Professorship of Embedded Systems and Internet of Things
Department of Computer Engineering
Technical University of Munich

Development and Evaluation of a Web of Thing-compliant Robot-Arm Simulation Framework

Zucheng Han

Master's Thesis

Development and Evaluation of a Web of Thing-compliant Robot-Arm Simulation Framework

Master's Thesis

Supervised by Prof. Dr. phil. nat. Sebastian Steinhorst
Professorship of Embedded Systems and Internet of Things
Department of Computer Engineering
Technical University of Munich

Advisor Fady Salama

Co-Advisor

Author Zucheng Han
Schleißheimerstr. 323
80809 München

Submitted on May 23, 2023

Declaration of Authorship

I, Zucheng Han, declare that this thesis titled "Development and Evaluation of a Web of Thing-compliant Robot-Arm Simulation Framework" and the work presented in it are my own unaided work, and that I have acknowledged all direct or indirect sources as references.
This thesis was not previously presented to another examination board and has not been published.

Signed:

Date:

Abstract

Internet of Things (IoT) technology has been widely implemented in many embedded devices and fields. In recent years, in the challenge of integrating booming IoT devices, researchers and scientists always tends to design a cross-platform IoT, which could manage all devices via same communication protocol. As the complement of the existed IoT standards, Web of Things (WoT) aims at merging valuable web protocols and tools into same framework.

For the target of describing the devices with WoT architecture, WoT architecture drafters tend to form a legible introduction called Thing Description(TD). This TD document records the web interface of device and its content is constituted of the static human readable information, which shows high potentiality to accurately describe most physical behaviors of IoT devices.

The data verification mechanism of TD document highly dependents on its JSON schema. At present, it can not clearly represent restrictions of the robotic workspace with the complex spatial shapes. Furthermore, for the robot with similar functionalities and structures, a method to automatically generate low-level TD document can obviously reduce the time in repeated device configurations.

This thesis tends to propose the common method to automatically generate Thing Description based on Robotic simulation framework. This robotic TD document could indicate the kinematics properties of robotic arm and contain the essential metadata about the robotic workspace and digital twin. We also design a method to automatically implement inverse kinematics conversion, which is suitable for both real robot and its digital twin. Besides, in the simulation framework, we successfully build two digital twin scenes about IoT platform. These two virtual scenes with full functionality and same response-ability as real scene can be used in research and teaching in the future.

Compare with other research on automatic TD generation, we successfully implement automatic TD generation for robots. Our robotic TD could accurately represent the robotic work circumstances and its properties, which is applicable to 95% of the robots on the market. Furthermore, our automatic TD generation approach is appropriate for 80% robots with similar structure on the market. In practice, users can use our TD document as benchmark. On basis of our TD action affordance, they can freely design more interaction affordances about robotic complex movements in their TD document.

Contents

1

Introduction

1.1 Motivation

In the industrial production scenarios, it usually involves the participation of robots. Manually filling TD document by engineers to these robots with similar structure will be complicated and tedious work. The user-defined TD document lacking common rules determined by designers is a major challenge for other developers to maintenance and modify.

Each robots have their own workspace at their work station. These workspace is normally represented as restriction values in TD document. In the related TD document, it is only exhibited by some numbers and units, which brings reading and comprehension difficulties for users. Besides, evaluating the restrictive values of robotic workspace by directly perform robotic motions at current work environment gonna be a impossible mission in many cases.

Therefore, we want to design a mechanism to automatically generate robotic TD document based on digital twin of robot. This robotic document have ability to exhibit the situation of real robot and it also includes the kinematics features of robot. The metadata including robotic digital twin in the TD document can be easily used to reproduce the robotic workspace and analyze collision state of robot in current environment.

1.2 Problem Statement

As a readable simultaneous document for both human and machine, the W3C Thing Description(TD) shows how the devices or entities are connected to the web. TD document also introduces the specifications how can we make interaction with devices and the required security level for this operation, which brings higher priority as the approach to describe real industrial devices compared with other candidates.

Just obeying on the basic rule of TD document, theoretically each user could freely design TD document to describe their devices. However, for some devices with similar utility such as lights, some types of sensors and robotic arm, a appealing TD document, which has ability to represent all features of this device and owns scalability and portability to same type devices, always will be popular for developers. The exploration to design structure of this TD document for robot will be a research problem.

Normally the TD document covers the restrictive value of device interactions. These value setting is highly relied on the current working environment of devices and it is rather difficult confirmed by execute device operations at real environment in some cases. To handle this situation, restrictive value estimation by the simulation of device behavior at digital twin would be considered as acceptable choice. Then how to generate a digital twin devices or scenes which can mimic the real device interactions also will be a problem.

For the above problems, we first want to discuss what kind of robotic TD document structure, which is capable of describing the complete kinematics characteristics of the robotic arm in the current industrial environment. And we also investigate what type of metadata does the TD document needs to include for the aim of demonstrating robotic workspace. Then we want to record our exploration and attempts for the problem of building digital twin scenes with full functionality and same reaction ability as real scenes.

We summarize our research problems as follows.

- The structure of TD to clearly describe robotic kinematics and workspace
- The method to realize the digital twin entity response ability same as the real entity

1.3 Contributions

In this thesis, we propose a category of TD document architecture clearly reflecting the features of the robot. Then we do some exploration and works on the automatic inverse kinematics conversion deployment and robotic TD generation based on digital twin of robotic work environment. Finally, during entire period of thesis, we have successfully built several digital twins based on the real IoT platform. These digital twins express the same complete functionality and response ability as the real platform.

In detailed practice, we proffer the following contributions:

- A kind of robotic TD structure to contain robotic kinematics parameters and related metadata for robotic workspace recover.
- Verify the possibility of integrating Web of Things library with robot simulation tool Coppeliasim.

- The method to implement dynamics parse for robotic digital twin at Coppeliasim based on URDF file.
- Implementation of automatic inverse kinematics conversion for robotic digital twin at Coppeliasim.
- Implementation of automatic metadata generation about robotic workspace at Coppeliasim.
- The approach to realize automatic robotic TD document generation based on digital twin of robotic work circumstances.
- Some fully functional and same reactive digital twin of the real IoT platform.

1.4 Structure of The Thesis

In the thesis, it is divided into several chapters. The second chapter State of the Art (SotA) introduces some basic concepts and current developments about technology of WoT architecture and digital twin. The next chapter introduces some algorithms and tools, which are used for solving our research problem.

Then in the approach part, it includes the details of our robotic TD document structure and the method of automatically generating robotic TD document. For the chapter related work, it make an introduction to some papers and research progress about automatic TD generation based on other metadata. After that, in the chapter of evaluation, We evaluate the stability of our digital twin scenes and the accuracy of the workspace represented by the TD file. We also prove our automatic inverse kinematics conversion is suitable for both real robot and virtual digital twin at the same.

At the end of thesis, it takes a discuss to the restrictions and application scope of our automatic TD generation approach. References and appendix with code and are attached after the conclusion.

2

State of the Art

2.1 Web of Things (WoT) Architecture

In recent years, a large amount of embedded devices participate in our daily life via Internet of Things (IoT) technology. With the popularity of IoT devices, some company and institutes want to draft a uniform standard protocol for the integration and management of all IoT devices[1]. A noteworthy research direction is to treat IoT devices as Web of Things (WoT) which support open web standards such HTTP or MQTT protocol[2].

In the WoT architecture, some widely acknowledged web technologies and software such as Javascript, Ajax, PHP can help us to minimize the development time on initializing protocol layers of WoT devices[3]. These devices supporting WoT architecture can be accessed by some web mechanisms such as browsing and linking.

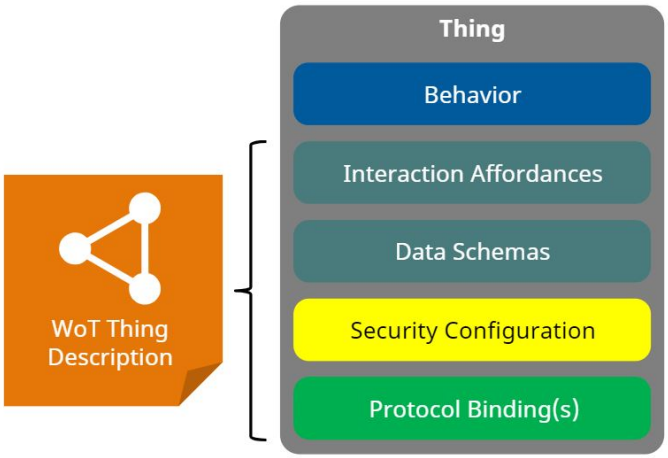


Figure 1: Architectural Aspects of a Thing Description[4]

For the purpose to record metadata and interactions method with devices, WoT researchers design a format called Thing Description (TD). In default situation, TD document is encoded as JSON format. From the perspective of available resources by TD document, TD document is composed of three interaction affordances: Properties, Actions and Events. And each affordances includes its own security authentication method and protocol bindings. In other words, the TD document describes the WoT architecture of current device.

```
{
    "@context": [
        "https://www.w3.org/2019/wot/td/v1",
        {    "@language": "en"    }
    ],
    "@type": "Thing",
    "title": "PanTilt1",
    "id": "virtualPantilt-1",
    "description": "virtual PanTilt 1 in Coppeliasim",
    "security": "nosec_sc",
    "securityDefinitions": {
        "nosec_sc": {   "scheme": "nosec"   }
    },
    "properties": {
        "panPosition": {
            ...
            "forms": [...]
        },
        ...
    },
    "actions":{
        "goTo": {
            ...
            "forms": [...]
        },
        ...
    },
    "events": {
        "detectColor": {
            ...
            "forms": [...]
        },
        ...
    }
}
```

Listing 1: The example of Thing Description

The interaction affordances in the TD are associated with the device behavior. In the specific

TD, we tend to use properties in the interaction afforadaces to record the device state and use actions to explain the physical behavior of devices such as motions and visualization. For the events part in the TD, after users subscribe event, the device will send message to the subscriber when the event is trigger in special scenario.

As a JSON format composed of fixed vocabulary term, the TD could be directly validated by the corresponding JSON schema. After we finish a TD document, JSON schema have ability to help us the test the correctness of our TD document. Furthermore, it exists NumberSchema in the TD JSON schema, which is mainly used for the validation of input or output value from the interaction affordances.

NumberSchema in TD data schema vocabulary terms[5]			
Vocabulary term	Description	Assignment	Type
minimum	specifies a minimum value and includes lower limit	optional	double
exclusiveMinimum	specifies a minimum value and excludes lower limit	optional	double
maximum	specifies a maximum value and includes upper limit	optional	double
exclusiveMaximum	specifies a maximum value and excludes upper limit	optional	double

In the thesis, in our robot TD, we have used NumberSchema to make sure the joint angle given by users is not beyond the physical restriction of robotic joints. But for special situation such as the workspace of robot represented in the Cartesian coordinates, NumberSchema could only comprehensibly explain the linear restriction. Therefore, on the basis of NumberSchema validation method, we use the link part of TD for validation as altenative method.

The WoT architecture provides a new innovative method for users to interact with devices in a same platform. The part of our main work in this thesis is to design a TD format based on WoT architecture and this TD document could precisely represent the features of robot.

2.2 The applications of Digital Twin

In the early stage of feasibility evaluation, from the perspective of cost reduction, algorithm verification, researchers are more inclined to simulate the behavior of the entities in the computer software[6, 7]. These simulated entities are usually treated as the digital twin.

As the digital representation of a pre-planned or actual product, system, and process in the real world, the digital twin technology has already been applied in industrial fields. In the factory, the digital twin mainly take participation in the production, prognostics, and health management (PHM) at the product lifecycle[8]. By simulating and visualizing digital twin of

product, engineers and developers can evaluate the performance of product and early warning of possible equipment failures at high effect.

The IoT technology is mainly applied to the low-power and performance restricted devices such as sensors, which generally have only a few states. It brings convenient for us construct the digital twin of these device via finite state machine (FSM) method. The main trend of digital twin application in IoT platform is multiagent and visualization[9]. Researchers take attempts such as abstract the layers and visualize the object containers to improve the response speed and stability of IoT system.

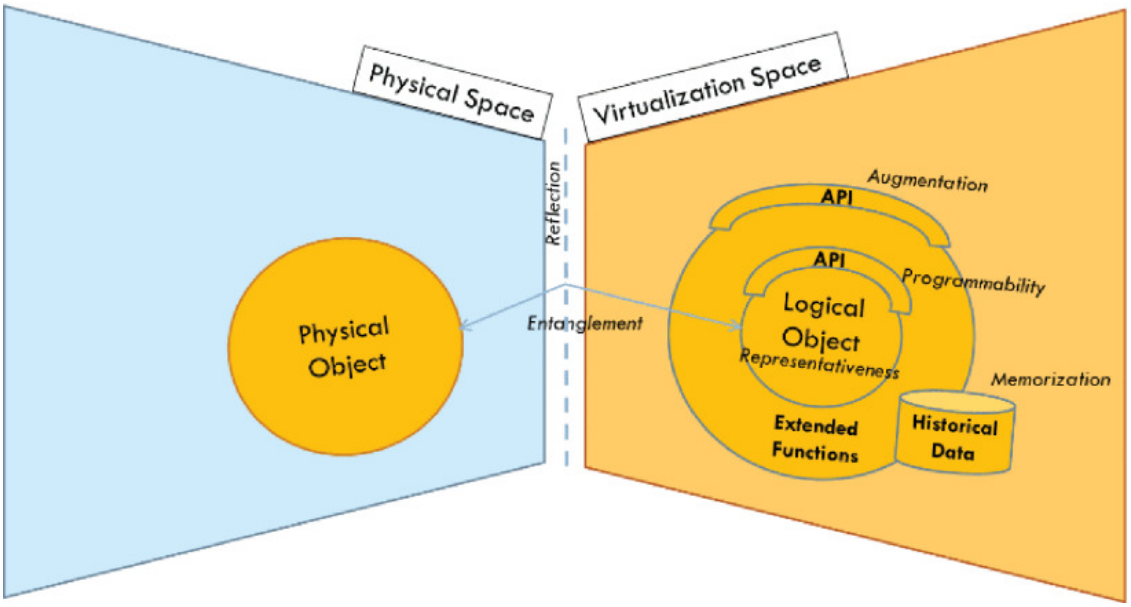


Figure 2: The architecture of digital twin in IoT device[9]. Researchers summarize some common features of the digital twin. The digital twin is placed in a virtual scene, researchers use the program script to simulate the mechanism of physical object. By means of API in the logical object, users do further development on the digital twin, which will generate same behavior as the physical object. And the historical data also is saved between the augmentation layer and the programmability layer.

In the thesis, our main research target are the different types of robotic arm. Compared with direct deploying in real robot with high damage risk, some algorithm in our thesis such as robotic workspace generation might be more suitable for the digital twin. And this algorithm is of vital importance for us to automatically generate robotic TD document. Therefore, we need to design the digital twin which contains the robot and its work environment and has same behavior as the real robot.

3

Algorithms and Tools

3.1 Algorithms

3.1.1 The generation of robotic accessible points in workspace

To estimate the extreme working limit range of a robotic arm, researchers and engineers often prefer to generate the workspace of robot. The prerequisite of building robotic workspace is to activate all modality of robot system. In the process of algorithm implementation, the robotic arm is expected to finish huge amount of motions and gestures in a short time. Therefore, we can only deploy this algorithm in the digital twin of robot.

A table recording all accessible position of robot and its collision state is very significant for us. Not only it is essential for the robotic workspace generation, but also it might helpful for us to estimate the position of obstacles in robotic workspace.

The key to generating points that accurately represent the robotic workspace is adjusting the angle of the robotic joints. There are two method for us to set the angle of robotic joints. The first method to set joint angle is random generation. As long as there are sufficient random points, the entire work area of robot can be covered accurately.

Another method is to divide the each joint limitation into several small areas. We assume the amount of areas is n, and normally the range of n is $10 < n < 200$. Then the regions from different joints with different angles are combined in sequence to generate the final joint angles of the robot.

Assume that the robot owns m joints. The main restrictions for the second method is that the algorithm complexity is $O(n^m)$. And it dose not exist any possibility to realize algorithm optimization. In most cases, we need to prepare more time for the second method and get the result without significant improvement.

After consideration, we finally use the first method. The structure of first algorithm is recorded as follows. For each accessible point, we also record the current collision state. This table also be recorded in the link part of our robotic TD. Users can also use this table to do further development or analysis.

Algorithm 1 The generation of robotic accessible points

for $point = 1, 2, \ldots, randomsteps$ **do**
    /* The randomsteps equals the amount of random points */
    for $i = 1, 2, \ldots, joint$ **do**
        /* Generate a random joint angle for each joint in the robot */
        /* The jointLimitLows, jointLimitRanges are limitation of robotic joints */
        $jointAngle = jointLimitLows[i] + math.random() * jointLimitRanges[i]$
        The corresponding digital twin joint move based on jointAngle
    end for
    Record the current collision state and position of the point
end for

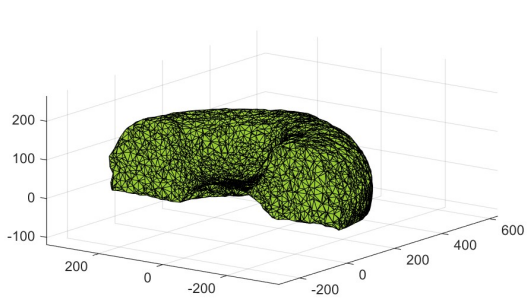
3.1.2 The reconstruction of robotic workspace based on the list of accessible points

The workspace of robot equals the area that the robotic end effector can reach. If we have plenty of reached points, which are evenly distributed in the workspace of robot, then we can take these points as point cloud and theoretically regenerate the robot's workspace by surface reconstruction method.

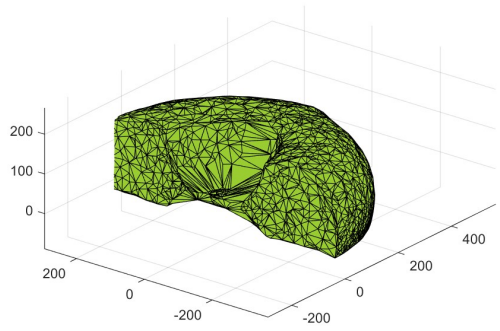
At present there are many method and code library, which support surface reconstruction method. But most of them requires points in point cloud have normals. So, we tend to a surface reconstruction method called alpha shape[10] without requirements of normals. By defining the bounding of generalized polygons via alpha shape value, this alpha shape method could generalize sets of points into polygons. And based on different alpha value, the point cloud consisted of accessible points will be regenerated into different shapes.

With the help of our accessible points generation algorithm in the last section, we have successfully obtained the reachable points of the Uarm robot in its working environment. So We try to use different alpha values to reconstruct the workspace of Uarm robot and take a quick evaluation of this algorithm performance.

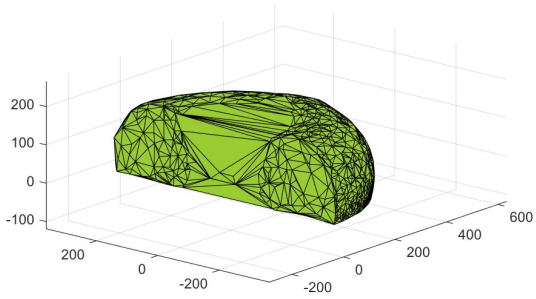
It can be seen from the following figures that the lower the alpha value, the more details is included in the generated workspace model, while the higher alpha value tends to generate a convex polyhedron as the workspace model. A restriction for this method is that we must manually modify the alpha value to generate the desired workspace shape we want. And too low alpha value might result in the inability to generate a complete model. In practice, we have willing to generate workspace shape as convex polyhedron at high alpha value.



(a) alpha = 20



(b) alpha = 100



(c) alpha = 400

Figure 3: The surface reconstruction is on the basis of alpha shape. Towards points in the point cloud, we try to use the different alpha value to rebuild the robotic workspace shape. At low alpha value, the shape can be more accurate. But the alpha value needs to be manually estimated according to the different point sets.

3.1.3 Determine if the point is inside a polyhedron

Assume that we have a target point that we want robotic arm to reach. Before we active the robot joint motion, it will be better for us to check if this target point is really located in the robotic workspace. Generally, we can treat the robotic workspace as a polyhedron.

For some polyhedron with simple structure such as cube and regular tetrahedron, just comparing the distance between target point and vertex could assist us to finish the quick judgement. But in many case, in the influence of external obstacles at robotic work environment, the robotic workspace might be formed as a complex polyhedron with huge amount of vertex, edges and flat polygonal faces.

In the computer, the polyhedron object is saved as the composition of a large number of triangular geometries, which covers the surface of the polyhedron. Presumed that the normal

vector of the triangle is known, using the target point and edge point in triangular to construct a line, then we can straight forward to use dot product to calculate the angle between line and normal vector. This method also help us to confirm whether the point is above or below the triangle shape.

Although, with the help of alpha shape method, we can easily get the non-convex shape with higher precision. Considering that we can not manually judge the best alpha value for every robotic workspace and we should also focus on balance between computation time and model accuracy, we want to simplify the current workspace of robot as a convex polyhedron shape. By validating if the target point is below all triangle shape from the polyhedron, we have a ability to conclude if a target point is located in the robotic workspace. The summary of this algorithm as follows.

Algorithm 2 The point in convex polyhedron

Require: $targetPoint$
Initialize: $count = 0$
/* The numberOFface is the amount of faces covering the polyhedron */
for $face = 1, 2, \ldots, numberOFface$ **do**
    /* Get vector between target point and a point in face */
    $vector = \vec{pointInface} - \vec{targetPoint}$
    /* Vector dot product, get the cosine of the angle between vectors */
    $dotVal = \vec{vector} * \vec{normalsInface}$
    /* The dotVal less than 0 means the angle between vectors is more than 90 degrees */
    if $dotVal < 0$ **then**
        /* Target point is below the current face */
        $count++$
    end if
end for
if $count == numberOFface$ **then**
    /* Target point is below each faces */
    the point is in this polyhedron
else
    /* Target point is above some faces */
    the point is not in this polyhedron
end if

Further more, for those robotic workspace as non-convex shape, we just need to decompose the shape into convex shapes. But the decomposition of non-convex is highly dependent on the manual parameters setting. These mini convex shape is hard for users to imagine and remodel. So, in link part of our robotic TD structure, we only save the whole robotic workspace as single simplified convex shape and check target point based on current shape.

3.1.4 Robotic coordinates mapping from digital twin to real world

In practice, the coordinates of robotic end effector in digital twin is obviously not same as in the real world. Moreover, some developers have willing to represent the robotic coordinates as the unit of millimeter. In this situation, the algorithm implementing automatic coordinates transformation between digital twin and real object is very critical for us.

In order to comprehend this coordinates transformation, I want to introduce a concept of mapping. In mathematics the mapping could be explicated as function. Using function to map the coordinates in domain of digital twin to the real world domain. The mathematics formula of mapping relationship is represented as follows:

$$\Phi \to \psi$$

$$\chi \to \kappa \cdot \varepsilon + \theta$$

In the particular implementation, the coordinate transformation relationship from the same coordinate axis is actually a linear relationship. Therefore, we assume that it exists position axis x_1, x_2 at domain of digital twin, which are corresponding to the position axis y_1, y_2 at real robot domain. We get the following equations, and the mapping relationship between two domains can be obtained by solving the binary linear equation system.

$$y_1 = \mathrm{k} \cdot x_1 + \mathrm{b}$$

$$y_2 = \mathrm{k} \cdot x_2 + \mathrm{b}$$

$$k = \frac{y_2 - y_1}{x_2 - x_1}$$

$$b = y_1 - \frac{x_1(y_2 - y_1)}{x_2 - x_1}$$

Besides, it is deserved to notice the selection of coordinate points. Considering it has measurement error from instruments in the coordinates from real robot domain. If two points in same axis from same domain are too close, it will cause to a large deviation in the mapping value, which make TD document not well express the workspace of the real robot.

In the thesis, we will give same joint position to the real robot and its digital twin at same time. After both robots have finished their movements, we record the current Cartesian position of them. Theoretically two robotic target points to finish the minimum requirements of our algorithm. But it is a little hard for us to find two points that they have large difference in the x,y and z axis at the same time. We would like to choose three or four points as candidate points and only get the biggest difference part of these points in same axis.

3.2 Tools

3.2.1 Using Standard Triangle Language (STL) file to store robotic workspace shape

In the previous algorithm, we mention what type of metadata from robotic workspace shape is necessary for us to verify if target point in the shape. At present, we want to choose a proper file format to save these data. This kind of file format must hold the vertex position and normal vector of sub-surface from polyhedron at the same time.

Compared with other candidates such as object file(OBJ) and Standard for the Exchange of Product Data(STEP), the STL file not only does not contains unnecessary information about colour, texture, but also it has a good balanced between model size and accuracy. Here is the structure of the STL file. For the each face in the polyhedron, STL will record its face normals and the position of three edges. Both are essential required data for our algorithm.

```
solid AssimpScene
    facet normal -0.6929 0.0400 -0.7198
        outer loop
            vertex -0.4314 -0.1700 0.0145
            vertex -0.5120 -0.1593 0.0927
            vertex -0.3914 0.5652 0.01699
        endloop
    endfacet
    facet normal ......
    ......
    endfacet
    ......
    endfacet
endsolidAssimpScene
```

Listing 2: Data structure of STL file

At the link part of our robotic TD document, it will contains a link to indicate the simplified convex shape of robotic workspace saved as STL file format. Users can directly entry this link to get the data structure as above.

3.2.2 Unified Robotics Description Format(URDF) to describe robot

For the goal of replicating the behavior of robot and its response to circumstance, we need to find approaches to get both digital twin of robot and robot work environment. And the Unified Robotics Description Format(URDF) can help us to generate the precise digital twin of robot.

Obeying the XML specification rules, URDF records the structure of the robot by clarify the affiliation of joints. And in the link part, it use collision and inertial as label to save the dynamics and kinematics properties of robot. The shape file about robotic link also is saved in the collision part. Many robot simulation software supports URDF importer plugin, which could realize automatic import and generate a proper digital twin of the robot. Here is the basic structure of URDF file

```
<robot name="template">
    <link name="base">
        <visual>
            ...
        </visual>
        <collision>
            ...
        </collision>
        <inertial>
            ...
        </inertial>
    </link>
    ......
    <joint>
        <axis xyz=" 0 0 1"/>
        <parent link="base"/>
        <child link="link1"/>
        <origin/>
    </joint>
    ......
</robot>
```

Listing 3: URDF Elements and Attributes

Although URDF file support XACRO extensions and gazebo element for simulation, these part can not be recognized by the robot simulation software Coppeliasim. In the thesis, the URDF file we discuss, it is only the initial version without any extensions.

URDF file has ability to describe robot with complex mechanical structure that it includes a lot of free-moving or assitant joints such as Dobot. But these robot models are only rare cases in real life. In the industrial scenario, the most of robots have similar structures that each joint works as motor and is controllable. Therefore, for the TD part of recording robotic joints restrictions, our robotic TD document will exclude the description of robotic assist or free joints and only log the information of main joints.

3.2.3 The robot simulation software Coppeliaism

Nowadays, it exists several robot simulation software such as Gazebo, Webots in the market, which both own the capability to accurately simulate the behavior of the robot. But for the purpose of integrating WoT library, we prefer to the software that could be accessed via remote API by Typescript environment. In this situation, the robot simulation software Coppeliasim becomes our best option.

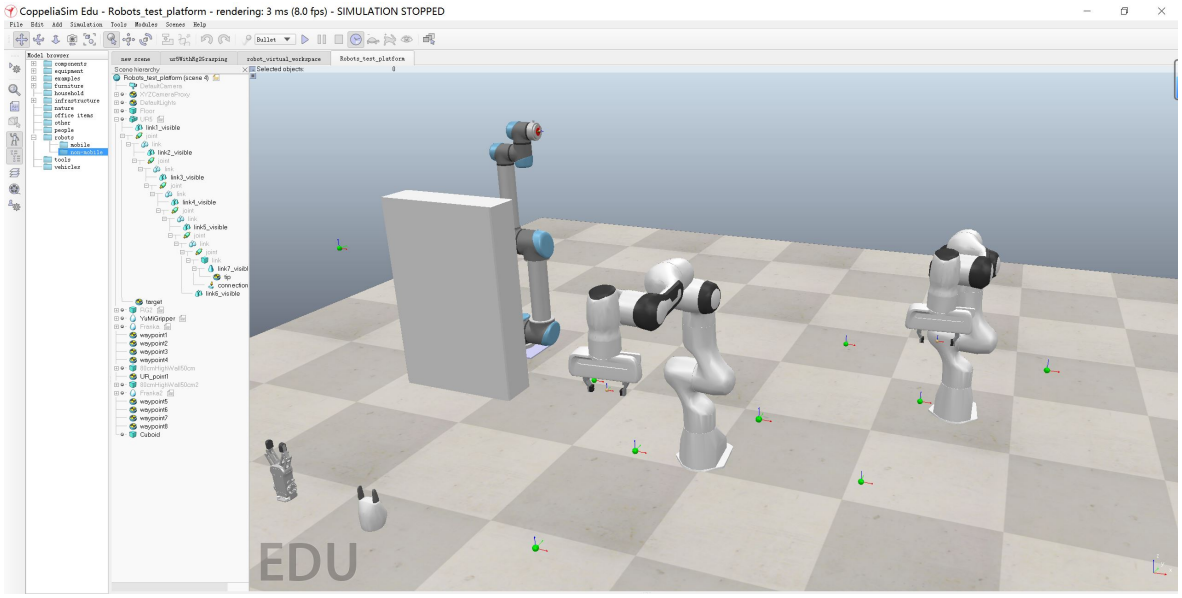


Figure 4: Coppeliasim graphical user interface[11]

The Coppeliasim software provides a flexible, expandable, abundant functionality robot simulation framework called V-REP[11]. Based this simulation framework, we can straightly deploy our control algorithm and inverse kinematics calculation in embedded script and associate this script with robot model.

As one of the most features in Coppeliasim, the embedded script is coded by Lua, which is an effective and lightweight scripting language and widely used in many fields such as industrial applications or games. In this thesis, our main method to make interactions with the robot in Coppeliasim is to invoke the functions we pre-defined in the embedded script.

4

Approach

4.1 The structure of Thing Description to represent kinematics of robot and its working space

After we have introduced the related algorithm and tools used as analyze and research, we will formally begin approach part to exhibit our main contributions. As the research question in this thesis, we first want to introduce the structure of Thing Description (TD) document, which have ability to correctly represent the kinematics and workspace of robotic arm.

In the state of the art, we mention that the TD document is mainly compose of three parts: property, action, event. The event normally is trigger in special condition and will only be published to the subscriber. In this situation, we don't give thought to describe event in our robotic TD document to include the information of robotic arm.

For the property part of robotic TD document, we generally write the underlying information of robotic structure, such as the robotic joint types and limits, robotic Cartesian working space limits. Therefore, the current robotic TD document only includes two properties: **getJointposition**, **getCartesianposition**, which stands for the most fundamental information of robotic arm.

In the properties of **getJointposition**, the number of joint object is dependent on the specific robot. Some robotic arms, such as the UR series, will have 6 main joints. And for the robot with complicated mechanism structures such as Dobot or Uarm, we only record the information of their main joint in our robotic TD. For the maximum of minimum value in the property of **getCartesianposition**, it relies on the size of the robot's working space in its work environment.

```
"properties": {
    "getJointposition":{
        "title": "get position of each joint",
        "type": "object",
        "properties":{
            "joint1":{
                "type": "number",
                "unit": "deg or meter",
                "minimum": -360,
                "maximum": 360
            }
            ...
            "jointX":{...}
        },
        "forms":  [...]
    },
    "getCartesianposition":{
        "title": "get Cartesian position of the robot end effector",
        "type": "object",
        "properties":{
            "x":{
                "type" : "number",
                "minimum": -100,
                "maximum": 100,
                "unit": "millimeter"
            },
            "y":{...},
            "z":{...}
        },
        "forms":  [...]
    }
}
```

Listing 4: The template structure of properties in robotic TD

Except for the maximum and minimum value of robotic joint or Cartesian limits, the unit of value is also essential part for us. Obviously, from the dynamics movement perspective of robot structure, not all joints act as rotational movement between links. For the robots containing prismatic joints, which takes translational movement, we need to set its unit as meter or millimetre. A common example for this kind of robot is Collaborative M1 Pro from the Dobot company[12].

In the Cartesian limits, the unit of value is significantly helpful for us to understand largest workspace of robotic arm. To the action part of TD document, we comply with the same design rules about joint or Cartesian restrictions as properties. It only constitutes two actions as well: **moveTojointPosition**, **moveTocartesianPosition**. We can control the motion of

robot based on joint or Cartesian coordinates by invoking the corresponding action.

```
"actions": {
    "moveTojointPosition":{
        "title": "let robot move according to joint position",
        "input": {
            "type": "object",
            "properties":{
                "joint1":{
                    "type": "number",
                    "unit": "deg or meter",
                    "minimum": -360,
                    "maximum": 360
                }
                ...
                "jointX":{...}
            }
        },
        "forms":  [...]
    },
    "moveTocartesianPosition":{
        "title": "let robot move according to Cartesian position",
        "input": {
            "type": "object",
            "properties":{
                "x":{
                    "type" : "number",
                    "minimum": -100,
                    "maximum": 100,
                    "unit": "millimeter"
                },
                "y":{...},
                "z":{...}
            },
        },
        "forms":  [...]
    }
}
```

Listing 5: The template structure of actions in robotic TD

As a short summary, we want to elaborate on the relationship between the two actions and properties. In the robotics, we can describe the trajectory of the robot motion in different coordinate systems. The joint space is used in the robotic forward kinematics (FK). We just need to give target joint angle to each robotic joint and the robot will move based on its joint position. And in the Cartesian space, we only need to deliver the space coordinates of target position. Through the inverse kinematics (IK) calculation we can get the required joint angle

for robot to this target position. Sometimes, it exists multiple joint angle solution for robot to reach this position.

Combine the robotic kinematics process with our TD document, users can invoke the action **moveTocartesianPosition** to calculate the required joint position by inverse kinematics. After reach the target position, they can read properties **getJointposition**, **getCartesian-position** to obtain the current joint angle or the position of robotic end effector. They can also invoke the action **moveTojointPosition** to reach position by forward kinematics. Then they can use same properties to get robotic position in different coordinates

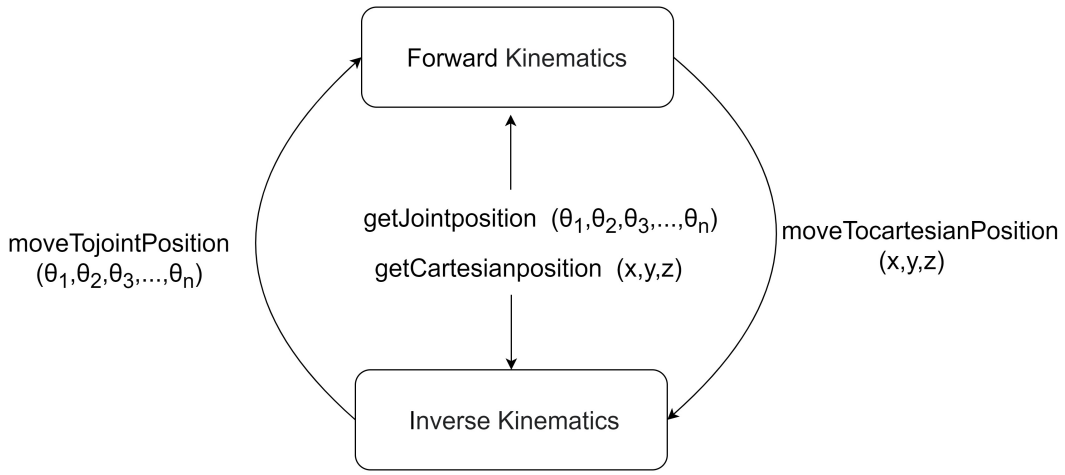


Figure 5: The relationship between interactions in Thing Description and robotic kinematics

The above part of TD document mere describes the robotic motion control mode and extreme admission value in robotic joints or Cartesian coordinates. And due to constraints of TD document construct rules, we can not clearly demonstrate what does the robotic workspace look like by properties or actions in the TD document.

The strict data structure rule of property and action part in TD document make it a little hard for us to write a simple and logical content as representation of robotic workspace. In this situation, we pay more attention to the link part of TD document.

In the minds of WoT architecture drafters, link part in TD document will provides people some download links. Users will have ability to get files and metadata related with the current TD by access links. TD document normally does not have any strict rules or requirements to the file type in the link part, which bring us opportunities to propose freely.

After discussion and consideration, we think the link part of TD document should includes the following metadata. Without doubt, the first object should be obliged to the shape of robotic workspace. Users can get authority to download the related shape file of robotic workspace

through the link. The format of shape file is STL. To be consistent with our previous algorithm, the STL file only saves a convex polyhedron and is located in the link part of robotic TD.

```
  ...
 "links": [
   {
     "href": "http://localhost:8000/<robot name>/robot_shape.stl",
     "type": "model/stl",
     "rel": "workspace"
   },
   {
     "href": "http://localhost:8000/<robot name>/data_point.csv",
     "type": "text/csv",
     "rel": "dataset-points"
   },
   {
     "href": "http://localhost:8000/<robot name>/digital_twin.ttt",
     "type": "application/octet-stream",
     "rel": "Coppeliasim scene"
   }
 ],
   ...
```

Listing 6: The template structure of links in robotic TD

Robotic workspace shape is generated from the data points. These data cloud points record coordinate values, which equals that the end effector of robotic arm has reached to this position. Supported by different surface reconstruction algorithm such as Alpha shape method[10], we can get either convex or non-convex workspace shape from current data points. Therefore, the initial data points table is of vital importance for us, when we want to analyze the robotic workspace detailed.

Besides, it usually exists some obstacles in robotic workspace, which cause collision when the robotic arm try to take movement. In this circumstances, we want to the indicate the collision state of each data points in our data points table. The collision state of the table offers us the feasibility to use some classification algorithms such as KNN, SVM or other machine learning algorithm predicting the collision state of the future target position of robot.

The third essential metadata for us is the digital twin of robot work environment, which reproduces the position relationship between robots and other objects in the real scene. In our TD document, this digital twin will marked as the scene file used in the Coppeliasim software. We believe that these three metadata can precisely represent the robotic workspace and circumstance.

Due to limited space of pages, we will not show other essential parts in the TD document,

such as **id**, **security** and **securityDefinitions**, which are not associated with the dynamics and kinematics of robot. But our method will also automatically generate these TD parts according to user requirements to construct a complete robotic TD.

4.2 Integrating Web of Thing library via remote API in Coppeliasim

Actually it exists many robot simulation software such as Webots, Gazebo and ROS in market, which owns a amount of large users and community groups. They both have capability to accurately simulate the motion of robot. But We take a first consideration to the software, which could be directly interacted with our WoT code library.

The entire WoT library is running in the Typescript environment. In this scenario, the robot simulation software which supports communication via Typescript code is preferable for us. Just like most of IoT devices, Coppeliasim software developers have prepared the Application programming interface (API) for users, which provides the feasibility of further development. These remote API can be coded by Javascript or Typescript.

Therefore, We think the integration of WoT server with Coppeliasim software could be practicable. Compared with other simulation software, robot simulation software Coppeliasim same could correctly simulate the motions and kinematics of robot. Besides, the biggest advantage of Coppeliasim in comparison with other candidate is that we can also freely add and modify user-defined scene in the Coppeliasim.

Although Coppeliasim officially advertises that we can directly control the trajectory of the robot in the scene through the movement functions in remote API. In the latest version of Coppeliasim, the API that controls the movement of the robotic arm involves activating callback function, which never exists in Typescript code environment.

In this situation, we decide to write the embedded scripts for virtual robots as robotic driver at first. Our pre-defined scripts includes the functions to control robot movement. Then we can use methods in remote API to directly call functions from our scripts. And this method also help us to reduce the communication time used to call the remote API and improve the running efficiency of the Coppeliasim scene.

Here we try to design three scenes and write its related scripts, which shows how to access parameters and objects in Coppeliasim scenes via remote API by Typescript. The first scene includes two virtual lights. We can control the state and color of lamp bulb. It is worth noting that the robot simulation software Coppeliasim does not do much processing in image texture and light scattering. Therefore, we can only get the mapped color when we observe the object in the scene from a certain angle.

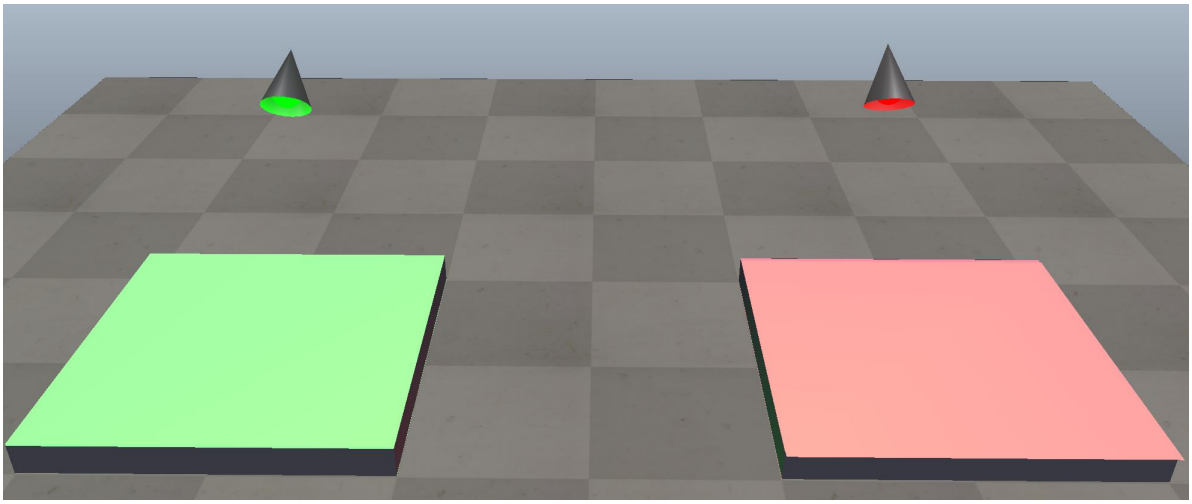


Figure 6: The first scene: Virtual light

And the second scene shows how to implement inverse kinematics calculation to virtual robot model in Coppeliasim and make robot move to the target position via remote API. In this scene, we can make robotic arm move to either the Cartesian coordinates or move based on its joint angles.

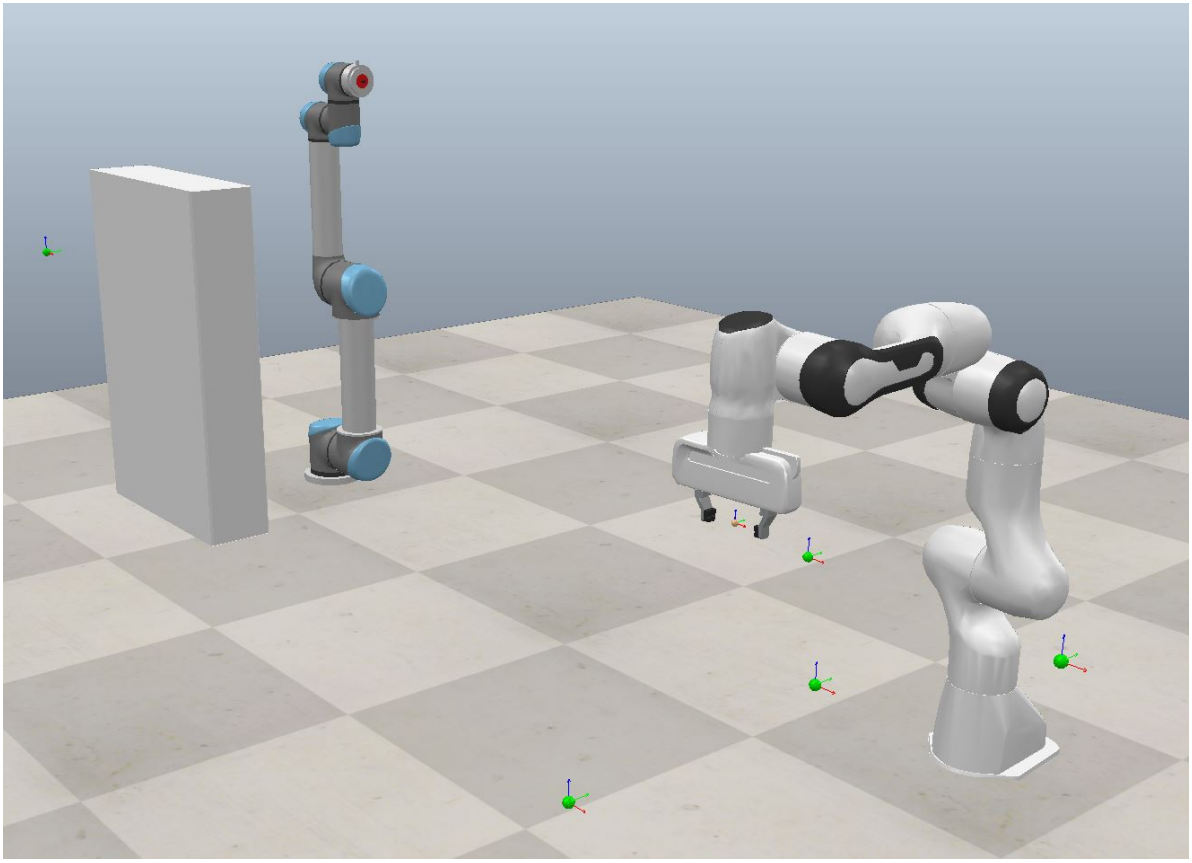


Figure 7: The second scene: Virtual robot

The third scene is a more complex scene. In this scene, we tend to simulate a complete production process. We imitate the real IoT platform in the lab to build this virtual scene. At present, it could successfully visualize some operations such as the conveyor belt running, cube color check and return cube. However, we still not achieve the digital twin of scene that holds full functionality and same response ability as the real scene.

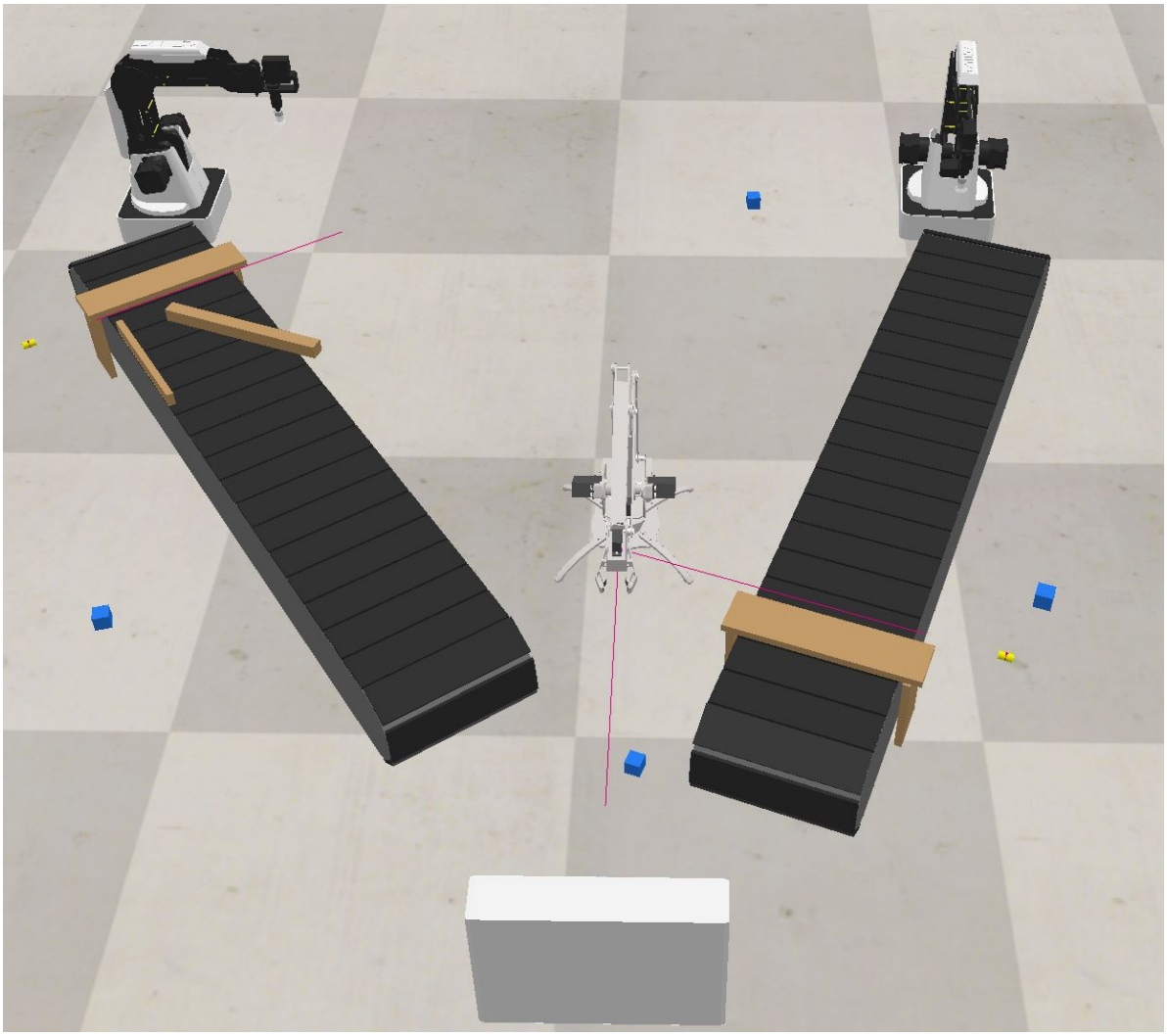


Figure 8: The third scene: Virtual lab

During the construction of three scenes, we have confirmed the software structure of virtual devices system in the Coppeliasim. The whole software development is completely based on the object-orient process(OOP). It means we treat each type of virtual device in Coppeliasim as class. To the same type of virtual device in different scene, we just need to generate the instance of device class to manage them. To our further development of new objects and scenes in the Coppeliasim, it also keeps this rule.

Furthermore, we also need to take consideration that the WoT server and instance of robot class interact asynchronously with the robots in the Coppeliasim scene. After we call the move

function in the instance of class, this function only return value when the motion of virtual robot in scene is finished.

The remote API of Coppeliasim supplies the blocking functions for movement. But both of these function involves the using of their call function, which is impossible implemented in the Typescript code environment. We need to do further development based on current API to design our own blocking functions of movement.

Luckily, the Coppeliasim embedded scripts are supported to be coded by Lua, which contains a thread mechanism called coroutine[13]. Coroutine in Lua is only executed once and then it will go to dead state and be automatically resumed by the Lua scripts. In practices, each time we call our movement function, we will put it in a coroutine of Lua script. By checking state of the Lua coroutine we can know if the movement of the robot arm in the Coppeliasim environment has ended.

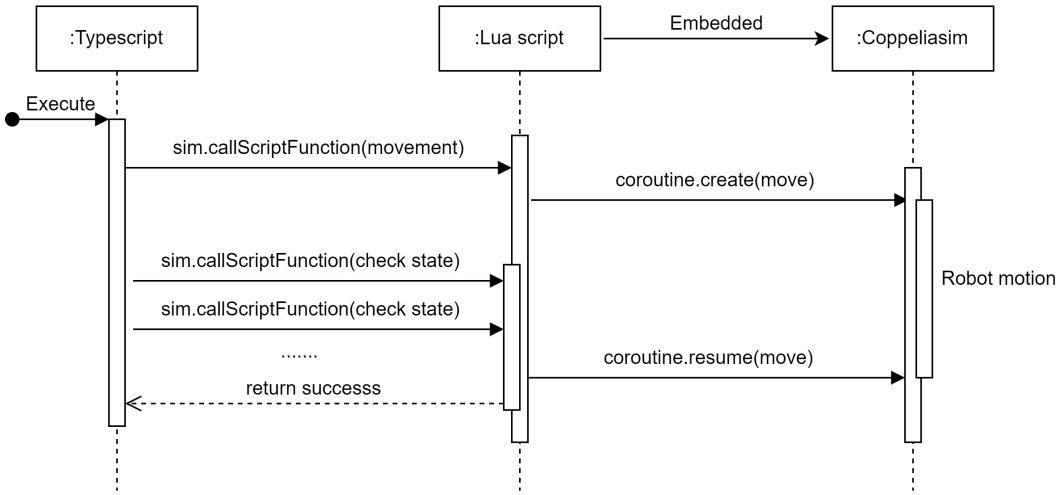


Figure 9: Sequence diagram for activating robot movement function

In brief, we can control the most of parameters and objects from Coppeliasim scenes via remote API. It also proves the reliability of running a large amount of virtual devices at same time in Coppeliasim. Now we just need to generate new instance in WoT server based on current virtual devices class, which could realize the integration of WoT library and Coppeliasim.

According to our rules of robotic TD, we manually write our first TD document of robot, which is consumed by our WoT server. After proving the feasiblity of using remote API in Typescript, it is not difficult for work for us to integrate the WoT library with remote API and make WoT server have ability to communicate with Coppeliasim scene.

On the basis of previous virtual robot scene, now we do a little modification and verify the

communication stability between our WoT server and the current Coppeliasim scene. In the following figure, we invoke the action **moveTocartesianPosition** via WoT client. When the WoT server receives action requirement, it will interact with Coppeliasim scene via Lua script to make robotic arm move. This result also shows the potentiality of building accurate digital twin of robotic work environment.

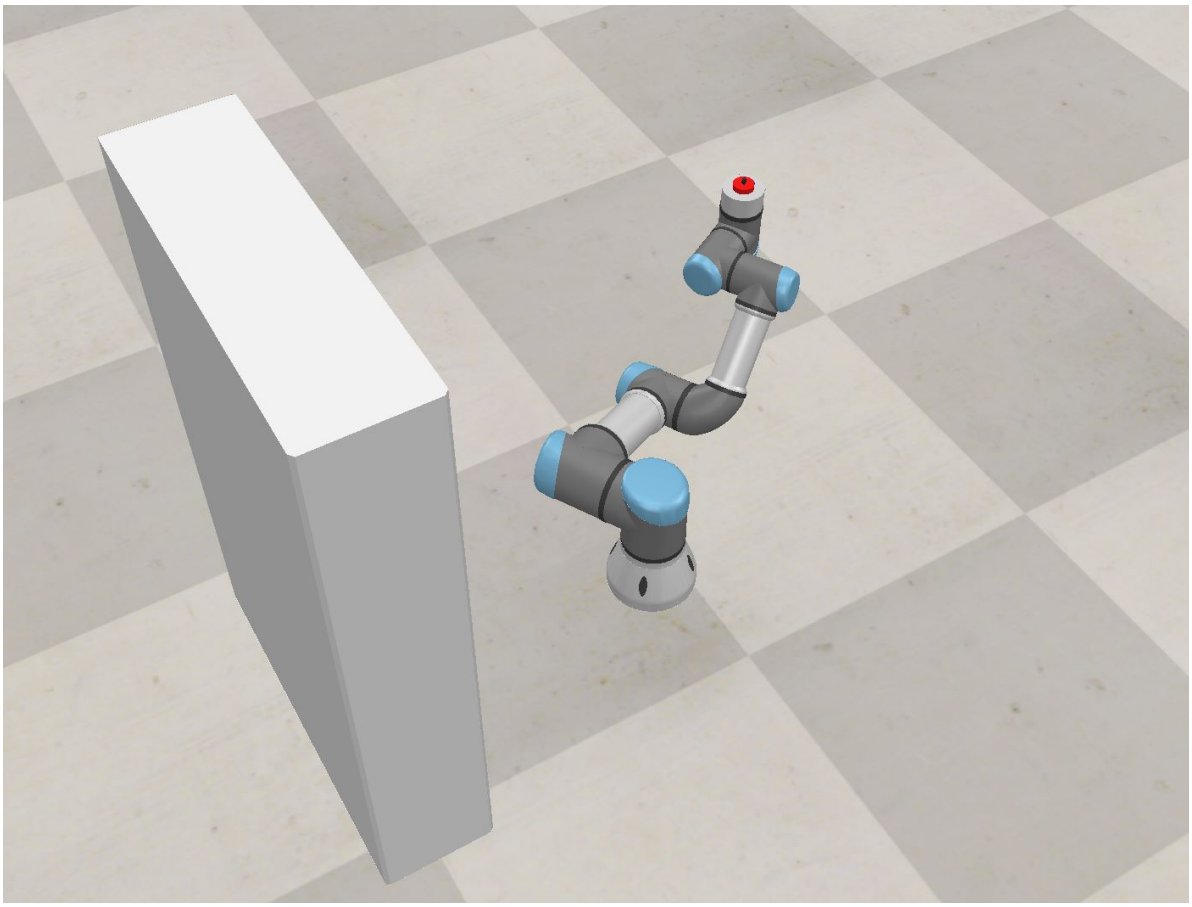


Figure 10: Robot motion control via WoT architecture

4.3 Implementation of automatic digital twin generation from robot URDF file

Multiple default robotic arm models are already established into the Coppelliasim software. However, it will exist infinite robotic arm models in the real world. Just like TD document describing devices, Unified Robot Description Format (URDF) could express robotic mechanical structure and dynamic properties of robot. If we want to develop robotic TD document for any kind of robots, we must extract the digital twin of robot from any URDF file.

In the Coppeliasim graphical interface, it exists URDF imported plugin, which help us to

import the robot model into current Coppeliasim scene. But this plugin don not have ability to correctly parse the dynamic relationship between robotic joints and links, which leads to robotic arm crash in scene.

Moreover, it could also be feasible that we access the URDF importer plugin and its functions via the remote API from Typescript code. So, on the basis of Coppeliasim official plugin, we do further development and realize code implementation of correctly importing robot model in Coppeliasim scene.

The workflow of generating the digital twin of the manipulator model by URDF is as follows. First we will inspect the integrity and correctness of URDF file. Then we use the remote API in Coppeliasim URDF importer plugin to load robot model in the Coppeliasim scene. After this, we will set control model of all joints from robot as position control and parse the dynamics properties of robotic links. Finally we will save this digital twin of robot as model in the local computer.

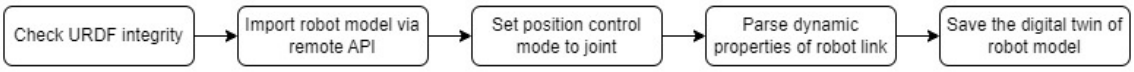
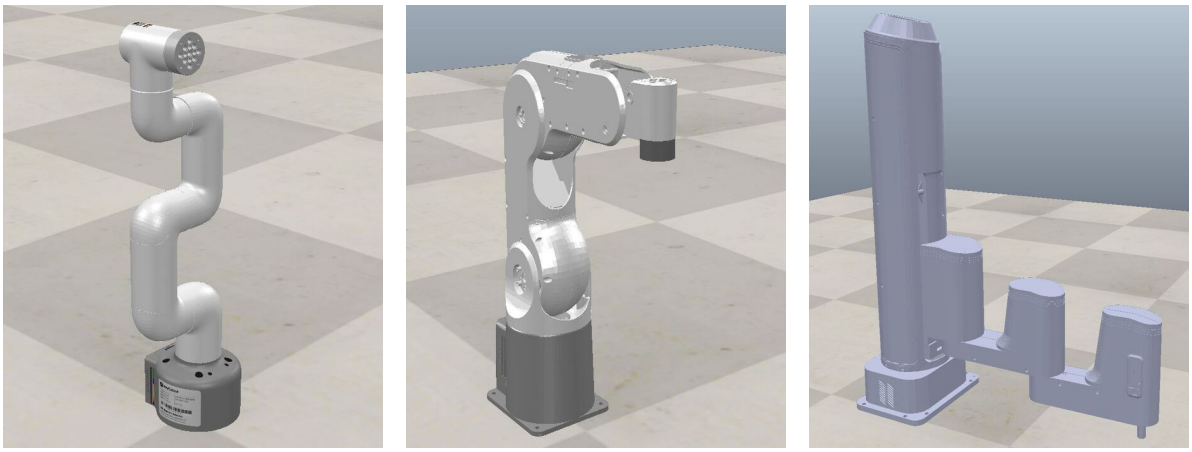


Figure 11: Workflow for generation of robot digital twin

Through verification in other tasks and experiments, our digital twin robot model with reliable dynamic properties could robust work in the simulation environment, without the occurrence of mechanical arm collapse and other problems. There are some robot model examples such as myCobot and M1 Pro robot, which is both extracted from their URDF files.



(a) myCobot robot (b) MyPalletizer robot (c) M1 Pro robot

Figure 12: Some robot models imported from URDF

4.4 Implementation of automatic inverse kinematics calculation for robots

Realizing the automatic inverse kinematics (IK) calculation is a critical part of verifying the accurate of robotic digital twin. Normally, we prefer to send the command of target coordinates to the robotic arm instead of adjusting joint angles. The digital twin of robot has quite similar kinematic behavior as real robot. It means IK calculation procedure in virtual robot could also be deployed in the real robot.

The Coppeliasim software has a fully functional IK library and related IK tutorial support. The IK configuration of the robot can be realized by setting related library functions in embedded scripts. However, it might be a little boring job for us to write IK scripts to each type of robot in Coppeliasim. For the infinite type of robots in the world, the method of automatic IK configuration could effectively improve the efficiency of our robot TD file generation.

Actually, the most of robot in the real world owns a common structure that each robotic joint equals to a controllable motor. A interesting thought for us to set a Coppeliasim embedded script called robot driver script, which could automatically analyze properties and limitation of robotic joints, then automatically produces the corresponding IK method. Each time we just need associate this robotic script with related virtual robot. We can control robotic motion in our robot WoT server.

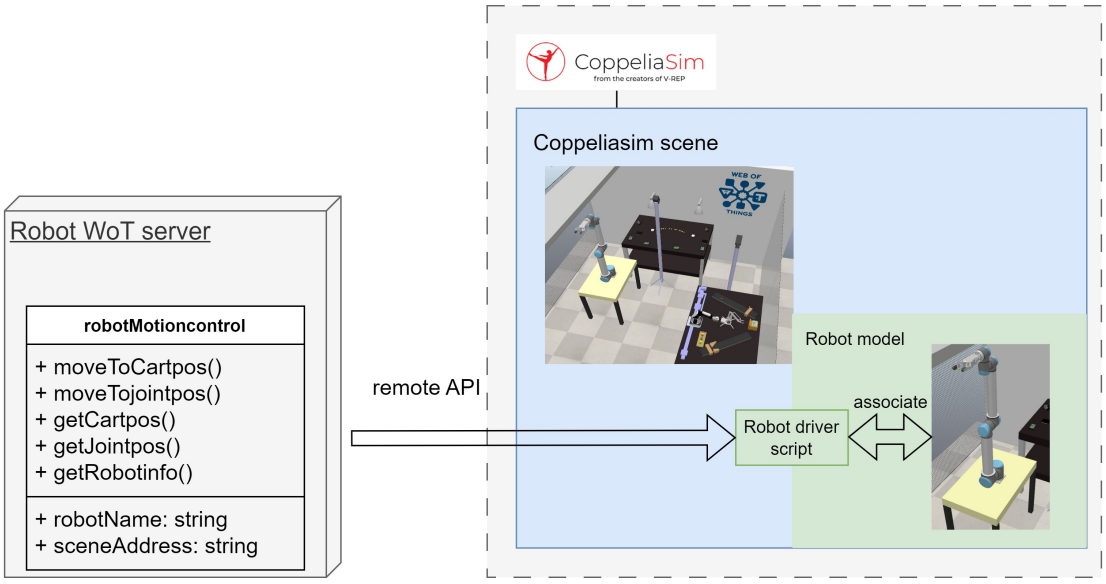


Figure 13: Interact with robot digital twin via driver script

Except for automatic implementation of IK process, by call the corresponding functions our script could also fetch robotic information from digital twin of robot in Coppeliasim. These robotic information includes limitation and type of joint, the name of robot. But our robotic driver script could only be suitable for the robots with common structures. To the robot with assist joints like Uarm, meARM or Dobot, we need to try to simplify their structure or manually write the script for this kind of robot and calculate joint compensate for their assist joints before parsing the inverse kinematics process.

So far, we have already provided code script and IK parsing algorithm support for actions and properties in our TD document. By using our WoT server, we have ability to manipulate the digital twin of robotic arm with common structures at Coppeliasim.

It is also worthy noticed that the coordinates definition in TD document is fully based on Coppeliasim scene, which might differ from coordinates definition in real robotic arm. Therefore, if we want the TD document from digital twin of robot truly suit for the real robot, we need to do coordinates transfer between the virtual scene and real scene and indicate it in the TD document. And the coordinates mapping algorithm we mentioned in the algorithm part could help us to solve the problem.

4.5 Implementation of automatic robotic TD document generation based on robot from digital twin scene in Coppeliasim

In the previous sections, we prove the possibility to make interactions with Coppeliasim scene only by remote API instead of Coppeliasim interface and show how to automatically take robotic inverse kinematic conversion for the robot with common structure and combine it with WoT server. Rely on our rules of robotic TD document, we need to find an approach to extract necessary information from current robotic digital twin at the virtual scene, in order to achieve TD document automatic generation.

According to the classification of robotic workspace, our TD file interactive methods are respectively based on Cartesian coordinate domain and manipulator joint coordinate domain. In order to fill maximum and minimum value from the interaction methods **getCartesianposition** and **moveTocartesianPosition** at Cartesian coordinate in our TD document, we need to obtain extreme working range of the robot at the Cartesian coordinate system.

The robotic accessible points from workspace generation algorithm in the thesis will be suitable for this problem. After we get the accessible points list, we will take out the interval of the points in the list under the x, y, and z coordinate axes and maximum and minimum value in the corresponding interval is the data we want.

In last section about robotic IK, our robot driver script includes a function, which could record the related features of joints. So we can use this robot information to set restrictions of joints in the property **getJointposition** and in the action **moveTojointPosition**. These interaction methods work at robotic joint coordinate domain. The data structure of the robot information is as follows.

```
{
  "jointAmount": 6,
  "robotName": "ur3_robot",
  "jointLimitLows": [
    -6.28, -6.28, -6.28, -6.28, -6.28, -3.14
  ],
  "jointLimitHighs": [
    6.28, 6.28, 6.28, 6.28, 6.28, 6.28
  ],
  "jointTypes": [
    "Revolute_joint", "Revolute_joint",
    "Revolute_joint", "Revolute_joint",
    "Revolute_joint", "Revolute_joint"
  ]
}
```

Listing 7: A example output for the robot information

After consideration, just like our generic robot driver script fitting for the most of robot with similar structure, we also want to design a robotic workspace generation script, which could be also applied in the most of robots. In the code practices, for the robot with common mechanical joints and links, we can directly use remote API to call metadata generation function in our script and automatically generate TD by Typescript code.

The flow of automatically generating TD for the robot with common structure is recorded as follows. First, we should get the Coppeliasim scene address and robot name for the instance initialization of class **robotDescriptiongenrate**. Then we use the function **robotInfogeneation** to make digital twin of robot produce the required metadata. Finally, we can use the function **generateTD** to automatically generate the robotic TD document based on current robot model. This generated robotic TD could be directly consumed by the WoT server.

In the graph, we list four robot metadata, which is essential for the robotic workspace generation. The first metadata is the STL file that records the simplified convex shape of robotic workspace. The second CSV file contains the robotic accessible points and its collision state in workspace. The JSON file with basic information of robot obeys the data structure we mentioned. The last metadata is the digital twin scene. Since we have extracted all useful information from the third metadata JSON file, we do not put the JSON metadata in the link part of our robotic TD document.

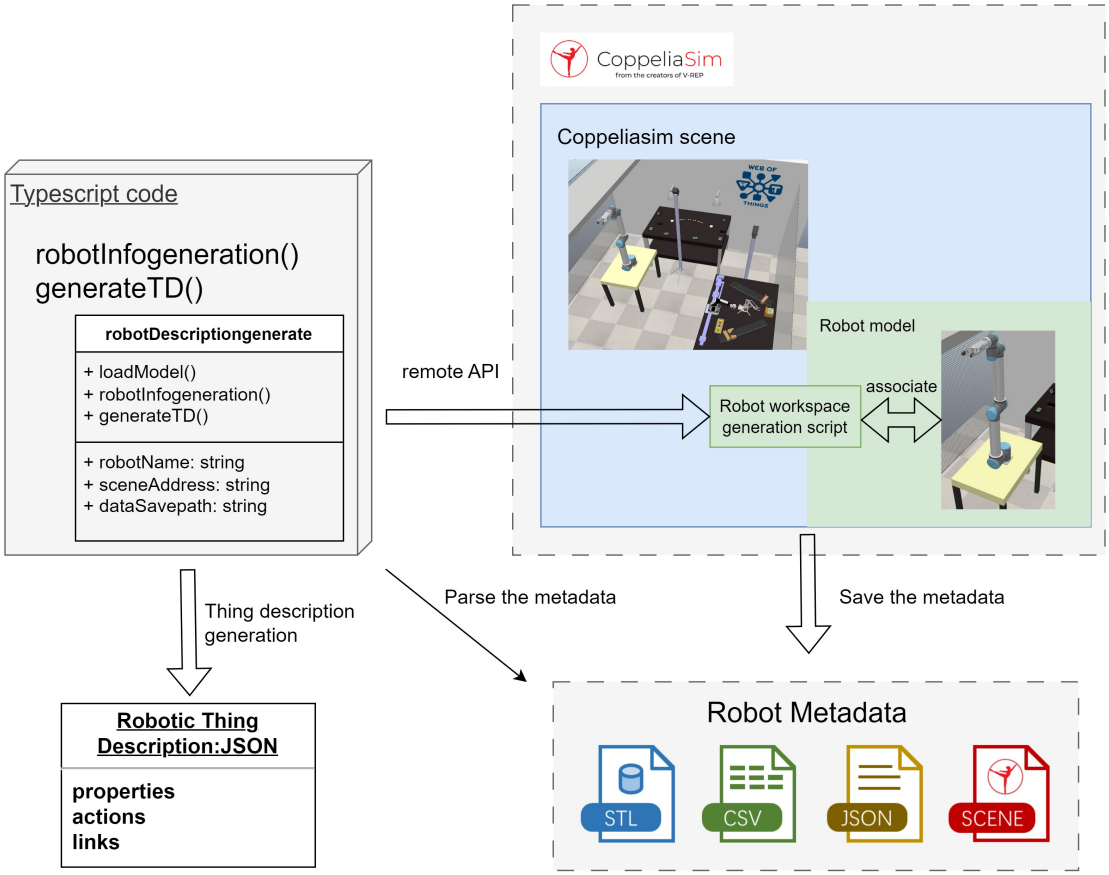


Figure 14: Thing Description generation for common robots

However, for the robot with complex structure such as Uarm and Dobot. When we randomly set the angle to main joint in those robots for the algorithm of generating points in workspace, we must manually compensate its auxiliary joint according to the robotic mechanical structure. The manual offset value is quite different for this type of robot. Otherwise, we must try to simplify the complex structure robot into the standard structure. After our inspection and testing, the simplified method is only applicable to a small number of robots such as Dobot. The more free joints there are in those robot, the harder for us to do simplification.

In this situation, we will write workspace generation script for those robots by hands. We directly run the Coppeliasim scene simulation at the Coppeliasim software interface. These script will be executed in Coppeliasim without help of remote API. And it will automatically save the required metadata in the computer. Finally, we still use the instance from the same class **robotDescriptiongenrate**, but we only call the function **generateTD**.

For the robot that each joint is controllable, We have actualized to automatically generate the metadata required by TD document and generate corresponding TD files by parsing the metadata. To the robot with special structures, if we have the metadata that comply with

our TD document design rule, we also have ability to finish the work of automatic robotic TD generation.

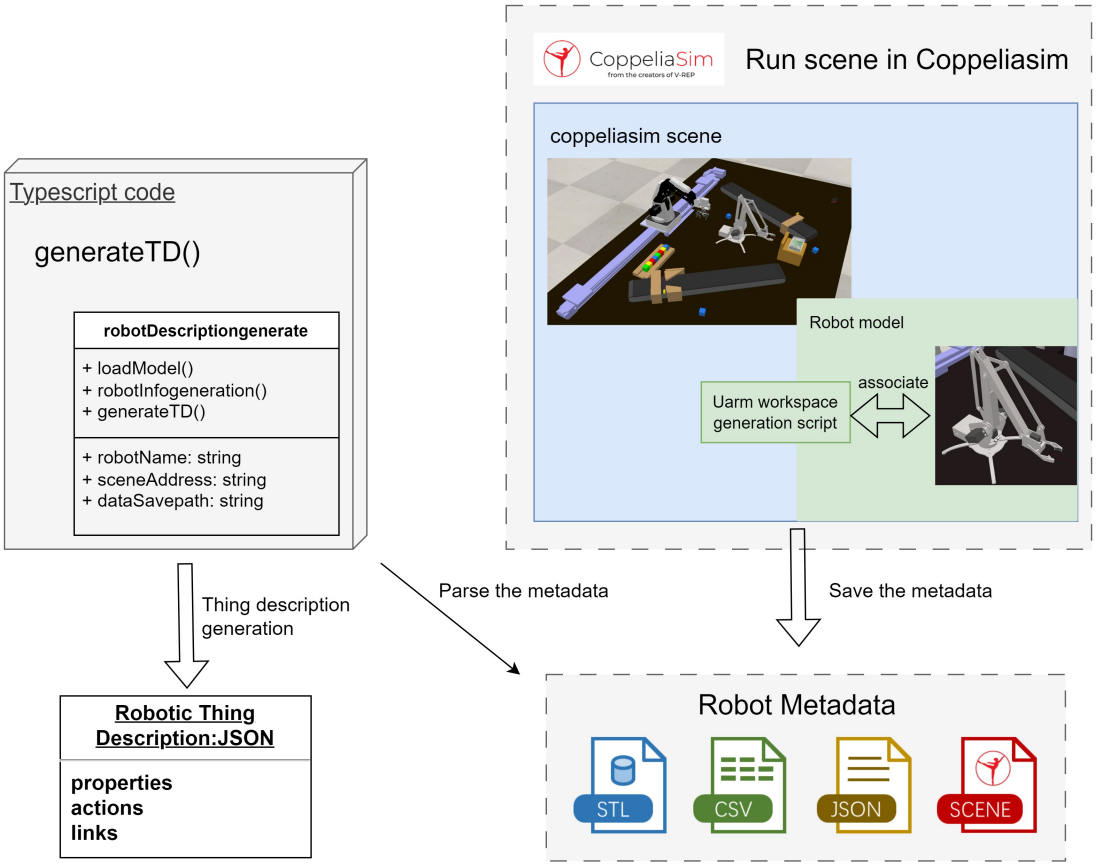


Figure 15: Thing Description generation for Uarm robot

Although we propose the approach to perform TD document automatic generation, the current coordinates in the robot TD document is still based on the coordinates system from the digital twin scene. If we want to use our coordinates mapping algorithm for the automatic coordinates transfer between the real scene and digital twin, We must need the reference coordinate point that the real robot and virtual robot has same pose as a comparison.

In summary, we have the script to automatically generate required metadata and the method to parse metadata for automatic TD document generation. The accuracy of robot model from URDF file has a high reliability. However, the precision of digital twin scene make a final decision to the precision of robotic workspace, which has also the influence on the precision of robotic TD document. In the next section, We will show the approach to manually build a digital twin scene as accurate as possible.

4.6 Build the digital twin of IoT remote lab scene with same functionality and response ability

In the previous section, we introduce the method to generate the digital twin of robot based on URDF. This virtual robot has same action behavior as the real robot. But, if we want to produce a satisfied TD document representing accurately the workspace of robotic end effector, it is necessary for us to set up the digital twin of robotic working environment. In the thesis, we want to validate our robotic TD by robot Uarm and UR10. So we decide to design two digital twin of IoT scenario: IoT platform and the entire IoT lab.

Since we want to design a digital twin of the IoT platform, it must exist same kind of real IoT platform in the world, which is placed at IoT remote laboratory. The IoT platform has been running stably for several years and has been used for teaching and research many times. The research about system description[14] for mash up TD also is verified in this platform.

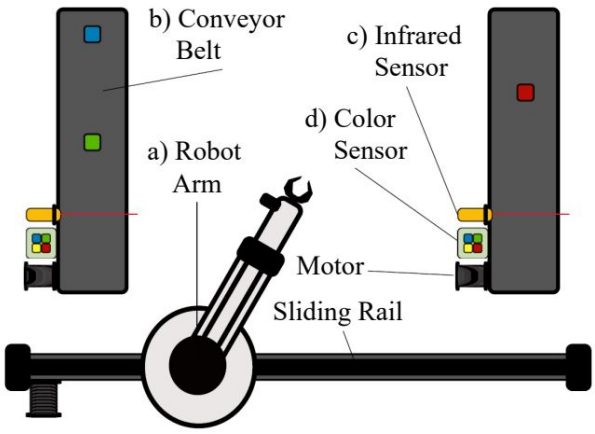


Figure 16: IoT scenario for system description verification[14]

The above figure only shows a part of the whole IoT platform. In the real IoT remote laboratory, the Uarm and dobot robot constitute the main component of this IoT platform. The two robot are responsible respectively for managing the cube from the warehouse and transporting the cube to the desired area for color detection.

In the previous approach part, we prove the potentiality and possibility of replicate the IoT platform in the Coppeliasim virtual scene at the visualization phase. Compared with the IoT platform in the real world, our initial virtual IoT platform is only able to execute similar action such as cube move and check cube color. There are still large deviations in the size of the device and the relative position of the device between two scenes.

This IoT platform contains many types of devices, such as conveyor belts and infrared sensor with complex shapes and mechanical structures. The common method of scene reconstruction

through point cloud can not be directly deployed in the scene of Coppeliasim. So after discussion, we decided to simplify the structure of some devices and manually measure the relative distance between each device.

After adjusting the relative distance between devices to make sure as consistent as possible to the IoT platform in the lab, we have successfully achieve digital twin of IoT platform with same full functionality as the real scene. A rough outline of the virtual scene is shown below, and the real scene of IoT platform is also recorded as a comparison.

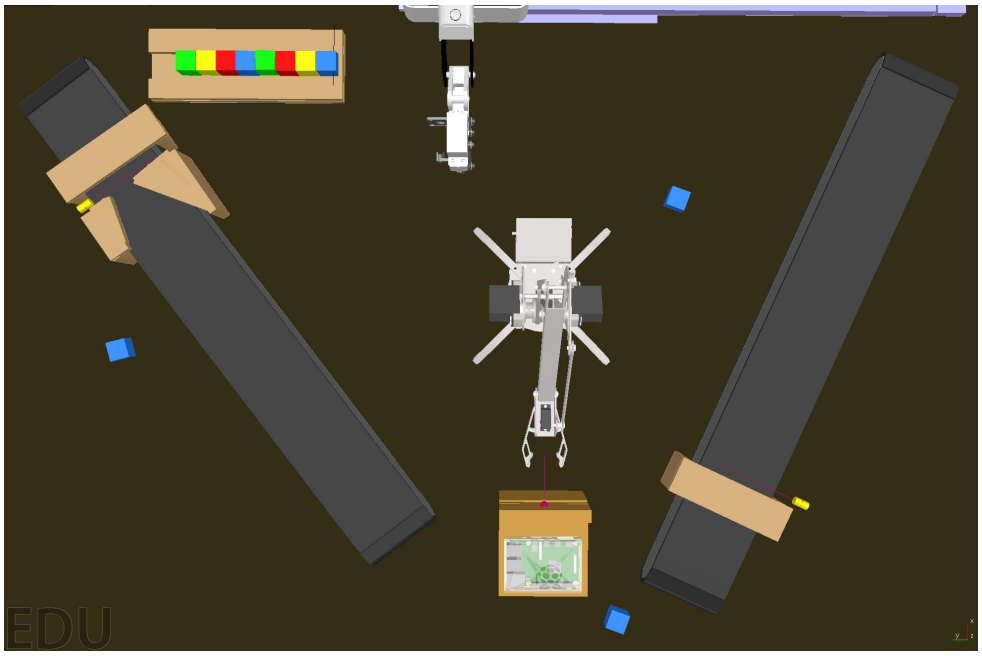


Figure 17: The digital twin of IoT platform

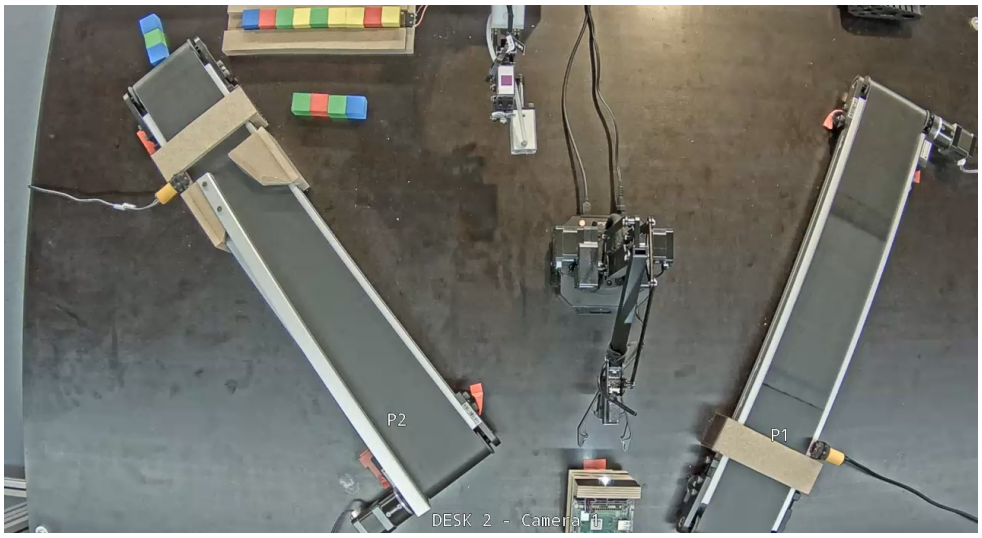


Figure 18: The real IoT platform

In the contrast with the real IoT platform, entities in the digital twin such as wires and conveyor belts are abstracted and simplified, and the color of Uarm also is changed to white. We write Coppeliasim embedded scripts for each device in the scene and make sure the digital twin hold the same full functionality as the real scene.

After realizing the same functionality, we need to make the virtual IoT platform behave same response as real IoT platform. As the experiment platform based on WoT architecture, the real IoT platform obeys the standard of WoT protocol and its server need to consume TD document. Therefore, we should write TD document for the virtual devices. To achieve the response ability same as the real devices, we will set the interaction method name and configure input and output parameters restrictions in the virtual TD same as the real TD.

Drew on TD document of real devices, it is not a complex work for us to rewrite functionality same Thing Description for virtual devices. For some difficult simulated parameters and actions such as the Uarm robot beep, which have no influence for whole motion of robot, our virtual devices TD do not include.

Figure 19: The digital twin of IoT remote lab

Then, on account of verifying the TD document of UR10 robot, we should design a digital twin which includes UR10 robot and its circumstance. In the IoT lab, the UR10 robot is placed on a yellow table and theoretically the UR10 robotic end effector can touch any device in the room. So the digital twin of UR10 robot must include the whole IoT lab.

In this situation, we will first set up the general outline of the room, and put the UR10 robot table in the predetermined position. Then we add two tables with lots of IoT devices. In the real IoT remote lab, the IoT platform is placed on the right table. So in this digital twin of IoT lab, we also integrate the digital twin of IoT platform.

For top table in the lab, referring to the real IoT devices at the top table in the lab, we make attempts to design digital twin of the light and the Pan-Tilt. In addition, in order to improve the similarity between the digital twin and real scene, we have also built corresponding models such as camera and computer server cabinet in our digital twin of IoT lab. These models do not have the matching functionality

In a word, so far we have successfully constructed two digital twins of the IoT platform and the IoT lab with full functionality and the same responsiveness. We should first verify the stability of our digital twin scenarios. Afterwards, we verify the accuracy of the robotic Thing Description generated automatically by our approach on the digital twin robot Uarm and UR10, which also provides a basis for modifying the TD of the robot in the real scene.

5

Related Work

5.1 Thing Description automatic generation based on different external metadata

With popularity of WoT architecture technology, many laboratories at universities and companies have established IoT platform fully based on WoT technology[15, 16, 17]. And for each devices in the WoT platform, developers need to humanly define behavior of devices and write specific TD document. Obviously, manually writing and modifying TD document is a long time-consuming and repetitive work for researchers.

The TD document is assembled by the fixed words and phrases arranged according to clear grammar rules. And same type devices always have similar properties and actions. These discover give us the probability of implementing automatically generating TD document by parsing external metadata from devices.

The most common metadata are the different line protocol standard from other IoT platform with similiar interactions rules. From the consideration of increasing scalability and compatibility between different technology standard, the proper convert method between different IoT architecture is required. Some researchers have proposed to realize the two-way conversion between TD document under the WoT architecture and other description files from different IoT platform[18, 19].

The ECHONET device objects(EDO) concept is committed to integrate fragmented smart home devices from different device manufacturers by standard data model[18, 20]. And Similar to TD in WoT, ECHONET API file also documents the interactive types and access methods of devices in the form of words and phrases.Therefore, based on the ECHONET Lite web API, some researchers design a conversion method to generate WoT Thing Description automatically instead of manual reconstruction.

For each interactions in the Thing Description (TD), it would provide a website link. By access this website link we can invoke the corresponding interactive method. But TD only is treated as descriptive file, developers still need finish code implementation for website link specified in the TD. And in some phenomena, the website link service has been supported by the RESTful web services. For the corresponding service links in RESTful, some researchers proposed how to convert these service links into interactive methods in TD[19].

Another common metadata is the TD document at the lower layer. In some cases, after we have already get TD document for each devices. Then We want to build a mash-up TD as system description to manage all devices in current platform[14, 21]. The method called Atomic Mashup Generator for the Web of Things(A-MaGe)[21] has been proposed by researchers, which could automatically analyze the properties from existed TD and integrate it to new TD at high layer.

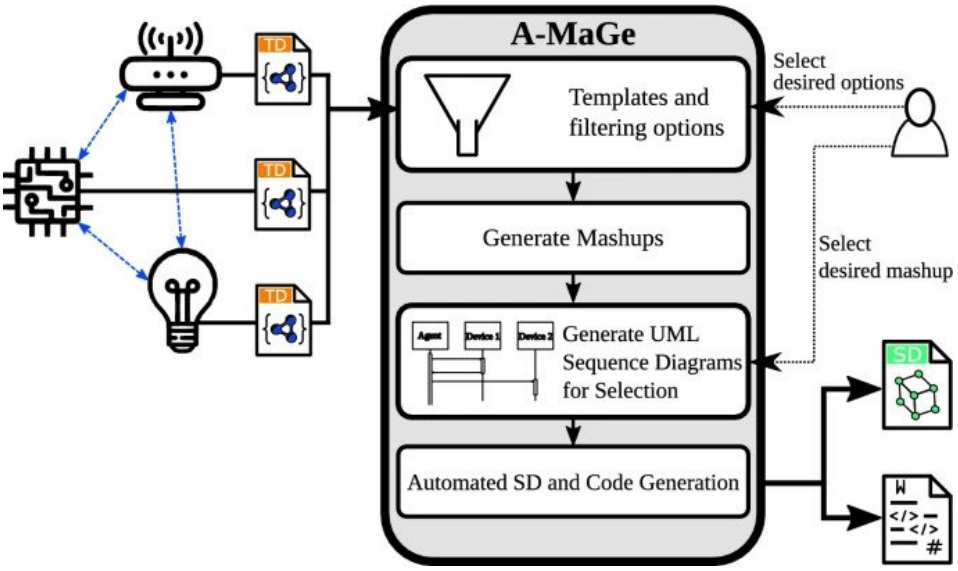


Figure 20: Architectural of A-MaGe method[21]

For a huge WoT platform system, we can set Mash-up Thing Description to manage multiple devices via one TD document. And for the Open Platform Communications(OPC) system, some researchers develop a method to automatically generate TD document by analyzing information model of system[22]. This TD document could also be updated by itself after new device is connected with OPC system.

5.2 Thing Description is consumed by digital twin

Thing Descriptions(TD) provides ways to interact with devices. In some cases the developer need to consider they define TD first before the hardware is accessible. Some researchers propose the concept of virtual-thing and present a heuristic method, which could simulate

the behavior of WoT entity based on current pre-defined TD[23]. In this paper, TD mainly records the state of devices and it shows how to effectively abstract interactions of devices through finite state machine.

The TD document could also be consumed by simulated device, which brings us the potentiality that we use the same TD to describe the digital twin of devices first. Some researchers design a Semantic digital twin method[24], which use TD to define interoperability of digital twin. By monitoring the digital twin devices, it could help us to evaluate the reliability and IoT devices in advance.

On the basis of having a large number of digital twin equipment, some scientists try to design the digital twin of the scene as a framework to manage the interaction of virtual devices and achieve the fully simulation to the behavior of real scene. The Virtual Environment of Things (VEoT)[25] is proposed, researchers design a hierarchical smart gateway realize the communications between the virtual environments and the real world.

Digital twin is not just virtualization of device or importing models into computers. The functional integration and abstraction of real devices for constructing a new systems are also important applications of digital twins. On the basis of physical things and devices, some researchers propose a concept called Shadow Things[26]. They try to abstract the TD document features of real devices via IoT gateway to generate new TD document as application layer for users. This Shadow Things also allows the no-code deployment for virtual devices.

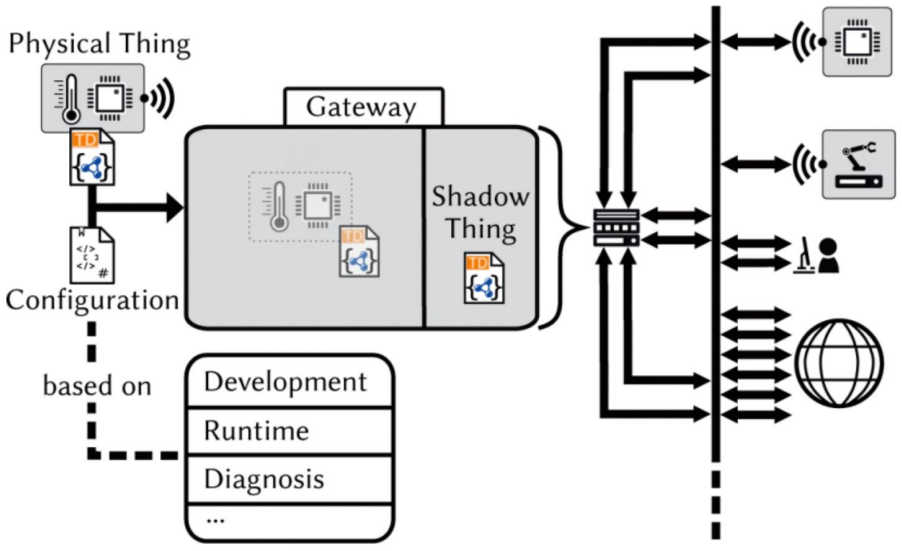


Figure 21: The Shadow Things in WoT architecture[26]

5.3 Researchs about robot simulation in the Coppeliasim

As a robot simulation software, coppeeliasim has attracted a lot of attention from researchers and engineers for its support in complex simulation scene design[11]. In the thesis, our main work about TD document and code implementation must be verified and evaluated with the support of this software. These papers, which also use this software tool, might have enlightening significance for our work.

For some robot prototypes with complex structures, evaluation on simulation software Coppeliasim at an early stage will be an effective and efficient method. For the target to analyze the dynamics properties of Spherical Parallel Manipulators, researchers first import the robot model into Coppeliasim[27] Then they make the 3D-printed prototype and compare the its performance with virtual robot.

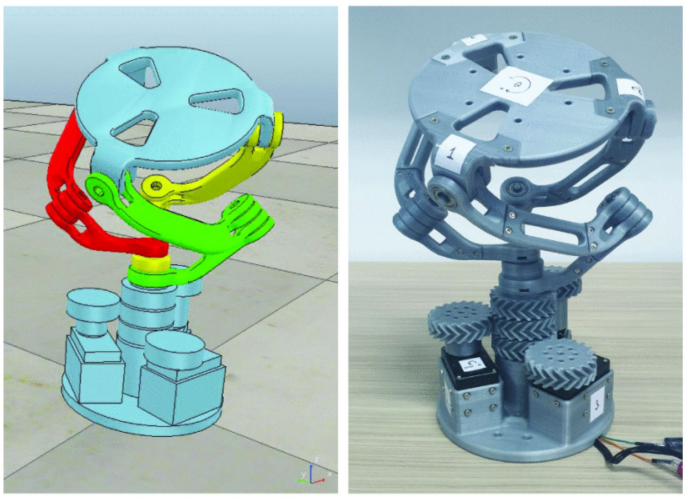


Figure 22: Spherical Parallel Manipulators in Coppeliasim[27]

In addition to directly interacting with Coppeliasim on the graphical interface, users can also control the virtual devices in the Coppeliasim scene through the remote API. This provides the potential possibility for us to interact with software under other frameworks. Some researchers make attempts to use Virtual Reality(VR) equipment to control the devices in the Coppeliasim environment[28].

6

Evaluation

6.1 Evaluate the digital twin of IoT lab scenes

In the moment we have two digital twin scenes: IoT platform and IoT lab. Both digital twin scenes possesses the full functionality and same response ability as the real scenario. For the target of evaluating the performance of two digital twin scenes. We want to design a work flow, which could be applied into the virtual scene and real scene at same time.

6.1.1 The metrics to evaluate the digital twin of IoT lab scenes

For the IoT platform part, We specify a series of action streams to handle the cube. We can observe the motion difference between two scenes to validate the reliability of digital twin. First the cube is taken out of the warehouse, transported by the robotic arm to the designated location for color detection. Finally the cube is returned to the warehouse after the color detection is successful. The entire cube process sequence refers to the template provided by the developer of the real IoT platform.

The IoT platform will execute the following steps:

1. The Dobot robot takes cube from the warehouse and places it on the right conveyor belt
2. The right conveyor belt moves forward and stops until the infrared sensor detect cube
3. The Uarm robot grabs the cube on the right conveyor belt and takes it away
4. The Uarm robot grabs the cube and moves to the color detection area.
5. Displays the current RGB value read by color sensor
6. The Uarm robot moves the cube to the left conveyor belt
7. The left conveyor belt moves backward and stops until the infrared sensor detect cube
8. The Dobot robot fetches the cube and returns it to warehouse

For the step 5, the reason we only debug and output the current RGB value is that the color sensor is sensitive to light flux. The RGB value output for the same color at different times will exist a large deviation.

For the metrics to evaluate the digital twin of the IoT lab scene, this virtual scene not only integrates the IoT platform digital twin and UR10 robot digital twin, but also it includes some other devices in the real IoT remote lab. So, we will give more detailed evaluation of UR10 robot digital twin in the further section. In this evaluation section, except for the action sequence of cube, we want to evaluate the performance and stability of other digital twin devices that are colored lights and the Pan-Tilt in Raspberry Pi board.

6.1.2 The code implementation and setup for the evaluation

The WoT server are responsible for managing the IoT devices in the platform. So we only need to write WoT client to make interactions with real scene and virtual scene. Because the name of interaction method in virtual TD and real TD is same, in theory we only need a single client template, which has ability to communicate with both real scene and its digital twin at the same time.

In the whole action sequence, we must call multiple WoT interaction methods such as **invokeAction** or **readProperty** to same device at the different time. So we plan to use Object-oriented programming(OOP) method and abstract the interaction method with devices as class things. After initialization of class by using credentials to access the corresponding links, we can get the instance of class.

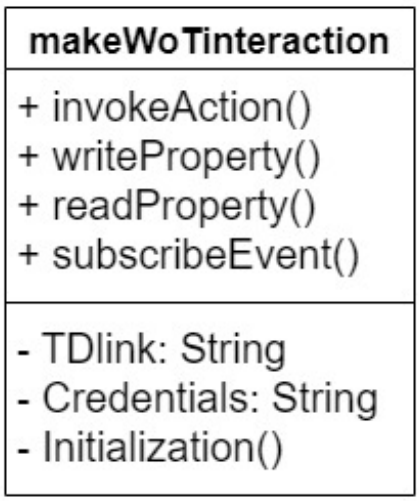


Figure 23: class diagram for wot client interactions

6.1.3 The evaluation results for the digital twin scenes

After finish the code preparation and setup, we begin to run the client script and monitor cube motion trajectory and the motion posture of the robotic arm in the real IoT platform and its digital twin. The whole cube process flow requires a few minutes, so we only record some pictures for the important motion of robot. The viewing angle of the virtual camera is limited, so the viewing angles of the two pictures are slightly different.

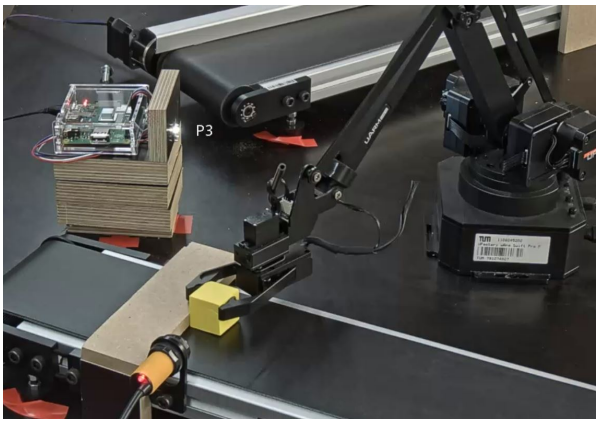


Figure 24: Clamp cube by real Uarm

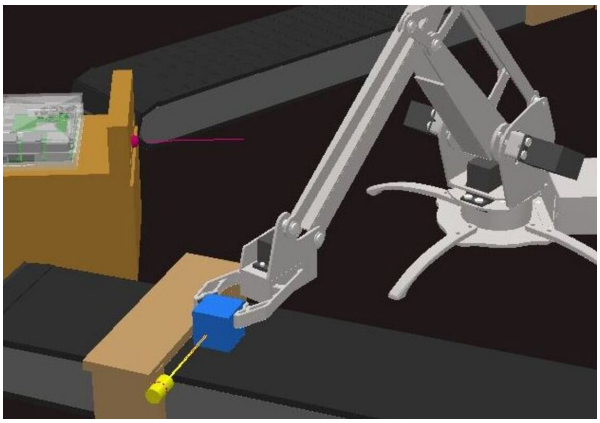


Figure 25: Clamp cube by virtual Uarm

The above pictures shows the Uarm robot pick the cube from the conveyor belt. In the code practice, we give same target position value to two WoT server. The two Uarm robot almost have the same movement posture.

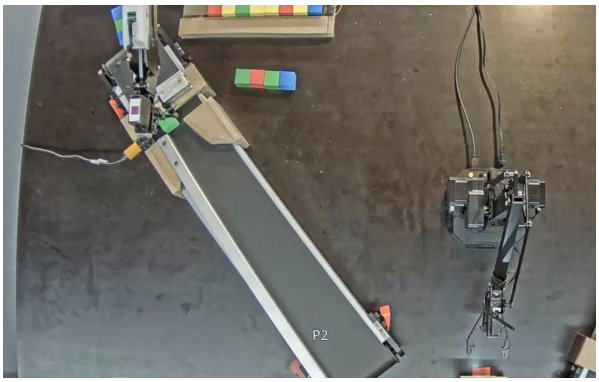


Figure 26: Cube return by real Dobot

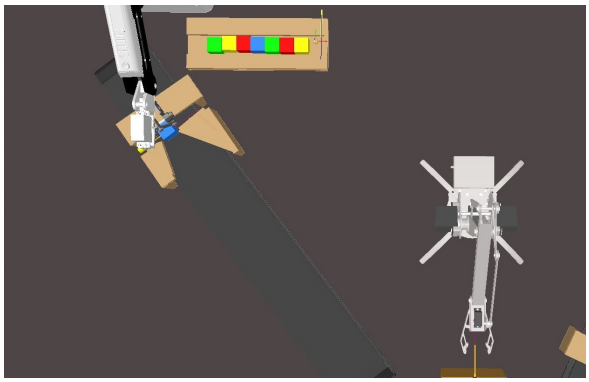


Figure 27: Cube return by virtual Dobot

We also record the pictures of Dobot motions, the Dobot robot will take the cube and put it back to warehouse. In the digital twin of Dobot, we simplify the Dobot structure and make it suitable for our common robot driver. But the digital twin of Dobot still has similiar kinematics behavior as the real Dobot.

From the figures we can see clearly that the robotic motion in virtual scene is approaching the real scene. And the other entities from the IoT lab scene also has the desired behavior. The

wonderful and satisfied demonstration proves the reliability of the digital twin scene built by us and its similarity to the real scene. With the help of the above digital twin scenario, we will continue to evaluate and verify the accuracy of the Uarm and UR10 manipulator Thing Description files and the trustworthiness of the method we designed for automatic inverse kinematics calculations.

In the digital twin of IoT lab, except for the digtial twin of UR10 robot, by replicating real devices in the lab, we also build two extra IoT device that is the virtual lights and the Pan-Tilt in Raspberry Pi. At early stage of attempt to aggregate WoT servers with Coppeliasim, we designed a virtual light, which are ported to our virtual scene of IoT lab.

Figure 28: Virtual light in the digital twin of IoT lab

Another device in our virtual IoT lab is the Pan-Tilt, which is directly installed on the Raspberry Pi board. The motivation we want to construct the digital twin for this kind of device is that it also owns the similar structure as the common robots we mentioned in the thesis.

The Pan-Tilt only has two links and two joints driven by mg-90 steering gear. Our automatic robotic TD generation method also could be applied in this devices. So, referring to the TD from the real Pan-Tilt device and on the basis of our automatic generated TD, we manually write the virtual TD for this digital twin. Now this digital twin of Pan-Tilt gains the same response ability as the real devices.

This device is not the most suitable prototype for validating the our robotic TD automatic generation algorithm, so we just modeled the Pan-Tilt approximately. And actually in the Thing Description of Pan-Tilt devices, the related motion actions and properties only work in the joint position coordinates. Besides, the real Pan-Tilt device could be attached with the camera. The camera is connected with Raspberry Pi and users can get pictures from the

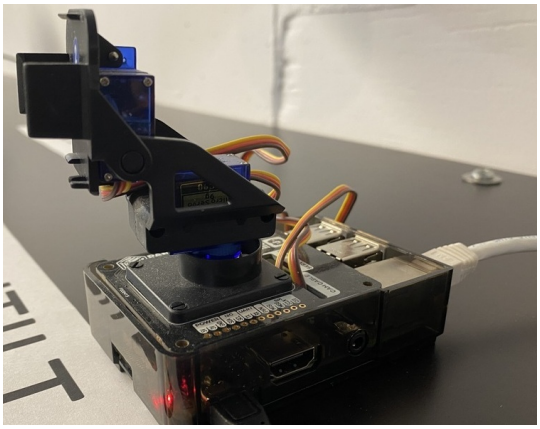


Figure 29: Real Pan-Tilt device

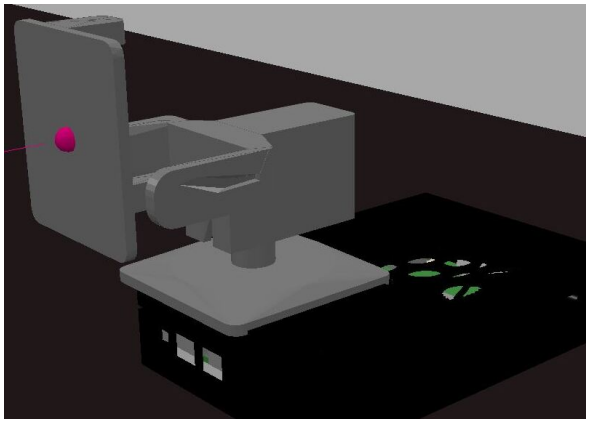


Figure 30: The digital twin of Pan-Tilt device

camera. In the virtual scene, we just equipped the device with a color sensor as replace.

In the future, for the students who want to make exercises or do their assignments in the devices from the IoT lab, they can check their code and script in our digital twin of IoT platform at first. This virtual scene could effectively help us to save some time on tidying cube and device maintenance in the real IoT platform.

6.2 Evaluation of the workspace represented by Uarm Thing Description

With the help of our digital twin scene, we associate our Uarm workspace generation script with Uarm digital twin and produce the required metadata. Based on these metadata, it is not a difficult work for us to use our algorithm to automatically generate the Uarm Thing Description. Before modifying the real Uarm TD according to the information from our virtual TD, we need to verify the accuracy of workspace in our TD for both Uarm robots.

6.2.1 The metrics to evalute the workspace reprsented by Uarm Thing Description

In the digital twin scene with full functionality and response ability same, the simplest way for us to verify our robotic TD is that we give a position coordinate to Uarm digital twin and check if the movement of Uarm exists collision at Coppeliasim scene. If the path is safe in the virtual Uarm, we directly deliver the same position coordinates to the real Uarm. By comparing the movement of the Uarm robot and its digital twin, we can determine whether the Uarm robot workspace generated by the digital twin scene can truly represent the workspace of the real Uarm robot.

In our metrics, it has an important requirements that position in the real Uarm robot could also

guide the virtual robot to same target with same motion. And it is worth noting the workspace description of two Uarm TD document is based on different coordinates and different distance units. So we must use our coordinates mapping algorithm in our thesis when we generate the Uarm TD document.

6.2.2 The code implementation and setup for the evaluation

In our original conception, the verification of Uarm robot motion is highly coupled with Coppeliasim simulation software. But for some target points that are obviously beyond the working range of the robot, the verification of these points in Coppeliasim will slow down the running efficiency of the program on the one hand. And on the other hand, these points may cause the robotic IK calculate to the infinite value and make the Coppeliasim software collapse.

Therefore, in the thesis, we propose a independent algorithm to check if the point sits in the polyhedron. And in addition, after we get a point in the robotic workspace, by using the accessible point table in our metadata, We will count how many collision points around this point, and try to estimate the probability of collision in this way without the help of Coppeliaism simulation software.

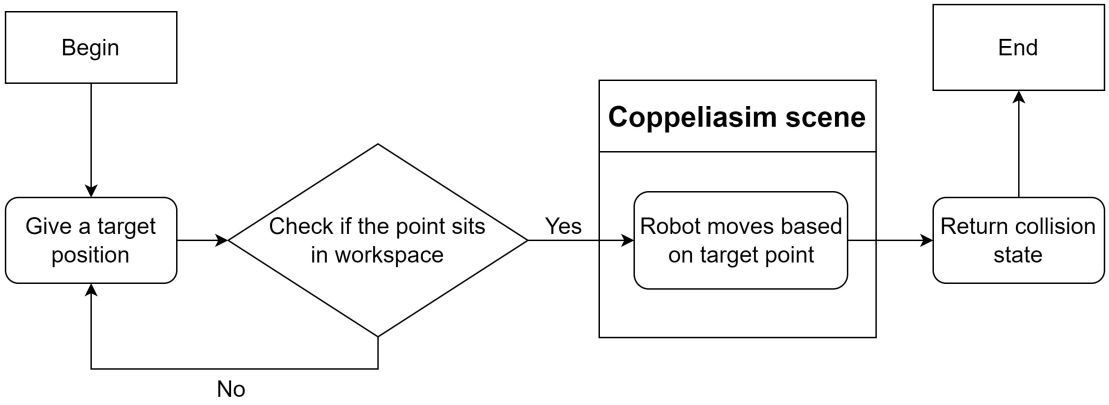


Figure 31: The workflow of checking point in workspace

6.2.3 The evaluation results for the workspace represented by the Uarm Thing Description

After the evaluation criteria and code preparation are completed, we begin to evaluate the accuracy of the workspace represented by our Uarm TD file. The first step for us is to use our approach to generate the Uarm TD and metadata. Then we compare our virtual Uarm TD with the real Uarm TD.

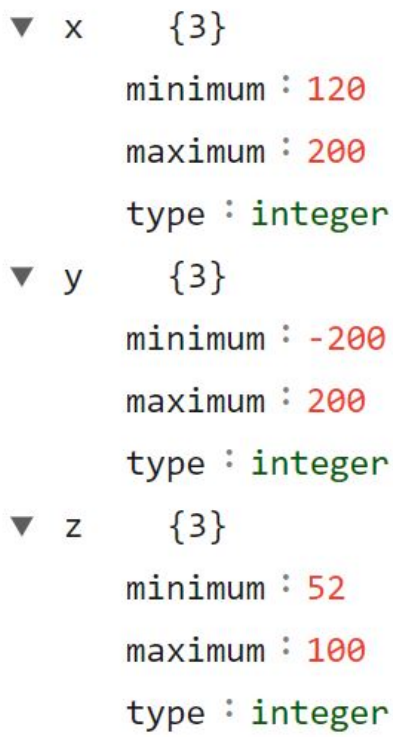


Figure 32: The Cartesian restrictions in real Uarm Thing Description

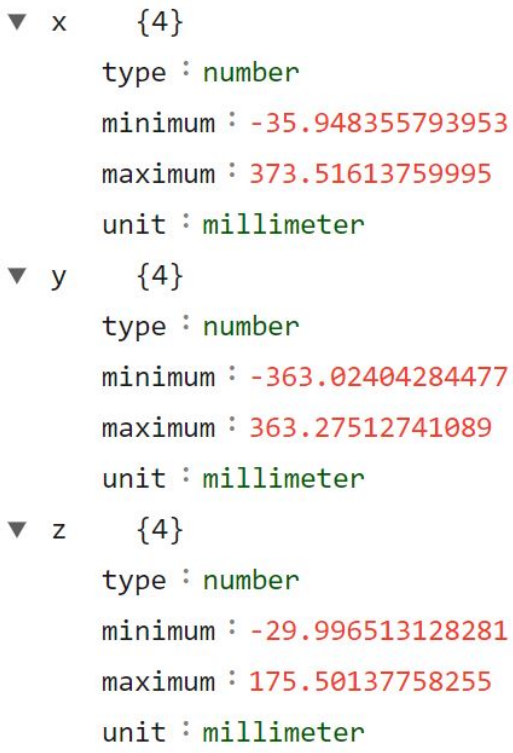


Figure 33: The Cartesian restrictions in virtual Uarm Thing Description

In the real TD from Uarm robot, developers knows it must occur collision when the Uarm robot work in its maximum range. Therefore, They just manually estimated an area that is completely safe without any collisions. This area actually only occupies a small part of the real Uarm robot working space.

Benefiting from our coordinate mapping algorithm, the Cartesian coordinate system in the TD file generated by the digital twin Uarm is consistent with the coordinate system of the real Uarm TD file. By observing the difference between the two TD files, we preliminarily confirm that our TD file generated by the digital twin robot can accurately represent the Uarm extreme workspace.

Obviously, Uarm robot has the high possibility of collision in the its entire work areas. For the intention of indicating obstacle area in Uarm workspace, the accessible point table in the metadata from our TD could help us to solve this problem.

Our accessible point table record huge amount of the point coordinates, which means that the digital twin robot has reached to this position in the simulation process of workspace generation. This table includes the robotic end effector collision state as well. In our digital twin scene, points without collisions are marked white and points with collisions are marked red. This method help us clearly to know the collision state of Uarm in its workspace.

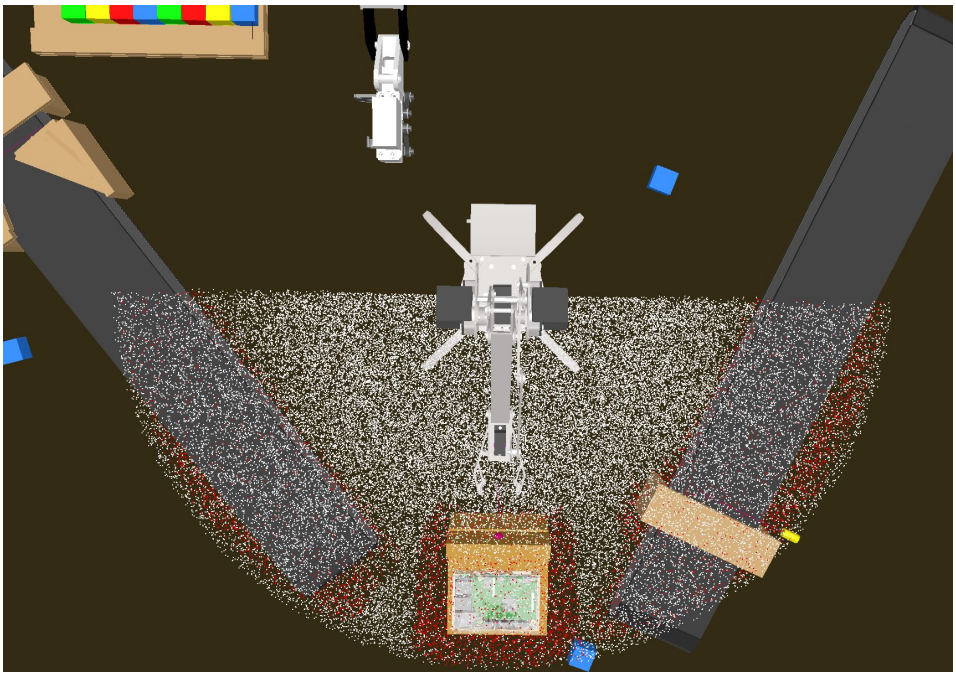


Figure 34: The collision state in Uarm digital twin workspace(top view)

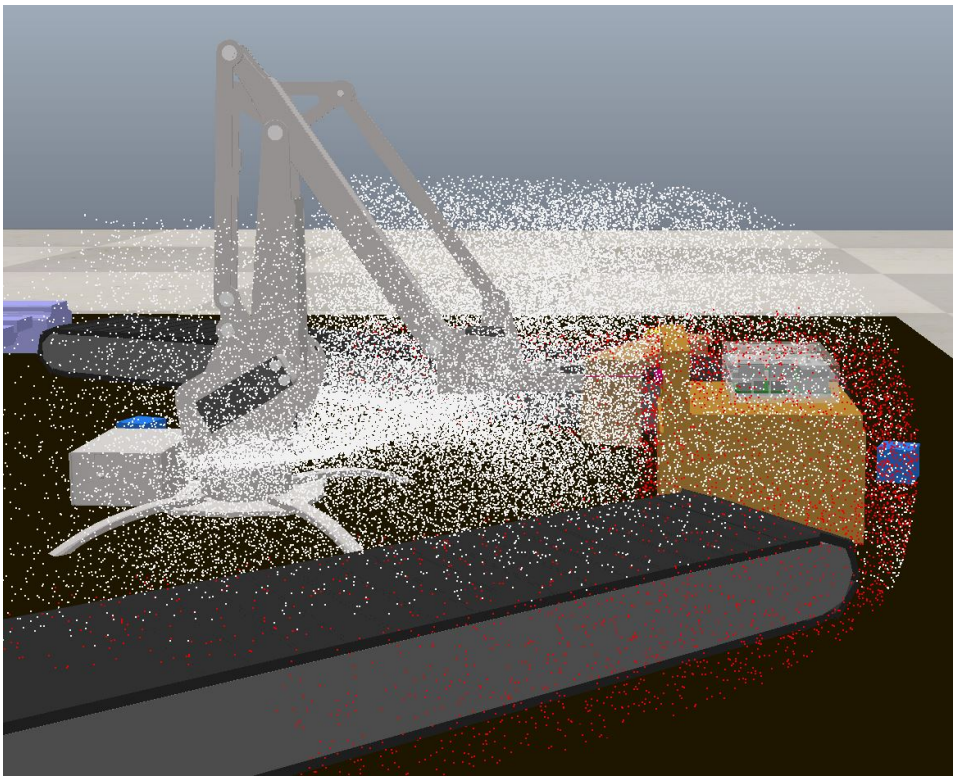


Figure 35: The collision state in Uarm digital twin workspace(side view)

As we imagined, when the Uarm end gripper is closed to the conveyor belt, color inspection box, the Uarm robotic arm is very prone to collisions. The points on the table help us to further confirm that the Uarm TD file we generated could accurately express the Uarm workspace.

In the last evaluation, in order to examine the response ability of virtual IoT platform, we have give many points, which is located in the initial work area from the real Uarm TD. It proves the motion of digital twin Uarm is quite similiar as the motion of real Uarm in the complete safety area.

Therefore, in this part evaluation, the target point we choose is above the color detection platform, and its coordinates are expressed as $\{"x" : 320, "y" : 0, "z" : 130\}$, which belongs to the coordinate system from real Uarm Thing Description. Observing the Uarm mechanical structure in the real scene and virtual scene, we finally successfully verify that the workspace from our digital twin Uarm robot TD could precisely represent the workspace for the real Uarm robot.

Figure 36: Real Uarm reach the target position

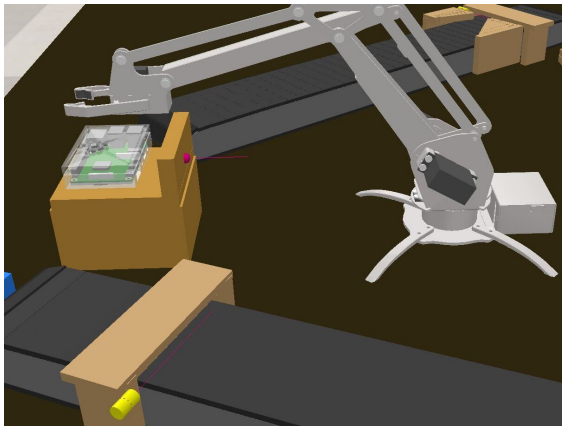


Figure 37: Virtual Uarm reach the target position

Based on the above evaluation results, we believe that the TD files generated from our Uarm digital twin can represent the workspace of real Uarm robots with millimeter-level accuracy. This also provides theoretical support for us to modify the TD file from the real Uarm robot to expand its workspace.

Actually, our TD document only includes the kinematics properties and workspace related metadata, which is more like the basic configuration file for robots. In the real robot TD, developers tend to design mash-up TD and they will integrate a series of actions into an interaction method such as **returnCube** in the TD file. In practice, our automatically generated TD can be used as the material for generating a mash-up TD, or other users can update their own robotic TD documents based on the metadata and robot workspace information contained in our robotic TD.

6.3 Evaluation of the inverse kinematic calculation from UR10 digital twin and the workspace represented by UR10 robot Thing Description

This is the final part of our evaluation and verification. We go back to verify the precision of UR10 robot workspace in the digital twin of IoT remote lab. UR10 robot belongs to the robot with common structures, which means that our automatic IK conversion and robotic workspace generation method both can be appropriate to this robot.

6.3.1 The metrics to evalute the inverse kinematic calculation from UR10 digital twin and the workspace represented by UR10 robot Thing Description

While verifying the accurate of workspace expressed by the UR10 robot TD, we also hope to evaluate our inverse kinematics conversion method generated by our UR10 robot digital twin at the same time. So, different from straightly calling the interactive method in the Uarm robot WOT server to control the robot movement, at beginning, we will control the digital twin of UR10 robot move to the target Cartesian position. The target position input of value must follow the restrictions in our TD document.

Then we can get the current joint position of the robotic digital twin and we will input this joint position value into the real UR10 robotic arm. After the real UR10 robot movement is finished, we will read the current Cartesian position value in the terminal of UR10 robot. Theoretically, the target coordinate value we input on the digital twin robot should be the same as the value displayed on the real UR10 robot terminal.

This target Cartesian position must be included in the workspace represented by our UR10 robot TD document. In other words, if we can prove the reliability of our automatic TD conversion method, we can potentially validate the accurate UR10 workspace represented by our virtual robot TD.

Besides, in the consideration of safety the protecting other devices in the lab, in our digital twin IoT lab, we add a virtual wall between the UR10 robot platform and other table with devices, which makes the generated workspace smaller than expected.

6.3.2 The code implementation and setup for the evaluation

For the robot with common structure, we don not need to manually write Coppeliasim embedded script for robotic workspace generation and add auxiliary joints compensation in the process of automatic IK conversion by hand. All we need to do generate the instance of class and call corresponding functions in our repository.

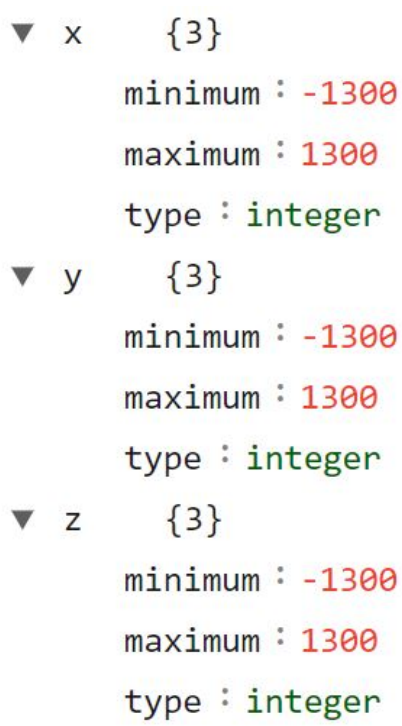


Figure 38: The Cartesian restrictions in real UR10 Thing Description

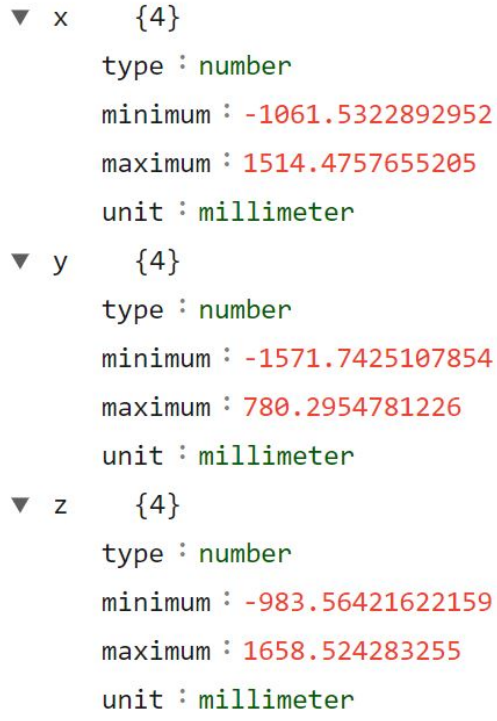


Figure 39: The Cartesian restrictions in virtual UR10 Thing Description

Using the digital twin of UR10 robot, we have successfully generated the corresponding robot Thing Description. Compared with the working range of the manipulator set artificially, the workspace of the robotic arm generated by our algorithm is more accurate.

Same with the mechanism in the Uarm robot, before formally transferring the target position value of the UR10 robot to the Coppeliasim scene, we will use the same algorithm to detect whether the point is in the robot's workspace and estimate the probability of collision event based on the collision status of the surrounding points.

In addition to the demand of performing mapping algorithm on points in different Cartesian coordinate systems, when we will start with operating the UR10 robot, we found that when the UR10 robot and its digital twin are in the same initial posture, the second and fourth joint angle of the real UR10 robot is negative 90 degrees. So when we use our robot motion control functions, we add a joint compensate for the function.

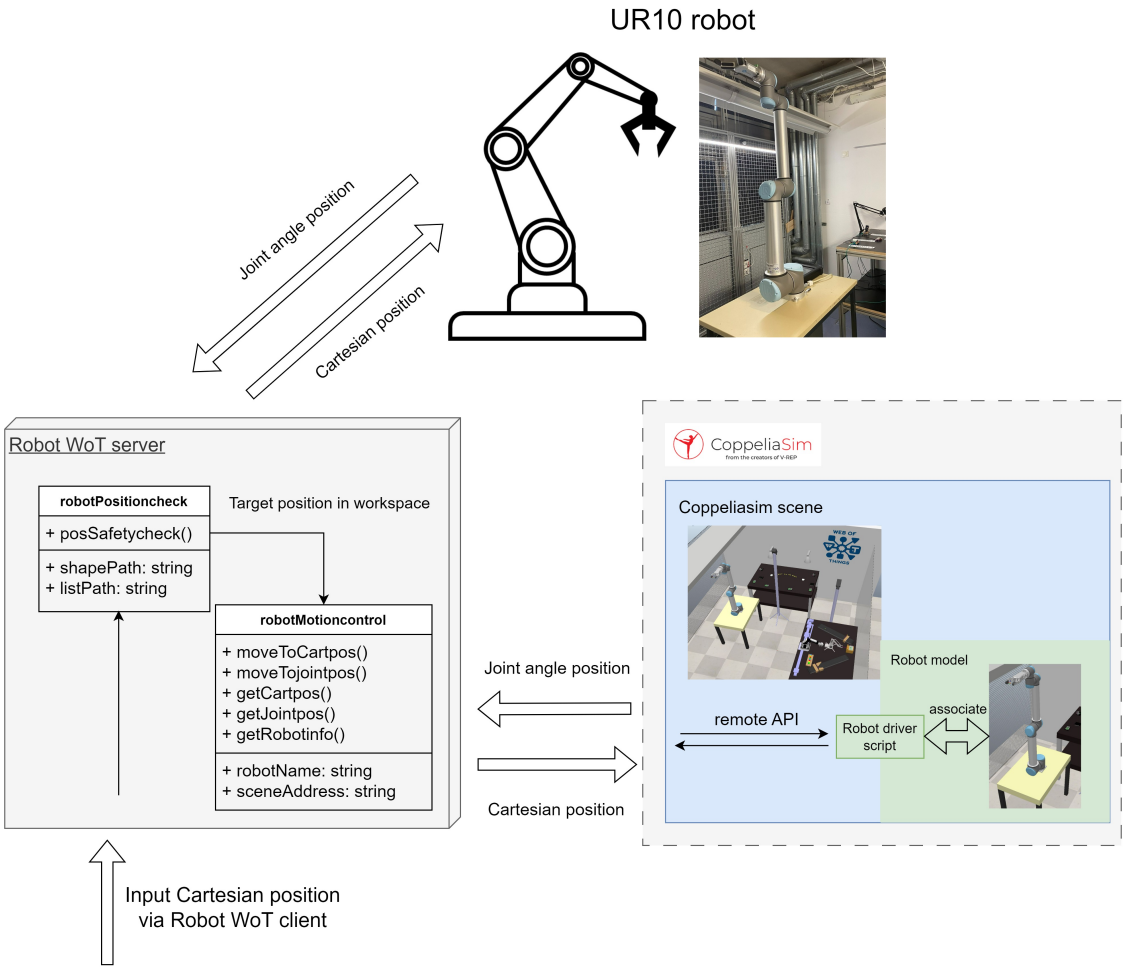


Figure 40: The workflow of digital twin UR10 robot verification

6.3.3 The evaluation results for the inverse kinematic calculation from UR10 digital twin and the workspace represented by UR10 robot Thing Description

According to the above verification workflow, we first selected several points in Cartesian coordinates. Then we input these coordinate points into our Typescript code. In a short time we obtained the current joint angle of UR10 robot from the digital twin scene. We give the current joint angle to the terminal of the UR10 robotic arm, and record the position of the end effector of the UR10 robotic arm. This position is displayed on the terminal screen.

Here we list a table, which records the some target position and its corresponding coordinates in the robotic digital twin and real robot. The unit of coordinates is millimeter. Obviously, compared with the current target position in the real robot, our digital twin of robot and scene can reach millimeter-level accuracy. The related figures of the robotic posture in the real robot and its digital twin are recorded as follows.

Target position List		
Target position in the Cartesian system	Current Cartesian position in the virtual robot	Current Cartesian position in the real robot
(500, -890, 1000)	(500.08, -890.09, 999.65)	(495.94, -882.35, 990.96)
(400, -490, -200)	(399.87, -489.82, -199.97)	(401.23, -495.34, -203.35)
(780, 500, 900)	(781.24, 500.54, 898.42)	(780.69, 487.71, 897.65)

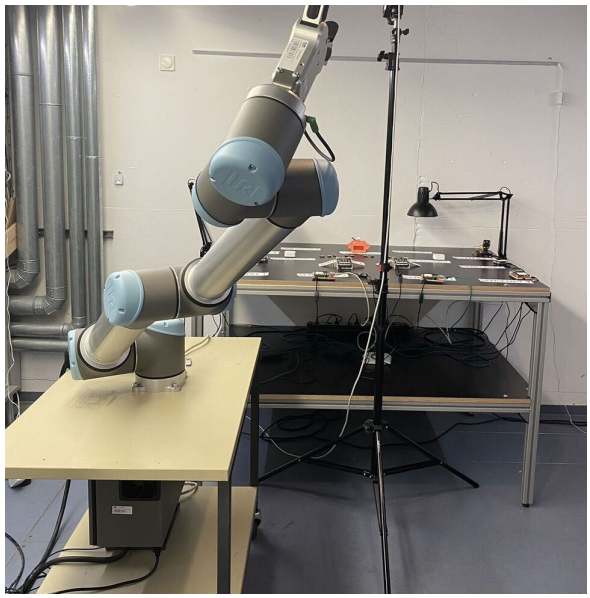


Figure 41: The posture of real UR10 robot at the first position

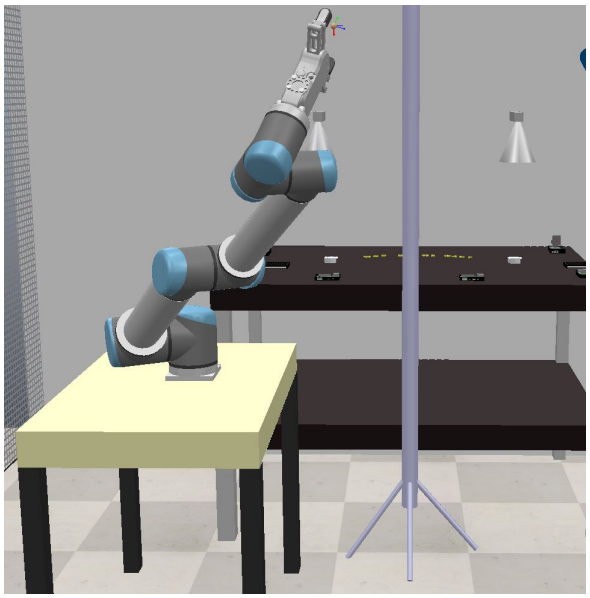


Figure 42: The posture of virtual UR10 robot at the first position

Figure 43: The posture of real UR10 robot at the second position

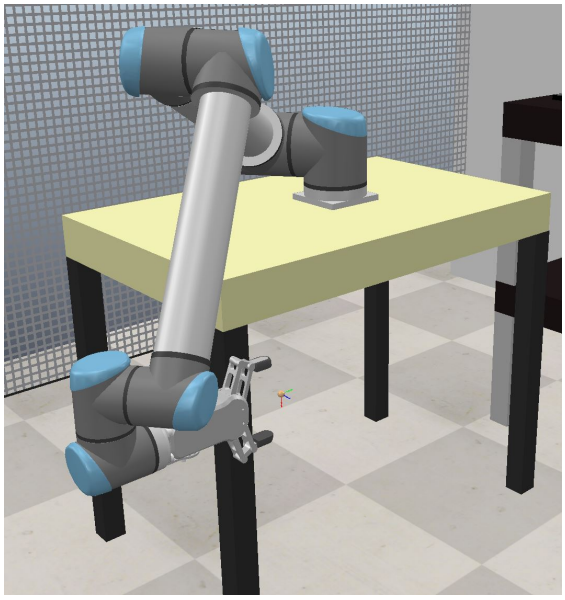


Figure 44: The posture of virtual UR10 robot at the second position

Figure 45: The posture of real UR10 robot at the third position

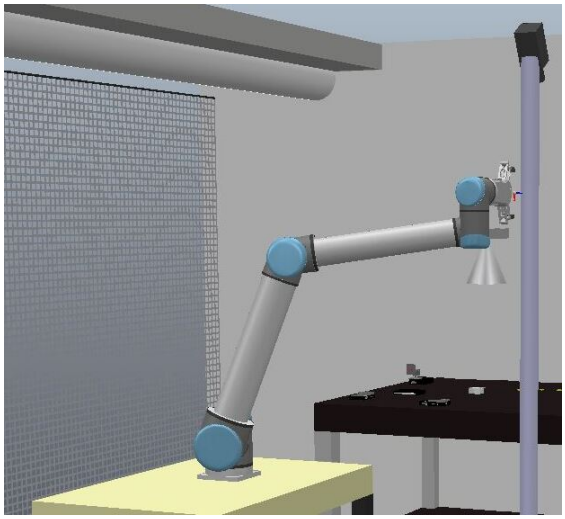


Figure 46: The posture of virtual UR10 robot at the third position

By comparing the input coordinates in our code and the output coordinates from the UR10 terminal, as well as the posture of the UR10 robot in the real scene and the digital twin scene. We confirm that our automatic IK conversion algorithm has a high reliability, and the generated TD file of UR10 robot can accurately represent the current workspace of robot and its kinematics properties.

7

Discussion

In the thesis, we want to design a structure of TD document suitable for the most type robots in the market and implement a method that could automatically generate this type of robotic TD document to accurately rebuild the workspace of robot. In order to have a clear thought for our goal, we read some papers about the digital twin and automatic TD document generation. Inspired by the ideas and algorithms in these literature, we make some improvements in these fields, which is helpful for us to solve our research problem.

In many literature, the creation of digital twin prototypes often precedes the manufacture of actual devices. Researchers will first verify the feasibility of their algorithms on digital twins, and then apply their proposed algorithms or new protocols to actual devices. And it exists many software at present, which could be used as digital twin to simulate the behavior of real entities. We do some attempts and exploration in designing digital twin scene which have same response ability as the real existing scenes based on the Coppeliasim software.

In related work about TD document automatic generation, researchers and scientists focus more attention to achieve descriptive file automatic conversion between WoT architecture and other IoT protocols. And Some papers deal with how to use existing TD document to generate new TD document as system descriptions to manage the devices. There are only a few research about how to utilize digital twin to generate TD.

Therefore, we do some research about robotic digital twin with similar dynamics and kinematics properties as real robot. On the basis of extracted features from current robotic digital twin, we propose a method to utilize these information to generation the TD document for the corresponding robots.

8

Conclusion

During the whole thesis, we successfully design the robotic TD, which includes the workspace of robot and related metadata. This TD document could consumed by the most type of robots. We also indicate what kind of metadata is crucial to generate our TD document.

In the robot simulation framework, for the first time, we realized the integration and interaction of WoT library with virtual scene in the Coppeliasim software. And we have also written several Coppeliasim embedded scripts that can be applied to automatically generate the inverse kinematic conversion and the metadata related with workspace representation for the most of robots with common structures. Furthermore, to avoid collapse of robot models exported from URDF, we propose a way to automatically parse robot dynamic parameters and load robot model from URDF to the Coppeliasim software.

Then in the evaluation part, referring to the behavior of real IoT platforms, we effectively construct two fully functional and same responsive digital twin scenes. We design a series of action steps to show the reliability of our digital twin scenes. Our coordinates mapping algorithm converts the coordinate system of the virtual scene to the real world, which makes the TD file created by the virtual digital twin can be directly consumed by the real robot. It essentially reduce the time required for users to be familiar with our digital twin.

In digital twin of of Uarm and UR10 robot, by evaluating the corresponding digital twin scene we finally prove the accurate of our TD document and our inverse kinematic conversion is fitting for both real robots and virtual robots at the same time. After the thesis, these two scenes can also be useful for the further research.

Although, more than 80% of the robots on the market have similar mechanical structures that their all joints are controllable. For the robot with assistant joints and complex structure such as Uarm and Dobot, we make some simplification and only record the kinematics properties of main joint in our TD. And our automatic robotic workspace and related metadata genera-

tion algorithm is only acceptable by the robot with common structure. In order to generate workspace and essential metadata in these rest 20% special robots, we need to manually add angle compensate to each assistant joint at Coppeliasim scripts.

In the future, we plan to do more attempts on how to automatically generate TD document for the robot with complex structure. Besides, at present the digital twin scenes manufactured by us are all built by hand. In order to improve the accuracy of the digital twin scene, we fine-tuned it many times, which took up a large part of time in the thesis. A method that can automatically generate high-precision digital twin scenes by scanning real-world scenes would be very helpful for further accuracy improvement of digital twin scene.

Reviewing the evaluation results and our contributions, We confirm that the TD document and workspace generation algorithm designed by us can be befitting to most industrial robots. In the industrial practices, engineers or developers can use our method to automatically generate the robotic TD from the digital twin of robot at Coppeliasim software. They can also check the safety of robotic motions in our digital twin at first before it is executed in the real robot.

In addition, when maintenance of laboratory equipment is mandatory in some situations, the digital twin of IoT platform produced by us can effectively simulate the behavior of real device, which avoids temporary interruption of research and teaching. Using our digital twin scenes in some teaching tasks can reduce the time spent by the lecturer on real devices.

A

Appendix

A.1 The Thing Description for Uarm robot with special structure

```
{
  "title": "coppeliasim_virtualrobot_uarm",
  "id": "urn:dev:ops:32473-virtual-uarm",
  "description": "things description for uarm in current coppeliasim scene",
  "links": [
    {
      "href": "http://localhost:4000/uarm_robot/uarm_shape.stl",
      "type": "model/stl",
      "rel": "workspace"
    },
    {
      "href": "http://localhost:4000/uarm_robot/uarm_data_point.csv",
      "type": "text/csv",
      "rel": "dataset-points"
    },
    {
      "href": "http://localhost:4000/uarm_robot/Uarm_TD_verification.ttt",
      "type": "application/octet-stream",
      "rel": "coppeliasim scene"
    }
  ],
  "@context": ["https://www.w3.org/2019/wot/td/v1", { "@language": "en" }],
  "@type": "Thing",
  "security": "nosec_sc",
  "securityDefinitions": { "nosec_sc": { "scheme": "nosec" } },
  "properties": {
    "getJointposition": {
      "title": "get position of each joint",
```

```
    "description": "maxItems,minItems,minimum,maximum are decided by
↪ sprecific virtual robot",
    "type": "object",
    "properties": {
      "joint1": {
        "type": "number",
        "unit": "deg",
        "minimum": 0,
        "maximum": 180.00000500895786
      },
      "joint2": {
        "type": "number",
        "unit": "deg",
        "minimum": 0,
        "maximum": 135.0000003416235
      },
      "joint3": {
        "type": "number",
        "unit": "deg",
        "minimum": 42.00000116875664,
        "maximum": 142.00000167478154
      },
      "joint4": {
        "type": "number",
        "unit": "deg",
        "minimum": 0,
        "maximum": 180.00000500895786
      }
    },
    "readOnly": true,
    "observable": true,
    "forms": [
      {
        "href": "http://localhost:8090/coppeliasim_virtualrobot_uarm/
↪ properties/getJointposition",
        "contentType": "application/json",
        "op": ["readproperty"],
        "htv:methodName": "GET"
      }
    ]
  },
  "getCartesianposition": {
    "title": "get cartesian position of robotic end-effector",
    "description": "minimum,maximum is decided by specific robot",
    "type": "object",
    "properties": {
      "x": {
        "type": "number",
        "minimum": -35.948355793953,
```

```
          "maximum": 373.51613759995,
          "unit": "millimeter"
        },
        "y": {
          "type": "number",
          "minimum": -363.02404284477,
          "maximum": 363.27512741089,
          "unit": "millimeter"
        },
        "z": {
          "type": "number",
          "minimum": -29.996513128281,
          "maximum": 175.50137758255,
          "unit": "millimeter"
        }
      },
      "observable": true,
      "readOnly": true,
      "forms": [
        {
          "href": "http://localhost:8090/coppeliasim_virtualrobot_uarm/
  ↪ properties/getCartesianposition",
          "contentType": "application/json",
          "op": ["readproperty"],
          "htv:methodName": "GET"
        }
      ]
    }
  },
  "actions": {
    "moveTojointPosition": {
      "title": "let robot move according to joint position",
      "description": "maxItems,minItems,minimum,maximum are decided by
  ↪ sprecific virtual robot",
      "input": {
        "type": "object",
        "properties": {
          "joint1": {
            "type": "number",
            "unit": "deg",
            "minimum": 0,
            "maximum": 180.00000500895786
          },
          "joint2": {
            "type": "number",
            "unit": "deg",
            "minimum": 0,
            "maximum": 135.0000003416235
          },
```

```
        "joint3": {
          "type": "number",
          "unit": "deg",
          "minimum": 42.00000116875664,
          "maximum": 142.00000167478154
        },
        "joint4": {
          "type": "number",
          "unit": "deg",
          "minimum": 0,
          "maximum": 180.00000500895786
        }
      }
    },
    "forms": [
      {
        "href": "http://localhost:8090/coppeliasim_virtualrobot_uarm/actions
↪ /moveTojointPosition",
        "contentType": "application/json",
        "op": ["invokeaction"],
        "htv:methodName": "PUT"
      }
    ]
  },
  "moveTocartesianPosition": {
    "title": "make robot move according to cartesian position",
    "description": "minimum,maximum are decided by sprecific virtual robot",
    "input": {
      "type": "object",
      "properties": {
        "x": {
          "type": "number",
          "minimum": -35.948355793953,
          "maximum": 373.51613759995,
          "unit": "millimeter"
        },
        "y": {
          "type": "number",
          "minimum": -363.02404284477,
          "maximum": 363.27512741089,
          "unit": "millimeter"
        },
        "z": {
          "type": "number",
          "minimum": -29.996513128281,
          "maximum": 175.50137758255,
          "unit": "millimeter"
        }
      }
```

```
      },
      "forms": [
        {
          "href": "http://localhost:8090/coppeliasim_virtualrobot_uarm/actions
    ↪ /moveTocartesianPosition",
          "contentType": "application/json",
          "op": ["invokeaction"],
          "htv:methodName": "PUT"
        }
      ]
    }
  }
}
```

Listing A.1: The Thing Description example for Uarm robot

A.2 The Thing Description for UR10 robot with common structure

```
{
  "title": "coppeliasim_virtualrobot_UR10",
  "id": "urn:dev:ops:32473-virtual-UR10",
  "description": "The TD for the virtual ur10 robot",
  "links": [
    {
      "href": "http://localhost:4000/ur10_robot/UR10_shape.stl",
      "type": "model/stl",
      "rel": "workspace"
    },
    {
      "href": "http://localhost:4000/ur10_robot/UR10_data_point.csv",
      "type": "text/csv",
      "rel": "dataset-points"
    },
    {
      "href": "http://localhost:4000/ur10_robot/UR10_TD_verification.ttt",
      "type": "application/octet-stream",
      "rel": "coppeliasim scene"
    }
  ],
  "@context": ["https://www.w3.org/2019/wot/td/v1", { "@language": "en" }],
  "@type": "Thing",
  "security": "nosec_sc",
  "securityDefinitions": { "nosec_sc": { "scheme": "nosec" } },
  "properties": {
    "getJointposition": {
```

```
28      "title": "get position of each joint",
29      "description": "maxItems,minItems,minimum,maximum are decided by
    ↪ sprecific virtual robot",
30      "type": "object",
31      "properties": {
32        "joint1": {
33          "type": "number",
34          "unit": "deg",
35          "minimum": -360.00001001791,
36          "maximum": 360.00001001791
37        },
38        "joint2": {
39          "type": "number",
40          "unit": "deg",
41          "minimum": -360.00001001791,
42          "maximum": 360.00001001791
43        },
44        "joint3": {
45          "type": "number",
46          "unit": "deg",
47          "minimum": -360.00001001791,
48          "maximum": 360.00001001791
49        },
50        "joint4": {
51          "type": "number",
52          "unit": "deg",
53          "minimum": -360.00001001791,
54          "maximum": 360.00001001791
55        },
56        "joint5": {
57          "type": "number",
58          "unit": "deg",
59          "minimum": -360.00001001791,
60          "maximum": 360.00001001791
61        },
62        "joint6": {
63          "type": "number",
64          "unit": "deg",
65          "minimum": -360.00001001791,
66          "maximum": 360.00001001791
67        }
68      },
69      "readOnly": true,
70      "observable": true,
71      "forms": [
72        {
73          "href": "http://localhost:8001/coppeliasim_virtualrobot_UR10/
    ↪ properties/getJointposition",
74          "contentType": "application/json",
```

```
          "op": ["readproperty"],
          "htv:methodName": "GET"
        }
      ]
    },
    "getCartesianposition": {
      "title": "get cartesian position of robotic end-effector",
      "description": "minimum,maximum is decided by specific robot",
      "type": "object",
      "properties": {
        "x": {
          "type": "number",
          "minimum": -1061.5322892952,
          "maximum": 1514.4757655205,
          "unit": "millimeter"
        },
        "y": {
          "type": "number",
          "minimum": -1571.7425107854,
          "maximum": 780.2954781226,
          "unit": "millimeter"
        },
        "z": {
          "type": "number",
          "minimum": -983.56421622159,
          "maximum": 1658.524283255,
          "unit": "millimeter"
        }
      },
      "observable": true,
      "readOnly": true,
      "forms": [
        {
          "href": "http://localhost:8001/coppeliasim_virtualrobot_UR10/
  ↪ properties/getCartesianposition",
          "contentType": "application/json",
          "op": ["readproperty"],
          "htv:methodName": "GET"
        }
      ]
    }
  },
  "actions": {
    "moveTojointPosition": {
      "title": "let robot move according to joint position",
      "description": "maxItems,minItems,minimum,maximum are decided by
  ↪ sprecific virtual robot",
      "input": {
        "type": "object",
```

```
      "properties": {
        "joint1": {
          "type": "number",
          "unit": "deg",
          "minimum": -360.00001001791,
          "maximum": 360.00001001791
        },
        "joint2": {
          "type": "number",
          "unit": "deg",
          "minimum": -360.00001001791,
          "maximum": 360.00001001791
        },
        "joint3": {
          "type": "number",
          "unit": "deg",
          "minimum": -360.00001001791,
          "maximum": 360.00001001791
        },
        "joint4": {
          "type": "number",
          "unit": "deg",
          "minimum": -360.00001001791,
          "maximum": 360.00001001791
        },
        "joint5": {
          "type": "number",
          "unit": "deg",
          "minimum": -360.00001001791,
          "maximum": 360.00001001791
        },
        "joint6": {
          "type": "number",
          "unit": "deg",
          "minimum": -360.00001001791,
          "maximum": 360.00001001791
        }
      }
    },
    "forms": [
      {
        "href": "http://localhost:8001/coppeliasim_virtualrobot_UR10/actions
↪ /moveTojointPosition",
        "contentType": "application/json",
        "op": ["invokeaction"],
        "htv:methodName": "PUT"
      }
    ]
  },
```

```
    "moveTocartesianPosition": {
      "title": "make robot move according to cartesian position",
      "description": "minimum,maximum are decided by sprecific virtual robot",
      "input": {
        "type": "object",
        "properties": {
          "x": {
            "type": "number",
            "minimum": -1061.5322892952,
            "maximum": 1514.4757655205,
            "unit": "millimeter"
          },
          "y": {
            "type": "number",
            "minimum": -1571.7425107854,
            "maximum": 780.2954781226,
            "unit": "millimeter"
          },
          "z": {
            "type": "number",
            "minimum": -983.56421622159,
            "maximum": 1658.524283255,
            "unit": "millimeter"
          }
        }
      },
      "forms": [
        {
          "href": "http://localhost:8001/coppeliasim_virtualrobot_UR10/actions
    ↪ /moveTocartesianPosition",
          "contentType": "application/json",
          "op": ["invokeaction"],
          "htv:methodName": "PUT"
        }
      ]
    }
  }
}
```

Listing A.2: The Thing Description example for UR10 robot

A.3 The WoT client based on Typescript

```
import {makeWoTinteraction} from "./clientClass";
function delay(ms: number) {
    return new Promise( resolve => setTimeout(resolve, ms) );
}
```

```
// virtual devices url
let sensor1URL = "http://localhost:9000/virtualinfraredsensor1";
let sensor2URL = "http://localhost:9000/virtualinfraredsensor2";
let conveyor1URL = "http://localhost:9000/virtualconveyorbelt1";
let conveyor2URL = "http://localhost:9000/virtualconveyorbelt2";
let uarmURL = "http://localhost:9000/virtualuarm";
let dobotURL = "http://localhost:9000/virtualdobot";
let colorURL = "http://localhost:9000/virtualcolorsensor";

async function main() {
    // WoT client init
    let sensor1 = new makeWoTinteraction(sensor1URL);
    let sensor2 = new makeWoTinteraction(sensor2URL);
    let conveyor1 = new makeWoTinteraction(conveyor1URL);
    let conveyor2 = new makeWoTinteraction(conveyor2URL);
    let uarm = new makeWoTinteraction(uarmURL);
    let dobot = new makeWoTinteraction(dobotURL);
    let color = new makeWoTinteraction(colorURL);

    let P1 = {
        "x":192,
        "y":192,
        "z":87
    };
    let P4 = {
        "x":192,
        "y":192,
        "z":52
    };
    let P2 = {
        "x":200,
        "y":-200,
        "z":90
    };
    let P3 = {
        "x":180,
        "y":0,
        "z":60
    };
    let P5 = {
        "x":200,
        "y":0,
        "z":80
    };
    let P6 = {
        "x":200,
        "y":-200,
        "z":70
    };
```

```
54      await dobot.invokeAction("getCube");
55      await delay(35000);
56      await conveyor2.invokeAction("startBeltForward");
57      while (true){
58          if (await sensor2.readProperty("objectPresence") == true){
59              await delay(500);
60              await conveyor2.invokeAction("stopBelt");
61              break
62          }
63          await delay(800);
64      }
65      await uarm.invokeAction("gripOpen");
66      await uarm.invokeAction("goTo",P1);
67      await delay(6000);
68      await uarm.invokeAction("goTo",P4);
69      await delay(6000);
70      await uarm.invokeAction("gripClose");
71      await delay(4000);
72      await uarm.invokeAction("goTo",P1);
73      await delay(4000);
74      await uarm.invokeAction("goTo",P3);
75      await delay(4000);
76      console.log("current color RGB is");
77      console.log(await color.readProperty("color"));
78      await delay(2000);
79      await uarm.invokeAction("goTo",P2);
80      await delay(4000);
81      await uarm.invokeAction("goTo",P6);
82      await delay(4000);
83      await uarm.invokeAction("gripOpen");
84      await delay(4000);
85      await uarm.invokeAction("goTo",P5);
86      await delay(4000);
87      await conveyor1.invokeAction("startBeltBackward");
88      while (true){
89          if (await sensor1.readProperty("objectPresence") == true){
90              await delay(600);
91              await conveyor1.invokeAction("stopBelt");
92              break
93          }
94          await delay(1500);
95      }
96      await delay(4000);
97      await dobot.invokeAction("returnCube");
98  }
99  main();
```

Listing A.3: The WoT client template for action steps in evaluation part

Bibliography

[1] S. Madakam, V. Lake, V. Lake, V. Lake, *et al.*, "Internet of Things (IoT): A literature review," *Journal of Computer and Communications*, vol. 3, no. 05, p. 164, 2015. cited on p. 5

[2] D. Zeng, S. Guo, and Z. Cheng, "The Web of Things: A Survey," *J. Commun.*, vol. 6, no. 6, pp. 424–438, 2011. cited on p. 5

[3] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, "From the internet of things to the web of things: Resource-oriented architecture and best practices," *Architecting the Internet of things*, pp. 97–129, 2011. cited on p. 5

[4] M. Lagally, R. Matsukura, M. McCool, K. Toumura, K. Kajimoto, T. Kawaguchi, and M. Kovatsch, "Web of things (wot) architecture 1.1," Jan 2023. cited on p. 5

[5] S. Kaebisch, M. McCool, E. Korkan, T. Kamiya, V. Charpenay, and M. Kovatsch, "Web of things (wot) thing description 1.1," Jan 2023. cited on p. 7

[6] P. Augustine, "The industry use cases for the digital twin idea," in *Advances in Computers*, vol. 117, pp. 79–105, Elsevier, 2020. cited on p. 7

[7] F. Tao, M. Zhang, and A. Y. C. Nee, *Digital twin driven smart manufacturing*. Academic Press, 2019. cited on p. 7

[8] F. Tao, H. Zhang, A. Liu, and A. Y. Nee, "Digital twin in industry: State-of-the-art," *IEEE Transactions on industrial informatics*, vol. 15, no. 4, pp. 2405–2415, 2018. cited on p. 7

[9] R. Minerva, G. M. Lee, and N. Crespi, "Digital twin in the IoT context: A survey on technical features, scenarios, and architectural models," *Proceedings of the IEEE*, vol. 108, no. 10, pp. 1785–1824, 2020. cited on p. 8

[10] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel, "On the shape of a set of points in the plane," *IEEE Transactions on information theory*, vol. 29, no. 4, pp. 551–559, 1983. cited on p. 10, 21

[11] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *2013 IEEE/RSJ international conference on intelligent robots and systems*, pp. 1321–1326, IEEE, 2013. cited on p. 16, 40

[12] Dobot, "Introduction to the m1 pro collaborative scara robot." https://www.dobot-robots.com/products/dobot-series/m1-pro.html. Accessed: 2023-04-16. cited on p. 18

[13] R. Ierusalimschy, *Programming in lua*. Roberto Ierusalimschy, 2006. cited on p. 25

[14] A. Kast, E. Korkan, S. Käbisch, and S. Steinhorst, "Web of Things System Description for Representation of Mashups," in *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, pp. 1–8, IEEE, 2020. cited on p. 33, 38

[15] S. Huang, L. Li, H. Cai, B. Xu, G. Li, and L. Jiang, "A configurable wot application platform based on spatiotemporal semantic scenarios," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 1, pp. 123–135, 2017. cited on p. 37

[16] R. Pastor-Vargas, L. Tobarra, A. Robles-Gómez, S. Martin, R. Hernández, and J. Cano, "A wot platform for supporting full-cycle iot solutions from edge to cloud infrastructures: A practical case," *Sensors*, vol. 20, no. 13, p. 3770, 2020. cited on p. 37

[17] L. Sciullo, I. D. R. Zyrianoff, A. Trotta, and M. Di Felice, "Wot micro servient: Bringing the w3c web of things to resource constrained edge devices," in *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 161–168, IEEE, 2021. cited on p. 37

[18] V. C. Pham and Y. Tan, "Towards automated generation of data models for the echonet lite protocol," in *2021 IEEE 10th Global Conference on Consumer Electronics (GCCE)*, pp. 595–596, IEEE, 2021. cited on p. 37

[19] I. Zyrianoff, L. Gigli, F. Montori, C. Aguzzi, S. Kaebisch, and M. Di Felice, "Seamless integration of restful web services with the web of things," in *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pp. 427–432, IEEE, 2022. cited on p. 37, 38

[20] S. Matsumoto, "Echonet: A home network standard," *IEEE Pervasive computing*, vol. 9, no. 3, pp. 88–92, 2010. cited on p. 37

[21] E. Korkan, F. Salama, S. Kaebisch, and S. Steinhorst, "A-mage: Atomic mashup generator for the web of things," in *Web Engineering: 21st International Conference, ICWE 2021, Biarritz, France, May 18–21, 2021, Proceedings*, pp. 320–327, Springer, 2021. cited on p. 38

[22] Q.-D. Nguyen, S. Dhouib, J.-P. Chanet, and P. Bellot, "Towards a web-of-things approach for opc ua field device discovery in the industrial iot," in *2022 IEEE 18th International Conference on Factory Communication Systems (WFCS)*, pp. 1–4, IEEE, 2022. cited on p. 38

[23] H. B. Hassine, E. Korkan, and S. Steinhorst, "Virtual-thing: Thing description based virtualization," *arXiv preprint arXiv:1909.03297*, 2019. cited on p. 39

[24] S. Muralidharan, B. Yoo, and H. Ko, "Designing a semantic digital twin model for iot," in *2020 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 1–2, IEEE, 2020. cited on p. 39

[25] J.-W. Wu, D.-W. Chou, and J.-R. Jiang, "The virtual environment of things (veot): A framework for integrating smart things into networked virtual environments," in *2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom)*, pp. 456–459, IEEE, 2014. cited on p. 39

[26] E. Korkan, E. Regnath, S. Kaebisch, and S. Steinhorst, "No-code shadow things deployment for the iot," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, pp. 1–6, IEEE, 2020. cited on p. 39

[27] I. Tursynbek and A. Shintemirov, "Modeling and simulation of spherical parallel manipulators in coppeliasim (v-rep) robot simulator software," in *2020 International Conference Nonlinearity, Information and Robotics (NIR)*, pp. 1–6, IEEE, 2020. cited on p. 40

[28] B. Bogaerts, S. Sels, S. Vanlanduit, and R. Penne, "Connecting the coppeliasim robotics simulator to virtual reality," *SoftwareX*, vol. 11, p. 100426, 2020. cited on p. 40