DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

**SCHRIFTENREIHE DER FAKULTÄT FÜR TECHNIK
DER DUALEN HOCHSCHULE BADEN-WÜRTTEMBERG
RAVENSBURG**

2022/01

Applying game engine in robot simulation

Bastian Demmel, Marc Hölle, Prof. Dr. Thomas Dietmüller, Patrick Leber M.Sc.,
Prof. Dr.-Ing. Thomas Kibler

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

SCHRIFTENREIHE DER FAKULTÄT FÜR TECHNIK
DER DUALEN HOCHSCHULE BADEN-WÜRTTEMBERG
RAVENSBURG

2022/01

Applying game engine in robot simulation

Bastian Demmel, Marc Hölle, Prof. Dr. Thomas Dietmüller, Patrick Leber M.Sc.,
Prof. Dr.-Ing. Thomas Kibler

Impressum

Schriftenreihe der Fakultät für Technik
der Dualen Hochschule Baden-Württemberg Ravensburg

Herausgeber

Prof. Dr. Heinz-Leo Dudek
Prorektor und Dekan der Fakultät für Technik

Duale Hochschule Baden-Württemberg Ravensburg
Baden-Wuerttemberg Cooperative State University
Marienplatz 2
88212 Ravensburg
Deutschland

http://www.ravensburg.dhbw.de

2022/01, Februar 2022

ISBN 978-3-945557-11-2
ISSN 2199-238X
DOI 10.12903/DHBW_RV_FN_01_2022_DEMMEL_HOELLE_DIETMUELLER_LEBER_KIBLER

Druck und Verarbeitung

Gestaltung
Nicole Stuepp
DHBW Ravensburg
Marienplatz 2, 88212 Ravensburg

Druck
Frick Kreativbüro & Onlinedruckerei e.K.
Brühlstraße 6
86381 Krumbach

Applying game engine in robot simulation

Bastian Demmel[1], Marc Hölle[2], Thomas Dietmüller[3], Patrick Leber[4], Thomas Kibler[5]

ABSTRACT

In this paper, we investigate the use of the game engine "Unreal Engine 4" simulating a robotic setup. For this purpose, computer-aided design (CAD) data of a UR5 robot with a RG2 gripper were modified and transferred to the engine. Further an inverse kinematic was added to move the robot and finally simulate a realistic pick and place situation. The paper describes the implementation step by step and critically assesses the results.

1 Bastian Demmel, Baden-Wuerttemberg Cooperative State University, Study Program of Engineering, Student

2 Marc Hölle, Baden-Wuerttemberg Cooperative State University, Study Program of Engineering, Student

3 Prof. Dr.-Ing. Thomas Dietmüller, Baden-Wuerttemberg Cooperative State University, Department of Mechanical Engineering, Germany, 88045, Friedrichshafen, Fallenbrunnen 2

4 M. Sc. Patrick Leber, IWT Wirtschaft und Technik GmbH, Software Developer

5 Prof. Dr.-Ing. Thomas Kibler, Baden-Wuerttemberg Cooperative State University, Department of Electrical Engineering, Germany, 88045, Friedrichshafen, Fallenbrunnen 2

1 INTRODUCTION

During the last decade, advances in computer science enabled new methods in multiple areas of engineering. Tasks like simulations that formerly have been considered unsolvable or only solvable with immense computational resources became feasible. For this reason, replications of physical objects in digital environments emerged as a serious strategy for problem-solving [1, 2].
On the other hand, replication is an essential component in video games and the underlying game engines. However, primarily popular in the entertainment industry, game engines can be used in different contexts [3].
Indeed, industrial applications can profit from the extensive visualization capabilities, immersive experiences and real-time rendering provided by game engines. Additionally, game engines offer the possibility to simulate and visualize processes and products in open platforms [4].
The aim of this paper is to investigate these capabilities by developing a robot simulation using Epic Games Unreal Engine and thereby suggesting an alternative approach to existing visualization tools like ROS Gazebo. Therefore, a method in four steps is set up, different tools are applied and finally the results are presented. In conclusion, this paper summarizes both necessary aspects for the Unreal Engine to promote the success of a simulation and the reasons hindering a robot simulation with this approach.

2 GAME ENGINE

In this chapter, the basic concept of game engines will be introduced based on the Unreal Engine 4. Game engine is a software that enables the creation of video games. The software provides methods to abstract common functionalities which are typical in video games and allows the reuse of assets and code in different games [5].
A game engine works as a framework in game development and enables developers to reuse existing assets and code so that the development does not have to start from scratch every time a new video game is created.

These functionalities are typically found in game engines [5]:

1. Rendering engine: Rendering is the process of creating an image from the underlying graphics data [6]. The rendering engine is responsible for the screen display during the game play.

2. Input handling: Allows the user to interact with the game through keyboard and mouse or other hardware.
3. Game loop: Internal routines that are updated constantly during playing and are responsible for triggering events.
4. Physics engine: Simulates physical behavior of objects in the virtual world. Provides collision detection for objects.
5. Scene graph: Manages graphical elements and spatial arrangement of objects on the screen [7].
6. Animation: Motions of objects to provide a vivid gameplay.
7. Resource management: Efficient management of limited hardware resources, like memory and process time.

Modern engines such as Unreal Engine 4 also allow platform independent development, which makes it possible to deploy created software to various types of devices, such as smartphones or virtual reality headsets [8].

One of the main advantages of Unreal Engine compared to other game engines is that in addition to programming with C++ a visual script system called Blueprint is provided. Blueprint consists of a node-based interface through which programming is made accessible, even to non-programmers. Game elements can thereby be programmed by placing nodes and connecting them with wires. It is also possible to expand the functionalities by creating custom nodes with C++ programming language.
In comparison to CAD data, which use mathematical functions to define surfaces, game engines approximate surfaces through triangular meshes. These can be rendered very efficiently in graphics processing units (GPUs) to allow a photorealistic real-time image. Thus, CAD data needs to be converted and approximated by mesh representation through tessellation [6, 9].

3 METHODOLOGY

For the implementation, UR5 and RG2 CAD data has to be imported into Unreal Engine and has to be programmed to perform a pick-and-place simulation in motion. The acquired simulation is then evaluated with a set of test cases and with comparing the behavior of real UR5 robots. The implementation is done in four steps as shown in Figure 1.

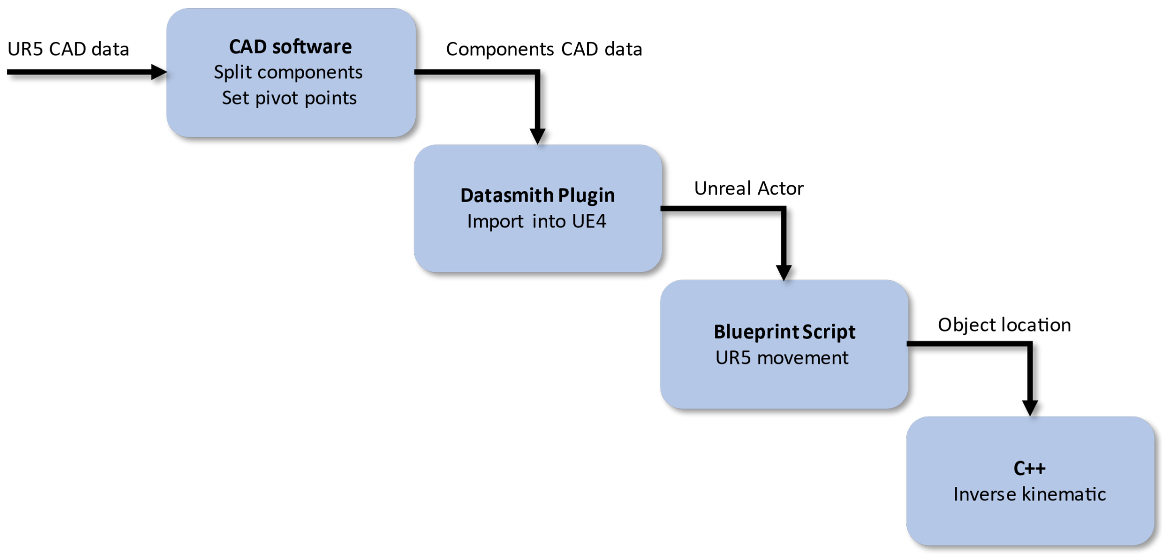

Figure 1: Implementation in four steps

First CAD data is gathered from UR and On Robots as manufacturer of robot and gripper. To achieve a realistic motion of the robot arm, the entire CAD model of the robot has to be cut into components such as joints and links first. This "disassembly" is done in CAD software Autodesk Inventor, as shown in Figure 2.

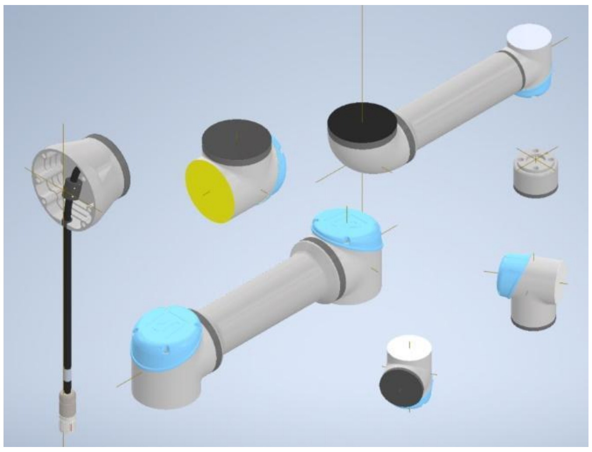

Figure 2: UR5 disassembled in Autodesk Inventor

Afterwards, pivot points have to be set for these components. The pivot points need to be centered in the components and their x-, y- and z-axis should align with the axis of rotation. Figure 3 shows the pivot point of a joint before and after - the pivot point is represented by a white dot and its x-, y- and z-axis highlighted. With this modification a rotation of the components is much easier to implement later on.

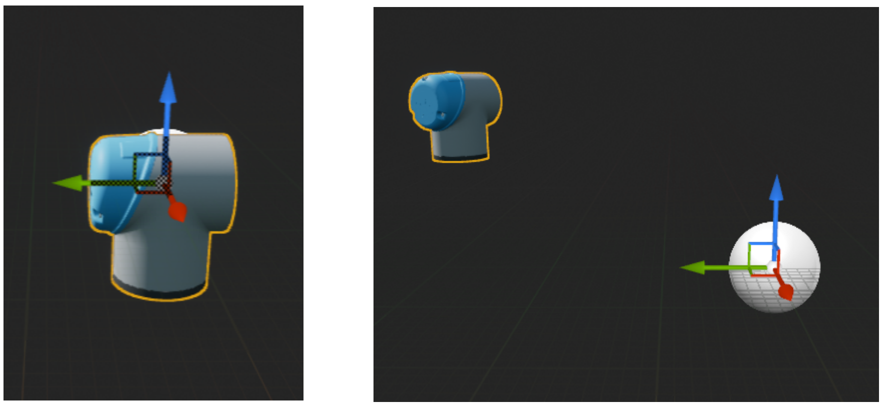

Figure 3: Pivot point placement on joints

The modified components are now imported from the CAD software to Unreal Engine using the Datasmith plugin.

In a second step, the components in Unreal Engine 4 are assembled into a functional robot model again - the so-called Actor. Joints and links are attached to each other as successive child components in a hierarchical structure. Thus, a kinematic chain is created and the rotation of a previous component in the chain causes all following components to move accordingly.

In the third step, a keyboard control is implemented to set the model to motion. This is realized by using Blueprint scripting. Since the pivot points are suitably placed during the modification step, every rotational axis gets two keys: one for a clockwise rotation and one for counterclockwise rotation. The opening and closing of the RG2 gripper are achieved through parallel kinematic computations, which results in the parallel movement of the tips of the gripper.

Finally, the joint angles associated with the desired position of the robot's effector have to be computed through inverse kinematic [10]. As a result of this, the robot picks up an object and puts it at a specified location.

Inverse kinematic describes the mapping from the effectors position and orientation (pose) to the corresponding joint angles. Thus, allowing the robot to move its effector to a given location.

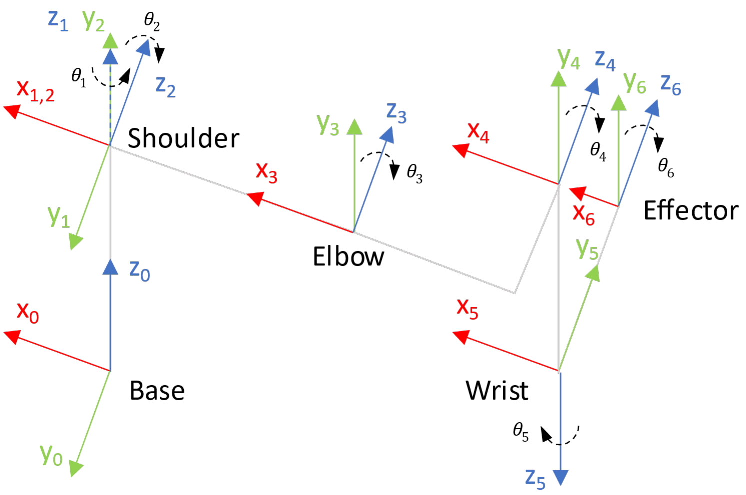

Figure 4: UR5 kinematic model [11]

The mathematical foundation for these computations is provided by Andersen [11] and the underlying model is depicted in Figure 4. Thereby, a coordinate system is assigned to every joint in such way that the z-axis aligns with its axis of rotation and the angle θ describes the rotation around this axis.

The implementation of the inverse kinematic is done in the last step using a custom Blueprint node in C++. Figure 5 shows how this node is integrated and used inside a Blueprint script.

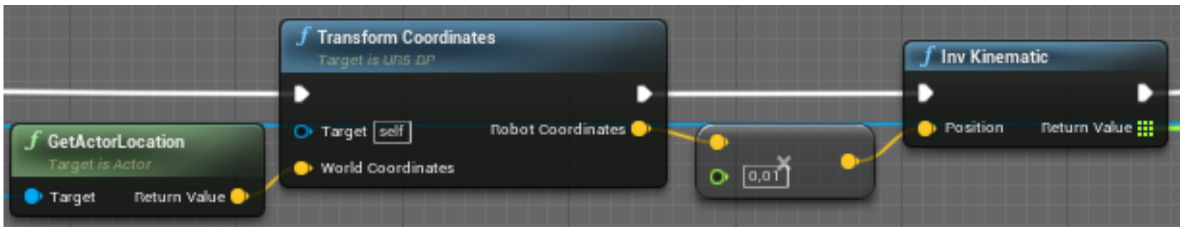

Figure 5: Integration of node

The custom node “Inv Kinematic” takes the desired position of the effector as an input parameter and returns the associated joint angles as a list of values. But before the location can be passed to the node “Inv Kinematic” it has to be transformed from world coordinates of the Unreal Engine into the reference frame of the robot and additionally scaled from centimeters to meters.

After computing the joint angles, a rotation must be applied to every joint of the robot according to the computed angels. To obtain a smooth movement of the robot, a linear interpolation is applied from current joint angle to the desired joint angle with small incremental steps, thus leading to a seemingly continuous movement.

Finally, the created model is packaged into an Unreal Plugin, which can be integrated into other projects.

4 RESULTS

This chapter provides an overview of the created model and discusses the results of the applied test cases.

The described model contains the following functions:

1. Keyboard control: The axis of rotation and the gripper can be controlled with the keyboard.
2. Move to pose: The arm can be moved into an adjustable pose.
3. Pick-and-place: A demo object can be picked up and placed at an adjustable location.
4. Inverse kinematic: Based on a desired location of the effector, the corresponding joint angles can be computed, and the effector can be moved to this location.
5. Plugin: The model can be transferred to different projects.

To test and compare the model with the real UR5, pick-and-place scenarios with objects in different regions around the robot were executed and analyzed.
Overall, the model proved to be very convincing with realistic motions. The resulting model of UR5 with attached gripper (left) and a detailed view of the gripper (right) are shown in Figure 6.

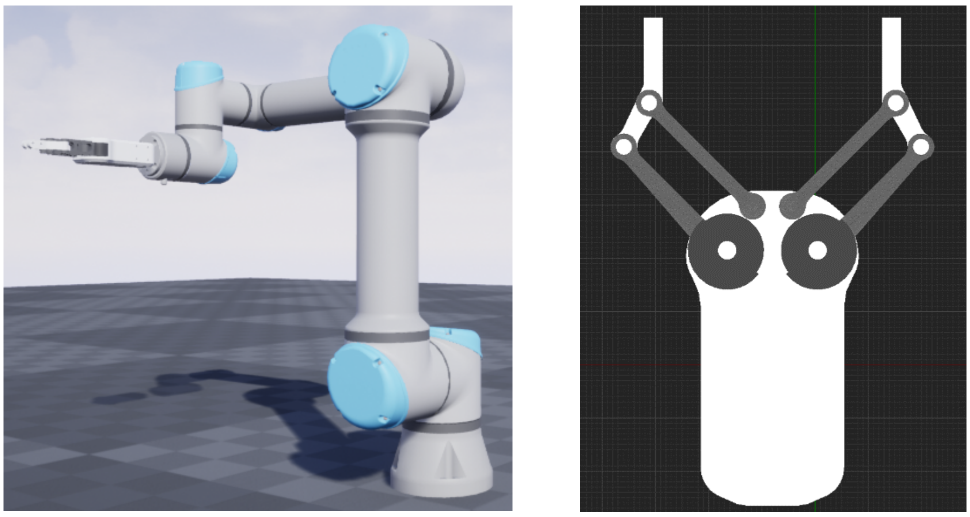

Figure 6: final model with gripper

Nevertheless, during validation tests some potentials were detected. Currently, the model cannot detect if objects are outside its workspace. This leads to two problems occurring, either if the object is placed out of the robot's range or if it is placed too close to the base component of the robot. In the first case, the model will align in a straight line with the object and thus move to the closest possible location. The second case is shown in Figure 7 (left). In this case, certain components of the robot collide while picking up the object. This can be resolved in further development steps through computing the object location relative to the model.

The right image in Figure 7 shows a slight offset that occurs during the pick-and-place. The magnitude of this offset varies with the location of the object. This is caused by the limited accuracy which the Unreal Engine allows for in the placement of components. A potential correction can be achieved through determining a vector from the offset effector location to the desired location. This vector can be added to the desired location and the inverse kinematic computed based on this corrected location.

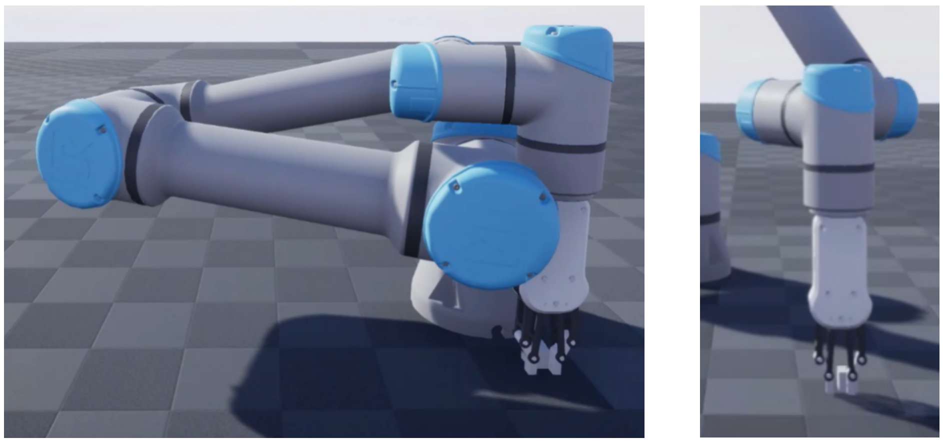

Figure 7: Component collision (left) and offset between object and gripper (right).

Depending on which quadrant relative to the base of the model the object is placed in, inefficient trajectories with similarities to a pirouette arise. This is the result of the linear interpolation in between joint angels, where the movement of the effector is not accounted for. To take its movement into account would require a path planning strategy [12].

5 CONCLUSION

The game engine Unreal Engine 4 allows the creation of very flexible and interactive models. Based on the Blueprint script system, modifications can be easily applied. With the possibility to create plugins, the created models can be reused and integrated in multiple projects. Due to platform independence offered by Unreal Engine, the created model can be used in virtual reality applications.

Additionally, real-time rendering enables interactive handling of the simulation in photorealistic representation. This allows a higher vividness than a statically rendered simulation.

The only disadvantage is the limited positioning accuracy of the components. Though the offset between components is minimal, it causes a small gap between gripper and object during some pick-and-place application. However, this offset could be eliminated by a correction procedure as described in this paper.

With the proposed model a new approach is found to use game engines to generate digital twins - as applied in Industry 4.0. A Digital twin connects physical and virtual world through continuous data exchange, the so-called digital thread [13, 14]. This means that a real robot is virtually represented by the digital twin and different scenarios can be simulated. Unreal Engine can display the digital twin on almost all devices. In this way, the scenarios can be reviewed easily in home office or evaluated by service engineers abroad.

In summary, the application of Unreal Engine in robot simulation was proven most widely successful which opens up new possibilities in the visualization of digital twins.

REFERENCES

[1] M. Schluse, L. Atorf, J. Rossmann: Experimentable Digital Twins for Model-Based Systems Engineering and Simulation-Based Development, 2017 Annual IEEE International Systems Conference (SysCon), 2017

[2] S. Jersov, A. Tepljakov: Digital Twins in Extended Reality for Control System Application, 43rd International Conference on Telecommunications and Signal Processing (TSP), pp. 274, 2020.

[3] J. Seppala: The Automotive Field Guide, Epic Games, 2020

[4] S. Kloiber, C. Schinko, V. Settgast, M. Weinzerl, T. Schreck. R. Preiner: Integrating Assembly Process Design and VR-based Evaluation using the Unreal Engine, VISIGRAPP 2020 - Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, pp. 271, 2020

[5] A. Andrade: Game engines: a survey, EAI Endorsed Transactions on Game-Based Learning 2, 2015.

[6] A. Nischwitz, M. Fischer, P. Haberäcker, G. Socher: Computergrafik und Bildverarbeitung, Band 1, Computergrafik, Vieweg+Teubner Verlag, Wiesbaden, 2012.

[7] H. Hussmann: The Scene Graph, [Online] Available: https://www.medien.ifi.lmu.de/lehre/ss12/cg1/vorlesung/cg1_6.pdfl Accessed on: 21 Feb 2021.

[8] Epic Games: Sharing and Releasing Projects, [Online] Available: https://docs.unrealengine.com/4.26/en-US/SharingAndReleasing/ Accessed on: 14 Feb 2021.

[9] Epic Games: Using Datasmith with CAD File Formats, [Online] Available: https://docs.unrealengine.com/4.26/en-US/WorkingWithContent/Importing/Datasmith/SoftwareInteropGuides/CAD/ Accessed on: 28 May 2021.

[10] J. Mareczek: Grundlagen der Roboter-Manipulatoren, Band 1, Modellbildung von Kinematik und Dynamik, Springer Verlag, Berlin, 2020.

[11] R. Andersen: Kinematics of a UR5, 2018.

[12] J. Mareczek: Grundlagen der Roboter-Manipulatoren, Band 2, Pfad- und Bahnplanung, Antriebsauslegung, Regelung, Springer Verlag, Berlin, 2020.

[13] L. Girletti, M. Groshev, C. Guimaraes, C. Bernardos, A. de la Oliva: An Intelligent Edge-based Digital Twin for Robotics, IEEE Globecom Workshops (GC Wkshps), 2020

[14] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee: Digital Twin in Industry: State-of-the-Art, IEEE Transactions on Industrial Informatics, pp. 2405–2415, 2019

Herausgeber

Prof. Dr. Heinz-Leo Dudek
Prorektor und Dekan der Fakultät für Technik

Duale Hochschule Baden-Württemberg Ravensburg
Baden-Wuerttemberg Cooperative State University
Marienplatz 2
88212 Ravensburg

ISBN 978-3-945557-11-2

ISSN 2199-238X

DOI 10.12903/DHBW_RV_FN_01_2022_DEMMEL_HOELLE_DIETMUELLER_LEBER_KIBLER