Trajectory Generation under Metric Interval Temporal Logic Specifications

DANIEL MÜLLER
Trajectory Generation under Metric Interval Temporal Logic Specifications

Master Thesis

Daniel Müller
dmul@kth.se
Automatic Control Department, EES
Electrical Engineering, KTH

Supervisor: Sofie Andersson
Examiner: Prof. Dimos Dimarogonas
Abstract

Metric interval temporal logic (MITL) provides a tool to formulate high level tasks in an easy way. This allows to specify behaviour of a dynamical system in a given environment which lies beyond the scope of simple stabilization. Finding an input of the system which satisfies the MITL formula, demands new techniques and algorithms. In this thesis, a novel approach is presented which abstracts the dynamical system into a time optimal weighted transition system (WTS) and converts the MITL formula into a Timed Büchi Automaton (TBA). From the graph product of the WTS and TBA a sequence of environment states together with time constraints is obtained. Together with a user specified cost function, the sequence is translated into an optimization problem. The solution of this final optimization problem satisfies the MITL formula for the dynamical system in the given environment and is obtained by using methods from optimal control.

Sammanfattning

## Contents

1 **Introduction**  
   1.1 Motivation  1  
   1.2 Related Work  1  
   1.3 Contribution  2  

2 **Problem Definition**  
   2.1 System Dynamics  3  
   2.2 Environment  3  
   2.3 Metric Interval Temporal Logic  4  
   2.4 Time Constraints and Run  5  
   2.5 Problem Statement  6  

3 **Solution Approach**  
   3.1 Solution of Problem 1  7  
   3.1.1 Weighted Transition System  7  
   3.1.2 Choosing the Points of the WTS  8  
   3.1.3 Inputs and Weights of the WTS  10  
   3.1.4 Theoretical Property of the WTS  13  
   3.1.5 Timed Büchi Automaton  15  
   3.1.6 Building the Graph and Graph Search  17  
   3.2 Solution of Problem 2  26  
   3.2.1 Formulation of the Optimization Problem  26  
   3.2.2 Computational Approach  27  
   3.2.3 Initial Guess for the Linear Case  30  

4 **Robustification Approach**  
   4.1 Controller  33  
   4.2 Minimal Distance  35  

5 **Extension to the Multi-Agent Case**  
   5.1 Problem Definition  37  
   5.2 Solution Approach  38  
   5.2.1 Building the Graph and Graph Search  38  
   5.2.2 Optimization Problem  40  

6 **Case Study**  
   6.1 Single Agent Case  41  
   6.1.1 Problem  41  
   6.1.2 Solution  42  
   6.1.3 Results  45  
   6.1.4 Theoretical Property  50  
   6.2 Nonlinear Single Agent  50  
   6.2.1 Problem  50  
   6.2.2 Results  51  
   6.3 Multi Agent Case  53  
   6.3.1 Problem  53  
   6.3.2 Results  54  
   6.3.3 Lower Interval Constraint  54  
   6.4 Multiple Quadcopter  57
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Why points of the WTS have to be on the edge</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>Optimizing over the whole path vs Optimizing part by part</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>Environment for Example 3.1</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>Resulting WTS for Example 3.1</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>TBA for Example 3.3</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>Büchi WTS for Example 3.3</td>
<td>24</td>
</tr>
<tr>
<td>7</td>
<td>Double checking lower constraints for Algorithm 2</td>
<td>25</td>
</tr>
<tr>
<td>8</td>
<td>Signal flow diagram</td>
<td>33</td>
</tr>
<tr>
<td>9</td>
<td>Robust environment state</td>
<td>36</td>
</tr>
<tr>
<td>10</td>
<td>Environment case study 1</td>
<td>42</td>
</tr>
<tr>
<td>11</td>
<td>TBA case study 1</td>
<td>43</td>
</tr>
<tr>
<td>12</td>
<td>Solution case study 1</td>
<td>46</td>
</tr>
<tr>
<td>13</td>
<td>Disturbed solution case study 1 room 2</td>
<td>47</td>
</tr>
<tr>
<td>14</td>
<td>Disturbed solution case study 1 room 6</td>
<td>48</td>
</tr>
<tr>
<td>15</td>
<td>Influence of a more precise optimization on the disturbed trajectory in case study 1</td>
<td>49</td>
</tr>
<tr>
<td>16</td>
<td>Extra points of the extended WTS satisfying Theorem 1 in case study 1</td>
<td>50</td>
</tr>
<tr>
<td>17</td>
<td>Environment case study 2</td>
<td>51</td>
</tr>
<tr>
<td>18</td>
<td>WTS solution in case study 2</td>
<td>52</td>
</tr>
<tr>
<td>19</td>
<td>Optimal solutions for case study 2</td>
<td>53</td>
</tr>
<tr>
<td>20</td>
<td>Environment case study 3</td>
<td>54</td>
</tr>
<tr>
<td>21</td>
<td>Optimal solutions case study 3</td>
<td>55</td>
</tr>
<tr>
<td>22</td>
<td>Optimal solutions for the extended MITL formula in case study 3</td>
<td>56</td>
</tr>
<tr>
<td>23</td>
<td>Environment case study 4</td>
<td>57</td>
</tr>
<tr>
<td>24</td>
<td>Solution case study 4</td>
<td>59</td>
</tr>
<tr>
<td>25</td>
<td>Modified Djikstra algorithm issue</td>
<td>61</td>
</tr>
<tr>
<td>26</td>
<td>Issue performing transitions under disturbances</td>
<td>62</td>
</tr>
<tr>
<td>27</td>
<td>Double integrator trajectories</td>
<td>64</td>
</tr>
</tbody>
</table>
1 Introduction

This section starts with giving a motivation for the thesis. Afterwards, in Section 1.2 the related work is presented. The introduction ends with the contribution of this thesis as well as an overview in Section 1.3.

1.1 Motivation

When it comes to stabilizing a system, the different fields in control theory provide excellent tools to do so. However, not every dynamical system should simply be stabilized. Often it is desired that these systems behave in a certain way. For instance, a robot could have the following task "After you enter the kitchen, go to the living room within the next hour". Converting this task into an input for the robot requires new methods and cannot simply be reformulated into a stabilization task.

Such tasks can be expressed using temporal logic. Adding the ’Until’ operator to the Boolean logic operators enables the formulation of these complex high level tasks. Research on obtaining a controller or an input for the dynamical system with specifications given in temporal logic so far has mainly focused on linear temporal logic which neglects the time aspect. The aforementioned task would be then simplified to ”After you enter the kitchen, go to the living room”.

However, it is obvious that time can play a vital role in many tasks. Therefore, it is of interest to consider metric interval temporal logic (MITL) formulas which are also able to take time into account. The time aspect makes the whole problem more challenging. So far most of the research which considers a dynamical system, an environment together with a MITL formula makes strong restrictive assumptions about one or more aspects of the problem.

The goal of this thesis is to provide a novel method which is less restrictive about the assumptions.

1.2 Related Work

Although the research topic is quite young, papers on this problem have already been published and the solution approaches can be divided into two categories. One translates the task in a Büchi Automaton which will lead to a graph search which then results ultimately in a solution, whereas the other one translates the task into a mixed integer linear problem (MILP) where a solution is found via optimization. The solution approach presented in this work combines some aspects from both methods.

In the following, papers which use linear temporal logic formulas are presented first. Although they don’t take time into account like MITL formulas, the presented papers build the foundations to the approach of this thesis. Afterwards, MILP approaches are presented and, finally, papers which consider MITL tasks and solve it via a Büchi Automaton.

The work on linear temporal logic formulas (LTL) has a rich literature such as [23] which translates the given formula into a Büchi Automaton and obtains optimal solutions regarding time. Similar but closer to MITL, in the sense that time bounds are taken into account, is [11] which considers LTL formulas together with a global time bound on the whole task. Further, the authors consider a dynamical system with multi affine state nonlinearity and linear input. The system is abstracted by worst time estimation for a given controller. This is a key point in the solution approach. Without an abstracted system dynamics a graph search would not be possible. The approach in this thesis, heavily builds on this fact. As already mentioned a huge disadvantage is the limitation to LTL.

Unlike the previous work, [14], [28] and [22] try to avoid the construction of a Büchi Automaton by solving the problem via a mixed integer linear problem (MILP). At the example of the vehicle routing problem [14] translates LTL formulas into a MILP.

[28] expands this idea to metric temporal logic (MTL) formulas and dynamical systems by translating the formula into constraints on the dynamical state at certain times. Nonlinear dynamics have to be linearized, in order to reformulate the problem as a MILP. In this framework a changing environment could also be taken into account. MTL also allows for tasks which have to be fulfilled at an exact point.
in time e.g. "Be in the living room in exact 10 seconds". This extends the MITL formulas which is why [28] considers a "richer language".

Although the MTL formula can be translated into a MILP, it is not guaranteed that a solution is obtained. This is one of the key findings in [22], which translates the MTL formula into an optimization problem using a "robustness measure" which does not require integer variables. The resulting cost function, however, is nonlinear and non-differentiable. To solve this, a smooth approximation of the cost function has been used. Furthermore [22] shows how it outperforms other methods using MILP. Another key aspect is that this setup also allows nonlinear systems. A disadvantage, however, is the fact that the convexity of the optimization problem heavily depends on the 'content' of the MTL formula. Despite this fact, [22] could solve a variety of difficult problems. Due to these strong results, this work is compared to the method presented in this thesis, in Section 6.

The following papers use Timed Büchi Automaton (TBA). They are only related to the problem which is considered here. Therefore, they are only briefly discussed. Interesting work on MTL, such as [13], describes a method to do online planning for changing environments as well as changing MTL specifications, using RRTs for the trajectory generation, [17], on the other hand, develops a switching strategy for an autonomous dynamical switching system which satisfies a given MTL formula.

The papers below deal with MITL formulas. For instance, the authors in [10] maximize the probability for satisfying a MITL formula using a Markov decision process while considering stochastic dynamic equations. [24] takes multi-agent systems into account which need to solve a cooperative task. A decentralized robust controller for each agent is proposed. This allows for an abstraction of the dynamical part of the system into a weighted transition system (WTS) which is then combined with the TBA derived from the MITL formula. From this, a hybrid controller is obtained which satisfies the MITL formula.

Closer to the work presented here are the following papers. [29] uses Input Output Timed Automaton to represent a given MITL formula. This is then combined with a transition system which abstracts the system dynamics of e.g. a robot and its environment. The resulting Timed Automaton is then implemented in UPPAAL and a path is generated via the verification tool. [21] also focuses on multi-agent systems together with MITL formulas. Just like the methods above it makes use of a WTS and TBA. The described method avoids unnecessary waiting times of faster agents. While [29] doesn’t take dynamics into account, [21] limits itself to only first order dynamics. Finally [4] considers the same problem as it is done in this thesis. Once again, a controller is introduced. This allows to create a WTS through a worst time estimation. The MITL formula is then translated into a TBA and a solution is obtained via a graph search on the product of TBA and WTS. Unlike the previous two papers a linear dynamical system is considered while creating the WTS. Since this is the closest paper to the method in this thesis, a comparison with that paper is done in Section 6.

1.3 Contribution

In this thesis, a novel approach is presented for solving the aforementioned problem. Under some assumptions, lower constraints in the MITL formula can be taken into account. Furthermore the approach is able to deal with multi agent systems and can also cope with nonlinear dynamics. As already mentioned, the solution approach abstracts the dynamical system. This abstraction is done in a new way. It can be shown that the presented weighted transition system outperforms every weighted transition system created by a controller under a verifiable condition. For the final solution, a cost function can be chosen by the user, thus giving the user the ability to specify a preferred solution.

The thesis is structured in the following way. Section 2 gives a detailed definition of the considered problem. Section 3 describes the solution approach and Section 4 as well as Section 5 extend the problem definition and state how the solution approach has to be adapted. In Section 6 the approach is compared to state of the art methods. The results and the presented solution approach are discussed in Section 7. Section 8 concludes this thesis and Section 9 gives an outlook on future work.
2 Problem Definition

Given an environment with some partitions, a metric interval temporal logic (MITL) formula and a dynamical system, the goal is to steer the dynamical system from one part or state of the environment to another such that the given formula is satisfied. A more precise formulation follows. First the dynamical system is described, then the environment and the MITL formula. At the end of this section, some additional definitions are made in order to give a precise formulation of the problem.

2.1 System Dynamics

The dynamical system with initial condition, $n$ states and $m$ inputs is given by

$$\dot{x}(t) = f(x(t), u(t)) \quad x(0) = x_0 \quad t \geq 0 \quad x \in \mathbb{R}^n.$$  \hspace{1cm} (1)

The standard assumptions for uniqueness and existence of a solution to (1) are made, i.e. we demand the vector field $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ to be uniformly Lipschitz continuous. Usually technical systems have some upper limit on the input which is captured by the following constraint.

**Definition 2.1.1.** Let $g : \mathbb{R}^m \to \mathbb{R}^{k_u}$ be a vectorfunction. The set

$$U := \{u \in \mathbb{R}^m | g(u) \leq 0\}$$

defines the input constraints, where $k_u$ is the number of constraints. Note that the inequality is element-wise.

**Remark 2.1.1.** Throughout this work, the time argument of $x(t)$ and $u(t)$ are sometimes drooped in order to ease the notation.

2.2 Environment

The idea of how to define the environment is taken from [15]. The underlying assumption is that the environment can be divided into convex polytopes. There are mainly two motivations for this partitioning.

1. It is rather intuitive to divide any given area into rectangles or other convex polytopes.

2. The optimization algorithm described in Section 3.2.2 performs much better in terms of convergence speed and robustness if the environment is divided into convex areas.

Therefore, it is assumed that the environment is given in this certain form or that it can be divided into convex polytopes.

Let $E \subseteq \mathbb{R}^n$ denote the environment. An environment state or a part of the environment is denoted by $\mathbb{P}^i \subseteq E$.

**Definition 2.2.1.** Let $\Pi = \{\mathbb{P}^i | i = 1...k_\Pi\}$ denote the set of all environment states, where $k_\Pi$ is the number of elements in $\Pi$, which is equal to the total number of environment states.

The union of all $\mathbb{P}^i \in \Pi$ is $E$, i.e.

$$E = \bigcup_{i=1}^{k_\Pi} \mathbb{P}^i.$$  \hspace{1cm} (2)

As mentioned above, each state $\mathbb{P}^i$ is a convex polytope which means that

**Definition 2.2.2.** Each environment state $\mathbb{P}^i$ can be described by

$$\mathbb{P}^i = \{x \in \mathbb{R}^n | \alpha_j^T x + b_j^i \leq 0, \ j = 1,...,k_{p^i}\}$$

where $k_{p^i}$ is the number of constraints which is needed to describe the environment state $\mathbb{P}^i$ and $\alpha_j^i \in \mathbb{R}^n, b_j^i \in \mathbb{R}$ for $i = 1,...,k_{p^i}$ define the constraint itself.
To keep the notation short the vector function \( \mathbf{p}_i(\mathbf{x}) \) is defined as

\[
\mathbf{P}_i = \{ \mathbf{x} \in \mathbb{R}^n | \mathbf{p}_i(\mathbf{x}) \leq 0 \}
\]

where \( \mathbf{p}_i(\mathbf{x}) = a_{ij}^T \mathbf{x} + b_{ij} \).

To solve the problem later on, it will be important to know which environment states \( \mathbf{P}_i \in \Pi \) can be reached from the current environment state \( \mathbf{P}_j \). These environment states will be called neighbours and are formally defined in the following.

**Definition 2.2.3.** The environment state \( \mathbf{P}_i \) is a neighbour of \( \mathbf{P}_j \) if

\[
\exists \mathbf{u}(t) \in \mathcal{U} \text{ for } t \in [0, T] \quad (2)
\]

\[
s.t. \ \mathbf{x}(0) \in \mathbf{P}_i \quad (3)
\]

\[
\mathbf{x}(t) \in \mathbf{P}_i \text{ for } t \in [0, T) \quad (4)
\]

\[
\mathbf{x}(T) \in \mathbf{P}_j \quad (5)
\]

and for a given input \( \mathbf{u}(t) \) according to \((2)-(5)\)

\[
\exists \varepsilon > 0 \text{ s.t. } \exists \mathbf{e} \text{ with } \varepsilon \geq ||\mathbf{e}|| \text{ s.t. } \mathbf{x}(t) + \mathbf{e} \in \mathbf{P}_k \text{ for } t \in [0, T] \quad (6)
\]

where \( \mathbf{P}_k \neq \mathbf{P}_i \), \( \mathbf{P}_k \neq \mathbf{P}_j \) and \( \mathbf{P}_i \neq \mathbf{P}_j \).

**Remark 2.2.1.** The expressions \((2)-(5)\) simply state that there exists an input to go from \( \mathbf{P}_i \) to \( \mathbf{P}_j \) without entering any other environment states. \((6)\) states that some robustness for \( \mathbf{x}(t) \) is required. It follows that cutting over corners is not allowed. For example a direct transition from A to C in Figure 2 is not a feasible transition. This is motivated by the fact that later on disturbances are taken into account, thus making transitions over a corner impossible since it can’t be guaranteed anymore that another environment state won’t be entered.

### 2.3 Metric Interval Temporal Logic

Since LTL has been mentioned several times in Section 1.2 a quick overview of LTL is given and how it is related to MITL without going into detail. LTL is a mathematical concept which allows us to express high level tasks, such as “After entering room A go to room B” or “Eventually go to room C”, by extending the Boolean operators with an Until operator. MITL builds on and extends LTL by taking time intervals into account. Therefore, it can formulate tasks with a time interval e.g. “After entering room A the robot must be in room B within 10 seconds”. In the last case, the robot has to be in room B within an arbitrary time between 0 and 10 seconds.

As already mentioned in the given examples, some atomic propositions like "being in room A" or "being in room B" which are either true or false are required, in order to use an MITL formula.

**Definition 2.3.1.** An atomic proposition \( \text{ap} \) is a statement over the system variables that is either true (\( \top \)) or false (\( \bot \)). AP describes a set of atomic propositions.

Each environment state \( \mathbf{P}_i \) has one or more atomic proposition \( \text{ap} \) according to Definition 2.3.1 which are true as long as \( \mathbf{x} \in \mathbf{P}_i \). In particular, it is forbidden that some \( \text{ap} \) is true in one part of \( \mathbf{P}_i \) and false in another part of \( \mathbf{P}_i \). This implies that

\[
\mathbf{P}_i \cap \mathbf{P}_j = \emptyset \quad \forall i \neq j \text{ and } i, j \in \{1, ..., k_{\Pi}\}.
\]

Otherwise the requirement above would be violated, since on these points the \( \text{ap} \) of \( \mathbf{P}_i \) and \( \mathbf{P}_j \) would become true, while on other points in \( \mathbf{P}_i \) the valid \( \text{ap} \) from \( \mathbf{P}_j \) could be false. In order to get the valid atomic propositions of an environment state \( \mathbf{P}_i \) the function \( L \) is introduced.
Definition 2.3.2. The function $L : \Pi \rightarrow 2^{AP}$ labels some atomic proposition to each state $P_i$, where $2^{AP}$ denotes the power set.

Connecting the atomic propositions forms the MITL formula, e.g. "After entering ... be in ..." or "entering ... implies ...". This is done via some operators which form, together with the atomic propositions, the syntax of MITL.

Definition 2.3.3. The syntax of Metric Interval Temporal Logic over a set of atomic propositions $AP$ is defined by the grammar

$$\phi ::= \top \mid \text{ap} \mid \neg \phi \mid \phi \lor \psi \mid \phi \mathcal{U}[a,b]\psi$$

where $\text{ap} \in AP$ and $\phi, \psi$ are formulas over $AP$. The operators are Negation ($\neg$), Conjunction ($\lor$) and Until ($\mathcal{U}$) respectively. The extended operators Eventually ($\Diamond$) and Always ($\Box$) are defined as:

$$\Diamond[a,b]\phi ::= \top \mathcal{U}[a,b]\phi \quad (7)$$

$$\Box[a,b]\phi ::= \neg \Diamond[a,b] \neg \phi \quad (8)$$

Further in this work, the following inequality for the parameters $a$ and $b$ has to hold

$$0 \leq a < b < \infty \quad (9)$$

Remark 2.3.1. Note that (9) doesn’t allow for $b$ to be infinity. This restricts the usual definition of MITL but also avoids cyclic solutions which go on forever. These kind of solutions are not considered in this work and therefore $b$ has to be finite.

Definition 2.3.4. A timed run is defined as

$$r_t = ([P_{\Omega(0)}, 0)(P_{\Omega(1)}, T_1) \ldots (P_{\Omega(N)}, T_N),$$

where $\Omega : \{0, \ldots, N\} \rightarrow \{1, \ldots, k_\Pi\}$ is an indicator function which maps the current step to the environment state.

The conditions on weather or not a timed run satisfies a MITL formula is given in the following.

Definition 2.3.5. The semantics of the satisfaction relation is defined as:

- $(r_t, i) \models \text{ap} \iff ap \in L(P_{\Omega(i)})$
- $(r_t, i) \models \neg \phi \iff (r_t, i) \not\models \phi$
- $(r_t, i) \models \phi \land \psi \iff (r_t, i) \models \phi$ and $(r_t, i) \models \psi$
- $(r_t, i) \models \phi \mathcal{U}[a,b]\psi \iff \exists T_j \in [a,b], \text{ s.t. } (r_t, j) \models \psi$ and $\forall i \leq j, (r_t, i) \models \phi$

where $(r_t, i)$ denotes the $i$th step of the sequence.

2.4 Time Constraints and Run

Since MITL formulas also consider time constraints it is important to know when the atomic propositions become true. The time $T_k$, as already used in Definition 2.3.4, denotes when $x$ is changing the sets from $P_{\Omega(k-1)}$ to $P_{\Omega(k)}$, and therefore the atomic proposition which is currently true changes from $L(P_{\Omega(k-1)})$ to $L(P_{\Omega(k)})$. $T_k$ is defined in such a fashion that right before $T_k$, $x \in P_{\Omega(k-1)}$ and at time $T_k$ and right after $x \in P_{\Omega(k)}$.  

5
Definition 2.4.1. The time $T_k \in \mathbb{R}^+$ s.t. $x(t) \in \mathbb{P}^{\Omega(k-1)}$ for $T_{k-1} \leq t < T_k$ and $x(T_k) \in \mathbb{P}^{\Omega(k)}$ denotes when $x(t)$ is changing the set from $\mathbb{P}^{\Omega(k-1)}$ to $\mathbb{P}^{\Omega(k)}$, where $\mathbb{P}^{\Omega(k)}$ is a neighbour to $\mathbb{P}^{\Omega(k-1)}$ according to Definition 2.2.3.

Again, to ease notation, all $T_k$ are stored in one vector. Let the time vector
\[ T = [T_1 \ldots T_N]^T \] (10)
be the vector with all the times $T_k$ according to the Definition 2.4.1. The vector $T$ has $N$ elements which means that there will be $N$ transitions. The following example motivates another short notation which is stated after the example.

Example 2.1. Consider a task where the agent is required to go from A to B and then to C. The whole task should be done within 10 seconds while the transition from B to C should be done within 5 seconds. Further assume that the agent starts in A and enters B at time $T_1 = 5$. After that, the agent goes from B to C within 4 seconds which results in $T_2 = 9$. In total, there are $N = 2$ steps. Since $T_2$ refers to the total time, this gives the time bound
\[ T_2 \leq 10. \]
However, this is not the only bound on $T_2$. The transition from B to C is the time difference between $T_2 - T_1$, thus leading to the second time bound
\[ T_2 \leq 5 + T_1. \]
For a shorter notation for the inequality constraints of $T_k$ the vector
\[ T_k = [T_k \ldots T_k]^T \]
of length $N$ is introduced.

Despite knowing when $x$ is changing the environment, it is also important to know the time limits for these transitions. This motivates the following definition.

Definition 2.4.2. The function $\tau^{up} : \mathbb{N} \times \mathbb{R}_+^N \rightarrow \mathbb{R}_+^N \cup \{\infty\}$ determines upper time limits for each time $T_k$. Analogously, $\tau^{low}$ determines lower limits on each $T_k$. This means that
\[ \tau^{up}(k,T) \leq T_k \leq \tau^{low}(k,T) \]

The exact constraints are obtained by the graph search algorithm later on, which is described in detail in Section 3.1.6.

Finally we define a run which is $N+1$ steps long since the last point in time the environment changes is $T_N$. Up to that time, the MITL formula has to be satisfied.

Unlike the usual definition, a run in this work is defined as

Definition 2.4.3. A run is a sequence of environment states and time constraints
\[ \text{run} = (\mathbb{P}^{\Omega(0)}, 0, \infty, \mathbb{P}^{\Omega(1)}, \mathbb{P}^{\Omega(1)}, \tau^{up}(1,T), \tau^{low}(1,T)) \ldots (\mathbb{P}^{\Omega(N)}, \tau^{low}(N,T), \tau^{up}(N,T)) \]

2.5 Problem Statement

The considered Problem is to find an input sequence $u(t)$ for the system (1) such that the MITL formula $\phi$ is satisfied, given the environment $E$ partitioned according to Section 2.2. The problem is split into two smaller problems:

Problem 1. Given the MITL formula $\phi$ together with the environment $E$ and the given dynamics from (1), find a run according to Definition 2.4.3 such that a timed run $r_t$ which satisfies all the time constraints of the run also satisfies the MITL formula $\models \phi \forall i$. 

6
Once Problem 1 is solved, the remaining issue is to check if there exists an input such that the sequence of states can be fulfilled subject to the given time constraints.

**Problem 2.** Find a control input \( u(t) \in U \) for a given run such that the trajectory \( x(t) \) generated by \( u(t) \) satisfies

\[
x(t) \in \mathbb{P}^{\Omega(k-1)} \quad \text{for } t \in [T_{k-1}, T_k]
\]

and \( T_k \) satisfies the \( k \)-th time constraints of the run.

# 3 Solution Approach

The solution approach is split according to the problems.

## 3.1 Solution of Problem 1

The solution is straightforward. First, a weighted Transition System is created according to the environment. Then, using the MITL formula, a TBA is created. By taking the product of the WTS and the TBA, a graph is built. Via a graph search algorithm a run according to Definition 2.4.3 is obtained.

### 3.1.1 Weighted Transition System

Given an environment \( E \) as well as some system dynamics \( 1 \), it is possible to abstract the continuous dynamics into a discrete transition system. It is important that the abstraction is conservative in order to ensure that transitions of the abstract model are also possible in the original model. Since the abstraction is based on defining multiple points and then calculating the time needed to go from one point to another, it is required to know to which environment state \( P_i \) the point belongs. This motivates the next definition.

**Definition 3.1.1.** Let the function \( P : E \to \Pi \) be defined as

\[
P(x) = P^i,
\]

where \( x \in \mathbb{P}^i \).

A weighted transition system has the following structure.

**Definition 3.1.2.** A weighted transition system is a tuple \( T = (V, V_0, \Sigma, \to, AP, L, d) \) where

- \( V = \{v_i : i = 1, 2, \ldots, k_v\} \) is a set of points with \( v_i \in \mathbb{R}^n \),
- \( V_0 = v_1 = x_0 \) is the initial state given by \( 1 \),
- \( \Sigma = \{\sigma_{ij} : i = 0, \ldots, k_v, j = 0, \ldots, k_v\} \) is a set of inputs,
- \( \to \subset V \times V \) is a transition map; the expression \( (v_i, v_j) \in \to \) expresses that a transition from \( v_i \) to \( v_j \) is possible,
- \( AP \) is a set of observations (atomic propositions) according to Definition 2.3.1
- \( L : \Pi \to 2^{AP} \) is an observation map according to Definition 2.3.2
- \( d : V \times V \to \mathbb{R}^+ \) The map \( d(v_i, v_j) \) returns a positive weight for the transition from \( v_i \) to \( v_j \).
Note that $T$ has a finite number of discreet states $v_i$. Thus, the abstraction eliminates the continuous part of the task which allows to apply a graph search later on. Now that the WTS has been stated, the following sections explain how the individual parts, which are not already defined in the definition of the WTS, can be obtained.

**Remark 3.1.1.** Note that the definition of the weighted transition system differs from the usual definition given in other papers. Here a point $v \in V$ refers to a state of the WTS.

### 3.1.2 Choosing the Points of the WTS

The first point of the set $V$ is always predefined by the initial point $x_0$. Choosing the number of points in $V$ as well as the location of the points is one degree of freedom of the method and determined by the user. The only restriction is that there should be at least one point in each environment state $P_i$. Thus, the number of points is always greater or equal to the number of environment states

$$k_H \leq k_v.$$  

Formally this means that

$$\forall P_i \in \Pi \exists v_j \in V \text{ s.t. } P(v_j) = P_i.$$  

(11)

A violation of this would mean that the WTS is not able to reach every environment state $P_i$. Obviously this can cause problems, especially if the MITL formula requires the agent to go to that specific environment state.

The complexity of the WTS is mainly determined by the number of points. An intuitive idea is to simply set one point in each of the environment states which will result in the WTS with the lowest complexity. Although this is true and is done in the case study Section 6.1, it is only valid for MITL formulas without lower constraints, i.e. the parameter $a$ in Definition 2.3.3 is always 0. The reason for that is explained and illustrated in Figure 1. The problem is that choosing the points of the WTS as in Figure 1a will create a ”false positive”. Assuming that $T_a < a \leq T_b$, for the transition in Figure 1a it is possible to fulfill the MITL formula

$$\phi = A U_{[a,b]} B$$

although the trajectory in Figure 1b will state that it is not. The semantic description of the formula $\phi$ is: "Stay in A until the time $a$. Afterwards go from A to B before time $b".

Due to this fact, it becomes clear that for each neighbour of an area, the area itself needs at least one point on its edge facing the neighbouring area. This will avoid false positive results if the MITL formula has lower bounds in the intervals which are unequal to zero. This results in

$$\sum_{i=1}^{k_H} \text{NumberOfNeighbours}(P^i) \leq k_v$$  

(12)

total points, where $\text{NumberOfNeighbours}(P^i)$ returns the number of neighbours of an environment state $P^i$.

Still, how to choose the points as well as its number remains one degree of freedom. For the minimal amount of points according to equation (12), the Algorithm 1 describes an automated way of choosing points. The underlying idea is to start out from the initial point $x_0$ and calculate the time optimal transition to the neighbouring environment states of $P(x_0)$. The final point $v_i$ of the time optimal transition then becomes a point of the WTS. This is then repeated for the newly obtained points, unless the transition from $P(v_i)$ to the neighbouring environment state of $P(v_i)$ has already been calculated.
Figure 1: The figure illustrates the problem when choosing the point $v_i$ of the WTS, for an area in its center. A WTS generated according to the left picture is not suitable for an MITL formula with lower constraints. Suppose a MITL formula, yields the following constraints, namely, that the transition from A to B should be done within the interval of $[a, b]$ where $0 < a$. Further assume that the left picture gives a transition time $T_b \geq a$ and the right picture gives a transition time $T_a < a$. According to the WTS generated from the points given in the left picture, this transition is valid. This is a "false positive" since the picture on the right indicates that the MITL formula is violated since the area B is entered before it is allowed to be entered.

**Algorithm 1:** Choosing Points Automatically

```
TransitionNotDoneToArea(1 : k_Π, 1 : k_Π) ← true
V ← ∅
add $x_0$ to V
List W ← V
while W is not empty do
    $v$ ← first element in W
    $P_j$ ← $P(v)$
    foreach $P_i$ ∈ neighbour($P_j$) do
        if TransitionNotDoneToArea(j, i) then
            $v'$ ← TimeOptimization($v, P_i$)
            add $v'$ to V
            add $v'$ to W at the last position
        end
    end
    remove $v$ from W
end
```
Remark 3.1.2. The optimization problem solved in TimeOptimization\((v_i, P^k)\) is the same as in [14] where

\[
\begin{align*}
\bar{p} &= v_i, \\
P^i &= P(v_i), \\
P^j &= P^k.
\end{align*}
\]

The function returns the final point \(x(T)\).

### 3.1.3 Inputs and Weights of the WTS

Transitions from the environment state \(P^i\) to \(P^j\) are only allowed if \(P^i\) is a neighbour to \(P^j\) according to Definition 2.2.3. Further, it can also happen that transitions in between two neighbouring environment states are forbidden by the problem. This could for instance represent a wall as it is the case for Example 3.1. Since transitions within one environment state itself won’t change any of the atomic propositions \(ap\) which are currently true, transitions within a environment state are forbidden as well. This does not mean that the WTS cannot stay in one environment state, it just means that changing the states of the WTS within the environment state is not allowed. The WTS is allowed to remain in the same state for some time. All in all, this means that, given a set of points \(V\), a transition from a point \(v_i \in V\) to \(v_j \in V\) has to fulfill the following conditions:

- \(P(v_i)\) is a neighbour of \(P(v_j)\),
- a transition between \(P(v_i)\) and \(P(v_j)\) is allowed by the problem,
- the transition does not happen within one environment state \(P(v_i) \neq P(v_j)\).

Additionally, in order to be part of the transition map \(\rightarrow\), there must exist a valid input to the transition. Therefore, the inputs in the set \(\Sigma\) determine the transition map \(\rightarrow\) and vice versa. This means that

\[
\sigma_{ij} \in \Sigma \iff (v_i, v_j) \in \rightarrow.
\]

The inputs \(\sigma_{ij}\) are obtained together with the weights, which is described in the following.

For calculating the weights the idea is to make use of the principal of optimality. Loosely speaking, the goal is to exploit the fact that optimal trajectories are optimal. This means that changing an optimal trajectory would result in a ‘less’ or ‘equally’ optimal trajectory but it will never become ‘more’ optimal. If now, for instance, a task is considered where one has to go from A to B and then to C, optimizing the whole task will always result in a better or ‘equally optimal’ trajectory than optimizing the way from A to B and after that optimizing the way from B to C. However, the latter task is simpler and most likely faster to solve. This will ensure that the weighted transition system is conservative, since optimizing the full way will always be ‘better’ or ‘equally optimal as optimizing over parts of the way. This is illustrated in Figure 2 where the parts of the red curve form together the trajectory obtained from the WTS, which corresponds to optimizing from A to B and from B to C separately. The green curve represents the optimal path for the whole trajectory.

A more formal way to describe this is given in the following. Consider the three optimization problems

\[
\begin{align*}
J &= \min_u T_a + T_b \\
s.t. \quad &\dot{x} = f(x, u) \\
x(0) &= x_0 \\
x(T_a) \in P^1 \\
x(T_a + T_b) \in P^2 \\
u \in U
\end{align*}
\]

\[
\begin{align*}
J_1 &= \min_u T'_a \\
s.t. \quad &\dot{x} = f(x, u) \\
x(0) &= x_0 \\
x(T'_a) \in P^1 \\
u \in U
\end{align*}
\]

\[
\begin{align*}
J_2 &= \min_u T'_b \\
s.t. \quad &\dot{x} = f(x, u) \\
x(0) &= x(T'_a) \\
x(T'_b) \in P^2 \\
u \in U
\end{align*}
\]
Figure 2: Example of designing the weight for the WTS. The green curve represents the trajectory for the optimization over to whole path. That means it is the solution for the time optimal problem of finding a path from A over B to C. The red curve is obtained by optimizing the required time for the way from A to B and then, starting form the endpoint, from B to C. The red curve is less optimal than the green curve symbolized by the length of the curve.

where the last two problems are solved in sequence since the optimization for $J_2$ requires $x(T'_a)$ as an initial condition, which is obtained by solving $J_1$. According to the principal of optimality this yields for the optimal values,

$$J \leq J_1 + J_2.$$ 

Otherwise the value for $J$ would not be optimal.

The weight and input between two given points $v_i \in V$ and $v_j \in V$ are obtained via solving the following problem

\begin{align}
J &= \min_u \int_0^T dt \\
\dot{x} &= f(x,u) \\
x(0) &= v_i \\
u &\in \mathbb{U} \\
x(t) &\in P(v_i) \text{ for } t \in [0,T_a) \\
x(t) &\in P(v_j) \text{ for } t \in [T_a,T] \\
x(T) &= v_j
\end{align}

where $0 < T_a \leq T$. The solution of (13) gives the weight

$$d(v_i, v_j) = J,$$

as well as the transition input

$$\sigma_{ij}(t) = u^*(t)$$
where \( u^*(t) \) is the optimal input. In this fashion the weights represent the optimal time required going from a point in \( P_i \) to a point in \( P_j \). The construction of a WTS is now illustrated as an example.

**Example 3.1.** Given the environment \( \mathbb{E} \) displayed in Figure 3, as well as the dynamical system

\[
\dot{x}(t) = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \quad x(0) = \begin{bmatrix} 6 \\ 4.5 \end{bmatrix}.
\]

The points \( v_1-5 \) are already chosen and have the following values

\[
v_2 = \begin{bmatrix} 3 \\ 7.5 \end{bmatrix}, \quad v_3 = \begin{bmatrix} 9 \\ 7.5 \end{bmatrix}, \quad v_4 = \begin{bmatrix} 3 \\ 1.5 \end{bmatrix}, \quad v_5 = \begin{bmatrix} 9 \\ 1.5 \end{bmatrix}.
\]

The WTS is illustrated in Figure 4. Due to the dynamics of the system not depending on the state and the symmetry of the problem, every transition has the same weight.

**Remark 3.1.3.** In general this is not the case. Usually one even has

\[ d(v_1, v_2) \neq d(v_2, v_1). \]
Figure 4: The figure shows the simplified WTS. One can see the different states together with the transition times to go from one state to another. Due to the symmetric structure of the problem and the fact that the dynamic of the system does not depend on the states, all transitions have the value of 1.5.

Formally, the WTS results in

\[ V = \{v_1, v_2, v_3, v_4, v_5\} \]
\[ V_0 = x_0 \]
\[ \Sigma \text{ consist of the optimal input trajectories} \]
\[ \rightarrow = \{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_1, v_5), (v_2, v_1), (v_3, v_1), (v_4, v_1), (v_5, v_1)\} \]
\[ AP = \{c, r_1, r_2, r_3, r_4\} \] where \(c\) is short for being in the corridor and \(r_i\) for being in room \(i\)

\[ L(P(v_1)) = c \]
\[ L(P(v_2)) = r_1 \]
\[ L(P(v_3)) = r_2 \]
\[ L(P(v_4)) = r_3 \]
\[ L(P(v_5)) = r_4 \]
\[ d(v_i, v_j) = 1.5 \forall (v_i, v_j) \in \rightarrow \]

The points of the example have been chosen in a rather simple fashion. Before moving on to the construction of the TBA, the next section considers whether the points of the WTS are chosen in a good or bad way.

3.1.4 Theoretical Property of the WTS

Some of the previous works like [11] and [4] first create a controller for every environment state \(\mathbb{P}_i\) which will then transfer the dynamical system to one of the neighbour environment states \(\mathbb{P}_j\). The weight for the transition system is then estimated by using the point \(x \in \mathbb{P}_i\) which needs the most time to reach \(\mathbb{P}_j\). This gives the weight for going from \(\mathbb{P}_i\) to \(\mathbb{P}_j\). In this case, the resulting WTS is called "control to facet WTS", whereas the WTS presented above is called "point WTS". Consider the optimization problem
\[ J = \min_u \int_0^T dt \]  
\[ \text{s.t. } \dot{x} = f(x, u) \]  
\[ x(0) = \bar{p} \]  
\[ u \in U \]  
\[ x(t) \in \mathbb{P}_i \text{ for } t \in [0, T) \]  
\[ x(T) \in \mathbb{P}_j \]  

(14a)

(14b)

(14c)

(14d)

(14e)

(14f)

where \( \bar{p} \in \mathbb{P}_i \).

**Theorem 1.** The point WTS will underestimate every control to facet WTS if for every \( P(v_i) = \mathbb{P}_i \) and its neighbours \( P(v_j) = \mathbb{P}_j \) there exists a point \( \bar{p} \in \mathbb{P}_i \) such that the weight \( d(v_i, v_j) \) of the point WTS is smaller than the optimal value of (14), i.e.

\[ d(v_i, v_j) \leq J. \]

**Proof.** Assume that controller for the control to facet WTS is calculated in an optimal way. This means that for every point in \( \mathbb{P}_i \) problem (14) is solved. Since it is time optimal there exists no faster controller. The weight \( d^*(\mathbb{P}_i, \mathbb{P}_j) \) for the control to facet WTS is now calculated by solving

\[ J_{\text{max}} = \max_{\bar{p} \in \mathbb{P}_i} \min_u \int_0^T dt \]  
\[ \text{s.t. } \dot{x} = f(x, u) \]  
\[ x(0) = \bar{p} \]  
\[ u \in U \]  
\[ x(t) \in \mathbb{P}_i \text{ for } t \in [0, T) \]  
\[ x(T) \in \mathbb{P}_j. \]  

(15a)

(15b)

(15c)

(15d)

(15e)

(15f)

Due to optimality, every other point \( \bar{p} \neq \bar{p}^* \) different from the optimal point will return a shorter or equal optimal time for (14)

\[ J \leq J_{\text{max}} = d^*(\mathbb{P}_i, \mathbb{P}_j). \]

The condition of the theorem yields

\[ d(v_i, v_j) \leq J \leq J_{\text{max}} = d^*(\mathbb{P}_i, \mathbb{P}_j). \]

Thus, taking any transition in the point WTS estimates a time less or equal to the transition in the control to facet WTS. Therefore, the control to facet WTS will be underestimated by the point WTS.

The following example checks this condition for the previous Example 3.1.

**Example 3.2.** Solving (14) for the point \( \bar{p} = [1, 0]^T \in \mathbb{P}_4 \) which is the corridor, yields an optimal value of \( J^* = 1.495 \) time units. As already stated, since the problem is symmetric and has a state invariant dynamic the same time can be assumed for every other room-to-corridor transition. Recall that the weight of the WTS is constantly 1.5. Hence, Theorem 1 cannot be applied. A better set of points \( V \) for the WTS can be constructed by moving every point \( v_i \) closer to the initial point \( x_0 \). Thus, the points are now given by

\[ v_2 = \begin{bmatrix} 5 \\ 6 \end{bmatrix}, \quad v_3 = \begin{bmatrix} 7 \\ 6 \end{bmatrix}, \quad v_4 = \begin{bmatrix} 5 \\ 3 \end{bmatrix}, \quad v_5 = \begin{bmatrix} 7 \\ 3 \end{bmatrix}. \]
The weights of the new WTS have a value of 0.76. Since the new weights are smaller than $J^* = 1.495$, the WTS underestimates every control to facet WTS for the transition from a room to a corridor. It remains to check if the same is valid for the transition from the corridor to a room. Solving (14) for $\bar{\rho} = [6, 3]^T \in \mathbb{P}^1$ to $\mathbb{P}^2$ (Room 2) yields $J^* = 3 > 0.76$. The condition for Theorem 1 holds, therefore the presented WTS "outperforms" every control to facet WTS.

**Remark 3.1.4.** At this point it should be noted that in the given example the point WTS still has the same complexity as a control to facet WTS.

### 3.1.5 Timed Büchi Automaton

As stated in the beginning of the Section 3.1 the ultimate goal is to perform a graph search which generates a run. Following this idea, the MITL formula has to be translated into a Timed Büchi Automaton (TBA) which can be represented as a graph. Before the TBA is stated, clock constraints have to be defined.

**Definition 3.1.3.** A clock constraint $\Phi_x$ is a conjunction of inequalities of the form $x \in a$ where $\in \{<, >, \leq, \geq\}$, $x$ is a clock and $a$ is some constant. Let $\Phi_X$ denote the set of clock constraints.

The TBA is given in the following definition.

**Definition 3.1.4.** A timed Büchi Automaton (TBA) is a tuple $A = (S, S_0, X, I, E, F, AP, \mathcal{L}, R)$ where

- $S = \{s_i : i = 0, 1, ..., k_S\}$ is a finite set of locations,
- $S_0 \in S$ is the set of initial locations,
- $X$ is a finite set of clocks,
- $I : S \rightarrow \Phi_X$ is a map labelling each location $s_i \in S$ with some clock constraints $\Phi_X$,
- $E \subseteq S \times S$ is a set of transitions and
- $F \subseteq S$ is a set of accepting locations,
- $AP$ is a finite set of atomic propositions,
- $\mathcal{L}$ is a labelling function, labelling every location with a subset of atomic propositions.
- $R : X \times S \rightarrow \{\top, \bot\}$ if true the clock $x \in X$ has to be reset to zero in location $s_i \in S$

The translation from a MITL formula to a TBA is not intuitive. The main result of [2], [7], [18] and [20] state that it is possible to construct a timed automaton from a MITL formula which is given in the following corollary.

**Corollary 1.** It is possible to construct a timed automaton from a MITL formula.
1. Define the initial location \( S_0 \) where all the \( L(P(x_0)) \) are active.

2. Consider all possible initial actions which could yield a satisfying timed run and create one location for each such action. For instance, if the formula is \( \phi = a \lor b \), the initial actions \( a \) and \( b \) result in a satisfying timed run. Create edges and define clock constraints accordingly.

3. Create a non-accepting location which handles all other possible actions. In the example above this would be \( \neg(a \lor b) \).

4. Iterate over step 2 and 3 considering the locations created in step 2 rather than the initial location. When performing step 3, there is no need to create new non-accepting locations, it is enough to create new edges to the already existing non-accepting location. As for step 2, it is not always needed to create a new location. In some cases a better solution is achieved by creating a transition back to another already existing location.

5. Mark the locations which satisfy the formula as the accepting locations.

6. Define one or two clocks \( x \in \mathcal{X} \) for each bounded temporal operator in the MITL formula, i.e for each clock constraint. If the interval which is bounding the operator includes 0, one clock is enough.

7. Define the labelling function in accordance with the created locations.

Detailed examples on how to apply the manual can be found in [3]. For illustration purposes the following example shows the TBA resulting from a MITL formula \( \phi \).

![TBA Diagram](image)

Figure 5: Illustrates the most important parts of a TBA generated by the MITL formula \( \phi = \Diamond_{[0,0.06]}r_2 \land (r_2 \Rightarrow \Diamond_{[0,0.3]}r_4) \). The TBA has 4 locations where \( s_3 \) is the goal location at which the MITL formula is satisfied and \( s_4 \) is the sink location at which the MITL formula has failed once it is reached. The conditions under which a change in the location happens, is displayed on the edges of the graph. They can either be atomic propositions becoming true like \( r_2 \) or some violation of the clock. Also displayed are the clock constraints for each location by the function \( I(\cdot) \).

**Example 3.3.** Consider the formula

\[
\phi = \Diamond_{[0,0.06]}r_2 \land (r_2 \Rightarrow \Diamond_{[0,0.3]}r_4),
\]
where \( r_2 \) and \( r_4 \) stand for the \( ap \) of being in room 2 and being in room 4, respectively, as illustrated in Figure 3. Further, it is assumed that the initial point is in the corridor. Semantically this expresses the task: "Go to room 2 within 0.06 time units and if you reach room 2, go to room 4 within 0.3 time units."

An overview with the most important properties of the resulting TBA is given in Figure 5. Formally, the TBA results in

\[
\begin{align*}
S &= \{ s_1, s_2, s_3, s_4 \} \\
S_0 &= s_1 \\
\mathcal{X} &= \{ x_1 \} \\
I(s_1) : 0 &\leq x_1 \leq 0.06 \\
I(s_2) : 0 &\leq x_1 \leq 0.3 \\
I(s_3) : 0 &\leq x_1 \leq \infty \\
I(s_4) : 0 &\leq x_1 \leq \infty \\
E &= \{ (s_1, s_2), (s_2, s_3), (s_1, s_4), (s_2, s_4) \} \\
F &= s_3 \\
AP &= \{ c, r_1, r_2, r_3, r_4 \} \text{ where } c \text{ is the corridor} \\
\mathcal{L}(s_1) &= \{ c, r_1, r_3, r_4 \} \\
\mathcal{L}(s_2) &= \{ c, r_1, r_2, r_3 \} \\
\mathcal{L}(s_3) &= \{ c, r_1, r_2, r_3, r_4 \} \\
\mathcal{L}(s_4) &= \{ c, r_1, r_2, r_3, r_4 \} \\
R(s_1) &= \text{false} \\
R(s_2) &= \text{true} \\
R(s_3) &= \text{false} \\
R(s_4) &= \text{false}. 
\end{align*}
\]

3.1.6 Building the Graph and Graph Search

Now that the WTS \( \mathcal{T} \) and the TBA \( \mathcal{A} \) have been built as described in the previous sections, the next step is to obtain a run according to Definition 2.4.3. First, the product between the WTS and the TBA \( \mathcal{A}^P = \mathcal{T} \otimes \mathcal{A} \) is built. The product is in the spirit of [4] and defined as follows.

\textbf{Definition 3.1.5.} The product between a WTS \( \mathcal{T} = (V, V_0, \Sigma, \rightarrow, AP, L, d) \) and a TBA \( \mathcal{A} = (S, S_0, \mathcal{X}, I, E, F, AP, L, R) \) is called Büchi WTS. It is defined as \( \mathcal{A}^P = \mathcal{T} \otimes \mathcal{A} = (Q, Q_0, \rightarrow, F^P, AP, L^P, I^P, \mathcal{X}^P, d^P, R^P) \) where

- \( q = (s, v) \in Q = S \times V \)
- \( q^\text{init} = S_0 \times V_0 \)
- The transitions \( \rightarrow \) are given by
  - \( (s, v) \leadsto (s', v') \) with \( s \neq s' \) and \( v = v' \). This can only occur if a time constraint has been violated. The weight for this transition is set to zero \( d^P(q, q') = 0 \).
  - \( (s, v) \leadsto (s', v') \) with \( s = s' \) and \( v \neq v' \) iff

17
Remark 3.1.5. Note that the product only allows transitions which are possible by the TBA as well.

In Example 3.3, the product of the WTS and TBA is illustrated in Figure 6.

Example 3.4. Using the WTS that was constructed in Example 3.1 and the TBA that was constructed

as the WTS. In that way it is guaranteed that a graph search over the product will result in a possible
timed run \( r_t \) according to Definition 2.3.4

The following example illustrates how the Büchi WTS is build.

Example 3.4. Using the WTS that was constructed in Example 3.1 and the TBA that was constructed

in Example 3.3, the product of the WTS and TBA is illustrated in Figure 6.

This yields the following for the state of the graph product

\[
Q = \{(s_1, v_1), (s_1, v_2), (s_1, v_4), (s_1, v_5)
(s_2, v_1), (s_2, v_2), (s_2, v_3), (s_2, v_4)
(s_3, v_1), (s_3, v_2), (s_3, v_3), (s_3, v_4), (s_3, v_5)
(s_4, v_1), (s_4, v_2), (s_4, v_3), (s_4, v_4), (s_4, v_5)\}
\]

\( q^{\text{init}} = (s_1, v_1) \).

The transitions from the first condition of Definition 3.1.5 are the transitions to the sink location \( s_4 \), namely

\[
\leadsto \{((s_1, v_1), (s_4, v_1))
((s_1, v_2), (s_4, v_2))
((s_1, v_3), (s_4, v_3))
((s_1, v_5), (s_4, v_5))
((s_2, v_1), (s_4, v_1))
((s_2, v_2), (s_4, v_2))
((s_2, v_3), (s_4, v_3))
((s_2, v_4), (s_4, v_4))\}.
\]
For all these transitions the weight $d^P$ is set to 0. For the rest of the transitions the weight is given by the WTS. The transitions for $(s, v) \rightsquigarrow (s', v')$ with $s = s'$ and $v \neq v'$, which means that the location of the TBA doesn’t change but the WTS changes its state, are

$$\Gamma_1 = \{(s_1, v_i), (s_1, v_j)\} \text{ with } (v_i, v_j) \in \to$$

$$\Gamma_2 = \{(s_2, v_i), (s_2, v_j)\} \text{ with } (v_i, v_j) \in \to$$

$$\Gamma_3 = \{(s_3, v_i), (s_3, v_j)\} \text{ with } (v_i, v_j) \in \to$$

$$\Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \subset \to.$$

The transitions for $(s, v) \rightsquigarrow (s', v')$ with $s \neq s'$ and $v \neq v'$, which are the transitions that let the TBA advance to the next state, are

$$\{(s_1, v_1), (s_2, v_3)\}, \{(s_2, v_1), (s_3, v_5)\} \subset \to.$$

The set of the final states of the product $AP$ are

$$F^P = \{(s_3, v_1), (s_3, v_2), (s_3, v_3), (s_3, v_4), (s_3, v_5)\}.$$

The rest of the tuple $AP$ is given either by the WTS or by the TBA as it is already stated in the Definition 3.1.5.

Once the product has been built, a graph search algorithm can be applied. To find the shortest solution a modified version of the Dijkstra algorithm [9] is used which is described in Algorithm 2. The main adjustments happen when a new node is evaluated. There are three scenarios which can occur:

S1 The distance to the new node is within the bounds of all clock constraints. Therefore, no adjustments have to be made.

S2 The distance to the new node violates a lower constraint. In this case the distance has to be adapted such that the violation of the lower constraint does not appear any more. The underlying assumption is that one can move slower in between the environment states or wait before moving. Since the distance in the graph corresponds to time, a slower transition can then satisfy the lower constraint. However, if there is a violation of the lower constraint a solution which uses only the inputs stored in the WTS violates lower constraints. Therefore, the inputs have to be calculated again. This issue is resolved when solving Problem 2.

S3 The distance to the new node is violating an upper constraint. In this case it is assumed that a transition to that node is not possible and therefore it is set to infinity. Since the weights given in Section 3.1.3 upper bound the optimization problem for solving Problem 2, the transition still might be possible. This is important to keep in mind if the graph search does not find a solution with a finite distance.

Remark 3.1.6. In all scenarios the distance is taken either from the starting node or from the last node in which the clock has been reset.

Algorithm 2 will return a sequence of states $q \in Q$ which can be used to build the indicator function $\Omega$ described in Definition 2.4.3. The obtained constraints from Algorithm 5 are given in the following form

$$\bar{\tau}_{\text{low}}(j, i) \leq T_i - T_{j-1} \leq \bar{\tau}_{\text{up}}(j, i)$$

with $T_0 = 0$ and $j < i \leq N$.
Remark 3.1.7. For the following notation, it should be pointed out that in order to keep the length of the vectors $\tau^{\text{up}}(k, T)$ and $\tau^{\text{low}}(k, T)$ for a certain $k$ constant and make it easier to read, the missing constraints are filled up with the redundant constraint of 0 and $\infty$.

The constraints are then given by

\[
\tau^{\text{up}}(k, T) = [\hat{\tau}^{\text{up}}(1, k) + T_{1-1}, \ldots, \hat{\tau}^{\text{up}}(N, k) + T_{N}] \quad (16)
\]

\[
\tau^{\text{low}}(k, T) = [\hat{\tau}^{\text{low}}(1, k) + T_{1-1}, \ldots, \hat{\tau}^{\text{low}}(N, k) + T_{N}] \quad (17)
\]

In that way $\tau^{\text{up}}(k, T)$ and $\tau^{\text{low}}(k, T)$ store all the upper and lower constraints for the time $T_k$. This is clarified with a practical example in Section 6.1.

With the information about the indicator function and the time constraints, it is now possible to build the run according to Definition 2.4.3

\[
\text{run} = (P^{\Omega(0)}, 0, \infty)(P^{\Omega(1)}, \tau^{\text{low}}(1, T), \tau^{\text{up}}(1, T)) \ldots (P^{\Omega(N)}, \tau^{\text{low}}(N, T), \tau^{\text{up}}(N, T)) \quad (18)
\]

This solves Problem 1.

Remark 3.1.8. If, for the obtained sequence scenario $S_2$ of the adapted Dijkstra search algorithm doesn’t occur, the whole task is already finished, since it would be possible to simply apply the inputs in $\Sigma$ of the WTS. However, continuing and solving Problem 2 gives the advantage that other cost functions could be considered or the trajectory can be improved with respect to time.
Algorithm 2: Graph Search algorithm

input : Büchi WTS $A^P$
output: Sequence of states $Sequence$,
       time constraints ($\tau_{low}, \tau_{up}$),
       distance $d_s$

/* Initialize */
for $i \leftarrow 1$ to $n$ do
  $\text{dist}(i) \leftarrow \infty$
  $\text{parent}(i) \leftarrow -1$
end

$\tilde{Q} \leftarrow Q$
$\text{dist}(q_{init}) \leftarrow 0$

/* Main Loop */
while $\tilde{Q}$ is not empty do
  $k \leftarrow$ state with the smallest distance $k \in \tilde{Q}$
  if $k \in F^P$ then
    break /* Found target state search is finished */
  end
remove $k$ from $\tilde{Q}$
$Sequence \leftarrow \text{GetSequence}(k)$

$x \leftarrow \text{EvaluateClocks}(Sequence, R, k)$
foreach neighbour $v$ of $k$ with $v \in \tilde{Q}$ do
  alternative $\leftarrow \text{dist}(k) + d(k, v)$
  if alternative $< \text{dist}(v)$ then
    foreach ClockConstraint $I^P(v)$ with the matching clock $x_i \in X$ do
      $(\text{LowerConstraint}, \text{UpperConstraint}) \leftarrow I^P(v)$ /* See fig. 7 */
      $(\text{PreviousLowerConstraint}, \text{PreviousUpperConstraint}) \leftarrow I^P(k)$
      if $x_i + d(k, v) > \min(\text{PreviousUpperConstraint}, \text{UpperConstraint})$ then
        alternative $\leftarrow \infty$
        break
      end
      if $x_i + d(k, v) < \text{LowerConstraint}$ then
        alternative $\leftarrow \max(\text{alternative}, \text{LowerConstraint} - x_i + \text{dist}(k))$
      end
    end
  end
  if alternative $< \text{dist}(v)$ then
    $\text{dist}(v) \leftarrow$ alternative
    $\text{parent}(v) \leftarrow k$
  end
end

$Sequence \leftarrow \text{GetSequence}(k)$

$(\hat{\tau}_{low}, \hat{\tau}_{up}) \leftarrow \text{GetTimeConstraints}(r)$

$d_s \leftarrow \text{dist}(k)$
Algorithm 3: GetSequence

\begin{itemize}
  \item \textbf{input}: State $k$
  \item \textbf{output}: Sequence
  \item $Sequence \leftarrow k$
  \item \textbf{while} $parent(k) \neq -1$
    \item $k \leftarrow parent(k)$
    \item $Sequence \leftarrow [k, Sequence]$
  \item \textbf{end}
\end{itemize}

Algorithm 4: EvaluateClocks

\begin{itemize}
  \item \textbf{input}: Sequence
  \item \hspace{1em} Reset States $R$
  \item \hspace{1em} Current State $k$
  \item \hspace{1em} $dist(\cdot)$
  \item \textbf{output}: Clocks $x$
  \item \textbf{foreach} Clock $x_j \in X$ \textbf{do}
    \item \hspace{1em} for $i \leftarrow 1 : \text{length}(Sequence)$ \textbf{do}
      \item \hspace{2em} /* checking if State of TBA at step $i$ is in $R$ */
      \item \hspace{2em} \textbf{if} $R^f(Sequence(i))$ \textbf{and} $x_j$ has not been reset in the previous step \textbf{then}
        \item \hspace{3em} $start \leftarrow i$
      \item \hspace{2em} \textbf{end}
    \item \hspace{1em} end
    \item \hspace{1em} $x_j \leftarrow dist(k) - dist(Sequence(start))$
  \item \textbf{end}
\end{itemize}
Algorithm 5: GetTimeConstraints

input : Shortest Path r

foreach Clock $x_i \in X$ do
    $\text{start}(x_i, 1) \leftarrow 1$
    for $j \leftarrow 2$ to $\text{length}(r)$ do
        if $R^P(x_i, r(1))$ and $x_i$ has not been reset in the previous step then
            $\text{start}(x_i, j) \leftarrow j$
        else
            $\text{start}(x_i, j) \leftarrow \text{start}(x_i, j - 1)$
        end
    end
end

for $j \leftarrow 1$ to $\text{length}(r)$ do
    forall $m$ do
        $X_m \leftarrow \emptyset$
    end
end

for $m \leftarrow 1$ to $j$ do
    foreach $x_i \in X$ do
        if $\text{start}(x_i, j) = m$ then
            add $x_i$ to $X_m$
        end
    end
end

foreach $m$ do
    $\tau_{\text{low}}(m, i) \leftarrow$ highest lower Constraint of $I^P(r(j))$ matching to the clocks of $X_m$
    $\tau_{\text{up}}(m, i) \leftarrow$ lowest higher Constraint of $I^P(r(j))$ matching to the clocks of $X_m$
end
Figure 6: The figure illustrates how the WTS and the TBA are put together when the product is calculated. The area enclosed by the dashed line is a location of the TBA. Inside of each location there is a WTS where the state which advances the TBA is removed. The sink location $s_4$ on the left is reached by every state in $s_2$ and $s_1$. The starting location is $s_1$ and the initial state of the Büchi WTS is $(s_1, v_1)$. 

24
Figure 7: Displayed is a simplified version of the graph product $A^P$. The figure illustrates that for a transition from $q_1$ to $q_2$ the clock $x_1$ must fulfill both upper constraints. Suppose the current state is $q_1$ and the current time is $x_1 = T_1$. Recall that $T_k$ is the earliest time for entering the next state. $T_2$ is approximated by the value of $x_1$ and $d^P(q_1, q_2)$. $q_2$ should be entered before 10 time units. However, since right before $T_2$ the state is still $q_1$, we have that $T_2 \leq 6$ because otherwise the clock constraint of $q_1$ would be violated and the MITL formula would be violated. If the clock gets reset in $q_2$ then only the clock constraint of $q_1$ has to be taken into account.
3.2 Solution of Problem 2

As already motivated at the end of last section, solving Problem 2 can be a necessary step in the solution. Another motivation which hasn’t been mentioned yet is that this problem can be efficiently solved, since many solvers work best with a good initial guess which is already given by the WTS. Therefore, the optimality of the trajectory can be improved in an efficient way. The solution approach for Problem 2 is to reformulate the obtained run given by the previous section into an optimal control problem. A good introduction to optimal control can be found in [16].

3.2.1 Formulation of the Optimization Problem

The constraints of the optimal control problem are given in several parts. The first parts (19a) and (19b) correspond to the dynamics of the agent (1). (19c) is given by the constraints from Definition 2.1.1. Finally, the constraints from (19d) to (19h) represent the conditions from the run which go from 1 up to N.

\[
\begin{align*}
\dot{x} &= f(x, u) \quad \text{(19a)} \\
\begin{bmatrix} x(0) = x_0 \end{bmatrix} &\in \mathbf{U} \quad \text{(19b)} \\
x(t) &\in \mathbf{P}^{\Omega(0)} \quad \text{for } t \in [0, T_1) \quad \text{(19c)} \\
\tau_{low}(1, T) &\leq T_1 \leq \tau_{up}(1, T) \quad \text{(19d)} \\
\vdots \\
x(t) &\in \mathbf{P}^{\Omega(N-1)} \quad \text{for } t \in [T_{N-1}, T_N) \quad \text{(19f)} \\
\tau_{low}(N, T) &\leq T_N \leq \tau_{up}(N, T) \quad \text{(19g)} \\
x(T_N) &\in \mathbf{P}^{\Omega(N)} \quad \text{(19h)}
\end{align*}
\]

where \( T_1 \) denotes a vector of size \( N \) stacked with the time \( T_1 \).

As already indicated at the end of Section 3.1, various cost functions can be considered. It is up to the user’s discretion on how to choose the costs. Most often one is interested in robustness or time optimality. Formal examples of cost functions are presented in Section 6. However, for formulating the optimal control problem the cost function is not specified which yields

\[
\begin{align*}
J = \min_u \int_0^{T_N} f_0(x, u, t) dt + F_0(x(T_N), T_N) \quad \text{(20a)} \\
\text{s.t. } (19) \quad \text{(20b)}
\end{align*}
\]

where \( f_0 \) is the cost function for the running costs and \( F_0 \) describes the terminal cost.

For a more convenient formulation (20) is rewritten by replacing all set operations by the inequality

\[
\begin{align*}
\dot{x} &= f(x, u) \quad \text{(19a)} \\
\begin{bmatrix} x(0) = x_0 \end{bmatrix} &\in \mathbf{U} \quad \text{(19b)} \\
x(t) &\in \mathbf{P}^{\Omega(0)} \quad \text{for } t \in [0, T_1) \quad \text{(19c)} \\
\tau_{low}(1, T) &\leq T_1 \leq \tau_{up}(1, T) \quad \text{(19d)} \\
\vdots \\
x(t) &\in \mathbf{P}^{\Omega(N-1)} \quad \text{for } t \in [T_{N-1}, T_N) \quad \text{(19f)} \\
\tau_{low}(N, T) &\leq T_N \leq \tau_{up}(N, T) \quad \text{(19g)} \\
x(T_N) &\in \mathbf{P}^{\Omega(N)} \quad \text{(19h)}
\end{align*}
\]
This will lead to a total number of

\[ k_{\text{con}} = 2n + k_u + 2N^2 + \sum_{i=0}^{N} k_{\Omega(i)} \]

constraints. Further, (21) has now the form of an optimal control problem with state variable inequality constraints (SVIC). These problems are well known to the literature. A theoretical review on the problem is given in [12]. Analytic solutions for these kind of problems exists only in special cases. For the general case a numerical approach is usually required.

### 3.2.2 Computational Approach

As stated in [5], the methods for solving an optimal control problem numerically can be divided into two different approaches. The first one is the indirect approach which uses the optimality conditions given by Pontryagin’s maximum principle. This yields a very precise solution but often requires a lot of computational power as well as a good initial guess. Next is the direct approach, which is numerically more robust and requires less computing power. However, the solution might not be as precise as the indirect approach.

In this thesis the second approach has been chosen because of the robustness and the less required computing power. In particular, the optimization method presented in the following is a direct collocation method, where the method for the collocation is the implicit midpoint method. The resulting problem is then solved by an interior point solver. A detailed description follows.

For this approach the state \( \mathbf{x} \) and the control input \( \mathbf{u} \) first have to be discretized. Each time interval \([T_k, T_{k+1}]\) is discretized with \( M \) points

\[
\Delta t_k = \frac{T_k - T_{k-1}}{M} \quad (22)
\]

\[
\mathbf{x}^k_j = \mathbf{x}(T_{k-1} + j\Delta t_k) \quad (23)
\]

\[
\mathbf{u}^k_j = \mathbf{u}(T_{k-1} + j\Delta t_k) \quad (24)
\]

where the subscript \( k \in \{1, \ldots, N\} \) indicates the \( k \)-th interval and the superscript \( j \in \{0, \ldots, M - 1\} \) denotes the \( j \)-th discretized point in that interval. In total, this results in \( N \) different discretization times
\( \Delta t_k \), \( NM \) discretization points for the input \( u \) and \( NM + 1 \) discretization points for the state variable \( x \). The state variable has 1 more discretization point due to the fact that the last point of the trajectory is not influenced by the last input which can therefore be neglected. Note that the initial time \( T_0 \) is 0.

The discretized optimization problem then results in

\[
J = \min_{\hat{x}} \sum_{k=1}^{N} \sum_{j=1}^{M} f_0(x^j_k, u^j_k, T_k + j \Delta t_k) \Delta t_k + F_0(x_N, T_N)
\]

s.t.

\[
x_0^1 - x_0 = 0
\]

\[
\frac{1}{\Delta t_k} (x^i_{k+1} - x^i_k) - f \left( \frac{x^i_{k+1} + x^i_k}{2}, u^i_k \right) = 0
\]

\[
g(u^i_k) \leq 0
\]

\[
\tau_{low}^i(k, T) - T_k \leq 0
\]

\[
T_k - \tau_{up}^i(k, T) \leq 0
\]

\[
p^{i(k-1)}(x^i_k) \leq 0
\]

\[
p^{i(N)}(x_N) \leq 0
\]

where the constraints are stated generically due to readability with exception of the last point which is \( x_N \). Note that for an easy notation in (25c) the equation

\[
x^M_k = x^0_{k+1}
\]

holds as well as \( x^M_N = x_N \). Further, instead of optimizing over \( u \), the optimization variable is

\[
\hat{x} = \begin{bmatrix}
  x^0_1 \\
  \vdots \\
  x^0_{M-1} \\
  x^M_N \\
  \vdots \\
  u^0_1 \\
  \vdots \\
  u^M_{M-1} \\
  T
\end{bmatrix}
\]

This makes the problem larger but at the same time computable. The number of optimization variables is

\[
k_{opt} = N + NMn + NMm + n.
\]

As for the constraints, equation (25b) yields \( n \) equality constraints, (25c) the so-called collocation conditions give \( NMn \) equality constraints, (25d) gives \( NMk_u \) constraints, (25e) and (25f) do not change compared to the previous optimization problem (21) and still contribute \( 2N^2 \) constraints. Finally, (25g) and (25h) cause \( k_{p_u(N)} + \sum_{k=0}^{N-1} k_{p_u(k)} \) \( M \) constraints. This results in a total number of

\[
k_{disc} = 2N^2 + k_{p_u(N)} + M \left( N(k_u + n) + \sum_{k=1}^{N-1} k_{p_u(k)} \right)
\]

constraints. The optimization problem (25) is formulated in such a way that it can be solved using an optimization program e.g. MATLAB or, as it was done here, Julia JuMP which is free and can be found on [1]. As for the solver itself, the IpOpt Solver is used which is an interior point method. This solver
is widely used and is considered one of the more robust solvers. A detailed description can be found in [26] or online at [27]. Note that not only Problem 2 is solved in this fashion, but also the optimization problem (13) described in Section 3.1.3, can be solved in this way. In addition to the input a solution will also give the optimal trajectory \( x^*(t) \). Finding a solution for the optimization problem (13) and (20) can be very difficult, especially if the dynamic is nonlinear. The rest of this section takes a closer look at the solver and gives practical hints on how to obtain a solution if initial attempts fail.

The solver also has theoretical convergence properties stated in [25] if certain assumptions are fulfilled. Before stating these assumptions, some equations have to be derived.

When the problem given in (25) is translated into code, the program reformulates the problem by taking all the inequality constraints (25d) - (25h), stacking them into a vector \( \hat{g} \) and adding an auxiliary variable \( \hat{w} \geq 0 \) s.t. \( \hat{g}_j \pm \hat{w}_j = 0 \) where the index describes the the \( j \)-th constraint. Enhancing the optimization variables with the auxiliary variables gives

\[
\tilde{x} = \begin{bmatrix} \hat{x} \\ \hat{w} \end{bmatrix}
\]

This results in a compact problem description

\[
\begin{align*}
J &= \min_{\tilde{x}} \hat{f}(\tilde{x}) \\
\text{s.t.} \quad &\hat{c}(\tilde{x}) = 0 \\
&\tilde{C}^{\hat{w}} \tilde{x} \geq 0
\end{align*}
\]

where

\[
\hat{f}(\tilde{x}) = \sum_{k=1}^{N} \sum_{j=1}^{M} f_0(x^j_k, u^j_k, T_k + j \Delta t_k) \Delta t_k + F_0(x_N, T_N)
\]

and \( \tilde{C}^{\hat{w}} \tilde{x} = \tilde{w} \). This yields the following Lagrangian

\[
\hat{L} = \hat{f}(\tilde{x}) + \lambda_1^T \hat{c}(\tilde{x}) + \lambda_2^T \tilde{C}^{\hat{w}} \tilde{x}.
\]

The following assumptions are now taken from [25]. Given a sequence \( \{\tilde{x}_k\} \) generated by the IpOpt Solver, we have that:

A1 The feasibility restoration phase of the IpOpt algorithm always terminates successfully.

A2 There exists an open set \( C \subseteq \mathbb{R}^{n_{ip}} \) with \( [\tilde{x}_k, \tilde{x}_k + \alpha_{k_{\max}}^i \tilde{d}_k] \subseteq C \), so that \( \hat{f} \) and \( \hat{c} \) are differentiable on \( C \), and their function values as well as their first derivatives are bounded and Lipschitz-continous over \( C \). Where \( \tilde{d}_k \) is the search direction and \( \alpha_{k_{\max}}^i \) is the maximal step size.

A3 The iterates \( \{\tilde{x}_k\} \) are bounded.

A4 The matrices \( W_k \) approximating the Hessian of the Lagrangian (27) of the NLP (26) are uniformly bounded.

A5 At all feasible points \( \tilde{x} \in C \), the gradients of the active contraints \( \nabla \hat{c}_1(\tilde{x}), ..., \nabla \hat{c}_{m_{ip}}(\tilde{x}) \), and \( \tilde{C}^{\hat{w}}_i \) for \( i \in \{j : \tilde{C}^{\hat{w}}_j \tilde{x} = \hat{w}^{(j)} = 0\} \) are linearly independent; \( \tilde{C}^{\hat{w}}_i \) being the \( i \)-th row vector of the matrix \( \tilde{C}^{\hat{w}} \).
A6 The matrices $W_k + \mu \hat{W}_k$ are uniformly positive definite on the null space of the Jacobian $\nabla \hat{c}|_{\hat{x}_k}$, with $\hat{W}_k = \begin{bmatrix} 0 & 0 \\ 0 & \text{diag}(\hat{w})^{-2} \end{bmatrix}$ and $\mu$ being the multiplier for the logbarrier function.

A7 There exist constants $\delta_0, \delta_\omega > 0$, such that, whenever the restoration phase is called because of the linear system becoming singular or the trial step size being to small in an iteration $k$ with $\theta(\hat{x}_k) \leq \delta_\theta$, it returns a new iterate with $\hat{w}_{k+1} \geq \hat{w}_k$ for all components satisfying $\hat{w}_k \leq \delta_\omega$.

**Theorem 2.** [25] If all the assumptions A1-A7 are satisfied then it is guaranteed that, in the limit, the IpOpt solver will find a feasible point $\tilde{x}$ such that
\[
\lim_{k \to \infty} c(\tilde{x}_k) = 0 \\
\lim_{k \to \infty} C^\omega \tilde{x}_k \geq 0
\]

The proof can be found in [25].

(A2) and (A4) gives some indication about the smoothness of the given constraints as well as the cost function. In order to fulfill this property, the user should verify that all the given functions are at least two times continuously differentiable. This can of course be violated by a given problem (e.g. non-smooth cost functions), however, it is advisable to approximate the non-smooth functions. These are also the only two assumptions which can be directly influenced by the user.

(A5) may be violated due to an infeasible problem which can’t be known beforehand. Also unknown and dependent on the problem, is (A3) which ensures that none of the optimizer diverges. (A6) gives some attraction property. (A7) keeps the logbarrier cost function bounded. (A1) is the most crucial assumption. Rather than just giving some verifiable property it states that a part of the algorithm never fails.

Note that all of the assumptions except for (A1) depend on certain iterations. Therefore, it can happen that in practice some of these assumptions may be violated on some parts of the search space but the IpOpt-solver is still able to find a solution since the iterates never enter these parts. If the solver fails, one of the following measures usually helps under the assumption that the problem is feasible.

- Use multiple restarts with different initial conditions for $\hat{x}$. This is especially useful if the new initial conditions are not simply random guesses. A good initial condition could be for instance a trajectory generated through the user by a good understanding of the system dynamics or by taking the solution of an easier convex problem which is both done in the case study Section 6. In the best case, a good initial condition is already a feasible solution, thus making it easier for the solver to further optimize the solution.

- The cost function can sometimes ‘lead’ the iterates in a ‘wrong’ direction from which the IpOpt solver can not recover. Therefore it is useful to set the cost function $f(\tilde{x})$ to zero, use the obtained solution as a warm start solution and then try to solve the Problem with the original cost function $f(\tilde{x})$.

- Since the linear inequality constraints e.g. $a \leq x^i \leq b$, are always satisfied due to the log barrier structure of the algorithm, it can help to reduce the search space by tightening the constrains.

These are rather practical tips and especially useful for the nonlinear case. In Section 3.2.3 a way is presented to find a good initial guess for linear systems which has not failed a single time during the case studies.

### 3.2.3 Initial Guess for the Linear Case

In this section, a way is presented on how to obtain a good initial guess in order to solve the optimization problem (25). It is assumed for this section that the dynamics of (1) is linear
\[
f(x, u) = Ax + Bu,
\]
as well as the input constraints

\[ g(u) = C_u u - b_u \leq 0. \]

Even in this case, the problem (13) and (21) both result in a non-linear and non-convex problem due to (25c) since \( \Delta t_k \) and \( x_k^i \) are both optimization variables. In the best case scenario this leads to a linear program with quadratic constraints where the quadratic part is caused by

\[ \Delta t_k a^i \frac{1}{2} (x_k^{i+1} + x_k^i) + \Delta t_k b^i u_k^i \]

(28)

where \( a^i \) and \( b^i \) denote the \( i \)th row of the matrix \( A \) or \( B \).

**Remark 3.2.1.** Note that for (21) a non-convex, non-linear cost function could also bring some difficulties in solving the optimization problem. However for now let’s assume that this is not the case.

Differentiating (28) two times for the vector \( \hat{x} \) yields the jacobian

\[
\hat{Q}^i = \begin{bmatrix}
0 & \cdots & 0 & -\frac{1}{2M} a^i & -\frac{1}{2M} a^i & 0 & \cdots & 0 & \frac{1}{M} b^i & 0 & \cdots & 0 \\
0 & \cdots & 0 & \frac{1}{2M} a^i & \frac{1}{2M} a^i & 0 & \cdots & 0 & \frac{1}{M} b^i & 0 & \cdots & 0 \\
\end{bmatrix}
\]

where 0 are zero blocks. \( \hat{Q}^i \) is a non-symmetric matrix with entries only in the \( 2M(N-1) + 1 + (k-1) \) and \( 2M(N-1) + 1 + k \) row depending on the time interval \( \Delta t_k \).

**Definition 3.2.1.** A matrix \( M \) is positive (negative) semidefinite if and only if \( z^* M z \) is real and \( z^* M z \geq 0 \) \( z^* M z \leq 0 \) for all non-zero complex column vectors \( z \). If \( M \) is neither positive semidefinite nor negative semidefinite, \( M \) is called indefinite.

If this is now applied to the matrix \( \hat{Q}^i \) one can easily see that the matrix is indefinite. This can also be seen when taking a closer look on equation (28) since positive semi definiteness of \( \hat{Q}^i \) would imply that the \( i \)th entry of the vector of the sequence \( x_k^i \) can only grow or remain equal which is not true for a general linear system. Therefore, the optimization problem is not convex. It is generally known that non-convex problems are hard to solve. The idea to tackle this problem is to solve a convex problem beforehand in order to get a good initial guess. From a good initial guess, which already satisfies almost all constraints of the optimization problem, it is much more likely to converge to a solution compared to an initial guess which violates all the constraints.

First, each \( \Delta t_k \) is set to a fixed value \( \delta_k \) which over approximates the optimal \( \Delta t_k^* \) and is simply guessed. As a consequence, (28) will become linear. Furthermore, the last entries of \( \hat{x} \), which are the components of the time vector \( T \), are fixed with

\[ T = \theta \]

where \( \theta \) is the vector of predefined guessed values for \( T \) according to the guess of \( \Delta t_k \), since both variables \( T \) and \( \Delta t_k \) are related via the equation (22). If the cost function (25a) is convex and time-independent, it can be kept as the cost. Otherwise, it is replaced with the quadratic costs

\[ J = \min_{\hat{x}} \sum_{k=1} \sum_{j=0} p_{edge}(x_k^j)^2 \]
where \( p_{\Omega(k)}(x^j_k) = a_{\text{edge}}^T x^j_k + b_{\text{edge}} \) is the constraint of \( \Pi_{\Omega(k)} \) which is the line separating \( \Pi_{\Omega(k-1)} \) and \( \Pi_{\Omega(k)} \). This means that
\[
 p_{\Omega(k)}(x(T_k)) = 0.
\]
This then results in the new problem
\[
 J = \min \sum_{k=1}^{N} \sum_{j=1}^{n} p_{\text{edge}}(x_k^j)^2
\]
\[
\text{s.t.} \quad \frac{1}{\delta_k}(x_k^{j+1} - x_k^j) - A \frac{1}{2}(x_k^{j+1} + x_k^j) - Bu_k^j = 0
\]
\[
x_0 - x_0 = 0
\]
\[
 C_u u_k^j - b_u \leq 0
\]
\[
p_{\Omega(k-1)}(x_k^j) \leq 0
\]
\[
p_{\Omega(N)}(x_N) \leq 0.
\]
Since according to Definition 2.2.2 \( p_{\Omega(k)} \) is linear, the constraints (29b)-(29f) are linear and therefore convex. Note that the constraints
\[
\tau_{\text{low}}(k, \theta) - \theta_k \leq 0
\]
\[
\theta_k - \tau_{\text{up}}(k, \theta) \leq 0
\]
have been neglected since the time \( T = \theta \) is fixed. The cost function
\[
p_{\text{edge}}(x_k^j)^2 = (a_{\text{edge}}^T x_k^j + b_{\text{edge}})(a_{\text{edge}}^T x_k^j + b_{\text{edge}})
\]
\[
= (x_k^j a_{\text{edge}}^T + b_{\text{edge}}^T)(a_{\text{edge}}^T x_k^j + b_{\text{edge}})
\]
\[
= x_k^j a_{\text{edge}}^T a_{\text{edge}} x_k^j + 2b_{\text{edge}} a_{\text{edge}} x_k^j + b_{\text{edge}}^2
\]
with \( a_{\text{edge}} a_{\text{edge}}^T \succeq 0 \)

is also convex, thus making the whole problem convex and easy to solve. Note that the solution of (29) satisfies all the constraints of (25) except for (25e) and (25f). Thus, solving (29) as an initial guess will result in much better performance regarding time and robustness of the IpOpt-solver. This has been observed throughout the case study in Section 6.

4 Robustification Approach

In a real-life application the system is often disturbed by some unknown forces. Simply applying the input obtained by solving Problem 2 as presented in the previous section will most likely not satisfy the MITL formula for a disturbed system. It is assumed that the dynamics of (1) is additively disturbed
\[
\dot{x} = f(x, u) + w
\]
where \( w \) is the disturbance. There are two methods which improve the robustness of the solution and make sure that the given problem in Section 2.5 is also fulfilled for the disturbed system. A feedback controller can limit the influence of disturbances online, but also offline measures can be taken when planning the trajectory.
4.1 Controller

For a general non-linear system it is hard to determine a controller which stabilizes every system. When it comes to non-linear systems, the controllers often depend highly on the nonlinearity itself. At this point, control techniques like model predictive control or backstepping are usually mentioned, but a good controller for a non-linear system depends highly on the given system. However, for the linear case

\[ f(x, u) = Ax + Bu + w \]  

(33)

statements can be made.

The requirements for the controller are

R1 The disturbed system (33) shall always stay within a certain range of the optimal trajectory.

R2 Since the input \( u \) is constrained, the input action caused by the controller should be as small as possible.

Both \( H_{\infty} \) as well as the Linear Quadratic Regulator (LQR) provide an effective way to tune this problem. During the case study the \( H_{\infty} \) showed slightly superior performance. Both designs return a linear controller which is why it is possible for a bounded disturbance to estimate how big the error will become and how much of the "input action" needs to be reserved for the controller.

Figure 8: The picture shows the signal flow for a given controller together with a given optimal trajectory.

Given a controller \( K(s) \), we consider the transfer function \( N(s) \) from \( w(t) \) to \( e(t) \). The matrices \( A_N, B_N, C_N, D_N \) realize the transfer-function \( N(s) \). Further, all eigenvalues of \( A_N \) are in the left half plane, i.e. \( \text{eig}(A_N) \in \mathbb{C}^- \), since otherwise the controller wouldn’t stabilize the system. We consider disturbance to be a sinusoidal signal

\[ w(t) = \sum_{j=1}^{N} w_j e^{i\omega_j t} \]

with \( \omega_j \) appearing pairwise, \( w_j \) being a real-valued vector. This yields for the error which is initially
\[ e(t) = \int_0^t C_N e^{A_N(t-\tau)}B_N w(\tau) d\tau + D_N w(t) \]

\[ = \int_0^t C_N e^{A_N(t-\tau)}B_N \sum_{j=1}^N w_j e^{i\omega_j \tau} d\tau + D_N \sum_{j=1}^N w_j e^{i\omega_j t} \]

\[ = \sum_{j=1}^N \int_0^t C_N e^{A_N(t-\tau)}B_N w_j e^{i\omega_j \tau} d\tau + D_N w_j e^{i\omega_j t} \]

\[ = \sum_{j=1}^N \left[ C_N e^{A_N(t+i\omega_j -(A_N)^{-1}B_N w_j} - \sum_{j=1}^N C_N e^{A_N t+i\omega_j I - A_N}^{-1}B_N w_j + \sum_{j=1}^N D_N w_j e^{i\omega_j t} \right] \]

\[ = \sum_{j=1}^N N(i\omega_j) w_j e^{i\omega_j t} - \sum_{j=1}^N C_N e^{A_N(t+i\omega_j I - A_N)^{-1}B_N w_j} \]

steady state response \[ \rightarrow 0 \text{ as } t \to \infty \]

Taking the norm of \( e(t) \) yields

\[ \|e(t)\| = \left\| \sum_{j=1}^N C_N e^{i\omega_j t}(i\omega_j I - A_N)^{-1}B_N w_j - \sum_{j=1}^N C_N e^{A_N t}(i\omega_j I - A_N)^{-1}B_N w_j \right\| + \left\| \sum_{j=1}^N D_N w_j e^{i\omega_j t} \right\| \]

\[ \leq \sum_{j=1}^N \|C_N e^{i\omega_j t}(i\omega_j I - A_N)^{-1}B_N w_j - C_N e^{A_N t}(i\omega_j I - A_N)^{-1}B_N w_j + D_N w_j e^{i\omega_j t} \| \]

\[ \leq \sum_{j=1}^N \|C_N(i\omega_j I - A_N)^{-1}B_N - C_N e^{(A_N - I\omega_j I)^{-1}B_N} + D_N \| \| e^{i\omega_j t} \| \| w_j \| \]

\[ = \sum_{j=1}^N \| C_N(I - e^{(A_N - I\omega_j I)^{-1}B_N} + D_N) \| \| w_j \| \]

\[ \leq \sum_{j=1}^N \max_{\omega_j} \| L(\omega_j, t) \| \| w_j \| \]

\[ = \max_{\omega_j} \| L(\omega_j, t) \| \sum_{j=1}^N \| w_j \|. \]

As mentioned earlier for a bounded disturbance, i.e.

\[ \sum_{j=1}^N \| w_j \| \leq \varepsilon_w, \]
it is guaranteed that the error will always be smaller or equal than

$$||e(t)|| \leq \max_{i\omega,t} ||L(i\omega,t)|| \eps_w = \eps_e.$$  

The estimation for the input $u$ follows the same steps. This yields

$$||u(t)|| \leq \max_{i\omega,t} ||L_u(i\omega,t)|| \eps_w = \eps_u$$

where $L_u(i\omega,t)$ is the same function as $L(i\omega,t)$ but built with the transfer function $N_u(s)$ from $w(t)$ to $u(t)$. Once $\eps_e$ and $\eps_u$ are obtained, it is important to factor them in during the offline calculation of the trajectory. As for the input constraints they get tightened by

$$U_{\text{new}} := \{ u \in \mathbb{R}^n | g(u + \eta) \leq 0 \}$$

where $||\eta|| \leq \eps_u$. It is clear that $U_{\text{new}} \subseteq U$, which ensures at the same time that there will always be 'enough' input for the controller such that the disturbances can be controlled. Taking $\eps_e$ into account is a little bit more difficult and discussed in detail in the next section.

4.2 Minimal Distance

Considering the problem given in the Section 2, it is often of interest to have a minimal distance to the edge of the environment state $P(x(t))$ in which the dynamical state is currently located. This means, that except for the transition between two environment states there should be a minimum distance to every other environment state as it already has been done in previous works such as [22]. Next to that there is also the simple thought that given a disturbed system as in (33), a trajectory for this system which is close to the edge might leave the current environment state, due to disturbances, and therefore violate the MITL formula. A minimum distance to the edge will further restrict the time bounds obtained in a run. Suppose that the system is disturbed. The error for following a given optimized trajectory

$$e(t) = x^*(t) - x(t)$$

where $x^*(t)$ is the optimal trajectory obtained by solving Problem 2 will not be greater than a certain value

$$||e(t)|| \leq \eps_e < \eps.$$  

This motivates the definition for the robust environment state, where the minimum distance between a point on the edge of the environment state $P^i$ and a point on the edge of its robust environment state $P^i_\eps$ is $\eps$.

**Definition 4.2.1.** Given an environment state according to Definition 2.2.2

$$P^i = \{ x \in \mathbb{R}^n | p^i(x) \leq 0 \}$$

the robust environment state is defined as

$$P^i_{\eps} = \{ x \in \mathbb{P} | p^i(x) \leq -\eps \}.$$  

Recall that the time $T_k$ is the time when a new environment state is entered. Due to the disturbance this time is now unknown. This motivates the introduction of two new times. $T_{\text{save}}^{k-1}$ denotes the time at which the trajectory leaves the robust environment state $P^i_{\eps}$. Further, $T_{\text{trans}}^k$ defines the time when the next robust environment state $P^i_{\eps}$ is entered. The robust environment states as well as the new times are both illustrated in Figure 9. It is easy to see that

$$T_{\text{save}}^{k-1} < T_k \leq T_{\text{trans}}^k.$$  

35
Without loss of generality we now consider the $j$th time constraint for the $k$th time $T_k$

$$\hat{\tau}_{low}(j+1,k) + T_j \leq T_k \leq \hat{\tau}_{up}(j+1,k) + T_j$$

where $\hat{\tau}_{low}(j+1,k)$ and $\hat{\tau}_{up}(j+1,k)$ are taken from (17) and (16). Performing now a simple worst case estimation yields the time constraints for $T_{k-1}^{\text{save}}$ and $T_{k}^{\text{trans}}$. For the lower bound the biggest value $T_j$ can become is $T_{trans,j}$ and the lowest value $T_k$ can have is $T_{save,k-1}$. Using the same idea for the upper limit results in the new constraints

$$\hat{\tau}_{low}(j+1,k) + T_{trans,j} \leq T_{save,k-1} < T_{trans,k} \leq \hat{\tau}_{up}(j+1,k) + T_{save,k-1}.$$ 

This also has certain implication on the optimization problem and the weight generation, as well as on the optimization problem (21). Beginning with the calculation of the weights for the WTS (13), the new optimization problem is now

$$J = \min_u \int_0^T dt \quad \text{(34a)}$$

s.t. \begin{align*}
\dot{x} &= f(x, u) \quad \text{(34b)} \\
x(0) &= v_i \quad \text{(34c)} \\
u &\in U_{\text{new}} \quad \text{(34d)} \\
x(t) &\in P_{\varepsilon}(v_i) \text{ for } t \in [0, T_{\text{save}}) \quad \text{(34e)} \\
x(t) &\in (P(v_i) \cup P(v_j))_{\varepsilon} \text{ for } t \in [T_{\text{save}}, T_1) \quad \text{(34f)} \\
x(t) &\in (P(v_i) \cup P(v_j))_{\varepsilon} \text{ for } t \in [T_1, T_{\text{trans}}) \quad \text{(34g)} \\
x(t) &\in P_{\varepsilon}(v_j) \text{ for } t \in [T_{\text{trans}}, T] \quad \text{(34h)} \\
x(T) &= v_j \quad \text{(34i)}
\end{align*}

where $P_\varepsilon(v_i) = P_k^\varepsilon$ is the the robust environment state of $P(v_i) = P_k$. The set $(P(v_i) \cup P(v_j))_{\varepsilon}$ is the union of $P(v_i)$ and $P(v_j)$ where every point of the set has a minimal distance $\varepsilon$ to the edge of $(P(v_i) \cup P(v_j))$ as it is illustrated in Figure 9. For the times the relation

$$T_{\text{save}} < T_1 < T_{\text{trans}} \leq T$$

holds. If the point $v_j$ is on the edge of $P(v_j)_{\varepsilon}$ then

$$T_{\text{trans}} = T.$$
If not, $T^{\text{trans}}$ is overestimated by the weight $d(v_i, v_j) = T^*$ which is, since it is bounded from above, increasing the conservatism but not generating any false positive.

The robust version for optimization problem (20) is

$$J = \min_u \int_0^{T_N} f_0(x, u, t) dt + F_0(x(T_N), T_N)$$

s.t.

$$\dot{x} = f(x, u)$$

$$x(0) = x_0$$

$$u \in U_{\text{new}}$$

$$x(t) \in \mathbb{P}^{Ω(0)}$$

$$x(t) \in (\mathbb{P}^{Ω(0)} \cup \mathbb{P}^{Ω(1)}) \setminus \mathbb{P}^{Ω(1)}$$

$$x(t) \in (\mathbb{P}^{Ω(0)} \cup \mathbb{P}^{Ω(1)}) \setminus \mathbb{P}^{Ω(0)}$$

$$\tau_{\text{low}}(1, T) \leq T_{\text{save}}$$

$$T_{\text{trans}} \leq \tau_{\text{up}}(1, T)$$

$$\vdots$$

$$x(t) \in \mathbb{P}^{Ω(N-1)}$$

$$x(t) \in (\mathbb{P}^{Ω(N-1)} \cup \mathbb{P}^{Ω(N)}) \setminus \mathbb{P}^{Ω(N)}$$

$$x(t) \in (\mathbb{P}^{Ω(N-1)} \cup \mathbb{P}^{Ω(N)}) \setminus \mathbb{P}^{Ω(N-1)}$$

$$\tau_{\text{low}}(N, T) \leq T_{\text{save}}$$

$$T_{\text{trans}} \leq \tau_{\text{up}}(N, T)$$

$$x(T_{\text{trans}}) \in \mathbb{P}^{Ω(N)}$$

The structural difference of this optimization problem compared to the (20) is the new constraints for the time in between $T_{\text{save}}^k$ and $T_{\text{trans}}^k$.

5 Extension to the Multi-Agent Case

So far the problem described in Section 2 only considers single agents. In the following, the problem is extended to multi agent systems where each agent has its own MITL task. Further, there is also a global MITL task which requires multiple agents to work together. The solution is obtained in a centralized fashion.

In the following Section 5.1 the problem is described in greater detail and in Section 5.2 the solution approach is presented.

5.1 Problem Definition

First we give a formal definition for a multi agent system. Afterwards the problem which is considered for the multi agent case is stated.

A multi agent system consist of multiple agents with the dynamics of (1) and is formally defined as

**Definition 5.1.1.** A multi agent system consist of $N_m$ agents, where for each $i \in \{1, ..., N_m\}$, the $i$th agent is governed by the dynamics

$$\dot{x}_i(t) = f_i(x_i(t), u_i(t)) \quad x_i(0) = x_i^0 \quad t \geq 0 \quad x_i(t) \in \mathbb{R}^n.$$
It is assumed that all agents are in the same given environment $E$. Further, for every agent there exists a MITL formula $\phi_i$ for $i \in \{1, ..., N_m\}$.

Each formula $\phi_i$ makes only statements about the $i$th agent. In other words, solving the task $\phi_i$ does not require another agent $j \neq i$. Cooperative tasks which require multiple agents to work together are expressed in the global MITL formula $\phi_{\text{global}}$. This yields the following problem definition.

Given a multi agent system as it is defined in Definition 5.1.1 together with an environment $E$, the goal is to find an input $u_i$ for each agent, such that the MITL task $\phi_i$ is satisfied for each agent, as well as the global task $\phi_{\text{global}}$.

The solution to this problem is described in the next section together with the necessary definitions for solving this problem.

5.2 Solution Approach

Basically, the solution approach follows the steps presented in the single agent case. However, since the cooperative task requires the agents to work together and do certain actions at specific times, some necessary extra steps have to be taken in order to fulfill the global MITL task $\phi_{\text{global}}$. Therefore, the agents have to be synchronized. Thus, it is not sufficient to simply solve the single agent problem for each agent multiple times.

The strategy of dividing the problem into two sub problems remains. This means that in the first part of the solution approach a graph is build up from which a run $\mathcal{R}_i$ for each agent can be obtained. It is important that the resulting graph considers all agents, in order to solve the cooperative task. In the second part, the optimization problem is built from all runs $\mathcal{R}_i$. The solution to the optimization problem then solves the problem. It is again emphasized that the optimization problem considers all agents at the same time. This will ensure that the solution satisfies the cooperative task $\phi_{\text{global}}$.

5.2.1 Building the Graph and Graph Search

The solution approach starts with creating a WTS $T_i$ according to Definition 3.1.2 for each agent. In the next step, for each agent its MITL task

$$\phi_i \text{ for } i \in \{1, ..., N_m\}$$

is translated into the TBA $A_i$. From the obtained WTS and TBA the product is built. This yields the Büchi WTS $A_i^P = T_i \otimes A_i$ according to Definition 3.1.5. So far there was no difference to the single agent case. This changes in the following. Now, the graph product of the $A_i^P$’s is build according to the following definition, which is a slight modification of [21].

**Definition 5.2.1.** Given $N_m$ Büchi WTS $A_1^P \ldots A_{N_m}^P$ according to Definition 3.1.5 their product $A_C = A_1^P \otimes \ldots \otimes A_{N_m}^P = (Q_C, Q_C^{\text{init}}, \preceq_C, F_C, A_P C, L_C, I_C, \mathcal{X}_C, R_C, d_C)$ is called collective Büchi WTS and is defined as:

- $q_C \in Q_C = Q_1 \times \ldots \times Q_{N_m}$.
- $Q_C^{\text{init}} = Q_1^{\text{init}} \times \ldots \times Q_{N_m}^{\text{init}}$.
- $q_C = (q_1, ..., q_{N_m}) \preceq_C q_C' = (q_1', ..., q_{N_m}')$ iff $\exists$ at least one $q_i \neq q_i'$ and $\forall q_i \neq q_i'$ the transition is allowed by the Büchi WTS $A_i$, i.e. $q_i \preceq_i q_i'$. 

38
The weight for this transition is obtained by

\[ d_C(q_C, q'_C) = \max_i d_i^F(q_i, q'_i) \]

where \( q_i \neq q'_i \).

- \( F_C = F_1^P \times \ldots \times F_{N_m}^P \).
- \( AP_C = \bigcup_{i=1}^{N_m} AP_i \).
- \( L_C(q_C) = \bigcup_{i=1}^{N_m} L_i(q_i) \).
- \( I_C(q_C) = \bigcup_{i=1}^{N_m} I_i^F(q_i) \).
- \( X_C = \bigcup_{i=1}^{N_m} X_i \).
- \( R_C(x_j, q_c) = R^P(x_j, q_i) \) where \( x_j \in X_i \subseteq X_C \).

The global MITL formula \( \phi_{\text{global}} \) is now translated into the TBA \( \mathcal{A} \). Finally, this product is again multiplied with the TBA \( \mathcal{A} \) generated by the global MITL Task \( \phi_{\text{global}} \). This is done according to the following definition:

**Definition 5.2.2.** The product between a \( \mathcal{A}_C = (Q_C, Q_G^{\text{init}}, \rightsquigarrow_C, F_C, AP_C, L_C, I_C, X_C, R_C, d_C) \) and a TBA \( \mathcal{A} = (S, S_0, X, I, E, F, AP, L) \) is defined as \( \mathcal{A}_G = \mathcal{A} \otimes \mathcal{A}_C = (Q_G, Q_G^{\text{init}}, \rightsquigarrow_G, F_G, AP_G, L_G, I_G, X_G, R_G, d_G) \)

- \( q_G = (s, q_C) \in Q_G = S \times Q_C \).
- \( Q_G^{\text{init}} = S_0 \times Q_C^{\text{init}} \).
- \( \rightsquigarrow_G \) is given by
  - \( (s, q_C) \rightsquigarrow_G (s', q'_C) \) with \( s \neq s' \) and \( q_C = q'_C \). This can only occur if a time constraint has been violated. The weight for this transition is set to zero \( d_G(q_G, q'_G) = 0 \).
  - \( (s, q_C) \rightsquigarrow_G (s', q'_C) \) with \( s = s' \) and \( q_C \neq q'_C \) iff
    * \( q_C \rightsquigarrow_C q'_C \)
    * \( L_C(q_C) \in L(s) \)
    * \( L_C(q_C) \in L(s) \)
    with the weight \( d_G(q_G, q'_G) = d_C(q_C, q'_C) \).
  - \( (s, q_C) \rightsquigarrow_G (s', q'_C) \) with \( s \neq s' \) and \( q_C \neq q'_C \) iff
    * \( q_C \rightsquigarrow_C q'_C \)
    * \( (s, s') \in E \)
    * \( L_C(q_C) \in L(s) \)
    * \( L_C(q'_C) \in L(s') \)
    * \( L_C(q'_C) \notin L(s) \)
    with the weight \( d_G(q_G, q'_G) = d_C(q_C, q'_C) \).
  - \( (s, q_C) \rightsquigarrow_G (s', q'_C) \) with \( s = s' \) and \( q_C = q'_C \) are forbidden.

- \( F_G = F_C \times F \).
- \( AP_G = AP_C \).
• $LG(q_G) = LC(q_C)$.
• $IP(q_G) = I(s) \cup I(q_C)$.
• $X_G = X_C \cup X$.

• $R_G(x_i, q_G) = \begin{cases} R(x_i, s) & \text{if } x_i \in X \\ R_C(x_i, q_C) & \text{otherwise.} \end{cases}$

Note that $A_G$ considers every agent as well as every MITL formula. Further, the global product $A_G$ has the same structure as the Büchi WTS defined in Definition 3.15. Thus, the same graph search algorithm can be applied. However, the graph search is slightly modified in order to further improve the quality of the solution which is described in the following.

Considering a sequence where one agent is supposed to stay two times in a row in the same environment state, the underlying idea is that the agent should move towards the environment state, which is coming up next for this agent, rather than standing still. For the algorithm this means that, whenever the new weight for a node is calculated, the weight for each agent depends on the time it already stayed in $P_l$. Let the time $T_a$ denote for how long the agent has been in $P_l$ without changing the environment. Suppose that $P_l$ was entered via the point $v_i \in P_l$. The weight $d_{new}$ for this agent doing a transition to $v_j \in P_k$ is then

$$d_{new}(v_i, v_j) = \begin{cases} d(v_i, v_j) - T_a & \text{if } d(v_i, v_j) - T_a \geq T_{min} \\ T_{min} & \text{otherwise}. \end{cases}$$

where

$$T_{min} = T - T_{save}$$

which is obtained when solving (34). This holds for the cases where $q_i \neq q_i'$. This will of course increase the computational effort but at the same time the solution is closer to the time-optimal solution since it is obvious that for a time-optimal solution the agent should only wait in some rare cases. Similar to the single agent case, the modified graph search returns a sequence of states $q_G \in Q_G$ together with time constraints. From that a

$$run_i = (P^\Omega(0), 0, \infty)(P^\Omega(1), \tau^{low}(1, T), \tau^{up}(1, T)) \ldots (P^\Omega(N), \tau^{low}(N, T), \tau^{up}(N, T))$$

can be build for each agent.

5.2.2 Optimization Problem

Now that for each agent a $run_i$ is obtained, the optimization problem can be built up again. As already mentioned in the beginning of the section, this optimization problem is solved in a centralized way. In the following it is described how the optimization problem is build and afterwards it is stated.

Each $run_i$ contains a different sequence of states. However, the time constraints $\tau^{low}$ and $\tau^{up}$ are the same for all agents. This is important in order to keep the agents synchronized. Thus, each agent has the same constraints as stated in (35) without constraints involving $\tau^{low}$ and $\tau^{up}$. All these constraints are then stacked together. Finally, the constraints involving $\tau^{low}$ and $\tau^{up}$ are added. This has to be done only once since they are the same for all agents. In the last step, a user-defined cost function is stated thus completing the optimization problem which is described generically in the following.
\[
J = \min_{u_i} \int_0^{T_N} f_0([x_1 \ldots x_{N_m}]^T, [u_1 \ldots u_{N_m}]^T, t)dt + F_0([x_1(T_N) \ldots x_{N_m}(T_N)]^T, T_N) \tag{37a}
\]

s.t.
\[
\dot{x}_i(t) = f_i(x_i(t), u_i(t)) \tag{37b}
\]
\[
x_i(0) = x_i^0 \tag{37c}
\]
\[
u_i \in U_{\text{new}}^i \tag{37d}
\]
\[
x_i(t) \in \mathbb{P}_{\Omega}^i(0) \text{ for } t \in [0, T_0^{\text{save}}) \tag{37e}
\]
\[
x_i(t) \in (\mathbb{P}_{\Omega}^i(0) \cup \mathbb{P}_{\Omega}^i(1)) \setminus \mathbb{P}_{\Omega}^i(1) \text{ for } t \in [T_0^{\text{save}}, T_1) \tag{37f}
\]
\[
x_i(t) \in (\mathbb{P}_{\Omega}^i(0) \cup \mathbb{P}_{\Omega}^i(1)) \setminus \mathbb{P}_{\Omega}^i(1) \text{ for } t \in [T_1, T_1^{\text{trans}}) \tag{37g}
\]
\[
\vdots
\]
\[
x_i(t) \in \mathbb{P}_{\Omega}^i(1) \text{ for } t \in [T_1^{\text{trans}}, T_N^{\text{trans}}) \tag{37h}
\]
\[
x_i(t) \in (\mathbb{P}_{\Omega}^i(N-1) \cup \mathbb{P}_{\Omega}^i(N)) \setminus \mathbb{P}_{\Omega}^i(N) \text{ for } t \in [T_{N-1}^{\text{save}}, T_N) \tag{37i}
\]
\[
x_i(t) \in (\mathbb{P}_{\Omega}^i(N-1) \cup \mathbb{P}_{\Omega}^i(N-1)) \setminus \mathbb{P}_{\Omega}^i(N) \text{ for } t \in [T_N, T_N^{\text{trans}}) \tag{37j}
\]
\[
\chi_i(T_N^{\text{trans}}) \in \mathbb{P}_{\Omega}^i(N) \tag{37k}
\]
\[
\tau_{\text{low}}^i(1, T) \leq T_0^{\text{save}} \tag{37l}
\]
\[
T_1^{\text{trans}} \leq \tau_{\text{up}}^i(1, T) \tag{37m}
\]
\[
\vdots
\]
\[
\tau_{\text{low}}^i(N, T) \leq T_{N-1}^{\text{save}} \tag{37n}
\]
\[
T_N^{\text{trans}} \leq \tau_{\text{up}}^i(N, T) \tag{37o}
\]

where the index \(i\) goes from 0 to \(N_m\)

### 6 Case Study

In this section, the presented method is compared to state of the art methods which solve the same problem. The methods for case 1 and case 3 are described in [3] and [4]. These methods make use of a controller which generates a WTS and builds, similar to the presented method, a TBA and then a global Büchi WTS. Finally, the controller is applied to the real system.

The other method, which is also compared to the approach developed in this thesis, is described in [22]. The method translates MITL formulas into non-differentiable optimization problems, which are then approximated by smooth functions. The comparison is done in the second and fourth case study.

#### 6.1 Single Agent Case

The problem for this single agent case study is taken from [3]. The procedure is described in greater detail to improve understanding.

##### 6.1.1 Problem

The dynamic of the single agent is
\[
\dot{x} = \begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u \tag{38}
\]
The initial state is given by

\[ x_0 = \begin{bmatrix} 3.75 \\ 2.75 \end{bmatrix} \]

and the input constrains are

\[ |u|_\infty \leq 20. \]

Further, the MITL formula is given as

\[ \phi = \Box_{[0, 0.06]} r_2 \land (r_2 \Rightarrow \Box_{[0, 0.3]} r_0) \]

and the environment is described in Figure 10.

![Figure 10: The figure illustrates the environment with its environment states. Solid lines represent walls and do not allow any transitions while dashed lines allow crossing. The points \( v_i \) are the points for the WTS.](image)

### 6.1.2 Solution

The steps described in Section 3 are now executed one by one. Therefore, the MITL formula is translated into a TBA illustrated in Figure [11]. Formally, the TBA \( A \) is according to Definition 3.1.4 given as

\[ S = \{ s_1, s_2, s_3, s_4 \} \]

\[ S_0 = s_1. \]

Further, the TBA has two clocks

\[ X = \{ x_1, x_2 \} \]

and only two guards with an upper limit different from infinity.

| \( I(s_1) \) | \( 0 \leq x_1 \leq 0.06 \) |
| \( I(s_2) \) | \( 0 \leq x_2 \leq 0.3 \) |
Figure 11: The figure illustrates the TBA generated from the MITL formula given in the Case Study 1

The set of transitions $E$ is given by

$$E = \{(s_1, s_2), (s_2, s_3), (s_1, s_3), (s_2, s_4)\}. \tag{39}$$

For simplicity the transition from a location to itself is neglected here. The accepting location is here

$$F = s_4$$

Since it is of interest if the agent is in a certain room, the set of atomic propositions consists of the rooms and the corridor

$$AP = \{r_1, r_2, r_3, r_4, r_5, r_6, c\}. \tag{40}$$

The labeling function is

$$L(s_1) = AP \setminus \{r_2\} = \{r_1, r_3, r_4, r_5, r_6, c\}$$
$$L(s_2) = AP \setminus \{r_6\} = \{r_1, r_2, r_3, r_4, r_5, c\}$$
$$L(s_3) = AP = \{r_1, r_2, r_3, r_4, r_5, r_6, c\}$$
$$L(s_4) = AP = \{r_1, r_2, r_3, r_4, r_5, r_6, c\}.$$ 

To complete the TBA, the clocks are only reset in the following locations

$$R(x_1, s_1) = \top$$
$$R(x_2, s_1) = \top$$
$$R(x_2, s_2) = \top.$$

At the other locations no resets take place.

As already stated at the beginning of the section, the method from [3] uses a controller. To be comparable, the solution should be able to deal with disturbances. For this particular case an $H_{\infty}$ controller $K(s)$ with state space matrices

$$A = \begin{bmatrix} -2723 & -132.9 \\ -133.9 & -2730 \end{bmatrix} \quad B = \begin{bmatrix} -9601 & -4.824 \\ -4.824 & -9601 \end{bmatrix}$$
$$C = \begin{bmatrix} -181.2 & -13.99 \\ -13.99 & -1821 \end{bmatrix} \quad D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$
was generated. Its goal is to stabilize the agent around the optimal trajectory. For the disturbance it is assumed that $\varepsilon_w = 2.9$. This yields the maximum gain from $w$ to $e$

$$\max_{i\omega, t} \| L(i\omega, t) \| = 0.0016.$$ 

Further the maximum gain from $w$ to $u$ is given by

$$\max_{i\omega, t} \| L_u(i\omega, t) \| = 1.0038.$$ 

With these values the robustness parameters are calculated as described in Section 4.1

$$2.911 \leq \varepsilon_u$$

$$0.0047 \leq \varepsilon_e$$

The minimal distance parameter $\varepsilon$ and the input reserve $\varepsilon_u$ are chosen to be

$$\varepsilon = 0.05$$

$$\varepsilon_u = 3.$$ 

Note that both parameters are bigger than they actually should be. $\varepsilon$ is even 10 times bigger than $\varepsilon_e$. For this, there are two reasons. First, as stated in Section 3.2.2, the optimization method which was chosen is not that precise. This means that, by simply applying the input to the open loop plant the calculated trajectory isn’t exactly followed. The effect is small but it can be seen in Figure 15. The second reason is that a bigger $\varepsilon$ as well as 'less' available input for the optimization causes the WTS to increase their values. However, it can be seen that even with such conservative assumptions a good trajectory is still obtained.

All in all this leads up to new constraints on the input, which is

$$|u|_\infty \leq 17.$$ 

These are now the new bounds which have to be taken into account.

The WTS is calculated by solving the optimization problem described in (34), where the points $v_i$ are chosen in an optimal fashion going from $x_0$ to one of the rooms $P^i$. The points are displayed in Figure 10. For completeness they are also stated here explicitly


The set of inputs is generated by the optimization, where $\sigma_{ij} = u^*$ is the optimal input generated when calculating the weight for $d(v_i, v_j)$. Thus, the set of inputs $\Sigma$ simply stores the optimal inputs $u^*$. Due to the structure of the problem which includes walls, the set of transitions is

$$\rightarrow = \{ (v_1, v_2), (v_1, v_3), (v_1, v_4), (v_1, v_5), (v_1, v_6), (v_1, v_7), (v_2, v_1), (v_3, v_1), (v_4, v_1), (v_5, v_1), (v_6, v_1), (v_7, v_1) \}.$$
The atomic propositions are the same as for the TBA and therefore the set of atomic propositions of the WTS is the same as for the TBA given in (40). The observation map \( L \) is given in the following.

\[
L(P(v_1)) = c \\
L(P(v_2)) = r_1 \\
L(P(v_3)) = r_2 \\
L(P(v_4)) = r_3 \\
L(P(v_5)) = r_4 \\
L(P(v_6)) = r_5 \\
L(P(v_7)) = r_6
\]

As mentioned earlier, solving the optimization problem (34) yields the following matrix for the weights

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.0601 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.0278 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.0586 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.0565 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.0380 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.1137 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

where each row denotes the starting point from \( v_1 \) up to \( v_7 \) and each column represents the target state again from \( v_1 \) to \( v_7 \). A zero indicates that there is no transition due to the convention of the MATLAB syntax. A transition, e.g. from \( v_1 \) to \( v_4 \) takes \( d(v_1, v_4) = 0.0152 \) time units. Note that the diagonal is also zero. This is caused by the fact that a transition from a state to itself is redundant since it will not advance the process in any way and therefore such transitions are not of interest.

Continuing with building the Graph using the TBA and WTS and performing the graph search gives the

\[
\text{run} = (\mathbb{P}^1, \begin{bmatrix} 0 & 0 \end{bmatrix}, \begin{bmatrix} \infty \end{bmatrix}, \begin{bmatrix} \infty \end{bmatrix}, \begin{bmatrix} 0.06 \end{bmatrix}, \begin{bmatrix} 0 \end{bmatrix}, \begin{bmatrix} \infty \end{bmatrix}, \begin{bmatrix} 0.3 + T_1 \end{bmatrix}, \begin{bmatrix} 0 \end{bmatrix}, \begin{bmatrix} \infty \end{bmatrix}, \begin{bmatrix} 0.3 + T_1 \end{bmatrix}).
\]

From this the final optimization problem, (35) is solved using the total time as the cost function.

### 6.1.3 Results

The time obtained by the Büchi WTS is 0.1059 time units which is twice as fast as the value given in [3] which was 0.2365. Note that, as it is right now, the problem is actually solved since the input stored in the WTS can simply be used to generate a trajectory. However, it is still desirable to improve the solution. In this case, the question arises how fast the task can be solved. Therefore, as already mentioned the time optimal trajectory is calculated. At this point, it should be remarked that the value 0.1059 is only an upper bound for the optimal time. Solving the optimization problem (35) yields that the problem is solvable within 0.1052 time units. For this particular case the method presented here was able to predict a solution more than twice as fast. The plot of the optimal trajectory is illustrated in Figure 12.

The simulation was also tested with disturbances. As already calculated in the previous section, the influence of the disturbance is around 0.0049 and therefore only parts of the plot are shown. In Figure 13 it can be clearly seen that all disturbed trajectories enter the second room.

The same plot is displayed in Figure 14 for room 6. Unlike the plot for room 2, where all points of the disturbed trajectories at time \( T^\text{trans}_1 \) have been within a radius of \( \varepsilon_e \) around the optimal trajectory, this is not the case in room 6. The reason for this, inaccuracy in the optimization step, has already
Figure 12: Displayed is the environment for the first case study. It can be seen that the optimal trajectory fulfills the MITL formula. The blue stars mark where the robust environment state of room 2 and room 6 are entered.

been mentioned. Increasing the number of discretization points for each interval $M$ results in a better trajectory, as it can be seen in Figure 15.
Figure 13: The figure displays a part of the room 2. The blue solid circle has the distance $\varepsilon$ to the blue star which is the point where the optimal green trajectory enters the robust environment of room 2. The blue dashed line has the radius of $\varepsilon_e$ and its center is also the blue star. The black curves are disturbed trajectories. The red stars indicate the latest time point when the black curves are supposed to cross the grey dashed line which represents the edge of room 2.
Figure 14: The figure displays a part of the room 6. The blue solid circle has the distance $\varepsilon$ to the blue star where is the point when the optimal green trajectory enters the robust environment of room 6. The blue dashed line has the radius of $\varepsilon$, and its center is also the blue star. The black curves are disturbed trajectories. The red stars indicate the end of the trajectory.
Figure 15: The dashed blue circle has the radius \( \varepsilon_e \) and is drawn around the endpoint of the optimal trajectory. The optimal trajectory in the left figure was generated with a total of 1800 discretization points and therefore not all points end up with in the blue dashed circle due to inaccuracy of the optimization. In the right figure, 12000 discretization points have been used and the effect of a finer discretization results in the endpoints of the disturbed trajectories, marked with the red star, being closer to the endpoint of the optimal trajectory.
6.1.4 Theoretical Property

Although the solution already outperforms the solution given in [3], in this section the condition of Theorem 1 is checked for this particular case. It turns out that the condition is not fulfilled. There is no point in room 4 and in room 6 from which a transition to the facet is slower than the transition from \( v_i \) to \( v_1 \) \( i \in \{4, 6\} \). Instead of moving the points, as it was done in Example 3.2, this time two new points are added

\[
v_8 = \begin{bmatrix} 2.4244 \\ 2.05 \end{bmatrix}, \quad v_9 = \begin{bmatrix} 4.5216 \\ 2.05 \end{bmatrix}.
\]

The new points are displayed in Figure 16 in red. Notice that transitions to the red points are only allowed for the transitions from \( v_5 \) to \( v_8 \) and from \( v_7 \) to \( v_9 \). From the red points transitions are allowed to any other point except points within \( P_1 \). In this fashion the new WTS also fulfills Theorem 1. The obtained solution doesn’t change for the new system since transitions from \( P_5 \) to \( P_1 \) and \( P_7 \) to \( P_1 \) are not part of the solution.

6.2 Nonlinear Single Agent

For the next case study, a car-like model is taken from [22]. Rather than comparing both methods here, the purpose of this case study is to show that the presented method also works for nonlinear models. Further, different cost functions are introduced to see the different effect on the resulting trajectory. Since [22] only calculates an optimal trajectory a controller will not be calculated for this case.

6.2.1 Problem

The nonlinear model is the unicycle,

\[
\begin{align*}
\dot{x} &= v \cos(\theta) \\
\dot{y} &= v \sin(\theta) \\
\dot{\theta} &= u
\end{align*}
\]
where \( v \) and \( u \) are the inputs and \( x, y \) and \( \theta \) are the states. The initial condition is given by
\[
\mathbf{x}_0 = [-2, -2, 0]^T
\]
and the input constraints are
\[
-0.5 \leq v \leq 0.5 \\
-0.5 \leq u \leq 0.5.
\]

Figure 17: The figures show how the environment was given and how it can be divided into smaller convex areas.

The environment is illustrated in Figure 17 together with the points for the WTS. The angle \( \theta \) for each point \( v_i \) is set to 0. The MITL formula is given by
\[
\phi = \Box_{[0,20]} \neg (x \in Unsafe) \land \Diamond_{[0,20]} (x \in Terminal).
\]
However, for this case study the MITL formula is changed slightly since the always operator \( \Box \) makes time optimal solutions senseless, since the task always finishes after exact 20 time units, even if the terminal region is reached earlier. Therefore, the new formula should become true once the terminal region is reached which is why the until operator is used. The new formula is given by
\[
\phi = (\neg (x \in Unsafe)) U_{[0,20]} (x \in Terminal).
\]
Although there are no disturbances in this case, a minimal distance to the edges of
\[
\varepsilon = 0.1
\]
should be achieved by the solution.

### 6.2.2 Results

Since the lower bound in the interval of MITL formula is 0, a solution can also be obtained via the WTS. This solution is displayed in Figure 18. Generating the WTS is not trivial since nonlinear optimization comes with multiple difficulties. However, this solution gives an upper bound on the time optimal trajectory by 17.43 time units.
Figure 18: Solution generated by the WTS. The blue dots are the points \( \mathbf{v} \) of the WTS. The green area is the terminal region and the red area is the Unsafe region. Areas enclosed by the dashed lines are the robust environment states.

In the following, the solution obtained by the WTS is used as an initial guess to obtain optimal solutions regarding different cost functions. The solutions can be seen in Figure 19 where the time optimal solution minimizes the cost function

\[
J = \min_u \int_0^{T_N} dt,
\]

the robust optimal solution minimizes the cost function

\[
J = \min_u \int_0^{T_1} (x(t) + 1.75)^2 dt + \int_{T_1}^{T_2} (y(t) - 1.75)^2 dt
\]

which corresponds to staying as far as possible away from the unsafe area, as well as the edge of the environment. Finally, the energy optimal trajectory minimizes the cost function

\[
J = \min_u \int_0^{T_N} \mathbf{u}^T(t) \mathbf{u}(t) dt.
\]

The optimal time solution gives a time of 13.24 which is 4.19 seconds faster than the solution given by the WTS.
6.3 Multi Agent Case

This problem is taken from [4]. The method used in [4] is the same as in [3].

6.3.1 Problem

The dynamic of the agents is given by

\[
\begin{align*}
\dot{x}_1 &= \begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix} x_1 + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u_1 \\
\dot{x}_2 &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x_2 + \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} u_2.
\end{align*}
\]

The input constraints are given by

\[||u_i||_\infty \leq 20 \text{ where } i \in \{1, 2\}.\]

As for the single agent case, a controller is needed. In order to ensure that the controller always keeps the agents close to their optimal trajectory, input has to be reserved for the controller. Thus, the input constraints for the optimization are given by

\[||u_i||_\infty \leq 17 \text{ where } i \in \{1, 2\}.\]
The minimum distance is chosen to be \( \varepsilon = 0.1 \). The points of the WTS have been obtained by Algorithm 1 and are illustrated together with the environment in Figure 20. Each agent has a MITL task which is in this case

\[
\phi_i = \Diamond_{[0,0.1]}(\text{Room2}) \land (\text{Room2} \Rightarrow \Diamond_{[0,0.3]}(\text{Room6})) \quad \text{with} \quad i \in \{1, 2\}
\]

for both agents. Furthermore, the global task which both agents have to fulfill together is given by

\[
\phi_{\text{global}} = \Diamond_{[0,1]}((x_1 \in \text{Room1}) \land (x_2 \in \text{Room2})).
\]

Figure 20: The figure illustrates the environment for the multi agent case described in Section 6.3. The blue points belong to the WTS of the first agent, while the red points belong to the WTS of the second agent.

6.3.2 Results

In the solution presented in [4] both agents first go to room 2 then to room 6 and after that they fulfill the global task by going to room 1 and room 2. The time it takes these agent to fulfill the task is 0.7404 time units.

The obtained solution here differs from [4]. The obtained sequence first fulfills \( \phi_{\text{global}} \) and after that the tasks for each agent. The predicted time from the WTS is 0.1809 time units. The final time of the optimal trajectory is 0.1656 time units and the trajectories for both agents are displayed in Figure 21. The predicted and obtained total time are both significantly faster than the solution given in [4].

6.3.3 Lower Interval Constraint

Since the points of the weighted transition systems have been chosen by the Algorithm 1 in such a fashion that it allows for lower constraints different from 0 in the interval of the MITL formulas, the following change for the global MITL task is considered

\[
\phi_{\text{global}} = \Diamond_{[0,0.25,1]}((x_1 \in \text{Room1}) \land (x_2 \in \text{Room2})).
\]  

(41)

The obtained sequence from the graph search yields the same sequence as in [4]. The WTS predicts a total time of 0.4614. The optimal solution is displayed in Figure 22. The final time is then 0.3729 time units.
Figure 21: The figure shows the solution for the given task in case study three. The blue trajectory belongs to agent 1, and the green trajectory to agent 2. The black stars indicate the initial states, the magenta stars the end states of the agents.
Figure 22: The figure shows the solution for the given task in the case study three with the adapted MITL formula given in [11]. The blue trajectory belongs to agent 1 and the green trajectory to agent 2. The black stars indicate the initial states. The magenta stars the end states of the agents.
6.4 Multiple Quadcopter

This case is taken from [22]. The models suggested in this case, are quadcopters which can be controlled using model predictive control.

6.4.1 Problem

The dynamics of the quadcopters are given by

\[
\dot{x}_i = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} x_i + \begin{bmatrix} 0.98 & 0 & 0 \\ 0 & -0.98 & 0 \\ 0 & 0 & 0.2 \\ 9.8 & 0 & 0 \\ 0 & -9.8 & 0 \\ 0 & 0 & 2 \end{bmatrix} u_i, \text{ where } i \in \{1, 2\}. \tag{42}
\]
The state space represents the x-y-z coordinates and its derivatives.

\[
\mathbf{x}_i = \begin{bmatrix}
x \\
y \\
z \\
v_x \\
v_y \\
v_z \\
\end{bmatrix}
\]

The environment is displayed in Figure 23 and has been distributed into convex parts using only cubes to simplify the implementation. The MITL formula for each agent \(a_i\) is

\[
\phi_i = \Diamond [0,4] (a_i \in \text{Terminal}) \land 
\Box [0,4] (a_i \in \text{Zone1} \Rightarrow z_i \in [1,5]) \land 
\Box [0,4] (a_i \in \text{Zone2} \Rightarrow z_i \in [0,3]) \\
\land \Box [0,4] \neg (a_i \in \text{Unsafe})
\]

where \(z_i\) is the third entry of the state vector \(x_i\) which corresponds to the height. Again, since the always term makes no sense for a time optimal optimization, the MITL formula is slightly changed using the until operator. This yields

\[
\phi_i = (\Box [0,4] (a_i \in \text{Zone1} \Rightarrow z_i \in [1,5]) \land 
\Box [0,4] (a_i \in \text{Zone2} \Rightarrow z_i \in [0,3]) \land 
\Box [0,4] \neg (a_i \in \text{Unsafe})) \\
\text{U}_{[0,4]} (a_i \in \text{Terminal}).
\]

The global task is given by

\[
\phi_{\text{global}} = \Box [0,4] (||a_1 - a_2||_2 \geq d_{\text{min}})
\]

where \(d_{\text{min}} = 0.2\) and the term \(||a_1 - a_2||_2\) corresponds to the distance between both agents in the 3 dimensional x, y and z space. Since the distance between both agents can’t be captured in the global Büchi WTS it is assumed that the global task is fulfilled. Hence

\[
\phi_{\text{global}} = \top.
\]

However, this task is not simply ignored. Instead of considering it during the graph search, the minimal distance is considered during the final optimization as a constraint.

The weighted transition systems have been generated by setting a point in the center of each cube displayed on the right side in Figure 23. The derivatives have been set to 0.

6.4.2 Results

The final trajectories are illustrated in Figure 24. From the graph search a solution could not be obtained at first. By softening the upper time bounds, the graph search obtained a solution which predicted a time of 5.7625 time units. The time optimal trajectories yield a time of 1.953 time units which fulfills the MITL task. A better prediction from the graph search, namely 3.899 time units, was obtained by structuring the environment differently and the point within each environment state was obtained by optimizing just to the surface of the environment starting out form the initial points.

In [22] it is stated that the authors believe that this task represents the type of problems which can’t be solved by (MILP). However, it turns out that the approach in [22] also needs 30 minutes until a solution is obtained. Further, the solution has 20 time discretization points. Compared to that, the solution approach in the present work needs 5 minutes to create the WTS for each agent, 5 seconds for the graph search and about 1 minute to complete the final optimization. This gives a total time of 11 minutes. Furthermore, the final solution has 800 time discretization points. The calculations were preformed with an Intel Core i5-3317U CPU @ 1.70GHz. The optimization was done on Julia 0.5.1 and for the graph search MATLAB 2016a was used.
Remark 6.4.1. The comparison of the computational time should be taken with a grain of salt. \[22\] uses better hardware (quad-core Intel i5 3.2GHz processor). However when it comes to optimization, julia usually performs better than MATLAB, which was used in \[22\]. Also the number of discretization points "plays a big role". By using 21 discretization points for the optimization for the WTS and a total of 56 points for the final optimization, the whole problem is solved within 2 minutes. The construction of the WTS takes about 50 seconds each, where for each WTS 96 optimization problems had to be solved. The final optimization only takes 2 seconds. The time for the graph search remains the same, namely 5 seconds.

![Optimal Trajectories](image)

Figure 24: The figure shows the optimal trajectories of the agents given in case study 4.

7 Discussion

In the following, different aspects of the presented method are discussed.

7.1 Case Study

The case studies have shown that the presented approach can not only compete with state of the art methods but also outperforms them in terms of quality of the solution and in some cases also in the computational time. However, the latter is usually not given since the optimization is computationally more expensive, which is probably the biggest drawback of the method.

The last case study described in Section 6.4 is probably the most interesting case, since the given task is very difficult and challenging. The presented approach here couldn’t find a solution via the graph search at first, thus measures had to be taken to tackle this problem. Other methods which could possibly help in this case are discussed in Section 9.2.

In the following some additional remarks about \[22\] are made. One goal of the paper was to find a method which solves the described problem in a fast way in order to apply MPC algorithms in a future
work. [22] states that the eventually operator \( \hat{\psi}_{[a, \infty]} \) causes trouble in the optimization, thus making the problem non-convex. Other tasks presented in [22] were solved within a fraction of a second. Compared to that the computational time of the solution approach presented in this paper does not increase because of the \( \hat{\psi}_{[a, \infty]} \) operator. As a matter of fact the computational time depends mainly on the following aspects.

- Number of environment states \( P^i \).
- Number of neighbours of each environment state \( P^i \).
- Whether lower bounds of the MITL formula have to be considered.
- Number of points for the discretization.
- The length of the MITL formula.

The first 3 aspects refer to the size of the WTS. The fourth aspect mainly determines the time of the optimization and the last aspect determines the size of the TBA. None of these aspects refer to the actual content of the MITL which is an advantage compared to [22]. It is conjectured that this fact makes the method more robust in obtaining a solution. A possible MPC scheme using the here presented approach is sketched in Section 9.4.

### 7.2 Building the WTS for the Nonlinear Case

Building a WTS for a nonlinear system turned out to be the most difficult part during the case study. The reason for that lies in the very nature of nonlinear optimization. The problems is that the solver might converge to an infeasible solution. Compared to the linear case, it takes a long time until the solver finds a feasible or infeasible solution. A good initial guess, as mentioned at the end of Section 3.2.2, is therefore very important. Finding a solution via multiple random initial guesses for the IpOpt Solver can require a lot of computational power and is hence inefficient. A good understanding of the nonlinear model from which a reasonable initial guess can be generated is more desirable, although this requires an intelligent user.

### 7.3 Selecting Points

As mentioned earlier, selecting the points for the WTS is a degree of freedom. The importance of this has been shown in the last case study of Section 6.4, where a "bad" selection of points resulted in not finding a solution via the graph search. The complexity of the WTS is determined by the amount of points and the number of connections each point has.

The presented method which selects the points automatically is motivated by the structure of the graph search algorithm. The neighbours of the environment state of the initial point are reached in optimal time. The same can be said about the newly generated points. Afterwards, it depends on the structure of the problem but in general the first two transitions are optimal.

When it comes to systems which loosely speaking "have a higher relative degree than 1", meaning systems which have also velocities as states or higher derivatives, such as a double integrator or the system in equation (42), the question arises how to choose the velocities. Setting them simply to zero seems not to be sufficient for every given task. This is a problem which can be solved by the approach described later in Section 9.2.

### 7.4 Modified Search Algorithm

The fact that the graph search doesn’t succeed every time, even though there exists a solution, has been demonstrated in Section 6.4. This is caused by the problem itself.
However, the modified search algorithm can also fail even though there exist a way through the graph. These solutions are neglected, not because of time constraints, but mainly because of the structure of the graph. This might happen only in an artificial case, since it was not encountered in any way during the case study. Therefore this might be more of an academic issue, nevertheless an example is given in the following.

![Graph](image)

Figure 25: To see is a graph where the modified Dijkstra algorithm fails. The solution is Start-B-C-Goal

**Example 7.1.** Consider the graph given in Figure 25. The only solution is the sequence Start-B-C-Goal. However, this sequence is not obtained. This is caused by the clock $x_1$. In the following it is described how the algorithm performs the search and how it fails.

Starting from the Start node the next nodes are A and B which both have the value of $\infty$ due to initialization. Therefore,

$$\text{distance}(A) = 1$$
$$\text{distance}(B) = 100.$$  

Since A is smaller than B, A is the next node which is explored. The next node C again has a value of $\infty$ due to the initialization. Thus,

$$\text{distance}(C) = 11.$$  

Since the distance to C is smaller than the distance to B which is 100, C is the next node which is explored again. The next node from C is the Goal state. If it wouldn’t be for scenario S3 which is the case when the clocks violate upper constraints, the search would end here. However, the clock of $x_1$ and $x_2$ have to be checked. Both clocks have not been reset so far and therefore the total time corresponds to both clocks. The total distance from Start over A and C to Goal is 21. It is easy to see that the clock constraint for $x_1 = 21 \leq 20$ has been violated, thus

$$\text{distance}(\text{Goal}) = \infty.$$  

The algorithm continues. Since the distance to B, which is 100, is smaller than to the Goal, the next node to explore is B. B has only C as a successor. The distance over B to C is 101 and greater than 11 which is the distance over A to C. Thus B will not be the parent of C. The algorithm ends with the shortest distance of $\infty$.

However the sequence Start-B-C-Goal satisfies all constraints. The total distance for this sequence is 111. $x_2$ has not been reset and therefore equals the total time. Its constraint is satisfied in the goal state since

$$x_2 = 111 \leq 200.$$
The clock $x_1$ gets reset in B. Thus, $x_1 = 11 \leq 20$.

The Example 7.1 illustrates how the reset of a clock can cause trouble. It is tempting to simply set the distance of the nodes equal to the value of clock $x_1$, however counter examples with multiple clocks can be constructed.

7.5 Disturbances

So far, the issue illustrated in Figure 26 has not been discussed. Assume that a transition from A to B has to be done. The blue line represents the optimal trajectory and the circles have the radii of $\varepsilon_e$. Suppose the green point has already crossed the border between A and B. The problem is now that the green point can be pushed back by the disturbance such that the agent performs the sequence A-B-A instead of A-B. As a matter of fact, it can actually have finitely many changes between A and B. To avoid this problem in this case the following property has to be fulfilled:

\[
\dot{x} \geq 0.
\]

For the more general case this would mean that

\[
\frac{d}{dt} p^{\Omega(k)}_i(x(t)) \leq 0 \text{ for } t \in [T^{\text{save}}_{k-1}, T^{\text{trans}}_k]
\]

where $p^{\Omega(k)}_i(x(t)) = a_i x(t) + b_i$ represents the line separating the states $P^{\Omega(k-1)}$ and $P^{\Omega(k)}$. During this thesis this property was assumed to be fulfilled and hasn’t been checked.

Figure 26: The figure illustrates the problem when transitioning from A to B under disturbances. The blue line is the optimal trajectory. The solid and dashed circle have the radius $\varepsilon_e$. The dashed circle is the circle at a later point in time. During a transition the disturbed trajectory can move from the green point the the red point, thus it is not guaranteed an A-B-A sequence will be avoided.

8 Conclusion

Given a dynamical system in an environment together with a MITL task, a novel approach has been presented which determines an input for the dynamical system such that the MITL task is fulfilled. The approach uses time optimization to generate a weighted transition system. Furthermore, a verifiable theoretical property has been given for the WTS which, when fulfilled, guarantees to give better time estimations than WTS generated from controllers as it is done according to already existing methods. The MITL formula is considered by the construction of a TBA. A sequence of environment states is obtained by a graph search on the product of WTS and TBA. The final trajectory is generated according
to a user-defined cost function. The approach has been extended to cope with multi agents as well as disturbances. Further, the approach can handle nonlinear systems as well as, under some assumptions, lower bounds unequal to 0 in the intervals of the MITL formula. This has been demonstrated during the case studies in which the approach has been compared to state of the art methods. It was also possible to find better solutions in either quality or computational time. The flaws and issues of this approach have been discussed in Section 7.

9 Future Work

Despite fixing the small issues described in Section 7, main improvements can be done in generating the WTS, handling failures of the graph search algorithm, finding global optimal solutions and developing a MPC scheme handling MITL formulas.

9.1 WTS Improvement

Theorem 4 already provides a verifiable condition if the presented WTS outperforms the "control to facet WTS". However, rather than checking a condition, it is more desirable to have a constructive proof which leads to the selection of the points \( v_i \) for a given dynamic.

Further, for systems with integrator chains, the question arises how to select the velocities or the derivatives, as it was already discussed in Section 7.3.

9.2 Refinement

Throughout the thesis it was always assumed that the graph search never fails. In the following, measures are described which can be taken if this is not the case. One was already presented in Example 3.2 where points \( v \) of the WTS were moved in order to get smaller transition times. Instead of moving points around also new points can be added. This requires a very good understanding of the system which is usually a drawback. In Section 6.4 it has been shown that softening the MITL constraint can also lead to the desired result. However, it is desirable to rely on a method which does not require a good knowledge of the system or luck. A possible solution and start for future work is presented in the following.

First, it should be pointed out that, for the presented abstraction method of the WTS, most of the already existing refinement methods as described in [8], [19] and [11] are not fully applicable here. These methods mainly rely on further partitioning of the environment states \( P \). This could result in worse WTS, where the transitions between environment states may become smaller since the states are not that 'big' anymore, but the whole time for fulfilling the MITL task could increase as it is demonstrated in the following example.

Example 9.1. Consider the double integrator

\[
\dot{x} = \begin{bmatrix} x_2 \\ u \end{bmatrix}
\]

with \( x(0) = x_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \) and \( u \in [-1, 1] \).

Further, assume that \( P^1 : x_1 \in [0, 1) \) and \( P^2 : x_1 \in [1, 2] \). Suppose it is required to perform a transition form \( P^1 \) to \( P^2 \) where \( v_1 = x_0 \) and \( v_2 = [1, 0]^T \). The optimal weight \( d(v_1, v_2) = 2 \) and the trajectories can be seen in Figure 27. Partitioning \( P^1 \) in half with \( P^3 : x_1 \in [0, 0.5) \) and \( P^6 : x_1 \in [0.5, 1) \) and choosing the new point \( v_3 = [0.5, 0]^T \) yields the optimization times

\[
d(v_1, v_3) = \sqrt{2} \\
d(v_3, v_2) = \sqrt{2}.
\]

Thus the total time going from \( P^1 \) to \( P^2 \) increases since \( 2 < \sqrt{2} + \sqrt{2} \). The trajectories are displayed in Figure 27.

63
Figure 27: Displayed are the trajectories of the simple double integrator system given in Example 9.1. On the left side the trajectories for going from $v_1$ directly to $v_2$ are displayed, whereas on the right side the trajectories for going from $v_1$ to $v_3$ and then to $v_2$ are illustrated.
Remark 9.2.1. It is clear that \( v_3 \) was chosen poorly but this emphasize what has been previously mentioned. It is desirable to find a method which doesn’t require a good system understanding.

The refinement method proposed here is described in more detail in Algorithm 6. In the following, the thoughts behind this algorithm are sketched.

The idea is to compute all the reachable states \( F_{\text{reachable}} \) of the Büchi WTS \( \hat{A}^P \) which are the closest to the goal states \( F^P \). This means that for each state \( q_i \in F^P \) it is checked if its predecessor \( q_j \) is reachable from \( Q_{\text{init}} \). If yes, the state is added to \( F_{\text{reachable}} \). Otherwise, the predecessor of \( q_j \) is checked and so on. While doing so, loops are avoided by memorizing which states have already been checked.

As soon as \( F_{\text{reachable}} \) is built, it is known that there exists a trajectory to the state \( q_i = (s, v) \in F_{\text{reachable}} \). From the graph search a \textit{run} can be build.

The optimization problem (20) with the adaptation \( x(T_N) = v \)

is then efficiently solved because of the initial guess given by the Büchi WTS. Also, the values of \( \hat{X}^P \) when applying the solution of the optimal control problem are calculated. This will give a new Büchi WTS

\[
\hat{A}^P \leftarrow (Q, q_i, \rightsquigarrow, F^P, AP, I^P, \hat{X}^P, d^P, R^P).
\]

Going from \( Q_{\text{init}} \) to \( F^P \) via \( q_i \) becomes less conservative, due to the optimization to \( q_i \). The new Büchi WTS \( \hat{A}^P \) starts from \( q_i \). Due to the less conservative estimation, it might be possible that the new \( F_{\text{reachable}} \) for the new Büchi WTS \( \hat{A}^P \) might even share states which \( F^P \) thus solving the problem.

The procedure described here is done repeatedly. The method fails if \( Q_{\text{init}} \) is the only state in \( F_{\text{reachable}} \) and there is no reachable state \( q_j \) from \( Q_{\text{init}} \) which can be reached by optimization.

Remark 9.2.2. It is important to note that the presented refinement algorithm only works in one direction. This means that if the refinement algorithm finds a solution, it is a solution. However it is not the other way around, i.e. there is no possibility to conclude non-existence of solutions. This is caused partly by the problem described in Section 7.4.

9.3 Global Optimal Solution

So far, the global optimal solution has not been calculated. Instead, of that the final optimization was just locally optimal since it is unknown if there exists another sequence of states which yields better results. One approach in order to get closer to the global optimum is to modify Algorithm 2. Instead of taking the weight of the WTS as the distance, the optimal control problem (20) for a desired cost function could be solved. This should lead to a global optimal solution regarding to a certain cost function. However, this will probably be computationally very expensive. Thus, this approach may only be computable offline.

9.4 MPC

In the following, a starting point is given to extend the presented approach to MPC. Creating the WTS takes most of the computational time. Under the assumption that the environment won’t change, the WTS can be computed offline. Assume, for the sake of simplicity, an environment where each environment state contains one point. The following task has to be solved repeatedly.

1. Calculate the time optimal trajectory to each point in the neighbouring state of \( P(x_0) \).
2. Replace the weights in the WTS with the newly obtained optimal times.
3. Update/Calculate the new Büchi WTS with the new values and an eventually new MITL formula.
4. Perform the graph search and obtain a \textit{run}.
5 Calculate the optimal trajectory and apply the first part, as it is usually done in MPC.

6 Measure $x$, define it as the new initial point $x_0$ and go back to the first step.

Reusing already existing optimal trajectories as new initial guesses should decrease the computational time significantly. A rough time estimation using the times given in the Remark 6.4.1 suggests that one iteration is roughly done in 9 seconds: 2 seconds to compute the trajectories to the neighbours, 5 seconds for the graph search and 2 seconds for the final optimization. This rough estimate suggests that the MPC scheme can be used for slow dynamical systems.
Algorithm 6: Refinement

| input : Büchi WTS $A^P$ |
| $k \leftarrow 0$ |
| $F^k \leftarrow F^P$ |
| $F_{temp} \leftarrow F^P$ |
| $F_{reachable} \leftarrow \emptyset$ |
| while $F_{temp}$ is not empty do |
| $k \leftarrow k + 1$ |
| $F^k \leftarrow \emptyset$ |
| foreach $q_i \in F_{temp}$ do |
| foreach $q_j$ where $q_j \rightsquigarrow q_i$ do |
| if $q_j \notin \bigcup_{i=0}^{k-1} F^i \cup F_{reachable}$ then |
| add to $F^k$ |
| end |
| end |
| $\hat{A}^P \leftarrow (Q, Q_{init}, \rightsquigarrow, F^k, AP, LP, IP, \hat{X}^P, d, R^P)$ |
| while graph search on $\hat{A}^P$ is successful do |
| get the last state of the graph search $q_{last}$ |
| add $q_{last}$ to $F_{reachable}$ |
| remove $q_{last}$ from $F^k$ |
| end |
| $F_{temp} \leftarrow F^k$ |
| end |
| if $Q_{init} \in F_{reachable}$ then |
| foreach $q_i \in \bigcup_{i=0}^{k-1} F^i$ where $Q_{init} \rightsquigarrow q_i$ do |
| Try to solve the optimization problem (20) for the run obtained from $Q_{init}$ to $q_i$ |
| if Optimization successful then |
| Create a new WTS with new initial point and times obtained from the optimization |
| if $q_i \in F^P$ then |
| return Success |
| else |
| return Refinement ($\hat{A}^P$) |
| end |
| end |
| return failure |
| end |
| foreach $q_i \in F_{reachable}$ do |
| if $q_i \in F^P$ then |
| return success |
| else |
| Get the sequence from the graph search to $q_i = (s, v)$ From the sequence generate a run |
| Solve optimization problem (20) where $x(T_N) = v$ |
| $Q_{init} \leftarrow q_{last}$ |
| $\hat{X}^P \leftarrow \text{EvaluateClocks} (\text{Sequence}, R^P, q_{last}, \text{dist})$ |
| $\hat{A}^P \leftarrow (Q, Q_{init}, \rightsquigarrow, F^P, AP, LP, IP, \hat{X}^P, d, R^P)$ |
| return Refinement ($\hat{A}^P$, Sequence) |
| end |
| end |
References


