Interactive Path Editing and Simulation System for Motion Planning and Control of a Collaborative Robot

Taeho Yoo and Byoung Wook Choi *

Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul 01811, Republic of Korea; kjkj0581@seoultech.ac.kr
* Correspondence: bwchoi@seoultech.ac.kr; Tel.: +82-02-970-6412

Abstract: Robots in hazardous environments demand precise and advanced motion control, making extensive simulations crucial for verifying the safety of motion planning. This paper presents a simulation system that enables interactive path editing, allowing for motion planning in a simulated collaborative robot environment and its real-world application. The system includes a simulation host, a control board, and a robot. Unity 3D on a Windows platform provides the simulation environment, while a virtual Linux environment runs ROS2 for execution. Unity sends edited motion paths to ROS2 using the Unity ROS TCP Connector package. The ROS2 MoveIt framework generates trajectories, which are synchronized back to Unity for simulation and real-world validation. To control the six-axis Indy7 collaborative robot, we used the MI05272 embedded board as an EtherCAT master. Verified trajectories are sent to the target board, synchronizing the robot with the simulation in position and speed. Data are relayed from the host to the MI05272 using ROS2 and the Data Distribution Service (DDS) to control the robot via EtherCAT communication. The system enables direct simulation and control of various trajectories for robots in hazardous environments. It represents a major advancement by providing safe and optimized trajectories through efficient motion planning and repeated simulations, offering a clear improvement over traditional time-consuming and error-prone teach pendant methods.

Keywords: ROS2; MoveIt; Unity; motion planning; simulation; collaborative robot

1. Introduction

In manufacturing, industrial robots perform tasks in routine activities and unhealthy workplaces, primarily relying on the teach pendant GUI and force guidance for path planning [1–3]. Recently, various types of robots have been applied in multiple fields, performing tasks tailored to each purpose; however, the traditional path planning methods for executing diverse and complex tasks not only take significant time but are prone to errors [4,5]. Furthermore, trajectories generated for tasks in hazardous environments must ensure safety, especially as these environments continue to grow alongside the expanding robot market.

To address these challenges, digital twins can simulate robots operating in hazardous environments to predict potential problems. By utilizing a digital twin, the real objects associated with tasks are duplicated to create a virtual environment where virtual models behave the same as the original objects [6–8]. The virtual environment for the robots is updated based on real-time data obtained from sensors and the simulation results of the generated trajectories are applied to real-time robot control, causing the real-world objects to change correspondingly [9,10]. Levine et al. [11] extended the learning of vision-based manipulation policies [12] and trained a corresponding inverse dynamics model using real-world data to transfer policies to a real robot. In this way, digital twin ensures the safety of robots operating in hazardous environments by providing simulation results where problems can be addressed.
In addition to digital twins, robots can simulate complex tasks in a virtual environment and successfully solve them using deep reinforcement learning methods [13,14]. Sim-to-real transfer involves pretraining in a virtual environment and then fine-tuning with real-world data [15]. The policies developed through transfer learning can be deployed to perform complex tasks with robots in the real world, and can reduce the training and trajectory generation time required by deep reinforcement learning [16]. Peng et al. [17] proposed a simple method to bridge the simulation–reality gap by randomizing the dynamics of the simulator during training. High-quality rendering was employed to bridge the reality gap in [18], while ref. [19] developed various locomotion skills for robots by randomizing dynamics models. However, the reality gap, which is the difference between the simulated and real environments, affects learning performance, making it crucial to consider the dynamics and observation delays when implementing the simulation environment.

Robot simulation systems commonly address real-world problems in virtual environments, facilitating the integration or expansion of technologies such as digital twins and the sim-to-real transfer process. Users of these systems should have the flexibility to edit the generated trajectories and consequently control the real-world robot in real-time. In this paper, we present a system that allows users to interactively edit the path of a collaborative robot. The system includes a simulation system for real-world robots.

Numerous studies have used visualization tools to monitor robots as they perform tasks and simulate their interactions with other objects in real time. These simulation and visualization processes demand significant computing power. Robotic simulators are rigorously assessed and chosen based on their capabilities, including 3D model implementation and user interfaces. One study [20] examined the features of Gazebo and Unity for simulating robotic assembly processes in manufacturing. In this research, the UR3 model from Universal Robots was used in the simulator to compare ROS integration, analyze performance, and conduct experimental research. The results showed that both platforms are suitable for robot simulation; however, while Gazebo offers advantages in integration with ROS, it has lower graphic quality compared to Unity. Other research [21] conducted a Systematic Literature Review (SLR) and meta-analysis of simulation platforms for applying AI techniques to Wheeled Mobile Robots (WMR).

In our research, we chose Unity in order to facilitate the convenient integration and expansion of various systems; it allows for the convenient implementation of a 3D virtual environment for robot simulation and enables the simulation of various planned paths and the application of algorithms.

In this paper, we describe a system developed within a Unity-based virtual environment that allows users to interactively edit the path of the six-axis Indy7 robot from Neuromeka [22] using a mouse. In this way, various objects and robots can be duplicated to configure the simulation environment in Unity, and multiple reinforcement learning algorithms can be implemented through ROS2. This allows for the simulation of complex tasks performed by robots in hazardous environments where direct path planning is limited and the efficient generation of multiple trajectories that can be applied to real-world robots.

Several studies have addressed path planning and simulation for articulated robots in virtual environments. In [23], the authors proposed a method using SolidWorks CAD for path planning and simulation of industrial robots in which trajectories for various tasks were generated through a GUI. However, due to its lack of integration with ROS2, its capabilities in path planning and controlling multiple robots are limited. In contrast, our system leverages ROS2 and Unity to generate and simulate robot trajectories, facilitating expansion to various virtual environments and multiple robots. In another previous study [24], a controller was implemented to manage multiple robots using the ROS MoveIt framework [25]. In [26], a control interface was developed to manipulate a six-axis articulated robot using a keyboard, with simulations conducted in Rviz. However, the ability to control the angles and velocities of all joints is crucial for complex task planning. Our system transmits trajectory segments generated in the ROS in real time to robots in both
real-world and virtual environments, ensuring synchronization with the segment data for control.

In [27], an AR environment was implemented with a 3D model of a six-axis articulated robot. Path planning was achieved using real-world markers and a beam search algorithm. Another study [28] employed a handheld pointer recognized by a motion capture system for path planning. However, these studies require simulation to verify whether the generated trajectory is suitable for performing tasks.

Our system includes a host device for interactive path editing and simulation as well as an embedded board for controlling the robot. On the host, users can conveniently plan the robot’s path within a 3D virtual environment via Unity using a mouse. The user-defined path is sent to ROS2 in a virtual Linux environment via the ROS TCP Connector [29]. In ROS2, information about the robot modeled in the virtual environment is available and the motion planning framework MoveIt generates a trajectory consisting of segments that contain the robot’s position and speed information. Each segment is transmitted back to Unity in real time through the ROS TCP Endpoint package [30], ensuring that the robot’s position and velocity in the simulation are synchronized with the trajectory generated in ROS2. When the user verifies that the simulation result is suitable for task performance, the trajectory data are transferred from ROS2 to the MIO5272 embedded board which controls the Indy7 robot. These trajectory data are delivered in real-time for each segment, synchronizing control of the real-world Indy7 with the position and velocity from the robot simulation in Unity. As a result, the user can directly apply and control the simulation results from the virtual environment on the real-world robot.

The rest of this paper is organized as follows: Section 2 introduces the system architecture and describes the method for implementing interactive path planning in a virtual environment and applying the simulation results to a real-world robot; Section 3 details the process of generating the trajectory of the user-planned path, controlling the six-axis collaborative robot in both virtual and real environments, and comparing the robot’s speed in each environment; finally, we summarize the experimental results and conclude the paper.

2. System Architecture

In this section, we provide a detailed description of the overall architecture and communications of the interactive path editing system. The system consists of a host device and an embedded board. The host device features an Intel Core i5 6600 CPU running at 3.3 GHz, 16 GB of SDRAM, and an NVIDIA GeForce GTX 1050 Ti GPU. The target device is an MIO-5272 embedded board, which contains an Intel Core i7 6600U CPU at 2.6 GHz and 16 GB of SDRAM. The host device uses the Windows operating system, and a VirtualBox virtual machine is employed to run Linux Ubuntu on the same device. The embedded board operates with a single operating system, which is Linux Ubuntu. Table 1 shows the hardware specifications of the interactive path editing system.

<table>
<thead>
<tr>
<th>Hardware Environment</th>
<th>Microprocessor Unit</th>
<th>Architecture</th>
<th>Operating System</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop (host)</td>
<td>Intel Core i5 6600</td>
<td>x86-64</td>
<td>Windows 10</td>
<td>22H2</td>
</tr>
<tr>
<td>Desktop (virtual machine)</td>
<td>Intel Core i5 6600</td>
<td>x86-64</td>
<td>Linux Ubuntu 22.04.4</td>
<td>6.5.0</td>
</tr>
<tr>
<td>Embedded board MIO-5272</td>
<td>Intel Core i7 6600U</td>
<td>x86-64</td>
<td>Linux Ubuntu 20.04.6</td>
<td>5.15.158-rt76</td>
</tr>
</tbody>
</table>

Unity is a software platform used for developing games and simulations, and is expanding its applications into fields such as robotics and autonomous driving. In this paper, we have implemented Unity 2022.3.17f1, which can be accessed for download in [31].
We employed ROS2 (Robot Operating System 2), a middleware that includes software libraries and tools for building robot applications. On the host device, Unity operates under a Windows environment to facilitate robot path planning and simulation, while ROS2 based on Linux runs within a VirtualBox for trajectory generation.

The embedded board, which controls the real-world Indy7 robot, also uses Linux, consistent with previous studies [32].

Preempt-RT is a set of patches for the Linux kernel aimed at minimizing latency to perform real-time tasks. For real-time control of the Indy7 robot, the Linux kernel on the target device was patched with Preempt-RT. Additionally, an EtherCAT Master was implemented on the target device to implement EtherCAT, which is a network protocol based on Ethernet for real-time distributed control of multiple devices.

The application for trajectory generation uses ROS2 Humble, while the motion control application uses ROS2 Galactic to maintain compatibility with the version developed in previous research. Figure 1 depicts the deployment diagram of the interactive path editing and simulation system.

Figure 2 shows the architecture of the robot simulation and control system. The ROS TCP Connector and ROS TCP Endpoint are packages that facilitate the transmission of ROS messages between Unity and ROS. Messages are transmitted over a TCP/IP socket and then converted to C# for Unity scripts [33,34]. The user-planned robot path information is transmitted from Unity to ROS2 via the ROS TCP Connector package.

In the system, all transmitted ROS messages are service interfaces. The Unity-based simulation and the ROS2-based motion control application act as clients and send request messages. Conversely, the ROS2 application responsible for trajectory generation through MoveIt functions as the server and sends response messages. In this way, the service interface performs transmission according to the service requested by the client, corresponding to the number of path points or the number of segments in the generated trajectory. For trajectory generation within ROS2, libraries for kinematics, dynamics, and motion planning are utilized.

Orocos is a library that specializes in modeling and computing the kinematics and dynamics of robots [35]. It can be integrated with ROS, and is included as a plugin in MoveIt for solving Inverse Kinematics. The Open Motion Planning Library (OMPL) is another crucial tool designed for implementing various sampling-based motion planning algorithms, and can also be integrated with ROS [36]. Utilizing OMPL it as a plugin in MoveIt, specific motion planning algorithms from the library can be selected to generate the robot’s trajectory.

After generating the trajectory in MoveIt using these libraries, it is sent back to Unity for simulation through the ROS TCP Endpoint. If the simulated trajectory proves suitable for the intended task, the trajectory data can be continuously transmitted in segments...
from Unity to ROS2 on the MIO-5272 embedded board, which controls the real-world Indy7 robot. The motion control application on the MIO-5272 consists of a Dynamics Node, which computes the kinematics and dynamics of the Indy7 robot using the Orocos KDL library. Additionally, a Control Node communicates with the Indy7 robot using the IgH EtherCAT protocol.

DDS (Data Distribution Service) is a networking middleware used for specifying publishers and subscribers to facilitate communication between nodes. In this system, it is used for node communication between the host and target device in ROS2. The Control Node receives real-time trajectory segments from the host device via DDS communication and controls the Indy7 robot using the IgH EtherCAT Master Core.

**Figure 2.** Architecture of the robot simulation and control system.

### 2.1. Simulation Environment Configuration

In this paper, we duplicated the Indy7 robot in Unity for the robot simulation environment. The 3D modeling files for the robots from Neuromeka are available on GitHub [37] in the xacro format, which details the robot’s structure and properties. The xacro files can be converted to URDF (Unified Robot Description Format) using the xacro command. In Unity, URDF files can be imported using the URDF Importer [38]. By referencing within the xacro file and applying the xacro command with the appropriate robot type and name parameters, the URDF file is generated and then uploaded along with the corresponding 3D modeling files to the Unity Assets and duplicated the Indy7 robot in the virtual environment using URDF Importer.

As shown in Figure 3, the structure of the Indy7 robot in Unity is organized hierarchically, where each subsequent link is a child of the preceding one. If link0 is the root body, then the robot is not capable of being anchored to the ground when in play mode. To address this, a base ground object must be added and designated as the root body in the Articulation Body component. The coordinates for the robot show the coordinates of the ground; thus, the immovable property is subsequently enabled to fix the Indy7 robot to the base.

**Figure 3.** Link architecture of the Indy7 robot in Unity.
To make for a more accurate simulation, any physical parameters not represented in the URDF should be set to the Articulation Body component. Based on our experiments with the Indy7 robot [39], we assigned stiffness and damping coefficients to each link. To ensure that these physical parameters were effectively applied, the Drive type was set to ‘Force.’ However, not all parameters may be successfully applied in play mode due to various constraints. To overcome this limitation, an initialize button was developed to allow for direct modification of the Indy7 robot’s properties through a C# script.

2.2. Interactive Path Editing

In the virtual environment, users can interactively edit the robot’s path tailored to specific tasks. In this system, users can determine the three-dimensional position of the robot’s end effector for path planning utilizing the mouse cursor.

Scripts for interactive path planning in Unity provide a ray interface and camera control for specifying path points of the end effector within a 3D virtual environment. The position information of each designated point is stored in a list through the path point list manager and later sent to ROS2 for trajectory generation. To facilitate communication with ROS2, the user control script uses ROS TCP Connector package and a path point converter to transform Unity coordinates for compatibility with ROS2 MoveIt. Additionally, it includes initialization to the first position for controlling the robot in the virtual environment and synchronizing with the velocity of the generated trajectory. User function button scripts offer various features to facilitate path planning and simulation of the robot in Unity as well as position measurement features for experimentation. Figure 4 illustrates the path editing and simulation system implemented in Unity.

![Path editing and simulation system in Unity.](image)

To specify the end effector’s position in Unity, the mouse cursor position corresponding to the camera screen is captured. An invisible ray is created from this position in the direction that the user’s camera faces using ScreenPointToRay(). The XZ plane coordinates are then obtained by Raycast() on a terrain object.

However, as there is no plane for the projection of the Y coordinate, a vertical plane is created at the previously established X and Z coordinates, oriented facing the user’s camera, in order to determine the projected height coordinate. The user sequentially selects the position of each coordinate to determine the position of the robot’s end effector. In this system, coordinates are selected in the order (X, Z, Y). To enhance user interaction, a graphical user interface (GUI) is provided displaying lines perpendicular to the X and Z axes that follow the user’s mouse cursor. Additionally, a line parallel to the Y axis is displayed which adjusts its length based on the selected height. Figure 5 displays the path planning in virtual environment for Indy7 robot. The starting point of the path is shown in green, the endpoint in red, and all other points in blue.
Figure 5. Interactive path planning in the virtual environment for the Indy7 robot.

The functions for interactive path editing and simulation of the Indy7 robot in Unity were implemented through C# scripts, and can be accessed via modeled buttons in the virtual environment. To verify the path editing process, the necessary results are displayed to the user via console messages. This system includes buttons for initializing the Indy7 robot, recording positions, and setting path points. Figure 6 shows the user function buttons and position data displayed via console messages in Unity. In the user function buttons, robot variable initialization is shown in yellow, the path point set button in blue, and the robot position recorder in red.

Figure 6. User function buttons and position data displayed via console messages in Unity.
The position data of the robot’s end effector specified by the user is transmitted to an ROS2 service node via a service interface. This node employs MoveIt’s inverse kinematics solver to verify whether the end effector can reach the designated position. If the solver successfully computes the angular positions of each link, these positions are relayed back to Unity through a response message and applied to the Indy7 robot. If the calculation fails, a message is displayed to the user prompting them to reselect the end effector’s position in the virtual environment. This process allows the user to interactively plan the path of the robot’s end effector by confirming valid positions. Figure 7 illustrates how to obtain inverse kinematics solution for an interactive edited position.

Figure 7. Sequence diagram of solving an interactive edited position.

2.3. Trajectory Generation and Simulation for Collaborative Robot

To utilize the MoveIt framework for generating the robot’s trajectory in ROS2, it is essential to create configuration files specific to MoveIt. The MoveIt Setup Assistant facilitates this by converting URDF or xacro files into a MoveIt configuration package. This configuration package encompasses information about the robot such as self-collisions, virtual joints, planning group, controllers, and kinematic solver. Therefore, we used the MoveIt Setup Assistant to generate the configuration package for Indy7.

When implementing a ROS2 robot application with the MoveIt framework, it is imperative to build the MoveIt configuration package. To confirm successful integration with the MoveIt framework, the demo file generated within the package can be launched. This allows for the visualization of the robot in Rviz2, providing a graphical interface to observe current robot state. Because Indy7 is described in xacro files, it is necessary to specify the robot type and name parameters in all launch files in order to ensure accurate setup and operation.

The settings for plugins such as OMPL and kinematics for robot trajectory generation can be configurable in the YAML files within the MoveIt configuration package. Our system utilizes the KDL kinematics plugin provided by the Orocos KDL package and employs the PRM (Probabilistic Roadmap) algorithm to generate trajectories between two points specified by the user.

The MoveIt main package is implemented on a virtual machine for the simulation and control system, utilizing the MoveIt framework and configuration package to validate the end effector’s position and generate trajectories. The control interface of the service interface package includes information on path points or segments and is synchronized with the trajectory’s velocity, transmitting these data for simulation and control of the Indy7 robot. Various functions of the MoveIt main package can be selected by the user through mode control. Figure 8 illustrates the implemented packages of MoveIt main package and service interface package with the ROS2 MoveIt framework on a virtual machine for trajectory generation.

The point data of the path planned by the user in Unity are transmitted to an ROS2 service node via a service interface. To constrain the robot’s end effector position, our system utilizes the kinematic constraint, path constraint, and workspace constraint APIs from MoveIt. Kinematic constraints restrict the position of the robot’s end effector or the angular
position of each joint, and are applied to each point along the Indy7 robot’s path. Path constraints limit the trajectory generation range. We implemented box-shaped constraints between two positions on the robot’s end effector path to restrict the end effector’s direction of movement. Finally, workspace constraints define the area within which the robot’s entire body can operate. We specified a box-shaped workspace encompassing both the Indy7 robot and the two points of the end effector’s path.

Figure 8. Packages for trajectory generation system in ROS2 MoveIt on virtual machine.

When all constraints between the two points have been imposed, the PRM algorithm in OMPL generates a trajectory that adheres to the specified constraints. This process is repeated for each point to obtain the trajectory of the path for the Indy7 robot. Consequently, both the current point number and total count information for every point in the path planned by the user in Unity are transmitted to the service node via the service interface.

In MoveIt, the time parameterization algorithm can generate the velocities in the trajectory. This algorithm produces the velocities for a single connected trajectory, combining all previously obtained trajectories between each pair of points. In this system, we use the iterative parabolic time parameterization algorithm to generate the velocities in the trajectory. Robots performing complex tasks with time constraints need to synchronize their speed with the task execution time. The trajectory consists of multiple segments; to achieve synchronization, the velocity of all segments is scaled at a constant ratio. For simulation in Unity, starting from the first point of the path, the system transmits the service interface to the service node for each segment. Using the segment and point number information, the angular position and velocity data of each joint of the Indy7 robot are received iteratively and synchronized with time according to the trajectory. Figure 9 depicts the sequence diagram of the trajectory generation and simulation system.

Figure 9. Sequence diagram of trajectory generation and simulation system.
2.4. Motion Control for Applying Simulation Result

We controlled the six-axis Indy7 collaborative robot from Neuromeka to apply the user-planned path to a real-world robot. As in previous studies, we used the MIO-5272 embedded board to control the Indy7 robot. In this system, we ported the main task and dynamics task in the motion control application to the ROS2 control node and dynamics node, respectively. Additionally, we implemented functionality in the control node to receive simulation results from the service node on the host device via DDS. Because segment data are transmitted in synchronization with the task’s performance time, the robot control application on the MIO-5272 must meet time constraints. Therefore, in order to control the Indy7 robot in real time on the MIO-5272, we implemented the IgH EtherCAT Master using the EtherCAT real-time communication protocol.

When the robot control application starts, the dynamics node uses the Orocos KDL library to solve dynamics problems and sends the data for the Indy7 robot to the control node via the message queue. Even if the service node does not send segments or if the control data field for the robot is empty, the dynamics node continuously sends the current robot position and dynamics data to the control node. Figure 10 illustrates the architecture of the Indy7 control system in MIO-5272 that was implemented for applying the generated trajectory.

![Architecture of the Indy7 control system in MIO-5272.](image)

The path verified through simulation in the virtual environment is transmitted as a service request to control the real-world Indy7 robot in the same manner as in Unity. DDS communication for receiving trajectory segments from the service node operates in parallel with the real-time task for EtherCAT communication, and is synchronized with the time constraints through a semaphore. When the control node receives a segment via a response message, it waits on the semaphore. When the real-time task for EtherCAT communication (running in parallel) completes the robot control, the semaphore is released; similarly, when the real-time robot control tasks for the EtherCAT communication are completed, the semaphore is released. Subsequently, the MIO-5272 embedded board transmits data to the actuators of the Indy7 robot via EtherCAT communication to control the joint angular positions. This process is repeated for each segment along the trajectory provided by the service node. Figure 11 shows the motion control flow for applying the Indy7 robot simulation.

As a result, users can plan the path of the Indy7 robot through the interactive path editing and simulation system, allowing complex tasks to be performed in hazardous environments using a mouse in Unity. Users can simulate the robot in real time through communication with ROS2. Additionally, the robot control application implemented in ROS2 on the embedded board receives trajectory data from the host device and controls the Indy7 robot, enabling the application of simulation results from the virtual environment.
3. Experiment

In this section, we report the experiments conducted to verify that the trajectory generated by the interactive path editing and simulation system implemented in this paper, showing that it can be applied to real-world robot control. In our experiments, we measured the velocity of the robot’s trajectory as it avoided an obstacle, for which we used the Indy7 collaborative robot and a drawer. To replicate the experimental environment, we created models of the robot and obstacle in the same positions in Unity. The user then planned a path to avoid the obstacle using the mouse, and we simulated it to verify the generated trajectory. The MIO-5272 received the trajectory segments from the simulation and controlled the real-world Indy7 robot according to the user-planned path.

To measure the velocity of the Indy7 robot simulated in Unity, we recorded the rotation property of a transform component, which changes according to the angular position of each joint. In the real world, the motion control results of the Indy7 robot are measured using an encoder, with the angular positions of the joints recorded through the EtherCAT module [40]. The angular positions of all joints of the Indy7 robot in both simulation and the real world were recorded at a 1 ms period, and the generated trajectories and velocities from MoveIt were compared for each segment.

The task of controlling the Indy7 robot in two environments is performed concurrently with the task of receiving segments from ROS2 MoveIt. In Unity, because the Indy7 robot is not simulated at a constant interval, we implemented a recorder task with a 1 ms period. On the other hand, the motion control task on the MIO-5272 operates at a 1 ms period; therefore, we incorporated a recording process to minimize the impact of multitasking. The same method was used to calculate the elapsed time in both environments. The pseudocode for measuring the angular of the Indy7 robot is provided in Algorithms 1 and 2. The source code for the experiments will be made available in our repository: https://github.com/SeoulTechEmbeddedLab/unity_indy7.git, accessed on 16 July 2024.

In the implemented system, the orientation of the Indy7 robot’s end effector is fixed, allowing specification of coordinates in terms of X, Y, and Z, which means that MoveIt cannot solve the inverse kinematics for all positions. Specifically, for the Indy7 robot, inverse kinematics cannot be solved for certain positions of the second joint. This limitation prevented the robot’s path from closely following the opposite wall of the obstacle during the experiment. Therefore, in this experiment the path of the Indy7 robot to avoid the obstacle was planned using positions where the inverse kinematics could be successfully solved.

The robot’s workspace area can solve the inverse kinematics for all points, and is visualized in the virtual environment to handle a wider range of positions. In future research, we aim to improve this system to ensure that the Indy7 robot’s end effector can reach all points within the workspace area. To this end, we will implement a system for interactively editing the orientation of the end effector in order to solve more inverse
kinematics problems for the robot. Figure 12 shows the simulation and motion control of the Indy7 robot for the path chosen to avoid the obstacle.

Algorithm 1: Pseudocode for measuring the rotation of all joints of the Indy7 robot in the Unity simulation environment

```java
set record_period to 1ms;
start_ticks = get_current_ticks();
Function update() /* repeated until the play mode ends */
if recorder on then
    WaitForSecondsRealtime(record_period);
    elapsed_ticks = get_current_ticks() − start_ticks;
    set rotation[1..6] to rotation property of ArticulationBody[1..6];
    set recorder_string to elapsed_ticks and rotation[1..6];
    WriteLine(recorder_string);
end
end
```

Algorithm 2: Pseudocode for measuring the angular position of all joints of the Indy7 robot on the MIO-5272

```java
set control_period to 1ms;
clock_gettime(start_time);
while (/) do
    wait control_period;
    Indy7.set_position(segment);
    if record mode then
        elapsed_time = clock_gettime(now) − start_time;
        angular_position[1..6] = read_encoder(1..6);
        set recorder_string to elapsed_time and angular_position[1..6];
        write(recorder_string);
    end
    Indy7_robot_control();
end
```

Figure 12. Simulation and motion control of the Indy7 robot for the path chosen to avoid the obstacle.

We compared the velocity of the trajectory generated by ROS2 MoveIt with the simulation and motion control results of the Indy7 robot. The initial joint angles of the Indy7 robot were set to zero degrees in both the virtual and real-world environments, requiring the robot to be controlled to its initial position on the planned path. Therefore, additional steps were taken to initialize the position of the Indy7 robot in both Unity and the MIO-5272. For the experiments, the total simulation and motion control time was set to one minute.
in order to assess the trajectory’s velocity. Figure 13 illustrates the velocity results of the generated trajectory and the simulation and motion control of the Indy7 robot. Simulation data were obtained using Algorithm 1, while motion control data were obtained through Algorithm 2. Each segment of the trajectory is represented in red, while the simulation and motion control of the Indy7 robot are shown by blue and green lines, respectively.

The path taken by the Indy7 robot to avoid the obstacle was restricted due to the inability to solve the inverse kinematics for a specific posture. Therefore, additional experiments were conducted to measure the velocity of all joints in the Indy7’s motion control. In the virtual environment, the path of the Indy7 robot’s end effector was planned to follow the shape of the alpha symbol. After generating the trajectory, both simulation and motion control were performed to measure the velocity of all joints. Figure 14 shows the simulation and motion control of the Indy7 robot moving along the shape of the alpha symbol. Figure 15 illustrates the velocity of the trajectory generated along the shape of the alpha symbol as well as the simulation and motion control of the Indy7 robot.

**Figure 13.** Velocities of Indy7 robot trajectory simulation and control result.

**Figure 14.** Simulation and motion control of the Indy7 robot moving along the shape of the alpha symbol.

In the experiment, the Indy7 robot followed a path in the shape of an alpha symbol that had been interactively planned using a mouse in the virtual environment. In this experiment, we performed both simulation and motion control. We compared the velocity of the Indy7 robot in the simulation and real-world environments, confirming that the segments generated by MoveIt were transmitted at precise times so that all movements
were synchronized. Thus, the motion control of the Indy7 robot adhered to the designated trajectory time of one minute set by MoveIt.

Figure 15. Velocities of trajectory generated along the shape of alpha symbol and simulation and motion control of the Indy7 robot.

However, because the simulation of the Indy7 robot and the real-time transmission of the trajectory occur simultaneously on the host device, the motion control time in the simulation often experiences delays, causing the speed to change abruptly in order to maintain positional synchronization. In the first experiment, abrupt changes were simultaneously observed at 11.10 s in joints 3 and 6 and at 23.10 s in joints 1, 2, 4, and 6. In the other experiment, changes were observed at 13.44 s and 15.58 s in joints 1, 4, and 6, at 17.92 s in joints 4 and 6, and at 50.48 s in joints 1, 2, and 5.

In future research, we will seek to smooth the trajectory velocity by considering task scheduling for simulation and segment transmission. On the MIO-5272, where only DDS communication with the host device and motion control of the Indy7 robot are performed, the speed does not experience delays or abrupt changes; as a result, the velocity of the Indy7 robot in real-world motion control and the simulation of the interactively planned path in the virtual environment reasonably followed the speed of the trajectory generated by ROS2 MoveIt.

4. Conclusions

In this paper, we have developed an interactive path editing and simulation system for motion planning and executed motion control based on the trajectory generated for the Indy7 collaborative robot. The interactive path editing system, implemented in Unity, allows users to plan the path of the Indy7 robot in the virtual environment by specifying X, Y, and Z coordinates using a mouse. The path information is transmitted to the ROS2 operating on a virtual machine on the same device, where the MoveIt motion planning framework generates the trajectory. The trajectory’s position and velocity data are transmitted in segments to the host device and MIO-5272 embedded board in real time, where they are applied to the simulation and motion control of the Indy7 robot.

To evaluate the system, we used the interactive path editing system to plan the path of the Indy7 robot using a mouse and measured the velocity of each joint. For the experiment, we simulated tasks involving the end effector avoiding an obstacle and drawing the shape of an alpha symbol. We then applied these simulation results to the robot’s motion control. The results showed that the Indy7 robot’s velocity during both simulation and motion control closely followed the trajectory generated by ROS2 MoveIt.

Our system can be used to simulate hazardous and complex tasks in the welding and material processing fields and apply them to robots. Additionally, robots equipped with
virtual torque sensors and a camera can be simulated in a virtual environment to detect and interact with dynamic obstacles. Based on sensor data obtained during simulations, reinforcement learning in the ROS can be employed to plan trajectories for complex tasks, requiring integration with sensor and camera interfaces. In particular, laser welding for various shapes of medical device parts demands complex trajectories. With this system, users can edit paths to quickly generate and simulate various trajectories and apply them to actual robots.

In future research, we plan to enhance the interactive path editing system to enable planning of the robot’s path with specified orientation and joint limits. Additionally, observation delays between the real-world and virtual environments can be significantly improved based on computing performance and communication between Unity and ROS2, helping to bridge the reality gap. Our aim is to control multiple intelligent robots through interactive path editing in a virtual environment enhanced by digital twin technology. Furthermore, we intend to incorporate sim-to-real transfer using reinforcement learning in order to develop an advanced intelligent control and simulation system.

Author Contributions: Conceptualization, T.Y. and B.W.C.; methodology, T.Y.; software, T.Y.; validation, T.Y.; investigation, T.Y.; resources, B.W.C.; data curation, T.Y.; writing—original draft preparation, T.Y.; writing—review and editing, T.Y. and B.W.C.; visualization, T.Y.; supervision, B.W.C.; project administration, B.W.C.; funding acquisition, B.W.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was financially supported by SeoulTech (Seoul National University of Science and Technology).

Data Availability Statement: The relevant data and source code are available upon request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References


40. Cho, S.Y.; Delgado, R.; Choi, B.W. Feasibility study for a python-based embedded real-time control system. *Electronics* 2023, 12, 1426. [CrossRef]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.