CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Agent-centric temporal logic planning for autonomous vehicles

Master's thesis in Computer science and engineering

ALINE EIKELAND

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Agent-centric temporal logic planning for autonomous vehicles

ALINE EIKELAND

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Agent-centric temporal logic planning for autonomous vehicles
ALINE EIKELAND

Supervisor: Nir Piterman, Department of Computer Science and Engineering
Examiner: Yehia Abd Alrahman, Department of Computer Science and Engineering

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

Agent-centric temporal logic planning for autonomous vehicles
ALINE EIKELAND
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

To responsibly introduce autonomous vehicles in practical use, we first need ensure their safety. However, testing autonomous vehicles require more time than what is actually feasible. Instead we can create a controller using synthesis, which provides a safe implementation for a formal description of the environment and system demands. Its practical use is limited due to how it is strongly affected by state space blowup, a factor heavily restricting the size of model variables.

We design an agent-centric model for a vehicle with the objective to avoid obstacles and navigate towards a target, believing that this novel perspective will limit the state space issues while it, unlike other models with similar purpose, does not require live synthesis or preplanning on a scaling number of specifications. Further, we showcase a possible framework for integrating the controller into a physical vehicle system. We also present a simulator, allowing one to freely explore the behaviour of the agent-centric model.

The final result is a model that is fully synthesised to a single controller in preplanning. The navigational model shows promise in its adaptability to unexpected map changes but fails to navigate through every task, getting stuck in certain situations and acting against common traffic norms in certain situations.

We believe that the agent-centric model could potentially be used in a more flexible vehicle controller, and that both the agent-centric spatial approach and the directional target navigation are possible perspectives worth considering for work in this area.

Keywords: agent-centric, controller, obstacle avoidance, formal methods, synthesis, linear temporal logic, autonomous vehicles, traffic safety, computer science, engineering.

Acknowledgements

I am deeply thankful to my supervisor Nir Piterman for invaluable advice and great support throughout the duration of the project.

My appreciation also goes to Birgit Grohe, Ingegerd Nilsson and Elke Mangelsen for helping me find my way towards this thesis.

Additional thanks to friends and family for the love and support. Special thanks to the cat, even if she did not do much.

Aline Eikeland, Gothenburg, October 2024

Contents

List of Figures

List of Tables

1
Introduction

The wildly complicated world of self-driving vehicles has a verification problem. The safety and reliability of an autonomous vehicle system is crucial, due to the possible danger and high stakes involved in its runs[1]. Formal verification methods are traditionally applicable on components of the system, but not on the controller coordinating its many parts[2],[3]. Bugs in the controller are discovered by running tests, running simulations, by using the system or not at all. When these bugs are revealed, the work of modifying the complex controller can be prolonged and complicated.

A systematic development of the controller can limit the prevalence of errors and prevent unnecessary delays caused by a project group having to rework faulty implementations. A strategy used for systematic controller development is reactive synthesis[4]. Reactive synthesis uses a formal specification, with atoms as state variables describing possible actions of the system and the environment, to generate a system design in the form of a state machine. The specification is translated to a two-player graph game, where the system is modeled as one player and the environment as the other. If there is a winning strategy for the system player, the specification is realizable. The strategy is then the given design, guaranteed to fulfill its specification by the method of construction. In the case of navigation, one constructs a graph game representing the response of the system to different events in the environment. A typical objective of such a graph game is reachability, or variations on reachability, that is the objective to reach certain states an infinite or finite amount of times.

For the purpose of self-driving vehicles, different models are used to describe the continuous states of the domain of the vehicle as discrete states[5]. This process is described as discretisation. Current approaches model an agent moving in a space, henceforth referred to as an arena. However, the complexity and scale of the arena is a complicating factor. The translation process from temporal logic specifications to finite state automata suffers from so called state blowup[2]. The possible states of the controller may be as many as every combination of every possible value of all system and environment variables. This massive computational complexity often renders the method unusable.

A method for reducing the complexity of the problem uses a receding horizon strategy[2]. It models the entire task and ensures that the full objective is possible, but finds a state on a limited distance and only computes a plan for reaching that target

state. When closing in on the target it recalculates for a limited distance further on. While the method greatly limits the complexity of the problem, it still scales inadequately for cases with large arenas or many environment variables[2],[6]. Since it has to recalculate the controller each time it changes horizon target, the method needs a reliable method of live synthetisation.

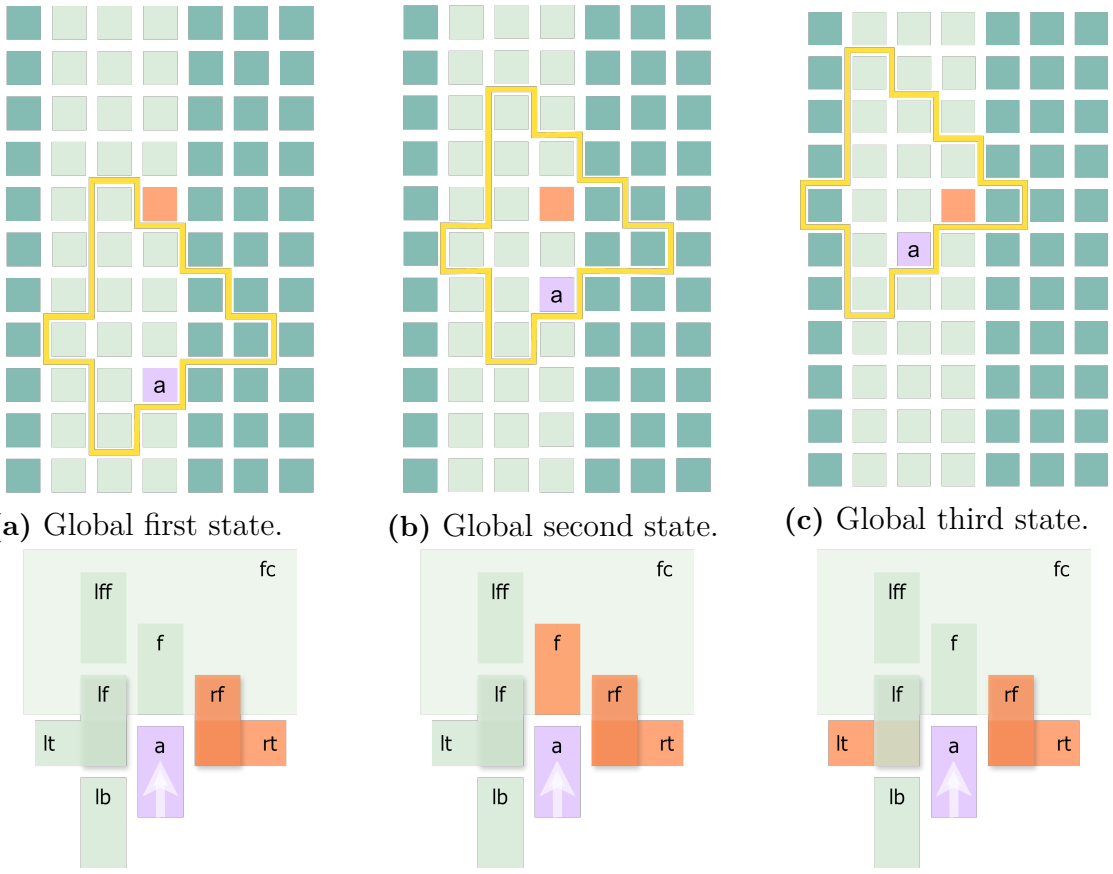
An alternative model would not describe an agent moving around in a space, but a static agent with an environment changing within the field of visibility of the agent. This would be a more intuitive model, since the reality for this type of system is that it controls the agent and not its space. It might also avoid the frequent recalculations associated with the receding horizon strategy. Since this model would not describe an arena containing an agent but an agent surrounded by a dynamic care region, it would potentially not have a limitation to the size of the arena.

1.1 An agent-centric approach

Let us look at an example of a vehicle passing a static obstacle, pictured in 1.1. In the first state 1.1a, the vehicle is moving northward towards its target location. At this point the vehicle is unbothered by the obstacle further forward, which is not yet obstructing the movement of the vehicle. Adhering to common traffic laws, the vehicle keeps to the right of the road if possible. In the second state, 1.1b, the vehicle is close enough to the obstacle that it cannot continue straight forward but needs to select an alternative path. Therefore, in the last state 1.1c it has shifted left towards the centre of the road in order to pass the obstacle.

In a classic approach for modeling the entire road snippet, the model describes a vehicle moving in an arena. It uses variables to track if there are obstacles in every single square of the discretised arena, or in some smaller care region. These variables refers to the same locations in each state of the journey. Another variable tracking the position of the vehicle changes as it moves, holding the value of the current position of the vehicle. From the first to the third state this model describes the position of the vehicle, and the existence of the marked obstacle, as soon as the obstacle is detectable. The model will then expect the static obstacle to remain in place indefinitely, and hence the value of the variable to remain the same.

The alternative agent-centric model only considers the space closest to the vehicle, that acutely may impact it. Regardless of the speed of the vehicle, it models a static agent, where the environment may move around it, depending on its actions. Its inputs describe whether or not there are obstacles in risk of being on a collision path with the agent. When revisiting the problem in 1.1, note that in the first state 1.1d, the obstacle is so far ahead of the agent that it does not yet need to consider it. However the care area indicates that moves to the right are prohibited. Why is not indicated by the crude inputs, but it is sufficient information to advice the next action: continue forward. Obviously from the global view 1.1a it is apparent that the blockage is impassable offroad terrain. In the second state 1.1e, both moves right and forward are blocked. This is the obstacle now presenting in front of the vehicle,



(a) Global first state. **(b)** Global second state. **(c)** Global third state.

(d) Agent-centric first state. **(e)** Agent-centric second state. **(f)** Agent-centric third state.

Figure 1.1: A top-view of three discretised snapshots of a vehicle traveling north, passing a static obstacle, marked a darker orange; seen in the upper row from a global perspective and in the second row from an agent-centric perspective, where the upper part of the map is in front of the agent. The cells portrayed in the regions in the lower row are marked with a yellow frame in the global views of the top row.

and a continuation of the offroad terrain. At this, the controller signals a shift to the left. In the third state 1.1f, the same obstacle that was in front of the vehicle in the last state has now moved to the right of it, what in the global perspective 1.1c appears as the vehicle moving around a static obstacle.

The agent-centric modeling should work as an alternative model to the space-centric. It has no need for runtime synthesis or large memories holding a large number of controllers, a memory demand scaling with the size of the task. With the use of an external third party navigational system tracking the positions of the agent and its target, the agent-centric model could bring additional benefits such as a reduction on the number of unnecessary assumption violations and faster recovery from some undesired transitions.

1.2 Aim

The aim of this project is to find an agent-centric temporal logic model for an autonomous vehicle, capable of navigating towards a target location while avoiding obstacles in its path. That it is agent-centric means that it is a model of a moving environment surrounding an agent vehicle, not a semi-static environment containing a moving vehicle.

The purpose of such a model is to have fewer states compared to a straight-forward implementation, advance the possible complexity of the task, usually limited by the vast amount of environmental space one has to regard, and increase or abolish the need for the size limitation of the modelled arena.

Compared to other typical state-space reducing models, an agent-centric model could potentially be more robust at handling some inconsistencies between the plan and the reality and not need to perform synthesis during the task runtime nor to carry a plan scaling with the size of the task.

1.3 Specification of the issue being investigated

The aims of the project can be expressed as a list of questions to answer:

- Can a specification of an autonomous vehicle controller be written from an agent-centric perspective, in a way that avoids both runtime synthesis and preplanning on a set of specifications scaling with problem size?
- How will the trajectory of surrounding static and moving obstacles be predicted, to allow for successful obstacle avoiding?
- How will navigation integrate into the agent-centric model?

1.4 Limitations

This project will not handle the discretisation of the environment, that is, representing the reality of a world with continuous values as a model with discrete values. Discretisation is quite a large subject of its own and would distract from the main purpose of the project. The text will also not discuss continuous time or disturbance. While these considerations are necessary in order to get practical use of the possible findings of this project, it is not quite relevant to the subject of this report.

Furthermore, the project will not detail vehicle specifics nor traffic conditions beyond what is determined to be useful for a large subsection of vehicles. While the thesis might exemplify its work for specific conditions, it will not shift focus towards vehicle mechanics or traffic jurisdiction.

1.5 Ethical concerns

Self-driving vehicles which might be able to hurt people, animals or the environment should not be allowed outside of research environments if they can not be proven safer than conventional vehicles[1]. Considering progress in the field of formal methods for autonomous vehicles could result in their wider use or increased acceptance, this project needs to be transparent with its contributions and prioritize quality, to not counteract its purpose of improved safety and correctness of vehicle controllers. Note that even if the synthesis process can be verified as correct, the formulation of the specification might be lacking. Testing and simulations should continue to play a significant part in the introduction of autonomous vehicles in practical use.

1.6 Thesis outline

This thesis starts with going through the relevant theory in chapter 2; including linear temporal logic, generalized reactivity, synthesis, the issue of state space explosion and one of its suggested remedies, the technique called receding horizon control. It continues in chapter 3 to explain the method used to run the system, focusing on the architecture of a controller of a potential physical system. In comparison to this physical system it presents a modification of this method and architecture used for the implementation of a simulator, a product of this project. Chapter 4 examines the main project artifact, a specification for an autonomous vehicle designed to work with the simulator and a potential physical system. It explains the various parts of the specification, shows how the vehicle handles a couple of example traffic situations and presents some problems of the model. Finally, chapter 5 discusses the potential takeaways from this project and suggests future areas of research.

2

Theory

This theory chapter first attempts to give a brief review of the research this project is based on, by explaining linear temporal logic, synthesis and generalized reactivity. It continues to explain the problems preventing wider practical use of the technique, as well as a strategy that attempts to solve or at least remediate those issues.

2.1 Linear temporal logic

To understand the rest of this thesis, one first must understand linear temporal logic, abbreviated LTL, first presented by Amir Pnueli in [7]. LTL is a formal system for temporal reasoning, composing propositions from logical atoms united by logical connectives[7],[8]. These are the familiar logical connectives $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction) and $\implies$ (implication). The modal temporal connectives new to this formal system are *next*, *always*, *eventually* and *until*.

A model σ for an LTL formula ϕ is an infinite sequence of states, a run[7],[8]. This run is an execution $\sigma = v_0 v_1 v_2 ...$ where v_i is the state of the execution σ at time i and each state is a valuation of the propositions of the subject formula. $v_i \models p$ means that for an atomic proposition p, v_i holds, that is, p holds at position i of execution σ. The rest of the semantics are defined inductively for formulas ϕ and ψ.

- $v_i \models \neg\phi$ iff $v_i \not\models \phi$
- $v_i \models \phi \vee \psi$ iff $v_i \models \phi$ or $v_i \models \psi$
- $v_i \models$ *next* ϕ iff $v_{i+1} \models \phi$
- $v_i \models \phi$ *until* ψ iff there is a k, $k \geq i$, such that $v_k \models \psi$ and for all j, $i \leq j < k$, $v_j \models \phi$

Additional connectives are essentially useful abbreviations for other connectives.

- $\phi \wedge \psi \equiv \neg(\neg\phi \vee \neg\psi)$, $\forall k \in [i, j).v_k \models \phi$
- $\phi \leftrightarrow \psi \equiv (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$
- *eventually* $\phi \equiv$ *true until* ϕ
- *always* $\phi \equiv \neg$ *eventually* $\neg\phi$

Lastly there is some additional necessary terminology on this topic. A boolean formula is a formula that does not use temporal connectives[9]. A boolean combination of formulas $\phi_1 ... \phi_n$ does not use temporal connectives except for those contained in the formulas ϕ_i. With that, one can move on to the next section for an introduction to synthesis.

2.2 Synthesis

Synthesis is the process of transforming a formal specification of a controller to an implementation that is guaranteed to fulfill that specification[4]. This program is a realisation of the specification. The realisation process is predecessed by a check of whether or not the specification is realisable in the first place. Briefly described, this is done by first transforming the LTL specification to a game structure and solving that game. That is, finding if there is any strategy the system can use to always win the game. If that is the case, the specification is synthesised to an actual implementation. This is an extraction of a strategy from the game.

A reactive or open system is a system that reacts to the actions of an uncontrolled environment. It is represented by an open controller, which for a valuation of uncontrolled input variables return a valuation of its controlled output variables.

2.2.1 GR(1)

Generalized reactivity, GR(1) provides a form for LTL specifications for which a synthesis can be performed in polynomial time with regards to the state space of the formula[9],[10]. The form is relatively restrictive but supposedly rich enough to specify many controllers well. A specification of a reactive system is phrased

$$\begin{aligned}(I_e \wedge \mathit{always}\ S_e \wedge \bigwedge_i \mathit{always\ eventually}\ P_e^i) \rightarrow \\ (I_s \wedge \mathit{always}\ S_s \wedge \bigwedge_i \mathit{always\ eventually}\ P_s^i)\end{aligned} \tag{2.1}$$

where I_e is a specification of the initial state of the environment and I_s likewise for the system, *always* S_e and *always* S_i are transition predicates or safety properties of the environment and system respectively and P_e^i and P_s^i are progress properties of the environment and system respectively. I_e, S_e and $\bigwedge_i$ *always eventually* P_e^i are assumptions on the environment while I_s, S_s and $\bigwedge_i$ *always eventually* P_s^i are guarantees from the system. Altogether this single implication can describe many systems quite well.

Boolean formulas I_v describes the initial state of the environment or system v[9]. For example, $I_e = \neg$ *collision_risk* states an assumption on the environment that the journey will not start with a collision risk.

The transition predicates *always* $S_v = \Phi_t^v$ express a sentiment about the value of a variable in the next state. The assertion on environment transitions is a boolean formula describing the relation $\phi_e(E, S, E')$ between valuations of environment and system variables in the current state, E and S respectively, and valuations of environment variables in the next state E'. The assertion on system transitions is a boolean formula describing the relation $\phi_s(E, S, E', S')$ between valuations of environment and system variables in the current state, E and S respectively, and valuations of environment and system variables in the next state, E'

and S' respectively. Note that assertions on system transitions may express sentiments about the next state valuations of both the system and of the environment, whereas the assertions on environment transitions only express valuations of the environment variables. This means that the consequences of the system output can be described by the valuation of the next state environment variables. For example, $\phi_t^e = always\ (obstacle\_in\_front\_of\_agent \wedge agent\_move = forward \rightarrow next\ collision\_risk)$ states that if there is presently an obstacle in front of the agent and the agent moves forward, the obstacle and agent will collide in the next step.

Progress properties $\bigwedge_i always\ eventually\ P_s^i = \phi_g^v$ or recurrence properties are properties that will always eventually become true. For example, $\phi_g^s = always\ eventually\ \neg agent\_move = halt$ states that eventually, the agent will do something else their turn but waiting.

Formula 2.1 can be transformed into a game where the winning condition is an expression based on the progress properties of the environment and system[9]. The winning condition is phrased

$$\begin{aligned}(always\ eventually\ p_1 \wedge ... \wedge always\ eventually\ p_n) \rightarrow \\ (always\ eventually\ q_1 \wedge ... \wedge always\ eventually\ q_m)\end{aligned} \tag{2.2}$$

. Again, every subformula p_i describes an assumption on the environment and every subformula q_j describes a guarantee of the system given the assumptions on the environment. This winning condition is the formula Generalized Reactivity(1) (GR(1)) and its synthesis is relatively efficient, at an upper bound complexity of $O(N^3)$, while the realisability check has upper bound complexity at $O(N^2)$, both with regards to the state space of the formula. The synthesis is done using a symbolic algorithm, meaning that it does not have to evaluate every state one by one, but can disregard sets of states without enumerating individual states. It does so until it finds a fixpoint, where no more sets of states can be disregarded.

Note that the synthesis process must practice the use of so called strict realisability, that the specification is not realised by a strategy that wins the game by ensuring that the assumptions on the environment does not hold[11],[9]. While that would satisfy the formula, the resulting synthesised controller would not force the system to fulfill the system guarantees, and such not have any real use. Hence, the synthesis process uses a more complex version of the implication 2.1 as input, to ensure strict realisability.

2.3 State space explosion

Many researchers on the topic of controller synthesis are concerned with the large state space[12],[13],[14],[6]. Different approaches to the problem has been taken.

A common approach is to subdivide the solution into several controllers which each have a smaller responsibility[2],[14]. Even if their responsibilities may overlap, they

might enable increasingly complex models since the individually are smaller than a corresponding comprehensive controller. These lower-lowel controllers are then used either sequentially or layered , organized by some higher level controller.

These situational controllers are planned partially or fully either before the mission or live during runtime, as they are needed. This will further be referred to as premission planning and live planning respectively.

2.3.1 Receding horizon control

Receding horizon control, as described in [15],[2], splits the navigational problem into many smaller problems. The states are grouped in a particular arrangement for each progress property of the specification. These arrangements are ordered arbitrarily. In every one of these arrangements, the states are divided into ordered groups where the last group W_0 contains states where the progress property is fulfilled, and the other groups W_n through W_1 are ordered by their "closeness" to W_0, where W_1 is the subdivision closest to W_0 and W_n is the one farthest from W_0.

A short-horizon specification is formulated for each group in each arrangement[2]. It specifies that one must reach a state in a group that is by a certain distance closer to W_0. This short-horizon specification W_i is the one realized, live in mission, when the vehicle finds itself in a current state v that happens to be an initial state of W_i. When finding the specification to realize, one first looks if for any i, $v \in W_0^i$, that is, if the system is fulfilling a progress property. If so, the state is in W_0^i and a short-horizon specification belonging to another progress property is found, W_k^{i+1}. The new short-horizon progress property is satisfied in states in W_j^{i+1} where $j < k$, that is, ordered as closer to W_0^{i+1}. One iterates on this, synthesising a new controller every time one has progressed closer to satisfying the contents of a progress property.

The divisions of the states into groups W_0^i through W_n^i for each i is performed manually[2]. Each short horizon specification must be found realizable in the premission planning and this require a proper ordering of the divisions, P_i. To help with that, an invariant Φ is defined as describing safe initial states for each short-horizon specification.

Applied to a vehicle navigation problem with a single progress property to reach location v_t, a discretisation of the arena is performed. This means that the road is divided into cross sections W_i through W_0 where $v_t \in W_0$[2]. A transition system T has cross sections as states and edges between all connected pairs of cross sections.We observe the current environment and system state as v. $v \in W^j$ where $j \leq i$. The goal planner then uses some common search algorithm to find a path from W^j to W_0^1. The trajectory planner synthesizes a specification representing the journey between the current cross section and a cross section a horizon length further on the path calculated by the goal planner. It then provides a controller responsible for executing moves with the next system state. If this moves it further along the path, it synthesizes a new controller.

Recall that in mission planning, one checks if the specification is realizable[2]. Live in mission, one synthesize a controller with a single initial state in the current state. Since the state blowup for checking if a specification is realizable is much smaller than for actually realizing a specification, receding horizon control allows solving of a navigational problem without the synthesis process being limited by the arena size.

An unexpected problem can present as for example a cross section being entirely and permanently blocked. In this case, the trajectory planner will alert the goal generator, which will regenerate the sequence with the transitions through the blocked section removed[2].

The horizon length is defined as the amount of steps closer to the target destination one must plan the short-horizon specification to[2]. Compared to a synthesis of a representation of the whole arena, using the receding horizon model improves the performance greatly if horizon length T is much smaller than the amount of divisions p.

Note that the method does not scale well for problems with many possible paths and. While a simple road network has groups W_i the size of a road cross section or perhaps the space inside an intersection, for example arenas with large open spaces can have many connected locations with equal "closeness" to the target. The specifications that navigate through these areas will have to consider many input variables. For navigational problems, the horizon length grows with the size of the number of possible unexcepted obstacles one can encounter. Imagine navigating to the center of a large open space. All tiles on distance n will be equally close and possible to move inside. The further from the target, the more tiles will be considered. While the problem is smaller than naively considering all tiles simultaneously, it grows depending on the distance to the target.

3
Methods

The main project delivery is a model in the form of a specification for a controller of an autonomous vehicle. When synthesised with an existing synthesis tool, the resulting state machine acts like a discrete controller, taking certain inputs of information about the state of the environment and outputting move directions to be applied to the vehicle.

This chapter will begin to briefly describe the role of the state machine acting as the logical controller. From there, it will describe the characteristics of a framework to embed the logical controller into. It will first present the proposed architecture and behaviour of a physical system, and then continue to show the implementation of a simulated vehicle system.

Here, the reader will also be provided with an action plan for different situations where the controller fails to handle the environmental inputs. The text will then continue to explain what particular circumstances to consider for the specification of the logical controller. Finally it will describe the different stages of production of the logical controller, to supply the reader with the option to replicate this work.

3.1 System

The autonomous vehicle that is the subject of this project is a simplified subject with two objectives - avoiding obstacles and navigating to a target location, hereforth referred to as simply the target. The target location is far enough such that modeling the entire network between the agent and the target would result in an enormous state space, too large to perform synthesis on.

The logical controller is a machine with a simple interface that accept and outputs the values of a few variables ranging over finite, discrete and small domains. It takes the input of the direction of the target, a target direction, and a set of inputs each declaring whether or not there are an obstacle in a specific location zone in the area surrounding the agent vehicle. The controller is expected to output a move of the agent. This should be a move forward, to the left and forward or to the right and forward; a turn left or right or a halt. The controller holds an internal state which enables it to know what moves are allowed by the environment and system at each certain point in time.

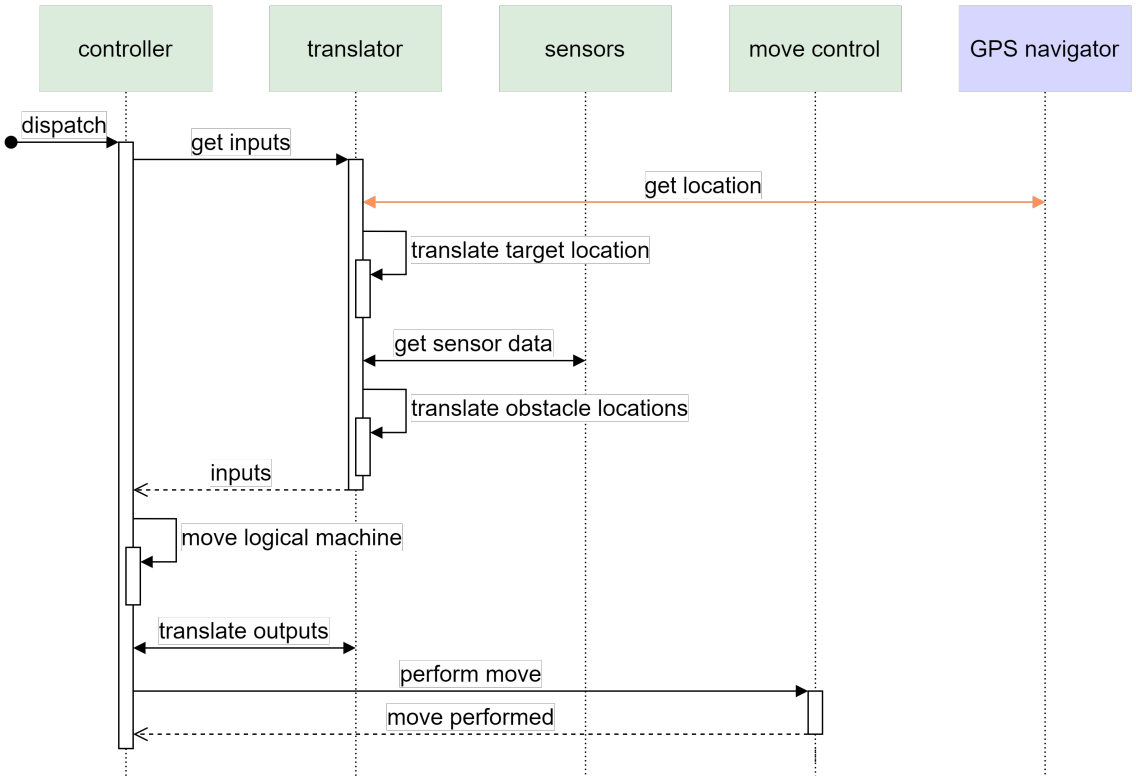


Figure 3.1: Sequence diagram displaying system interactions during a time step.

The logical controller needs a way of interacting with the lower level components of the vehicle. This interaction is fully facilitated by the topmost controller. Essentially, the topmost controller of this system is a wrapper around the specified logical controller, working at the top of the hierarchy in a large and complex framework to coordinate and translate between the logical controller and the complex reality of the physical vehicle system and surrounding environment.

During runtime the topmost controller holds and interacts with the logical controller, a translator, a sensor abstractor, a mechanical controller and a GPS navigator. The topmost controller performs a sequence of actions upon entering a new state, essentially translating the environment to logical inputs, running it through its logical machine, and later transforming the logical outputs to machine signals. Upon receiving feedback from the move control that the vehicle should have moved into a new state, the topmost controller restarts its cycle in a new time step. A sequence diagram describing the interactions of the topmost controller during a time step can be seen in figure 3.1.

Looking closer at this time step, the cycle begins with multiple actions inside the translator, which serves the logical controller as an interpreter of the environment. First, it calculates a target direction. This is an input representing the angle of the distance between the agent and the target against the trajectory of the agent. This information is supposed to help the vehicle to navigate towards the target. For this purpose, a GPS navigator locates the vehicle and target by some global

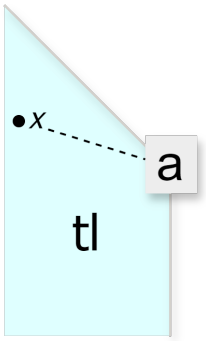


Figure 3.2: Map over the input to the controller for targets in different relative orientations to the agent. Target in marked location t would translate to input t_l.

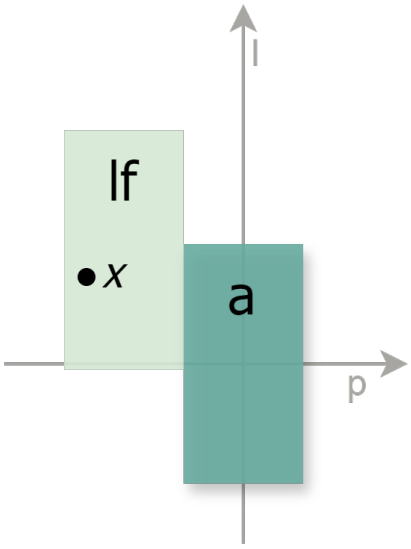


Figure 3.3: An example of a zone lf defined as perpendicularly within $(-3, -1)$ and laterally within $(0, 4)$. Location x at approximately $(-2.7, 1.5)$ is marked.

measurement. This information is sent to the translator. The translator then uses this information to calculate the input which is the target direction. Based on the information from the GPS navigator, it calculates the angle between the centerline of the agent and the line from the agent to the target. This angle will be classified as within a certain target direction, classes of spans of degrees of the angle. An example of a translation to a target direction can be seen in figure 3.2.

Now, the translator calculates the value of multiple environment variables that all concerns the existence of obstacles within a certain area, here referred to as a zone. Each one of these boolean variables simply declare whether there is an obstacle within a certain zone or not, and their purpose is to help the agent to avoid obstacle collisions.

A zone is defined by a lateral and a perpendicular span of distances from the agent.

A position within the system is denoted by its placement within a carteesian system where the center of the agent is located in origo. Distance is measured perpendicularly and laterally from the agent. The lateral direction is measured along the driving direction, the direction a driver would see forward from the drivers seat. Positive measurements are in front of the vehicle and negative behind the vehicle. The perpendicular axis is perpendicular to the lateral one. A driver would interpret negative perpendicular measurements as left of the agent and positive to the right. An example of a zone definition can be seen in figure 3.3.

The zone definitions are sent to a lower level sensor abstractor. It uses techniques such as image recognition and radar to determine whether or not there is an obstruction in each zone and additionally uses radar detection to determine if an obstruction is a moving object and if so, which direction it is currently moving. The sensor abstractor then forwards this information to the translator, which further abstracts it to a boolean value for each zone, using the data about possible direction of movement for possibly disregarding some obstacles. This could be for example obstacles already at a safe distance from the agent moving further away. The exact calculations made by the translator is explained in section 4.1.1 and the logical propositions of the environment variables concerning obstacles in tables 4.2, 4.3 and 4.5.

With the inputs translated, the topmost controller receives these inputs and feeds them to the logical controller. This controller has an internal state that combined with the environmental inputs indicates what moves are legal and what moves the environment can be expected to do next. On input, it deterministically updates its internal state and outputs the actions to be performed by the vehicle.

The controller now delegates the execution of this move to an appropriate instance, a move controller. The move controller must first transform this abstract instruction to a larger set of concrete machine signals and then execute and monitor those subtasks until they are deemed as completed. Then, it reports back its success status to the topmost controller.

Upon a response confirming that the move has been executed, the iteration starts over. A failure to execute the move will result in a mismatch between the physical reality and the abstract model of the logical controller. These mismatches can appear in multiple ways, and their handling is the subject of the next subsection.

3.1.1 Reality to model mismatch

The system might find itself in a situation where the physical reality does not match the abstract model of the logical controller. Then, due to the implementation of strict realisability in the GR(1) synthesis algorithm, explained in 2.2.1, as the assumption on the environment fails the entire specification fails. The controller then enters a failed state and restarts. In the case of the move controller properly recognising its failure to execute a move instruction, the topmost controller is notified of this deviation from the plan and is able to start a recover plan. However many

mismatches are unpredicted by the system and become apparent when the logical controller can not find a transition from its current internal state matching the provided inputs. Then the logical controller will note a failed state and prompt the restart.

These mismatches could happen for a multitude of reasons, including a failure of the environment to follow its defined logic, a wrong time for a new controller move, or a transition from the last state to the current that was not predicted by the controller - perhaps bad road traction caused the agent to end up in a different location than intended or an environment vehicle was speeding and therefore moved further than its predicted move in a time step.

The controller will attempt to recover from the problem by resetting the logical controller state to the initial state. If there is no transition from that initial state it means that the agent is in an illegal initial state. Hence the problem is beyond the responsibilities of the controller and should be passed to an entity handling emergency or otherwise unexpected situations.

3.1.2 Simulation counterpart

For the purpose of showcasing this project, a simulation program for agent-centric vehicle controllers was created. This program can be viewed and downloaded from a GitHub repo[1]. The simulator attempts to implement the behavior and responsibilities of the topmost controller. Lower level components serve to simulate the behaviour of its real life counterparts in order to provide the topmost controller with a similar interface.

The simulator allow the user to create their own maps with different agents with different starting positions and targets. It defines a road network by painting passable and non-passable locations. It allows static obstacles with fixed time spawns and fixed or randomized positions. It allows the user to define fixed paths for moving obstacles. This map and possible randomized start locations and static obstacle locations defines the runs of the agents.

The interface of the GUI application is pictured in 3.4, and is where the user can experiment the runs of map-defined agents all using the same logical controllers. A path to a map is given as an argument to the application. Buttons allow the user to step through a run. A dump option prints a history to a text file.

A time step of the simulation is visualised as a sequence diagram in 3.5. The initiating component is a task object holding the actors and their arena, an object with a global view over all location tiles. The actors are both environment objects with different internal logic and agents. Actors do not own a global view but are provided access in order to "read" the locations of its care area, an object which roughly works as a translator in this simulation. The environment objects uses fixed

[1] https://github.com/alinje/agent-centric-vehicle-model

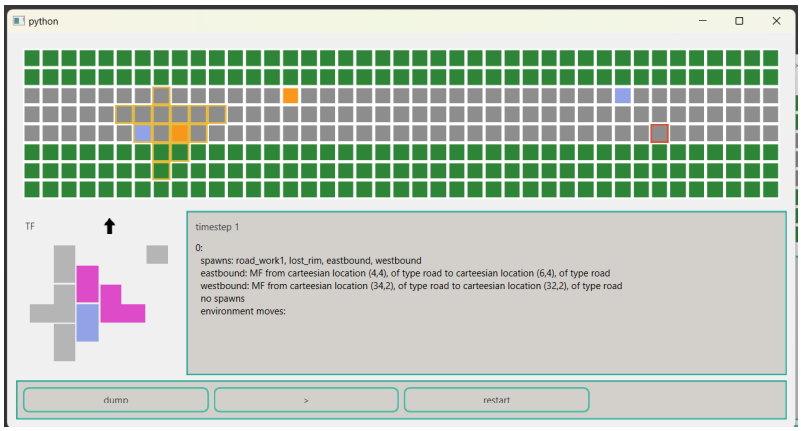


Figure 3.4: The Python tool used to analyse the synthesised controllers. The uppermost map show the full arena with pre-determined and runtime generated obstacles. The map below shows the environment variables fed to the controller.

instructions for their placements, possibly changing over time. They are provided with some simple rules for avoiding being the cause of unlawful collisions. Agents have logical controllers deciding their moves and work with translating the logical in- and outputs back and forth. These translations are done with the help of the so called care area which acts as a translator.

In a time step of the simulation, the agents operate first, then possible new spawns are placed and finally the environment moves. Note that in the order in the game of the synthesis process where the environment start and the agent react to the environment state. Hence a time step of the simulation essentially stretches over two halves of a time step of the logical specification. The resting state is placed here to provide the user with a view of what the agents are reacting to.

The task sequentially prompts a move from its agents in constant order given from the order of definitions in the map. By the constant sequential ordering the agents can act upon the actions of the other agents, which they recognise as part of their local environment. A time step of an agent begins with the agent object prompting the care area for the inputs. Each care area is associated with a certain logical controller. It might also hold flexible variables clarifying how to perform the interpretation of the logical variables. Using the arena handed down from the task, the care area translates the global information to information imitating what can be sensed by its associated controller. The care area first acts like a mask that for an arena input, reading only the information that should be accessible for the agent at that time. This information is then translated to the form of logical inputs and returned to the agent.

As the inputs are translated, the agent can now move the logical machine. The logical machine returns an output which represents a move to be performed by the agent. The agent translates this information with the help of the care area, updates its internal state and informs the arena of the changes. Documentation describing

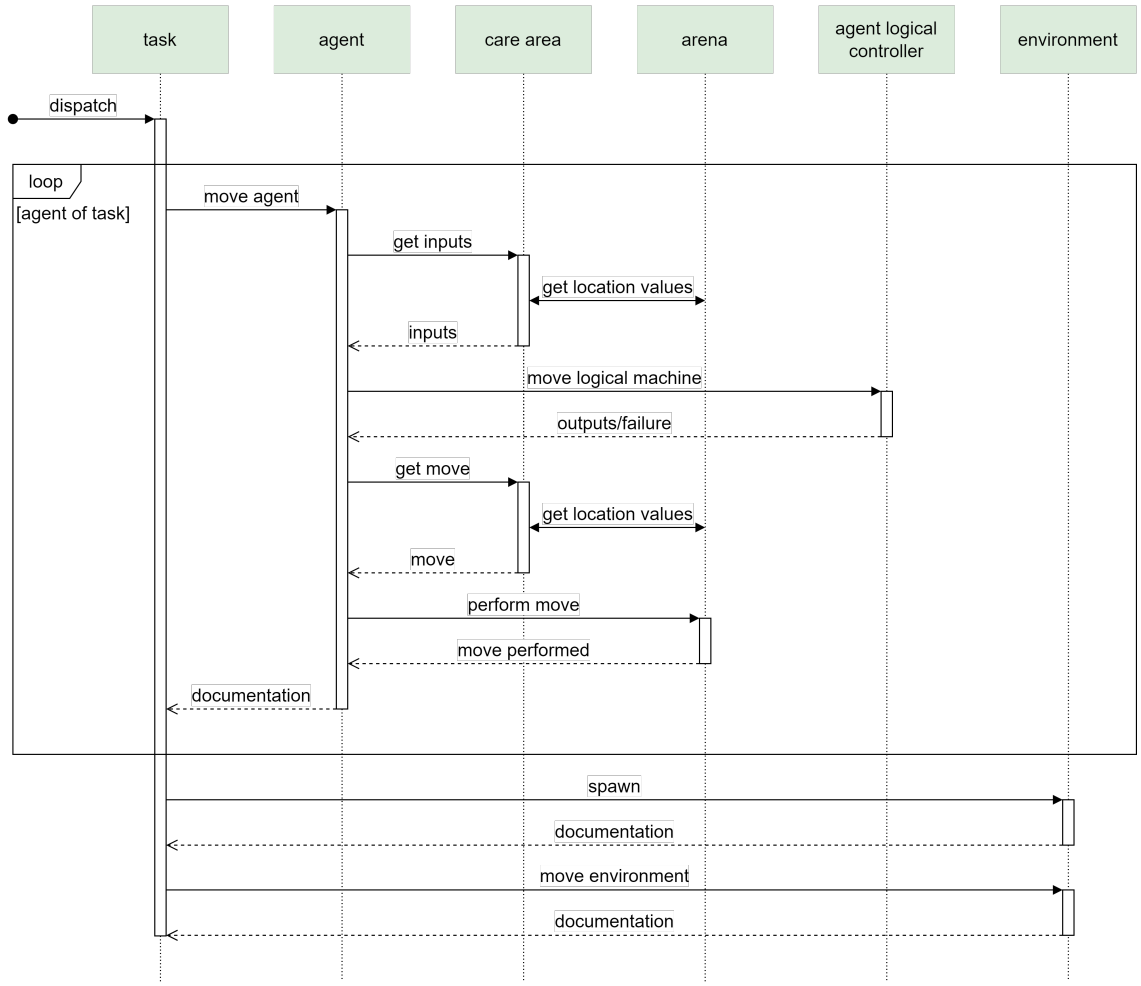


Figure 3.5: Sequence diagram displaying simulation system interactions during a time step.

the resulting move and new state of the agent is returned to the task.

With all agents moved, the task now updates the environment. First it spawns potential new environment objects at safe locations. Then it moves existing environment objects by their internal logic. These actions are also documented and returned to the task object, which completes the cycle by appropriately archiving the documentation.

As a final note on the simulation program, note that the arena of the simulation is fully discrete. For reason of simplicity of implementation, it uses locations with discrete coordinates. This means that all actor movements are between a full occupation of one location to a full occupation of another. Where the translator interacts with the real world, few points of interest would ever fall directly on a limit between multiple zones. The rare cases where a lower level sensor would measure them to this exact position can be dealt with in different ways. For example, an obstacle could be handled as one in each zone or one in the zone with the largest reason for concern is, or the least.

3.2 Demands on the logical controller

The vehicle specification is based on a vehicle navigating a right-hand traffic network. The system is assumed to adhere to certain traffic laws such as priority to the right, priority roads and passing in outer lane. This makes for an unsymmetrical model and allows experimenting with different behaviors for traffic obligations and rights for many situations. It also allows one to cut down further on the state space and create an unsymmetrical sensor space, considering that the sensors only need to detect oncoming vehicles on its left side and perpendicularly cutting vehicles from its right.

Obstacles can be static or moving, representing for example road construction work or oncoming traffic respectively. The possible trajectories of moving obstacles were derived from Swedish traffic laws. Traffic signals were not regarded as inputs.

Moving obstacles are assumed to move at the same speed as the agent, except passing vehicles which move at twice the speed. This is to reduce the state space. A model where vehicles are capable of moving at different speeds would require a lot of added complexity. For example, the length required to model different a passing vehicle would have to be doubled to be able to display cars passing in two different speeds. This additional complexity would not bring enough interesting results to be worth the extra power needed to perform the synthesis.

These aspects of the controller were logically specified in linear temporal logic, LTL, phrased in GR(1) to enable a synthesis process. This specification is explained in chapter 4 and given in its entirity in tables 4.1, 4.2, 4.3, 4.5 and 4.4. The model was synthesised using the Temporal Logic Planning Toolbox[16], TuLiP. It was synthesised to a Moore controller.

4
Results

This chapter details the result of this project, the agent-centric model. The model is a specification which uses variables from two different sets, controlled by the environment and the system respectively and shown in table 4.1. The specification consists of multiple sets of logical propositions. These are the initial properties of the environment, presented in table 4.2, environment safety properties, presented in table 4.3, system safety properties, presented in table 4.4, environment progress properties, presented in table 4.5, and the empty set serving as initial properties of the system and progress properties of the system.

The resulting model avoids both static and moving obstacles (representing crossing and passing vehicles as well as vehicles with paths perpendicular to the path of the agent). It navigates towards a target direction, presented as forward or achievable with a turn to the left or to the right.

The synthesis process took approximately 5 minutes and the resulting automaton has 769 states. Full versions of the Python code implementing the specifications with the help of the TuLiP library can be found as appendix A.

We begin this chapter with explaining some particular roles of the translator for the preprocessing of the inputs to the logical planner. We then continue to explain the logical specification. Here we go through the aspects related to obstacle avoidance, define the locations of the spatial zones and what inputs they relate to and explain the obstacle transitions and the progress properties of the environment. Further we explain the implementation of the target navigation of the model. We define the target directions and target transitions. With that we are able to understand the next subject, the move preferences of the agent. Finally we look at how the model handles some examples of problematic traffic situations. We look at some typical challenges, observe the behaviour of the model and attempt to decide whether its behaviour is acceptable. We classify some situations where the model act inappropriately.

4.1 Specification of the logical planner

The controller takes the input of a target direction, *target* with the possible values t_l, t_f, t_r stating whether the target is left, forward or right respectively, and a set of inputs *olff*, *olf*, *olb*, *of*, *oa*, *orf*, *ofc*, *olt* and *ort* declaring whether there is an obstacle in any of the zones *lff*, *lf*, *lb*, *f*, *a*, *rf*, *fc*, *lt* or *rt*. These zones are pictured in figure 4.1

and motivated in section 4.1.1. Furthermore, the controller is expected to output a move of the agent. This should be a move forward, to the left and forward, to the right and forward; a turn left or right or a halt. These moves are logically denoted $m_f, m_{slf}, m_{srf}, m_{tl}, m_{tr}$. Again, all variables are listed in table 4.1.

This section is dedicated to explaining the propositions making up the environment and system initial state, safety properties and progress properties assumptions and guarantees respectively. We describe these properties from what purpose they serve. First, we will look at what properties that are necessary to guarantee obstacle avoidance. Then, we explore what is needed for the controller to navigate towards a target.

4.1.1 Obstacle avoidance

A main responsibility of the vehicle is avoiding obstacles. These obstacles can be either be space that is not on a path on a map provided before the journey and confirmed to be impassable by the logical controller when encountered by the agent, or obstacles discovered at runtime by some sensor of the vehicle. The controller does not hold expectations on space that have not yet entered a zone. Regardless of the expectations of external components on specific zones, the zones are all represented equally, as binary inputs declaring whether or not they are occupied by an obstacle.

To limit the used state space, these zones are irregularly distributed around the agent and represent different types of obstacles and different sized areas, something further reasoned and discussed in section 5.1. Some zones differ in area coverage depending on the traffic situation. There are in total nine obstacle inputs. A map over the zones of the final model is pictured in 4.1 and the definitions of their associated environment variables are listed in table 4.1.

Before delving into the zone division, let us briefly consider what types of obstacles the agent might encounter. In relation to our agent-centric model, all obstacles are moving, but from a global perspective we can classify obstacles as either static or moving. Static obstacles may be roadwork, parked vehicles or unmapped changes to the road network. Hence, these obstacles essentially behave like offroad spaces, except that they are not mapped and hence can not be used for pre-mission planning. Moving obstacles present as other vehicles following the same set of traffic laws as the agent. Their objectives are unknown to the agent which has to consider all possible actions of surrounding vehicles to operate safely. Static and moving obstacles raise different challenges to the agent, which should respond to particular challenges with individually appropriate responses. Therefore, in some zones, solely either static or moving obstacles are considered. The specific reason is explained in the following zone explanation.

Now, having considered what an obstacle is, let us briefly recognise what the occupancy of a zone is. As previously explained in section 3.1, the translator holds the responsibility to evaluate occupancy for each zone in each time step. Each zone

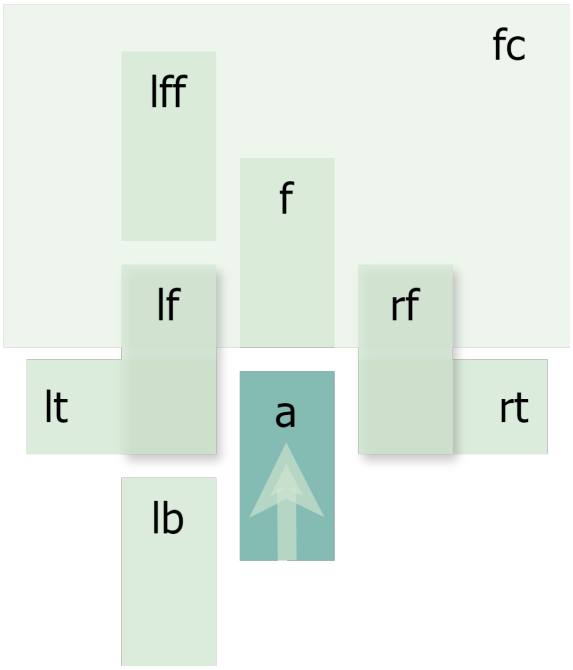


Figure 4.1: The zones whose obstacle occupancy are represented by environment variables.

is associated with a particular area, but depending on the zone the translator has different standards for what occupancy is. While occupancy always mean that underlying sensors have determined that the associated area is impassable, for some zones an area can be impassable without the translator communicating an obstacle to the logical controller. For example, the area to the left and behind of the agent is only occupied if the obstacle is laterally moving towards the agent. That is because an obstacle in that area which is moving away from the agent is not predicted to hinder any movement from the agent.

Note that the logical machine is not aware of the actual geographic position or size of any of the zones. All it considers is the relations between the zones. The translator may potentially change its definition of a zone depending on for example the road type, weather conditions or speed of vehicle; as long as the logical relationships noted in the specification, tables 4.1 through 4.5, hold. Later in this section we will explain how this flexibility is relevant to the zone *fc*.

At this point we may look at the individual zones, what they have in common and what sets them apart. As a natural starting point, let us look at zone *a* and its corresponding environment variable *oa*, stating whether or not there is any obstacle in the zone. This is the zone containing the agent, defined as an unsafe place to host an obstacle. If an obstacle were to enter the zone, a collision may not be avoidable. Hence, the initial assumption on the environment is that the agent does not start with an obstacle inside its own zone, since that would be an impossible situation for the system to solve. An important safety guarantee of the system that no obstacle is ever in this zone, since it should not risk the safety of the vehicle. These are the

simple propositions $\neg oa$, proposition 1 of table 4.2, and *always* $\neg oa$, proposition 1 of table 4.4 respectively.

The current state and the predicted next step state of obstacle inputs regarding zones lf, f, rf, lt and rt show if there are obstacles that could prevent a movement to these zones, since these are the zones one can move to by shifting forward and to the left, moving forward and shifting forward and to the right, turning left and turning right respectively. Most obstacles are problematic in this zone, but you might note that there is a possibility that an obstacle move out of the zone before we reach it. In that case, the zone is safe to enter. However, concluding whether an obstacle will remain or not is more than a simple speed check and should be handled logically. Since this would increase the the factors represented as input variables, it is not prioritized. Such, the agent is rather safe than sorry and does not consider the possibility that the obstacle may move away from the desired zone in time.

The inputs *olff* and *olb* only host relevant moving obstacles. Since they do not represent spaces the agent can move to in one step, static obstacles are not relevant since the model only plan one step ahead. *lb* is the space to the left of and behind the agent and *lff* is the space in front of and to the left of the agent where an oncoming vehicle this timestep would be in *lf* the next timestep.

The zone *fc* is the space where a moving obstacle with perpendicular movement to the agent will show when it threatens to intersect forward movement of the agent in the next timestep. Because of priority of the right, such an obstacle would prevent all forward movement of a vehicle except a turn to the right. This zone does not have any explicit spatial relation to the other zones. Based on the traffic situation this space could intersect either or both *rf* and *rt* or the zone could have no associated space.

Defining the area associated with the zone *fc* is a continuous task performed live during the mission. Sensors scan the road perpendicularly towards the right edge of the road, laterally from the back to the front of the zone f. If it finds a road edge parallel to the zone f with an intersecting road, an area associated with the zone *fc* is created from the first car length piece of that intersecting road. Note that this process is not yet implemented for the simulation, and such also not thoroughly tested. As of now the simulation accepts the location of the *fc* zone as a map input.

During the span of a full run using this technique, this spatial definition is likely to change several times. However, that will be registered and read by calculations in a translator. Due to weak transitions, the logical notion of *fc* can be kept constant.

Now, to follow the behavior of surrounding obstacles over time, and to ensure that the agent only performs moves determined to be safe from collision, propositions describing the behavior of obstacles over following time steps are included as safety environment propositions. These describe what we will now refer to as obstacle transitions and are presented as propositions 1-6 of table 4.3. Note that this section

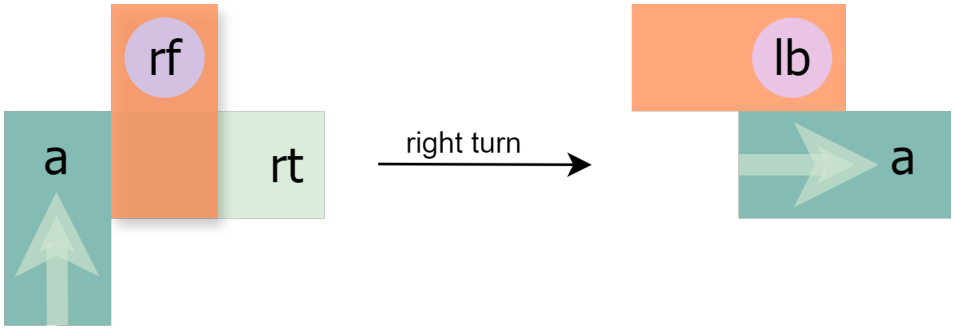


Figure 4.2: Example looking at an agent and a moving obstacle over two consecutive time steps. In the first, the object can be noted as in zone *rf* but not in *rt*. Because it shows in *rf* we know that it is a static obstacle, since no moving obstacle could legally be moving right of the path of the agent. After performing a right turn, the obstacle is in the zone *lb* of the agent, but *olb* is assumed to be false. This is because the obstacle is static, and will not show as an obstacle in the zone *lb*, which is only true for laterally approaching obstacles. It does however block moving obstacles from entering the zone, rendering its value guaranteed false. Hence the full statement is $move = m_{rt} \rightarrow ((orf \wedge \neg ort) \rightarrow next\ \neg olb)$

frequently refers to rows of the model tables for examples and explanations. When referencing an obstacle transition $M \rightarrow (C_1 \wedge ... \wedge C_i \wedge ... \wedge C_n)$ and discussing specifically one of its conjuncts C_i, we might refer to it as an implication to only that single conjunct, as $M \rightarrow C_i$. This is to simplify the explanation by removing clutter. Please observe that $M \rightarrow (C_1 \wedge ... \wedge C_i \wedge ... \wedge C_n)$ is logically equivalent to the more lengthy proposition $(M \rightarrow C_1) \wedge ... \wedge (M \rightarrow C_i) \wedge ... \wedge (M \rightarrow C_n)$. In this section, we prefer to discuss obstacle transitions as single conjuncts from the second form.

Some of these propositions, 1-6 of table 4.3, are transitions of the type "an obstacle in zone A in this timestep implicates an obstacle/lack of obstacle in zone B in the next timestep". These depend on the movement of the agent and hence these transitions are formulated as implications from the agent making a move in the current timestep to a conjunction of legal behaviour. For example, look at proposition 2a of table 4.3, $move = m_f \rightarrow ((of \vee ofc) \leftrightarrow next\ oa)$: if one moves forward, an obstacle in *f* or *fc* is logically equivalent to an obstacle located in *a* the next timestep.

For some obstacles we can be certain of their whereabouts in the next step, and define it as a meaningful logical proposition. However, several obstacles will move out of the care area. Some, we can not predict whether they move out of the care area or not and such make no meaningful predictions. While we can state for example $move = m_h \rightarrow (olf \leftrightarrow (olf \vee (\neg olf)))$, it is a transition statement that does not provide additional information of the behaviour of the environment.

Additionally, note that when these implications declare an absence of an obstacle there might still be an obstacle in that place, it is just not relevant to any predictable

current or future path of the agent. An interesting proposition spawning from this is example of this is $move = m_{rt} \rightarrow ((orf \wedge \neg ort) \rightarrow next\ \neg olb)$, saying that if we make a turn to the right, an obstacle in *rf* in the first state indicates no obstacle in *lb* in the second state. However, looking at this example shown graphically in figure 4.2, we can clearly see that the obstacle is actually located in the area defined as *lb* but *olb* evaluates as *false* since the obstacle is not moving in a way that threatens the path of the agent. Although contraintuitive this implication can be very helpful, since it shows situations where the zone is guaranteed to be free from threats to the agent.

Overlapping zones gives us additional information when considered together. The zone *olf* overlaps with *olt* and *orf* overlaps with *ort*. Knowing whether an obstacle is in both zones or any of the individual helps us specialize some of the transitions. The example in 4.2, $(orf \wedge \neg ort) \rightarrow next\ (\neg olb)$ which tells us that an obstacle that is in *rf* but not *rt* must be static and in *lb* after a right turn.

Other propositions, 7-15 of table 4.3, define the transitions of obstacles attempting to avoid unlawful movements of obstacles such as multiple obstacles on colliding course towards each other. These propositions define how multiple moving obstacles are passing or meeting the agent in the same frame. An example is $(move = m_h \wedge olff \wedge olf) \rightarrow next\ (olf \wedge \neg olb)$, which tells us that if there is an obstacle approaching from *lff* and some obstacle in *lf*, the obstacle in *lf* must be heading the same direction as the other obstacle, and so in the next state there is an obstacle in *lf* and an obstacle exiting the care area from *lb*.

To ensure conditions that allow the progression of the agent, an environment progress property states that there will always eventually appear a clear move. This is formulated *always eventually* $(\neg\ ((olf \vee olff \vee olb \vee ofc)\ \wedge\ (of \vee ofc)\ \wedge\ (orf \vee ofc) \wedge (olt \vee olff \vee olb \vee ofc)\ \wedge\ (olt \vee olff \vee olb \vee ofc)))$, proposition 1 of table 4.5, and can be understood as *always eventually* $(\neg\ (leftForwardShiftBlocked\ \wedge\ forwardBlocked \wedge rightForwardShiftBlocked\ \wedge\ leftTurnBlocked\ \wedge\ rightTurnBlocked))$. This tells us that there is no situation in the environment that forces an agent to halt forever, and further that there should be no legally reachable dead ends without the possibility to turn around.

Additional environment progress properties, 2-4 of table 4.5 states that the input variables representing the zones that can only show moving vehicles will eventually be unoccupied.

4.1.2 Navigation

The agent is also able to navigate to target locations and with that the agent has the ability to make turns. Indications for navigation are passed to the vehicle as the input variable *target*, with the possible values t_f, t_l and t_r, symbolizing that the target is located in front of the agent, to the left of the vehicle and to the right of the vehicle respectively. In front of the vehicle is here defined as within a 45 degree angle to the left and right against the lateral centerline, that is, all the space where a right

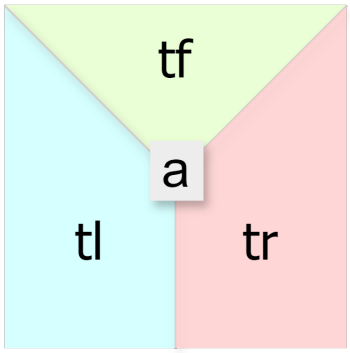


Figure 4.3: The target directions mapped around the agent.

angle turn would not lessen the degree between the vehicle lateral centerline and the line between the vehicle and the target. These target directions are visualized in 4.3.

Several target transitions can be defined, displayed as propositions 16-26 of table 4.3. These are environment safety properties expressing how a target direction might change after a specific movement. An example is $(move = m_{tl} \wedge target = t_f) \rightarrow next\ (target = t_r)$, which says that if the agent turns to the left when facing the target, the target will be located on the right side of the agent after the turn. Note that the limitation to right angle turns of the system, combined with the right angle of the target forward angle, gives that any turn from a position with the target in front of the agent would result in not facing the target in the next step.

With the introduction of a target direction, we state system safety propositions, 2-4 of table 4.4, intended to make the agent attempt to navigate towards the target. We denote these propositions as move preferences. Depending on the input target direction, different movements are able to bring the agent closer to the target. Best possible path can not be promised. A simple example is 3a of table 4.4, $target = t_l \rightarrow (\neg(olt \vee olff \vee olb \vee ofc)) \rightarrow move = m_t l$, which states that if the target is left of the agent and the path of a left turn is not blocked, the agent should turn left. These preferences rank all possible movement options. For example, presented as proposition 2c, $target = t_f \rightarrow (of \wedge \neg(orf \vee ofc)) \rightarrow move = m_{srf}$, if the path forward is blocked and the target is located forward, the agent should shift forward and to the right if possible. Further conjuncts of proposition 2 state that if a shift forward and to the left is not possible, the agent will prefer a shift forward and to the left and then a left turn.

Faced with the choice between multiple equally bad options, the agent will attempt to follow some blanket traffic laws, to synthesize consistent navigation preferences. The move preferences also indicate that the vehicle should always choose movement if possible. This can show problematic, as further explored in section 4.1.3.

Recall that the assumption on the environment states that there will always eventually occur a possible move, and that a space can not be indefinitely obstructed.

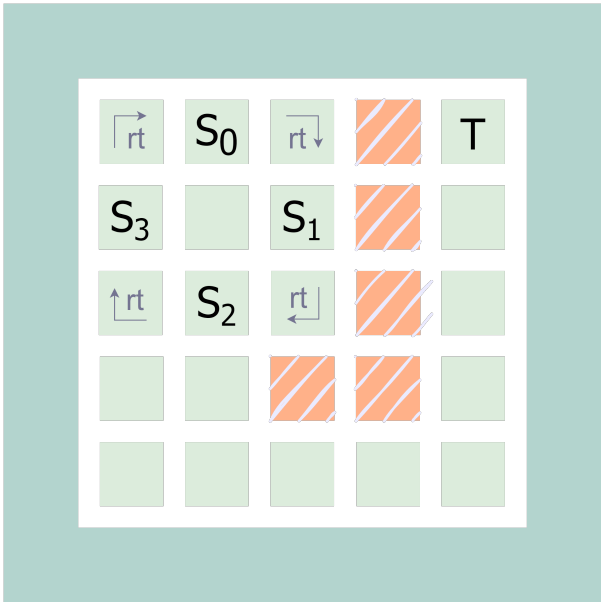


Figure 4.4: Example task where the system gets stuck in a loop. Movable space marked as a 5 times 5 grid of light squares. Hatched squares are runtime discovered obstacles. The agent starts in S_0 and the target is in T. The agent moves to S_1, S_2 and then S_3 through three right turns. It then repeatedly moves through S_0, S_1, S_2 to S_3. The vehicle would be able to operate to T, by making a left turn after S_3. However, at S_2 the target direction input is t_r, indicating that the target is on the right hand side of the agent and prompting a right turn output from the controller, which is accommodating the system safety property $target = t_r \rightarrow (\neg ort \rightarrow move = m_{tr})$, 4a of table 4.4.

The vehicle is not able to respond with any appropriate progression properties. The move preferences prompt the agent to move towards the target when possible, but the logical states are not defined as closer to or further from the target than any other, making guarantees about progression impossible with the variables of the current model.

4.1.3 Problematic behaviour

In some aspects the system do not operate as intended. It either does not follow common sense or traffic laws, or it fails to navigate towards its target. This section present some of this behaviour, labels the problems and explain possible reasons and solutions.

The model corrects trajectory in right angle turns, even though the direction of the target location is given in fan shape, 45 degrees off the centerline in both directions. The reason for the agent in this model only being able to make right angle turns is to limit the state space, since it would require a lot of additional environment variables to otherwise model the movement of obstacles before and after a turn. Under these conditions the target location inputs were designed. The classification of the target direction is due to the fact that a cruder definition of a correct course might lead to

issues navigating the last distance before the target, as well as the fact that generally, correcting the course proactively rather than reactively can be more efficient. However, this design means that if a target is located 45 degrees off the lateral centerline of the agent, in an open space the agent will correct its course by repeated turns, every other left and right. This will make the agent wiggle across the ideal path. An appropriate handling when taking left in a multi-file crossing would be to keep the path as close to a right angle as possible. This model, however, will cross back and forth over the diagonal between the agent and the target destination. This wiggle is a wasteful behaviour which may cause problems in interaction with other vehicles.

It is occasionally problematic that the agent will not wait on moving obstacles when there are other options available. This may lead to cumbersome detours. For example the agent might choose to take an alternative path instead of waiting on an opening for making a left turn. This problematic behaviour could be mitigated with a waiting turn, but there could still be situations where waiting multiple steps would be more reasonable than finding an alternative route, and yet the model would select an alternative route. Fully handling the issue would probably mean at least partially introducing separate variables indicating static versus moving obstacles. This adds to the state space explosion problem and in many aspects replicate the work of modeling movement over multiple time steps. It is such quite an expensive modification which is likely to be necessary when using the model for practical navigation tasks.

The model has no guaranteed progress towards the target and it shows to be problematic in multiple situations. Typically such problems occur when handling particular unusual arena layouts. The agent might become stuck in loops, where it will run in a circle indefinitely. An example layout can be seen in 4.4, where there is a possible path towards the target but the model is not able to find it. Instead it gets stuck in a loop. There are also situations where this looping behaviour also show for a model with a crude 180 degree definition of target forward, indicating the the zoning of target directions is not the issue. Unlike other models, such as the receding horizon control model, the agent-centric model does not guarantee any eventual progress towards the target. Instead of progressing, the vehicle might get stuck in a loop or move further away from the target. A progress property of the type "infinitely often we will move closer to the target" would be problematic since one may still under that condition more often move further from the target. A system safety property only allowing moves that bring the agent closer to the target would not allow for any non-direct paths. One may use partial targets to limit the extense of the problem, marking for example every crossing in a path, planned by an external path planner, as a partial target. The agent will then navigate to every partial target and upon reaching a partial target restart the controller with the target switched to the next on path. However, loops can still appear in some situations where a runtime discovered obstacle is larger than the care area of the agent. This remaining problem discussion is continued in section 5.3.

Table 4.1: Environment and system variables, constituting the input and output variables of the controller respectively.

type	name	value	description
env	olf	boolean	presence of obstacle in area where a left shift of the agent would cause the obstacle to risk collision with the agent the next time step
env	olff	boolean	presence of obstacle left and far forward of the agent moving laterally towards the agent
env	olb	boolean	presence of obstacle left and back of the agent moving laterally towards the agent
env	of	boolean	presence of obstacle forward of the agent
env	oa	boolean	presence of obstacle in immediate risk of collision with the agent
env	orf	boolean	presence of obstacle in area where a right shift of the agent would cause the obstacle to risk collision with the agent the next time step
env	ofc	boolean	presence of obstacle moving perpendicullarily to the trajectory of the agent in risk of collision with the agent the next time step
env	olt	boolean	presence of obstacle in area where a left turn of the agent would cause the obstacle to risk collision with the agent the next time step
env	ort	boolean	presence of obstacle in area where a right turn of the agent would cause the obstacle to risk collision with the agent the next time step
env	target	t_l, t_f or t_r	location of target expressed by angle towards current path of the agent: left, forward or right of agent
sys	move	m_{slf}, m_f, m_{srf}, m_h, m_{tl} or t_{tr}	output move of agent: shift left and forward, forward, shift right and forward, halt, turn left or turn right.

Table 4.2: Propositions defining the initial state of the environment of the agent-centric model.

#	proposition	description
1	$\neg\ oa$	there may not be an obstacle in unavoidable possible collision with agent

Table 4.3: Environment safety properties of the agent-centric model.

#[1]	proposition	description
1	$move = t_{slf} \rightarrow$	obstacle transitions of a forward shift move to the left
a	$((next\ oa \leftrightarrow (olf \vee olb \vee olff \vee ofc)) \wedge$	
b	$(of \leftrightarrow nextorf) \wedge$	
c	$((\neg olf \wedge olt) \rightarrow next \neg olb))$	
2	$move = t_f \rightarrow$	obstacle transitions of a forward move
a	$((of \vee ofc) \leftrightarrow next\ oa)$	
3	$move = m_{srf} \rightarrow$	obstacle transitions of a forward shift move to the right
a	$(((orf \vee ofc) \leftrightarrow next\ oa) \wedge$	
b	$(of \rightarrow next\ olf))$	
4	$move = m_h \rightarrow$	obstacle transitions of a halt move
a	$((olff \rightarrow next\ olf) \wedge$	
b	$(olb \rightarrow next\ olf) \wedge$	
c	$(next\ olf \rightarrow (olff \vee olf \vee olb)) \wedge$	
d	$(oa \leftrightarrow next\ oa) \wedge$	
e	$(of \leftrightarrow next\ of) \wedge$	
f	$(orf \leftrightarrow next\ orf)) \wedge$	
5	$move = m_{lt} \rightarrow$	obstacle transitions of a left turn move
a	$((olff \vee ofc \vee olt) \leftrightarrow next\ oa)$	
6	$move = m_{rt} \rightarrow$	obstacle transitions of a right turn move
a	$((ort \leftrightarrow next\ oa) \wedge$	
b	$((orf \wedge \neg ort) \rightarrow next\ olb))$	
7	$\neg(olff \wedge olb)$	no concurrent meeting and passing obstacles in left lane of agent
8	$((move = m_h \vee move = m_f) \wedge olff) \rightarrow next\ \neg olb$	no concurrent meeting and passing obstacles in left lane of agent
9	$((move = m_h \vee move = m_f) \wedge olb) \rightarrow next\ \neg olff$	no concurrent meeting and passing obstacles in left lane of agent
10	$(move = m_h \wedge olff \wedge olf) \rightarrow next\ (olf \wedge \neg olb)$	two obstacles may not simultaneously travel laterally backwards in its left lane
11	$(move = m_h \wedge olb \wedge olf) \rightarrow next\ (olf \wedge \neg olff)$	two obstacles may not simultaneously travel laterally forward in its left lane
12	$(move = m_f \wedge olf \wedge olb) \rightarrow next\ (olf \wedge \neg olff)$	two obstacles may not simultaneously travel laterally forward in its left lane
13	$(move = m_h \wedge olff \wedge olf) \rightarrow next\ (olf \wedge \neg olb)$	meeting two obstacles

[1] conjunct label, if applicable

#[1]	proposition	description
14	$(move = m_h \wedge olb \wedge olf) \rightarrow next\ (olf \wedge \neg olff)$	being passed by two obstacles
15	$(move = m_f \wedge olf \wedge olb) \rightarrow next\ (olf \wedge \neg olff)$	being passed by two obstacles
16	$move = m_h \rightarrow target = (next\ target)$	target is static when halting
17	$(move = m_f \wedge (target = t_l \vee target = t_r)) \rightarrow target = next\ target$	moving away from target will result in not facing the target
18	$((move = m_tl \wedge target = t_r) \vee (move = m_tr \wedge target = t_l)) \rightarrow ((next\ target) \neq t_f)$	moving away from target will result in not facing the target
19	$(move = m_{tr} \wedge (target = t_f)) \rightarrow (next\ target) = t_l$	turning when facing target will result in target changing
20	$(move = m_{tl} \wedge (target = t_f)) \rightarrow (next\ target) = t_r$	turning when facing target will result in target changing
21	$(move = m_{tr} \wedge (target = t_r)) \rightarrow (next\ target) \neq t_l$	turning towards the target will not move it to other side of you
22	$(move = m_{tl} \wedge (target = t_l)) \rightarrow (next\ target) \neq t_r$	turning towards the target will not move it to other side of you
23	$(move = m_{slf} \wedge target = t_f) \rightarrow (next\ target) \neq t_l$	shifting when target is in front will not result in target being on side you shifted towards
24	$(move = m_{srf} \wedge target = t_f) \rightarrow (next\ target) \neq t_r$	shifting when target is in front will not result in target being on side you shifted towards
25	$((move = m_{slf} \wedge target = t_l) \vee (move = m_{srf} \wedge target = t_r)) \rightarrow next\ (target \neq t_f)$	shifting along target front line will not move you to face target
26	$((move = m_{slf} \wedge target = t_r) \vee (move = m_{srf} \wedge target = t_l)) \rightarrow (target = next\ target)$	shifting away from target will keep target in same area

[1] conjunct label, if applicable

Table 4.4: System safety properties of the agent-centric model.

#[1]	proposition	description
1	$\neg oa$	agent may not be in unavoidable possible collision with obstacle
2	$target = t_f \rightarrow$	if target is forward of agent:
a	$(\neg(of \vee ofc) \rightarrow move = m_f) \wedge$	if no forward blockage, agent moves forward
b	$(((of \vee ofc) \wedge \neg of) \rightarrow move = m_h) \wedge$	if forward blockage is guaranteed temporary, agent waits
c	$((of \wedge \neg(orf \vee ofc)) \rightarrow move = m_{srf}) \wedge$	else if no right forward blockage, agent makes right forward shift move
d	$((of \wedge (orf \vee ofc) \wedge \neg orf) \rightarrow move = m_h) \wedge$	else if right forward blockage is guaranteed temporary, agent waits
e	$((of \wedge orf \wedge \neg(olf \vee olff \vee olb \vee ofc)) \rightarrow move = m_{slf}) \wedge$	else if no left forward blockage, agent makes left forward shift move
f	$((of \wedge orf \wedge (olf \vee olff \vee olb \vee ofc) \wedge \neg olf) \rightarrow move = m_h) \wedge$	else if left forward blockage is guaranteed temporary, agent waits
g	$((of \wedge olf \wedge orf \wedge \neg(olt \vee olff \vee olb \vee ofc)) \rightarrow move = m_{tl}) \wedge$	else if no left blockage, agent turns left
h	$((of \wedge olf \wedge orf \wedge (olt \vee olff \vee olb \vee ofc) \wedge \neg olt) \rightarrow move = m_h) \wedge$	else if left blockage is guaranteed temporary, agent waits
3	$target = t_l \rightarrow$	if target is left of agent:
a	$((\neg(olt \vee olff \vee olb \vee ofc) \rightarrow move = m_{tl}) \wedge$	if no left blockage, agent turns left
b	$(((olt \vee olff \vee olb \vee ofc) \wedge \neg olt) \rightarrow move = m_h) \wedge$	else if left blockage is guaranteed temporary, agent waits
c	$((olt \wedge \neg(olf \vee olff \vee olb \vee ofc)) \rightarrow move = m_{slf}) \wedge$	else if no left forward blockage, agent makes left forward shift move
d	$((olt \wedge (olf \vee olff \vee olb \vee ofc) \wedge \neg olf) \rightarrow move = m_h) \wedge$	else if left forward blockage is guaranteed temporary, agent waits
e	$((olt \wedge olf \wedge (\neg(of \vee ofc))) \rightarrow move = m_f) \wedge$	else if no forward blockage, agent moves forward

[1] conjunct label, if applicable

#[1]	proposition	description
f	$((olt \wedge olf \wedge (of \vee ofc) \wedge \neg of) \rightarrow move = m_h))$	else if forward blockage is guaranteed temporary, agent waits
4	$target = t_r \rightarrow$	if target is right of agent:
a	$((\neg ort \rightarrow move = m_{tr}) \wedge$	if no right blockage, agent turns right
b	$((ort \wedge \neg(orf \vee ofc)) \rightarrow move = m_{srf}) \wedge$	else if no right forward blockage, agent makes right forward shift move
c	$((ort \wedge (orf \vee ofc) \wedge \neg orf) \rightarrow move = m_h) \wedge$	else if right forward blockage is guaranteed temporary, agent waits
d	$((ort \wedge orf \wedge \neg(of \vee ofc)) \rightarrow move = m_f) \wedge$	else if no forward blockage, agent moves forward
e	$((ort \wedge orf \wedge (of \vee ofc) \wedge \neg of) \rightarrow move = m_h))$	else if forward blockage is guaranteed temporary, agent waits

[1] conjunct label, if applicable

Table 4.5: Environment progress properties of the agent-centric model.

#[1]	proposition	description
1	$\neg$	infinitely often, at least one move must be possible, and such one of the following conjuncts must be false
a	$((olf \vee olff \vee olb \vee ofc) \wedge$	left forward shift move blocked
b	$(of \vee ofc) \wedge$	forward move blocked
c	$(orf \vee ofc) \wedge$	right forward shift move blocked
d	$(olt \vee olff \vee olb \vee ofc) \wedge$	left turn move blocked
e	$ort)$	right turn move blocked
2	$\neg olff$	lff space can not be permanently occupied by moving objects
3	$\neg olb$	lb space can not be permanently occupied by moving objects
4	$\neg ofc$	fc space can not be permanently occupied by moving objects

[1] conjunct label, if applicable

5

Conclusion

Taken together, this project has resulted in a vehicle specification quite different from conventional space-centric models. This final chapter will attempt to discuss the possible benefits and disadvantages of working in this new perspective.

5.1 Domain-specific model

The resulting model is very specialised on its exact use-case, a simplified trial for a traffic vehicle of navigation and obstacle avoidance. The large amount of formulas describing the relations between different the actions of the environment and the system over time, the transitions, shown in tables 4.3 and 4.4, follow some certain patterns. Possibly some of these implications could be automatically generated, but context-based coverage and the unsymmetrical zone model means that many are very domain-specific. If the capacity of the size of the input specification of the synthesis process would increase, allowing for more complex models, defining this type of model might be overburdening.

The characteristic unsymmetrical zone mapping, with the lateral boundaries of the fields overlapping, was designed to limit the state space enough for the limited computing resources available for the synthesis process. Still, multiple qualities of this approach became apparent while working on this project

This mapping is supposed to limit the amount of variables needed to model the surroundings by defining the zones by how an obstacle inside the zone affects the vehicle. To be able to predict future implications of differently placed obstacles, additional partition is necessary. This increases the amount of spatial zones further and it approaches that of a grid partition.

This very domain-specific zoning complicates redesign and continuous development of the model. As it describes the situation of the vehicle more of its surroundings, it risks answering the same question it poses, becoming less relevant as a proved safe model. The chosen remediation of the limitation to the model size dominated much of the work of this project. This was not the intention of this project, and might limit the relevance of the resulting model.

5.2 Offline synthesis

Alternative state space-reducing methods usually come with some drawbacks. The method of state space reduction usually includes breaking the task down to multiple smaller tasks, which might require a large working memory, large storage facilities or fast computing powers. It might also require for some calculations to be performed during runtime and locally in the agent vehicle computer, increasing performance needs and demanding access to reliable computing and communication. The agent-centric model presented in this thesis does not need to perform live synthesis, neither does it have to carry many preloaded controllers for different occasions. This may have several positive ramifications.

A relevant comparison can be done with the receding horizon control model presented in [15]. It reduces the state space of the synthesis process by having the logical controller navigating short distances between mapped road cross sections and delegates finding the path between those cross sections to a navigational tool above the logical controller. This upper level navigational tool defines the closeness to the target for each road cross section and compares cross sections as further and closer to the target. It preplans the full journey by mapping a path between cross sections to the target, checking if a subjourney to a location closer to the target is realisable from every possible discretised location in the arena but only realises a specification for a couple of cross sections ahead, live during the mission when that particular journey is topical.

The receding horizon model synthesises new controllers frequently, at most after every move. This puts several demands on the capacity of the agent computer. First, it needs to have the working memory and calculating power to perform the synthesis. Second, it needs to have the memory to store a large amount of specifications, a space growing with the size of the arena. Third, this computer needs to be reliable - if the vehicle is not able to perform the synthesis, the journey can not continue.

Contrariwise the agent-centric model does not need to perform live synthesis. This means that one can use smaller working memory, storage memory and computing power for the use of the logical controller in the vehicle computer, since it has to carry a single logical machine but not many short horizon specifications, and, because the only computation it needs to perform is to use the lookup functionality of the logical controller.

However, in order for the agent-centric model to obtain potentially better fail recovery than the receding horizon model, as elaborated upon in sections 5.3 and 5.4, it needs to have reliable access to a GPS navigation tool. If not, it will not be able to correct its course receive the benefits further explained in section 5.4.

5.2.1 Scaling

Using specific controllers for different situations unavoidably means that space and power needed on- and offline will scale with the number of possible situations. In the case of the receding horizon controller, that is every added location to the discretised arena. Contrastingly, the agent-centric model uses a single controller. It does not have to store many controllers and hence the storage space needed during a live mission does not scale with the size of the arena.

Considering the scaling of the receding horizon controller, it is also notable that the horizon length may scale with the arena, simply based on an increase likelihood of a complex traffic situation appearing.

Imagine a very large arena. The oddest traffic solution appearing inside of that arena may define the horizon length for every short horizon specification of the solution. The receding horizon model, as a model that shrink the state space by navigating to some location deemed closer to the target, must always have a model large enough to include all connected locations of equal closeness, as well as all locations in the area noted as following in closeness order. This works badly with for example large open spaces, where many locations could have the same closeness ranking. One can not simply reduce the amount of locations in the same group during preplanning, since it is impossible to know where the agent will enter and which locations that will be occupied by obstacles.

The possible length of a path is what controls the horizon length, the possible length of a path to a closer location[2]. The horizon length is a factor of the computational complexity. Even if it only concerns a part of the arena, if it is known that certain areas will raise the computational complexity enough for it to be impossible for the agent computer to calculate a trajectory in time, it could be dangerous to enter the vehicle into traffic.

It may be necessary to view also this problem as a scaling issue. One might want to model a very long journey or use a very large map. Then it is more likely that one encounters a relatively unusual traffic situations, with a flat division of closeness ranking of the locations.

Of course, scaling the problem also raises the risk of running into the typical problematic situations of the agent-centric controller. Some of those will be elaborated upon in the next section, which focuses on the method of navigation.

5.3 Uncoupling the navigation from the map

The target navigation used in this project brings about its own challenges and potential benefits, something that will be elaborated on in this section. For this section, imagine placing partial targets at the center of cross sections of the road, as frequently as the cross sections of the receding horizon model. This allows a com-

parison of the both models on other factors than the placement of targets, a work performed by other components than the logical controller. By these rules, the agent-centric model will place its targets as in figure 5.3a on a network sectioned by the receding horizon control model as in figure 5.2a. In short, while both models uses the same checkpoints, one associates the target with the space that is also used for marking obstacles, something this section will refer to as absolute spatial navigation, and the other indicates the direction of the target, hereforth called directional navigation. Having clarified the placement of subtargets and the opposing models for navigating between them, it is time to look on a couple of particular traffic situations.

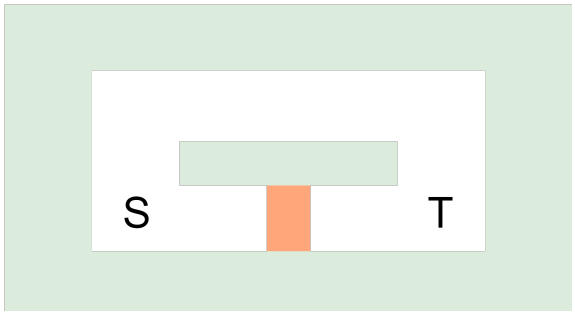


Figure 5.1: Simple map displaying the case of two rooms with two paths between them. Start location is marked s and target t. For this case, the southern passade is the shorter route as found by both the agent-centric model and the receding horizon control model. As only discovered when one has already entered the southern passage, it is blocked by an unexpected obstacle, rendering the northern passage part of any possible route between s and t. This is only solved by the receding horizon model, which is able to find a new path through the northern passage.

First, note the situation in figure 5.1 where a path is completely blocked by static obstacles, necessitating the use of an alternative route. Here, the logical controller of the receding horizon control model, using a spatial navigation model, fails since the assumption that there will be clear space in the target fails. As soon as it has observed that all space in the cross section is occupied, the inputs to the logical controller will not match an output. As the logical controller crashes, the high level route planner will find an alternative path and the agent will smoothly begin the alternative journey. This failure of the logical controller is desirable, since it means that model successfully recognises a blocked path as an illegal situation, guaranteeing that the vehicle will not get stuck attempting to solve an impossible problem.

The agent-centric however, is not able to inherently notice the lack of a passable path. The rather weak assumption on the environment here only gives that a possible move will eventually appear. However, the issue here is not that there are no possible moves, it is that the agent is not able to make new moves that progresses it towards the target. Although this logical model will never acknowledge it, the agent gets stuck in a loop.

This points towards a general problem with the navigational model of this project.

Since the target is not logically related to the same space which holds the existence of obstacles, the logical controller can not notice the inability of the environment to provide a path directly towards the target. For it to be able to handle problems like unmapped blockages, it either needs inputs that bridges the relationships of obstacles and the path to the target, or a higher level controller needs to discover the problem and interrupt the run. The receding horizon model shows an obvious advantage at recognising this type of situations where it is appropriate to fail and replan the mission.

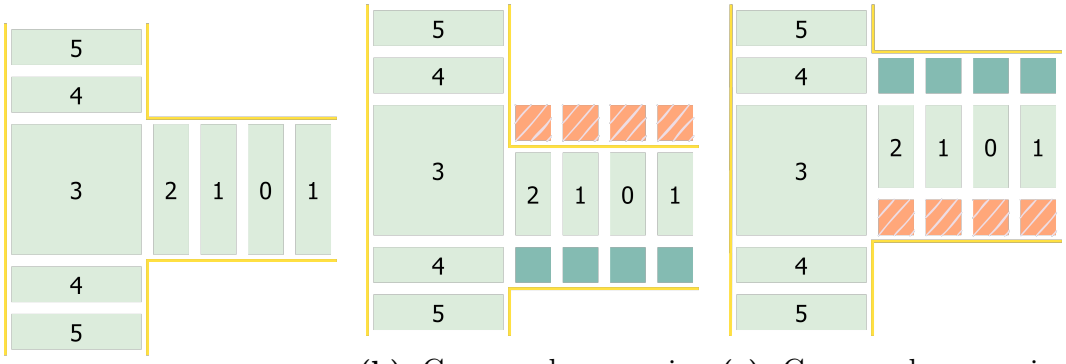
One may also accept this behaviour as a drawback of the model. With a large enough space mapped to spatial inputs, one might accept the possible unnecessary failures and reject inputs where progression in the mapped space can not be guaranteed.

However, due to how the spatial inputs of the agent follows it onto offroad area, handling the same complex obstacles may require a larger set of space segments in a directional navigational model than in a spatial navigational model. That is, to be able to discover an equal width blockage, the spatial inputs of the agent-centric model will have to cover a greater perpendicular width than the absolute spatial navigation, since it must be able to cover the width of the road even when its located to the very left or right of the road, where all of its spatial inputs to the right or left respectively are "wasted" indicating offroad space.

This is a clear drawback for the directional navigation. Are there any benefits?

Well, strong reliance on preplanning tightly coupled to a specific mapping makes for a model that adapts badly to unexpected discrepancies between the map and reality. Imagine that a marked cross road appears later than expected from the given map which one has preplanned for, either because of faulty mapping or because an undiscovered faulty transition of the agent vehicle. An example can be seen in 5.2. Here, we can see how an agent might perceive less usable space than what is actually available because of the preplanning not adjusting to the real road circumstances. The model can handle unexpected obstacles, and given that it can perceive offroad space and obstacles as somewhat equivalent space, will likely treat the irregularities between the mapping and the reality as unexpected obstacles. Depending on the severity of the faulty mapping, and circumstances of other unexpected obstacles, this may or may not result in a rerouting of the vehicle due to a road being incorrectly perceived as blocked, or render the task impossible.

This may be considered a general problem of models that use preplanning on a mapped traffic network which can never be properly synced during runtime. As the specification itself depend on the concrete mapping, and a discretisation definitively declares the relationships between specific parts of the network, the plan becomes rather rigid and difficult to adapt, even if some component of the agent can observe mismatches between the internal model and the environment live and has the capacity to attempt to correct the map. The difficulty is that an entire network needs



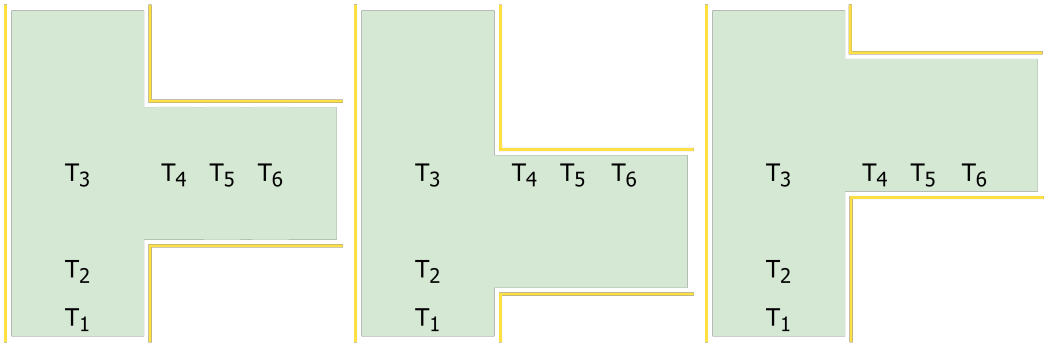
(a) Crossroad as mapped. **(b)** Crossroad appearing earlier than expected. **(c)** Crossroad appearing later than expected.

Figure 5.2: Example of the consequences of preplanning mapping mismatching reality, displaying a receding horizon control model specified on a bad mapping of a road network, where a crossroad does not appear as expected. The yellow frame mark the actual placement of the road. The numbered blocks are road sections of the mapping used for preplanning. The numbers indicate the closeness ranking in the ordering of the road cross sections. The dark green space is useable road space mistakenly marked as offroad. The orange striped space is offroad space mistakenly interpreted as road space but blocked, hence perceived as containing unexpected obstacles.

to be correctly remapped. If a crossroad appear sooner than expected, should the map shift the entire intersecting road backwards? Is the main road shorter than expected? Or is the network correct but the output movement of the agent been incorrectly observed? There is an infinite amount of possible corrections. It is likely that a few mapping errors will lead to repeated attempts at correction from the agent, while it might fail or move between multiple different faulty routes.

While one might expect for conventional global traffic maps to precisely locate most important traffic networks in large cities, less traveled and more remote networks are likely to have more frequently occuring and more severe mapping errors.Consequently, rural navigation might be a problematic domain for models requiring very precise maps. In some areas of use, this may show very problematic. Conceptualise a task in accident recovery in an area affected by a natural disaster. With the dangers of rescue missions and other work, this is a great use case for automatic vehicles. But due to previously subpar mapping or distortments from the disaster, it is likely that the placement of paths and their intersections are only rough estimations. This is a clear situation where one can not rely on a method that requires preplanning on a precise map.

An directional navigational model may adapt and for example dynamically place the partial targets at recognised intersections. This is not something implemented in this project, but may be a subject for future research. Because the logical controller is not aware of the actual location of a target, it can be moved during runtime. An external module might hence correct the map during the task. Within certain



(a) Crossroad as mapped. **(b)** Crossroad appearing earlier than expected. **(c)** Crossroad appearing later than expected.

Figure 5.3: Example of placing partial targets at cross sections of the road. As visible in the figure, a small offset between the map and reality does not limit the movements of the agent. It does of course shift the probable path towards the side of the road and a larger offset than shown in this figure might require an adjustment of the relationships between the partial targets. However with directional targets this adjustment is not on the level of the logical controller.

constraints a move will not result in a failure of the controller, beyond those a simple restart of the controller would be needed. Its decoupled nature may make it more suitable for tasks with many unknown factors.

Concluding this section, there is good reason to keep the directional navigation model in mind. To mitigate its problems with recognising dead ends, there are possibly two different tracks of modification of the model. One can attempt to modify the logical model by providing additional environmental inputs that relate obstacle placement to the target direction. This might negatively affect the adaptation skills discussed in the later part of this section. Another option is to delegate the identification of issues and modification of the task to a component outside of the logical controller. For example, an upper level controller can move the target after detecting that the agent is returning to a location. Finally, one can accept the limitations of the model and work to increase its capacity to more liken that of absolute spatial navigation. This solution might be some type of hybrid of agent-centric and absolute modelling.

5.4 Fail recovery

As previously discussed, a mismatch between the logical model and physical reality, resulting in a failure of the controller, might be desired or undesired depending on the context.

Section 5.2 mentioned that it may be beneficial to not need reliable access to a method of synthesis on vehicle during runtime. Reliable access to synthesis would for the vehicle either require dedicated calculating power on vehicle, or reliable

communication to a remote computer with that dedicated calculating power. Such communication would probably limit a vehicle to staying close range to a reliable wireless network, or using satellite internet. When the agent-centric model fails, the controller does not need to perform any heaving computing, but it should update its location against GPS navigation. Otherwise, if it is in a safe situation, which translates into a legal initial state, it can just restart the controller.

The agent-centric model will only fail at inconsistent environment or system behaviour that might change the implications for the system. If a sole obstacle stays in the zone in front of the agent, it does not matter whether both vehicles are unexpectedly far to the left in the lane. If the obstacle were to swerve farther than the agent, it might enter another zone. Upon this, a failure will be appropriate as the reality has changed enough for the obstacle to have other possible implications for the agent. A failure is initiated, on the grounds that the mismatch affects the agent.

5.5 Future research

There are multiple possible areas of interest for further research. Development of the agent-centric model should focus on resolving the issues critical to solving the mission, find quantifiable measures of performance to enable comparisions with other vehicle specifications and find further apt subjects of abstraction to limit the state space of the synthesis process. Future work may also look further at dynamic definitions of input and output variables.

The simulator uses a discrete grid of locations, complicating exploration of disturbance and interruptions. Further, the simulator is unable to perform complex route planning, and uses a single target at a predetermined destination. A better simulator would enable concrete experiments on the theoretic advantages of the model discussed in sections 5.3 and 5.4.

The model needs to perform more consistently traffic-appropriate. Previously discussed behaviour that needs to be addressed include "the wiggle", choosing the diagonal in open spaces like traffic crossings and not waiting for an opening in busy traffic.

5.6 Conclusion

In conclusion, this project has resulted in a novel type of modelling for logical specifications of controllers for autonomous vehicles. This model have some interesting properties that need to be further researched, but already presents some benefits in comparision to other models for similar purposes.

The agent-centric model demands less of the physical system than other models. It fails less, for better and worse. It is not as vulnerable to mismatches between its

logical model and the physical reality. It is unfortunately not able to discover and fail at some impassable obstacles.

Its failures are less problematic than those of other models for similar purposes, since a restart does not require live synthesis.

Future research may develop improved agent-centric models. These should work to alleviate issues with failing when appropriate, to find quantifiable measures to compare it to other models and to further improve the performance by abstract model variables.

Bibliography

[1] J. Krook, "Correct-by-construction tactical planners for automated cars," 2019.

[2] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Transactions on Automatic Control*, vol. 57, no. 11, pp. 2817–2830, 2012.

[3] M. Y. Vardi, "An automata-theoretic approach to linear temporal logic," *Logics for concurrency: structure versus automata (8th Banff Higher Order Workshop Proceedings)*, pp. 238–266, 1995, Digitalized 2005. DOI: https://doi.org/10.1007/3-540-60915-6_6.

[4] E. M. Clarke, T. A. Henzinger, H. Veith, R. Bloem, *et al.*, *Handbook of model checking*. Springer, 2018, vol. 10.

[5] Z. Ramezani, J. Krook, Z. Fei, M. Fabian, and K. Akesson, "Comparative case studies of reactive synthesis and supervisory control," in *2019 18th European Control Conference (ECC)*, IEEE, 2019, pp. 1752–1759.

[6] M. Fält, V. Raman, and R. M. Murray, "Variable elimination for scalable receding horizon temporal logic planning," in *2015 American Control Conference (ACC)*, IEEE, 2015, pp. 1917–1922.

[7] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, ieee, 1977, pp. 46–57.

[8] N. Piterman and A. Pnueli, "Temporal logic and fair discrete systems," *Handbook of Model Checking*, pp. 27–73, 2018.

[9] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive (1) designs," in *Verification, Model Checking, and Abstract Interpretation: 7th International Conference, VMCAI 2006, Charleston, SC, USA, January 8-10, 2006. Proceedings 7*, Springer, 2006, pp. 364–380.

[10] R. Bloem, K. Chatterjee, and B. Jobstmann, "Graph games and reactive synthesis," *Handbook of model checking*, pp. 921–962, 2018.

[11] U. Klein and A. Pnueli, "Revisiting synthesis of gr (1) specifications," in *Haifa Verification Conference*, Springer, 2010, pp. 161–181.

[12] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1989, pp. 179–190.

[13] O. Kupferman and M. Y. Vardi, "Safraless decision procedures," in *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, IEEE, 2005, pp. 531–540.

[14] E. S. Kim, M. Arcak, and S. A. Seshia, "Compositional controller synthesis for vehicular traffic networks," in *2015 54th IEEE Conference on Decision and Control (CDC)*, 2015, pp. 6165–6171. DOI: 10.1109/CDC.2015.7403189.

[15] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning for dynamical systems," in *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, IEEE, 2009, pp. 5997–6004.

[16] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray, "Tulip: A software toolbox for receding horizon temporal logic planning," in *Proceedings of the 14th international conference on Hybrid systems: computation and control*, 2011, pp. 313–314.

A

Full specification

This is the full Python source file synthesised to a logical controller by TuLiP. Note that technical limitations of the synthesis tool lead to us having to use another syntax for the Python source file. Here, X is equivalent to *next*, && to $\wedge$, || to *lor* and ! to $\neg$.

```
from tulip.spec import GRSpec

def leftTurnBlockedSpec() -> GRSpec:
env_vars = {
    # static, oncoming and overtaking obstacles
    # to the left and slightly forward of the agent
    'olf':   'boolean',
    # oncoming obstacles to the left
    # and further forward of the agent
    'olff':  'boolean',
    # overtaking obstacles to the left
    # and besides and slightly behind the agent
    'olb':   'boolean',

}

def targetOrientationSpec() -> GRSpec:
env_vars = {
    # obstacle occupancies
    # static, oncoming and overtaking obstacles
    # to the left and slightly forward of the agent
    'olf':   'boolean',
    # oncoming obstacles to the left
    # and further forward of the agent
    'olff':  'boolean',
    # overtaking obstacles to the left
    # and besides and slightly behind the agent
    'olb':   'boolean',
    # static obstacles in front of the agent
    'of':    'boolean',
    # obstacles colliding or risking collision with the agent
    'oa':    'boolean',
```

```
    # static obstacles to the right and slightly forward of the agent
    'orf':   'boolean',

   # obstacles with paths risking to perpendicularily
   # cross forward or left movement of the agent
    'ofc':   'boolean',

    # obstacles that prevents a left turn of the agent
    'olt':   'boolean',
    # obstacles that prevents a right turn of the agent
    'ort':   'boolean',

    # orientation of the target location: left, forward or right
    'target': [
        't_l',
        't_f',
        't_r',
        # BUG in tulip, should not exist:
        # https://github.com/tulip-control/tulip-control/issues/238
        't',
    ],
}

sys_vars = {
    'move': [
        'm_slf',
        'm_f',
        'm_srf',
        'm_h',
        'm_tl',
        'm_tr',
    ],
}

env_init = {
    # does not start in an obstacle
    '! oa',
}

sys_init = {}

leftForwardObstaclesTransitions = [
    '((X oa) <-> (olf || olb || olff || ofc))',
    '(of <-> X orf)',
    '(((! olf) && olt) -> (X (! olb)))',
```

```
]
forwardObstaclesTransitions = [
    '((of || ofc) <-> X oa)',
]
rightForwardObstaclesTransitions = [
    '((orf || ofc) <-> X oa)',
    '(of -> X olf)',
]
haltObstaclesTransitions = [
    '(olff -> X olf)',
    '(olb -> X olf)',
    '((X olf) -> (olff || olf || olb))',
    '(oa <-> (X oa))',
    '(of <-> X of)',
    '(orf <-> X orf)',

]

leftTurnObstaclesTransitions = [
    '((olff || ofc || olt) <-> X oa)',
]
rightTurnObstaclesTransitions = [
    '(ort <-> X oa)',
    '((orf && (! ort)) -> X (! olb))',
]

env_safe = {
    # obstacles moving in unison
    f'(move = "m_slf") -> ({' && '.join(leftForwardObstaclesTransitions)})',
    f'(move = "m_f") -> ({' && '.join(forwardObstaclesTransitions)})',
    f'(move = "m_srf") -> ({' && '.join(rightForwardObstaclesTransitions)})',
    f'(move = "m_h") -> ({' && '.join(haltObstaclesTransitions)})',
    f'(move = "m_tl") -> ({' && '.join(leftTurnObstaclesTransitions)})',
    f'(move = "m_tr") -> ({' && '.join(rightTurnObstaclesTransitions)})',


    # moving obstacles can not merge
    # not meeting and passing at the same time
    '! (olff && olb)',
    '((move = "m_h" || move = "m_f") && olff) -> X (! olb)',
    '((move = "m_h" || move = "m_f") && olb) -> X (! olff)',
    # meeting two obstacles
    '((move = "m_h") && olff && olf) -> X (olf && (! olb))',
    # being passed by two obstacles
    '((move = "m_h") && olb && olf) -> X (olf && (! olff))',
    '((move = "m_f") && olf && olb) -> X (olf && (! olff))',
```

```
    # some aspects of the target transitions
    # are known even in this simple discretization
    # target is static when halting
    '(move = "m_h") -> (target = (X target))',

    # moving away from target will result in not facing the target
    '((move = "m_f") && (target = "t_l" \
        || target = "t_r")) -> (target = X target)',
    '((move = "m_tl" && target = "t_r") \
        || (move = "m_tr" && target = "t_l")) -> ((X target) != "t_f")',

    # turning when facing target will result in target changing
    '((move = "m_tr") && (target = "t_f")) -> ((X target) = "t_l")',
    '((move = "m_tl") && (target = "t_f")) -> ((X target) = "t_r")',

    # turning towards the target will not move it to other side of you
    '((move = "m_tr") && (target = "t_r")) -> ((X target) != "t_l")',
    '((move = "m_tl") && (target = "t_l")) -> ((X target) != "t_r")',

    # shifting when target is in front will not
    # result in target being on side you shifted towards
    '((move = "m_slf") && (target = "t_f")) -> ((X target) != "t_l")',
    '((move = "m_srf") && (target = "t_f")) -> ((X target) != "t_r")',

    # shifting along target front line will not move you to face target
    '(((move = "m_slf") && (target = "t_l")) \
        || ((move = "m_srf") && (target = "t_r"))) -> (X (target != "t_f"))',

    # shifting away from target will keep target in same area
    '((move = "m_slf" && target = "t_r") \
        || (move = "m_srf" && target = "t_l")) -> (target = (X target))',
}

leftForwardShiftBlocked = '(olf || olff || olb || ofc)'
forwardBlocked = '(of || ofc)'
rightForwardShiftBlocked = '(orf || ofc)'
leftTurnBlocked = '(olt || olff || olb || ofc)'
rightTurnBlocked = 'ort'

pathExistsGuaranteed = f'(! \
    ({leftForwardShiftBlocked} \
    && {forwardBlocked} \
    && {rightForwardShiftBlocked} \
    && {leftTurnBlocked} \
```

```
    && {rightTurnBlocked}))'

# if multiple paths are available, agent attempts to approach target
movePrefsTargetForward = [
    f'((! {forwardBlocked}) -> (move = "m_f"))',
    # if it is a temporary block, we wait for a path to open up
    f'(({forwardBlocked} && (! of)) -> (move = "m_h"))',
    # if target is forward, one should keep right
    f'((of && (! {rightForwardShiftBlocked})) -> (move = "m_srf"))',
    f'((of && {rightForwardShiftBlocked} && (! orf)) -> (move = "m_h"))',
    f'((of && orf && (! {leftForwardShiftBlocked})) -> (move = "m_slf"))',
    f'((of && orf && {leftForwardShiftBlocked} && (! olf)) -> (move = "m_h"))',
    # or turn left
    f'((of && olf && orf && (! {leftTurnBlocked})) -> (move = "m_tl"))',
    f'((of && olf && orf && {leftTurnBlocked} && (! olt)) -> (move = "m_h"))',
]
movePrefsTargetLeft = [
    f'((! {leftTurnBlocked}) -> move = "m_tl")',
    f'(({leftTurnBlocked} && (! olt)) -> move = "m_h")',
    f'((olt && (! {leftForwardShiftBlocked})) -> move = "m_slf")',
    f'((olt && {leftForwardShiftBlocked} && (! olf)) -> move = "m_h")',
    f'((olt && olf && (! {forwardBlocked})) -> move = "m_f")',
    f'((olt && olf && {forwardBlocked} && (! of)) -> move = "m_h")',
]
movePrefsTargetRight = [
    f'((! {rightTurnBlocked}) -> move = "m_tr")',
    f'((ort && (! {rightForwardShiftBlocked})) -> move = "m_srf")',
    f'((ort && {rightForwardShiftBlocked} && (! orf)) -> move = "m_h")',
    f'((ort && orf && (! {forwardBlocked})) -> move = "m_f")',
    f'((ort && orf && {forwardBlocked} && (! of)) -> move = "m_h")',
]


sys_safe = {
    # no collision
    '! oa',

    # towards the target
    f'((target = "t_f") -> ({' && '.join(movePrefsTargetForward)}))',
    f'((target = "t_l") -> ({' && '.join(movePrefsTargetLeft)}))',
    f'((target = "t_r") -> ({' && '.join(movePrefsTargetRight)}))',

}

pathWillExist = [
    pathExistsGuaranteed,
```

```
    # space can not be permanently occupied by moving objects
    '(! olff)',
    '(! olb)',
    '(! ofc)',
]

env_prog = pathWillExist
sys_prog = {}

spec = GRSpec(env_vars, sys_vars, env_init, sys_init,
              env_safe, sys_safe, env_prog, sys_prog)

return spec
```