Faculty of Science, Technology, and Medicine

07.11.2024

Technical Report - Pick & Place Assembly Simulation

Submitted by:
Author: José Francisco Brás Loureiro
Supervisor: Prof. Dr. Thomas Engel
Reviewer: Felix Saretzky

Contents

List of Figures

List of Tables

Chapter 1

Technical Review

As established, the main objective of this paper is the evaluation of certain data-driven RCA methods regarding their application in manufacturing. To apply any method, data is required. Hence, a manufacturing system needs to be chosen. In our case, we chose to simulate an assembly line in CoppeliaSim and used the real-time data to test RCA methods. In the following sections, we will go over the decision process behind simulation over a real-world, and close with a detailed description of the selected model.

1.1 Manufacturing System

1.1.1 Trends

Manufacturing systems are embracing various trends [1]. It is possible to distinguish between different types of change, as outlined by ElMaraghy et al. [1]. On the one hand, there are innovations in technology, driving sensor technology, hybrid manufacturing, digitalization and more [1]. Other angles include advancements in product characteristics, which can be attributed to the adoption of more efficient production processes, increased modularity, and developing customer demands [1].

Business models from previous eras are gradually evolving to produce less waste and move to flexible manufacturing processes in order to generate more value for the customers. This shift is aligned with novel concepts like *circular economy* and sustainability [1].

Behind all those advances lie different factors, identified by Phuyal, Bista, and Bista [2] to be additive manufacturing, cloud computing, big data, simulations, and more.

1.1.2 Why Simulation

Simulation technology went through different advancements. The early days' simulation software were single-task-oriented modelling and analysis tools [3]. The next big leap introduced simulation capabilities into Computer-Aided-Design software. The present-day models are referred to as *Digital Twins*. They provide process and material handling optimization methods, as well as virtual commissioning [3].

As underlined in the previous chapter, manufacturing systems (MS) nowadays are becoming more complex [4]. Facing automation, big data, smart manufacturing, and more, manufacturing systems are constantly undergoing transformation caused by increased product variety and greater responsiveness [1]. To combat that, new smart manufacturing systems are increasingly looking towards simulation software, as it allows for evaluation of systems and other changes before physical implementation begins. Testing in a virtual environment of production flows and controls helps to efficiently and cost-effectively inspect production systems [2]. Generating a replica of the the physical world, the observations can be used to plan, schedule and optimize manufacturing systems [2]. A holistic approach of the system, i.e. supply chain and production, helps to facilitate the preparation of cost estimation, profit loss analysis and manufacturing processes.

Overall, simulation software helps to accelerate and optimize entire development processes. Using simulation software is crucial in testing data-driven RCI methods. While it would have been possible to use a physical production line, the technical feasibility of interventions is limited. A malfunction can be observable or detectable, but the actual failure cannot be forced.

Lastly, using simulations allows for rapid prototyping and custom system modelling. Overall, simulations might benefit most from the addition of actual physical systems. Having physical references helps to fast-forward the process, and it can be used to better conceptualize future alterations.

1.1.3 Simulation Software

As briefly mentioned in the previous chapter, there are many software tools available. In our case, we preferred CoppeliaSim Edu (Version 4.6.0) [5] due to its wide range of features and flexibility. Featuring multiple programming approaches makes CoppeliaSim very accessible. Our simulation, on the other hand, has been programmed entirely in Lua[1] — a small and simple object-oriented programming language. This was not chosen but comes as a consequence of working with provided model scripts.

[1] https://www.lua.org/

Thanks to the remote APIs, it is possible to access and control the simulation over external channels. One of the most important features is the implementation of physics engines, which allow for rigid body physics, collision detection, and soft body dynamics. This enables us to simulate real-world physical scenarios and events. Physics-based simulations can be very restrictive [6]. To combat that, CoppeliaSim uses a hybrid approach and only switches to dynamics if kinematics fail. Centered around robotics, it features multiple modules for robot control and path planning. The addition of dedicated vision and proximity modules makes it possible to design complex control systems.

After installation and opening the application, a blank scene will be available, as shown in Figure 1.1. Each scene will contain a main script for running the simulation. On the left side is the Model browser, with ready-to-use objects. There are various objects, including grippers, sensors, conveyor belts, robots, and more. By drag-drop, it is possible to add the desired models onto the scene. Afterwards, the object will be added to the scene hierarchy, as depicted in Figure 1.2. In the hierarchy, you see all the objects. Some objects can be linked to other objects to form more complex objects, as presented in Figure 1.2. Not every scene object has to have physical meaning. For instance, some elements can be used for controlling the model. The IRB140 robot consists of many links and joints, all connected via parent-child relationships. There are multiple scene object types available. The most used objects are shapes, joints, dummies, sensors, and cameras. For further information, please visit the manual[2].

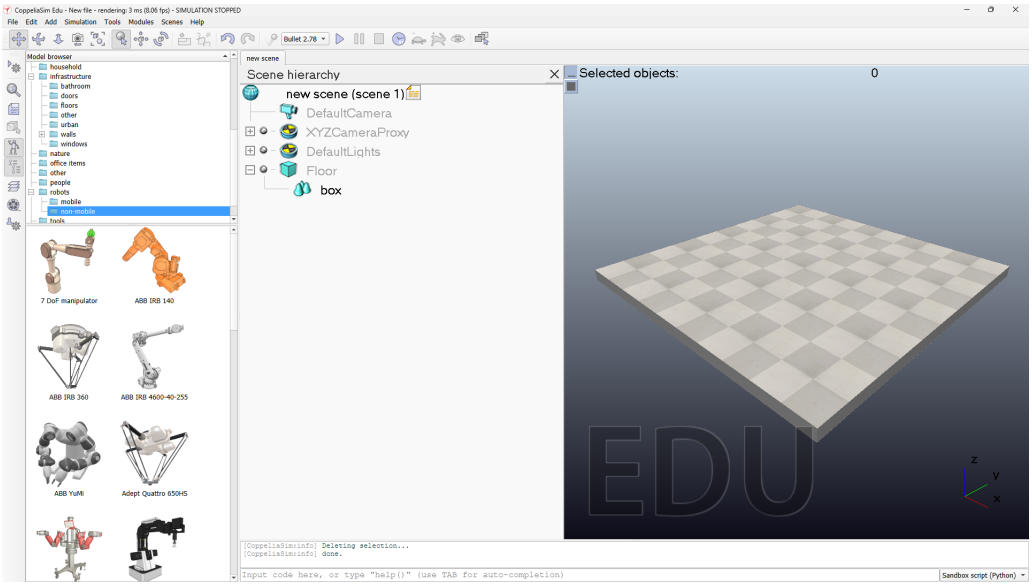


Figure 1.1 CoppeliaSim Blank Scene

[2]https://manual.coppeliarobotics.com/en/objects.htm

To add some functionalities to the objects, customization, and child scripts are used. The customization scripts are always active, and the child scripts only run, when the simulation has been launched. The objects featuring scripts are indicated by the script icon next to their name, as presented in Figure 1.2. Both script types make use of predefined callback functions, which get triggered, when certain actions take place.

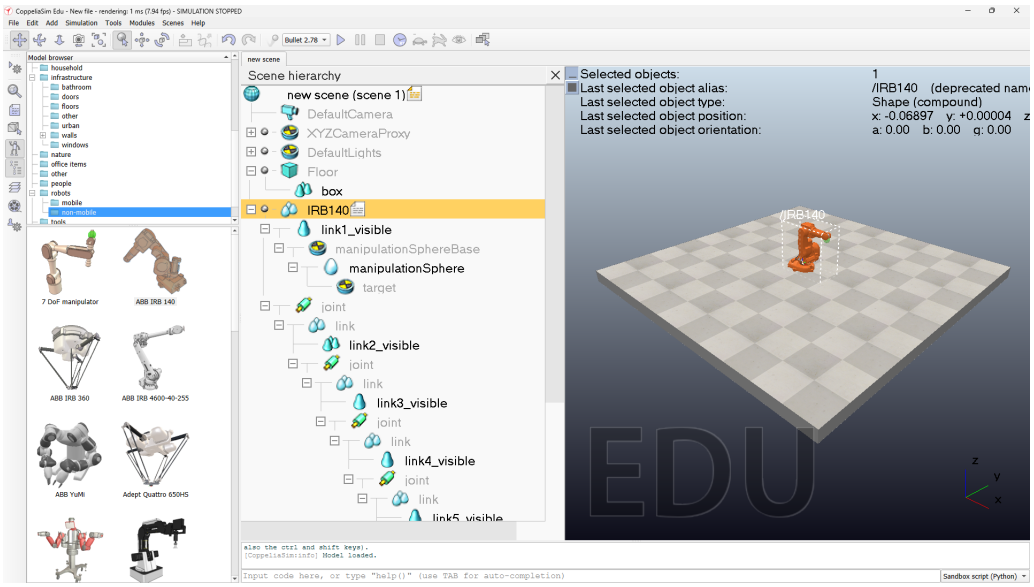


Figure 1.2 CoppeliaSim Scene Hierarchy

CoppeliaSim features an extensive API with many functions capable of accessing the parameters and properties of the scene objects, e.g. position. The simulation settings can be found in the menu bar under “Simulation”. The most important parameters are the Simulation time step and the Dynamics time step. The simulation runs in a simple loop, where each new simulation timestamp the main script gets executed. On top of that, CoppeliaSim distinguishes between threaded and non-threaded code. Threaded code will be paused and restarted, while non-threaded code needs to be finished within the time step. Lastly, there is an option to run the simulation in stepping mode, which can be used to synchronize external applications.

1.2 Simulation - Pick and Place

The previous section outlined key features and elements of CoppeliaSim. This section is intended to explain the chosen scenario. Confronted by the huge

capabilities provided by CoppeliaSim, many scenarios could have been simulated. We decided to emulate a simple Pick and Place assembly line as presented in Figure 1.3. This choice is grounded on an actual assembly line, which connects multiple elements together. It featured laser barriers, cameras and inductive sensors to control the process. In our case, the control logic works in a similar fashion.

To accelerate the development process of this simulation, already defined and designed models were used. The model scripts had to be adjusted for the chosen scenario. They were provided with the CoppeliaSim installation package and can be found under the following directory:

```
"...\CoppeliaRobotics\CoppeliaSimEdu\bwf\modelScripts"
```

This library seems to be been put together for BlueWorkForce (BWF)[7, 8], a company specifically working with robots. In 2019 the company has been acquired by another robotics company, called OnRobot [9].

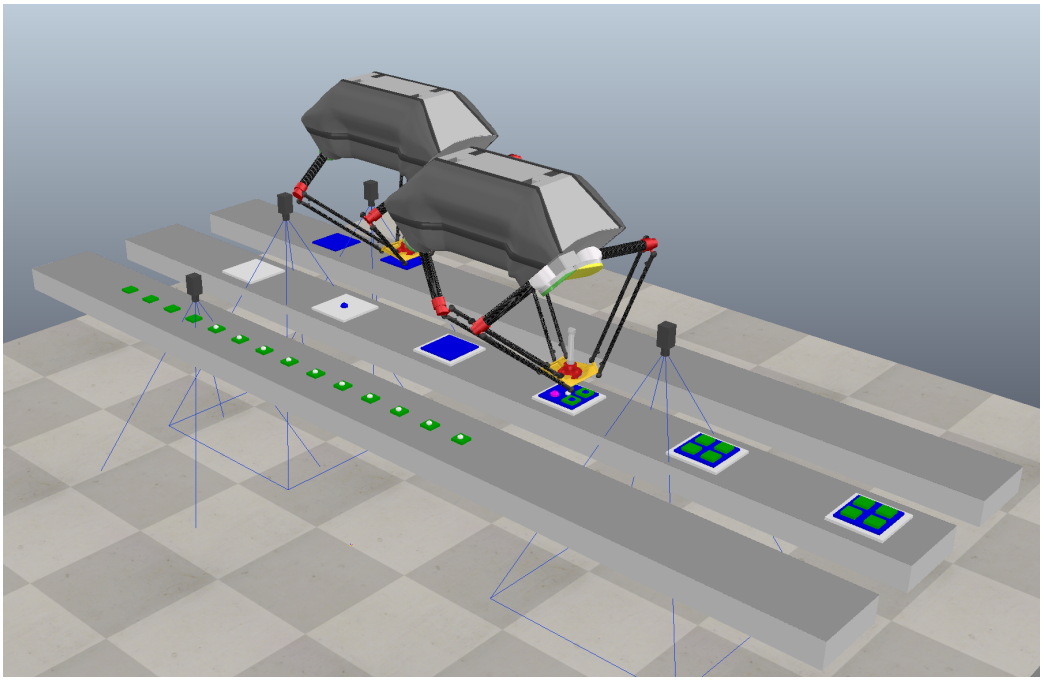


Figure 1.3 Assembly Line

1.2.1 Environment and Elements

Essentially, we have a simple assembly line composed of 3 conveyor belts parallel to each other. As seen in Figure 1.3, two distinct parallel robots will be used to

fit a white tray with a blue insert, which down the line will be populated with 4 individual green items, as demonstrated in Figure 1.4.

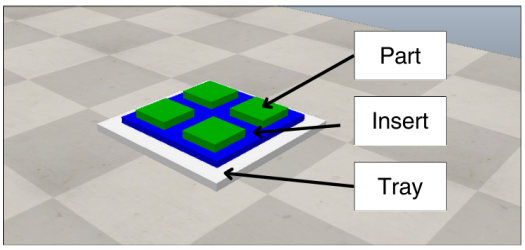


Figure 1.4 Complete Assembly

Besides the conveyors, the robots, and the parts, other elements were used. First, cameras have been positioned along the production line. Second, some hidden objects specifically used for robot control were implemented. Which includes detection barriers, tracking windows, and transfer windows. To avoid the simulation from crashing, the parts reaching the end of the assembly line drop into a part sink. It removes the object from the scene.

The parts are all dropped onto the conveyor according to fixed drop rates, which are being carried out by feeder objects. The available parts for the feeder are stored in a part repository. In Table 1.1, is a summary of all the essential parts. In Figure 1.5, all the key objects are labelled.

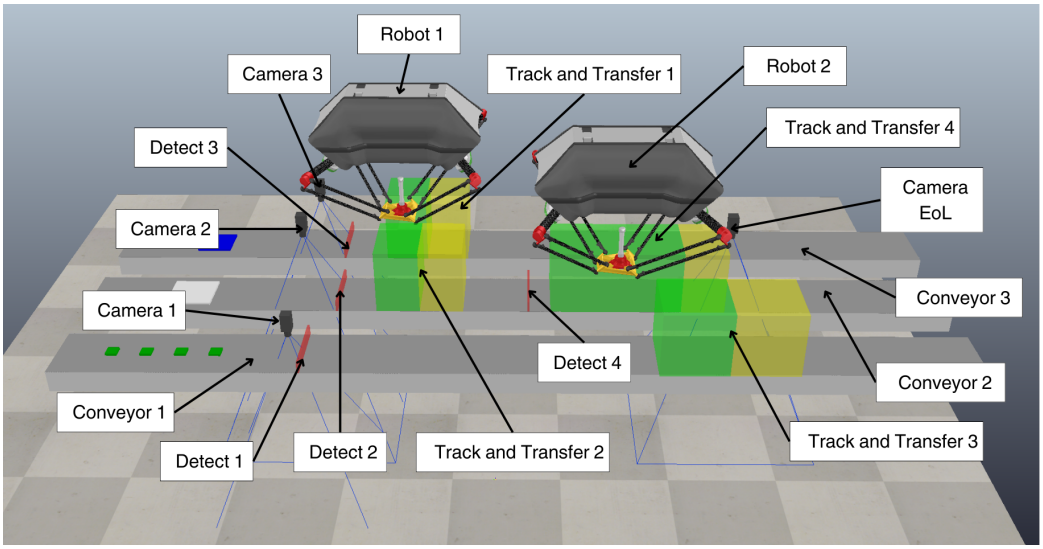


Figure 1.5 Simulation Key Objects

Table 1.1 Scene Objects and relative Tasks

Scene Object	Task
Camera 1	Measure each part dimension and enable detection
Camera 2	Measure each tray dimension and enable detection
Camera 3	Measure each insert and enable detection
Detect 1	Detect and label parts
Detect 2	Detect and label trays
Detect 3	Detect and label inserts
Detect 4	Detect and label tray with inserts
Track and Transfer 1	Track inserts for pickup and transfer to next
Track and Transfer 2	Track tray for placement and transfer to next
Track and Transfer 3	Track parts for pickup and transfer to next
Track and Transfer 4	Track tray with insert for placement and transfer to next
Robot 1	Performs pick and place - Robot moves, Suction build up, Robot moves, Suction off
Robot 2	Performs pick and place - Robot moves, Suction build up, Robot moves, Suction off
Camera End of Line	Check if the tray has an insert with all 4 parts - a Total Score will be computed, designating NOK or OK

Assumptions

The first assumption involves the part feeders. In the real-world, no part is equal, thus a certain size distribution has been put in place. Unfortunately, there is no library for normal distribution in Lua. A simple algorithm is used to solve that,

$$Z = \sqrt{-2ln(X)}cos(2\pi Y) \tag{1.1}$$

, with X and Y being independent random variables distributed over an interval $(0, 1)$. This algorithm 1.1 is called the Box-Müller algorithm [10]. To verify this algorithm, a simplified version of the simulation has been created. The results are presented in Figure 1.6. It clearly showcases how the algorithm provides a good normal distribution. This has been used more than once in the simulation, particularly for noise creation.

Following that assumption, the gripper of the robot has been adapted to feature actual states — ON/OFF. Considering a simple venturi suction gripper,

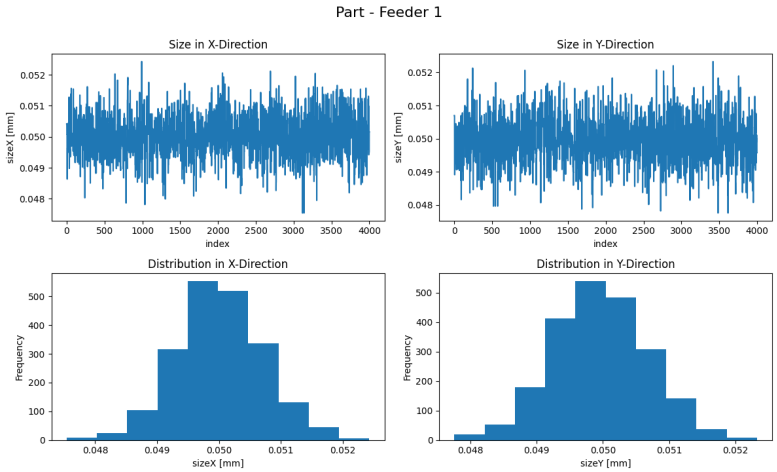


Figure 1.6 Part - Feeder 1: Normal Distribution

some pressurized air supply would be required. As in most production lines, we assume there would be a compressed air system, which continuously provides the same pressure. To operate the gripper, a pneumatic release valve would help create a negative pressure inside the suction cup, which will be used for lifting parts. It has been assumed that, the vacuum built-up takes roughly 0.10 seconds. To release the parts, the actuator would break up the vacuum. The benefits of vacuum grippers are their precision, stability, and durability [11]. To make it realistic, some normal distributed noise has been added. Thus, during the simulation, the robot assumes an inlet pressure fluctuating below the maximal availability of 100%. When the actuator switches on, the system creates a vacuum, and the losses in the system would create a pressure drop. The lower pressure, will be used as an indicator for the gripper's performance. No pressure, would mean not venturi effect, hence no suction force.

Furthermore, the robot, as defined by BWF requires a few parameters. The robots use the *Ruckig online trajectory generator* [12] to determine the path from point A to point B. Based on that trajectory, the inverse kinematics values for the joints are calculated. That trajectory depends on the start velocity, the end velocity, the maximal velocity, and the maximal acceleration. This method does not allow manipulating certain joint speeds. Because of that, a simple noise term was added to the maximal trajectory velocity to still create some variance in the values.

Lastly, similar to the robot velocity or the gripper air supply, some noise has been added to the conveyor belt speeds.

Interoperability

One could argue that this model does not reflect a very complex system or scenario. But it would be very simple to replace the camera with any other sensor type, e.g., inductive. Each individual feature can be programmed. The support of multiple languages and external applications make CoppeliaSim and the laid-out approach interoperable with other scenarios.

1.2.2 Control Logic

To have a working system, a certain control logic was put in place. First, the various feeders drop, according to a certain frequency, parts onto the conveyor belts. The moving conveyors transport the items through the assembly line. Each item will be scanned by the cameras, and the data will be stored. Depending on the sizes, detection will be enabled. When the parts continue through the detection barriers and detection for that part has been enabled, they get marked for the upcoming pick and place operations. This sequence of steps can be expressed by the algorithm 1. When an item has not been tagged, it will be ignored by the robots and continues to the end of the line.

Algorithm 1 Detection

Require: $min\ X, max\ X, min\ Y, max\ Y \geq 0$
$tagged\ parts \leftarrow \emptyset$
while $simulation$ is running **do**
  for $part$ in $detection\ window$ **do**
    if $camera$ is $enabled$ and $detection\ window$ is $enabled$ **then**
      $X \leftarrow camera\ size\ X$ ..... *Measured Dimensions by Camera*
      $Y \leftarrow camera\ size\ Y$
      if $X \in [min\ X, max\ X]$ and $Y \in [min\ Y, max\ Y]$ **then**
        $tagged\ parts \leftarrow part$
      end if
    end if
  end for
end while
return $tagged\ parts$

After being tagged, all the parts continue their journey down the assembly line, until they enter the various tracking and transfer windows. All tagged parts will be analyzed and identified. The velocity vectors and other relevant information for each part will be extracted. This is necessary for the following pick and place process. As long as the items remain in the tracking window, the associated robot

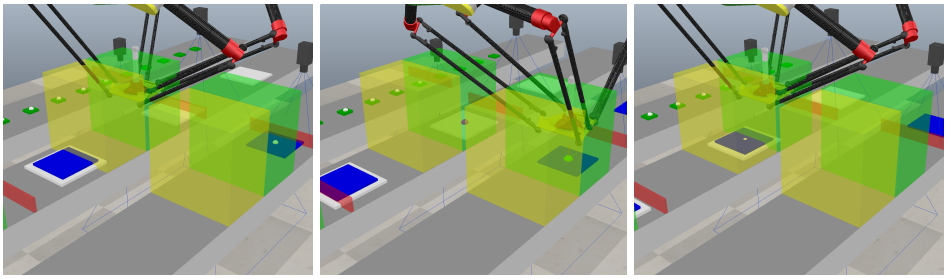
can grip it. Items, which unfortunately did not got picked in time, pass through the transfer window, where at the exit of it, their tag will be deleted. Algorithm 2 explains this logical structure for the transfer windows.

Every time a tagged part becomes available, the robot computes the joint velocities to reach its location. When the moving target is reached, the robot activates the gripper and waits until a vacuum has been established. Upon creating enough suction force, the item gets lifted. As soon as a destination object in the opposite tracking window has been identified, the robot computes new joint velocities for the journey towards the targeted location. Every motion of the robot, requires the switch from *non-threaded* to *threaded operation*, else the motion could be interrupted. Once the motion is complete, the robot's script switches back. The release of the items takes the same amount of time as the vacuum build up. Once the item get picked up, their tag will be deleted, as described in the algorithm 3. The robot can be configured to create a permanent bond between the attached parts.

Algorithm 2 Track and Transfer

Require: $tagged\ parts \neq \emptyset$

while $simulation$ is running **do**
  for $part$ in $tagged\ parts$ **do**
    if $part$ in $tracking\ window$ **then**
      $tracked\ parts \leftarrow part$
    else if $part$ in $transfer\ window$ **then**
      $transfer\ parts \leftarrow part$
    else if $part$ passed trough $transfer\ window$ **then** *. . . . Not gripped.*
      stop $tracking$
    end if
  end for
end while
return $tracked\ parts, transfer\ parts$



(a) (b) (c)

Figure 1.7 a) Detected Insert, b) Picking of Insert, c) Placed of Insert

Figure 1.7 shows what the detection, pick, and place process looks like for an insert. The detection barrier successfully detected the insert and the tray. Following that, the tracking window correctly marked the insert, and the robot perfectly approaches the part. The robot moved towards the target tray and dropped it close to the surface of the tray. While the robot is moving in the direction of the target location. There is an error because the tray continues moving, and the robot determined the trajectory based on its past location — the moment the insert was picked. To combat that, an offset to the robot's drop location has been added.

Key Functions

There are some key functions and modules used throughout the simulation. The most used functions were the *readCustomDataBlock* and *writeCustomDataBlock*, those are specifically used to assign data to the objects. Alongside them, *packTable* and *unpackTable* were used to pack and unpack the data.

Some image analysis functions from the *simVision* API[3] were used to transform the RBG images from the Vision sensors into measurements. The vision callback function[4] is called when a vision sensor is implemented and receives a new image. By using the *sensorImgToWorkImg*-function, the image gets first transformed into a work image. Then the image will be processed by the *edgeDetectionOnWorkIm*-function, which detects edges on work images. Lastly, the *blobDetectionOnWorkImg* -function is used to identify closed features on work images. It returns the number of blobs detected, the relative sizes, and more.

The detection, track, and transfer window feature proximity sensors[5], which are trigger by the sensing callback function. When any object is in the sensor's

[3]https://manual.coppeliarobotics.com/en/simVision.htm
[4]https://manual.coppeliarobotics.com/en/visionCallbackFunctions.htm
[5]https://manual.coppeliarobotics.com/en/proximitySensors.htm

vicinity, the callback function is called.

Lastly, the robots use the *simIK* API to create a kinematic chain of all links and joints. As mentioned, the *Ruckig tajectory generator is used* alongside the *moveToConfig*-function [6] from the regular API to generate the motion for the robot.

Algorithm 3 Pick and Place

Require: $tracked\ parts \neq \emptyset$

while $simulation$ is running **do**
  for $part$ in $tracked\ parts$ **do**
    $dest \leftarrow part\ destination$
    $picking \leftarrow start$
    move to $pick\ location$
    $picking \leftarrow done$ ..... *Gripper activation*
    stop $tracking$
    if $picking$ is $done$ **then**
      $drop \leftarrow drop\ location$
      if $drop$ is equal to des **then**
        $place \leftarrow start$
        move to $drop\ location$
        $place \leftarrow done$ ..... *Gripper release*
      end if
    end if
  end for
end while

1.3 Conclusion

In this chapter, the simulation used for the analysis of RCA methods has been explained. The advantages of using CoppeliaSim, a simulation tool, for real world scenarios has been outlined. The various elements of the model have been described. The overall structure follows the schema as presented in Figure 1.8. The control logic has been presented. The previous sections showcased that a model, in combination with the imposed assumptions, can be used to create a real-world equivalent of a physical manufacturing system.

[6] https://manual.coppeliarobotics.com/en/regularApi/simMoveToConfig.htm

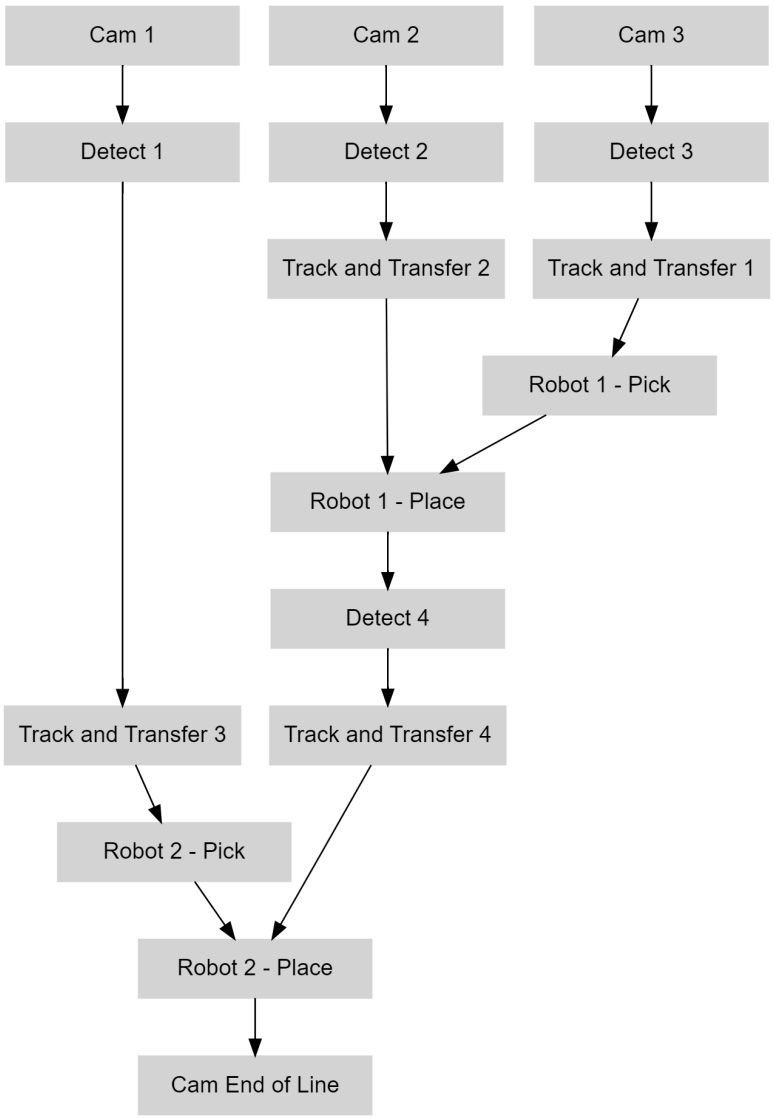


Figure 1.8 Process Flowchart of the Assembly Line

Chapter 2

Data Generation

This chapter will be used to outline the data generation process centred around the idea of gathering data from a simulated manufacturing line. First, the setup, implemented for data handling and collection, will be explained. Then, the various scenarios for the analysis will be presented.

2.1 Setup

2.1.1 Data handling

Considering a simulation model, we can distinguish between process and system data. The former correlates to data generated while the simulation is running. The latter is used to define the system. Hence, for the subsequent evaluation, mainly process data will be used. Thus, in the various object scripts, data management had to be considered.

As mentioned in the earlier chapter, one can use the *writeCustomDataBlock* to assign variables to the object and the *readCustomDataBlock* to access the data. This *read-write* behaviour is essential in CoppeliaSim. It is the only method to exchange information between objects. To access the information of another object, the object's handle is required, which is the unique identifier associated with the object.

Relevant information has to be selected. For instance, it is possible to save all data created by a manufacturing process. But this can create unnecessary resource expenses for processing and storage infrastructures. On the other side, selecting a subset of variables leads to bias, which reinforces hidden confounders. Thankfully, in a simulated environment, confounders are carefully controlled decisions. They only exist if the simulation was designed that way.

Making the transition from the real world into the virtual world effective

requires that the simulation match the physical model as much as possible. However, not every single aspect of a real-world machine can be expressed by data or computational models. So, the design process of the simulation requires simultaneous thinking about how a certain variable could be logged in the physical world and how the machine operates. Considering the data content of each variable, can be related to a certain sensor type. For example, joint speeds could be measured by a rotary sensor. Altogether, based on the chosen simulation, data originating from the camera, the conveyors, and the robots was deemed relevant, as presented in Table 2.1.

As mentioned, information transfer, and thus data exchange, only happens over this *read-write* process. Going back to the previous chapter implies that all the control aspects have been structured around that, as presented in Figure 2.1. The detection trigger variable is stored in the camera, and each time step the detection barrier reads the trigger and decides if it should trigger or not. Similarly, the information about the already detected and tagged parts is retained in the track and transfer object, which is read by the robot.

The data mentioned in the Table 2.1 only exists during a certain period in time. The objects only hold one timestamp of each variable. Each new time step, the data will be rewritten. This approach minimizes the computational load. Simulations, collecting data over longer periods of time, crash when storage runs out.

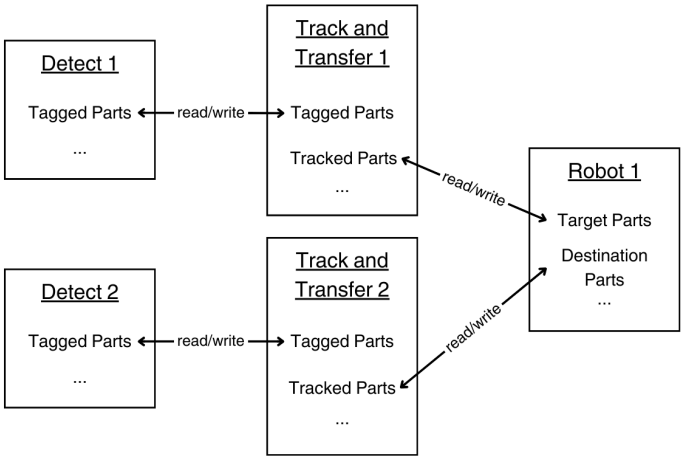


Figure 2.1 Data Exchange

Table 2.1 Data Sources and Variables

Object	Variables	Sensor Type	Description
Camera 1	Part Size, Part Position, Detection for Barrier 1	Vision Sensor	Camera 1 measures the size of the green parts and enables based on that the detection barrier.
Camera 2	Tray Size, Tray Position, Detection for Barrier 2	Vision Sensor	Camera 2 measures the size of the white tray and enables based on that the detection barrier.
Camera 3	Insert Size, Tray Position, Detection for Barrier 3	Vision Sensor	Camera 3 measures the size of the blue inserts and enables based on that the detection barrier.
Camera End of Line	Parts Sizes, Tray Size, Insert Size, Parts Positions, Tray Position, Insert Position	Vision Sensor	Camera End of Line measures all the items. All the measurements will be saved.
Robot 1	Robot Joint Velocities, Air Supply, Gripper Vacuum, Max Trajectory Velocity	Accelerometer, Rotary Sensor, Torque Sensor, Pressure Sensor	The joint speed of the robot will be saved. The provided compressed air supply and the resultant gripper vacuum pressure will be measured.
Robot 2	Robot Joint Velocities, Air Supply, Gripper Vacuum, Max Trajectory Velocity	Accelerometer, Rotary Sensor, Torque Sensor, Pressure Sensor	The joint speed of the robot will be saved. The provided compressed air supply and the resultant gripper vacuum pressure will be measured.
Conveyor 1	Conveyor Speed	Accelerometer, Rotary Sensor, Torque Sensor, Pressure Sensor	The conveyor speed will continuously be measured.
Conveyor 2	Conveyor Speed	Rotary Sensor, Torque Sensor	The conveyor speed will continuously be measured.
Conveyor 3	Conveyor Speed	Rotary Sensor, Torque Sensor	The conveyor speed will continuously be measured.

2.1.2 Data Collection

Due to the fact that the data only exits for one instance, expressed the need for a data collection system. A few options are available. First, it would have been possible to store the data using an array variable. The variable would have been used to collect the relevant data, and at the end of each simulation, the *afterSimulation* callback function, alongside CoppelaiSim's local file management, would have been used to write a simple log-file. This approach is very simple and works fine, but a real production system would quickly max out the computational capacities. A remote approach seems to be more aligned with a real-world application, and thus has been implemented in this case as well.

Enabling remote access, CoppeliaSim allows the implementation of a *ZeroMQ* remote client[1] in *Python*. *ZeroMQ*[2] is a messaging platform that enables the possibility to exchange information with other applications. A few communication strategies are possible, e.g., Client-Server and Publish-Subscribe.

The remote client is capable of accessing the information of the simulation, without any new manipulations of the simulation's programming. It can be operated either in asynchronous or synchronous mode. For collecting time-relevant data, synchronous or *stepping-Mode* needs to be activated. The remote API gives access to the regular API of CoppeliaSim. This makes it possible to remotely modify any parts of the simulation, which makes the introduction of interventions possible.

New data will be available each simulation time step. The client retrieves that information and processes it. Binary information, *True-False*, will be changed to $(1, 0)$, and *None* or *Nil* will become 0. The data will be filtered and renamed to better fit the subsequent evaluation. At the end of the process, the data will be saved as a CSV file. Table 2.2 and 2.3 point out how the simulation data has been renamed. The following figures, 2.2 to 2.6, illustrate the data collected during normal operation. To facilitate the data collection process, the remote-API has been used to start and stop the simulation. By disabling the rendering of the scene in CoppeliaSim, the computations can be accelerated.

[1] https://manual.coppeliarobotics.com/en/zmqRemoteApiOverview.htm
[2] https://zguide.zeromq.org/

Table 2.2 Data Variables

Object	Simulation Data	Analysis Data
Camera 1	• sizeX • sizeY	• cam_1_X • cam_1_Y
Camera 2	• sizeX • sizeY	• cam_2_X • cam_2_Y
Camera 3	• sizeX • sizeY	• cam_3_X • cam_3_Y
Camera End of Line	• part1sizeX • part1sizeY • part2sizeX • part2sizeY • part3sizeX • part3sizeY • part4sizeX • part4sizeY • tray1sizeX • tray1sizeY • tray2sizeX • tray2sizeY	• EoL_3_X • EoL_3_Y • EoL_4_X • EoL_4_Y • EoL_5_X • EoL_5_Y • EoL_6_X • EoL_6_Y • EoL_1_X • EoL_1_Y • EoL_2_X • EoL_2_Y
Robot 1	• jointVelo1 • jointVelo2 • jointVelo3 • jointVelo4 • maxVel • gripperSupply • gripperVacuum	• rob_1_1 • rob_1_2 • rob_1_3 • rob_1_4 • rob_1_maxVel • rob_1_supply • rob_1_vacuum

Table 2.3 Data Variables **continued**

Object	Simulation Data	Analysis Data
Robot 2	• jointVelo1 • jointVelo2 • jointVelo3 • jointVelo4 • maxVel • gripperSupply • gripperVacuum	• rob_2_1 • rob_2_2 • rob_2_3 • rob_2_4 • rob_2_maxVel • rob_2_supply • rob_2_vacuum
Conveyor 1	• speed	• con_1
Conveyor 2	• speed	• con_2
Conveyor 3	• speed	• con_3

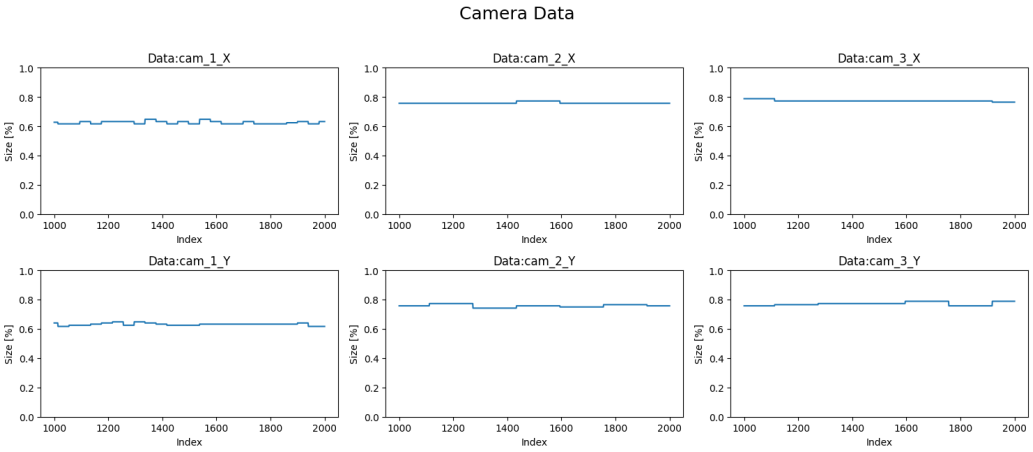


Figure 2.2 Camera Data Normal Scenario

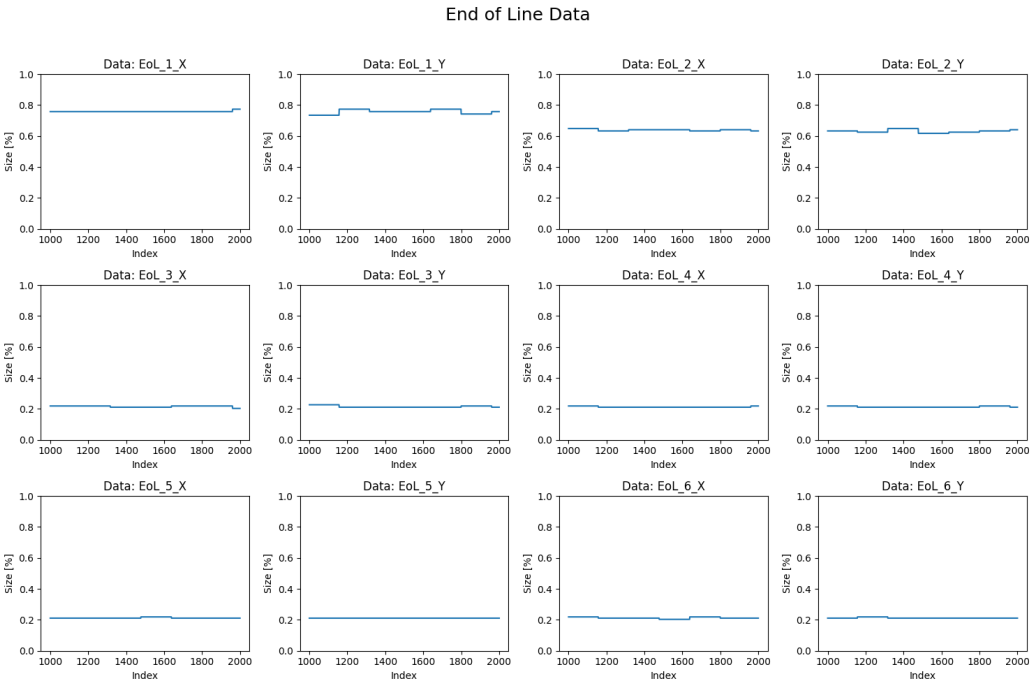


Figure 2.3 Camera End of Line Normal Scenario

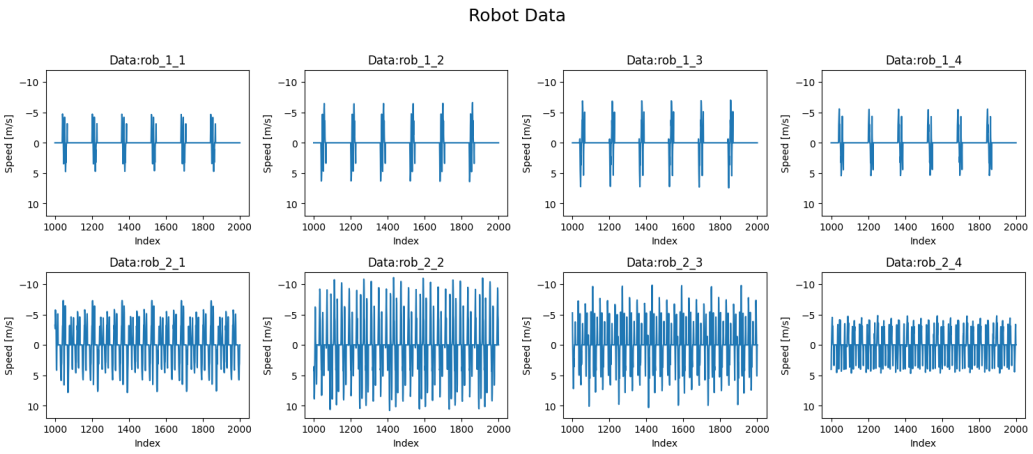


Figure 2.4 Robot 1 and 2 Normal Scenario

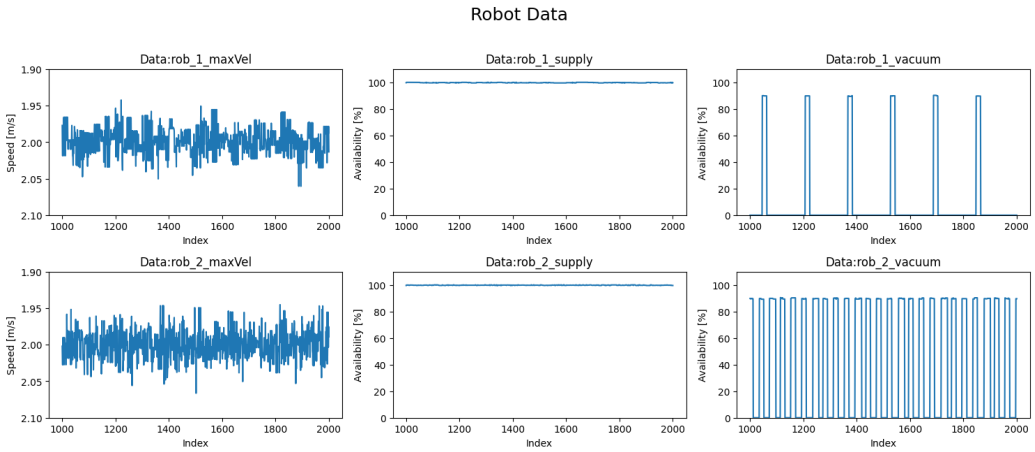


Figure 2.5 Robot 1 and 2 Normal Scenario **continued**

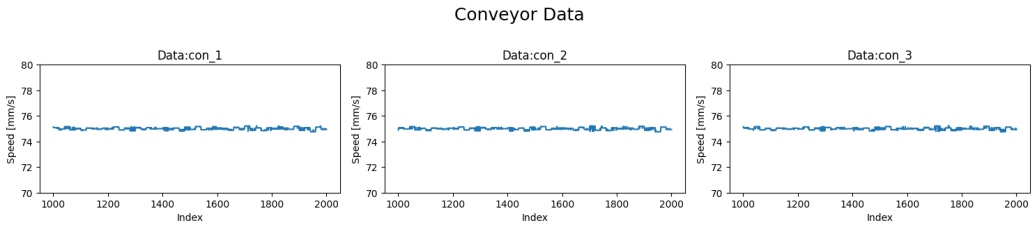


Figure 2.6 Conveyor Normal Scenario

2.2 Scenarios

To assess the data-driven RCA methods, two types of data are required. First, we require normal data, as presented in the figures 2.2 to 2.6, and abnormal data, meaning something has deviated from the normal operation. In the manufacturing sector, it is possible to classify numerous scenarios as abnormal. It always depends on the area of interest. In our case, we will focus on the end-of-line product. Any deviation from the typical product as presented in 1.8 would be classified as abnormal. The overall completion of the end product will be used to assess the product. 100% means every element is present, and any *Score* below that would be classified as abnormal.

Deviations in a simulated environment are the result of purpose-built functions. This action, of forcing a change in the model, is known as hard or soft interventions. To create a simulated failure, it is important to remember the logical structure of the model. Overall, it was possible to come up with some interventions, as summarized in Table 2.4. In the second column is a possible

scenario described that could have caused the failure. The third column lists the metrics, which will be affected first by the intervention. Those metrics will later become the root cause that the RCA algorithms should be trying to identify.

When opening the scene, a second window will open, as presented in Figure 2.7, which make it possible to intervene in the running simulation. Note that not all interventions from Table 2.4 are included.

To speed up the data generation process, the interventions were triggered through the remote client. In total, 64 interventional and 8 normal data sets with a duration of 100 seconds were generated. Four types of interventions were used: `interGripper1`, `interSize1`, `interFeeder3`, `interVelo2`. Each intervention has 16 datasets.

The various interventions were selected because of their position in the assembly line. Going through the structure (Figure 1.8), it is possible to identify some levels of interest. The first level will include the cameras, the detectors, and the conveyor belts. The next two levels will be track-transfer windows with their associated robots. The last level will be the end-of-line camera.

Table 2.4 Abnormal Scenarios

Intervention	Scenario	Metrics	Function
Gripper 1 Air Supply	Leak in the air supply line.	rob_1_supply	interGripper1
Gripper 2 Air Supply	Leak in the air supply line.	rob_2_supply	interGripper1
Max Velocity Robot 1	Allowable max velocity gets reduced, due to overheating.	rob_1_maxVel	interVeloRob1
Max Velocity Robot 2	Allowable max velocity gets reduced, due to overheating.	rob_3_maxVel	interVeloRob2
Deactivate Camera 1	Camera is broken or dirty.	cam_1_X, cam_1_Y	interCamera1
Deactivate Camera 2	Camera is broken.	cam_2_X, cam_2_Y	interCamera2
Deactivate Camera 3	Camera is dirty.	cam_3_X, cam_3_Y	interCamera3
Conveyor Speed 1	Friction increase.	con_1	interConveyor1
Conveyor Speed 2	Motor overheating.	con_2	interConveyor2
Conveyor Speed 3	Bearings broken.	con_3	interConveyor3
Part not available	Part stock is empty.	cam_1_X cam_1_Y	interFeeder1
Tray not available	Tray stock is empty.	cam_3_X, cam_3_Y	interFeeder2
Insert not available	Insert stock is empty.	cam_2_X, cam_2_Y	interFeeder3
Part Size NOK	Part size out of tolerance.	cam_1_X, cam_1_Y	interSize1
Tray Size NOK	Tray size out of tolerance.	cam_3_X, cam_3_Y	interSize2
Insert Size NOK	Insert size out of tolerance.	cam_2_X, cam_2_Y	interSize3

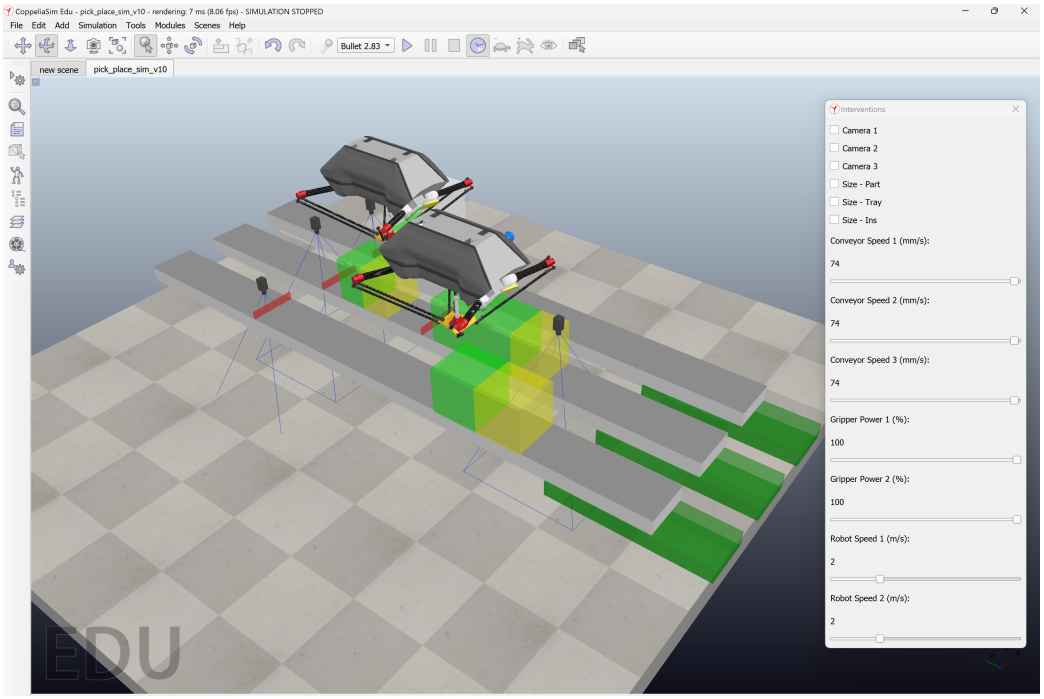


Figure 2.7 Simulation withcustom Intervention Window

2.3 Conclusion

The data generation and collection process using CoppeliaSim is very easy and does not require a lot of programming skills. However, being able to access the data only over the *read-write* process was a bit limiting. Using the remote client API was pretty straightforward. No port-address was needed. The client automatically linked to the open scene in CoppeliaSim and directly had access to the data available in the model. Furthermore, having the ability to use full-featured Python made it possible to use more advanced data processing and visualization methods. Generally, the data collection worked perfectly, but sometimes due to the stepping mode, the simulation got stuck, which led to the addition of a small wait time between interventions. Considering the various interventions, they are based on some potential real-world scenarios. This reverse thinking approach has been used throughout this work. The focus always lied behind the idea of how to simulate an actual production line and what the data could look like. Lastly, it was possible to put together a list of potential interventions, of which only four have been rigorously simulated. The raw data and the processed data has been saved accordingly.

References

[1] Hoda ElMaraghy et al. “Evolution and future of manufacturing systems”. In: *CIRP Annals* 70 (2 Jan. 2021), pp. 635–658. ISSN: 0007-8506. DOI: 10.1016/J.CIRP.2021.05.008.

[2] Sudip Phuyal, Diwakar Bista, and Rabindra Bista. “Challenges, Opportunities and Future Directions of Smart Manufacturing: A State of Art Review”. In: *Sustainable Futures* 2 (Jan. 2020), p. 100023. ISSN: 2666-1888. DOI: 10.1016/J.SFTR.2020.100023.

[3] Jiewu Leng et al. “Digital twins-based smart manufacturing system design in Industry 4.0: A review”. In: *Journal of Manufacturing Systems* 60 (July 2021), pp. 119–137. ISSN: 0278-6125. DOI: 10.1016/J.JMSY.2021.05.011.

[4] Dimitris Mourtzis. “Simulation in the design and operation of manufacturing systems: state of the art and new trends”. In: *International Journal of Production Research* 58 (7 Apr. 2020), pp. 1927–1949. ISSN: 1366588X. DOI: 10.1080/00207543.2019.1636321/ASSET/BE8A2F08-DE76-4F94-B511-B7E38DD3B68B/ASSETS/IMAGES/TPRS_A_1636321_F0011_OC.JPG. URL: https://www.tandfonline.com/action/journalInformation?journalCode=tprs20.

[5] E. Rohmer, S. P. N. Singh, and M. Freese. “CoppeliaSim (formerly V-REP): a Versatile and Scalable Robot Simulation Framework”. In: *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*. www.coppeliarobotics.com. 2013.

[6] Jaemin Yoon, Bukun Son, and Dongjun Lee. “Comparative Study of Physics Engines for Robot Simulation with Mechanical Interaction”. In: *Applied Sciences 2023, Vol. 13, Page 680* 13 (2 Jan. 2023), p. 680. ISSN: 2076-3417. DOI: 10.3390/APP13020680. URL: https://www.mdpi.com/2076-3417/13/2/680/htmhttps://www.mdpi.com/2076-3417/13/2/680.

[7] Josefine Older Steffensen. *Blue Workforce | A revolution in robotics | Danish Robot Technology*. URL: https://scanmagazine.co.uk/blue-workforce/.

[8] Coppelia Robotics. *Coppelia Robotics visiting BlueWorkforce*. July 2016. URL: https://www.youtube.com/watch?v=JxmNZSpmrk4.

[9] Eugene Demaitre. *OnRobot acquires Blue Workforce assets, robotics developers - The Robot Report*. Apr. 2019. URL: https://www.therobotreport.com/onrobot-acquires-blue-workforce-assets-robotics-developers/.

[10] Statistics How To. *Box Muller Transform: Simple Definition - Statistics How To*. URL: https://www.statisticshowto.com/box-muller-transform-simple-definition/.

[11] Eurotech. *How Does A Vacuum Gripper Work?* 2023. URL: https://eurotech-vacuum-technologies.com/how-does-a-vacuum-gripper-work/#:~:text=Vacuum%20grippers%20offer%20several%20advantages%20over%20traditional%20mechanical,ability%20to%20handle%20delicate%20objects%20without%20causing%20damage..

[12] Lars Berscheid and Torsten Kröger. "Jerk-limited Real-time Trajectory Generation with Arbitrary Target States". In: *Robotics: Science and Systems XVII* (2021). URL: https://github.com/pantor/ruckig#readme.