VR and Depth Camera based Human-Robot Collision Predictor System with 3-Finger Gripper Assisted Assembly Device

Imre Paniti
SZTAKI, Centre of Excellence in Production Informatics and Control;
Budapest, Hungary
Széchenyi István Egyetem,
Győr, Hungary
imre.paniti@sztaki.hu

János Nacsa
SZTAKI, Centre of Excellence in Production Informatics and Control;
Budapest, Hungary
Széchenyi István Egyetem,
Győr, Hungary
nacsa.janos@sztaki.hu

Péter Kovács
SZTAKI, Centre of Excellence in Production Informatics and Control
Budapest, Hungary
peter.kovacs@sztaki.hu

Dávid Szűr
SZTAKI, Centre of Excellence in Production Informatics and Control
Budapest, Hungary
david.szur@sztaki.hu

Abstract*—It is known that Human-Robot Collaboration (HRC) performance is improved in some assembly tasks when a robot emulates the effective coordination behaviors observed in human teams, but a close interaction could cause collisions which should be avoided. There are several methods that can be used to communicate the intension of the robot. However, these are mainly acoustic or visual signals. In this paper a Virtual Reality and Depth Camera based System is presented where vibration signals are used to alert the user of a probable collision with a 2-Finger gripper equipped Robot. Experimental tests are investigated in an assembly task with - another 3-Finger - gripper which functions as a Flexible Assembly Device.

***Keywords*—Collaborative Robot, Human-Robot Collaboration, Virtual Reality, Collision Prediction**

I. Introduction

According to the Digital Economy and Society Index Report 2019 by the European Commission [1], the share of large enterprises that use industrial or service robots is four times higher than the share of SMEs and the use of robots varies strongly according to company size.

One of the recurring questions in a semi-robotized industry is: how to make a production more efficient?

Based on a study in [2] a robot could reduce the human's idle time by 85% in an assembly operation. So applying collaborative robots (cobots) in a factory for assembly tasks could lead to a more efficient work. This statement should also be valid for the assembly of varying products or product families, which require a set of different fixtures, reconfigurable fixtures like the Parallel Kinematic Machine-based ones in [3] or a fixed, but flexible gripper presented in this article.

However, the problem is that despite well described task sequences the changeover from one product to another in a collaborative operation could lead to human failures and consequently to collisions with the cobot.

By definition a cobot has to operate with safety installations (protective stop execution by reaching a certain force in a collision), according to ISO/TS 15066:2016, ISO 10218-1 and ISO 10218-2, but these protective stops could cause a significant cumulative delay in the production.

Review articles like the work of Hentout et al. [4] and Zacharaki et al. [5] presenting solutions for pre-collision approaches in the frame of Human-Robot Interaction (HRI). Both surveys mention the work of Carlos Morato et al. [6] who created a framework with multiple Kinects to generate a 3D model with bounding spheres of human's movements in real-time. The proposed framework calculates human-robot interference in a 3D space with a physics-based simulation engine, but the pre-collision strategy for safe HRC is manifested in the complete stop of the robot. This is indeed a safe protocol, reduces the production break time, but does not eliminate it fully.

The aim of this paper is to underline the importance of a new pre-collision strategy, especially when flexible/reconfigurable fixtures are used.

II. Experimental environment

Most often, robots are moved on predefined trajectories that are fixed in the robot's program and a new task typically involves starting a new robot program. Another way is to move the high level robot control out of the robot into a computer and the robot gets continuously via a stream the required moving and other actions. In this case, the robot runs a general-purpose program or framework that interprets and executes the instructions received from outside. In this scenario this framework is called URSZTAKI and was developed earlier in the Institute. URSZTAKI has three kinds of instructions: (a) basic instructions of the robot's programming language, (b) instructions of the robot add-ons (e.g. gripper, force sensor) integrated into the robot language by the vendors of the accessories, and (c) frequently used more complex task instructions (e.g. putting down an object when it is unknown how far the table is). The third type of instructions are the real features of URSZTAKI.

It should be mentioned that the expansion of the UR robot's functions and language is possible with the help of so-called URCAPs and currently URSZTAKI can also be installed as an URCAP (which is a platform where users, distributors and integrators can present accessories that run successfully in UR robot applications [7]).

The experimental setup is a UR10 robot with a force sensor and a 2-Finger gripper. The environment was designed to support different assembly tasks either fully robotized or collaborative. To mount partly or even fully different components universal mounting technology was required instead of special fixtures. Another gripper is used - a 3-Finger one - that allows a wide variety of fixings. All three fingers can be moved independently of the selected adaptive gripper that is fixed on the robot work table.

The 3-Finger gripper from RobotiQ [8] has 4 different modes to its fingers (see Fig. 1).

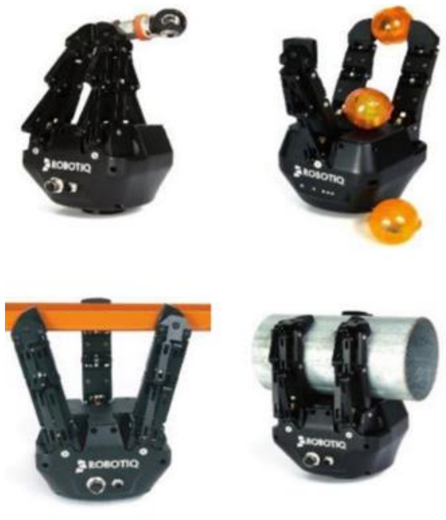
Fig. 1. Four different modes of the 3-Finger RobotiQ gripper [8].

In the “pinch” mode on the top left side of Fig. 1, the gripper acts as a 2-Finger one because it fingers “B” and “C” on the same side move close together. The next mode is “Scissor” when exactly this closing-opening ability is used on picking up an object. In the third “wide” mode, the “B” and “C” fingers are fan-like and they provide a wide grip for longer objects. In the case of the leftmost “normal” grip, the three fingers are moving in parallel and, depending on the relative position of the object, fingertips also turn on it.

From the software point of view both grippers can be directly programmed from the robot's program code. Because both grippers are from the same manufacturer, the commands of one of the grippers had to be modified to avoid conflict between the individual instructions.

A typical scenario is that the robotic arm is lifting a part to the fixed gripper which grabs it, and after that another part is placed or pressed by the robotic arm on the part fixed by the immobile gripper. There are some tasks, like the insertion of a spring in a housing, which have to be done by the human operator (see Fig. 2).

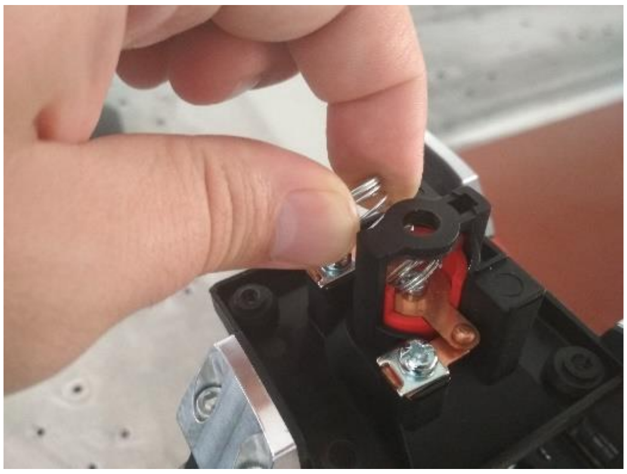
Fig. 2. Illustration of a manual spring insertion.

In the environment it is also possible that the robot holds a screwdriver and fastens the assembled parts with screws (see Fig. 3 and Fig. 4).

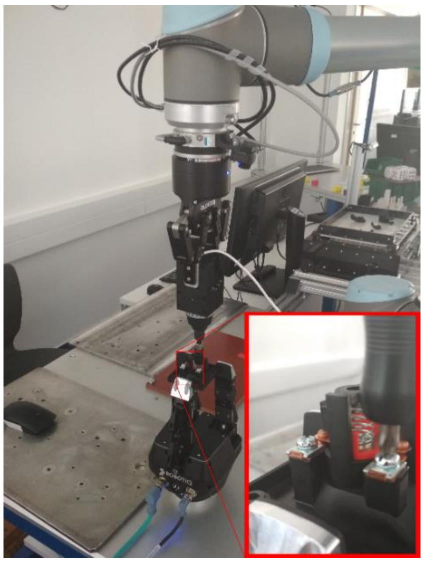
Fig. 3. Illustration of the robotised screwdriving of a Push-Button element.

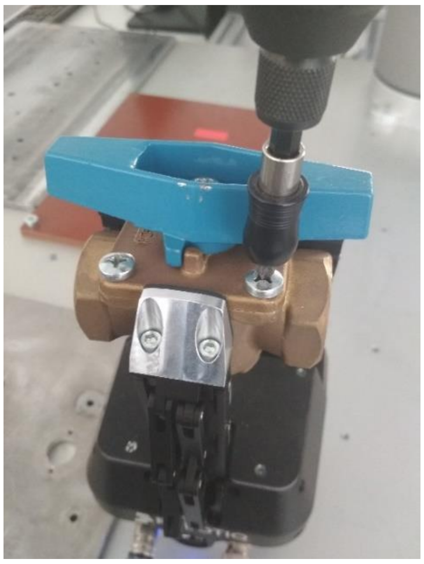
Fig. 4. Illustration of the robotised screwdriving of a Ball Valve element.

The proposed solution with the immobile 3-Finger gripper satisfies the requirements of a flexible fixture for certain parts. Human-robot collision problems might occur in this case when the human operator forgets the predefined assembly task sequence for a new product, grasps for an assembly part and the trajectory of his hand intersects the trajectory of the robot.

III. PRE-COLLISION APPROACH AS A PREDICTOR

In order to avoid collisions with the robot, either the robot trajectory has to be modified in real-time (which might cause additional production time) or the human operator has to be warned with a pre-defined understandable signal so he/she can modify his/her movement in time. The warning signal could be a visual, an acoustical or a tactile signal. In this paper the latter has been developed as a part of a PREdictor of HUman-RObot COllision (PREHUROCO) framework. The subject of the prediction in this case are the predetermined movements of the robot which will occur after a certain time. So a similar framework had to be created as described in [6], but instead of a digital twin of the robot (real-time 3D visualization of the robot) a pre-played robot model motion was used.

A. Requirement analysis

The requirement analysis of the PREHUROCO system showed that the following features are desired on the candidate software library:

1) Fully open-source: The system must comply with all security requirements in a real manufacturing system, therefore full control over source code is mandatory.

2) Modular: The system should be divided into various software components so the candidate software library must support responsibility encapsulation.

3) Distributed: In a manufacturing system, many computers and IoT devices are connected together, therefore the PREHUROCO software components must have the ability to run on different computers or IoT devices.

4) Cross-platform: As a distributed requirement shows many computers and devices are connected together with various operating systems, therefore the candidate framework should be cross-platform.

5) Programming Language Variability: As distributed and cross-platform requirements showed the variability of the devices and computer operating system in manufacturing scenarios are high therefore the candidate software library should support different Application Programming Protocols (APIs).

6) Scalability: PREHUROCO software components should be developed independently of whether they run on the same computer or not. In terms of performance, the software components can be easily put together into one machine, into one application or can be distributed.

7) Rapid prototyping: The candidate framework should provide examples or even pre-made components that can be improved during the implementation of PEHUROCO because the proposed system should deal with:

- Rigid-body simulation.
- Visualization, even VR or AR.
- Real-time 3D scanning.
- X3D model format.
- Various communication protocols.

Unity Engine and Unreal Engine are well known cross-platform game engines. ApertusVR [9] is a software and hardware vendor-free open-source software library. It offers a no-vendor-lock-in approach for integrating Virtual Reality technologies into industrial software systems. The comparison of the candidate frameworks considering the requirements is summarized in Table 1.

TABLE I. COMPARISON OF DIFFERENT FRAMEWORKS CONSIDERONG THE REQUIREMENTS OF PREHUROCO

Requirement	Unity Engine	Unreal Engine	ApertusVR
Open source	Partially	Yes	Yes
Modular	Yes	Yes	Yes
Distributed	Partially	Corner Case	Yes
Cross-platform	Yes	Partially	Partially
Prog. Lang. Variability	Partially	Corner Case	Yes
Scalability	Partially	Partially	Yes
Rapid Prototyping	Yes	Yes	Yes

Based on the requirement analysis of PREHUROCO the ApertusVR software library was chosen for implementing the system. By the help of the ApertusVR software library, a distributed software ecosystem can be created via Intranet/Internet. Basically, it can be divided into two major parts, Core and Plugins. The Core system is responsible for the Internet/Intranet communication among the participants of the distributed software ecosystem and synchronizes the information between them during the session. The plugin mechanism makes the possibility to extend the capability of any solution which is created by the ApertusVR library. Plugins can access and manipulate the information within the core system.

B. Explanation of the PREHUROCO system

The system is distributed to five major responsibilities:

1) 3D scanning of the human operator.
2) Streaming the joint angles of the robot.
3) Collision Detection between the human and the robot.
4) Alert human for the possible collision.
5) Visualize the whole scenario.

These responsibilities were implemented by the help of the ApertusVR library and each was encapsulated into six plugins [10]: 1: Collision Detection Plugin, 2: Visualization Plugin, 3: Kinect Plugin, 4: WebSocket Server Plugin, 5: X3D Loader Plugin, 6: NodeJS Plugin

The 7th element is a WebSocket Client, which is implemented in the form of a HTML site with jQuery JavaScript library and Vibration API call [11] for mobile phones, but for more convenient use the WebSocket Client could run also on a Smart Watch.

Fig. 5 shows the realized system with the connections and applied protocols in an experimental set-up with an UR5 robot.

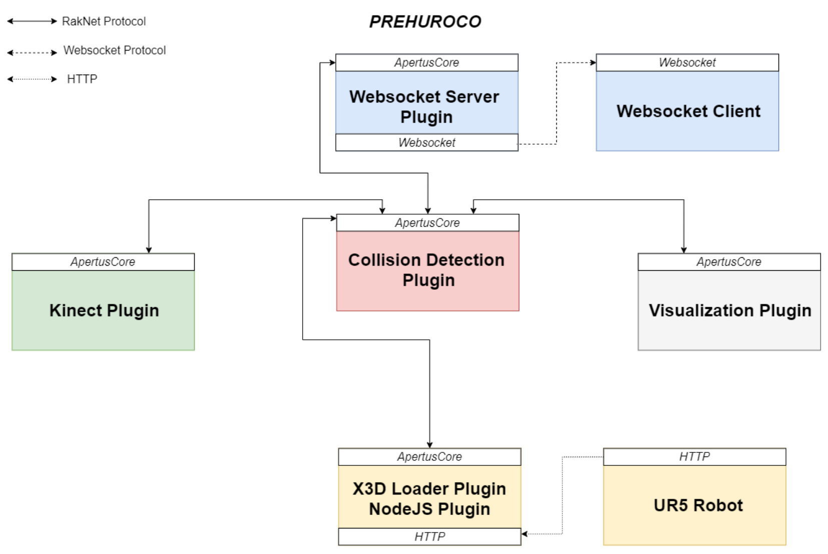

Fig. 5. System elements and connections of PREHUROCO with the applied protocols.

Collision Detection Plugin [12]: This Plugin was created based on the pre-made "bulletPhysics" Plugin of ApertusVR. Previously this plugin was able to run rigid body simulation but collision events were not raised during the simulation. The rigid body abstraction of ApertusVR was extended by the functionality of collision events.

Visualization Plugin [13]: This Plugin used as-is from the repository of ApertusVR for visualization purposes.

Kinect Plugin [14]: This Plugin was created based on the pre-made "Kinect" Plugin of ApertusVR. Previously this plugin was able to create the skeleton of the tracked human or even its point cloud but rigid bodies were not created. For collision detection, rigid bodies are mandatory therefore rigid bodies were created based on the geometries of the human skeletons.

WebSocket Server Plugin [15]: This Plugin was created based on the pre-made "webSocketServer" Plugin of ApertusVR. Previously this plugin was able to forward all events which are raised in the Core. For collision detection, only the collision event of the rigid bodies is necessary. During the implementation of that plugin, a filter feature was added to forward only the desired event into the WebSocket connection.

X3D Loader Plugin [16]: This Plugin was created based on the pre-made "X3DLoader" Plugin of ApertusVR. Previously this plugin was able to parse x3d format and create only the geometries of the robot. For the collision detection, rigid bodies are mandatory therefore rigid bodies were created based on the parsed geometries.

NodeJS Plugin [17]: This Plugin used as-is from the repository of ApertusVR. This plugin allows to run a web server for receiving the joint angle of the UR5 robot via HTTP requests.

In the PREHUROCO system, these Plugins are encapsulated into different applications. These different applications can be run on different computers to distribute the computational power and achieve real-time collision prediction. As the diagram shows these applications communicate over Internet/Intranet communication via different protocols.

The collision detection application has to be run on a High Performance Computing (HPC) server to process the virtual collisions in real-time.

The Kinect application can run on a dedicated computer of the Kinect device or on the same computer which calculates the virtual collisions.

The X3DLoader Plugin and the NodeJS Plugins are integrated into one application and can run in the dedicated computer of the UR5 Robot.

The WebSocket Server application can also be run on a different computer to ensure security and locality requirements.

The joint positions are stored in a jsonlist file which is generated by executing the whole robot program. During the execution the joint positions are “grabbed” and saved with a given frequency.

The speed of the simulation is equal to the speed of the robot movement and the "forecast" can be determined with the delay between the simulation starting time and the real robot execution start time.

IV. RESULTS

The proposed Framework has been tested on two local network topologies.

In case the calculations had been divided into a cloud service based computer (with 4 Virtual CPUs, 8GB RAM running a Windows 10 Operating System) and a HPC Server (Ideum with Intel i7-8700, RTX 2080 8GB GDDR6 NVIDIA graphics card, Dual 250GB NVMe M.2 SSD, 32GB 2400MHz DDR4 RAM running a Windows 10 Operating System) the collision events were delivered to the WebSocket Client with significant delay.

By running all ApertusVR Plugins on the Ideum and sending only the collision events via Wireless LAN Connection (2.4Ghz WiFi) the user experience was quasi real-time.

Fig. 6 shows a virtual collision test running on the Ideum (HPC Server) with the skeleton model of a single operator (1), virtual UR5 robot movement simulation (2), real robot (3), Kinect Sensor (4), and a Mobile Phone (5) with Android Operating System, running the WebSocket Client to vibrate the device. The 3D Scene is visualized with a top camera view, but arbitrary camera views are possible.

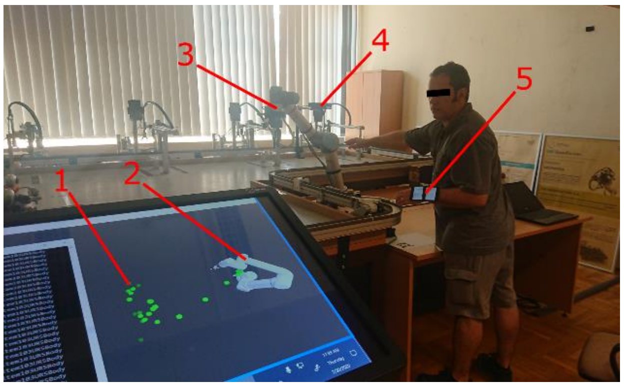

Fig. 6. Virtual collision test.

The Kinect Plugin creates a simplified skeleton model from the human operator which needs improvement. An anthropomorphic skeleton model or voxelization could be a solution in the future.

It has to be underlined that the communication time increased by the human reaction time should not exceed the ΔT time between the pre-played simulated motion and the actual motion of the robot. Jsonlist file of the simulated UR5 robot movement is provided in [18].

V. CONCLUSION

In this paper a commercially available gripper as a flexible fixture for assembly and a new pre-collision approach as a predictor for Human-Robot Collaboration were presented. The proposed Framework had been realized with the help of a modular, distributed, open-source cross-platform (ApertusVR) with different programming API support and scalability solution.

Seven interconnected system modules have been developed with the goal to monitor the movement of the human operator in 3D space, calculate collisions with a virtual robot, which movements are pre-played compared to the movement of the real robot and alert the human operator before a real collision can happen. Successful virtual collision tests showed that the operator receives the warning signal immediately in the form of a mobile device vibration to modify the planned movement.

In some cases, real-time path planning is required in a changing environment, e.g. when the position of the workpiece to be gripped is variable (e.g. bin picking). In a collaborative environment, this is a serious security challenge that the whole system has to manage. The static parts of the environment can be checked time by time with collision detection but the presence of the human means that the "simple" collision detection is not enough. This was the main reason for the current research and development presented in this paper.

ACKNOWLEDGMENT

This research has been supported by the „Felsőoktatási Intézményi Kiválósági Program – Digitális ipari technológiák kutatása a Széchenyi István Egyetemen" project (20523-3/2018/FEKUTSTRAT) and by the GINOP-2.3.2-15-2016-00002 grant on an "Industry 4.0 research and innovation center of excellence".

REFERENCES

[1] Digital Economy and Society Index Report 2019, Integration of Digital Technology, https://ec.europa.eu/newsroom/dae/document.cfm?doc_id=59979. Accessed 21 September 2020.

[2] Shah, J., Wiken, J., Williams, B., & Breazeal, C. (2011, March). "Improved human-robot team performance using chaski, a human-inspired plan execution system". In Proceedings of the 6th international conference on Human-robot interaction, pp. 29-36.

[3] Gaspar, T., Ridge, B., Bevec, R., Bem, M., Kovač, I., Ude, A., & Gosar, Ž. (2017, July). "Rapid hardware and software reconfiguration in a robotic workcell". In 2017 18th International conference on advanced robotics (ICAR), IEEE, pp. 229-236.

[4] Abdelfetah Hentout, Mustapha Aouache, Abderraouf Maoudj & Isma Akli: "Human–robot interaction in industrial collaborative robotics: a literature review of the decade 2008–2017", Advanced Robotics, 33.15-16 (2019): 764-799.

[5] Zacharaki, A., Kostavelis, I., Gasteratos, A., & Dokas, I. (2020). "Safety bounds in human robot interaction: a survey. Safety science", 127, 104667.

[6] Morato, C., Kaipa, K. N., Zhao, B., and Gupta, S. K. (January 22, 2014). "Toward Safe Human Robot Collaboration by Using Multiple Kinects Based Real-Time Human Tracking", ASME. J. Comput. Inf. Sci. Eng. March 2014; 14(1): 011006.

[7] URCap Software Platform of Universal Robots, https://www.universal-robots.com/articles/ur/urcap-software-platform/. Accessed 21 September 2020.

[8] RobotiQ website, www.robotiq.com. Accessed 10 Aug. 2020.

[9] ApertusVR Documentation, GitBook, https://apertus.gitbook.io/vr/. Accessed 10 Aug. 2020.

[10] PREHUROCO sample files on Github, https://github.com/MTASZTAKI/ApertusVR/tree/0.9.1/samples/collisionDetection. Accessed 21 September 2020.

[11] Vibration API (Second Edition), W3C Recommendation 18 October 2016, https://www.w3.org/TR/vibration/. Accessed 10 Aug. 2020.

[12] Collision Detection Plugin, ApertusVR on Github, https://github.com/MTASZTAKI/ApertusVR/tree/0.9.1/plugins/physics/bulletPhysics. Accessed 21 September 2020.

[13] Visualization Plugin, ApertusVR on Github, https://github.com/MTASZTAKI/ApertusVR/tree/0.9.1/plugins/render/ogreRender. Accessed 21 September 2020.

[14] Kinect Plugin, ApertusVR on Github, https://github.com/MTASZTAKI/ApertusVR/tree/0.9.1/plugins/track/body/kinect. Accessed 21 September 2020.

[15] Websocket Server Plugin, ApertusVR on Github, https://github.com/MTASZTAKI/ApertusVR/tree/0.9.1/plugins/languageAPI/webSocketServer. Accessed 21 September 2020.

[16] X3D Loader Plugin, ApertusVR on Github, https://github.com/MTASZTAKI/ApertusVR/tree/0.9.1/plugins/languageAPI/jsAPI/nodeJsPlugin/js/plugins/x3dLoader. Accessed 21 September 2020.

[17] NodeJS Plugin, ApertusVR on Github, https://github.com/MTASZTAKI/ApertusVR/tree/0.9.1/plugins/languageAPI/jsAPI/nodeJsPlugin. Accessed 21 September 2020.

[18] Jsonlist file of the simulated UR5 robot movement, https://github.com/MTASZTAKI/ApertusVR/blob/89aefbc9b2a0e7524092b87d728ad539cfc0a856/plugins/languageAPI/jsAPI/nodeJsPlugin/js/plugins/httpSimulator/ur5.jsonlist. Accessed 16 September 2020. Accessed 21 September 2020.