2018 15th International Conference on
Control, Automation, Robotics and Vision (ICARCV)
Singapore, November 18-21, 2018

Interactive Analysis and Visualization of Digital Twins in High-Dimensional State Spaces

Linus Atorf
Jürgen Roßmann
Institute for Man-Machine Interaction (MMI)
RWTH Aachen University
52074 Aachen, Germany
{atorf,rossmann}@mmi.rwth-aachen.de

***Abstract*— Digital Twins (DTs), an emerging concept from Industry 4.0, are virtual representations of real technical assets. Multi-domain 3D simulation systems can bring DTs to life, even before their physical counterparts are finished. A DT's internal state can be fed from its real twin or generated by simulation. Access to this high-dimensional state of a DT is the key for various analysis and visualization methods presented in this paper. We introduce a generic formalism of state space for DTs and utilize it in an application scenario for automated driving. Throughout this example, methods for state logging and replays, data analysis, and visualization within 3D simulation frameworks are presented. Clear definitions for state variables, vectors, trajectories, and time series help slicing the DTs' state spaces of enormous dimensionality. The presented methodology does not only support the development of intelligent algorithms for autonomous driving, but is also the basis for further use cases of DTs involving optimization, mental models, and decision support systems.**

I. Introduction

According to Gartner, Digital Twins (DTs) are one of the top ten technology trends for 2018 [1]. Gaining traction from the Industry 4.0 movement started in 2011, they now play a key role not only for industry, cyber-physical systems, and the Internet of Things, but are also suitable for a wide range of applications. A survey of available literature concerning DTs and their usefulness is given in e.g. [2].

A DT is the virtual representation of a real technical asset, including its appearance, behavior, and interfaces. The "data" of DTs, i.e. their internal state, can originate from different sources: a DT can be operated "in sync" with its physical twin for e.g. monitoring purposes. Another possibility is to generate the DT's state using simulation. Once a sufficient model is developed and all components needed by a certain scenario are available as DTs, a close-to-reality multi-domain 3D simulation system can "drive" the state variables of DTs and thus bring them life. Sensor and actuator simulation modules make it possible to use DTs in pure virtual environments, even before their real counterparts are built. The concept of so-called Experimentable Digital Twins (EDTs) was presented before, see e.g. [3].

The ideas in this paper concern the state space of DTs and how insights can be gained from analyzing "historic twin data", i.e. their state trajectories. Even though we develop the concepts using example data from EDTs in 3D simulation environments (so-called Virtual Testbeds), our findings are independent from the nature of the data sources: DTs operated in sync with their real twins can also profit from our methodology.

Certain properties of digital entities are typical for the whole "digital world" and lead to the observed paradigm shifts. One such paradigm is that, from a technical standpoint, copies of digital data are essentially free and modifying data is very easy. As example, consider word processing software, where the consequences are ubiquitous: copying words or editing them later on is core functionality and was not possible with physical typewriters.

In this paper, we harness such properties of DTs. As "copies are cheap", we can instantiate many different variants and simulate their outcome for comparison. To us, DTs are fully transparent, as they "live" inside the memory of computers which can easily be accessed. Thus, their internal state is available for analysis as described in the rest of this submission.

Scenario Description

The application examples in this paper originate all from one scenario, which is taken from the automotive sector and is used to support the development of self-driving cars. The setup is loosely based on a traffic situation from 2016 in Florida, USA, leading to the crash of a Tesla Model S into a truck [4]. The DT of a car with autonomous driving assistance software enabled is approaching a road

Fig. 1: Simulated scene with Digital Twins of multiple vehicles. A self-driving car approaches an intersection with oncoming traffic. The truck is a about to make a left turn.

978-1-5386-9582-1/18/$31.00 ©2018 IEEE

intersection (the simulated scene is depicted in Fig. 1). The oncoming traffic consists of a truck which is about to turn left and cross the approaching car's traffic lane. The scenario begins at a point in time where many possible outcomes can unfold. Given a constant behavior of the truck, i.e. its driver keeps making the intended turn, there is still enough time for the car's (virtual) driver to complete an emergency brake or turn maneuver, or a combination of both. One such possible turn of events is simulated using DTs and shown in Fig. 2. Without changing the car's steering or speed however, i.e. by not intervening, a crash into the truck is inevitable.

Fig. 2: Trajectory of a possible evasion maneuver performed by the car's Digital Twin in simulation.

II. Formalism

For the shown use case, the key feature of EDTs is their ability to accurately simulate possible state trajectories, i.e. predict future outcomes for given situations. The foundation of all applications described later is full access to the DT's internal state within the simulator. In this section, we establish a clear formalism in order to describe methods for data logging, replay, analysis, and optimization.

A. Input Data

EDTs are fed with a stream of input data, usually commands by either users or agent software. This is denoted by an input function f_{in}

$$\underline{u}(t) = f_{\text{in}}(t) \quad t \in \mathbb{R}^{\geq 0} \tag{1}$$

which produces an input vector $\underline{u}(t)$:

$$\underline{u}(t) = [u_1(t),\ u_2(t),\ \ldots]^T \quad u_i \in \mathbb{V} \tag{2}$$

In computer simulations, $\mathbb{V}$ is made up of all possible values that variables can take, i.e. values for datatypes such as booleans, integers, floating point numbers, as well as strings and complex datatypes such as arrays, structures, or enumerations. In this case, the car's input variables may be steering angle, accelerator and brake pedal positions, current gear chosen, etc. For a software agent controlling the car, input variables may also include target speed or other "higher level" information such as way points for a planned route.

For a given input function $f_{\text{in}}(t)$, a complete input trajectory $\boldsymbol{U}(t)$ up to time t can be produced ($t, t' \in \mathbb{R}^{\geq 0}$):

$$\boldsymbol{U}(t) = \left\{ f_{\text{in}}(t') \mid 0 \leq t' \leq t \right\} \tag{3}$$

Basically, $\boldsymbol{U}(t)$ is a "sequence of input commands".

B. Simulation State

An EDT consists of an executable simulation model M. The simulator is able to evolve the state of this model over time and generate a vector $\underline{s}(t)$ given its initial state $\underline{s}_0 := \underline{s}(0)$ and an input trajectory $\boldsymbol{U}(t)$:

$$\underline{s}(t) = M(\underline{s}_0,\ \boldsymbol{U}(t),\ t) \tag{4}$$

Similar to the elements of input vectors, elements of $\underline{s}(t)$ are typed values:

$$\underline{s}(t) = [s_1(t),\ s_2(t),\ \ldots,\ s_N(t)]^T \quad s_i \in \mathbb{V},\ N \in \mathbb{N} \tag{5}$$

Those state variables $s_i(t)$ comprise all relevant properties of the simulation model including physical quantities such as positions, velocities, masses, temperatures, joint angles, to just name a few. Geometric information (lengths, dimensions), program counters and pointers and many other "pieces of information" (such as e.g. visualization options) also belong to this definition. N denotes the dimensionality of the state space; in our example, we have $N = 169,171$. The $s_i(t)$ may stay constant during a particular simulation run—in fact, a huge part of them usually do. In our case, only $N_{\text{log}} = 2,348$ of N state variables changed during a simulation run. In a way, $\underline{s}(t)$ can be imagined as the contents of the computer's memory where the EDT is executed.

Similar to the definition of $\boldsymbol{U}(t)$ in (3), the full state trajectory $\boldsymbol{S}$ of an EDT's simulation run until time t becomes:

$$\boldsymbol{S}(\underline{s}_0,\ \boldsymbol{U}(t),\ t) = \{M\left(\underline{s}_0,\ \boldsymbol{U}(t'),\ t'\right) \mid 0 \leq t' \leq t\} \tag{6}$$

According to the formalism just introduced, the rendered screenshot in Fig. 1 depicts a single state $\underline{s}(t_0)$, which we can name as $\underline{s}_0$ without loss of generality—i.e. $t_0 = 0$ marks the start of our simulation runs and is kept constant.

C. Simulation Variants

Now, there may be different input trajectories $\boldsymbol{U}_j(t)$ which are identified by their index $j = 1, \ldots, n$ with $j \leq n \in \mathbb{N}$. In the given scenario with the same constant initial setup, different inputs lead to different simulation variants—in a reproducible way due to determinism of M. Hence, it is sufficient to identify a complete simulation variant and all consequences by j. As $\underline{s}_0$ does not change, a convenient way to express (4) is:

$$\underline{s}(j,t) = M(\underline{s}_0,\ \boldsymbol{U}_j(t),\ t) \tag{7}$$

To further simplify the notation, we assume a certain maximum simulation time that fits our scenario, e.g. $t_{\text{max}} = 20\,\text{s}$. In turn, a shorter notation to identify a state trajectory follows:

$$\boldsymbol{S}(j) = \left\{M\left(\underline{s}_0,\ \boldsymbol{U}_j(t'),\ t'\right) \mid 0 \leq t' \leq t_{\text{max}}\right\} \tag{8}$$

It is now obvious that $\boldsymbol{S}(j)$ just consists of the matching states $\underline{s}(j,t)$:

$$\boldsymbol{S}(j) = \left\{ \underline{s}(j,t') \mid 0 \leq t' \leq t_{\text{max}} \right\} \tag{9}$$

Note that thanks to M, the simulator can reproduce a full state trajectory $\boldsymbol{S}(j)$ for any given j. One of many possible variants $\boldsymbol{S}(j)$ is visualized in Fig. 2.

Applying this concise notation to elements of $\underline{s}(j,t)$ from (5) yields

$$\underline{s}(j,t) = [s_1(j,t),\ s_2(j,t),\ \ldots]^T \quad s_i(j,t) \in \mathbb{V} \tag{10}$$

where we write $s(i,j,t) := s_i(j,t)$. The state trajectory $\boldsymbol{S}(j)$ can also defined via state variables:

$$\boldsymbol{S}(j) = \left\{ s(i,j,t') \mid \forall\ i,\ 0 \leq t' \leq t_{\text{max}} \right\} \tag{11}$$

D. Time Series

Having access to every state variable $s(i,j,t)$, it is easy to define a time series $\underline{\tau}(i,j)$ that contains the complete history of values (within t_{max}) of the selected i-th item and j-th variant:

$$\underline{\tau}(i,j) = \left\{ s(i,j,t') \mid 0 \leq t' \leq t_{\text{max}} \right\} \tag{12}$$

E. Summary

We now have four important "building blocks" of simulation state for the various visual analysis methods described in the following sections. Listed bottom-up, they are:

1) **State variable** $s(i,j,t)$: Current value of the i-th property of the DT for variant j at time t.
2) **Time series** $\underline{\tau}(i,j)$: History of all values that occurred for the i-th property in variant j, i.e. for all times $0 \leq t' \leq t_{\text{max}}$.
3) **State vector** $\underline{s}(j,t)$: Full state of the complete DT, comprising all state variables $s(i,j,t)\ \forall\ i$ for variant j at time t.
4) **State trajectory** $\boldsymbol{S}(j)$: Contains the DT's complete state history until t_{max} for given variant j.

III. Data Logging and Replay

In order to properly analyze and visualize simulation results, simulation state needs to be logged during simulation runs. The goal is to be able to reproduce full state trajectories without having to invoke calculations involving M.

A. Formal Description

The key concept here is to monitor state variables $s(i,j,t)$ during simulation and store their new value whenever state changes. Such changes are called events, which we number in their order of occurrence using ascending integers $k \in \mathbb{N}$. Each state variable gets its very own numbering, i.e. we actually have k_{ij} for each unique combination of (i,j)—it is not necessary to write these additional indices down, however.

The simulation time of the event is denoted by $t_{ij}^k \in \mathbb{R}^{\geq 0}$. Even if several events have the same timestamp, the order of events is still preserved thanks to k. The actual value of the state variable produced by such an event is:

$$x_{ij}^k = s(i,j,t_{ij}^k) \quad x_{ij}^k \in \mathbb{V} \tag{13}$$

All relevant data of an event is hence contained in the following tuple:

$$E = \left(i,\ j,\ k,\ t_{ij}^k,\ x_{ij}^k\right) \tag{14}$$

It should now be clear that the indices in the value x_{ij}^k and its "time of creation" t_{ij}^k are introduced so we can always easily match time and value. Note that the i-th state variable of variant j has its very own timeline, as events can happen independently. We can name all timestamps of this state variable

$$\underline{T}_{ij} = \left[\ t_{ij}^1,\ t_{ij}^2,\ \ldots,\ t_{ij}^k,\ \ldots\right]^T \tag{15}$$

and the matching values accordingly:

$$\underline{X}_{ij} = \left[\ x_{ij}^1,\ x_{ij}^2,\ \ldots,\ x_{ij}^k,\ \ldots\right]^T \tag{16}$$

The values $\underline{X}_{ij}$ are precisely the contents of time series $\underline{\tau}(i,j)$ with timestamps t' taken from $\underline{T}_{ij}$.

B. Storing Data

Logging simulation data for later processing, analysis, and visualization now just means to store all incoming events E and increase k. This format is especially suitable for storing data in five-column tables of relational databases; the tuple (i,j,k) can be used as primary key. Other table-, column-oriented or NoSQL databases, as well es key-value stores or time series databases are suitable alternatives of course. Even CSV-textfiles are a valid approach. No matter what data store is used, the event tuples E must be properly serialized into text or binary streams. This may need some implementation work when the values x_{ij}^k contain complex datatypes.

For storage back-ends, we have had good experience with "classic" SQL databases such as SQLite [5] and PostgreSQL [6]. SQLite is able to handle the data load of about 100k events per second simulation time on commodity hardware when the configuration is adapted to our usecase (e.g. turning off transaction safety). For more details, including a database schema, see [7]. Another recommendation is to use InfluxDB, a special purpose time series database [8].

In the end, the choice of data store is mainly influenced by the flexibility and performance characteristics when reading the data back for analysis.

C. Querying Data

Having a database or other archive with saved simulation data is only useful when data can be restored from it. The basic requirement is to be able to load state variables $s(i,j,t)$, as all other entities (time series, state vectors, and state trajectories) can be reconstructed from it. In theory, the only challenge in providing $s(i,j,t)$ from data store is to find the greatest k' that satisfies $t_{ij}^{k'} \leq t$ and then return $x_{ij}^{k'}$. This works as value $x_{ij}^{k'}$ is valid from time $t_{ij}^{k'}$ until it changes to value $x_{ij}^{k'+1}$ at time $t_{ij}^{k'+1}$.

For a given i and j, the events are ordered by k and hence also by t_{ij}^k. This means binary search in $\underline{T}_{ij}$ is sufficient and yields both $t_{ij}^{k'}$ and $x_{ij}^{k'}$ with complexity $\mathcal{O}(\log N_{ij})$, where $N_{ij} \in \mathbb{N}$ is the number of elements in $\underline{T}_{ij}$ (and in $\underline{X}_{ij}$).

For real world performance on commodity hardware, the chosen data store back-end can make big differences when loading data, depending on the size of the log and on the queries that should be performed on it. If a relational SQL database with a five-column table holding the event tuples E as defined in (14) is used, convenient and fast SQL queries can be used to fetch whole $\underline{T}_{ij}$ and $\underline{X}_{ij}$ vectors at a time—that means it is easy to restore desired time series $\underline{\tau}(i, j)$. For details and similar usage examples, see [7].

D. Simulation Playback

The mechanism for playing back whole simulation runs is essentially the same as described previously: state data needs to be queried from the data store and then applied to the 3D simulation environment. As the simulator supports rendering a scene from state, i.e. drawing $\underline{s}(j, t)$ in 3D, it makes no difference whether $\underline{s}$ was produced via calculations or loaded from replay data. For real-time simulation playback of given variant j, we only need to advance the desired time t' in small steps dt' and then load and display $\underline{s}(j, t')$ each time.

The speed of replay animation can be changed of course to allow for "slow-motion" or "fast-forward" functionality. As our presented way to structure simulation data provides random access to state in t and j, replays can even be run backwards, variants can be changed on the fly, or desired points in time can be jumped to.

E. Performance Considerations

Depending on the extent of the simulated scenario, enabling smooth real-time replays can be challenging, which of course depends in turn on available hardware.

As mentioned before, the size N of our scenario's state vector $\underline{s}(j, t)$ is $N = 169,171$ for all variants j, with $N_{\log} = 2,348$ state variables s_i that actually changed values during a simulation run. On average, $78,105$ events per second simulation time were recorded into an SQLite database, with a data rate of about 18 MiB/s using text serialization. While simulation and data logging can be performed in near real time, querying all data back from the database takes a couple of minutes on a commodity workstation. To enable fluent replays and seamless "jump to point in time"-functionality without any lag, the whole replay log is loaded back into memory and converted into binary serialization. This means, the times and values for all n variants, i.e. $(\underline{T}_{ij}, \underline{X}_{ij}) \; \forall \; i \; \forall \; j$, are kept in RAM; in this case $N \cdot n = 25,828$ time series with a total of $N_E = 12,887,239$ events. Total memory consumption of replay data is about 1.1 GiB.

When a state $\underline{s}(j, t)$ is applied, $N_{\log} = 2,348$ binary searches have to be performed. Benchmarking showed the required CPU time for this is negligible; the most expensive tasks are deserializing and applying values x_{ij}^k and rendering the scene from $\underline{s}(j, t)$.

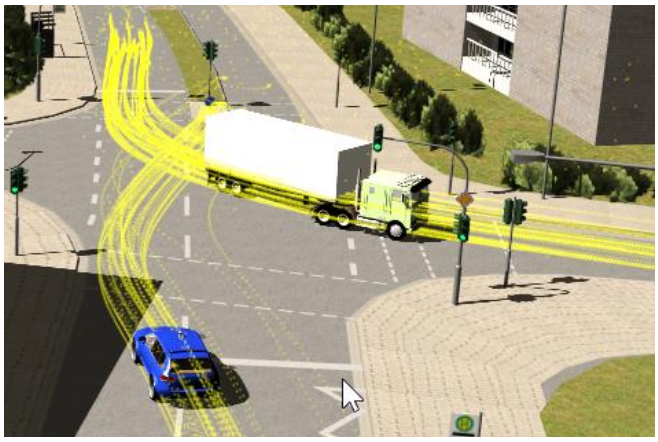

Fig. 3: Many time series $\underline{\tau}(i, j')$ (yellow points) for a fixed variant j' and several properties i, drawn into the 3D scene. Shown are positions of different car parts over time.

IV. Analysis and Visualization

Main advantage of our presented simulation state logging mechanism is that it enables many different powerful analysis and visualization methods, replay just being one of them. Even though our data store allows very flexible queries which make classical data analysis on e.g. time series data quite a pleasant task, in this section we focus on visualization metaphors that harness the full power of DTs within 3D simulators, providing the basis for sophisticated decision support systems or mental models, such as described in [9].

A. 3D Time Series

Time series are a very common approach to visualize and analyze the history of variables of dynamic processes. The classic way is to plot scalar values or single elements of multi-dimensional quantities over time on a 2D canvas—well-known time series charts. As we have access to the full simulation state at all times, we know the "context" of data creation: not only temporal details of an event, but also spacial information, i.e. when and where the data point was created, within the 3D scene.

This new understanding enables plotting time series directly into the 3D scene as shown in Fig. 3. Shown are multiple time series $\underline{\tau}(i, j')$ for a given variant j' and several state variables i, here geometrical positions of different car parts. Note that a full state trajectory $\boldsymbol{S}(j')$ contains *all* time series $\underline{\tau}(i, j') \; \forall \; i$ as defined in (**??**) and shown in Fig. 2, whereas here in Fig. 3 only *some* time series with selected i are drawn.

For further analysis, a colormap could be applied onto the point trajectories, indicating e.g. speeds of the moving parts or other scalar quantities.

B. Isochrone Maps

The previous subsection only deals with state data of a *single* variant j'. However, new insights can be gained by comparing a state variable i' across *all* variants $j = 1, \ldots, n$. As next example, we choose i' to represent a position on the

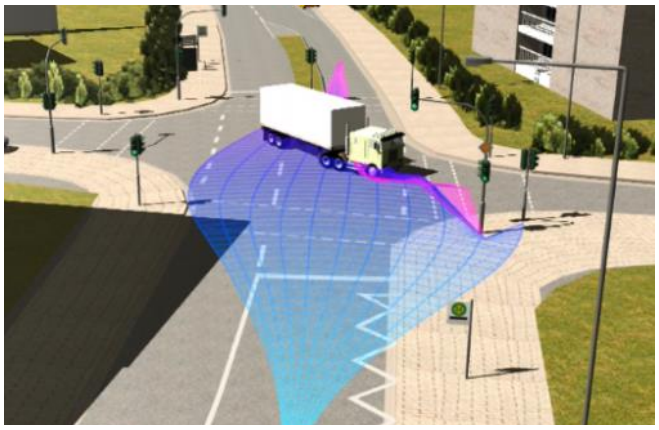

Fig. 4: Isochrone map. Plotted are positions of the car's front over equidistant points in time for all variants. Isochrones are horizontal lines of constant color, connecting states from different variants at the same time.

Fig. 5: "Variant ghosts" of Digital Twins. Shown are states $\underline{s}(j, t')\ \forall\ j$, i.e. all possible variants j for fixed time t' superimposed in one 3D scene.

car's front geometry and create an equidistantly spaced time vector $\underline{T}' = [t'_1,\ t'_2,\ \ldots,\ t'_l,\ \ldots]$ with $t'_l = t_0 + l\Delta t$ and $\Delta t \in \mathbb{R}^{\geq 0},\ l \in \mathbb{N}$.

We then plot the state values at time points in $\underline{T}'$ as dots into the 3D scene for all variants, i.e. the dots are $s(i', j, t_l)\ \forall\ j\ \forall\ t_l \leq t_{\text{max}}$. The lines connecting $s(i', j, t_l)$ for l fixed and along $j = 1, \ldots, n$ are called isochrones. Each of them visualizes the state value of all variants for the same given point in time t_l. When isochrones for increasing time steps are colored accordingly as shown in Fig. 4, viewers can get a good impression of how the whole family of DTs evolves. It can be observed that evasion maneuvers to the right lead to crashes with the traffic light post.

C. Variant Ghosts

While isochrones are a great tool for analysis and visualization, the only use a tiny fraction of the whole simulation state space, as typically only a single property i is selected. 3D time series on the other hand can show many variables at once, but only for one single variant j. We now introduce the concept of "variant ghosts" of DTs in order to compare different simulation variants based on their full state. For an arbitrarily chosen time t', the state vectors for all variants $\underline{s}(j, t')\ \forall\ j$ are superimposed into one single 3D scene. The result is shown in Fig. 5. One particular j' is currently selected an drawn regularly, while all other twin versions $j \neq j'$ are rendered as variant ghosts; here slightly transparent with different colors according to j.

V. Interactive Analysis

The previous section showcases different visualization metaphors for simulation data analysis using DTs. However, these visualizations still have many open parameters, the most important being the choice of property i, variant j and simulation time t for state variables $s(i, j, t)$. Features such as colormaps, transparencies, and data point spacings are further options that can be selected or iteratively adapted.

A. Hardware User Interface

Efficient data analysis requires a flexible and responsive user interface, i.e. the parameters mentioned above must be controllable by analysts in real-time. While a well-designed software GUI is a sensible first step, more sophisticated methods for human-computer interaction are needed. Experience in this area can be transferred from music studios and video editing rooms, where professional users are confronted with a great variety of data sources, tracks, and parameters. There, prevalent hardware controls are faders and rotary knobs, which enable humans to easily set parameters within ranges and get instant feedback. Borrowing concepts from this field, we extended our simulation environment with an interface for MIDI controllers [10].

Most importantly, we now have hardware sliders to control variant j and simulation time t for the different visualization modes presented above, as well as many more sliders and rotary encoders for all other visualization settings.

B. Interactive Velocity Maps

Another way to create interactive data analysis tools involving EDTs is to link classic plots directly with state vectors. We illustrate this concept using the example of a velocity map as shown in Fig. 6. Here, a quantity called "average model speed" $\overline{v}(j, t)$ is plotted as map over variant j and time t. It is calculated by taking the average spacial change of all recorded moving positions within one simulation run, which gives an idea of the "level of overall motion" of all variants compared. But the shown velocity map is just a canvas for the actual interaction! Moving the mouse cursor or dragging the marked circle on a touch-sensitive device changes the position (j, t). Both parameters j and t are continuously sent to the simulation and analysis environment, displaying the state $\underline{s}(j, t)$ in one of many possible modes, as presented before. The picture shown in Fig. 3 is precisely the state indicated by the current marker position (j', t') in Fig. 6. Thus, the classic 2D plot is interactively linked with the 3D scene and is instantly updated whenever the cursor

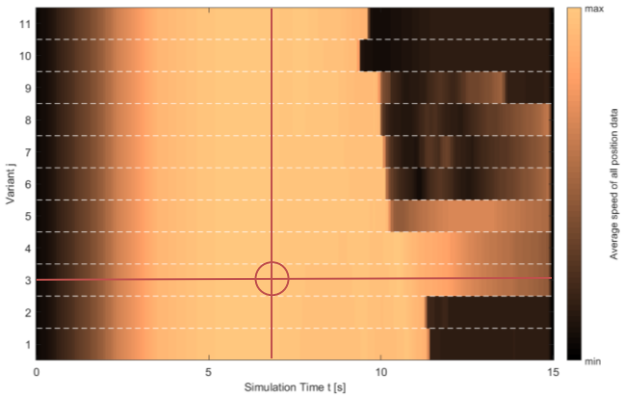

Fig. 6: Velocity map showing "average model speed" (spacial change of all recorded moving positions) across variant j and time t. The marked coordinate (j, t) represents the state visualized in Fig. 3.

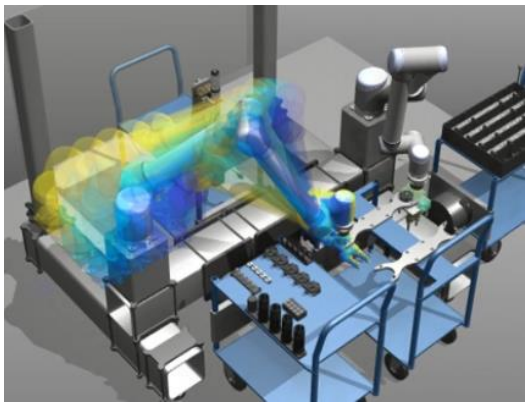

Fig. 7: Variant ghosts of Digital Twins in ReconCell [11], illustrating different base positions for a UR-10 robot.

position changes. Note how only variants $j = 3$ and $j = 4$ avoid crashes entirely, and $j = 5$ has only a slight impact (change in speed) at $t \approx 11\,\mathrm{s}$.

VI. Further Applications

The presented example above features just one possible application scenario for automated driving. However, the formalism and mechanisms developed in this paper have a wide variety of applications to many different fields. We demonstrate this using a scenario from the EU H2020 project "ReconCell" [11] for the field of industrial robotics. The project's aim is to provide reconfigurable robotic workcells for SMEs with relatively small batch sizes. Thus, reconfigurations of these cells are required as to adapt to changing manufacturing processes for different product variants. Here, DTs and especially the analysis of their state trajectories play a key role. Only when possible setups can be simulated and investigated virtually before their real twins are built, the required cell utilization can be guaranteed. Fig. 7 shows variant ghosts of the DT of a UR-10 robot. In this example, the robot's base position was varied to evaluate the consequences on overall cycle time and energy consumption for a given process. With access to the DTs complete state space at all times for all simulated variants, optimizing the cell configuration or finding the best input trajectory becomes a manageable task. We have performed simulation-based optimization for ReconCell before, see e.g. [12].

VII. Conclusion

In this contribution, a formalism to describe state of DTs is introduced. Multiple instances of DTs are simulated and their state variables are logged for analysis and visualization. We show how the high-dimensional state space of DTs and their trajectories can be dissected to gain insights, to enable further processing, or to provide the basis for further application scenarios such as simulation-based optimization or decision support systems.

The example scenario from automated driving opens up interesting possibilities: agent software itself can use the concepts of this paper; agent strategies can be evaluated using DTs with state logging and thus support the development of algorithms. This is true everywhere DTs are applicable.

Acknowledgements

This work was supported in part by the research project INVIRTES, funded by the German Aerospace Center (DLR) with funds provided by the Federal Ministry of Economics and Technology (BMWi) under grant number 50 RA 1306, and by the H2020 project ReconCell under grant H2020-FoF-680431.

References

[1] D. W. Cearley, B. Burke *et al.* (2017) Top 10 Strategic Technology Trends for 2018. ID: G00327329. [Online]. Available: https://www.gartner.com/ngw/globalassets/en/information-technology/documents/top-10-strategic-technology-trends-for-2018.pdf

[2] E. Negri, L. Fumagalli, and M. Macchi, "A review of the roles of digital twin in cps-based production systems," *Procedia Manufacturing*, vol. 11, pp. 939 – 948, 2017, FAIM 2017, Modena, Italy.

[3] M. Schluse, M. Priggemeyer, L. Atorf, and J. Rossmann, "Experimentable Digital Twins—Streamlining Simulation-based Systems Engineering for Industry 4.0," *IEEE Transactions on Industrial Informatics*, vol. PP, no. 99, pp. 1–9, 2018.

[4] National Transportation Safety Board. (2017) Accident Report NTSB/HAR-17/02, PB2017-102600: Collision Between a Car Operating With Automated Vehicle Control Systems and a Tractor-Semitrailer Truck Near Williston May 7, 2016, Florida.

[5] D. Hipp, D. Kennedy, and J. Mistachkin. (2018) SQLite. [Online]. Available: https://www.sqlite.org

[6] The PostgreSQL Global Development Group. (2018) PostgreSQL Database. [Online]. Available: https://www.postgresql.org

[7] L. Atorf, T. Cichon, and J. Rossmann, "Flexible data logging, management, and analysis of simulation results of complex systems for erobotics applications," in *ESM*, Leicester, UK, 2015, pp. 385–391.

[8] InfluxData. (2018) InfluxDB. [Online]. Available: https://www.influxdata.com/time-series-platform/influxdb/

[9] J. Rossmann, E. Guiffo Kaigom, L. Atorf, M. Rast, G. Grinshpun, and C. Schlette, "Mental Models for Intelligent Systems: eRobotics Enables New Approaches to Simulation-Based AI," *KI - Kuenstliche Intelligenz*, vol. 28, no. 2, pp. 101–110, 2014.

[10] The MIDI Association. (1996) MIDI 1.0 Specification. [Online]. Available: https://www.midi.org

[11] The ReconCell project consortium. (2017) A reconfigurable robot workcell for fast set-up of automated assembly processes in SMEs. [Online]. Available: http://www.reconcell.eu/

[12] L. Atorf, C. Schorn, J. Rossmann, and C. Schlette, "A framework for simulation-based optimization demonstrated on reconfigurable robot workcells," in *IEEE International Symposium on Systems Engineering (ISSE), Vienna, Austria, pp. 178-183*, 2017.