# Test Design Based on Model Mismatch for Controlled Systems

Rebecka Winqvist
Mentor: Richard M. Murray, Co-mentor: Sofie Haesaert

***Abstract*—Finding safety and functionality guarantees for sophisticated safety-critical systems such as intelligent robots, autonomous vehicles and cyber-physical infrastructures is imperative. Recently developed frameworks enable automated synthesis of controllers and allows for a reduction in design faults. These methods rely on accurate models of systems with specifications expressed in temporal logics. Albeit being verified on a model of the system, the controller must also be tested on the true system. This project investigates the possibility and efficiency of designing tests based on the model mismatch, i.e. the model's deviation from the true system, with the aim to introduce a new method for designing tests to verify the robustness of the controller. The proposed approach of adapting model-based testing tools for controlled systems shows great promise and future work would involve investigating smart model deviations and generating executable test cases for the true cyber-physical system.**

## I. Introduction

The idea of describing and verifying control systems with respect to Linear Temporal Logic (LTL) specifications originates from the domain of software and hardware verification, where such methods have proven great success [1]. The Python-based TuLiP toolbox employs similar formal methods for synthesizing controllers that come with behavioral guarantees over models of systems with respect to specifications expressed in LTL [2] [3]. The design process thus relies on accurate descriptions (models) of systems and their behavior. However, since the dynamics of most Cyber-Physical Systems (CPS) rarely is precisely known, models are often built on empirical approximations based on data obtained from the system. It is therefore critical to test and verify correct behavior of the designed controller on the true system, albeit the controller already being verified on the model.

As of today, there exists some preliminary research on testing of dynamical (stochastic) systems. For example, in [4], they present a Statistical Model Checking (SMC) method for determining the probability that a system satisfies a certain temporal logic property with a certain confidence. In [5], however, they argue that for some applications that calculated probability might not be sufficient information. Instead they propose a different approach that, in addition to SMC, also distinguishes between system designs that satisfy the property to varying degrees. Both methods utilize the notion of robustness for Metric Temporal Logic (MTL) and can be applied to both stochastic systems as well as finite state systems.

Model Based Testing (MBT) is a test generation technique proven successful in software testing. The method relies on models of the System Under Test (SUT) when deriving test cases [6]. In this project, the possibility of adapting MBT tools for designing tests for control systems is investigated. Essentially, the approach involves investigating which transitions in the abstract finite-state model of the system are more sensitive to mutations in the model and then exploiting these transitions by applying different input sequences to the controller. In particular, Linear Time Invariant (LTI) reactive systems described by the following form are considered in this project

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t) + Ed(t) \\ y(t) &= Cx(t) + Du(t) + Fd(t) \end{aligned} \tag{1}$$

where $x(t)$ is the current state of the system, $u(t)$ the current input to the system, $y(t)$ the output of the system, and $d(t)$ the exogenous disturbances.

## II. Linear Temporal Logic

Temporal logic is a formalism for specifying and verifying properties of reactive systems. LTL is a modal temporal logic suited for specifying linear time properties, with modalities referring to time. Using LTL, relations between the states of a system can be expressed in a mathematically precise notation, that is, the relative *order* of events can be specified [7]. However, LTL cannot be used to specify precise timing of those events. For example, "The ball hits the ground once it has been dropped" would be a supported statement, whereas "The ball hits the ground three seconds after it has been dropped" would not.

An LTL formula, $\varphi$, follows a specific syntax and may be constructed from atomic propositions, Boolean connectors and the two basic temporal modalities $\bigcirc$ (pronounced "next") and $\mathcal{U}$ (pronounced "until") according to the following grammar [7]

$$\varphi ::= \text{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi_1 \; \mathcal{U} \; \varphi_2 \tag{2}$$

where $a \in AP$ is an atomic proposition. $\wedge$ and $\neg$ can be used to derive all other Boolean connectives and the temporal modalities $\Diamond$ (pronounced "eventually") and $\Box$ (pronounced "always"). Figure 1 presents a few simpler LTL formulae and their respective state sequences.

The semantics of an LTL formula, $\varphi$, is defined by the following language

$$Words(\varphi) = \left\{\sigma \in (2^{AP})^{\omega} \mid \sigma \models \varphi\right\} \tag{3}$$

where $\sigma$ is referred to as a word in the $2^{AP}$ alphabet. A word $\sigma = \sigma_1, \sigma_2, \dots$ , with suffix $\sigma(t) = \sigma_t, \sigma_{t+1}, \dots$ , satisfies a

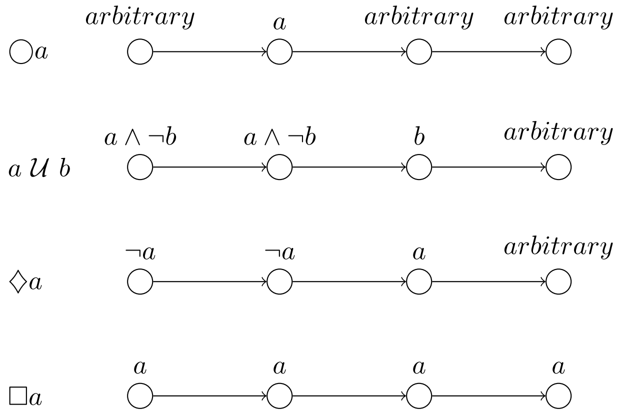

Fig. 1. Simpler LTL formulas and their respective state sequences.

specification $\varphi$ iff $\sigma(t)$ satisfies $\varphi$ for all $t$. Below follows the LTL semantics for a few formulas [8]

$$\begin{array}{ll}
\sigma \models true & \\
\sigma \models a & \text{iff } a \in \sigma_1, \text{ i.e., } \sigma \models a \\
\sigma \models \varphi_1 \wedge \varphi_2 & \text{iff } \sigma \models \varphi_1 \text{ and } \sigma \models \varphi_2 \\
\sigma \models \neg\varphi & \text{iff } \sigma \not\models \varphi \\
\sigma \models \bigcirc\varphi & \text{iff } \sigma[1\ldots] = \sigma_1, \sigma_2, \ldots \not\models \varphi \\
\sigma \models \varphi_1 \,\mathcal{U}\, \varphi_2 & \text{iff } \exists j \geq 0.\ \sigma[j\ldots] \models \varphi_2 \text{ and} \\
 & \sigma[i\ldots] \models \varphi_1, \text{ for all } 0 \leq i < j
\end{array}$$

As stated above, LTL formulas are not well suited for describing any precise timings for events. That could, however, be desirable in some applications, e.g. when specifying the dynamics of a real-time system [8]. One such subset of LTL, where the modalities have been augumented with time constraints, is the General Reactivity (1) (GR(1)) form, on which the formulas are written as follows

$$\varphi = (\varphi_e \rightarrow \varphi_s) \tag{4}$$

where $\varphi_e$ is an assumption of the environment in which the system operates, and $\varphi_s$ the specifications the system must fulfill. These are described by the LTL formulas

$$\varphi_e = \theta^e_{init} \wedge \bigwedge_{i \in I_e} \Box\psi^e_i \wedge \bigwedge_{k \in K_e} \Box\Diamond J^e_k = \varphi^e_i \wedge \varphi^e_t \wedge \varphi^e_g \tag{5}$$

and

$$\varphi_s = \theta^s_{init} \wedge \bigwedge_{i \in I_s} \Box\psi^s_i \wedge \bigwedge_{k \in K_s} \Box\Diamond J^s_k = \varphi^s_i \wedge \varphi^s_t \wedge \varphi^s_g \tag{6}$$

respectively, where $\varphi_i$ are the initial conditions, $\varphi_t$ the safety or transition statement, and $\varphi_g$ the fairness and goals statement. The safety statement specifies the desired and/or allowed actions, while the fairness and goals statement specifies the desired and possible actions for the system to make [9].

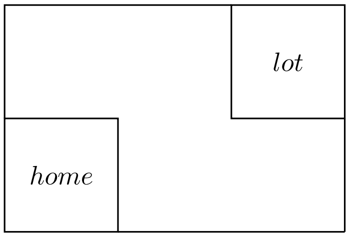

Fig. 2. The 3x2 workspace for the car in the case study. The two labels *home* and *lot* encapsulate two different areas and act as atomic propositions.

## III. Case study

The case study was chosen to be an autonomous parking example provided by the TuLiP toolbox. The example considers a car operating in the continuous 3x2 workspace depicted in Figure 2. The dynamics of the car are described by

$$x(t+1) = Ax(t) + Bu(t) + Ed(t) = x(t) + u(t) + d(t) \tag{7}$$

where

$$\begin{aligned}
x(t) &\in [0,3] \times [0,2] \\
u(t) &\in \mathbb{U} = [-0.1, 0.1]^2 \\
d(t) &\in \mathbb{D} = [-0.01, 0.01]^2.
\end{aligned}$$

The two labels $home$ and $lot$ in Figure 2 encapsulate two different areas in the workspace and act as atomic propositions, which evaluate to $true$ whenever $x(t) \in [0,1] \times [0,1]$ and $x(t) \in [2,3] \times [1,2]$, respectively.

The system considered is a reactive system, meaning it interacts with the environment it operates in. In this case study, the environment generates a park signal to which the car should respond. The signal is represented by the atomic proposition $park$ that evaluates to $true$ whenever the park signal is turned on.

The safety statement and goal statement for the system are given as the GR(1) formulas

$$\begin{aligned}
\varphi_t = &((\neg park \wedge \bigcirc\neg park) \Rightarrow \bigcirc\neg lot) \wedge \\
&((park \wedge \bigcirc park) \Rightarrow \bigcirc\neg home)
\end{aligned} \tag{8}$$

and

$$\varphi_g = (\Box\Diamond(park \Rightarrow lot)) \wedge (\Box\Diamond(\neg park \Rightarrow home)) \tag{9}$$

respectively. The formula in (8) states that the car is not allowed to visit the parking lot in the next time instant unless the park signal has been turned on in at least the previous two, nor is is it allowed to visit $home$ in the next time instant if the park signal has been turned on in the previous two. The formula in (9) simply tells the car to move towards $lot$ whenever the park signal is turned on and move towards $home$ whenever it is turned off.

### A. TuLiP controller synthesis

Based on the dynamics in (7) and the specifications in (8) and (9), a controller is designed using the TuLiP toolbox. In the synthesizing process, TuLiP first constructs a finite transition system of the system, also known as the abstraction process. Simply put, TuLiP divides the workspace into multiple partitions and finds feasible transitions between them. The abstraction process is proposition preserving, meaning that all continuous states within a cell in the discrete abstraction satisfy the same set of propositions as in the continuous state

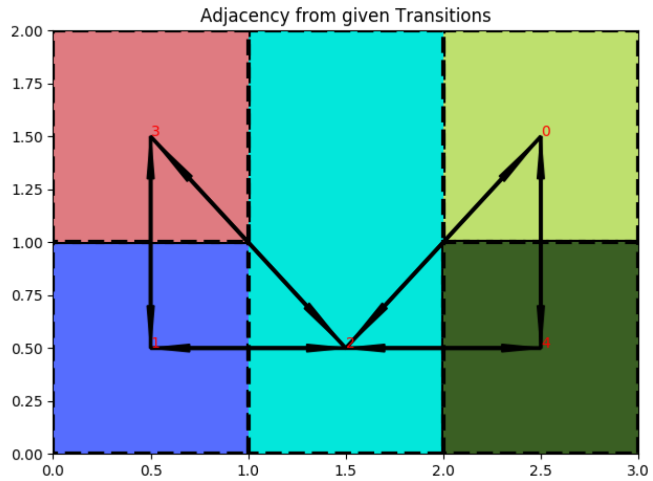

Fig. 3. The discrete abstraction, consisting of five different partitions, of the workspace in the case study. Here, $lot$ and $home$ are represented by the cells numbered with 0 and 1, respectively.

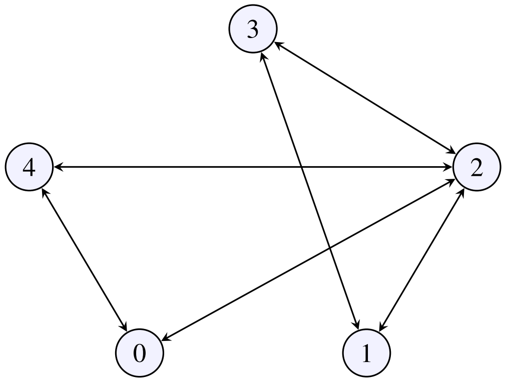

Fig. 4. The finite transition system associated with the discrete abstraction in Figure 3.

space [2]. The discrete abstraction of the workspace in the case study is depicted in Figure 3 and the associated discrete transition system in Figure 4. The cells in the abstraction are described as polytopes as follows

$$\mathcal{S}_i = \{x \mid L_i x \leq M_i\}. \tag{10}$$

The controller consists of two parts; one high-level (discrete) part responsible for path planning and one low-level (continuous) part responsible for computing the control input sequences. The high-level part is represented by a Finite State Machine (FSM) with the nodes representing the internal states of the controller. The edges represent the transitions between the internal states and contains information about which system state the controller will steer into and the associated park signal causing the action. The full system with the two-layer controller is depicted in Figure 5.

## IV. Methodology

It is widely known that the aim of testing is to either prove that the system does not behave correctly, or to gain confidence that it does. The proposed approach focuses on the former, namely to try to get the synthesized controller to fail to fulfill the specifications in (6), by using model based testing.

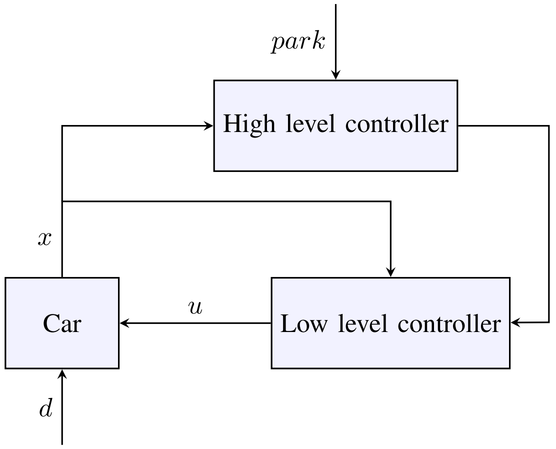

Fig. 5. Block diagram of the system with the two-layer controller.

### A. Model Based Testing

Model based testing is a software testing technique for automatic generation of test cases. MBT relies on models of the System Under Test (SUT) or the environment and is mainly used for functionality testing. A simplified description of the MBT process using models of the SUT is depicted in Figure 6.

As of today, there exist several methods for applying model based testing. In [6], they present a taxonomy of different approaches and below follows a summary of how MBT is thought to be adapted to fit the goals of this project:

- *Paradigm* - The notation used to describe the model. For this application, it would be natural to use a *transition-based notation*, as this notation focuses on describing the transitions between the states in the system. This notation would thus allow for the usage of the finite transition system shown in Figure 4.
- *Test selection criteria* - The test selection criteria is a description of what properties of the system the test should cover. A mutation coverage *fault-based criteria* would be considered in this case study, as this criteria is particularly practical when the model is a description of the SUT. The process using this criteria consists of mutating the original model and then generate tests to distinguish differences between the two models.
- *Test generation technology* - Model checking would be used as test generation technology. Model checking is a technology used to falsify or verify properties of a system by exhaustively and automatically check whether a model meets a certain specification.

Based on this, the approach for this project was outlined and it consists of the following two steps

1) find the transitions in the abstract finite-state model most sensitive to mutations in the model,
2) exploit the above transitions by applying different input sequences to the controller.

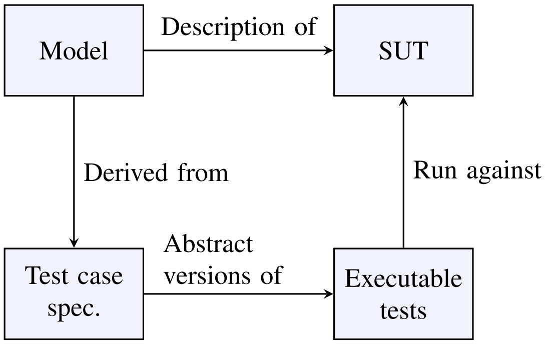

Fig. 6. Simplified flowchart of the MBT test generation process.

### B. Set of feasible initial states

Consider the feed-forward system in (1) and its corresponding discrete finite transition system in Figure 4. Given the set $\mathcal{S} = \{\mathcal{S}_0, \mathcal{S}_1, \dots, \mathcal{S}_4\}$ of discrete controlled states and a fixed time horizon $N$, $\mathcal{S}_j$ is said to be reachable from $\mathcal{S}_i$, written $\mathcal{S}_i \rightsquigarrow \mathcal{S}_j$, if there exists a control signal sequence

$$u(0), u(1), \dots, u(N-1) \in \mathbb{U}$$

such that for any sequence of disturbances

$$d(0), d(1), \dots, d(N-1) \in \mathbb{D}$$

the following holds

$$\{x(0), x(1), \dots, x(N-1)\} \in \mathcal{S}_i \quad \text{and} \quad x(N) \in \mathcal{S}_j$$

where $x(i)$ represents a continuous controlled state obtained in one of the cells in Figure 3.

By rewriting the expression of $x(t)$ to

$$x(t) = A^t x(0) + \sum_{n=0}^{t-1} (A^k B u(t-1-k) + A^t E d(t-1-k))$$

it can be seen that Equation (7) can be written as [10]

$$L \begin{bmatrix} x(0) \\ \vec{u} \end{bmatrix} \leq M - Gd \tag{11}$$

where

$$L = \begin{bmatrix} L_i A & L_i B & 0 & \dots & 0 \\ L_i A^2 & L_i AB & L_i B & \dots & 0 \\ \vdots & & & & \\ L_j A^N & L_j A^{N-1} B & \dots & L_j AB & L_j B \end{bmatrix}$$

$$G = \begin{bmatrix} L_i E & 0 & 0 & \dots & 0 \\ L_i AE & L_i E & 0 & \dots & 0 \\ \vdots & & & & \\ L_j A^{N-1} E & L_j A^{N-2} & \dots & L_j AE & L_j E \end{bmatrix}$$

$$\vec{u} = \begin{bmatrix} u(0) \\ u(1) \\ \vdots \\ u(N-1) \end{bmatrix}, \quad M = \begin{bmatrix} M_i \\ M_i \\ \vdots \\ M_j \end{bmatrix}.$$

The set $S_0$ containing the feasible initial states, $x(0)$, from which $\mathcal{S}_i \rightsquigarrow \mathcal{S}_j$, can then be obtained by projecting (11) onto to $dim(x(0))$. In Figure 7, $S_0$ for the transition $\mathcal{S}_2 \rightsquigarrow \mathcal{S}_0$ in the case study, with $N = 5$ , is plotted.

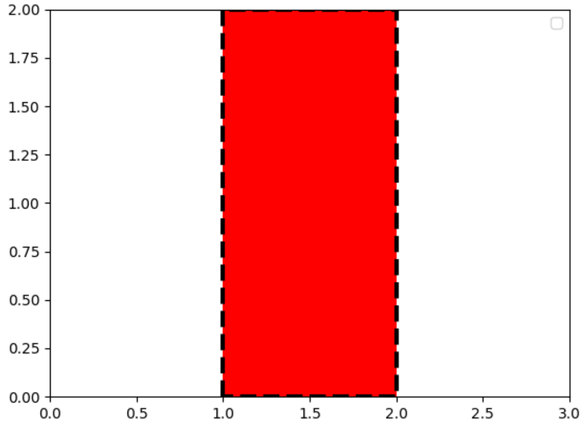

Fig. 7. The set $S_0$ containing the feasible initial sets for the transition $\mathcal{S}_2 \rightsquigarrow \mathcal{S}_0$. Here, $S_0$ constitutes the entire cell $\mathcal{S}_2$ when $N = 5$.

### C. Critical disturbances

To find the transitions in the system most sensitive to model mutations, one could investigate how the set $S_0$ behaves when the model changes, and especially how it alters with respect to increased disturbances. From Equation (11), it is clear to see that the greater the matrix multiplication $Gd$ is, the more narrow the boundaries of the polytope become. Thus, by introducing a slack vector $\varepsilon$ and solving the following optimization problem

$$\begin{aligned} \underset{\varepsilon}{\text{minimize}} \quad & min(\varepsilon_i) \\ \text{s.t.} \quad & \varepsilon = M - L \begin{bmatrix} x(0) \\ \vec{u} \end{bmatrix} + Gd \\ & d \in \mathbb{D} \end{aligned} \tag{12}$$

one would obtain the critical disturbances most likely to breach the polytope. Feeding these disturbances into the system would then increase the chance of the controller failing.

### D. Transition triggering

A transition in the system is exploited by being repeatedly triggered. By frequently revisiting the same discrete state in the finite transition system, the chances of the car being steered into an unfeasible continuous state within that partition, i.e., such that $x(0) \notin S_0$, increases. In other words, by repeating the same transition several times, an error in the controller is more likely to be found. To induce such a behavior in the system, a tweaked version of Dijkstra's algorithm is used to find a park signal sequence that would cause these actions.

Dijkstra's algorithm is an algorithm for finding the shortest paths from a given source vertex, $s \in V$, to every other vertex in a weighted, directed graph $\mathcal{G} = (V, E)$. A summarized version of the algorithm is presented below; $c_{ij}$ represents the weight on the edge $(i, j)$, $y_i$ represent the shortest path to vertex $i$ and $p_i$ its precursor.

1) Create an empty set to hold all visited veritces and a set containing all unvisited vertices;

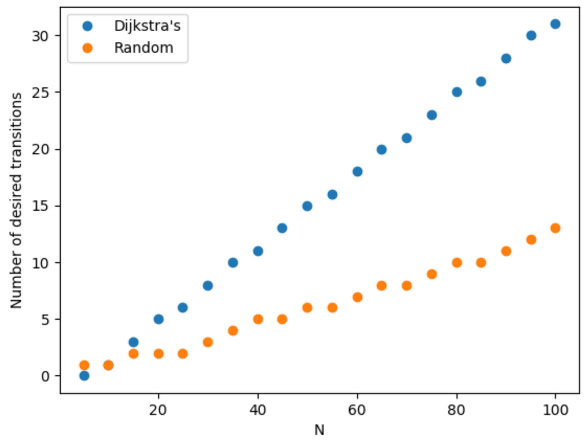

Fig. 8. Comparison between using Dijkstra's algorithm for computing the park signal sequence and randomizing it.

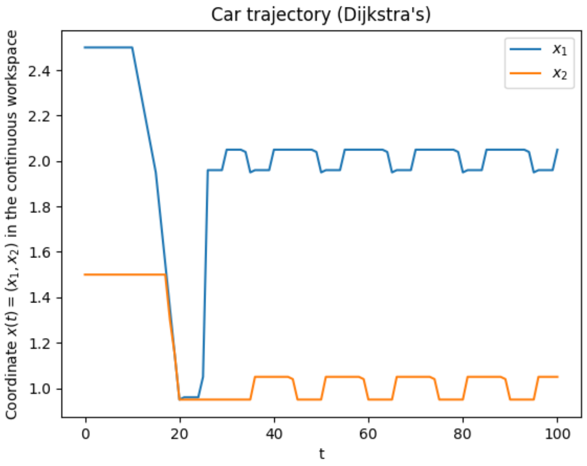

Fig. 9. The car's trajectory in the continuous workspace when responding to the park signal sequence generated by Dijkstra's algorithm.

2) initialize the distances to $y_s = 0$ and $y_j = \infty$;
3) find the closest vertex, $i$, that has not yet been visited;
4) investigate every edge, $(i, j)$, from $i$ and if $y_i + c_{ij} < y_j$, update the ditsnace to vertex $j$ to $y_j = y_i + c_{ij}$ and set $p_j = i$;
5) add vertex $i$ to the set of visited vertices;
6) return to step 3 unless all vertices have been visited.

For a more exhaustive description see [11].

Figure 8 depicts the number of times the desired transition is taken for different horizon lengths. The result show that Dijkstra's algorithm outperforms the randomized results. The same phenomenon can also be seen in Figures 9 and 10, where the car's trajectories in the continuous workspace responding to the different park signal sequences are plotted. In all figures, the park signal sequence has been generated to trigger the transition between the discrete states 2 and 0 in Figure 4.

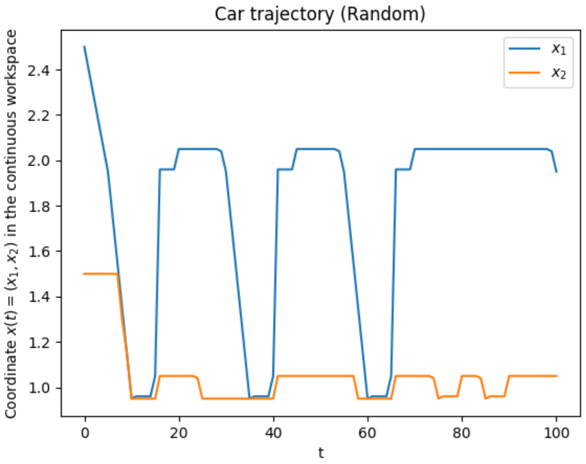

Fig. 10. The car's trajectory in the continuous workspace when responding to the randomized park signal sequence.

## V. Summary and Conclusions

Up to this point, methods for both steps in the presented approach in Sectio Due to shortage of time, no actual executable test cases were produced or run on the true system. The closest to a final result is the use of Dijkstra's algorithm to trigger transitions in the system.

## VI. Future work

## Acknowledgments

I would like to thank my mentor Richard Murray for valuable inputs and discussions during this project, and my co-mentor Sofie Haesaert for her immense support and assistance throughout this summer. I would also like to thank Petter Nilsson who acted as my stand-in co-mentor during the last weeks of my stay.

## References

[1] T. Kirkpatrick. (2007) Edmund Melson Clarke. online. [Online]. Available: https://amturing.acm.org/award_winners/clarke_1167964.cfm

[2] I. Filippidis, S. Dathathri, S. C. Livingston, N. Ozay, and R. M. Murray, "Control design for hybrid systems with Tulip: The Temporal Logic Planning toolbox," in *2016 IEEE Conference on Control Applications (CCA)*, Sept 2016, pp. 1030–1041.

[3] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. Murray, "Tulip: A Software Toolbox for Receding Horizon Temporal Logic Planning," in *2011 International Conference on Hybrid Systems: Computation and Control (HSCC)*. ACM, 2011.

[4] A. David, D. Du, K. Larsen, A. Legay, M. Mikučionis, D. Poulsen, and S. Sedwards, "Statistical Model Checking for Stochastic Hybrid Systems," *Electronic Proceedings in Theoretical Computer Science*, vol. 92, August 2012.

[5] H. Abbas, B. Hoxha, G. Fainekos, and K. Ueda, "Robustness-guided temporal logic testing and verification for Stochastic Cyber-Physical Systems," in *The 4th Annual IEEE International Conference on Cyber Technology in Automation, Control and Intelligent*. IEEE, 2014.

[6] M. Utting, A. Pretschner, and B. Legeard, "A Taxonomy of Model-based Testing Approaches," *Softw. Test. Verif. Reliab.*, vol. 22, no. 5, pp. 297–312, Aug. 2012. [Online]. Available: http://dx.doi.org/10.1002/stvr.456

[7] C. Baier and J. Katoen, *Principles of Model Checking*. 55 Hayward Street, Cambridge, MA 02142: The MIT Press, 2008.

[8] J. Ouaknine and J. Worrell, "Some Recent Results in Metric Temporal Logic," in *Formal Modeling and Analysis of Timed Systems*, F. Cassez and C. Jard, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–13.

[9] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of Reactive(1) Designs," in *Proceedings of the 7th International Conference on Verification, Model Checking, and Abstract Interpretation*, ser. VMCAI'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 364–380. [Online]. Available: http://dx.doi.org/10.1007/11609773_24

[10] T. Wongpiromsarn, "Formal Methods for Design and Verification of Embedded Control Systems: Application to an Autonomous Vehicle," Ph.D. dissertation, California Institute of Technology, 1200 E. California Blvd., May 2010.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.