RobWoT: Generating Real-Time Digital Twin Simulations for the Robotic Web of Things

Fady Salama[1], Zucheng Han[1], Ege Korkan[2], Sebastian Käbisch[2], Sebastian Steinhorst[1]
[1] *Technical University of Munich*, Germany, Email: {fady.salama, zucheng.han, sebastian.steinhorst}@tum.de
[2] *Siemens AG*, Germany, Email: {ege.korkan, sebastian.kaebisch}@siemens.com

***Abstract*—A cornerstone of the Fourth Industrial Revolution is Internet of Things (IoT) technologies, which enable unprecedented connectivity in industrial settings, facilitating novel approaches for industrial automation. However, the high fragmentation in the IoT ecosystem hinders the potential of IoT technologies in the industry. To solve this problem, the W3C proposes the Web of Things (WoT) and its Thing Description (TD), a JSON-LD document for describing the API exposed by a device, and the inputs and outputs of each interaction. In contrast to simple devices, describing the capabilities of robots and accurately depicting their workspace using the TD is a significant challenge and has to be done manually. Furthermore, there is a lack of simulation frameworks that help validate Web-enabled robots. In this paper, we introduce RobWoT, a method and an open-source simulation framework for automatically generating WoT-compliant Digital Twin (DT) and its corresponding, annotated TD using only a Unified Robotic Description Format (URDF) as an input. These DTs can be used to preemptively test a robot's movement command to prevent undesired behavior or collisions, enabling simulation-in-the-loop scenarios. We evaluate the execution time of our generation algorithm and the accuracy of a DT's movement compared to a physical robot. Our evaluation shows that our generation method scales linearly with collidable artifacts in the simulation scene and that our DT deviates only 6.8 mm on average from the target position. Our proposed method makes developing and verifying Web-enabled robotics applications more standardized, streamlined, and easy to set up.**

***Keywords*—Internet of Things, Web of Things, Robotics, Digital Twins, Simulation**

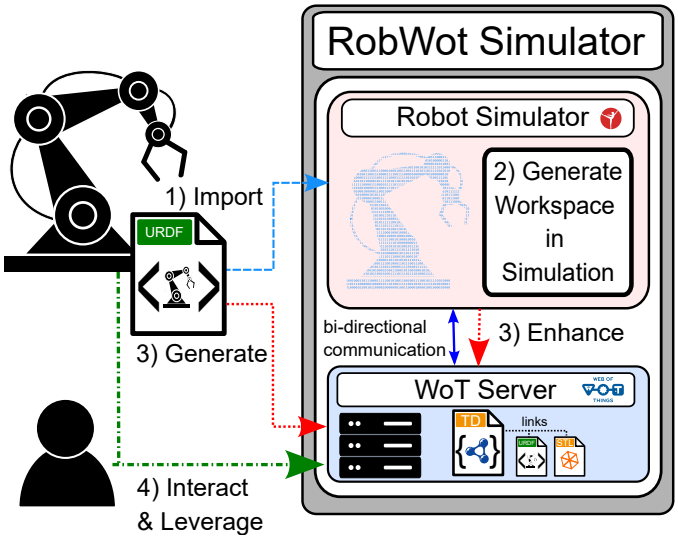


Fig. 1: In this paper, we introduce **RobWoT**, a method and an open-source simulation framework for automatically generating Web of Things (WoT)-enabled Digital Twins (DTs) and their corresponding Thing Description (TD) using only a robot's Unified Robotic Description Format (URDF) file as an input. After 1) importing the URDF in the Coppeliasim simulator, 2) we generate the workspace based on the simulation scene, and 3) use the workspace information alongside the information in the URDF to generate a server for controlling the DT, and the corresponding TD that links both the URDF and workspace STL file. 4) The DT can be, e.g., used to develop, verify correct behavior to avoid collisions and simulation-in-the-loop applications.

I. Introduction

Adopting Internet of Things (IoT) solutions in industrial applications is a cornerstone of Industry 4.0 and its vision of smart, interconnected, and digitally-enabled factories, as argued by [1]. Evidently, IoT-enabled robots are significant actors in such smart factories, providing innovative options for designing manufacturing processes by utilizing the ability to control such robots remotely using established internet technologies. Furthermore, IoT-enabled robots open the door for manufacturing data accumulation and collection based on integrated sensors, which can be used to optimize the manufacturing process to increase manufacturing efficiency and minimize cost and errors. As such, the robotics industry is forecast to significantly expand in the following years [2], with different companies offering different smart robot solutions and services. However, this results in high fragmentation in the IoT landscape in general and in the Internet of Robotic Things (IoRT) more specifically, minimizing the interoperability of such devices and services, which in turn defeats the benefit of using smart robots and minimizes the impact that can be gained by utilizing such solutions.

[3] argues that one way to solve this issue is to utilize Web technologies to ensure interoperability on the application layer of the Internet Protocol (IP) stack, where data formats and communication protocols are standardized. The Semantic Web, an extension of the Web, also provides the means to uniformly describe data in a machine-readable fashion, facilitating semantic reasoning, which robotic systems can utilize to plan and execute actions autonomously. [3] also notes that this approach has not been used to its full potential, as most services provided by robots in literature are not exposed as an Application Programming Interface (API) nor are they described using semantic technologies.

As such, the World Wide Web Consortium (W3C) proposed the Web of Things (WoT), a set of standards that facilitate the interoperability between IoT devices and services, called Things, in the context of this paper. At the core of these

```
{"title": "Robot with 2 DoF",
 "links": [{ "rel": "model",
    "href": "http://example.com/model.urdf",
    "type": "application/urdf+xml"}],
 "properties": {
    "cartesianPosition": {
        "type": "object",
        "properties": {
            "x": {"type": "number", "minimum":
↪  -5,"maximum": 5},
            "y": {"type": "number", "minimum":
↪  -5,"maximum": 5}},
        "forms": [{"href": "http://rb/pos"}]}},
 "actions": {
    "moveToCartesianPos": {
        "input": {
        "type": "object",
        "properties": {
            "x": {"type": "number", "minimum":
↪  -5,"maximum": 5},
            "y": {"type": "number", "minimum":
↪  -5,"maximum": 5}},
        "forms": [{"href": "http://rb/move"}]}}}}
```

Listing 1: This is an example of a Thing Description (TD) for a robot arm with two Degree of Freedom (DoF) and is controlled using Inverse Kinematics (IK). The robot exposes one property affordance called `"cartesianPosition"` (Lines 5-11) and one action affordance called `"moveToCartesianPos"` (Line 12-19). Reading the property would yield an object containing the x and y coordinates of the end-effector, each being a number between -5 and 5. `"moveToCartesianPos"` expects an input with the same format.

standards is the Thing Description (TD), a JSON-Linked Data (JSON-LD) document that is both human- and machine-readable and that describes the API provided by any Thing regardless of which application layer protocol it uses for its communication.

Problem Statement: The W3C TD enables the amalgamation of robotics and Web technologies, simplifying the process a developer needs to develop a working solution by providing all the information needed to communicate with a robot that exposes its services as a Web API and provides a JSON-Schema-based solution for describing input and output payloads. However, robots provide a unique challenge when it comes to describing their capabilities using the TD, as the JSON-Schema-based approach is not expressive enough to describe the limitations on the input and output values for controlling a robot using Cartesian coordinates and Inverse Kinematics (IK). Furthermore, there is a lack of simulation and testing frameworks that target Web-enabled robots and robot arms without a cloud connection. Spinning up such simulations is currently manual and arduous, but it is essential for testing robotic applications with regard to their correctness and safety.

Contributions: In this paper, we aim to address the shortcomings mentioned above by introducing **RobWoT**, a set of methods and an open-source framework that is capable of generating a fully functioning DT that can run in real-time or faster than real-time and can be used to calculate the workspace of a robot arm in a scene and generate an STL file describing the workspace. Furthermore, the DT exposes the same functionality as the actual device and can be used for development to verify the correctness of the Cartesian inputs and verify possible collisions before aggregating the same command to the actual device, ensuring the correct behavior of the device and mitigating any collisions. In particular, we perform the following contributions:

- Introducing a method that takes the URDF of any arbitrary robot arm as an input and generates a Web-based DT of it, including its TD and that can be controlled using both Forward Kinematics (FK) and IK in Section III-A1.
- Introducing a method for generating and annotating the workspace of a robot arm in the TD automatically based on the environment in which the DT resides, including the minimum and maximum values for Cartesian coordinates and joints and both a point cloud and an STL representation of the workspace in Section III-A1 and Section III-A2.
- Proposing a simulation-in-the-loop approach for validating the input values for FK and IK control and evaluating the feasibility in Section III-A3.

The rest of the paper is structured as follows: Section II introduces the state of the art, and in Section IV we evaluate our approach. Section V discusses the limitations of our approach and Section VI gives an overview of related work. Section VII concludes this paper.

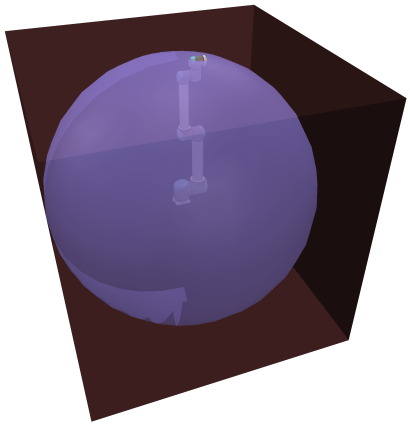


Fig. 2: Describing the workspace of a robot arm is a non-trivial task. We look at the UR10 robot as an example. With six Degree of Freedom (DoF), this robot can reach any point in a sphere with a diameter of 2.6m around its base joint, shown in purple. In the Thing Description (TD) of the robot, however, it is only possible to specify the lower and upper bounds of the volume in each axis. This description always spans a cuboid volume larger than the robot's workspace, shown here in brown. Furthermore, we are considering the simple case of the robot in an empty environment with no other objects. Describing a robot's workspace based on the surrounding environment is even more complex and harder to describe mathematically.

II. State of the Art

A. The Web of Things (WoT) and the Thing Description (TD)

The W3C introduced the Web of Things (WoT) [4] as an application-layer solution to the high fragmentation that the IoT landscape is currently suffering from. It consists of several standards built on top of well-established Web standards. The core of these standards is the Thing Description (TD) [5], which defines a description format in the form of a JSON-Linked Data (JSON-LD) document that is both human- and

```xml
<robot name="1 DoF Robot">
    <link name="base"></link>
    <link name="link"></link>
    <joint name="joint" type="revolute">
        <limit lower="-3.14" upper="3.14"/>
        <parent link="base"/> <child link="link"/>
    </joint>
</robot>
```

Listing 2: This is an example of a simple Unified Robotic Description Format (URDF) file describing a one Degree of Freedom (DoF) robot. The URDF describes robots using a set of `<link>` elements (Lines 2-3) connected using `<joint>` elements (Lines 4-7) to form a tree-like structure. The URDF can also describe the structure, visual, collision geometry, kinematics, and dynamics of robots, but this was omitted here for brevity.

```
solid example

facet normal 0.0e0 0.0e0 1.0e0
    outer loop
        vertex 1.0e0 0.0e0 0.0e0
        vertex 0.0e0 1.0e0 0.0e0
        vertex 0.0e0 0.0e0 0.0e0
    endloop
endfacet

endsolid example
```

Listing 3: An STL file describes the surface of any three-dimensional object in the form of a triangulated surface. The file lists all the triangles of the triangulated surface using their vertexes and the normal of the surface pointing outwards from the object.

machine-readable. The TD describes the API exposed by a Thing and any metadata needed for interacting with a Thing. The TD abstracts all possible interactions into three types of interaction affordances:

- **Property Affordances**, which map to any variable or states exposed by a Thing. Related operations are reading, writing, and observing. For robot arms, joint positions are the primary properties to expose.
- **Action Affordances**, which map to actions that the Thing can perform. Action affordances are invoked. Moving a robot arm to a certain Cartesian position can be modeled as an action affordance.
- **Event Affordances** represent any events or notifications that a Thing can emit. Related operations are subscribing and unsubscribing. In the case of a robot arm, a possible event affordance can be collision detection.

Things that expose TDs are called Producers, while those that consume TDs to start interacting with Producers are called Consumers. Consumers can perform operations by sending a certain request to the producer, who may send a response as a result. The request and the response may include input and output payloads, respectively, which the TD describes using a subset of JSON Schema. This subset is capable of describing data of types `"null"`, `"boolean"`, `"integer"`, `"number"`, `"string"`, `"array"`, and `"object"` and constraints that apply to each type such as minimum and maximum values for `"integer"` and `"number"`, patterns for `"string"` and schema for items and properties inside an `"array"` and `"object"`, respectively. For robot arms, joint and Cartesian positions are usually modeled as `"number"` and constrained by giving the minimum and maximum values of each joint position or each dimension in the case of the Cartesian position. In the case of the Cartesian position, the volume spanned by the boundaries defined in the JSON Schema is usually bigger than the actual workspace volume, especially if the robot uses one or more revolute joints. Describing the actual workspace to validate that a Cartesian coordinate resides in a robot arm's workspace is a challenge that is currently not possible to perform using the TD. This limitation can be visually explained in Figure 2.

B. Unified Robotic Description Format (URDF)

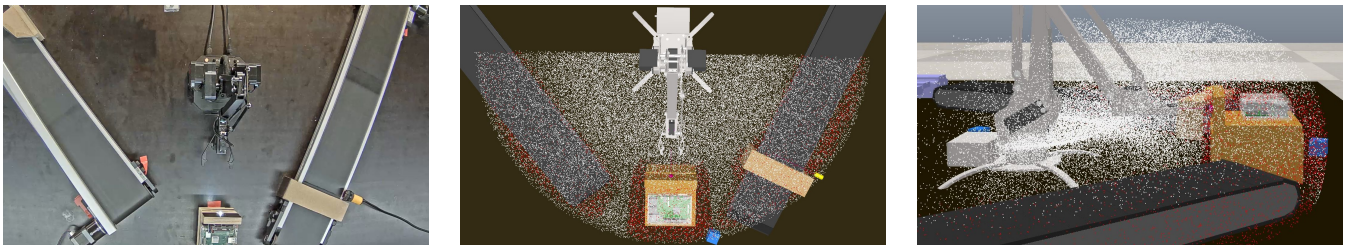
The Unified Robotic Description Format (URDF) [6] is a human- and machine-readable XML format for describing the structure, visuals, collision geometry, kinematics, and dynamics of robots. It is used mainly in conjunction with Robot Operating System (ROS), a software development kit for robotic applications, as an exchange format for robot models. The URDF represents robots using a tree structure containing a series of links joined together with a series of joints. For each `<link>` it is possible to describte its `<intertial>`, `<collision>` and `<visual>` properties. A `<joint>` element must specify its type as revolute (one DoF rotating around joint axis with limits), continuous (one DoF rotating around joint axis without limits), prismatic (one DoF sliding parallel to joint axis), fixed (zero DoF), floating (six DoF rotating and sliding around all axes) or planar (two DoF in a plane perpendicular to joint axis). Furthermore a `<joint>` must specify the `<parent>` and `<child>` link and additional optional data such as `<limit>` and `<dynamics>` properties.
The URDF cannot describe all types of robots, such as parallel robots or robots that are not using rigid bodies as links. Additionally, the URDF does not describe if a joint is controllable.

C. STL

The STL file format is a text format used to describe 3D models as an unstructured triangulated surface. The file consists of a list of all triangles that make up the surface by listing the vertexes of each triangle, with the surface normal pointing outwards. STL can be used to describe the surface of point clouds.

D. Robotics Simulations

Robotics simulation frameworks are development kits that provide the tools needed to build robot models and scenes in which they reside, provide physics engines for kinematic and dynamics calculations, and the algorithms used to perform high-level functionality such as IK calculations and robot model importing. Usually, such tools provide a Graphical User Interface (GUI) environment that helps set up simulation scenes visually. To run a simulation, a calculation time step Δt_{calc} needs to be specified, which is then used by the physics engine to incrementally solve the differential equations needed to calculate the positions of all moving objects inside a scene. Additionally, the simulation time step Δt_{sim} specifies how



(a) Real Uarm Setup **(b)** Workspace point cloud in DT - Top-view **(c)** Workspace point cloud in DT - Side-view

Fig. 3: Figure (a) shows a real setup of the Uarm robot and two conveyor belts. A DT of this scene was constructed to calculate the workspace available for the Uarm in this setup. The labeled point cloud representing the workspace is shown in Figures (b) and (c), with white points denoting accessible positions and red points representing inaccessible positions

often the simulation scripts are called and how often the rendering is updated.

III. RobWoT Framework

To solve the shortcomings as mentioned above, we propose **Rob**otics **W**eb **o**f **T**hings **(RobWoT)**, a method and a framework that takes a URDF as an input and automatically generates the corresponding WoT-enabled DT including its TD annotated with the correct constraints based on its joint constraints and the available workspace in the simulated scene. Additionally, we aim to provide a solution to the IK constraints by linking the STL file modeling the workspace to the TD and investigating a simulation-in-the-loop approach for validating FK and IK constraints. Robotics Simulation frameworks offer the tools needed to model robot DTs for Web-enabled robots. However, currently, the process of modeling such a DT remains a manual endeavor that requires setting up the simulation scene, importing the robot model, and, then manually exposing the FK and IK functionalities of the robot using a Web server using the desired web protocol. Furthermore, the resulting API would need to be documented in a way that explains the inputs and outputs of each interaction, which is a challenge to do in a declarative manner as explained in Section II-A.

A. Method

*1) **Automated DT Generation**:* The process of generating the DT from the URDF of a robot can be laid out into the following steps:

1) The process starts by importing the URDF in the simulation framework so it can be parsed. The robot's joints are counted, and their types are determined (please refer to Section II-B).
2) Once the importing is finished, the robot's DT performs N_{move} random movements in the simulation scene to determine its effective workspace based on its joint constraints and the other collidable objects inside the simulation scene. This generates a labeled point cloud that denotes the robot's end effector position and whether a collision was detected, as shown in Figure 3b and Figure 3c. The labeled point cloud is stored as a CSV file.
3) After generating the point cloud, a convex polyhedron is generated around the non-collision points using the α-shape method. The resulting polyhedron is stored as an STL file.
4) The information gathered in the first step is used to automatically generate a WoT-compliant server that exposes FK and IK functions of the robot provided by the simulation framework. The DT exposes this API using a TD which includes two property affordances called `"getJointPosition"` and `"getCartesianPosition"`, and two action affordances called `"moveToJointPosition"` and `"moveToCartesianPosition"`. The TD also has links to the point cloud CSV file, the STL file representing the workspace, and the URDF file of the robot.
5) The coordinate system of the DT and the real robot are usually not calibrated and would show a different Cartesian position for the end-effector. However, a linear transformation can transform the virtual coordinate into the real coordinate. Assuming that $\mathbf{x}_{\text{real}}$ denotes a vector in the real coordinate system and $\mathbf{x}_{\text{virtual}}$ denotes a vector in the virtual coordinate system, the transformation matrix can be described as follows:

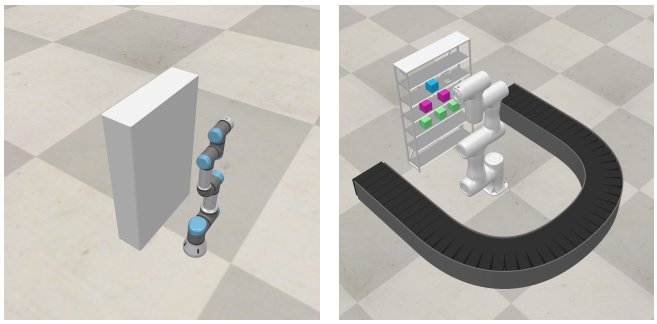
$$\mathbf{x}_{\text{real}} = s \cdot \mathbf{R} \cdot \mathbf{x}_{\text{virtual}} + \mathbf{t} \tag{1}$$

with $s \in \mathbb{R}$ a scaling factor, $\mathbf{t} \in \mathbb{R}^3$ the translation vector and $\mathbf{R} \in \mathbb{R}^{3\times3}$ the rotation matrix. Using four samples, it is possible to calculate both $\mathbf{R}$ and $\mathbf{t}$ with the Kabsch-Umeyama algorithm [7], while the scaling factor s is determined by unit conversion.

This correction ensures that the input and output values of the DT and the real robot remain the same.

*2) **Validating IK inputs using STL file and Point Cloud**:* By providing the robot's workspace as an STL file, any consumer can use the file to validate if a certain Cartesian point is inside the workspace polyhedron. However, constructing the STL file using the α-shape method may fail for some workspaces. In these cases, we propose the following method to check if a certain position is accessible or not:

1) Define a percentage threshold.
2) Find and build a set of the N-nearest point to the investigated position



(a) First scene with the UR3 robot **(b)** Second scene with the CR3 robot

Fig. 4: These figures show two robots and two scenes used in our generation time evaluation. Figure (a) shows the simplest scene with the UR3 robot, and Figure (b) shows the CR3 in the second scene. The other robots and scenes are omitted for brevity but can be viewed in our repository.

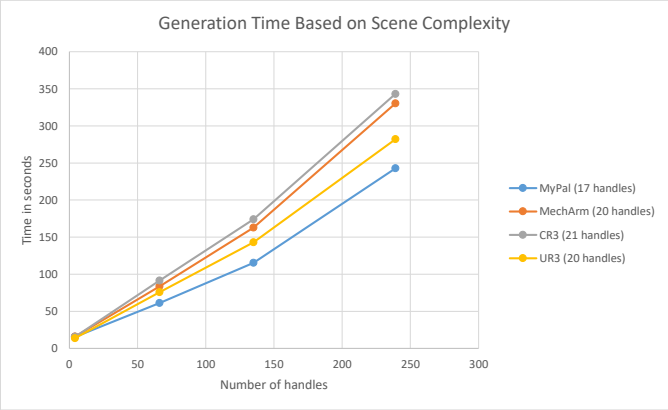


Fig. 5: We evaluated the time needed to generate the Web server and the annotated TD for four robots and in four scenes with increasing complexity. The time increases linearly with the number of handles inside a scene. An object in Coppeliasim is usually made up of multiple handles, such as the links and joints of a robot. The generation time also depends on the complexity of the robot arm itself, its DoF, and geometry.

3) Check if the percentage of inaccessible points in this set is higher than the defined threshold; if so, then the point is inaccessible

*3) **Validating FK and IK inputs using simulation-in-the-loop**:* Having a DT of a real robot and its environment facilitates a different and novel approach for validating that a robot's movement does not result in a collision with other objects in the environment. The advantage of this approach is that it can take changes in the environment into account and validate accordingly, which is impossible using the static STL file generated at the beginning of the simulation. This is done by first sending the movement command to the DT, which tries to perform it and check for any collisions. The same command is relayed to the device if no collision is detected.

B. Implementation

We implemented our method as an open-source solution[1] using Coppeliasim [8] as a simulation framework and the node-wot library[2] to handle generating the WoT server for the DT. Coppeliasim is an extensive robotics simulation application that provides all the tools for importing URDF files, calculating FK and IK, and simulating robotics using different physics engines. Additionally, it offers a WebSockets remote API option that allows any JavaScript/Typescript application to communicate with it, giving access to all functions and libraries inside it. node-wot is the reference implementation of the W3C Scripting API written in JavaScript/Typescript. It is meant to facilitate the development of WoT Consumers and Producers application logic without the need to be concerned about setting up the underlying communication protocol. Our implementation provides simple Typescripts libraries that can automatically generate the DT including its WoT-compliant Web server and sample Consumers, also written using node-wot, that can interact with the DT. However, our method can be used with any simulation framework and any programming language, as long as there is a way to generate a Web server capable of controlling and actuating the simulation and the DT, respectively.

[1] Our implementation and evaluation data are available through this link and will be made public upon the acceptance of the paper: https://github.com/tum-esi/RobWoT.git

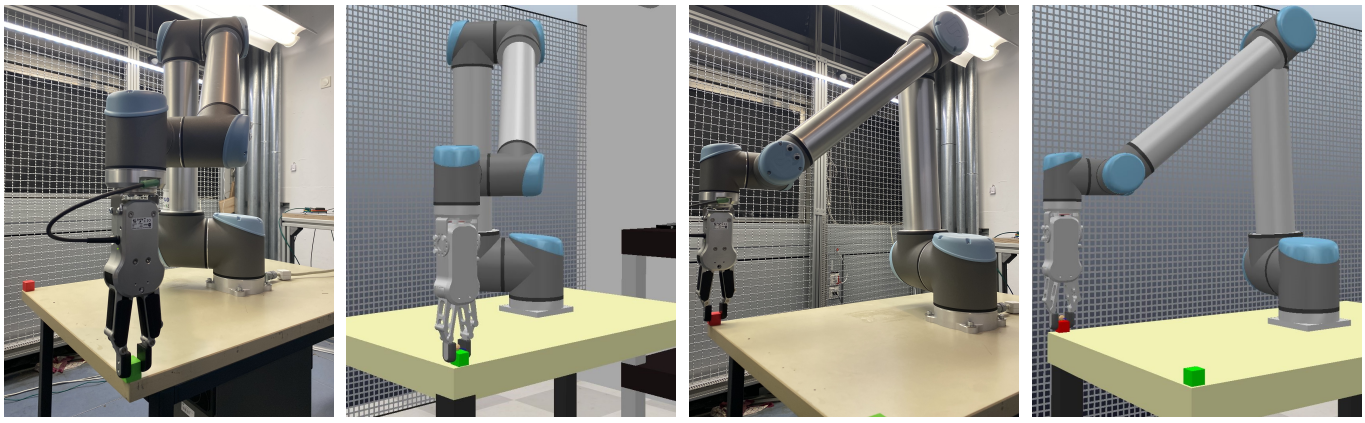
[2] https://github.com/eclipse-thingweb/node-wot

IV. Evaluation

To evaluate our approach, we perform two experiments. The first experiment evaluates the time needed to generate a Web server and its annotated TD using our approach. By contrast, the second experiment evaluates the accuracy of the calibrated DT movement using IK compared to its physical robot. All evaluations were performed on a computer with 11th Gen Intel Core i5-1135G7 Processor running at 2.40GHZ, an NVIDIA GeForce MX450 GPU, 16GB of RAM, and 512GB of SSD NVME WDC SN730 storage. The evaluations were run on a Windows 10 operating system, using Coppeliasim version 4.4.0 and node-wot version 0.8.5.

A. Evaluating Generation Time

To evaluate the generation time of our approach, we conducted the evaluation using four robot models. Each robot was put in 4 scenes of incremental complexity. The first scene contains four collidable handles, the second 66, the third 135, and the fourth 239 handles. An object can consist of multiple handles, and each one is considered when checking for a collision. The generation process was performed three times for each robot in each one of the four scenes, and then the average was calculated. The evaluation does not take the time for importing the URDF of each robot into account, as it is negligible compared to the execution time of our algorithm for generating the workspace. Figure 5 shows the results of the evaluation. The time needed for generating the Web server and the TD scales linearly with the increase of handles in the simulation scene. It also depends on the complexity of the robot arm itself, its DoF, and geometry. The time needed remains in a couple of minutes for the most complex scenes. It is faster than any manual approach for setting up the DT, calculating the workspace, and writing the corresponding TD.



(a) Real UR10 robot grabbing green cube **(b)** DT of UR10 robot grabbing green cube **(c)** Real UR10 robot grabbing red cube **(d)** DT of UR10 robot grabbing red cube

Fig. 6: We perform our IK for the calibrated DT using an industrial-grade UR10 robot. We manually moved the robot arm to a certain coordinate and recorded these coordinates. We, then, invoked the `"moveToCartesianPosition"` of the DT using the coordinates in the previous step. After the DT performs the movement, we read the joint positions using the `"getJointPosition"` property affordance, which we propagate back to the physical robot and compare with the result position with the recorded position. Two positions in which the robot is aiming to grab a 2.5 cm cube are shown here. This illustrates the accuracy of the DT.

B. Evaluating IK of calibrated DT

We evaluate the accuracy of the IK control of our a calibrated DT to check if the auto-generated and calibrated DT is capable of representing the physical robot movements with minimal error. To perform this evaluation, we used an industrial-grade UR10 robot arm. We manually moved the robot arm to a certain coordinate and recorded these coordinates. We, then, invoked the `"moveToCartesianPosition"` of the DT using the coordinates in the previous step. After the DT performs the movement, we read the joint positions using the `"getJointPosition"` property affordance, which we propagate back to the physical robot and compare with the result position with the recorded position. We have done this procedure 14 times and calculated the distance error for each position. A couple of the positions are shown in Figure 6, and our measurement results can be viewed in Figure 7. The average offset is approximately 6.8 mm, with the highest error being around 14.7 mm. This level of accuracy gives us confidence that our approach is viable in industrial scenarios and is high enough to handle centimeter-precise tasks such as grabbing 2.5 cm cubes as seen in Figure 6.

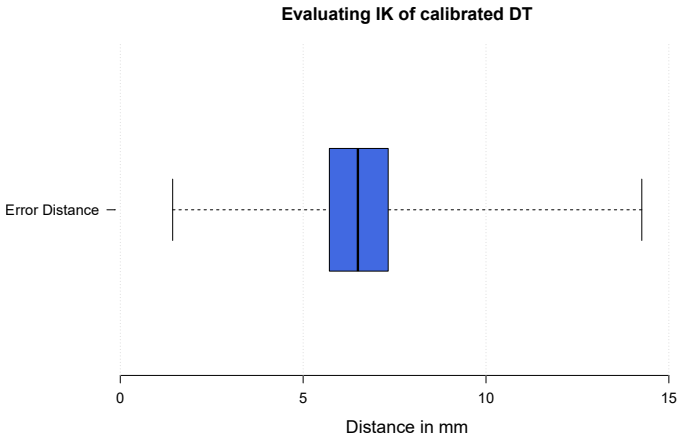


Fig. 7: This Figure illustrates the distribution of the data gathered in the experiment described in Section IV-B. The average offset from the recorded position is approximately 6.8 mm. The lowest offset is 1.4 mm, and the highest is 14.2 mm

V. Current Limitations and Future Work

Our method assumes that all joints described in the URDF are controllable. However, this is not the case for some robots that rely on auxiliary joints to function properly, such as the Uarm and the Dobot Magician. Such robots require additional manual steps to correctly calculate their IK.

Furthermore, our method is as accurate as the simulation scene in which the robot is imported. Currently, setting up the actual simulation scene remains a manual task that must be done before importing a robot's URDF. This can result in inaccuracies in the generated workspace, leading to faulty behavior. The process is also arduous and time-consuming. Our future work aims to use a scene description format to describe a whole environment, including all the robots inside it, which is generated using lidar point clouds and image recognition techniques.

VI. Related Work

Web-enabled robotics and web-enabled DT has been gaining increasing attention in the literature.

[9] proposes a novel approach that leverages DTs and integrates them with ontology and cloud computing technologies intending to provide effortless integration, configuration, and monitoring of industrial robotic systems. The DT of the robots is implemented as an HTML5 application, built on top of Babylon.js, which uses WebGL for browser-side rendering.

The models of the robots are fetched from the cloud in GLFT format automatically.

[10] introduced a novel DT platform for a human-in-the-loop control of robotics using VR technology. Human control of the DT is propagated to the actual robot arm using very low latency.

[11] manually built a cyber-physical system consisting of a 4-DoF robot arm controlled using an ESP32 module, which connects to the Web server using WebSockets. The Web server exposes a WebGL-based DT of the robot arm that can be viewed and controlled in the browser. This DT is mainly developed for human interaction.

[12] investigated the process of developing Web DTs, called WDTs, in the paper and demonstrated this concept by manually developing a web-based DT for a mini-scale assembly line. The virtual model of WDT was manually built using Unity and then exported to WebGL. The WDT can be controlled using a physical PLC. To connect the PLC to the WDT, a Node.js proxy was developed, which acts as an OPC UA Client and a WebSocket server. The Node.js proxy communicates with the PLC over OPC UA and relays messages over Websocket to the WDT of the assembly line. According to the paper, this setup can help students learn manufacturing without costly engineering tools, facilitate research and enable web-based simulation for the industry.

[13] proposed DTs to simulate human behavior and Autonomous Mobile Robots (AMRs). The DT is built on top of the Robot Operating System (ROS) framework. In contrast, the simulation uses Unity for human entities and the Gazebo robotics simulator for Autonomous Mobile Robots (AMRs). Communication between these components happens over MQTT.

[14] proposed a method for automatically generating a DT modeled using the TD and mimicking the behavior of existing Web Things using the Markov Decision Process. The DT can be used to simulate the behavior of a Thing. The method works with the assumption that Things only contain discrete properties, limiting the universal applicability of this method.

Our approach aims to facilitate the development of Web of Robotic Things systems and minimize the manual work needed to set up simulations and DTs for them. Our approach would eliminate the manual work needed to set up the DTs in [9]–[13]. Furthermore, our approach does not enforce a certain communication protocol for communicating with the DT and does not rely on specific frameworks. Our method aims to generate TDs and DT for robots, leveraging domain-specific information and tools for the generation process. This focus on the robotics domain allows us to generate TDs and DTs curated specifically for robotics, which are more expressive and accurate as opposed to a more general approach proposed by [14].

VII. Conclusion

In this paper, we presented **RobWoT**, a method and simulation framework for automatically generating the WoT-enabled DT of a robot and its corresponding TD using only the URDF of a robot as an input. Our method requires minimal human intervention and aims at generating TDs for robots that can describe the actual workspace available for the robots, which is not possible using current TD mechanisms. We propose methods for describing a robot's workspace using an STL file. For complex workspaces that cannot be constructed accurately, we proposed a simulation-in-the-loop approach for validating whether a Cartesian point is inside the workspace. We evaluated the time required to generate WoT-enabled DT and TD and the accuracy of our DT IK compared to a real robot. Our evaluation shows that our approach is highly scalable and that, on average, the distance error between the virtual and real end-effector is only 6.8 mm, and is, therefore, viable for industrial applications.

References

[1] I. H. Khan and M. Javaid, "Role of Internet of Things (IoT) in Adoption of Industry 4.0," *Journal of Industrial Integration and Management*, vol. 07, no. 04, 2022. [Online]. Available: https://doi.org/10.1142/S2424862221500068

[2] I. Research, "Size of the market for industrial robots worldwide from 2018 to 2020, with a forecast through 2028 (in billion U.S. dollars) [Graph]," October 25 2021. [Online]. Available: https://www.statista.com/statistics/728530/industrial-robot-market-size-worldwide/

[3] A. Kamilaris and N. Botteghi, "The Penetration of Internet of Things in Robotics: Towards a Web of Robotic Things," *J. Ambient Intell. Smart Environ.*, vol. 12, no. 6, January 2020. [Online]. Available: https://doi.org/10.3233/AIS-200582

[4] Michael Lagally, Ryuichi Matsukura, Michael McCool, Kunihiko Toumura, Kazuo Kajimoto, Toru Kawaguchi, Matthias Kovatsch, "Web of Things (WoT) Architecture 1.1," https://www.w3.org/TR/2023/PR-wot-architecture11-20230711/, accessed: July 13, 2023.

[5] Sebastian Kaebisch, Michael McCool, Ege Korkan, Takuki Kamiya, Victor Charpenay, Matthias Kovatsch, "Web of Things (WoT) Thing Description 1.1," https://www.w3.org/TR/2023/PR-wot-thing-description11-20230711/, accessed: July 13, 2023.

[6] Dave Coleman, Hirotaka Yamada, "urdf/XML," http://wiki.ros.org/urdf/XML, accessed: July 13, 2023.

[7] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, 1991.

[8] E. Rohmer, S. P. N. Singh, and M. Freese, "CoppeliaSim (formerly V-REP): a Versatile and Scalable Robot Simulation Framework," in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013, www.coppeliarobotics.com.

[9] T. Hoebert, W. Lepuschitz, E. List, and M. Merdan, "Cloud-Based Digital Twin for Industrial Robotics," in *Industrial Applications of Holonic and Multi-Agent Systems*, V. Mařík, P. Kadera, G. Rzevski, A. Zoitl, G. Anderst-Kotsis, A. M. Tjoa, and I. Khalil, Eds. Cham: Springer International Publishing, 2019.

[10] I. A. Tsokalo, D. Kuss, I. Kharabet, F. H. P. Fitzek, and M. Reisslein, "Remote Robot Control with Human-in-the-Loop over Long Distances Using Digital Twins," in *2019 IEEE GLOBECOM*, 2019.

[11] S. Khrueangsakun, S. Nuratch, and P. Boonpramuk, "Design and Development of Cyber Physical System for Real-Time Web-based Visualization and Control of Robot Arm," in *2020 ICCRE*, April 2020.

[12] S. Konstantinov, F. Assad, W. Azam, D. Vera, B. Ahmad, and R. Harrison, "Developing Web-based Digital Twin of Assembly Lines for Industrial Cyber-physical Systems," in *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*, May 2021.

[13] Y. Fukushima, Y. Asai, S. Aoki, T. Yonezawa, and N. Kawaguchi, "DigiMobot: Digital Twin for Human-Robot Collaboration in Indoor Environments," in *2021 IEEE Intelligent Vehicles Symposium (IV)*, 2021.

[14] L. Sciullo, A. Trotta, F. Montori, L. Bononi, and M. Di Felice, "WoTwins: Automatic Digital Twin Generator for the Web of Things," in *2022 IEEE WoWMoM*, 2022.