# Investigations on collaborative remote control of virtual robotic manipulators by using a Kinect v2 sensor

*Magdalena* Banda[1] *Valentin* Ciupe[1,*], *Cristian* Moldovan[1] , *Inocențiu* Maniu[1] and *Robert* Kristof[1]

[1]Politehnica University of Timisoara, Mechatronics Department, 300222, 1 M. Viteazul Bd., Timisoara, Romania

**Abstract.** The present work deals with design aspects and testing results of a novel solution regarding the remote control of two virtual robotic manipulators by using hands motion tracking and palms gesture recognition. The spatial position and orientation of the user's left and right hands' wrists are interpreted by a custom built software application and then forwarded as positioning data and gripper status to a V-Rep server running the simulation scene. The goal of the presented setup is for the user to be able to manipulate objects in the scene in a collaborative way, i.e. pick one object with the first robot and give it to the second one for elsewhere depositing. The real-life applications for this approach are to provide a framework for simplified operator training and programming of robots used in pick-and-place or assembly operations, and also in healthcare for assisting patients in hand - coordination and mobility recovery.

## 1 Introduction

Remote control of robotic manipulators is an interesting and attractive idea, considering the increasing number of robotic applications in industry and other service areas. Most of the times programming an industrial/serial robot involves learning how to operate the robot and even if basic principles of motion programming are similar between manufacturers, due to the particularities of each model, regular users shy away when it comes to modify some positions or teach new paths.

The goal of the present work is to start the development of a novel framework that will allow remote control of robotic manipulators. In particular, the paper deals with the hardware & software design aspects and testing results, related to the remote control of two virtual robotic manipulators.

For this type of control, the user should move the hands towards desired target positions. A Kinect for Xbox One will track the hands motion and recognize palms gestures and then forward this data to a virtual simulation environment (V-Rep) that animates a scene with two robotic manipulators, accordingly.

The Kinect sensor comes handy when the user has to interact with the environment without contact. One could imagine scenarios where the user has to control some form of

* Corresponding author: valentin.ciupe@upt.ro

object manipulation without any contact, such as clean rooms, or where a joystick is not practical or an exoskeleton is too big/heavy/expensive.

Figure 1 represents the core idea of the application, which is targeted at the user experience. The user should be able to manipulate objects in a virtual scene or conduct manipulation in a collaborative way (pick one object with the first robot and give it to the second one for elsewhere depositing).

The system designed for this task requires simple setup (no complicated calibration procedures, not too many interconnects) and provides a short learning curve. The proposed solution is expandable to real life robotic manipulators and is mainly usable for training and robot programming and also in healthcare applications.

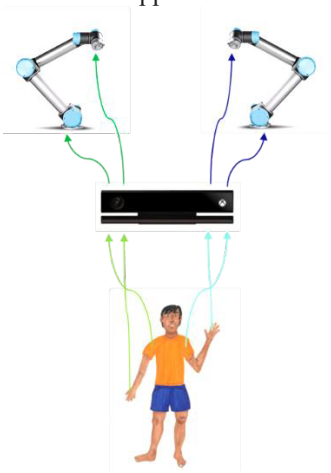


**Fig. 1.** Idea – remote control of robotic manipulators.

Other related projects involve the Kinect sensor (or other motion tracking) in robot motion and control, with real manipulators and virtual ones: RoKi - Robot control using Microsoft Kinect [1] (control of two KUKA robots); Kinect Hand Guiding of Robot Arm [2] (control of a UR10 robot); Imitation in V-Rep using Kinect [3] (control of a humanoid robot) – one of the few applications involving both Kinect v2 and V-Rep; Operate robots using LeapMotion in V-Rep [4] (KUKA robot control using different hand motion tracking); V-Rep Dynamixel Haptic [5] (developed by one of the authors - UR5 robot remote control in V-Rep but using a set of smart servo motors as a master device, with force-feedback).

Considering the above projects, one could notice that ***up to now there is no similar approach***, i.e. controlling two manipulators with Kinect v2 sensor within the V-Rep simulation platform.

## 2 Application design

By integrating adapted hardware components (sensor, PC) with custom designed software parts (positioning program, simulation environment scene) the goal of the current research could be achieved, the required elements and the relations between them being detailed in the following paragraphs.

### 2.1 Getting the user's pose

In order to get the user's postural pose, and especially the wrists' positions and orientations a Kinect for Xbox One sensor [6] was employed (fig. 2). The sensor itself was designed for gaming consoles and is essentially a combination of color and infrared cameras

that are able to register depth information. Since the sensor has internal processing it can track up to 6 bodies and generate the skeletons of tracked bodies, providing 3D position information for 25 body joints.

Among other characteristics of the Kinect sensor, those making it a viable choice for PC-based body tracking are: wide-angle ToF camera with active IR sensor; 1080p RGB camera and 512x424 IR camera providing 30 FPS, a FoV of 70x60 deg., measuring distance from 0.5 up to 4.5 m, an object pixel size of 1.4…12 mm all these combined with a full featured Windows/Visual Studio SDK.

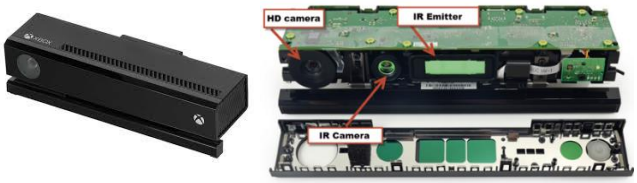



**Fig. 2.** The Kinect for Xbox One (Kinect v2) sensor.

When a person walks into the view of the Kinect sensor, internally the sensor determines a basic skeleton for that person [7]. For the current application only the joints highlighted in fig. 3 present importance and are further processed for the following tasks – related to the virtual robotic manipulators:
- *SpineShoulder* (base for distance and for when hand overlaps shoulder)
- *ShoulderLeft* and *Right* (base for relative distance to wrist, attached to robot's base)
- *WristLeft* and *Right* (desired position, attached to robot's TCP)
- *HandLeft* and *Right* (desired orientation, attached to robot TCP)
- *Hand_state_Left* and *Right* (this is not a joint, but a signal, used to de-/activate griper).

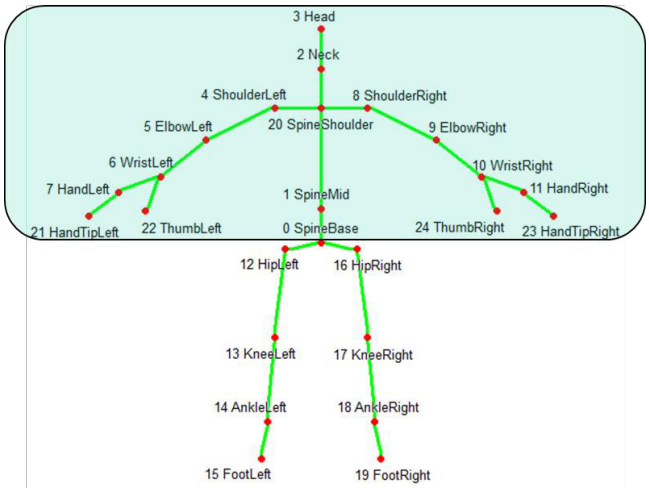



**Fig. 3.** Kinect v2 skeleton arrangement

## 2.2 Virtual simulation environment

The other component of the application is a virtual environment that can simulate and display the robots' positions/motion and provide a visual feedback to the user. This approach was selected for portability of the system and also due to the lack of two identical real manipulators available on site. Other requirements for the simulation environment are: to allow for fast scene definition (ready-made robots and other objects and equipment), to

admit/process external data and to have an easily configurable physics engine. Based on these criteria the general 3D and parametric modeling programs were ruled out and the Virtual Robot Experimentation Platform (V-Rep) [8] was selected. This software platform allows the use of various robots from its own database (or the creation of user defined components), and it has the possibility of attaching Lua scripts to objects in order to program their behavior and interaction. But V-Rep can also act as a server permitting clients to access a remote variant of almost all internally supported functions and is free for educational purposes. A V-Rep screen capture of the scene developed for this application is displayed in fig. 4.

This scene consists in having 2 x UR10 robotic manipulators with suction cups as grippers and with inverse kinematics solver attached to both robots. The simulation is running a server, expecting remote commands (positions & orientations of both TCPs and vacuum grippers state). The scene also features 2 x cubes with mass and collision properties enabled, that will be used as remote manipulation training targets.

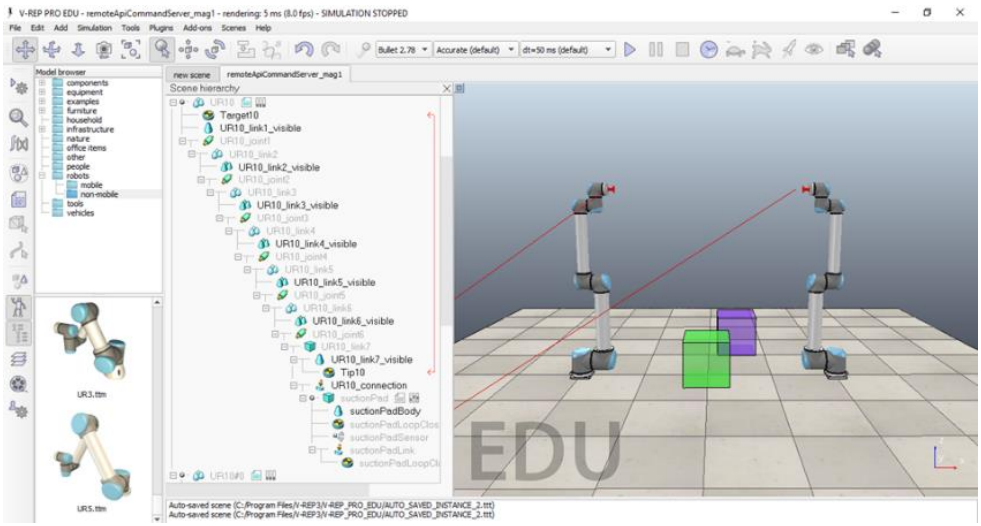



**Fig. 4.** Simulation scene form V-Rep.

Interfacing Kinect with V-Rep is appealing because it offers a simple, yet effective virtual environment for the user to interact with, compared with other simulators or CAD environments.

## 2.3 System architecture

The overall architecture of the system is composed of both hardware and software elements, as depicted in figure 5.

In order for the Kinect sensor to be compatible with a PC, some adaptation has to be done, since the Kinect for Windows commercial adapter was not available for purchase in the area. The hardware mod used the information form [9] and required the proprietary connector to be replaced with a regular USB3.0 type A plug and to add a 12V p.s. connector. After connecting the sensor to a 12V/2A power supply and to the PC, it was detected by Windows and installed correctly.

The software components include the V-Rep simulator (running a remote API server) and the custom built Positioning program that relays the user's joint positions to the virtual robotic manipulators.

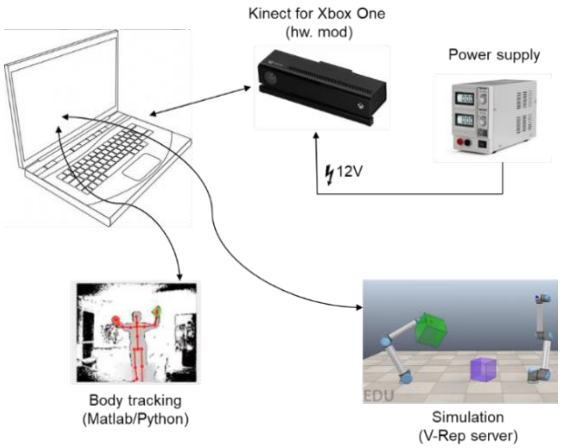



**Fig. 5.** Overview of system architecture.

General workflow of the application implies that the spatial position and orientation of the user's left and right hands' wrists are:
- extracted from Kinect skeleton (x, y, z; qw, qx, qy, qz),
- interpreted and post-processed by a custom software application (Positioning program),
- forwarded as positioning data and gripper status to the V-Rep server running the simulation scene.

## 3 Positioning program

Since V-Rep does not directly support the Kinect v2 sensor, a software interface had to be created in order to control the virtual robot manipulators. Two options were explored and the positioning programs were developed by the authors:
- a Matlab program using Kin2 (a Kinect v2 toolbox for Matlab) [10] for skeleton data and the remote API commands for robots positioning;
- a Python program using PyKinect2 (a wrapper to expose Kinect for Windows v2 API in Python) [11] also combined with the remote API commands for positioning.

Due to the fact that V-REP natively supports both Matlab and Python and has remote API wrappers and usage examples for them allowed for proper code integration.

The same approach was used for both options regarding the wrists coordinates, the three-dimensional difference between the *shoulder* (or *spine_central* when *shoulder* and *wrist* align towards the sensor) and the *wrist*, such that the position of the user in front of the sensor is less relevant, as long as the distance to it is ~1,5m.

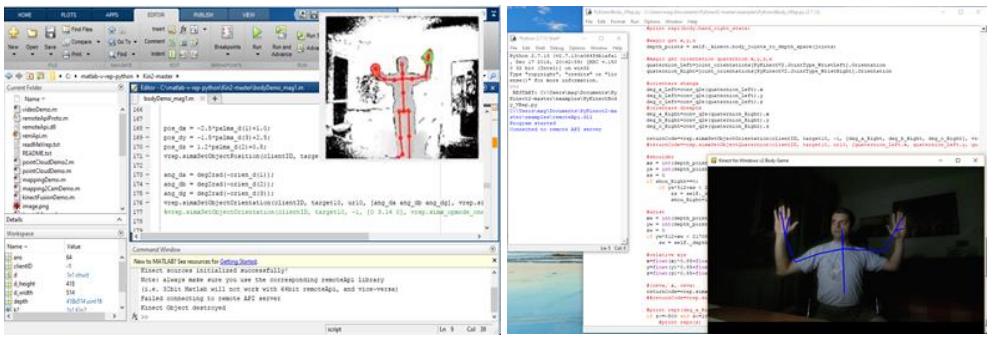
The Matlab program uses the depth frames for joint-sensor distance with the skeleton overlaid upon this image. The code reads the wrists linear coordinates and orientation quaternions and sends the filtered (moving average filter) and scaled values as parameters in the *vrep.simxSetObjectPosition* and *simxSetObjectOrientation* [12] commands to the V-Rep server. Only the translations are scaled in order to cover as much as possible the robots' workspace. Along scaling an offset is also needed because of different distances between robots (and user's shoulders). The scaling and offset absolute values range from 0.8 to 2.5 and have been experimentally determined considering the Kinect values obtained for one specific user standing at 1,5m away from the sensor.

The relations between Kinect values and V-rep values are expressed in (1), where $P_{xyz}$ are the V-Rep values, $T_{xyz}$ are the Kinect values and $k_{xyz}$, $c_{xyz}$ are the scaling and offset factors.

Although the Kinect is offering quaternions for non-leaf joint rotations, the toolbox has the possibility to provide Euler converted angles that are needed for V-Rep object orientation.

$$\begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} k_x \cdot T_x + c_x \\ k_y \cdot T_y + c_y \\ k_z \cdot T_z + c_z \end{bmatrix} \quad (1)$$

Figure 6 a) is displaying a depth image from the Kinect, with overlaid skeleton (shown in red, with right palm closed and left palm opened).



a) b)

**Fig. 6.** a) Matlab code and b/w depth image; b) Python code and color image.

The Python program also uses the depth frames for joint-sensor distance but the skeleton is overlaid on the color image (only the required joints and bones are drawn – fig. 6 b). In this case the wrist coordinates are processed in the same way as before but for rotations, Quaternion-Euler conversion is needed and this is implemented into the code, as per (2). Following the above described algorithm the scaled and filtered values are sent to the V-Rep simulation server.

$$\begin{bmatrix} R_E \\ P_E \\ Y_E \end{bmatrix} = \begin{bmatrix} atan2\left(2 \cdot (q_w \cdot q_x + q_y \cdot q_z), 1 - 2 \cdot (q_x^2 + q_y^2)\right) \\ asin\left(2 \cdot (q_w \cdot q_y + q_z \cdot q_x)\right) \\ atan2\left(2 \cdot (q_w \cdot q_z + q_x \cdot q_y), 1 - 2 \cdot (q_y^2 + q_z^2)\right) \end{bmatrix} \quad (2)$$

In terms of functionality and user experience both approaches produced similar results. The advantage of Python variant is that it doesn't need Matlab to be installed, but the Kinect library is a little bit more limited.

The flowchart in fig. 7 describes how the data acquiring and processing is done and also how the positions and orientations angles of the desired joints are forwarded to the V-Rep simulation server.

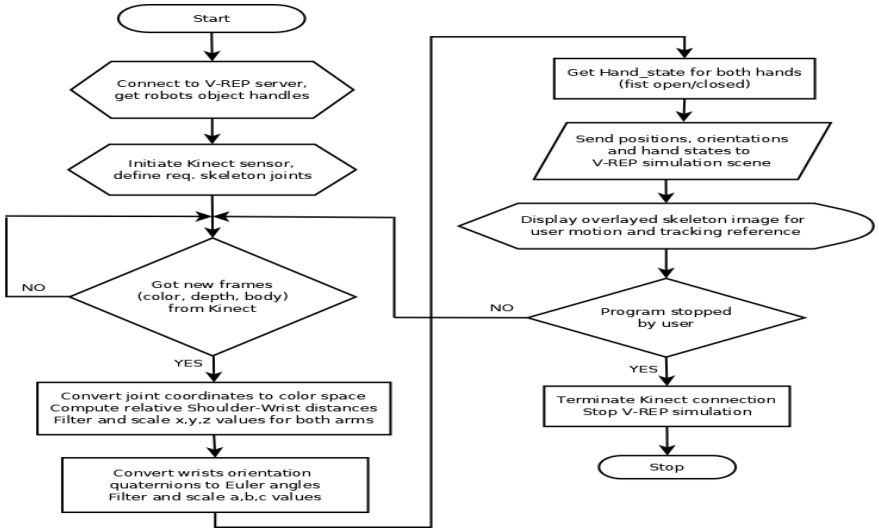
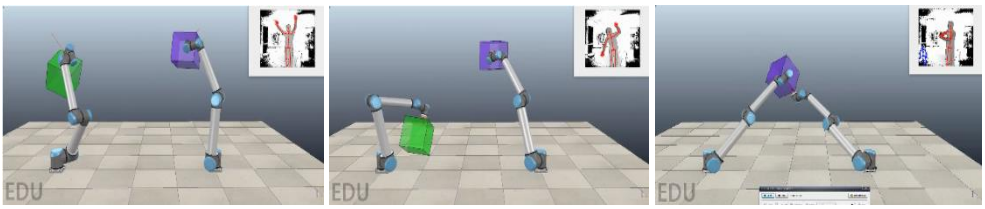



**Fig. 7.** Simplified flowchart of the Positioning program.

## 4 Results

Implementation of the Positioning program and testing the proposed solution resulted in a set of collaborative manipulation experiments where the user could grab the two colored cubes and place them elsewhere or could handover one cube. Since using the Kinect, a natural user interface (NUI) is employed, this confirmed that untrained users could learn to control the robots very fast. The subjects (aged 13 – 55) needed an average time of only 2 minutes of learning and testing before useful manipulation. Fig. 8 presents one of the authors – skeleton in the upper right corner – successfully manipulating 2 cubes (a, b) and handing over the blue cube, from the left robot to the right robot (c).



a) b) c)

**Fig. 8.** a), b) Manipulation example with 2 cubes; c) Collaborative manipulation – cube handover.

Video demos of the above examples can be seen at the following links:
https://www.youtube.com/watch?v=906uyp1m03g
https://www.youtube.com/watch?v=mlUnzK2Qetc

## 5 Future work

Being able to move the simulated robots using the Kinect sensor is a good proof of concept, but the ultimate goal is to connect to the real robots: KUKA KR15+KR125, ABB YuMi and other available robots in our labs that support remote positioning commands (fig. 9). Another idea is to auto calibrate the motion tracking for maximum robot workspace related to maximum user arms extension and to build a GUI for easy app setup. Enhanced

results might be expected by using a second Kinect for improved body tracking and also by using the Kinect mic array for voice commands (record path points etc.).

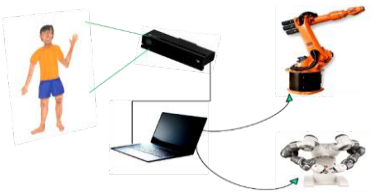


**Fig. 9.** Proposed further development concept.

## 6 Conclusions

The designed system architecture and the positioning programs developed by the authors provide a novel framework for simplified operator training and programming of robots. The Kinect sensor is accurate enough for 3D tracking of user's arms. In order to get advantage of the robot's workspace an optimal distance of 1.5m from sensor must be ensured (for an averaged size person). Untrained users needed short time for learning and testing before manipulation. The proposed concept is suitable for simple manipulation (pick and place) and collaborative manipulation (objects handover). Depending on the scaling in the positions processing program, the amplitude of user's motions can be extended or limited for rehabilitation purposes (assisting in hand coordination and mobility recovery).

## References

1. RoKi - Robot control using Microsoft Kinect, *www.youtube.com/watch?v=kECNyr7v0kM* (acc. 02.19)
2. S. Moe, I. Schj˘ lberg, Real-time hand guiding of industrial manipulator in 5 DOF using Microsoft Kinect and accelerometer, *Proceedings - IEEE International Workshop on Robot and Human Interactive Communication*, 644-649, DOI: 10.1109/ROMAN.2013.6628421, (2013)
3. Imitation in V-Rep using Kinect, *www.youtube.com/watch?v=kFE0lhytqPY* (acc. 02.19)
4. Operate robots using LeapMotion in V-Rep, *www.youtube.com/watch?v=BT3umY1lGu8* (acc. 02.19)
5. V. Ciupe, E-Ch. Lovasz, e.a, Testing the haptic exoskeleton actuators in a virtual environment, *2015 IFToMM*, DOI: 10.6567/IFToMM.14TH.WC.PS13.015, (2015)
6. Wikipedia, The Kinect sensor, *https://en.wikipedia.org/wiki/Kinect* (acc. 02.19)
7. R. Hoover, Adventures in Motion Capture: Using Kinect Data (Part 3), *www.sealeftstudios.com/blog/blog20160715.php* (acc. 02.19)
8. Coppelia Robotics, Virtual Robot Experimentation Platform, *www.coppeliarobotics.com* (acc. 02.19)
9. The OpenKinect project, *www.openkinect.org/wiki/Kinect_2* (acc. 02.19)
10. J. Terven, D. Cordova-Esparza, Kin2. A Kinect 2 Toolbox for MATLAB, *Science of Computer Programming*, DOI: 10.1016/j.scico.2016.05.009, (2016)
11. PyKinect2 GitHub project, *www.github.com/Kinect/PyKinect2* (acc. 02.19)
12. Coppelia Robotics, V-Rep Remote API, *www.coppeliarobotics.com/helpFiles/en/remoteApiOverview.htm* (acc. 02.19)