing Machinery, 2022, pp. 481–486. ISBN: 9781450391429. DOI: 10.1145/3489517.3530489.

[137] R. C. Mayer, J. H. Davis, and F. D. Schoorman. “An Integrative Model of Organizational Trust”. In: *The Academy of Management Review* 20.3 (1995), pp. 709–734. ISSN: 03637425. DOI: 10.2307/258792.

[138] R. McAllister et al. “Concrete Problems for Autonomous Vehicle Safety: Advantages of Bayesian Deep Learning”. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. Twenty-Sixth International Joint Conference on Artificial Intelligence. Melbourne, Australia: International Joint Conferences on Artificial Intelligence Organization, 2017, pp. 4745–4753. ISBN: 978-0-9992411-0-3. DOI: 10.24963/ijcai.2017/661.

[139] K.N. McGuire, G.C.H.E. de Croon, and K. Tuyls. “A comparative study of bug algorithms for robot navigation”. In: *Robotics and Autonomous Systems* 121 (2019), p. 103261. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2019.103261.

[140] J. Michels, A. Saxena, and A. Y. Ng. “High speed obstacle avoidance using monocular vision and reinforcement learning”. In: *Proceedings of the 22nd International Conference on Machine Learning*. ICML ’05. Bonn, Germany: Association for Computing Machinery, 2005, pp. 593–600. ISBN: 1595931805. DOI: 10.1145/1102351.1102426.

[141] A. Mishra et al. “Why? Why not? When? Visual Explanations of Agent Behaviour in Reinforcement Learning ”. In: *2022 IEEE 15th Pacific Visualization Symposium (PacificVis)*. Los Alamitos, CA, USA: IEEE Computer Society, 2022, pp. 111–120. DOI: 10.1109/PacificVis53943.2022.00020.

[142] L. J. Molnar et al. “Understanding trust and acceptance of automated vehicles: An exploratory simulator study of transfer of control between automated and manual driving”. In: *Transportation Research Part F: Traffic Psychology and Behaviour* 58 (2018), pp. 319–328. ISSN: 13698478. DOI: 10.1016/j.trf.2018.06.004.

[143] M. Montemerlo et al. “Junior: The Stanford entry in the Urban Challenge”. In: *J. Field Robot.* 25.9 (Sept. 2008), pp. 569–597. ISSN: 1556-4959. URL: https://dl.acm.org/doi/abs/10.5555/1405647.1405651.

[144] J. Moody, N. Bailey, and J. Zhao. “Public perceptions of autonomous vehicle safety: An international comparison”. In: *Safety Science* 121 (2020), pp. 634–650. ISSN: 09257535. DOI: 10.1016/j.ssci.2019.07.022.

[145] O. Musir and M. Celik. “Visual-based obstacle avoidance method using advanced CNN for mobile robots”. In: *Internet of Things* 31 (2025), p. 101538. ISSN: 2542-6605. DOI: 10.1016/j.iot.2025.101538.

[146] S. Nahavandi et al. “A Comprehensive Review on Autonomous Navigation”. In: *ACM Comput. Surv.* 57.9 (2025). ISSN: 0360-0300. DOI: 10.1145/3727642.

[147] M. S. Nawaz et al. *A survey on theorem provers in formal methods*. 2019. DOI: 10.48550/arXiv.1912.03028. arXiv: 1912.03028.

[148] A. Newell and S. K. Card. "The Prospects for Psychological Science in Human-Computer Interaction". In: *Human–Computer Interaction* 1.3 (1985), pp. 209–242. ISSN: 0737-0024, 1532-7051. DOI: 10.1207/s15327051hci0103_1.

[149] D. Omeiza et al. "Explanations in Autonomous Driving: A Survey". In: *IEEE Transactions on Intelligent Transportation Systems* (2021), pp. 1–21. ISSN: 1524-9050, 1558-0016. DOI: 10.1109/TITS.2021.3122865.

[150] A. Orthey, C. Chamzas, and L. E. Kavraki. "Sampling-Based Motion Planning: A Comparative Review". In: *Annual Review of Control, Robotics, and Autonomous Systems* 7.1 (2024), pp. 285–310. ISSN: 2573-5144. DOI: 10.1146/annurev-control-061623-094742.

[151] K. Othman. "Exploring the implications of autonomous vehicles: a comprehensive review". In: *Innovative Infrastructure Solutions* 7.2 (Apr. 2022), p. 165. ISSN: 2364-4176, 2364-4184. DOI: 10.1007/s41062-022-00763-6.

[152] K. Othman. "Public acceptance and perception of autonomous vehicles: a comprehensive review". In: *AI and Ethics* 1.3 (2021), pp. 355–387. ISSN: 2730-5953, 2730-5961. DOI: 10.1007/s43681-021-00041-8.

[153] Oxford English Dictionary. *trust, n.* In: Oxford University Press [Online], 2024. URL: https://doi.org/10.1093/OED/5777528687 (visited on 01/13/2025).

[154] D. Pagojus et al. "Simulation and Model checking for close to real-time overtaking planning". In: *Electronic Proceedings in Theoretical Computer Science*. 2021, pp. 20–37. DOI: 10.4204/EPTCS.348.2.

[155] R. Parasuraman and V. Riley. "Humans and Automation: Use, Misuse, Disuse, Abuse". In: *Human Factors: The Journal of the Human Factors and Ergonomics Society* 39.2 (1997), pp. 230–253. ISSN: 0018-7208, 1547-8181. DOI: 10.1518/001872097778543886.

[156] R. Parasuraman, T. B. Sheridan, and C. D. Wickens. "A model for types and levels of human interaction with automation". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 30.3 (2000), pp. 286–297. ISSN: 10834427. DOI: 10.1109/3468.844354.

[157] C. Pek et al. "Using online verification to prevent autonomous vehicles from causing accidents". In: *Nature Machine Intelligence* 2.9 (2020), pp. 518–528. ISSN: 2522-5839. DOI: 10.1038/s42256-020-0225-y.

[158] G. Peng et al. "VILO SLAM: Tightly Coupled Binocular Vision–Inertia SLAM Combined with LiDAR". In: *Sensors 2023, Vol. 23, Page 4588* 23.10 (May 2023), p. 4588. ISSN: 1424-8220. DOI: 10.3390/S23104588.

[159] A. Pnueli. “The temporal semantics of concurrent programs”. In: *Theoretical Computer Science* 13.1 (1981), pp. 45–60. DOI: 10.1016/0304-3975(81)90110-9.

[160] B. Porr and F. Wörgötter. “Inside embodiment – what means embodiment to radical constructivists?” In: *Kybernetes* 34.1 (Jan. 1, 2005), pp. 105–117. ISSN: 0368-492X. DOI: 10.1108/03684920510575762.

[161] K. Raats, V. Fors, and S. Pink. “Trusting autonomous vehicles: An interdisciplinary approach”. In: *Transportation Research Interdisciplinary Perspectives* 7 (2020), p. 100201. ISSN: 25901982. DOI: 10.1016/j.trip.2020.100201.

[162] N. Rajabli et al. “Software Verification and Validation of Safe Autonomous Cars: A Systematic Literature Review”. In: *IEEE Access* 9 (2021), pp. 4797–4819. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3048047.

[163] B. Rammstedt and O. P. John. “Measuring personality in one minute or less: A 10-item short version of the Big Five Inventory in English and German”. In: *Journal of Research in Personality* 41.1 (2007), pp. 203–212. ISSN: 00926566. DOI: 10.1016/j.jrp.2006.02.001.

[164] Y. Ran et al. “A Review of 2D Lidar SLAM Research”. In: *Remote Sensing 2025, Vol. 17, Page 1214* 17.7 (Mar. 2025), p. 1214. ISSN: 2072-4292. DOI: 10.3390/RS17071214.

[165] M. Reda et al. “Path planning algorithms in the autonomous driving system: A comprehensive review”. In: *Robotics and Autonomous Systems* 174 (2024), p. 104630. ISSN: 09218890. DOI: 10.1016/j.robot.2024.104630.

[166] P. Reignier. “Fuzzy logic techniques for mobile robot obstacle avoidance”. In: *Robotics and Autonomous Systems* 12.3 (1994), pp. 143–153. ISSN: 0921-8890. DOI: https://doi.org/10.1016/0921-8890(94)90021-3.

[167] L. Ren, W. Wang, and Z. Du. “A new fuzzy intelligent obstacle avoidance control strategy for wheeled mobile robot”. In: *2012 IEEE International Conference on Mechatronics and Automation*. 2012, pp. 1732–1737. DOI: 10.1109/ICMA.2012.6284398.

[168] J. Rix. “From Tools to Teammates: Conceptualizing Humans’ Perception of Machines as Teammates with a Systematic Literature Review”. In: *55th Hawaii International Conference on System Sciences (HICSS)*. ScholarSpace, 2022. DOI: 10.24251/HICSS.2022.048.

[169] T. Rosenbloom. “Crossing at a red light: Behaviour of individuals and groups”. In: *Transportation Research Part F: Traffic Psychology and Behaviour* 12.5 (2009), pp. 389–394. ISSN: 1369-8478. DOI: 10.1016/j.trf.2009.05.002.

[170] S. Ross, G. J. Gordon, and J. A. Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by G. Gordon, D. Dunson, and M. Dudík. Vol. 15. Proceedings of Machine Learning Research. PMLR, 2011, pp. 627–635. URL: https://proceedings.mlr.press/v15/ross11a.html.

[171] C. Rudin. "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead". In: *Nature Machine Intelligence* 1.5 (May 13, 2019), pp. 206–215. ISSN: 2522-5839. DOI: 10.1038/s42256-019-0048-x.

[172] A. Rumpf. *Dijkstra's and A* Search Algorithms for Pathfinding with Obstacles*. [Online]. 2017. URL: https://demonstrations.wolfram.com/DijkstrasAndASearchAlgorithmsForPathfindingWithObstacles/ (visited on 01/21/2025).

[173] S. J. Russell, P. Norvig, and M. Chang. *Artificial intelligence: a modern approach*. English. Fourth edition / Global contributing writers, Ming-Wei Chang [and eight others]. Harlow, United Kingdom: Pearson, 2022. ISBN: 9781292401171, 1292401176;

[174] B. Schoettle and M. Sivak. *Public opinion about self-driving vehicles in China, India, Japan, the US, the UK, and Australia*. Tech. rep. University of Michigan, Ann Arbor, Transportation Research Institute, 2014. URL: https://hdl.handle.net/2027.42/109433.

[175] N. Shankar. "Verification by Abstraction". In: *Formal Methods at the Crossroads. From Panacea to Foundational Support*. Ed. by B. K. Aichernig and T. Maibaum. Red. by Gerhard Goos, Juris Hartmanis, and Jan Van Leeuwen. Vol. 2757. Series Title: Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, pp. 367–380. ISBN: 978-3-540-20527-2 978-3-540-40007-3. DOI: 10.1007/978-3-540-40007-3_23.

[176] E. Shi, T. M. Gasser, et al. "The Principles of Operation Framework: A Comprehensive Classification Concept for Automated Driving Functions". In: *SAE International Journal of Connected and Automated Vehicles* 3.1 (Feb. 18, 2020), pp. 12–03–01–0003. ISSN: 2574-075X. DOI: 10.4271/12-03-01-0003.

[177] W. Shi and L. Liu. *Computing Systems for Autonomous Driving*. Cham: Springer International Publishing, 2021. ISBN: 978-3-030-81563-9 978-3-030-81564-6. DOI: 10.1007/978-3-030-81564-6.

[178] B. Siciliano and O. Khatib, eds. *Springer Handbook of Robotics*. Cham: Springer International Publishing, 2016. ISBN: 978-3-319-32550-7 978-3-319-32552-1. DOI: 10.1007/978-3-319-32552-1.

[179] D. Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature16961.

[180] S. Singh. *Critical reasons for crashes investigated in the national motor vehicle crash causation survey*. (Traffic Safety Facts Crash•Stats. Report No. DOT HS 812 506) [Online]. 2018. URL: https://crashstats.nhtsa.dot.gov.

[181] A. Singletary et al. "Comparative Analysis of Control Barrier Functions and Artificial Potential Fields for Obstacle Avoidance". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 8129–8136. DOI: 10.1109/IROS51168.2021.9636670.

[182] R. C. Smith and P. Cheeseman. "On the Representation and Estimation of Spatial Uncertainty". In: *The International Journal of Robotics Research* 5.4 (1986), pp. 56–68. DOI: 10.1177/027836498600500404.

[183] T. Speith. "A Review of Taxonomies of Explainable Artificial Intelligence (XAI) Methods". In: *2022 ACM Conference on Fairness, Accountability, and Transparency*. FAccT '22: 2022 ACM Conference on Fairness, Accountability, and Transparency. Seoul Republic of Korea: ACM, 2022, pp. 2239–2250. ISBN: 978-1-4503-9352-2. DOI: 10.1145/3531146.3534639.

[184] E. S. Spelke and K. D. Kinzler. "Core knowledge". In: *Developmental Science* 10.1 (2007), pp. 89–96. ISSN: 1363755X. DOI: 10.1111/J.1467-7687.2007.00569.X.

[185] International Organization for Standardization. *Road vehicles – Functional safety*. (ISO Standard No. 26262-1:2018) [Online]. 2018. URL: https://www.iso.org/standard/68383.html (visited on 01/07/2025).

[186] H. Sun, W. Zhang, R. Yu, et al. "Motion Planning for Mobile Robots—Focusing on Deep Reinforcement Learning: A Systematic Review". In: *IEEE Access* 9 (2021), pp. 69061–69081. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3076530.

[187] J. Sun, H. Zhang, H. Zhou, et al. "Scenario-Based Test Automation for Highly Automated Vehicles: A Review and Paving the Way for Systematic Safety Assurance". In: *IEEE Transactions on Intelligent Transportation Systems* 23.9 (2022), pp. 14088–14103. ISSN: 1524-9050, 1558-0016. DOI: 10.1109/TITS.2021.3136353.

[188] E. Tabassi. *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*. NIST AI 100-1. Gaithersburg, MD: National Institute of Standards and Technology (U.S.), 2023. DOI: 10.6028/NIST.AI.100-1.

[189] H. Tan, X. Zhao, and J. Yang. "Exploring the influence of anxiety, pleasure and subjective knowledge on public acceptance of fully autonomous vehicles". In: *Computers in Human Behavior* 131 (2022), p. 107187. ISSN: 07475632. DOI: 10.1016/j.chb.2022.107187.

[190] S. Tang. *Generating a Bezier Curve by the de Casteljau Algorithm*. [Online]. 2014. URL: https://demonstrations.wolfram.com/GeneratingABezierCurveByTheDeCasteljauAlgorithm/ (visited on 01/21/2025).

[191] S. Teng et al. "Motion Planning for Autonomous Driving: The State of the Art and Future Perspectives". In: *IEEE Transactions on Intelligent Vehicles* (2023), pp. 1–21. ISSN: 2379-8904, 2379-8858. DOI: 10.1109/TIV.2023.3274536.

[192] S. Thorpe, D. Fize, and C. Marlot. "Speed of processing in the human visual system". In: *Nature* 381 (1996), pp. 520–522. DOI: 10.1038/381520a0.

[193] G. Tuncay et al. "Resolving the Predicament of Android Custom Permissions". In: *Network and Distributed System Security Symposium*. 2018, pp. 1–15. DOI: 10.14722/ndss.2018.23221.

[194] C. Turner and R. McClure. "Age and gender differences in risk-taking behaviour as an explanation for high incidence of motor vehicle crashes as a driver in young males." In: *Injury control and safety promotion* 10.3 (2003), pp. 123–130. DOI: 10.1076/icsp.10.3.123.14560.

[195] I. Ulrich and J. Borenstein. "VFH+: reliable obstacle avoidance for fast mobile robots". In: *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*. Vol. 2. 1998, 1572–1577 vol.2. DOI: 10.1109/ROBOT.1998.677362.

[196] A. M. Varghese and V. R. Jisha. "Motion Planning and Control of an Autonomous Mobile Robot". In: *2018 International CET Conference on Control, Communication, and Computing (IC4)*. 2018, pp. 17–21. DOI: 10.1109/CETIC4.2018.8530879.

[197] V. Venkatesh et al. "User Acceptance of Information Technology: Toward a Unified View". In: *MIS Quarterly* 27.3 (2003), pp. 425–478. ISSN: 02767783. DOI: 10.2307/30036540.

[198] P. de Vries, C. Midden, and D. Bouwhuis. "The effects of errors on system trust, self-confidence, and the allocation of control in route planning". In: *International Journal of Human-Computer Studies* 58.6 (2003), pp. 719–735. ISSN: 1071-5819. DOI: 10.1016/S1071-5819(03)00039-9.

[199] F. Walker et al. "Trust in automated vehicles: constructs, psychological processes, and assessment". In: *Frontiers in Psychology* 14 (2023), p. 1279271. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2023.1279271.

[200] D. Walter, H. Täubig, and C. Lüth. "Experiences in Applying Formal Verification in Robotics". In: *Computer Safety, Reliability, and Security*. Ed. by Erwin Schoitsch. Red. by D. Hutchison et al. Vol. 6351. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 347–360. ISBN: 978-3-642-15650-2 978-3-642-15651-9. DOI: 10.1007/978-3-642-15651-9_26.

[201] L. Wang and F. Cai. "Reliability analysis for flight control systems using probabilistic model checking". In: *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS* 2017-Novem (2017), pp. 161–164. DOI: 10.1109/RAM.2017.7889773.

[202] W. Wei et al. "Enhancing Solid State LiDAR Mapping with a 2D Spinning LiDAR in Urban Scenario SLAM on Ground Vehicles". In: *Sensors 2021, Vol. 21, Page 1773* 21.5 (Mar. 2021), p. 1773. ISSN: 1424-8220. DOI: 10.3390/S21051773.

[203] M. Weißmann et al. "Model Checking Industrial Robot Systems". In: *Model checking software (SPIN 2011)*. Vol. 6823. Springer Berlin Heidelberg, July 2011, pp. 161–176. ISBN: 978-3-642-22305-1. DOI: 10.1007/978-3-642-22306-8_11.

[204] J. M. Wing. "A specifier's introduction to formal methods". In: *Computer* 23.9 (1990), pp. 8–22. ISSN: 0018-9162. DOI: 10.1109/2.58215.

[205] E. Winter. "The shapley value". In: *Handbook of Game Theory with Economic Applications*. Vol. 3. Elsevier, 2002, pp. 2025–2054. ISBN: 978-0-444-89428-1. DOI: 10.1016/S1574-0005(02)03016-3.

[206] Z. Xu et al. "What drives people to accept automated vehicles? Findings from a field experiment". In: *Transportation Research Part C: Emerging Technologies* 95 (2018), pp. 320–334. ISSN: 0968090X. DOI: 10.1016/j.trc.2018.07.024.

[207] Y. Yao and E. Atkins. "The Smart Black Box: A Value-Driven High-Bandwidth Automotive Event Data Recorder". In: *IEEE Transactions on Intelligent Transportation Systems* 22.3 (2021), pp. 1484–1496. DOI: 10.1109/TITS.2020.2971385.

[208] E. Yurtsever et al. "A Survey of Autonomous Driving: *Common Practices and Emerging Technologies*". In: *IEEE Access* 8 (2020), pp. 58443–58469. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2983149.

[209] B. Zhang, Z. Peng, et al. "2DLIW-SLAM:2D LiDAR-inertial-wheel odometry with real-time loop closure". In: *Measurement Science and Technology* 35.7 (Apr. 2024), p. 075205. ISSN: 0957-0233. DOI: 10.1088/1361-6501/AD3EA3.

[210] T. Zhang, D. Tao, et al. "The roles of initial trust and perceived risk in public's acceptance of automated vehicles". In: *Transportation Research Part C: Emerging Technologies* 98 (2019), pp. 207–220. ISSN: 0968090X. DOI: 10.1016/j.trc.2018.11.018.

[211] X. Zhang and V. S. Sheng. *Neuro-Symbolic AI: Explainability, Challenges, and Future Trends*. 2024. DOI: 10.48550/arXiv.2411.04383. arXiv: 2411.04383.

[212] X. Zhang, T. Zhu, et al. "Local Path Planning of the Autonomous Vehicle Based on Adaptive Improved RRT Algorithm in Certain Lane Environments". In: *Actuators* 11.4 (2022). Article No. 109. ISSN: 2076-0825. DOI: 10.3390/act11040109.

[213] T. Zhao et al. "Formal Certification Methods for Automated Vehicle Safety Assessment". In: *IEEE Transactions on Intelligent Vehicles* 8.1 (2023), pp. 232–249. ISSN: 2379-8904, 2379-8858. DOI: 10.1109/TIV.2022.3170517.

[214] X. Zhong et al. "Hybrid Path Planning Based on Safe A* Algorithm and Adaptive Window Approach for Mobile Robot in Large-Scale Dynamic Environment". In: *Journal of Intelligent & Robotic Systems* 99.1 (2020), pp. 65–77. ISSN: 0921-0296, 1573-0409. DOI: 10.1007/s10846-019-01112-z.

[215] S. Dal Zilio et al. "A formal toolchain for offline and run-time verification of robotic systems". In: *Robotics and Autonomous Systems* 159 (2023), p. 104301. ISSN: 0921-8890. DOI: 10.1016/j.robot.2022.104301.