The Eleventh International Conference on
Engineering Computational Technology
23–25 August, 2022 | Montpellier, France

Proceedings of the Eleventh International Conference on
Engineering Computational Technology
Edited by B.H.V. Topping and P. Iványi
Civil-Comp Conferences, Volume 2, Paper 16.2
Civil-Comp Press, Edinburgh, United Kingdom, 2022, doi: 10.4203/ccc.2.16.2

Robotic Simulation to implementation: An industrial case study

M. Gautam, H.-M. Yonamine and F. Christophe

HAMK Tech Robotics Research group, Häme University of Applied Sciences
Riihimäki, Finland

Abstract

This study presents an industrial case study of developing a robotics application from a virtual environment and directly deploying the programs to a collaborative robot. In the physical setup, we tested how unchanged code developed in the virtual environment performs when faced with the real concrete setup. Our study outlines remaining issues dealing, for example, with the surface roughness of wooden parts that the robot needs to grab. In the meantime, these are parameters that are difficult to consider in a virtual environment. In this specific example, deployment from simulation to actual environment must undergo through several fixing steps, and therefore might require as much effort as deploying the actual solution directly to the floor. However, we also noticed that developing in a simulation environment provides plenty of advantages such as not having to interrupt the production, fast development of parametrized robot movements, and being able to rapidly produce different working solutions for grabbing and disposing of pieces.

Our study concludes that deployment of offline programming requires detail understanding of the virtual simulation software and robot programming interface. The balanced combination of offline simulation and online programming is required at the end for actual implementation.

Keywords: robotic simulation, visual components, ursim, wood industry.

1 Introduction

Robotics simulation has been in robotics field for decades. Pan et al. [1] present an in-depth article on different programming methods used in the industry. Moving away from manufacturer-based software such as ABB Robot Studio or KUKAsim, generic factory, robot simulation and offline programming has started to get more popular, as for example Visual Components (VC) or RoboDK. We evaluate in this paper how applicable VC is for the deployment of offline programs. We identify the challenges to successfully deploy solutions straight from VC. Universal robot simulator (URSIM) was used for offline programming in addition to VC.

The industrial task is to assemble wooden pallets on a rotating table. The robot task is to pick and place wooden planks. Nailing of the planks is done by a machine that integrates a rotating table. For this experiment, the robot used is UR10.

Our study outlines issues in developing simulation, for example, with the surface roughness of wooden parts that the robot needs to grab. These types of parameters are difficult to consider in a virtual environment. We conclude that the deployment of offline programming requires detail understanding of the virtual simulation software and robot programming interface. The balanced combination of offline simulation and online programming is required at the end for actual implementation.

2 Methods

This section is organized according to Figure 1 where we present our method. First, we explain how the virtual environment was developed and robot programmed. Second, we present how to deploy programs to the actual robot. Third, we explain how our tests were achieved in the physical setup.

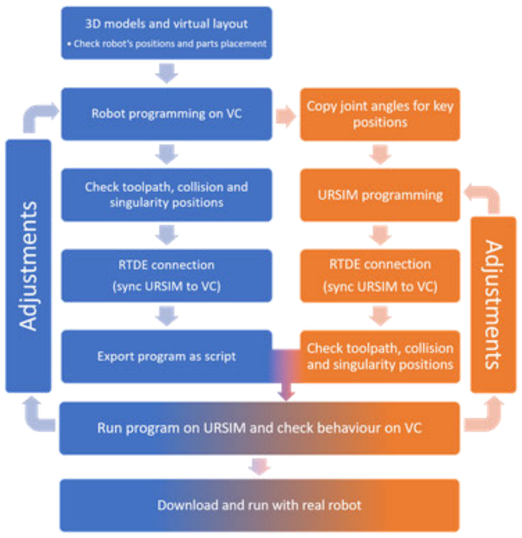

3D models and virtual layout
• Check robot's positions and parts placement
Robot programming on VC
Copy joint angles for key positions
Check toolpath, collision and singularity positions
URSIM programming
RTDE connection (sync URSIM to VC)
RTDE connection (sync URSIM to VC)
Export program as script
Check toolpath, collision and singularity positions
Adjustments
Adjustments
Run program on URSIM and check behaviour on VC
Download and run with real robot

Figure 1: Flowchart of our proposed development process.

1. Three-dimensional (3D) environment modelling and robot programming

Required measurements were taken from the factory workstation. From the combinations of the measurements and the 3D models, the virtual environment was set, and then the robot was programmed. The virtual layout created in VC is shown in Figure 2.

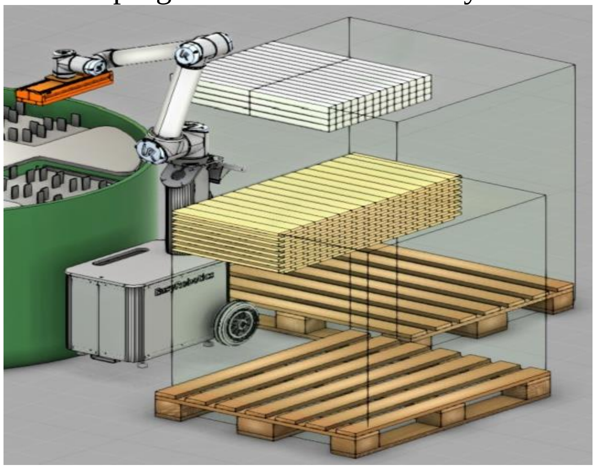


Figure 2: Virtual layout.

Base frame feature on VC was utilized to attach a set of points to one object. Using these bases meant that once the program is done, the positions should adjust itself whenever the objects are moved.5 bases were defined as shown in Figure 3. They were numbered according to the order in which they were used. Collision avoidance feature in VC was used to ensure no collision between robot tool and objects in the environment.

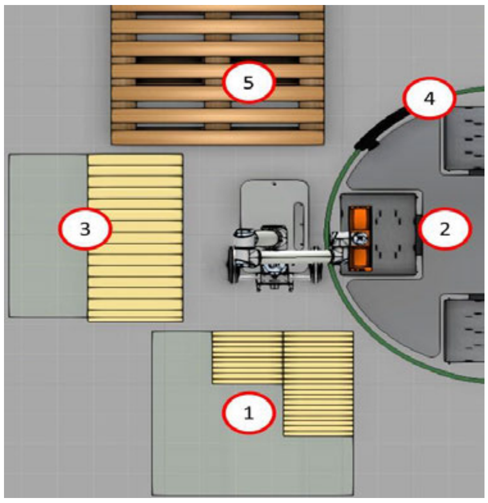


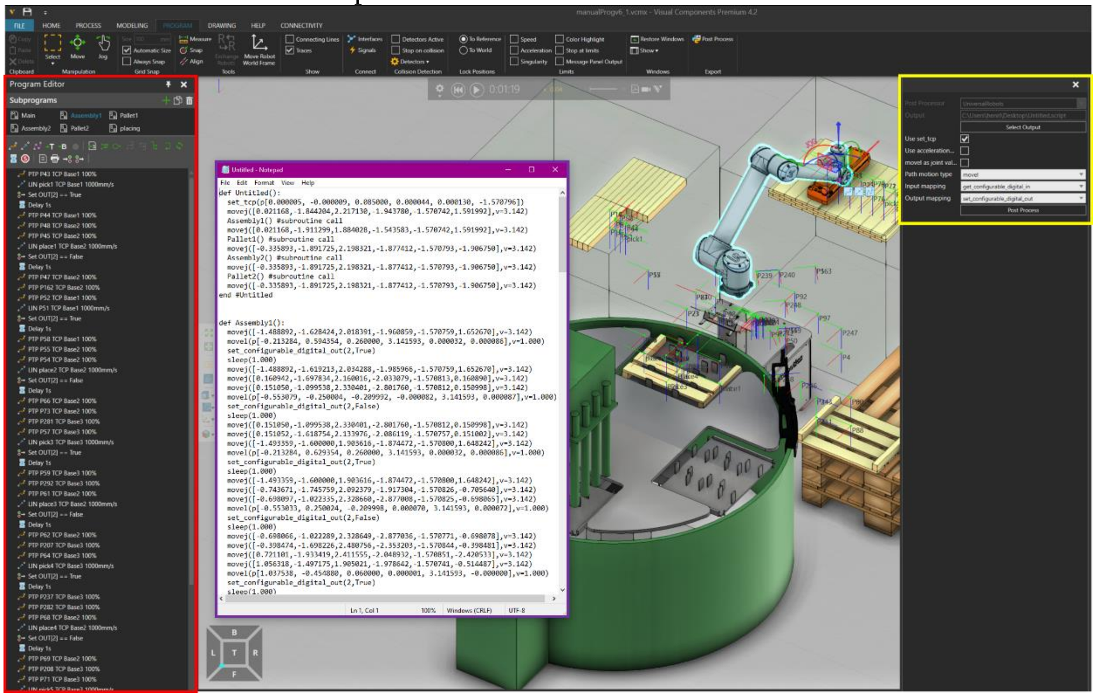
Figure 3: Bases from virtual environment 1) Side plank stack, 2) Rotating table, 3) Flat plank stack, 4) Flipping fixture 5) Bigger pallet for stacking the assembled pallets.

2. Deploying Programs to real robot

Two approaches were used for deploying the program. The first approach was to deploy VC developed program as script utilizing URSIM as validation tool for program. The second approach was to deploy robot program developed in URSIM utilizing VC simulation to obtain initial joint positions of robot relative to objects location and validating the developed program behaviour according to the simulated environment.

2.1 Exporting script from VC

VC script was exported and ran on URSIM.URSIM was connected via real time data exchange (RTDE) connection to VC. This connection enables the URSIM to control the robot present on Visual Components. The script file shown in Figure 4 was then saved on an usb stick and uploaded to the robot.



```
PTP P43 TCP Base1 100%
LIN pick1 TCP Base1 1000mm/s
Set OUT[2] == True
Delay 1s
PTP P44 TCP Base1 100%
PTP P48 TCP Base2 100%
PTP P45 TCP Base2 100%
LIN place1 TCP Base2 1000mm/s
Set OUT[2] == False
Delay 1s
PTP P47 TCP Base2 100%
PTP P162 TCP Base2 100%
PTP P52 TCP Base1 100%
LIN P51 TCP Base1 1000mm/s
Set OUT[2] == True
Delay 1s
PTP P58 TCP Base1 100%
PTP P55 TCP Base2 100%
PTP P54 TCP Base2 100%
LIN place2 TCP Base2 1000mm/s
Set OUT[2] == False
Delay 1s
PTP P66 TCP Base2 100%
PTP P73 TCP Base2 100%
PTP P281 TCP Base3 100%
PTP P57 TCP Base3 100%
LIN pick3 TCP Base3 1000mm/s
Set OUT[2] == True
Delay 1s
PTP P59 TCP Base3 100%
PTP P292 TCP Base3 100%
PTP P61 TCP Base2 100%
LIN place3 TCP Base2 1000mm/s
Set OUT[2] == False
Delay 1s
PTP P62 TCP Base2 100%
PTP P207 TCP Base3 100%
PTP P64 TCP Base3 100%
LIN pick4 TCP Base3 1000mm/s
Set OUT[2] == True
Delay 1s
PTP P237 TCP Base3 100%
PTP P282 TCP Base3 100%
PTP P68 TCP Base2 100%
LIN place4 TCP Base2 1000mm/s
Set OUT[2] == False
Delay 1s
PTP P69 TCP Base2 100%
PTP P208 TCP Base3 100%
PTP P71 TCP Base3 100%
```

```
def Untitled():
  set_tcp(p[0.000005, -0.000009, 0.085000, 0.000044, 0.000130, -1.570796])
  movej([0.021168,-1.844204,2.217130,-1.943780,-1.570742,1.591992],v=3.142)
  Assembly1() #subroutine call
  movej([0.021168,-1.911299,1.884028,-1.543583,-1.570742,1.591992],v=3.142)
  Pallet1() #subroutine call
  movej([-0.335893,-1.891725,2.198321,-1.877412,-1.570793,-1.906750],v=3.142)
  Assembly2() #subroutine call
  movej([-0.335893,-1.891725,2.198321,-1.877412,-1.570793,-1.906750],v=3.142)
  Pallet2() #subroutine call
  movej([-0.335893,-1.891725,2.198321,-1.877412,-1.570793,-1.906750],v=3.142)
end #Untitled


def Assembly1():
  movej([-1.488892,-1.628424,2.018391,-1.960859,-1.570759,1.652670],v=3.142)
  movel(p[-0.213284, 0.594354, 0.260000, 3.141593, 0.000032, 0.000086],v=1.000)
  set_configurable_digital_out(2,True)
  sleep(1.000)
  movej([-1.488892,-1.619213,2.034288,-1.985966,-1.570759,1.652670],v=3.142)
  movej([0.160942,-1.697834,2.160016,-2.033079,-1.570813,0.160890],v=3.142)
  movej([0.151050,-1.099538,2.330401,-2.801760,-1.570812,0.150998],v=3.142)
  movel(p[-0.553079, -0.250004, -0.209992, -0.000082, 3.141593, 0.000087],v=1.000)
  set_configurable_digital_out(2,False)
  sleep(1.000)
  movej([0.151050,-1.099538,2.330401,-2.801760,-1.570812,0.150998],v=3.142)
  movej([0.151052,-1.618754,2.133976,-2.086119,-1.570757,0.151002],v=3.142)
  movej([-1.493359,-1.600000,1.903616,-1.874472,-1.570800,1.648242],v=3.142)
  movel(p[-0.213284, 0.629354, 0.260000, 3.141593, 0.000032, 0.000086],v=1.000)
  set_configurable_digital_out(2,True)
  sleep(1.000)
  movej([-1.493359,-1.600000,1.903616,-1.874472,-1.570800,1.648242],v=3.142)
  movej([-0.743671,-1.745759,2.092379,-1.917304,-1.570826,-0.705640],v=3.142)
  movej([-0.698097,-1.022335,2.328660,-2.877008,-1.570825,-0.698065],v=3.142)
  movel(p[-0.553033, 0.250024, -0.209998, 0.000070, 3.141593, 0.000072],v=1.000)
  set_configurable_digital_out(2,False)
  sleep(1.000)
  movej([-0.698066,-1.022289,2.328649,-2.877036,-1.570771,-0.698078],v=3.142)
  movej([-0.398474,-1.698226,2.480756,-2.353203,-1.570844,-0.398481],v=3.142)
  movej([0.721101,-1.933419,2.411555,-2.048932,-1.570851,-2.420533],v=3.142)
  movej([1.056318,-1.497175,1.905021,-1.978642,-1.570741,-0.514487],v=3.142)
  movel(p[1.037538, -0.454880, 0.060000, 0.000001, 3.141593, -0.000000],v=1.000)
  set_configurable_digital_out(2,True)
  sleep(1.000)
```

Post Processor: UniversalRobots
Output: C:\Users\[illegible]\Desktop\Untitled.script
Select Output
Use set_tcp: [✓]
Use acceleration...: ☐
movel as joint val...: ☐
Path motion type: movel
Input mapping: get_configurable_digital_in
Output mapping: set_configurable_digital_out
Post Process

Figure 4: Example of application script developed in VC.

2.2 Deploy URSIM developed program

A robot program was developed in URSIM with variable positions listed below.

1. Home position, or safe position.
2. Picking approach position for the first side plank.
3. Approach for placing position, side plank 1.
4. Approach for placing position, side plank 2.
5. Transition position, from table to stack and vice versa.

6. Approach position for placing flat plank.
7. Approach position for picking flat plank

The variable positions were used for incremental movement generation by the robot program to complete the task. The variable position and the program working principle could be seen in Figure 5.

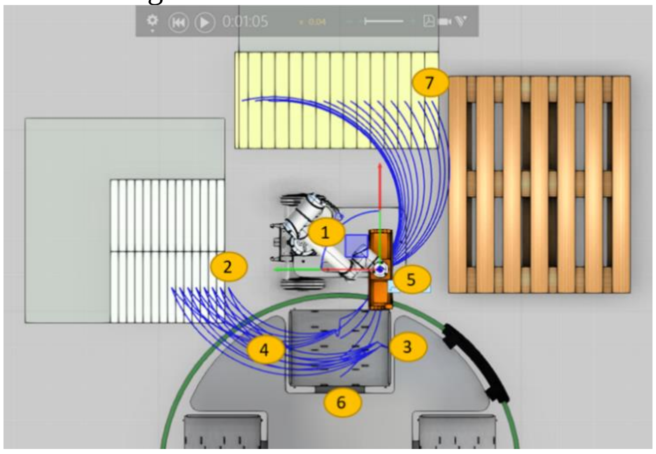


Figure 5: Simulated environment representing seven variable positions of robot.

The developed program movements were verified by running the program in URSIM and visualizing the movement in VC simulation. The developed program (.urp) was uploaded to the real robot using an usb drive.

3. On-site testing and needed adjustments

For testing on-site, a robot program developed in URSIM was used. The robot was taken to the factory site shown in Figure 6. 7 variable positions had to be adjusted according to the current layout. The layout was similar, the adjustments were easy to make.

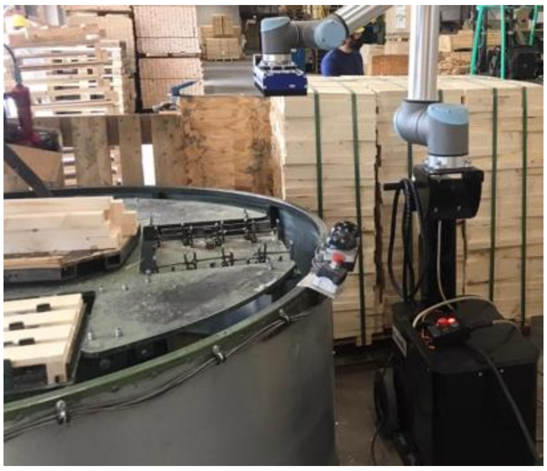


Figure 6: Layout on-site.

3 Results

This section report results in three segments as follows:

1. Running the VC script on robot
2. URSIM Program on robot
3. On-site test results

1. Running the VC script on robot

When running the script on robot, the linear movements had errors resulting in wrist twisting (Wrist 3). The tool would rotate while going downwards for the picking/placing position. One solution would be to simply change those 'moveL' to 'moveJ' commands, just like all the other movements, but that would result in a higher speed when reaching the parts, which may lead to collisions and/or emergency stops triggered by the robot. The other solution was to create a subprogram for this movement, which would be only from the approach position to the picking/placing position. All the other movements remained the same. Using this routine on the subprograms solved the tool twisting movement, at the same time as it shortened the number of lines on the program. All the other movements were taught manually, so all the approach points were independent. There is currently no feature on VC programming tab to make incremental movements or work with variable positions. There is no problem with them being independent, but it would take a lot longer to adjust it point by point.

2. URSIM program on robot

Programming on URSIM, with an urp program, where variable positions and incremental positions was an advantage. The planks were stacked in a known pattern in simulation and planks dimensions were known. This made it easy to set incremental movements relative to a defined initial position. These increments would be equivalent to the plank’s width, meaning that the approach positions would update itself while the program loops. Since the other movements were still scripted from VC, it was needed to teach those initial positions either by giving its coordinates or by the robot joints values. After a few iterations, it was noticed that it is better to teach the position by copying the joint values over the coordinates. By using the coordinates, a few positions put the robot in an awkward pose or resulted in some unexpected movements overall, due to the different joint configuration when setting the pose. It worked better when copying the joint values as shown in Figure 7. All the approach positions were set with the same height difference, so that the subprogram would work in every situation. The developed program was then uploaded to the robot and movement test were done without the actual setup. No problems were observed on the movement of the robot.

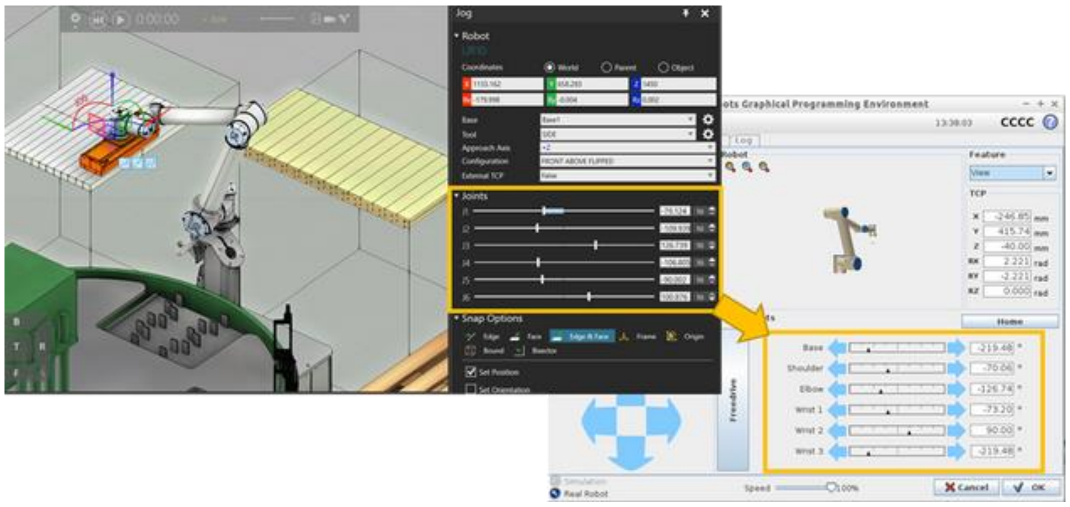


Figure 7. Joint values from VC to URSIM.

3. On-site test results

Even after adjustment of variable positions, the robot position deviations were high. The stacks of wood were not perfectly flat as assumed in the simulated version. The gripper tool could not grasp the planks due to this position error. This meant that it was not possible to assemble the pallets. The movements were replicated by the robot repeatedly, but without grasping wooden planks.

4 Conclusions and Contributions

For the application, the pick and place and palletizing applications are standard industrial problems and can be successfully deployed with the selections of engineered tools for the purpose and use of experienced system integrators. The lightweight collaborative robot is suitable for the application due to limited payload of the pallet and easy to interface system and tools available in the market. Online programming methods can resolve the problem discussed above about programming in virtual environment. The virtual environment is important for selection of tools and equipment's and preparing a requirement list for automatizing the application without interrupting the current production.

Onsite calibration of real robot is required to adjust the program generated in virtual environment compensating the variation in real and virtual world models. Machine vision, force sensors and a device to extend the robot's work envelope would be beneficial for onsite calibration and solving grasping issues due to planks position variability.

The deployment of offline programming requires detail understanding of the virtual simulation software and robot programming interface. Online programming methods can resolve the problem discussed above about programming in virtual environment.

Our future research will consist in reducing differences between physical setups and virtual models with the help of scanning capabilities, machine vision and force feedback sensors. In this way, we will approach the concept of virtual twin where things can be easily modified in the virtual world and instantly applied to its twin in the physical world.

Acknowledgements

The authors would like to thank European Union Regional Development Fund, Regional Council of Häme, Helsinki-Uusima Regional Council and Riihimäki City for funding the RoboTS project under which the presented work was carried out.

References

[1] Z. Pan, J. Polden, N. Larkin, S.V. Duin, J. Norrish. “Recent progress on programming methods for industrial robots”. In: Robotics and Computer-Integrated Manufacturing 28.2 (2012), pp. 87– 94. ISSN: 0736-5845. DOI: “https://doi.org/10.1016/j.rcim.2011.08.004”.