machines

MDPI

Article

Development of Standalone Extended-Reality-Supported Interactive Industrial Robot Programming System

Andrija Devic, Jelena Vidakovic * and Nikola Zivkovic

Research and Development Institute Lola Ltd., 70A Kneza Viseslava Street, 11030 Belgrade, Serbia; andrija.devic@li.rs (A.D.); nikola.zivkovic@li.rs (N.Z.)
* Correspondence: jelena.vidakovic@li.rs

Abstract: Extended reality (XR) is one of the most important technologies in developing a new generation of human–machine interfaces (HMIs). In this study, the design and implementation of a standalone interactive XR-supported industrial robot programming system using the Unity game engine is presented. The presented research aims to achieve a cross-platform solution that enables novel tools for robot programming, trajectory validation, and robot programming debugging within an extended reality environment. From a robotics perspective, key design tasks include modeling in the Unity environment based on robot CAD models and control design, which include inverse kinematics solution, trajectory planner development, and motion controller set-up. Furthermore, the integration of real-time vision, touchscreen interaction, and AR/VR headset interaction are involved within the overall system development. A comprehensive approach to integrating Unity with established industrial robot modeling conventions and control strategies is presented. The proposed modeling, control, and programming concepts, procedures, and algorithms are verified using a 6DoF robot with revolute joints. The benefits and challenges of using a standalone XR-supported interactive industrial robot programming system compared to integrated Unity–robotics development frameworks are discussed.

Keywords: robot programming; XR; augmented reality; virtual reality; Unity; physics engine; inverse kinematics; trajectory planner

check for updates

Citation: Devic, A.; Vidakovic, J.; Zivkovic, N. Development of Standalone Extended-Reality-Supported Interactive Industrial Robot Programming System. *Machines* **2024**, *12*, 480. https://doi.org/10.3390/machines12070480

Academic Editor: Raul D. S. G. Campilho

Received: 30 May 2024
Revised: 5 July 2024
Accepted: 9 July 2024
Published: 17 July 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Robots are important in high-mix, low-volume manufacturing due to their versatility in performing manufacturing tasks [1]. There is a high demand for the development of robotics and automation solutions that are less expensive, safer, easier to install and re-program, where the operator can work as an assistant to robots to solve unpredictable problems, and that can be adapted by small- and medium-sized enterprises (SMEs) [2,3].

Programming of industrial robots using conventional robot programming methods is time-challenging and often demands expert knowledge [1–3]. The most widely used programming method for industrial robots is programming using teaching pendants [1]. This method, also known in the literature as the Lead-Through method [4], is an online programming method that typically takes considerable time, especially if a robot has more degrees of freedom than the joystick of the teaching pendant (due to the separate programming of the position and orientation of the robot's end-effector). Using conventional Walk-Through online programming, the human operator directly interacts with the robot, i.e., physically moves the robot to the desired configuration [4]. This method allows intuitive robot programming; however, it is limited as only a few lightweight robots offer the necessary zero-torque control [4]. Online robot programming methods result in downtime (the robot must be present during programming), which additionally prolongs changeover times when users introduce new applications. Besides the aforementioned online programming methods, there are also offline programming methods that do not

require the physical presence of the robot during programming [1]. Offline programming methods require high-fidelity 3D modeling of robot and its workspace in a virtual environment and are time-consuming [1]. For example, if a new object of manipulation is introduced into the robotic application, its 3D model has to be developed using appropriate software. The simulation module needs to be tested on real robots before actual offline programming can be performed, and these tests may require more effort to refine the models to achieve high fidelity [1]. Despite their efficiency, most commercial solutions are subjected to high-cost licenses and have limited flexibility and range of applications [1]. One of the significant factors prohibiting the widespread application of robots by small and medium enterprises (SMEs) is the high cost and necessary skill for robot programming and re-programming [1,5,6]. In the last several years, there has been an effort to increase robotic and automation technology in SME pipelines. Although some progress has been made, there are still challenges to be overcome [5,6].

The arising concept of Industry 5.0 (i5.0) refers to the human-centric approach in manufacturing systems and focuses on enabling technologies for improving human–machine interaction, such as extended reality (XR), which has become a popular multidisciplinary research field over the last decade. XR, comprising augmented reality (AR), virtual reality (VR), and mixed reality (MR), has been increasingly used in networks of connected physical systems and human–machine communication. XR tools are used in several aspects of the manufacturing process, some major examples being assembly assistance, product design, maintenance, operators' training, service, repair, etc. [6,7]. Recently, XR-assisted software solutions have been proposed in the robot programming domain [1,2,4,7–17].

This article presents a comprehensive procedure for developing a Unity-based standalone XR-enhanced interactive industrial robot programming application. The system is designed to offer cross-platform solution allowing users with limited robot programming skills, as well as experts, to leverage the advanced visualization capabilities of XR technologies for programming complex robot trajectories and facilitating the debugging process.

The methodology proposed within the research involves a comprehensive approach involving elements of robot programming and control within a Unity-based XR environment. Its key components are as follows:

1. Robot modeling in Unity.
2. Control design in Unity (including solving inverse kinematics, developing a trajectory planner, and designing motion controllers).
3. Development of XR-enhanced robot functionalities in Unity.
4. Development of post-processor for offline robot programming languages.
5. Integration and verification.

The conceptual design and implementation of the system are presented; it offers novel XR-supported robot programming approaches, such as AR/VR-assisted Linear-Path Lead-Through (AR/VR-LT), AR/VR-assisted Linear-Path Walk-Through (AR/VR-WT), and AR/VR-assisted Debugging Tool. Industrial 6 DoF robot RL15 [18,19] is used to verify developed modeling, control, and programming concepts, procedures, and algorithms.

The result of this study is an open-source application, L-IRL XR, release 1, which is publicly available for use and further development at [20] under a GPL license. The download links for the executable and source files are included in [20]. The proposed framework is a cross-platform solution that is executable with a uniform code that is compatible across various devices (PC, macOS, Android, and iOS).

This article is divided as follows. In Section 2, related work and the main contributions of the article within the domain of research interest are given. The robot modeling procedure and control design strategy in Unity are presented in Section 3. Section 4 details the design of the XR-enhanced robot programming and simulation framework, including the system architecture and a description of the developed robot programming functionalities. Discussion is presented in Section 5; finally, concluding remarks are given in Section 6 of the article.

2. Related Work and Key Contributions

Considering the literature and available commercial solutions [1,2,4,7–17,21–28], the overall employment of XR HMI in industrial robotics can be separated into three categories: (1) robot programming, (2) system engineering, and (3) personnel training (see Figure 1). In the domain of robot programming, XR-technology represents a significant aid via its visualization tools as it allows better physical context awareness for the programmers.

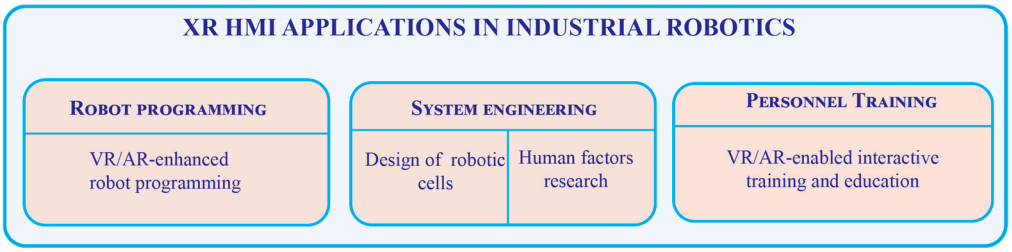


Figure 1. XR-assisted HMI applications in industrial robotics: Robot programming [1,2,4,7–17]; System engineering-Design of robotic cells [21,22], System engineering—Human factors research [23–26]; Personnel training [27,28].

Researchers seek ways to design intuitive and safe robot programming tools using emerging technologies. Often, simulation technology advancements depend on budgets that are available to build simulators and train people to use them [29,30]. Game engines are considered a cost-effective and flexible solution for developing an XR application [8]; over the years, the expansion of game engines and their capabilities has led to their increasing utilization in research involving robotics and multibody system simulation. The significant advantage of game engines like Unity [31] and Unreal Engine [32] over dedicated robot simulation software—which include frameworks and libraries that are predominantly used as dynamics engines, such as MuJoCo [33], DART [34], and Simbody [35] and standalone simulators such as Gazebo [36], Coppelia Robotics [37], Cyberbotics Webots [38], etc.—is their integrability with different AR/VR SDKs. Compared to specialized software such as ROS-Gazebo [39], game engines lack some of the control design tools (inverse kinematics solvers, path planners, motion controllers); for this reason, developers typically use the integration of game engines with a dedicated robotics development framework [40].

Unity, one of the most popular game engines, is a leading platform for the development of XR applications with high-quality 3D rendering, which can be deployed across a variety of platforms and can integrate different XR SDKs such as Google's ARCore, Apple's ARKit, Vuforia, and Microsoft's MRTK; these are some of the most prominent frameworks at the moment [33,41,42]. Conventional offline industrial robot programming can be performed in robot's joint space or by programming the robot's tool center point (TCP) motion in Cartesian space. In Cartesian space, TCP motion is programmed by using the following: (1) point-to-point (PTP) motions where a user defines the initial and final poses while the trajectory planner defines the path between the two poses, according to the specific criteria; (2) continuous-path (CP) motion, where a user defines a path to be followed by TCP [43], such as linear, circle, etc. In order to achieve the TCP motion programming in Cartesian space, several integrations of Unity with ROS [44] (ROS–Unity suite) and offline/online programming environments like RoboDK [45] have been proposed recently; these leverage the ROS/RoboDK-integrated trajectory planner and its integrated numerical inverse kinematics solvers. The authors of [1,8,11–14] used the ROS–Unity suite; the work presented in [16] is based on the RoboDK/Unity suite, and [15] is based on the Matlab/Unity suite.

Compared to using Unity–robotics development framework software suites, a standalone XR-supported industrial robot programming system delivers several advantages:

- Reduced latency and computational overheads;
- Easier customization for developers;

- Elimination of the need for additional computers running ROS or similar software, thereby reducing the necessary hardware requirement and enhancing user mobility.

A standalone app eliminates the need for inter-process communication between Unity and ROS or similar software, which can introduce additional latency and computational overheads. This results in more streamlined and efficient execution. At the same time, there is no need for additional software installations or their integration with different devices, nor is there a need for developers to become acquainted with additional software, such as ROS [44] (ROS2 v.humble), RoboDK [45] (v4.0.0.), and similar. The development process in only one IDE is simpler than that of two IDEs for reasons such as simpler debugging, simpler testing, and better customizability.

The authors of [17] present a framework enabling end-users to manage complex robotic workplaces using a tablet and AR. The article proposes two different system integrations: standalone and integration based on ROS. Meanwhile, coding is performed in Python. However, a standalone solution does not include an augmented robot; rather, it uses a real robot, which suggests that robot downtime is performed when programming in AR. The authors of [2] propose an AR-supported robot programming system developed in C/C++ language. Regarding the robot programming method, pick-and-place programming (PTP robot programming) is proposed in [2], while [1,2,4,8,11,13] propose emulated Walk-Through and emulated Lead-Through [4,16].

The Unity-based solutions proposed in [1,8,11–13,16] do not tackle the integration of the robot dynamics in an augmented/virtual robot model, nor the setting of the parameters of the X drive component for articulation body. Transformation of the left-handed Unity frames to end-user frames is not discussed.

Motivation for the development of the proposed XR-supported robot programming system includes the following aspects.

- Develop AR/VR-supported robot programming framework that enables the following:
  - (1) Industrial robots programming using conventional offline robot programming methods such as CP robot programming in an online regime in augmented/virtual reality. In this way, the downtime of conventional offline robot programming is avoided.
  - (2) Using the advanced visualization capabilities and spatial awareness that the XR environment provides (compared to 2D-display-based virtual robot programming frameworks) for verifying the programmed path and for the debugging of robot programs.
- Develop cross-platform solution for a more consistent and adaptable experience for both developers and end-users.
- Standalone solution which integrates augmented/virtual robots.
- Open-source and customizable solution for developers. Contribute to research and developer communities by making the solution open-source.
- The main contributions of the research can be summarized as follows:
- Providing open-source, cross-platform, standalone XR-supported interactive robot programming system prototype.
- Development of the method for closed-form inverse kinematics implementation in Unity based on frames transformations.
- Development of the method for motion controller set-up in Unity that avoids oscillatory response in robot joint motion and takes into account installed actuators' capabilities.
- Development of novel tools which allow for linear motions robot programming in Cartesian space in an AR/VR environment.
- Development of a novel AR/VR-assisted debugging system.

3. Robot Modelling and Control Design in Unity

The design of standalone XR-enhanced robot programming tools in Unity requires the development of a robot's 3D model, the implementation of inverse kinematics algorithm

for a specific robot, and, depending on the application, may require the implementation of the trajectory planner. Coding is performed in C#, as Unity supports the C# programming language natively.

The main requirements that were set for the development are as follows:

- The developed system should be cross-platform or cross-platform-adaptable;
- The developed system should integrate robot dynamics;
- The developed system should be able to verify the programmed path in an extended reality environment;
- The developed system should integrate a trajectory planner, enabling the programming of CP motion along linear paths in Unity;
- The developed system should be open-source and customizable;
- The developed system should integrate a post-processor for exporting the programmed paths as offline instruction code readable by the robot controller.

The architecture diagram for the proposed framework is given in Figure 2.

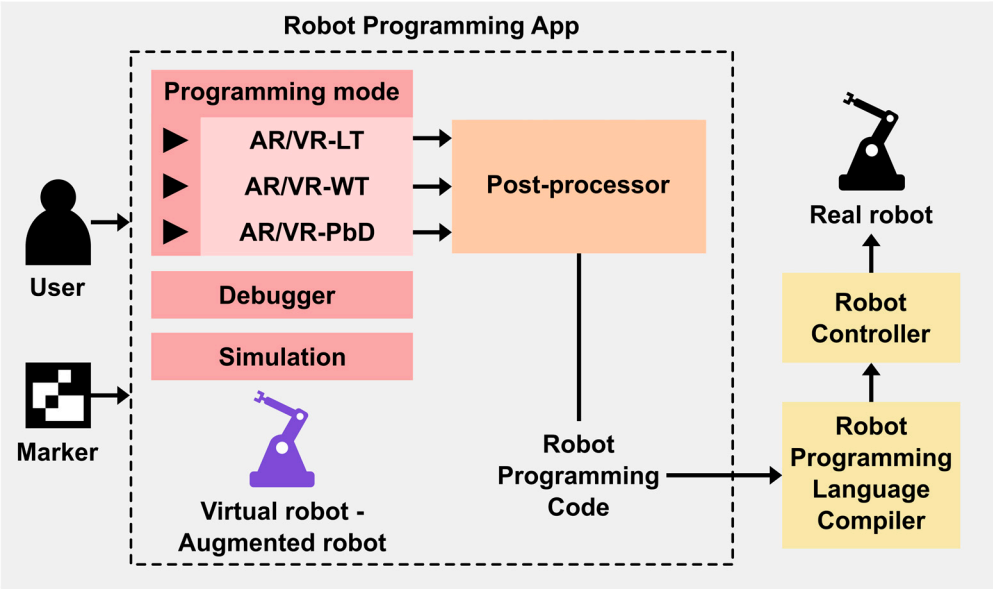


Figure 2. System architecture for the proposed XR-assisted robot programming framework.

The approach to robot modeling in Unity is to create a detailed 3D robot model in a 3D-design software and import it into Unity. Herein, the import is performed by using XML file formats such as URDF (Universal Robot Description Format) [46] and glTF (GL Transmission Format) [47] (used for textures), which are open-source standards equipped with elements and attributes specific to robot kinematic and dynamic structure. SolidWorks [48] is used to design a 3D robot model as it allows model exportation to the mentioned XML formats using free and open-source add-ons [49,50]. The creation of the robot model in Unity is structured into Procedure 1, Appendix A.

Unity's built-in physics engine, Nvidia PhysX, is used in this study. Apart from Nvidia PhysX, there is a possibility of using the MuJoCo (Multi-Joint dynamics with Contact) engine as a plug-in library. However, MuJoCo computes convex-hull approximations based on the 3D model, which may cause difficulties for highly detailed robot models with concave surfaces regarding visual fidelity and collision detection, enforcing additional efforts with implementation. Herein, the Articulation Body class [51] from Nvidia PhysX is used for the robot dynamics integration and motion controller set-up. In this research, a method for setting motion controller parameters, taking into account the applied actuators' capabilities, has been proposed.

Several AR-supported interactive robot programming tools have been developed and integrated within one marker-based Unity-developed application, which is presented

herein. Vuforia [41], an AR software development kit for smart portable devices (smartphones, tablets, head-mounted see-through devices, or smart glasses) that provide essential tools for creating AR applications, is used for generating AR robot image overlay in a real laboratory environment. The developed AR app is cross-platform within the realm of PC, macOS, Android-based, and iOS-based devices. Due to the system's cross-platform adaptability, implementing the developed robot programming framework into a virtual reality (VR) environment is straightforward: developers simply need to utilize the appropriate VR SDK instead of Vuforia. This study used the Oculus Rift S VR headset to experience VR; its SDK is Meta XR All-in-One SDK (UPM) [52]. GUI development in Unity is based on the mixed reality toolkit [42].

3.1. Implementation of the Inverse Kinematics and Frames Transformations

The connection between the Euler angles, typically used by robot programmers to define the orientation of the TCP with regard to the reference frame, is commonly realized via a rotation matrix. In Equation (1), ${}^{1}\mathbf{R}_{2}$ is a 3×3 rotation matrix, also known as a cosine direction matrix, representing the rotational transformation of the first frame (herein named FRAME1) to the second frame (herein named FRAME2). Columns of ${}^{1}\mathbf{R}_{2}$ can be regarded as scalar products of unit vectors $\mathbf{x}_1$, $\mathbf{y}_1$, $\mathbf{z}_1$ constituting FRAME1 and unit vectors $\mathbf{x}_2$, $\mathbf{y}_2$, $\mathbf{z}_2$ constituting FRAME2. The rotation matrix and the position vector form a 4×4 homogeneous transformation matrix (HMT), which is widely used in the robotics community to define the poses (positions and orientations) of the frames attached to robot links.

$$ {}^{1}\mathbf{R}_{2} = \begin{bmatrix} \cos(\mathbf{x}_2,\ \mathbf{x}_1) & \cos(\mathbf{x}_2,\ \mathbf{y}_1) & \cos(\mathbf{x}_2,\ \mathbf{z}_1) \\ \cos(\mathbf{y}_2,\ \mathbf{x}_1) & \cos(\mathbf{y}_2,\ \mathbf{y}_1) & \cos(\mathbf{y}_2,\ \mathbf{z}_1) \\ \cos(\mathbf{z}_2,\ \mathbf{x}_1) & \cos(\mathbf{z}_2,\ \mathbf{y}_1) & \cos(\mathbf{z}_2,\ \mathbf{z}_1) \end{bmatrix}. \quad (1) $$

The Unity engine was not originally designed for robotics applications; as a result, certain conventions and practices commonly used in robotics may require additional adaptation. Key conventions for kinematic modeling in robotics, contrary to Unity, use right-handed frame orientation. Poses of a TCP in Unity editor are defined using Cartesian coordinates x, y, z, and Euler angles X, Y, and Z, representing the angles of rotation (in the clockwise direction) about the x, y, and z axes with regard to left-handed Unity BASE reference frame (here denoted as BASE_U) [53]. Unity uses the z-x-y Euler angles sequence of extrinsic rotation [54]. This sequence corresponds to the y-x-z sequence of intrinsic rotation. In order to control the robot movements by a user in Unity in Cartesian space, a solution for the inverse kinematics problem (IK algorithm) for the selected robot has to be integrated. Integrating the IK algorithm defined with regard to reference frame BASE_IK, oriented differently to BASE_U frame, implies the mapping of the TCP pose from the BASE_U frame to the BASE_IK frame. The numerical values of the TCP rotation matrix and position vector with regard to the BASE_IK frame are the input into the IK algorithm. Transformation from any left-handed frame to any right-handed frame by application of three successive rotations is impossible. For this reason, it is advisable for the left-handed BASE_U reference to be selected to match the axes of frame BASE_IK as closely as possible in order to obtain the simplest TCP pose transformations using axes transformation. This is demonstrated in detail in a case study represented in Section 3.3.

The IK algorithm is executed for each rendering frame based on Unity's frame rate. The IK algorithm includes checking whether the defined joint positions are within their limits based on the robot's workspace. Speeds are calculated for each rendering frame using the Euler differentiation method to verify that the defined joint speeds remain within the motor limits.

3.2. Robot Dynamics Integration and Controller Design

Dynamics play a highly significant role in the synthesis of control algorithms, the mechanical structure design, and the motion simulation of robots. For the purpose of robotic research and with the intention of being used for realistic physics behaviors in the context

of industrial applications simulation, in addition to standard rigid body dynamics, NVIDIA PhysX SDK offers the reduced coordinate articulations feature based on Featherstone's Articulation Body Algorithm (ABA) for forward dynamics computation [51,55]. Energy in mechanical systems can be stored in a spring or an inertia. In NVIDIA PhysX, articulations are driven by "joint acceleration springs" [51]. ArticulationDrive is a struct component implemented in Unity Engine PhysicsModule, which simulates the application of forces and torques to the interconnected bodies. The drive component executes force/torque F using user-defined stiffness and damping components:

$$F = \text{stiffness}(\text{currentPosition} - \text{target}) - \text{damping}(\text{currentVelocity} - \text{targetVelocity}). \quad (2)$$

In the scientific literature related to Unity and within Unity tutorials, there is no general rule or recommendation to set the stiffness/damping parameters, so users usually tune them case by case to match the real robot as closely as possible. Herein, we propose a simple rule for setting parameters of the X drive component of the Articulation Body, which avoids oscillations in the robot links motion and takes into account motor possibilities.

Adopting $F = m\ddot{x}$ in Equation (2) and converting the equation into continuous error domain yields:

$$m\ddot{e} - b\dot{e} + ke = 0, \quad (3)$$

where k is the spring stiffness constant, b is the damping coefficient, and $e = e(t)$ is the error signal, e = target – current position. Second order systems in general have complex-valued natural responses. The characteristic equation of the system given in Equation (3) is:

$$ms^2 - bs + k = 0, \quad (4)$$

The second order polynomial Equation (4) has two solutions (pole locations of the system):

$$s_{1,2} = \frac{b \pm \sqrt{b^2 - 4km}}{2m}, \quad (5)$$

If the roots have an imaginary component, i.e., $b < 2\sqrt{km}$, then the response has an oscillatory component. To avoid oscillations in system response Equation (5), the damping coefficient $b \geq 2\sqrt{km}$ has to be adopted. The critically damped case where $b = 2\sqrt{km}$, and the poles of the second order system coincide on the real axis, is the fastest response without oscillations.

Within X drive parameters selection, it can be adopted that the spring stiffness constant k and the damping coefficient b correspond to the proportional K_P and derivative K_D gains of the positional PD controller, respectively; mass m is the mass of the link, and the relation $K_{Di} = 2\sqrt{K_{Pi}m_i}$ is adopted for $i = 1, 2 \ldots n$. In order to achieve realistic robot simulation, herein we assume that the maximum force/torque generated by an actuator (drive limit parameters in X drive component), the input of which is the output of the feedback controller; this corresponds to the assumed maximum allowed error in position E_{Pmaxi} and to the assumed maximum allowed error in the velocity E_{Vmaxi} of the joint i:

$$\tau_{\text{max}i} = K_{Pi}\text{E}_{\text{Pmax }i} + K_{Di}\text{E}_{\text{Vmax}i}, \quad (6)$$

where the maximum output (after the gear transmission) driving torque/force generated by the motor in joint i, $\tau_{\text{max}i} = r_i\tau_{M\text{max}i}$, r_i is the gear ratio, $\tau_{M\text{max}i}$ is the maximum motor torque at the rotor shaft, $\text{E}_{\text{Pmax}i}$ is the maximum allowed position error, and $\text{E}_{\text{Vmax}i}$ is the maximum allowed velocity error. Solving Equation (6) for $K_{Di} = 2\sqrt{K_{Pi}m_i}$, and adopted assumed maximum acceptable error in position and velocity for i = 1, 2, …, n values $K_{\text{D}i}$, $K_{\text{P}i}$, have been obtained:

$$K_{Pi} = \frac{2mE_{\text{Vmax}i}^2 + E_{\text{Pmax}i}\tau_{\text{max}i} - 2\sqrt{m^2E_{\text{Vmax}i}^4 + mE_{\text{Pmax}i}\tau_{\text{max}i}E_{\text{Vmax}i}^2}}{E_{\text{Pmax}i}^2}, \quad (7)$$

from which the derivative K_D gain is calculated as:

$$K_{Di} = 2\sqrt{K_{Pi}m_i}, \tag{8}$$

3.3. Case Study-6DoF Industrial Robot RL15

For the verification of the proposed procedures, algorithms, and programming concepts, the 6DoF serial robot with revolute joints RL15 is considered, Figure 3. Technical parameters of RL15 robot are given in Appendix B.



Figure 3. 6DoF robot RL15: (**a**) in laboratory setting; (**b**) 3D model designed in SolidWorks; (**c**) Unity model.

For the RL15 robot, the inverse kinematics solution in [18,19] uses a right-handed frame presented in Figure 4a, with an adopted notation that A is the rotation about the x-axis, B is the rotation about the y-axis, and C is the rotation about the z-axis (positive in a counterclockwise direction). The IK solution in [18,19] uses specific Euler angles sequence of intrinsic rotations to form the TCP rotation matrix with regard to the BASE_IK frame.

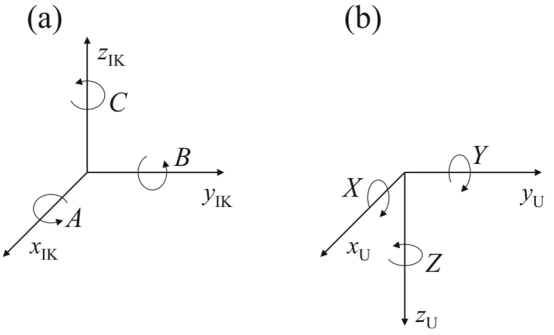


Figure 4. (**a**) BASE_IK frame; (**b**) selected BASE_U frame.

In order to obtain the TCP rotation matrix and the TCP position with respect to the BASE_IK frame (shown in Figure 4a)—this was used within the previously developed IK [18,19], a left-handed BASE_U reference frame (shown in Figure 4b) is selected to match the axes of the frame in Figure 4a as closely as possible, in order to obtain simplest TCP poses transformations. From Figure 4, it follows:

$$x_{IK} = x_U, y_{IK} = y_U, z_{IK} = -z_U, \tag{9}$$

$$A = -X, B = -Y, C = Z, \tag{10}$$

An arbitrary Euler angle sequence can be used to calculate the rotational matrix, which is a numerical input into the IK algorithm for the RL15 robot developed in [18,19]. If the y-x-z sequence of intrinsic rotation is adopted, then the following is obtained:

$$^{\text{BASE_IK}}\mathbf{R}_{\text{TCP}} = \mathbf{Rot}(y,\ B)\mathbf{Rot}(x,\ A)\mathbf{Rot}(z,\ C) = \begin{bmatrix} \cos B\cos C + \sin A\sin B\sin C & \cos C\sin A\sin B - \cos B\sin C & \cos A\sin B \\ \cos A\sin C & \cos A\cos C & -\sin A \\ \cos B\sin A\sin C - \cos C\sin B & \sin B\sin C + \cos B\cos C\sin A & \cos B\cos A \end{bmatrix} \quad (11)$$

The IK algorithm for the RL15 robot developed in [18,19] is implemented in Unity in C# language.

For the RL15 robot, the controllers' gains (X drive class stiffness and damping parameters) using the presented method in Section 3.2.—, i.e., Equations (10) and (11) for $i = 1, 2\ldots, n$ links—are given in Table 1. Motors' limits manually entered into the X drive class are given in Table A2 in Appendix B.

Table 1. X drive class stiffness and damping parameters for robot RL15.

Joint k	E_{Pmax} [rad]	E_{Vmax} [rad/s]	$k_i = K_{Pi}$	$b_i = K_{Di}$
1	0.02	0.02	95,283.3	11,516.7
2	0.02	0.02	221,489	8160.72
3	0.02	0.02	100,166	5734.751
4	0.02	0.02	21,165	1979.84
5	0.01	0.01	75,287.2	844.82
6	0.005	0.005	36,626.9	363.121

The motion of the RL15 robot based on the proposed modeling and controller set-up methods is shown in the video provided in the Supplementary Materials. In AR applications, the robot is scaled down for the purposes of the demonstration video. The user sets the desired path in the Cartesian coordinate system, and the inverse kinematics calculates the movement of the joints.

4. Design of XR-Enhanced Robot Programming and Simulation Framework

In this section, the interactive AR/VR-enhanced, standalone robot programming and simulation tools developed in this research are presented. Several robot programming concepts and tools are proposed:

- AR/VR standalone Linear-Path Lead-Through robot programming tool (AR/VR-LT);
- AR/VR standalone Linear-Path Walk-Through robot programming tool (AR/VR-WT);
- AR/VR standalone Robot Programming by demonstration tool (AR/VR-PbD);
- AR/VR standalone Simulator tool for offline robot programming languages scripts (AR/VR-S);
- AR/VR standalone Debugging tool for industrial robot arms in AR/VR environment (AR/VR-D).

There is some discrepancy regarding the terminology used for robot programming methods in the literature. To avoid confusion with the classifications and terminology, the terminology used herein is in compliance with that in [56,57].

To verify the proposed programming concepts requiring trajectory generation, a trajectory planner (TP) has been developed and integrated into Unity, Appendix C. The GUI for the developed AR/VR-enhanced robot programming tools is given in Figure 5.

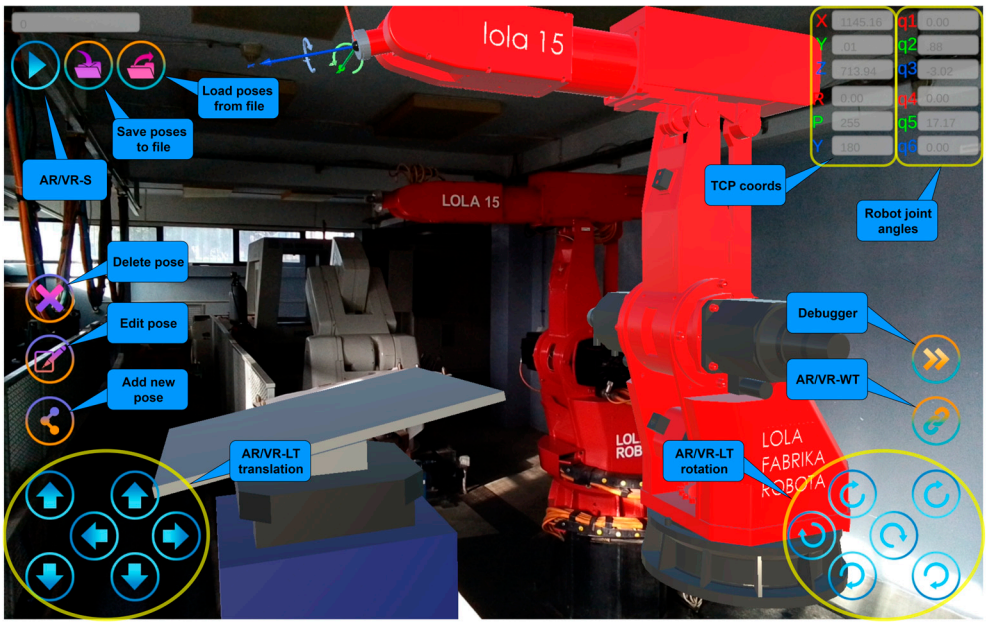


Figure 5. The GUI of the developed AR/VR−enhanced robot programming application.

4.1. Description of the Functionalities of the XR-Enhanced Robot Programming Tools

4.1.1. AR/VR Linear-Path Lead-Through

AR/VR-LT implies a Lead-Through programming method where the TCP motion is programmed using a teaching pendant in an AR/VR environment. GUI has been created with buttons and functionalities to emulate teaching pendants in AR/VR (Figure 6). The user can move TCP using developed GUI buttons and, after moving the TCP to the desired poses, they can choose to generate linear path segments between the selected waypoints in Cartesian space, or they can save the poses into the memory for later usage.

4.1.2. AR/VR Linear-Path Walk-Through

The AR/VR-WT robot programming method is conceptualized as an integration of the Walk-Through method and linear CP robot programming method in an AR/VR environment. This method harnesses the advantages of time-efficient and intuitive Walk-Through robot programming [1,4] and applies them to the linear CP method. AR/VR-WT implies the manual "dragging" of the TCP position and/or the orientation of a robot in an AR/VR environment by using an emulated robot handle—a VR headset joystick or a phone/tablet for an AR implementation. In the AR environment, to allow the user to move the robot's TCP as intuitively as possible, the TCP coordinate frame becomes a child of the AR camera frame, implying that the TCP pose follows the motion of the phone/tablet executed by a user. In the VR environment, the user moves the joystick to change the TCP pose according to the position and orientation of the joystick. After moving the TCP to the desired poses, the user can choose to generate linear path segments between the selected waypoints in Cartesian space, or they can save the poses into the memory for later usage. The AR/VR-WT robot programming method is depicted in Figure 7a (AR) and Figure 7b (VR).

4.1.3. AR/VR- Programming by Demonstration

The AR/VR-PbD method is built upon the previously described AR/VR-WT tool, where the TCP is dragged by the operator in an AR/VR environment. Contrary to the AR/VR-WT method, where the linear trajectory between desired waypoints is defined and interpolated by the planner, here, the trajectory generation is performed by recording the motion of the TCP. In other words, a trajectory is generated directly by the operator's hand movement, which the TCP follows. The validity of the execution of the programmed

movement of the physical (real) robot is ensured by the inverse kinematics calculation (with accompanying IK checks on joints' position and speed described in Section 3.1) for every rendering frame, as well as by the integration of dynamics and the control method using X drive from the Articulation Body class into the system (for a more detailed explanation, see Sections 3.1 and 3.2). The AR/VR-PbD inherits the intuitiveness of the conventional Walk-Through robot programming method in an AR/VR environment (see Figure 8a (AR) and Figure 8b (VR)).

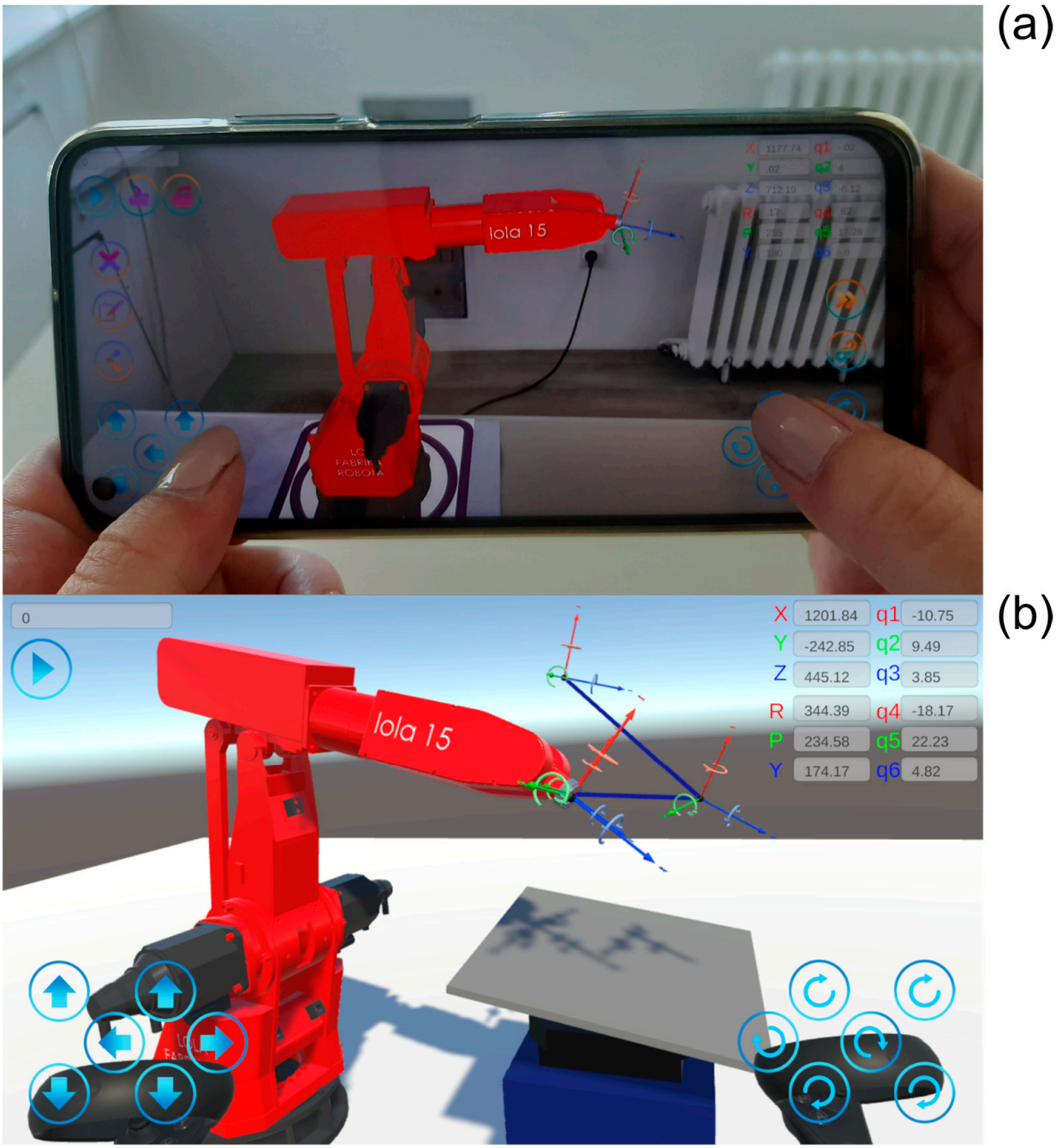


Figure 6. GUI buttons for AR/VR−LT programming method: (**a**) AR environment; (**b**) VR environment.

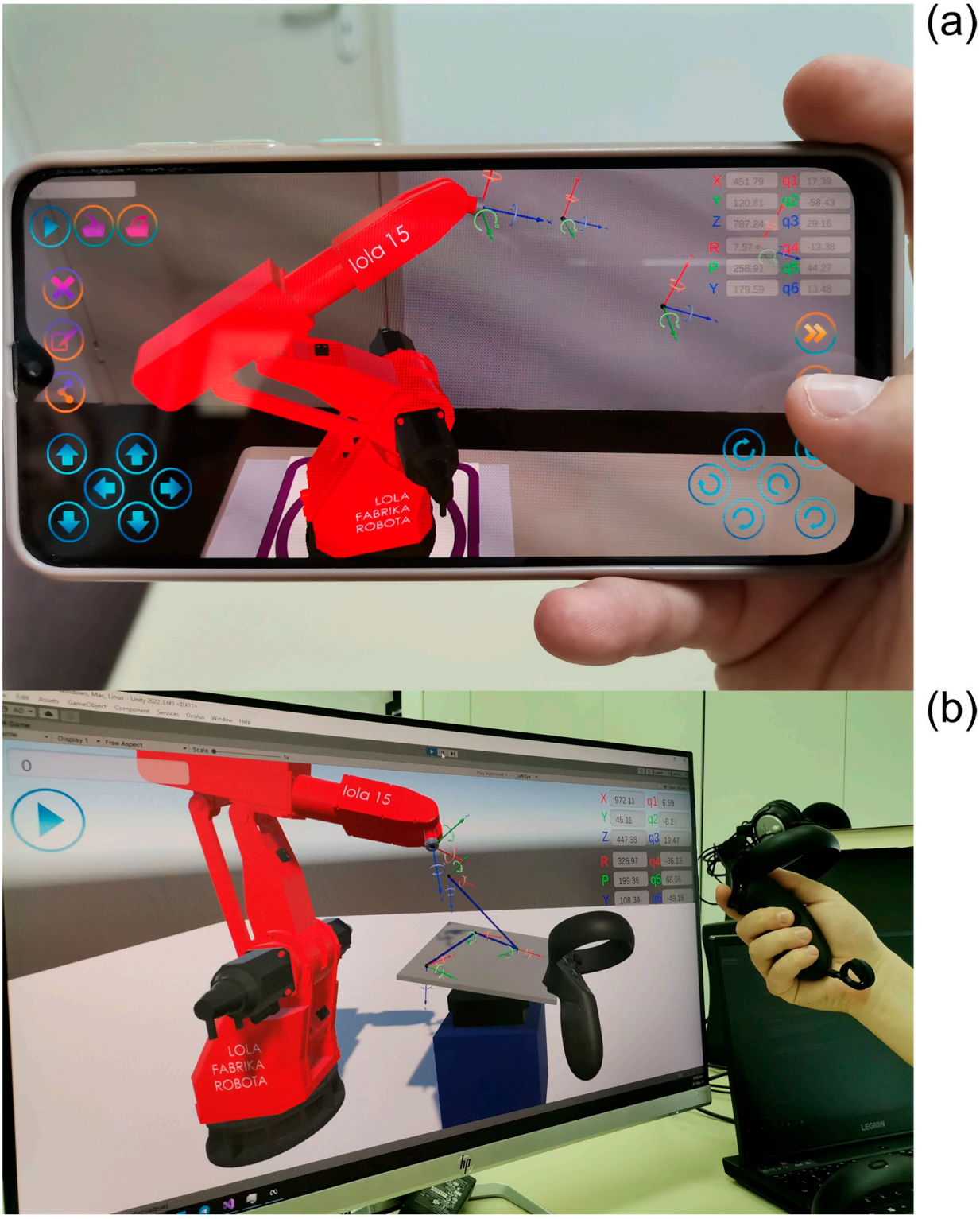


Figure 7. Robot programming by AR/VR−WT method: (**a**) AR environment; (**b**) VR environment.

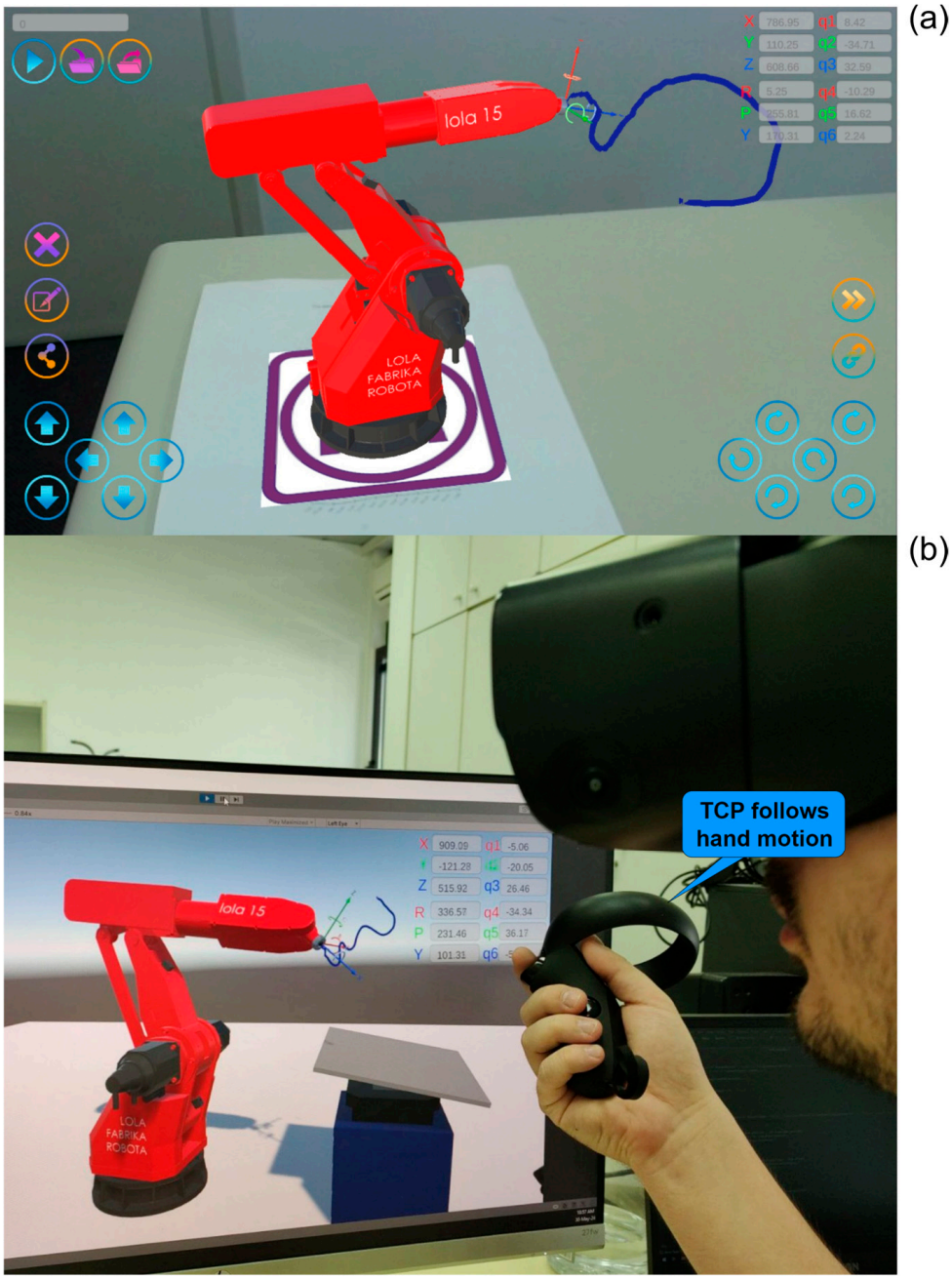


Figure 8. AR/VR−PbD programming method: (**a**) AR environment; (**b**) VR environment.

4.1.4. AR/VR-Simulator

In the simulator mode (AR/VR-S), Figure 9a,b, a visual playback of the programmed robot motion is conducted in an AR/VR environment, which is useful for the programmed trajectories advanced visual inspection and verification in a safe environment. Users can inspect the robot path from a close distance, from multiple directions, use the integrated slider, pause the operation, "inspect" programmed speed, etc. The attainable trajectory is defined in one of two ways:

(1) Using previously described AR/VR-supported tools;
(2) Reading poses from a text file obtained as the output from offline robot programming languages.

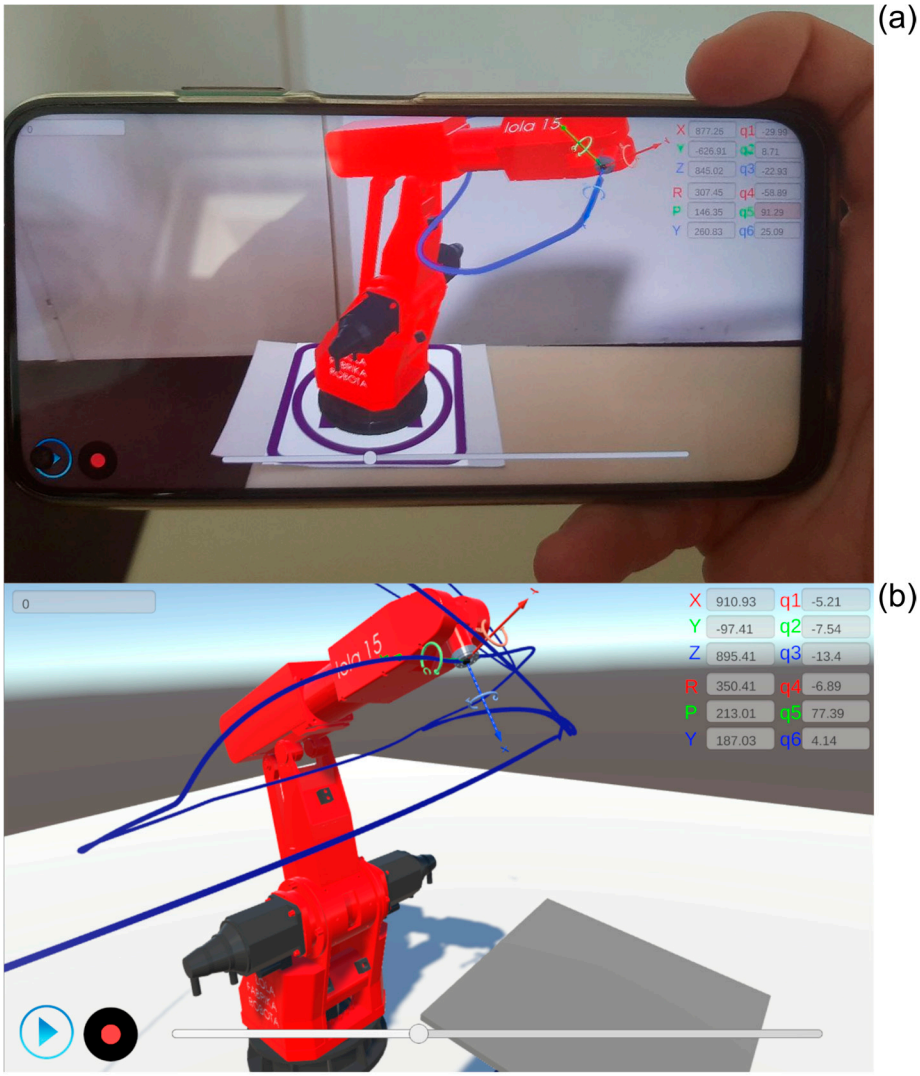


Figure 9. Trajectory validation using AR/VR−S: (**a**) AR environment; (**b**) VR environment.

Regarding the latter, a parser for reading outputs of selected offline robot programming languages has to be developed. In this research, a simple parser specifically designed to interpret the output of the L-IRL robot programming language [18] has been developed.

4.1.5. AR/VR-Debugger

Industrial robots' programming presents a unique set of challenges regarding the debugging problem. One of the challenging situations in this sense is a debugging process for continuous-path TCP motion due to the highly nonlinear and coupled nature of the inverse kinematics solution, which can make the cause-and-effect relationships between the robot's joint motion and a TCP motion in Cartesian space difficult to interpret. AR/VR provides many opportunities for enhancing debugging by providing immersion and allowing the developer to visualize problematic TCP poses and path segments that are superimposed in situ in the real world, intuitively displaying the limitations and discontinuities in the robot's real-world view [7,10].

This research introduces an AR/VR Debugger Tool (AR/VR-D), which is designed to provide straightforward motion application program debugging and trajectory correction to ensure achievability within the specific robot's workspace parameters. Firstly, by using the AR/VR-D tool, the user is able to inspect the unattainable TCP path in the AR/VR environment. Figure 10a shows the example of the unattainable TCP path, i.e., an array of successional poses that the robot's TCP should go through for a programmed robot

motion with some of the poses classified to be unattainable by the trajectory planner due to the robot limits being exceeded within the robot's workspace. The unattainable path is generated by implementing a modified IK algorithm, in which joint angle limits have been removed. The user can then examine the unattainable TCP trajectory, helped by the visual color presentation of the predefined TCP path and a vertical stack of numerical values of the robot's joint angles, which are calculated using original IK algorithm (with accompanying checks on joint limits) (see Figure 10a). In Figure 10a, it can be seen that the TCP path can take a blue, yellow, or red color: unachievable poses are shown in red; poses nearing unattainability—specifically, those within a defined half-range of 10° to the joint limits—are shown in yellow. In the case presented in Figure 10a, the impossibility of achieving programmed TCP trajectory is a consequence of exceeding the maximum allowed angle of the 5th link (highlighted in red). The slider is added so the user can slide along the trajectory in order to visualize the mechanism between TCP pose change and joint position changes in critical path segments.

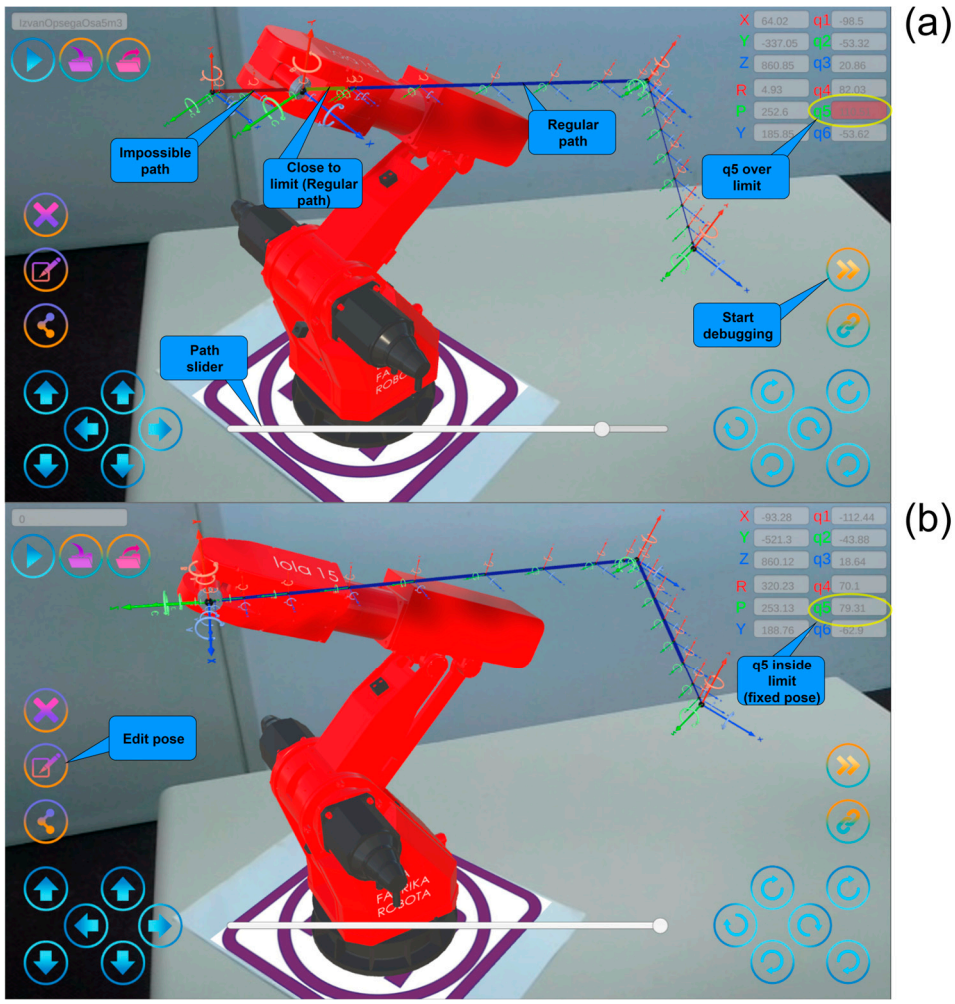


Figure 10. AR/VR Debugger Tool (AR/VR−D) GUI: (**a**) an example of unattainable TCP path; (**b**) correcting unachievable TCP trajectory in EDIT mode.

The user can correct unachievable TCP trajectory in EDIT mode by selecting and editing the unattainable TCP poses (Figure 10b). An unattainable pose is selected by bringing the TCP close to it and by pressing and holding the EDIT button. When the TCP pose snaps to an unattainable pose, the user is enabled to freely move the TCP to find a new attainable pose. The displayed robot joint angles and TCP Cartesian coordinates facilitate the editing of the problematic pose.

4.2. System Architecture and Workflow Diagram

Figure 11 shows the system components and information flow for each developed function. The general, more detailed flow chart of the system is given in Figure 12.

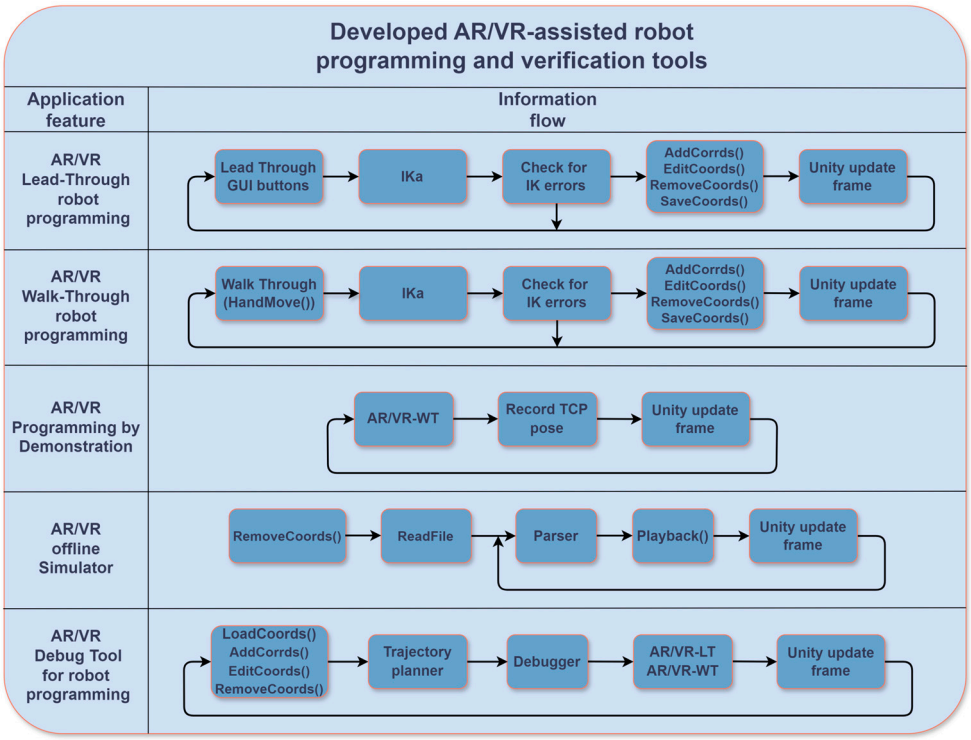


Figure 11. Developed standalone XR-assisted robot programming and verification tools.

The integrated functionalities that have been developed and implemented as C# scripts in Unity for the tools presented in Figure 11 are given in Figures 13 and 14 below.

AR/VR Walk-Through (AR/VR-WT) allows the user to move the robot TCP by "dragging". Figure 13 shows the flow chart of the developed AR/VR-WT feature. Let us assume that the user is using an AR application, and that the AR-experiencing device is a smartphone/tablet. The Update() function is the main function of C# code in Unity, and it is executed once for each rendering frame. By holding the AR/VR-WT button (see Figure 5) at the beginning of the first frame, a new (temporary) GameObject called Temp is created; TCP coordinates are assigned to it and the TCP coordinate system becomes a child of a phone/tablet AR camera coordinate system. This part of the program code is executed every time when the AR/VR-WT key is pressed again. After that, still holding the AR/VR-WT button, Temp GameObject has the desired TCP coordinates in space at all times, so they are simply transferred to the TCP. In a similar way, the TCP robot moves in VR.

Figure 14 describes the basic implemented functions of the L-IRL XR application that allow programming in the AR/VR environment. The AddCoords() function allows the user to visually set a new frame (pose) in the robot workspace. Such poses are used to generate the trajectory. Programmed TCP poses in Cartesian space can be redefined by the user using the EditCoords() function, which is also used for unattainable trajectories. The LoadCoords(), SaveCoords(), and RemoveCoords() functions allow the user to load poses from a .txt file, save poses to a .txt file, or completely erase them from the robot's workspace.

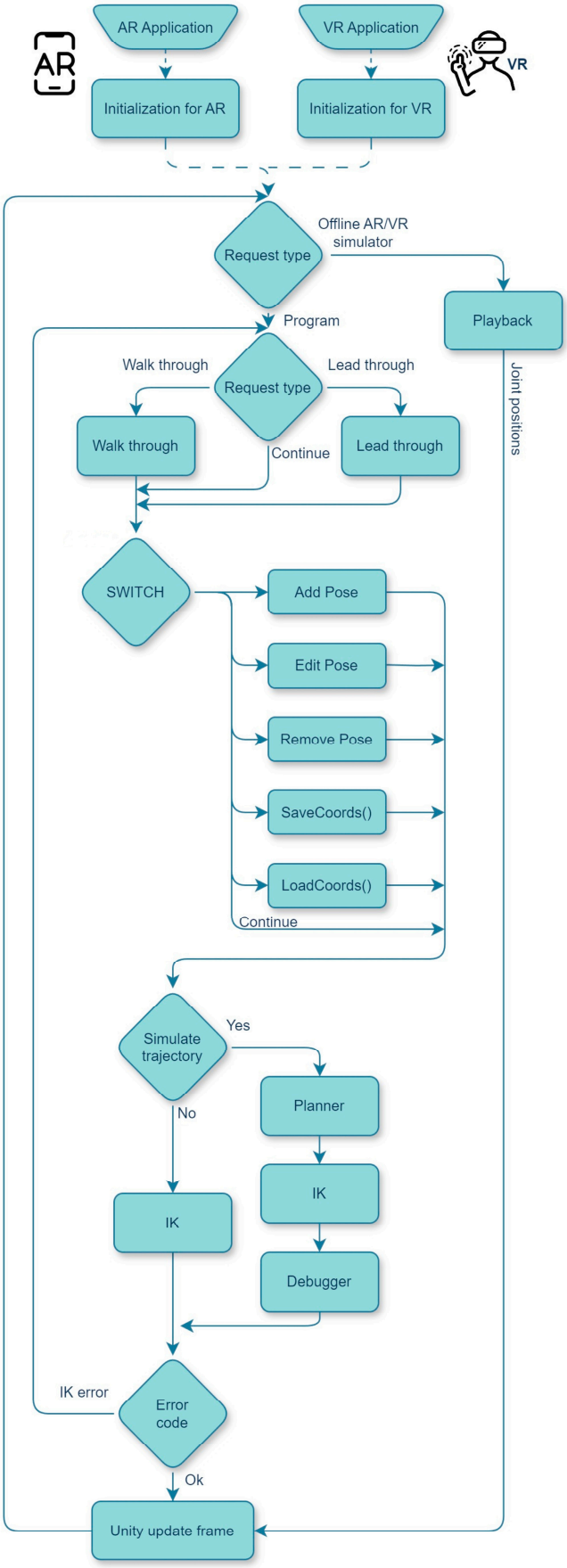


Figure 12. General system flow chart of the proposed cross-platform system.

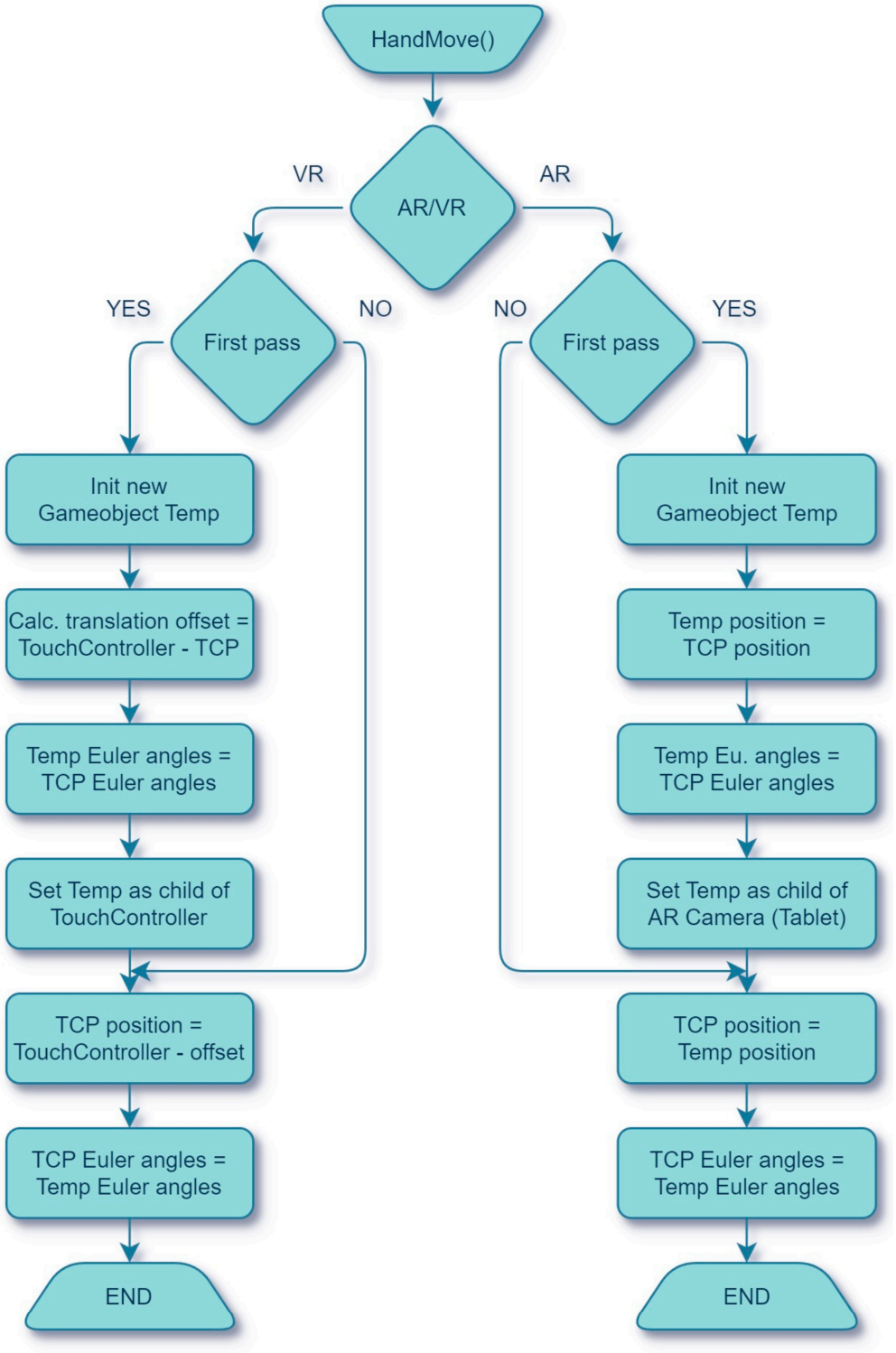


Figure 13. Flow chart of developed AR/VR-WT.

4.3. Post-Processor

A simple post-processor is developed for the exporting of programmed paths as offline instruction code; these are readable by the robot controller. Post-processing outputs are demonstrated in the video within the Supplementary Materials.

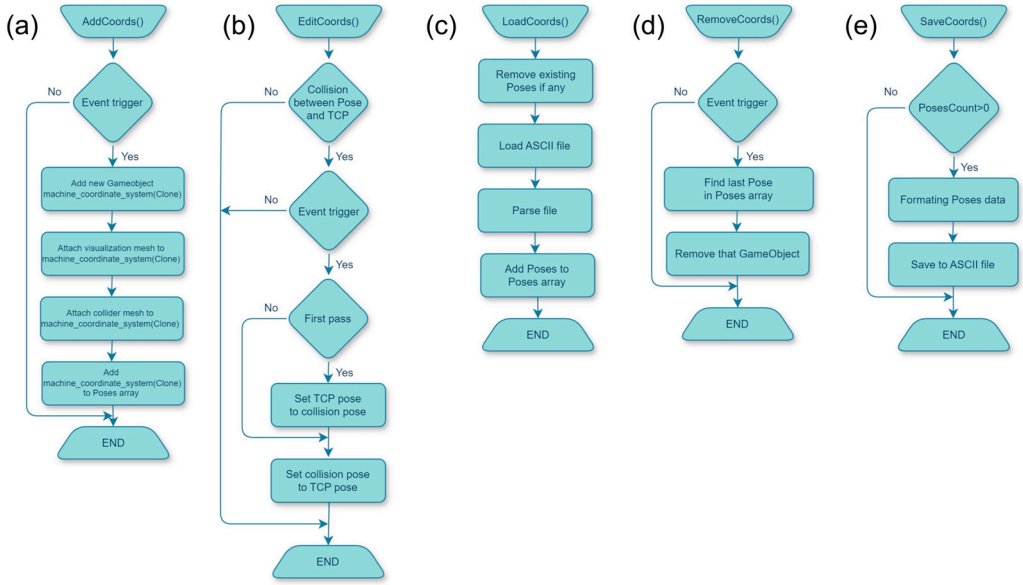


Figure 14. Flow charts of the other developed AR/VR functionalities: (**a**) AddCoords; (**b**) EditCoords; (**c**) LoadCoords; (**d**) RemoveCoords; (**e**) SaveCoords.

5. Discussion

For robotic applications, some of the prime features that need to be considered when selecting robot modeling, control, and simulation environments are simplicity, reliability, safety, and precision. Low cost and customizability are important as well. XR technology offers many advantages in enhancing human–robot interaction, which can be valuable for both novices and experts.

In this research, a standalone XR-supported robot programming framework for industrial robots is proposed. The absence of communication between the XR-experiencing device and the ROS-enabled computer contributes to minimizing latency and computational overheads in the system. Consequently, it could be argued that the proposed solution enables smooth running on less-expensive devices.

From a developer's point of view, the absence of the need to develop system interaction between Unity and robotics development frameworks/environments (APIs) is advantageous. Achieving real-time synchronization between robotics frameworks such as ROS and Unity can be challenging, particularly when dealing with robotics applications that require precise timing and coordination. Delays or synchronization issues may impact the accuracy of simulations. Both ROS and Unity are complex systems with their own learning curves. Also, ROS and other robot frameworks evolve over time with new releases, and compatibility with different versions has to be ensured.

One of the shortcomings of the Unity-based standalone robot programming software at the moment is the simulation of motion controllers, as limited motion controller design is available compared to the ROS–Unity suite. Even though real robot motor controller settings are usually not available to users, it could be useful for research to simulate advanced motion control laws in an XR environment. Still, XR enhancement in robot programming is mostly used for superior visualization benefits.

The standalone feature enables better mobility of the end-user, as it is not necessary to connect devices on the same network, contrary to the solutions integrating ROS [1,8,11–14] and RoboDK [16].

The solution presented in [17] has a standalone mode in which the programming of a real—not augmented—robot is performed (which causes robot downtime). The solution presented in [2] is a standalone solution with an integrated kinematic and dynamics model offering pick-and-place (PTP) robot programming. It uses additional external libraries for robot dynamics and kinematics, as well as the plotting library. Compared to the solution

presented in [2], which is developed in C/C++, the system presented herein is simpler to develop, as it is based on Unity and its supporting libraries.

The positive impact of XR-based HMIs in robot systems has been proposed as a solution for improving the time and cost efficiency of the robot programming process [1,2,4], which could positively influence the percentage of robot employment in SMEs. One important aspect of the proposed system's applicability, particularly for SMEs, is the cost of integrating the novel XR-assisted system into the production line. Software development costs generally include licensing fees and development expenses. While Unity is mostly free for developers and entirely free for end-users, ROS is also free, and RoboDK is a commercial software requiring end-user licenses as it must run on an additional computer. As the proposed system is standalone, development costs could be higher compared to ROS–Unity suite-based solutions [1,8,11–14], but not significantly, whereas hardware costs are lower. Implementation, training, maintenance, and operational costs depend on the specific solution's design and implementation details.

The ROS dependency of the ROS–Unity suite-based solutions in [1,8,11–14] makes cross-platform compatibility difficult since ROS is primarily targeted to certain Linux distributions. There is an ROS version for Windows, but it is not as well documented and supported as the Linux versions. The application presented in this article is standalone and exclusively Unity-based, which inherently facilitates cross-platform implementation. The solution proposed in this article is an AR cross-platform and AR/VR cross-adaptable solution. Cross-platform and cross-platform adaptability features offer a more adaptable experience for both developers and end-users.

Regarding motion programming functionalities, this research proposes two novel XR-robot programming concepts: AR/VR Linear-Path Lead-Through and AR/VR Linear-Path Walk-Through. AR/VR-LT and AR/VR-WT represent the integration of classical Lead-Through and Walk-Through online programming methods with offline continuous-path programming [43], while inheriting the simplicity of online Lead-Through [1] methods and the simplicity/intuitiveness of the online Walk-Through method [1,4], applying them to the domain of offline programming for continuous robot paths. One of the significant advantages of programming in an AR environment is that the robotic environment does not have to be modeled, including new robot manipulation objects or working surfaces.

All the presented functionalities of the XR-supported robot programming system are based on robot models with integrated dynamics, including motor capabilities with a novel method for controller set-up in Unity, which is one of the contributions of this paper.

The proposed open-source framework for XR-enhanced robot programming could be used as a complementary tool for commercial software, offering the advanced visualization capabilities of XR technology for trajectory verification and debugging purposes.

6. Conclusions

In this study, the design and implementation of a standalone XR-enhanced interactive industrial robot programming system using the Unity game engine is presented. The proposed open-source framework aims to deliver a cross-platform solution to minimize the effort and time invested by human–robot programmers, as well as system developers, and provide novel programming concepts leveraging the advantages of online robot programming methods to offline robot programming using augmented and virtual reality. The here-proposed development methodology includes a procedure for creating a robot model in Unity using CAD modeling; additionally, it includes the implementation of a solution to the inverse kinematics problem, appropriate frame transformations, and a method for motion controller design in the robot's joint space, which avoids the oscillatory response in robot joint motion in Unity and takes into account the installed actuators' capabilities. The robot programming system provides an XR-assisted simulation tool and an XR-assisted debugging tool for the enhanced visual trajectory verification and debugging process. Verification of the proposed modeling, control, and programming concepts, procedures, and algorithms is performed using a 6DoF robot. The methodology demonstrates a de-

tailed and structured approach to developing a standalone, cross-platform, interactive, XR-supported industrial robot programming system; it leverages the strengths of Unity and XR technologies to enhance user experience and efficiency in robot programming.

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/machines12070480/s1, Video S1: L-IRL XR I video.

Author Contributions: Conceptualization, A.D. and J.V.; methodology, A.D. and J.V.; software, A.D.; validation, A.D., J.V. and N.Z.; formal analysis, J.V. and N.Z.; investigation, A.D., J.V. and N.Z.; resources, A.D., J.V. and N.Z.; data curation, A.D. and J.V.; writing—original draft preparation, A.D. and J.V.; writing—review and editing, N.Z.; visualization, A.D., J.V. and N.Z.; supervision, J.V.; project administration, J.V.; funding acquisition, A.D., J.V. and N.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Serbian Ministry of Science, Technological Development and Innovations, grant number 451-03-66/2024-03/ 200066. The APC was funded by Lola Institute Belgrade.

Data Availability Statement: Data is contained within the article and Supplementary Material.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Appendix A. Procedure for Creating the 3D GameObject Robot in Unity

Table A1. Procedure for creating the robot GameObject in Unity using SolidWorks and URDF.

SolidWorks	Unity
1. Design of robot 3D model, creation of $n + 1$ robot links. 2. Local frames definition of robot links: the assigned frames are positioned in the joints, the frame orientation coincides with the orientation of the WORLD (global) frame from the CAD scene. 3. Export of the CAD robot model into URDF file format using the URDF exporter [49]. Selecting the option of manual assignment of local frames to links and assigning frames that coincide local frames in the SolidWorks model. 4. Export of the CAD robot model into glTF file format using glTFExporter add-on [50].	5. Import of the CAD robot model into Unity using URDF importer add-on [58]. 6. Import of the CAD robot model into Unity using glTF importer add-on. 7. Redefine Unity URDF model textures. Replace URDF textures with glTF textures. Remove the imported glTF robot model.

In order to avoid complex calculations, it is of crucial importance to set the relevant frames in a correct manner in different development environments that need to be integrated, i.e., CAD software and Unity [31] (v2022.3.6f1). In Step 2, the frames' orientations are chosen to identify potential errors of the robot kinematics algorithm within integration with Unity. The URDF importer is used to automatically create an Articulation Body component for every robot link, which are intended to simulate robot dynamics. URDF exporter does not export the texture (color) of the robot shell, which can be useful for robot visualizing and for observing situations where the robot potentially passes through a singular point if the robot link is symmetrical. Textures from the imported URDF file are replaced with textures from imported glTF model.

Appendix B. Technical Characteristics of 6DoF Robot RL15

The masses of links obtained from the URDF file based on the robot CAD model, gear ratios, and maximum torques, all obtained from robot documentation, are given in Table A2. RL15 axes' movement limits are given in Table A3.

Table A2. Robot RL15 model parameters.

Joint k	Mass [kg]	Max. Motor Torque [Nm]	Gear Ratio	Max. Joint Torque [Nm]	Motor and Gearbox Inertia [kgm^2 10^{-3}]
1	347.95	17.8	120	2136	4.2
2	75.17	38.6	119	4593.4	9.0
3	82.34	17.8	119	2118.2	9.0
4	46.3	3.47	133.4	462.9	0.8
5	2.37	3.47	219.4	761.32	1
6	0.9	3.47	53.3	184.95	0.7

Table A3. Robot RL15 axes movement limits [7,8].

Joint k	Axes Limits [°]	Velocity Limits [r/min]
1	[−150, 150]	3500
2	[−120, 80]	2800
3	[−73, 73]	3500
4	[−250, 250]	3400
5	[−110, 110]	3400
6	[−200, 200]	3400

Appendix C. Trajectory Generation

In order to verify the concept of the proposed AR/VR -LT, AR/VR -WT, and AR/VR -D tools in Section 4, a simple trajectory planner was developed in this study and implemented as a C# script in Unity. In this research, the trajectory generation is conceptualized in a way that, based on end-user input (selected poses of TCP-attached frames in AR/VR environment), a linear path of the TCP between two subsequent positions in Cartesian space is executed.

In order to avoid sudden jumps in the required motor torque between subsequent linear path segments in Cartesian space, a smoothing of the linear changes in path segment projections in x, y, and z axes directions and smoothing of the linear changes of Euler angles is performed using a cubic polynomial. For illustration, a general coordinate p is considered here. In Equation (A1), $\tau = t/\ \mathrm{T_F}$ is normalized time. Also, normalized coordinate $\pi = p/\mathrm{P_F}$ has been introduced, where $\mathrm{P_F}$ is the finite value of the coordinate p change set by user.

$$\pi(\tau) = a_3\tau^3 + a_2\tau^2 + a_1\tau^1 + a_0. \tag{A1}$$

To obtain polynomial coefficients $a_{0..3}$ in $\pi(\tau)$, four initial and final values in Equation (A1) have to be set, and herein they are selected as follows:

$$\pi(0) = 0;\ \pi(1) = 1;\ \dot{\pi}(0) = 0;\ \dot{\pi}(1) = 0, \tag{A2}$$

From Equations (A1) and (A2), the following is obtained:

$$\pi(\tau) = -2\tau^3 + 3\tau^2 \tag{A3}$$

In Figure A1, the change of position, speed, and acceleration for the normalized general coordinate π has been presented.

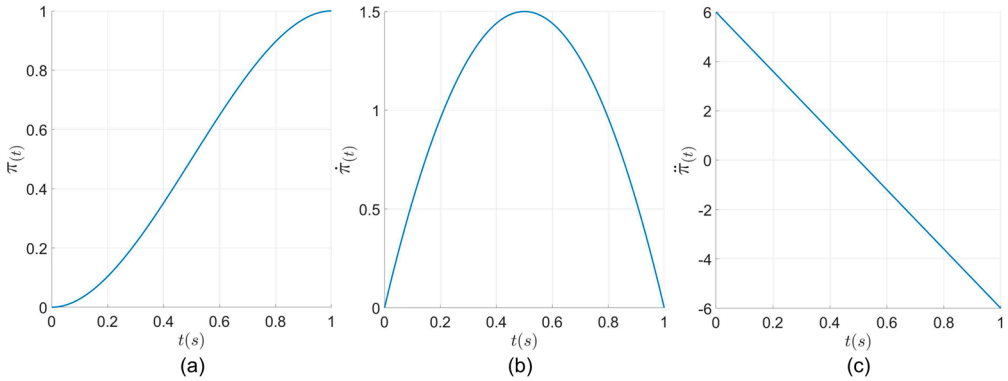


Figure A1. Change of (**a**) position, (**b**) speed, and (**c**) acceleration for the normalized coordinate π.

Given that the Unity frame rate is variable and not predictable, in this research, the value of the TCP's Cartesian coordinates and Euler angles in real-time is calculated from the analytical expression Equation (A3) for the time instance corresponding to the current frame rate. The speed is specified implicitly by the end-user via the TIME parameter (T_F). For every rendering frame, IK has been executed, including the checks on position and speed limitations for every joint/motor. The information flow in the designed trajectory planner is summarized in Table A4.

Table A4. TP algorithm. Input is a user-defined programmed pose of the TCP.

Step	Step Description
1.	Definition of (smoothed) linear path between starting and ending TCP poses. Time change from starting to ending Euler angles is selected to be linear.
2.	Calculation of current poses for current time instance τ using Equation (A3).
3.	IK algorithm execution for the current TCP pose.
4.	Check whether joint positions q_j, $j = 1, 2, \ldots, n$ (n is the number of DoFs) are within their limits $q_{j\max}$, $j = 1, 2, \ldots, n$ for the defined workspace. If not return error (joint position q_j out of limit).
5.	Calculation of joint speeds $\dot{q}_j = (q_j - q_{j\text{previous}})/\Delta t$, $j = 1, 2, \ldots, n$.
6.	Check whether joint speeds $\dot{q}_j$, $j = 1, 2, \ldots, n$ are within their limits $\dot{q}_{j\max}$, $j = 1, 2, \ldots, n$. If not return error (joint speed $\dot{q}_j$ out of limit)

References

1. Ong, S.K.; Yew, A.W.W.; Thanigaivel, N.K.; Nee, A.Y.C. Augmented Reality-Assisted Robot Programming System for Industrial Applications. *Robot. Comput.-Integr. Manuf.* **2020**, *61*, 101820. [CrossRef]
2. Fang, H.C.; Ong, S.K.; Nee, A.Y.C. Interactive Robot Trajectory Planning and Simulation Using Augmented Reality. *Robot. Comput.-Integr. Manuf.* **2012**, *28*, 227–237. [CrossRef]
3. Brogårdh, T. Present and Future Robot Control Development-An Industrial Perspective. *Annu. Rev. Control* **2007**, *31*, 69–79. [CrossRef]
4. Gaschler, A.; Springer, M.; Rickert, M.; Knoll, A. Intuitive Robot Tasks with Augmented Reality and Virtual Obstacles. In Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China, 31 May–7 June 2014; pp. 6026–6031.
5. Berger, S.; Armstrong, B. The Puzzle of the Missing Robots. *MIT Case Stud. Soc. Ethical Responsib. Comput.* **2022**. Available online: https://mit-serc.pubpub.org/pub/puzzle-of-missin (accessed on 11 July 2023). [CrossRef]
6. Perzylo, A.; Rickert, M.; Kahl, B.; Somani, N.; Lehmann, C.; Kuss, A.; Profanter, S.; Beck, A.B.; Haage, M.; Rath Hansen, M.; et al. SMErobotics: Smart Robots for Flexible Manufacturing. *IEEE Robot. Autom. Mag.* **2019**, *26*, 78–90. [CrossRef]
7. Collett, T.H.J.; MacDonald, B.A. An augmented reality debugging system for mobile robot software engineers. *J. Softw. Eng. Robot.* **2010**, *1*, 18–32.
8. Lotsaris, K.; Gkournelos, C.; Fousekis, N.; Kousi, N.; Makris, S. AR Based Robot Programming Using Teaching by Demonstration Techniques. *Procedia CIRP* **2021**, *97*, 459–463. [CrossRef]
9. Coronado, E.; Itadera, S.; Ramirez-Alpizar, I.G. Integrating Virtual, Mixed, and Augmented Reality to Human–Robot Interaction Applications Using Game Engines: A Brief Review of Accessible Software Tools and Frameworks. *Appl. Sci.* **2023**, *13*, 1292. [CrossRef]

10. Ikeda, B.; Szafir, D. An AR Debugging Tool for Robotics Programmers. In Proceedings of the 4th International Workshop on Virtual, Augmented, and Mixed Reality for HRI, Boulder, CO, USA, 8–11 March 2021.
11. Ostanin, M.; Mikhel, S.; Evlampiev, A.; Skvortsova, V.; Klimchik, A. Human-Robot Interaction for Robotic Manipulator Programming in Mixed Reality. In Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 31 May–31 August 2020; pp. 2805–2811.
12. Hernandez, J.D.; Sobti, S.; Sciola, A.; Moll, M.; Kavraki, L.E. Increasing Robot Autonomy via Motion Planning and an Augmented Reality Interface. *IEEE Robot. Autom. Lett.* **2020**, *5*, 1017–1023. [CrossRef]
13. Gadre, S.Y.; Rosen, E.; Chien, G.; Phillips, E.; Tellex, S.; Konidaris, G. End-User Robot Programming Using Mixed Reality. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 2707–2713.
14. Matour, M.-E.; Winkler, A. Intuitive Robot Path Planning through Augmented Reality. In Proceedings of the 2023 27th International Conference on Methods and Models in Automation and Robotics (MMAR), Miedzyzdroje, Poland, 22–25 August 2023; pp. 27–32.
15. Chacko, S.M.; Granado, A.; Kapila, V. An Augmented Reality Framework for Robotic Tool-Path Teaching. *Procedia CIRP* **2020**, *93*, 1218–1223. [CrossRef]
16. Vidakovic, J.; Devic, A.; Lazarevic, I.; Zivkovic, N. Design of Augmented Reality-Based Android App for Simulation and Programming of Industrial Robots. In Proceedings of the International Conference of Experimental and Numerical Investigations and New Technologies CNNTech 2023, Zlatibor, Serbia, 4–6 July 2023; Springer Nature: Cham, Switzerland, 2023; pp. 239–245.
17. Kapinus, M.; Materna, Z.; Bambušek, D.; Beran, V.; Smrž, P. ARCOR2: Framework for Collaborative End-User Management of Industrial Robotic Workplaces using Augmented Reality. *JINT*, 2023; *submitted*. [CrossRef]
18. Kvrgic, V. Development of Intelligent System for Control and Programming of Industrial Robots. Ph.D. Dissertation, Faculty of Mechanical Engineering, University of Belgrade, Belgrade, Serbia, 1998.
19. Kvrgic, V.; Vidakovic, J. Efficient Method for Robot Forward Dynamics Computation. *Mech. Mach. Theory* **2020**, *145*, 103680. [CrossRef]
20. Available online: https://github.com/andrejevica4/L-IRL-XR (accessed on 10 July 2023).
21. ABB Offers Augmented Reality on a Smartphone to Simplify Robot Installations. Available online: https://new.abb.com/news/detail/66541/ar-smartphone-robot-installations (accessed on 18 December 2023).
22. Szybicki, D.; Kurc, K.; Gierlak, P.; Burghardt, A.; Muszyńska, M.; Uliasz, M. Application of virtual reality in designing and programming of robotic stations. In *IFIP Advances in Information and Communication Technology, Collaborative Networks and Digital Transformation, Proceedings of the 20th IFIP WG 5.5 Working Conference on Virtual Enterprises PRO-VE 2019, Turin, Italy, 23–25 September 2019*; Camarinha-Matos, L.M., Afsarmanesh, H., Antonelli, D., Eds.; Springer International Publishing: Berlin/Heidelberg, Germany, 2019; pp. 585–593.
23. Fuste, A.; Reynolds, B.; Hobin, J.; Heun, V. Kinetic AR: A Framework for Robotic Motion Systems in Spatial Computing. In Proceedings of the Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems, Association for Computing Machinery, New York, NY, USA, 25 April 2020; pp. 1–8.
24. Yang, W.; Xiao, Q.; Zhang, Y. HA R 2 bot: A Human-Centered Augmented Reality Robot Programming Method with the Awareness of Cognitive Load. *J. Intell. Manuf.* **2023**, *35*, 1985–2003. [CrossRef] [PubMed]
25. Magnenat, S.; Ben-Ari, M.; Klinger, S.; Sumner, R.W. Enhancing Robot Programming with Visual Feedback and Augmented Reality. In Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education, Vilnius, Lithuania, 4–8 June 22 2015; ACM: New York, NY, USA; pp. 153–158.
26. Togias, T.; Gkournelos, C.; Angelakis, P.; Michalos, G.; Makris, S. Virtual Reality Environment for Industrial Robot Control and Path Design. *Procedia CIRP* **2021**, *100*, 133–138. [CrossRef]
27. Oliveira, D.M.; Cao, S.C.; Hermida, X.F.; Rodriguez, F.M. Virtual Reality System for Industrial Training. In Proceedings of the 2007 IEEE International Symposium on Industrial Electronics, Vigo, Spain, 4–7 June 2007; pp. 1715–1720.
28. Karagiannis, P.; Togias, T.; Michalos, G.; Makris, S. Operators Training Using Simulation And VR Technology. *Procedia CIRP* **2021**, *96*, 290–294. [CrossRef]
29. Garratt, W.; Rithviik, S.; Wang, F. Achieving Interoperability Between Gaming Engines by Utilizing Open Simulation Standards. In Proceedings of the 2023 Simulation Innovation Workshop (SIW), Orlando, FL, USA, 13–17 February 2023; Simulation Interoperability Standards Organization-SISO: St. Petersburg, FL, USA; pp. 1–14.
30. Hiorns, B. Why Defence Simulation Systems Need Interoperability. Available online: https://www.novatech.co.uk/blog/simulation-interoperability (accessed on 11 October 2023).
31. Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine. v2022.3.6f1. Available online: https://unity.com (accessed on 18 December 2023).
32. Unreal Engine | The Most Powerful Real-Time 3D Creation Tool. Available online: https://www.unrealengine.com/en-US/ (accessed on 18 December 2023).
33. Todorov, E.; Erez, T.; Tassa, Y. MuJoCo: A Physics Engine for Model-Based Control. In Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura-Algarve, Portugal, 7–12 October 2012; pp. 5026–5033.
34. DART: Dynamic Animation and Robotics Toolkit. Available online: https://dartsim.github.io/ (accessed on 16 November 2023).
35. SimTK: Simbody: Multibody Physics API: Project Home. Available online: https://simtk.org/projects/simbody/ (accessed on 16 November 2023).

36. Gazebo. Available online: https://gazebosim.org/home (accessed on 16 November 2023).
37. Robot Simulator CoppeliaSim: Create, Compose, Simulate, Any Robot-Coppelia Robotics. Available online: https://www.coppeliarobotics.com/ (accessed on 16 November 2023).
38. Cyberbotics: Robotics Simulation with Webots. Available online: https://cyberbotics.com/ (accessed on 16 November 2023).
39. Gazebo_ros_pkgs. Available online: https://github.com/ros-simulation/gazebo_ros_pkgs (accessed on 11 January 2024).
40. Meng, W.; Hu, Y.; Lin, J.; Lin, F.; Teo, R. ROS+unity: An Efficient High-Fidelity 3D Multi-UAV Navigation and Control Simulator in GPS-Denied Environments. In Proceedings of the IECON 2015-41st Annual Conference of the IEEE Industrial Electronics Society, Yokohama, Japan, 9–12 November 2015; pp. 002562–002567.
41. Getting Started | Vuforia Library. Available online: https://developer.vuforia.com/library/ (accessed on 18 December 2023).
42. Marlenaklein-msft Mixed Reality Toolkit 3 Developer Documentation-MRTK3. Available online: https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk3-overview/ (accessed on 18 December 2023).
43. Chettibi, T. Smooth point-to-point trajectory planning for robot manipulators by using radial basis functions. *Robotica* **2019**, *37*, 539–559. [CrossRef]
44. ROS: Home. ROS2 v.humble. Available online: https://www.ros.org/ (accessed on 11 January 2024).
45. Simulator for Industrial Robots and Offline Programming-RoboDK (v4.0.0.). Available online: https://robodk.com/ (accessed on 18 December 2023).
46. Tola, D.; Corke, P. Understanding URDF: A Survey Based on User Experience 2023. In Proceedings of the 2023 IEEE 19th International Conference on Automation Science and Engineering (CASE), Auckland, New Zealand, 26–30 August 2023; pp. 1–7.
47. glTF-Runtime 3D Asset Delivery. Available online: https://www.khronos.org/gltf/ (accessed on 18 December 2023).
48. 3D CAD Design Software | SOLIDWORKS. Available online: https://www.solidworks.com/home-page-2021 (accessed on 18 December 2023).
49. Sw_urdf_exporter-ROS Wiki. Available online: https://wiki.ros.org/sw_urdf_exporter (accessed on 18 December 2023).
50. GLTFExporter–Three.Js Docs. Available online: https://threejs.org/docs/#examples/en/exporters/GLTFExporter (accessed on 18 December 2023).
51. Articulations—NVIDIA PhysX SDK 3.3.4 Documentation. Available online: https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/3.3.4/Manual/Articulations.html?highlight=articulation (accessed on 18 December 2023).
52. Oculus SDK. Available online: https://developer.oculus.com/downloads/package/meta-xr-sdk-all-in-one-upm/ (accessed on 5 March 2024).
53. Technologies, U. Unity-Scripting API: Transform.eulerAngles. Available online: https://docs.unity3d.com/ScriptReference/Transform-eulerAngles.html (accessed on 18 December 2023).
54. Available online: https://docs.unity3d.com/Manual/QuaternionAndEulerRotationsInUnity.html#:~:text=Euler%20angle%20rotations%20perform%20three,change%20while%20the%20rotations%20occur (accessed on 11 July 2024).
55. Featherstone, R. The Calculation of Robot Dynamics Using Articulated-Body Inertias. *Int. J. Robot. Res.* **1983**, *2*, 13–30. [CrossRef]
56. Biggs, G.; MacDonald, B. A survey of robot programming systems. In Proceedings of the Australasian Conference on Robotics and Automation, Brisbane, Australia, 4–6 December 2003; Volume 1, pp. 1–3.
57. Heimann, O.; Guhl, J. Industrial Robot Programming Methods: A Scoping Review. In Proceedings of the 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, 8–11 September 2020; pp. 696–703. [CrossRef]
58. URDF-Importer. Available online: https://github.com/Unity-Technologies/URDF-Importer/tree/main (accessed on 12 January 2024).

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.