LUA PROGRAMMING IN HRC WORKSTATION DESIGN

Jiménez Sánchez, Juan Luis

KTH Industrial Engineering and Management

Master of Science Thesis MKN 2018
KTH Industrial Engineering and Management
Machine Design
SE-100 44 STOCKHOLM

Sammanfattning

Ett nära samarbete mellan mänskliga operatörer och industrirobotar är ett sätt att möta utmaningarna av ökad global konkurrens och demografiska förändringar för tillverkningsföretag i de utvecklade länderna. Dessa sammansättningssystem för humant industrirobotar (HRC) kombinerar mänsklig flexibilitet, intelligens och taktil känsla med robothastighet, uthållighet och repeterbarhet. Den nuvarande personliga säkerhetslagstiftningen begränsar emellertid de möjliga samarbetsansökningarna som kan genomföras i praktiken, men stora forskningsinsatser görs för att möjliggöra ett praktiskt genomförande av dessa framtida arbetsstationer.

När begränsningarna i säkerhetslagstiftningen tas upp, och samarbetssystemen kan genomföras, kommer ett behov att simulera dessa system stiga. Virtuella simuleringar är en viktig komponent i modern produktionssystemdesign och kommer att krävas i framtida montagearbetsstation design. En ny befintlig programvara är i utveckling som kan simulera, visualisera och utvärdera HRC-arbetsstationer. Det övergripande målet med simuleringsprogrammet är att designa "optimala" arbetsstationer, och de möjliggör utvärderingar av flera designalternativ för att nå denna "optimala". Skapandet av dessa designalternativ är utmanande idag eftersom det kräver mycket manuellt arbete. Syftet med denna avhandling är att ta itu med denna fråga genom att bidra till utvecklingen och förbättringen av simuleringsprogramvaran genom programmering av skript i Lua-språket. Skripten utvecklades genom en iterativ och trial-and-error-process, kombinerad med författarens förstahandsupplevelse i användningen av programvaran.

De resulterande skriptna möjliggör för användaren att utföra simuleringar på ett snabbt, effektivt, automatiserat och förenklat sätt jämfört med den traditionella metoden, vilket minskar behovet av manuellt arbete till ett minimum. En stor mängd simuleringar kan utföras på kort tid, även utan att det behövs mänsklig interaktion.

Dessutom, med resultaten av simuleringarna som bas, har matematiska optimeringstekniker använts för att hitta den optimala HRC-designen hos en fallstudiestation. Fallstudien har genomförts på en arbetsstation i en tung fordonstillverkare. Resultatet av ärendet framhäver de förbättringar som gjorts av programvaran av skripten och hur dessa kan användas för att effektivt utforma framtida HRC-arbetsstationer.

Abstract

One approach to address the future challenges of an increasingly global market's competition and the demographic changes of an aging workforce for manufacturing companies in developed countries is the close collaboration between human operators and industrial robots. These human-robot collaborative (HRC) assembly systems aim to combine the best characteristics of the human; its flexibility, intelligence and tactile sense, with robotic speed, endurance and repeatability. Although currently safety legislation limits the possible collaborative applications that could be implemented in practice, large research efforts are put in order to enable practical implementation of these future workstations.

Once these safety legislation limitations have been addressed, a need to simulate these systems before implementing them will arise. Virtual simulations are an important part of modern production system design and will be demanded in future assembly workstation design. A new existing software is in development that can simulate, visualise and evaluate HRC assembly workstations. The general goal with the simulation software is to design "optimal" workstations, and they enable evaluations of multiple design alternatives to reach this "optimum". The creation of these design alternatives is challenging today as it demands a lot of manual work. The aim of this thesis is to tackle this issue by contributing to the development and improvement of the simulation software through the programming of scripts in the Lua language. The scripts were developed through an iterative and trial-and-error process, combined with first-hand experience of the author in the usage of the software.

The resulting scripts enable the user to perform simulations in a swift, efficient, automated and simplified way in comparison to the traditional method, reducing the need of manual work to a minimum. A large amount of simulations can be performed in a short amount of time, even without the need of human interaction.

In addition, with the results of the simulations as a base, mathematical optimisation techniques have been employed in order to find the optimal HRC design of a case study station. The case study has been conducted at a workstation in a heavy vehicle manufacturer. The results of the case highlight the improvements made to the software by the scripts and how these can be used to efficiently design the HRC workstations of the future.

FOREWORD

This chapter includes a Foreword by the author to acknowledge collaborators, family and friends.

Despite being the sole author of this thesis, I would never have achieved it without the help and collaboration of a large group of people, whom I would like to thank here:

My supervisors, Fredrik Ore and Professor Xi (Vincent) Wang have had a huge impact on the results of my thesis and have greatly supported me with their counsel.

My colleagues at KTH, and my Scania colleagues, the whole division of TE (Global Industrial Development) in general, but also my TEID thesis group friends: Jim, Syed, Vijay, Vipasha and Xiaomeng.

The FCC, specifically Domenico, Johan, Niclas and Peter, for their close and fruitful collaboration in the development of my requests.

And last, but not least, my family and friends. Being abroad for my master studies in a foreign country can be hard for the closest ones, so thanks to my parents and my sister for their support and encouragement during these two years.

Juan Luis Jiménez Sánchez

Stockholm, June 2018

NOMENCLATURE

This chapter presents the abbreviations that are frequently utilized in the report.

Abbreviations

IPS	Industrial Path Solutions
HRC	Human Robot Collaboration
HRI	Human Robot Interaction
IMMA	Intelligently Moving Manikins
IRB	Industrial Robot
FCC	Fraunhofer-Chalmers Research Centre
API	Application Programming Interface
SAM	Standard Allowed Minute
KTH	Kungliga Tekniska Högskolan
CAD	Computer Aided Design
TCP	Tool Center Point
RULA	Rapid Upper Limb Assessment
KPI	Key Performance Indicator

TABLE OF CONTENTS

1 INTRODUCTION

This introductory chapter describes the background of the project, its purpose and research questions, and its delimitations.

1.1 Background

Simulation software are used in manufacturing companies early in production development processes to shorten development time, increase quality and reduce costs [1]. These tools are used to support decision making in the companies and are an integral part of the engineering activities in many manufacturing companies [2].

However, in design of Human-Robot Collaborative (HRC) workstations a commercial software has not yet been widely developed. There is a new software under development that enables simulation of collaborative tasks between human and industrial robots (Industrial Path Solutions, IPS) ([3], [4], [5]). IPS demands manual inputs to create the workstation and to perform the task allocation in an operation sequence. The software outputs are possible design alternatives and also quantitative numbers on operation time and biomechanical load for each of these alternatives.

But a general goal with the simulation software is to design “optimal” workstations, and they enable evaluations of multiple design alternatives to reach this “optimum”. The creation of these design alternatives is challenging today as it demands a lot of manual work. To meet this has a Lua API been developed in IPS in order to automate simulations through Lua programming. Through well-defined objectives and limits on the design variables, the goal to automate the simulation task and extract quantitative numbers on all the multiple layout alternatives [6], which can then be compared to each other in order to find the “optimal” solution depending on the designer’s criteria.

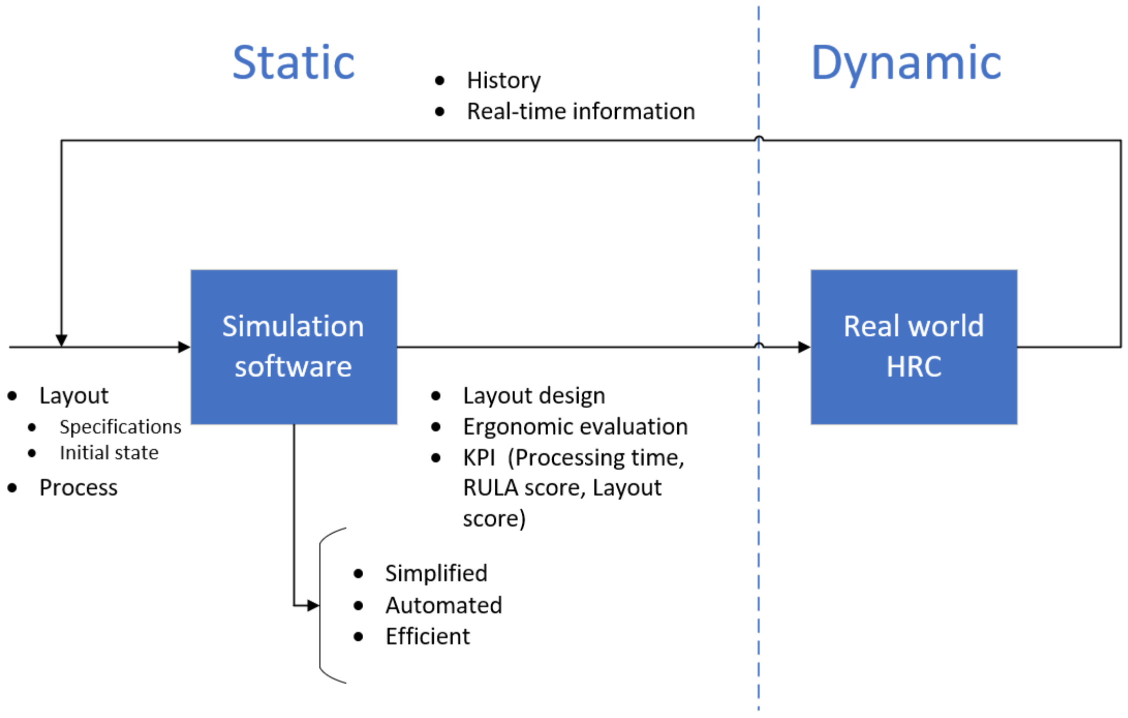


Figure 1. Static-Dynamic HRC diagram

Figure 1 summarizes the thesis environment and its relation to HRC. The thesis focuses on the "static" aspect of the design, the Simulation software. The software receives as main inputs the specifications and initial state of the layout, as well as the current process information (operation sequence). The author has performed improvements on the software so as to make it more simplified, automated and efficient, enabling a large number of simulations to be performed in a short time.

The outputs of the program are a layout design and an ergonomic evaluation, as well as quantitative values (KPIs) for the processing time and ergonomic assessment. A third KPI, the layout score, will be incorporated by the author of this thesis based on Muther's method for Systematic Layout Planning [7]. These outputs are then used in the "dynamic" environment of HRC, the real-world stations, to optimize them.

The feedback information of the real-world workstation, i.e. real-time information and histogram of events, could be used as a further input into the simulation software, as will be discussed in the future work (Chapter 5.3).

1.2 Purpose

The objective of this thesis is to improve the demonstrator software IPS for efficient simulation, visualisation, evaluation and optimisation of human robot collaboration (HRC) workstations in a heavy vehicle assembly environment. To do so, a Lua API embedded in the software will be utilized to develop a series of scripts, which will enable a user to perform several simulations in a simple and automated way, saving time and resources.

Once these scripts have been developed, a specific case study will be used to design an HRC workstation in an industry heavy vehicle manufacturer. With the quantitative outputs generated by the software in the simulations (the KPIs), the possible solutions may be compared to each other in an objective way in order to find an "optimal" solution design.

This objective is met through addressing the following research questions:

RQ1: *How can simulation software for design of human–industrial robot collaboration workstations be improved through programming of Lua scripts?*

RQ2: *How can a human–industrial robot collaborative workstation be optimised in an efficient way using the improved software?*

1.3 Delimitations

The case simulated in this thesis is from a single heavy vehicle manufacturer. The main purpose of using the case is not to design the best HRC systems but to test the developed scripts and highlight the improvements made to the demonstrator software; the single case company used does not affect the end result to any large extent.

The software utilized during the thesis (IPS HRC) is still in development, which involves the existence of a few errors and lack of functionalities that limited the thesis author's possibilities during the programming of the scripts.

2 FRAME OF REFERENCE

The reference frame is a summary of the existing knowledge and former research performed on the subject. This chapter presents the theoretical reference frame from the literature and state of the art study that is necessary to understand the thesis development.

2.1 Human-Robot Collaboration

The demographic changes of an increase in average age of the available workforce has to be addressed by adapting workstations to meet the new needs of the elder, since the increase in age increases the risk for musculoskeletal disorders ([8], [9]). In addition to improving the ergonomics of the workers ([10], [11]), the main reason to introduce robots in industry workstations is to increase productivity, reducing production times ([12]).

The vision of closer collaboration between human and robots was already expressed by Tan et al. ([13]): "Human-robot collaboration (HRC) is a dream combination of human flexibility and machine efficiency". A more recent definition of HRC ([14]) establishes how, to be considered a collaborative station, both the human and the robot have to simultaneously work with the same product.

In an HRC station the desired robotic features are handling speed, endurance and repeatability; whereas from the human, the preferred characteristics are flexibility, intelligence and tactile sense are desired ([15], [16]).

One other current development in order to meet increased global competition is to focus on virtual simulations of products and production processes in the manufacturing industry ([17]) in order to provide a design method for these future HRC workstations.

Human-Robot Collaboration is a subset of all research and applications inside Human-Robot Interaction. HRI includes a combination of a number of research areas such as cognition, linguistics and physiology research combined with engineering, mathematics, computer science and human factors ([18]). Walther and Guhl [19] present a classification of HRI that helps to describe the vide variety of human–robot systems in a structured way.

Operation modes in human and industrial robot collaboration are, according to the ISO standard ISO 10218 [20] divided into four modes: safety-rated monitored stop, hand guiding, speed and separation monitoring, and power- and force-limiting. These are described ([20], [21], [22]) in the following way: "Safety-rated monitored stop"; in this mode, when an operator enters the robotic work area, the robot stops and will automatically resume its actions when the human leaves the area. "Hand guiding" mode enables the human to control the robotic end-effector through designated controls while standing in the robotic work area and moving the end-effector to a designated position. When the human leaves the area, the robot starts its operation from that new position. "Speed and separation monitoring" enables the human to be present in the robotic work area while the robot is in operation. The distance between the human and the robot is constantly measured and when predefined thresholds are passed, the robot either slows down, stops or moves backwards from the human, all depending on the programmed responses. A "power- and force-limiting" system includes a weak and slow robot (compared to the standard industry robot) that is designed so as not to hurt humans in case of a collision.

Even though the collaborative modes are defined in the current robotic standard, the possibilities to build these HRC systems in industry are limited. Personal safety legislations in manufacturing industries are governed by the machine directive, which refers to standards in order to meet safety demands. ISO 10218 [20] regulates robots and robot system safety. This standard requires that some kind of fence (either physical or sensors acting as a fence) surrounds a traditional industrialised robot [23]. In HRC systems the robot is still considered dangerous, so in order to guarantee the safety of the human operator other systems than fences have to be used, since fences would impair collaboration.; great research efforts are being made in this field. Current state of the art includes multiple depth cameras supervising the HRC area ([24], [25]); robotic control systems having control of robot positions and movements [26], certified sensors assisting the depth cameras [25] and a network connecting all these systems into the goal of "a safe network of unsafe devices" [24].

This state of the art is constantly under development in order to enable use of HRC systems in manufacturing industries. Today there are actually fenceless industrial robots introduced in production environments. They are power- and force-limiting systems with small robots that have been installed without fences within the current machine directive. This is possible when the mandatory risk analysis shows that the risk for a human to work next to these robots is low, as discussed in [27]. These robots are designed to be weak, move with slow speeds, lack sharp edges and allow fenceless installation.

Once the limitations of safety legislation have been addressed and the collaborative systems can be implemented, a need to design simulate these systems will arise. HRC design methods presented in research publications are mainly limited to the work task allocation problem, i.e., which resource is most suitable to perform a certain task: the human or the industrial robot? Pini et al. [28] also base their design approach in the engineering design framework presented by Pahl and Beitz [29]. Chen et al. [30] present a method to use multi-objective optimisation techniques to choose a suitable task allocation based on assembly time and economic cost. Tsarouchi et al. have a similar approach in their task allocation method [31]. All these methods use time and cost as evaluation criteria, but none of them describes how to gather data into the selection process. One approach is to measure these before the task allocation can begin. However, in early phases of production design it is difficult to achieve these data since no physical workstation exists. This highlights the need of simulation software in order to make accurate production investment decisions early in the production development process.

In order to meet this need, a demonstrator simulation software is currently being developed (Industrial Path Solutions, IPS), making it possible to design and evaluate HIRC workstation layouts early in the production development phases so as to gather the desired data [32].

2.2 Industrial Path Solutions

Industrial Path Solutions (IPS) is a math-based software tool for automatic verification of assembly feasibility, design of flexible components, motion planning and optimization of multi-robot stations, and simulation of key surface treatment processes. IPS is developed by Fraunhofer-Chalmers Centre and Fraunhofer ITWM, and distributed by IPS AB and fleXstructures GmbH [33] [34].

For this thesis, a research version of the software in the field of Human-Robot Collaboration has been used (IPS HRC [3]), which enables simulation of hand-guiding HRC tasks in the environment. It can be used to analyse reachability for both industrial robots and humans, present layout alternatives and be a tool for risk assessment in HRC workstation design assignments. The software generates quantitative outputs considering operation time and biomechanical load assessments of the HRC workstation. These quantitative outputs can be used to compare alternative solutions in an objective way [4].

The program has several modules available, and a combination of the following three was used in the course of this project:

- IPS Rigid Body Path Planner: the IPS Path Planner lets simulation engineers import a scene geometry from any CAD system, as a VRML or JT file. Any object in the scene can be set as a so-called planning object, which IPS will find an efficient path for, provided that the object can be freely assembled along a path. The calculations done by IPS save the engineer a substantial amount of time, which otherwise would have to be put into manual planning of a collision free assembly path [35].

- IPS Robot Optimization: this module enables the user to define IRBs and their TCPs, automatically generate a robot path planning and tasks, and optimize robotic operations [36].
- IPS IMMA (Intelligently Moving Manikins): In order to analyse and control biomechanical motions performed by humans during assembly of e.g. cars, virtual modelling of mannequins is of great interest for the manufacturing industry [5]. This is addressed in IMMA by development of a computer environment where analyses of motions can be performed through simulation of manikins already in the production development phase. Such analyses minimize the risk of potential body joint and muscle problems for assembly personnel. Also, manikin analyses will help ensuring that the assembly motions are collision free both for the human and the object to be assembled. This type of computer analysis contributes to a more effective assembly process with a reduced number of injuries and a higher level of quality [5].

The primary interface of IPS is shown in Figure 2. It is composed of a 3-D visualization window which displays the current scene, as well as several menus on the top side, a log on the bottom part, and the Scene and Process trees on the left.

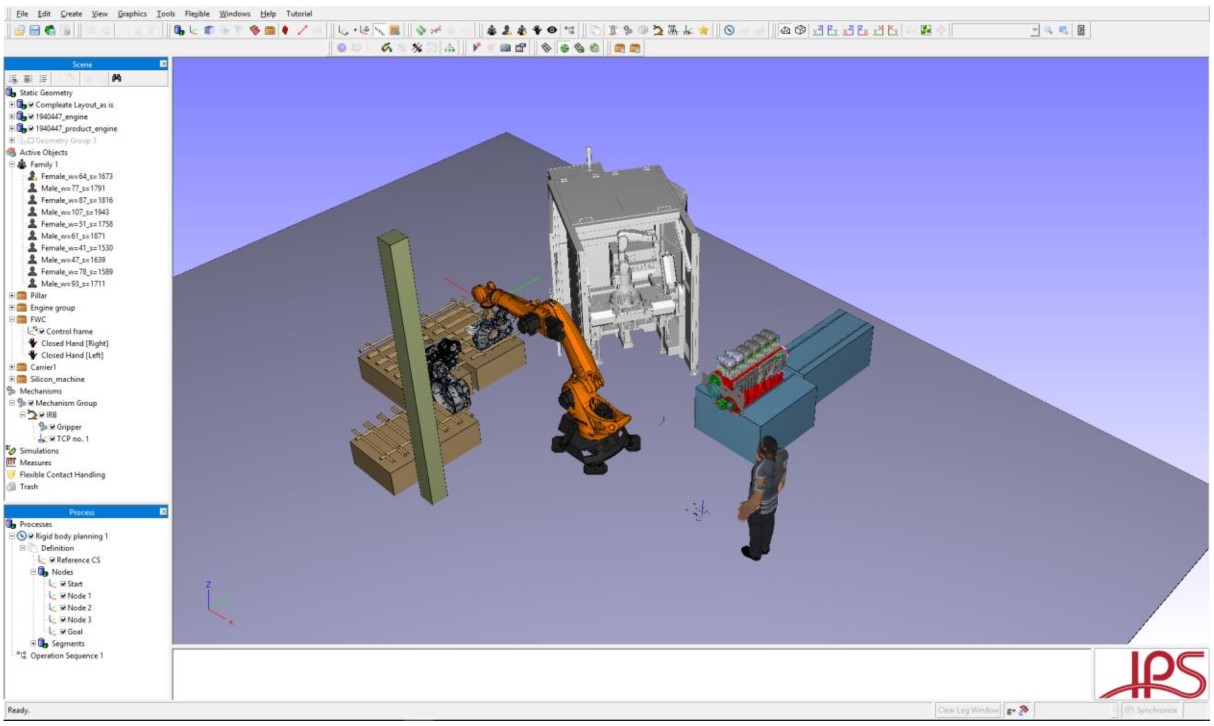
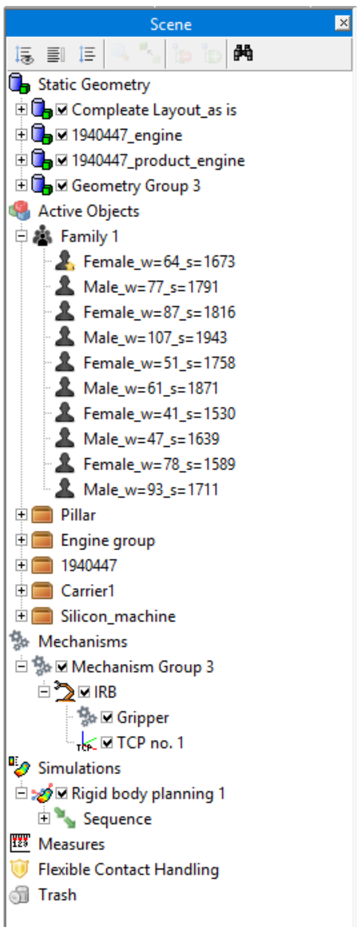


Figure 2. IPS HRC Interface

The Scene tree contains all the elements and information about geometries in the scene, objects, mechanisms, simulations, and others. For this thesis the following scene tree elements are significative:

- Static geometries: objects of the scenery which have collision detection but that cannot be utilized to generate path planning, nor be added to operation sequences, nor be interacted with by manikins or robots.
- Active objects: includes rigid body objects and manikin families.
    - Rigid body objects: transformed from static geometries, rigid body objects can be added as actors to an operation sequence, can be used to generate motions in the Rigid body path planner, and can be interacted with by the manikins and robots.

    - Manikin families: they are created within the software by the IMMA module with the information of an anthropometric database. A manikin family can consist of one or several manikins, and they can be added into an operation sequence to perform actions such as grasping objects, moving to positions, and assuming postures. The manikin in IMMA is built on a skeleton that consists of 81 segments connected by 74 joints resulting in 162 degrees of freedom.
- Mechanisms: include robot-related objects, such as the IRB itself, its defined TCP, and any grippers attached to it.
- Simulations: here are displayed the paths generated by the path planners that are saved to the scene. They can be added into operation sequences as motions for other objects to follow.



```
Scene
Static Geometry
  Compleate Layout_as is
  1940447_engine
  1940447_product_engine
  Geometry Group 3
Active Objects
  Family 1
    Female_w=64_s=1673
    Male_w=77_s=1791
    Female_w=87_s=1816
    Male_w=107_s=1943
    Female_w=51_s=1758
    Male_w=61_s=1871
    Female_w=41_s=1530
    Male_w=47_s=1639
    Female_w=78_s=1589
    Male_w=93_s=1711
  Pillar
  Engine group
  1940447
  Carrier1
  Silicon_machine
Mechanisms
  Mechanism Group 3
    IRB
      Gripper
      TCP no. 1
Simulations
  Rigid body planning 1
    Sequence
Measures
Flexible Contact Handling
Trash
```

Figure 3. IPS HRC Scene tree

The Process tree contains the elements and information related to path planning and operation sequences:

- Path planning: although the installed modules offer the possibilities of utilizing both a Rigid Body Path Planning and an IRB task planner for generating motions, for this thesis only the rigid body planning has been used, since the limitations of the existing IPS Lua API made it not possible to create a fully automated script with the IRB task planner.
    - Rigid Body Path Planning: offers the possibility of selecting a rigid body to perform a sequence of motions by moving between viapoints defined by the user. The planner automatically calculates a collision-free path for the body to travel from one viapoint to the next one until a goal position is reached. After a path has been calculated, it can be saved into the scene tree as a Simulation, which can then be incorporated into the operation sequence as motions.

- Operation sequence: an example of an operation sequence can be seen in Figure 4. An operation sequence can be defined by adding actors (e.g., a manikin family, an IRB, a rigid body) that perform a series of actions (e.g., grasp an object, move to a position, follow a motion path) in a sequential manner, which precedence orders that can be established between the actions of different actors. The duration of the actions performed by the IRB and the rigid bodies is based on the speed that is set for each of them, whereas the duration of manikin actions can be fixed by the user or automatically generated by a predetermined time standard method, SAM [37], included in IMMA.

The operation sequence is the ultimate goal of the scene, since by executing it the simulation is performed, generating a visual representation of the actions in the sequence. The generated replay contains information about the processing time for the whole sequence, and enables the user to perform an Ergonomic evaluation, which generates a series of .csv files (one for each manikin) for that particular simulation and stores them in a folder with timestamps. These .csv files contain information about all the joint position values of that specific manikin in each frame of the simulation. With this information, an ergonomic analysis method, such as RULA (Rapid Upper Limb Assessment) [38] can be used to generate the biomechanical load assessment.

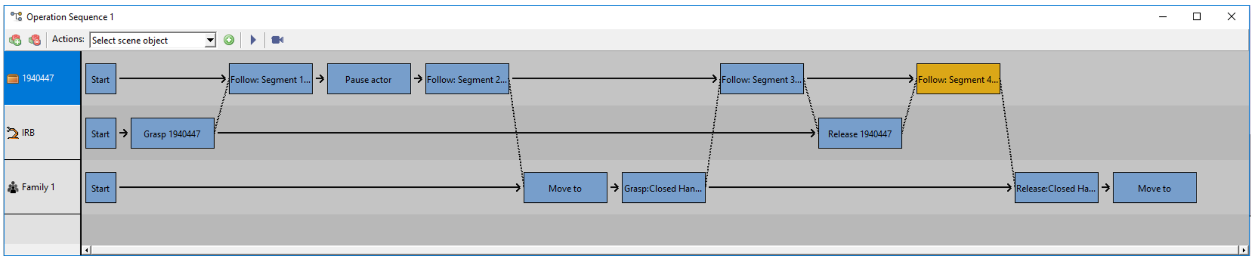


Figure 4. IPS HRC Operation Sequence

2.3 LUA

Lua is a free open-source, efficient, lightweight, embeddable scripting language. It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description [39] [40]

Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed, runs by interpreting bytecode with a register-based virtual machine, and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping [41].

Lua has been used in many industrial applications (e.g., Adobe's Photoshop Lightroom), with an emphasis on embedded systems. Several versions of Lua have been released and used in real applications since its creation in 1993. [42]

Lua is a fast language engine with small footprint that can be easily embedded into an application. Lua has a simple and well documented API that allows strong integration with code written in other languages. It is easy to extend Lua with libraries written in other languages and to extend programs written in other languages with Lua.

The connection between IPS and Lua is made through an API provided by the developers of the software. This Lua API contains the information for all the variables and functionalities that can be executed through the script inside IPS. Any text editor can be used to write the Lua code and save it as a .lua file, which can then be opened in IPS to run the script.

2.4 Case study

In order to test the effectiveness of the improvements in the software and the feasibility to use the scripts for HRC workstation design, a case study at the heavy vehicle manufacturer was proposed.

The flywheel cover assembly workstation at the engine assembly factory had prior to this project been identified as a potential case to be used in another research project [4]. It was of interest to simulate an HRC in that station since there were potential ergonomic difficulties when new and heavier components were to be introduced at the same time as the station processing time needed to be reduced.

In the current workstation, a worker picks up the flywheel cover from a carrier with the help of a hanging crane hook; then, the worker moves the cover into the silicon applying machine, where silicon is applied to the product. After this operation is finished, the worker picks the cover again and places it in the engine, where it is screwed in position by another worker; once the cover has been fixed to the engine, it is transported away, and a new engine arrives to the station, starting the process again.

The postures and forces that the worker in charge of retrieving the flywheel cover has to execute can lead to ergonomic issues, such as back injuries; thus, the aim of incorporating HRC in this station would be to install a robot that would be in charge of retrieving and moving the cover up to the step in which it has to be assembled into the engine, since this step requires the flexibility of the human to accurately place the cover in the correct position in the engine, and this would represent the point of collaboration. Thus, the value-adding tasks would be performed by the human and the non-value-adding tasks by the robot, as described in [43].

The assembly station was reproduced into the IPS software by importing the static geometries of the different objects that are part of the current station and that were previously modelled in a standard CAD tool (in this case, CATIA V5). Of these static geometries, five were defined as active objects: one of the carriers, the flywheel cover in the same carrier, the silicon machine, the engine, and a pillar (Figure 6).

Next, a Robot model was imported to be placed into the station as an IRB: the KUKA KR 210 R2700 PRIME [44]; this model had been already utilized in previous works [4] and fulfilled the requirements of reach and strength. In addition, a simplified gripper model was attached to the robot to represent a real-life gripper that would enable it to pick up and move the flywheel cover.

Finally, after all the geometries had been placed into the program, a family of manikins was imported to the scene; a total of 10 manikins (5 males and 5 females) of different height and weight given by the Swedish anthropometric database [45], with a confidence level of 95%. The resulting scene can be seen in Figure 5. A more detailed picture of the scene from the top view can be seen in Figure 6, including distances between objects and significant positions.

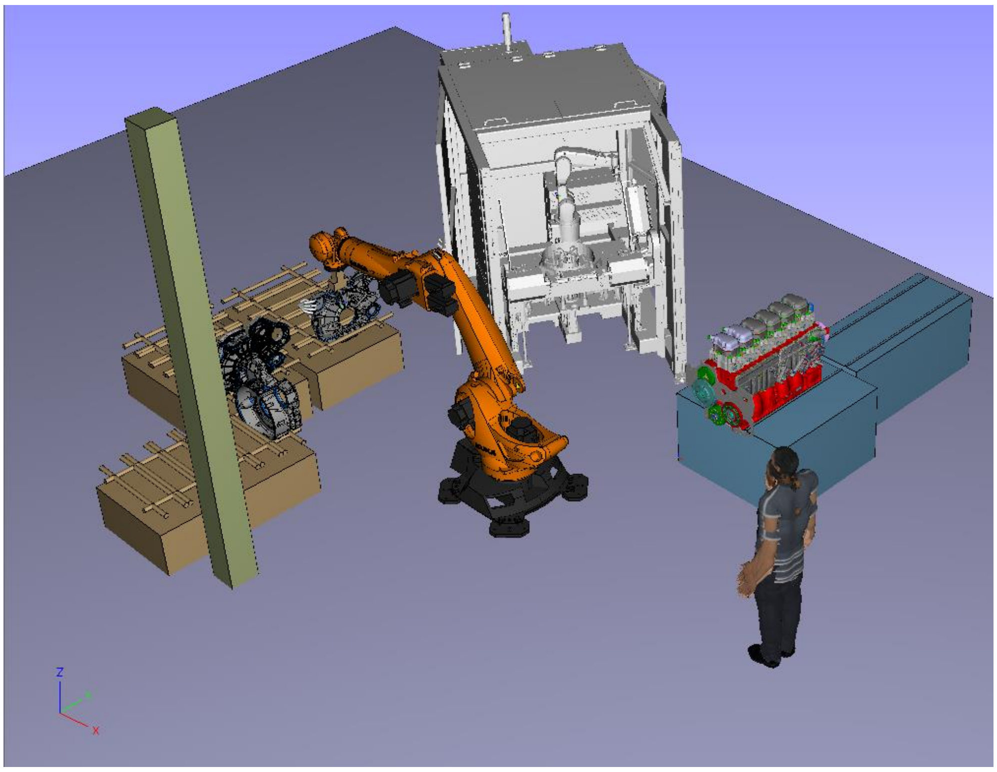


Figure 5. Case Study workstation in IPS

Once all the active objects, the IRB and the family of manikins have been placed in the scene, the logic of the path planning and the operation sequence was defined. A Rigid Body Path Planning was utilized instead of a robot planning; thus, the flywheel cover will be defined as the moving object with several viapoints and the IRB TCP will grasp it and follow its motions. This is due to the limitations of the Lua API, which does not contemplate the possibility of managing robot motions in the Operation Sequence through the scripts, and, since the aim of the case study is to test the developed scripts, those motions could not be used. Furthermore, in the current version of the software the IRB does not have collision detection, so in order to prevent the cover object from going through the robot during path planning a transparent “robot box” was placed in the position of the IRB, with the same dimensions as its base.

The starting position (viapoint 0) for the flywheel cover in the Rigid Body Path Planning would be in the carrier, with the IRB gripping it. Then, the cover would move to a point defined inside the silicon machine (viapoint 1); since the applying time for the silicon is the same for every product, it is not necessary to introduce it into the sequence, and thus after arriving into the machine the cover proceeds immediately to the next point, which is the Hand-over position (viapoint 2). Here, the family of manikins grabs the cover by two gripper points, placed by the author of the thesis to represent a comfortable posture for grabbing the object. After this, the manikin family proceeds to hand-guide the cover and the robot to a close position in front of the engine (viapoint 3), where the robot releases it and the human places the cover in the correct position (viapoint 4), ending the simulation. A fixed speed of 250 mm/s was set for calculating the duration of the product movements, whereas for the manikin operations the generated SAM [37] times were used.

The processing time of the simulation and the RULA ergonomic assessment score will be utilized as KPIs to redesign the layout and optimize the workstation, as is further discussed in point 3.1.

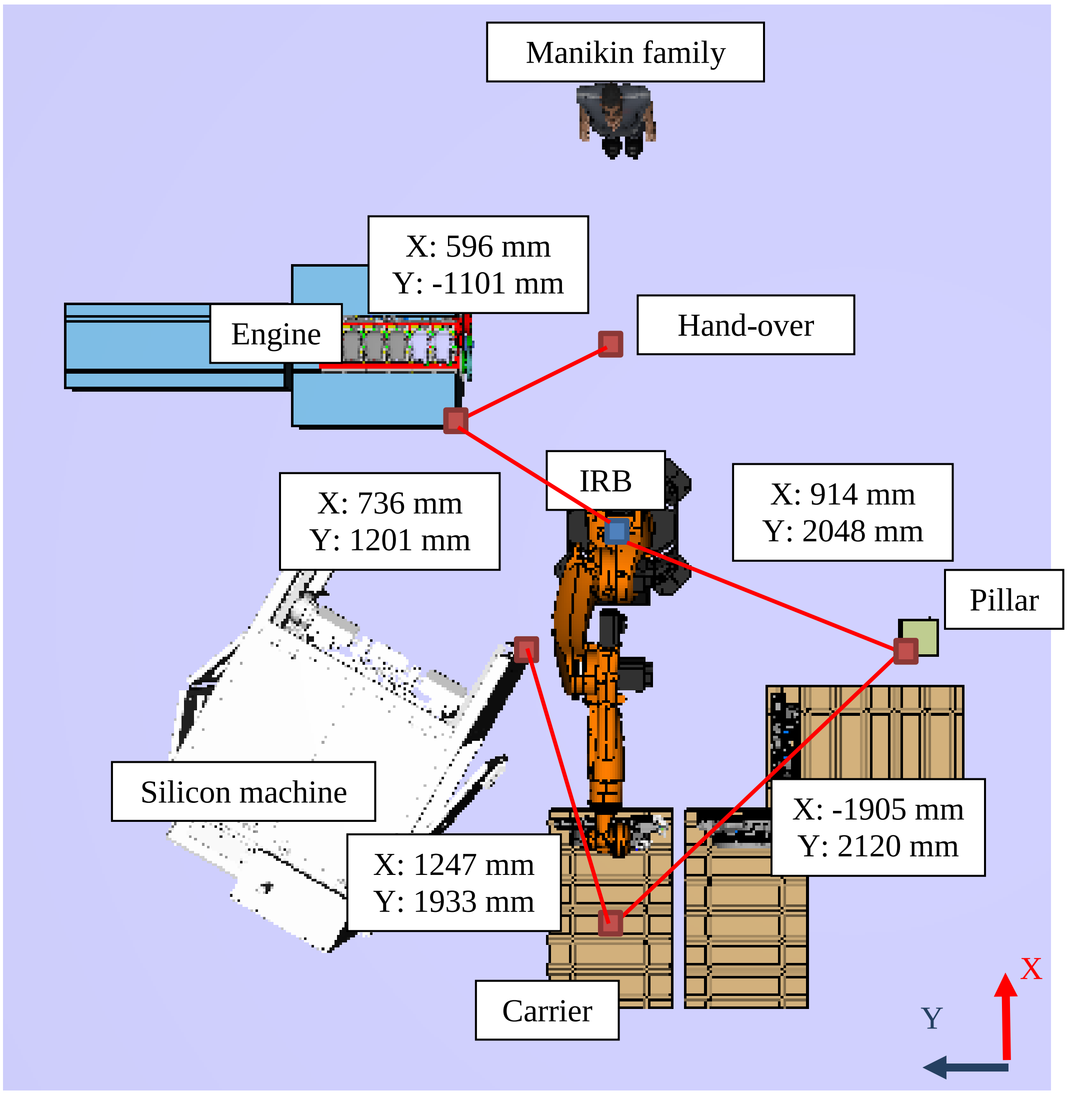


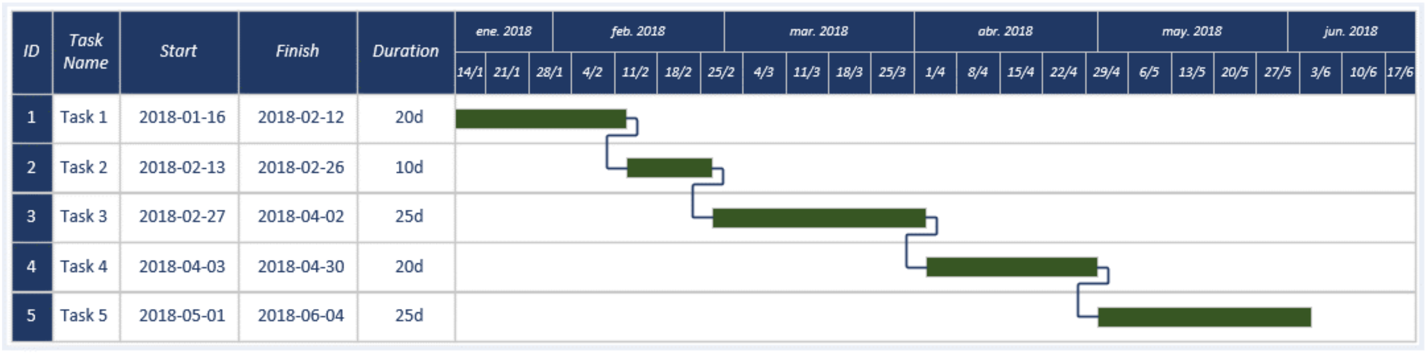
Figure 6. Top view of the workstation in IPS

In this chapter the methodology used in the thesis is described. The method is the structured process through which the author of the thesis reached the goals for the project.

The research presented includes software development in the growing human–industrial robot collaboration area. The design science research (DSR) concept is used as a methodological approach since it describes how to perform, evaluate and present design science research in a clear manner ([46], [47]).

The thesis project was divided into five tasks with defined timelines and milestones, which the author followed to reach the goals of the project. Moreover, a literature search has been performed continuously during the project in the area of human robot collaboration focusing on simulation of such systems. The aim of this search was to gather basic knowledge of the state of the art of human robot collaboration and simulation of such collaboration. The library database at KTH was used, as well as the previous work of the thesis supervisor in the field; the search method used was a systematic search with the following search terms: “HRC”, “Robot Human Collaboration”, “Workstation design”, “Automated workstation design”, “Workstation optimisation”. In the articles found a chain search was also made in order to find other interesting literature in order to make the review more comprehensive.

A Gantt diagram of the tasks is presented in Figure 7.



ID	Task Name	Start	Finish	Duration
1	Task 1	2018-01-16	2018-02-12	20d
2	Task 2	2018-02-13	2018-02-26	10d
3	Task 3	2018-02-27	2018-04-02	25d
4	Task 4	2018-04-03	2018-04-30	20d
5	Task 5	2018-05-01	2018-06-04	25d

Figure 7. Gantt diagram of the thesis schedule

- The first task was an introduction to the IPS software, the Lua language, and the definition of the case study. The IPS software introduction was carried out by replicating a simple HRC case in the program, which allowed the author of the thesis to become familiar with the functionalities of the program. For the Lua language, a seminar was held by the developers at Fraunhofer-Chalmers Research Centre in Göteborg, in which the author developed short scripts and learned the basics of the language. Finally, the case study was defined in collaboration with the thesis supervisor: the flywheel cover assembly station.
- For the second task, the case study was reproduced into an IPS scene, as explained in point 2.4. The constraints and variable parameters for the case were established with the assistance of the supervisor (point 3.1), as well as the KPIs used for the workstation optimisation. Initial drafts of the possible script improvements to the software were also written in this phase.
- The third task was focused on the programming of the scripts themselves, with intensive testing in the case study scene, and with frequent discussions and collaboration with the software developers and the thesis supervisor.
- After the scripts had been developed, the fourth task encompassed the simulation of the different scenarios for the case study workstation optimisation that were established in the

second phase, and that will be further elaborated in point 3.1. The data obtained from these simulations was analysed, and an “optimal” solution was defined.

- Finally, the fifth task was dedicated to the documentation of the thesis work, extraction of conclusions, recommendations about future work and further literature search.

3.1 Case study approach

The aim of the case study was to design an HRC workstation in an industrial case by utilizing the developed scripts to simulate different scenarios and compare the quantitative KPIs obtained as output from the simulation software in order to find an “optimal” solution for the station layout.

Three KPIs were defined in collaboration with the supervisor of the thesis: the processing time and RULA ergonomic score [48], both already integrated in the IPS software’s outputs, and a third parameter that is not implemented in the program and was defined for this thesis in order to have more objective indicators for the evaluation: the layout score, which would represent the qualitative analysis regarding safety, product flow and workflow of the station.

This layout score is calculated as the sum of individual scores for each pair of elements in the scene, which are the result of the division between their activity relationship closeness value [7], displayed in Figure 8, and the physical distance between the elements in the scene in the X-Y plane. Thus, the higher the distance between elements, the lower the layout score will be (except for elements whose closeness is not desired).

$$Individual\ score = Relationship\ Value / Distance\ (m) \qquad (1.1)$$

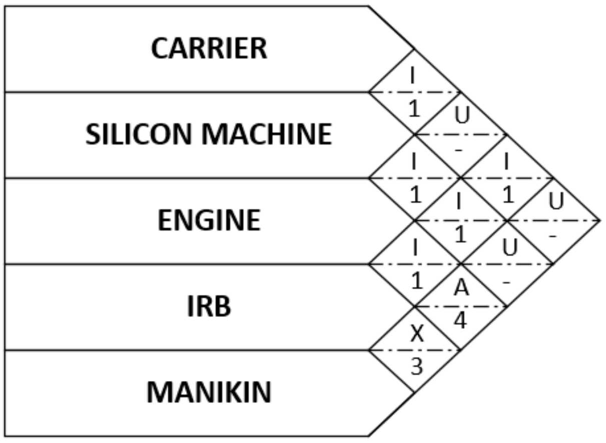
$$Layout\ score = \sum Individual\ score \qquad (1.2)$$

The relationship value is a qualitative assessment which has an arbitrary quantitative score assigned. In this case, the following scores have been assigned to each closeness value (Table 1):

Table 1. Quantitative scores assigned to qualitative values

Value:	**A**	**E**	**I**	**O**	**U**	**X**
Score:	100	75	50	25	0	-100

The qualitative assessment in the Activity relationship diagram was elaborated by the author of the thesis in collaboration with the thesis supervisor, and is based on Muther’s method [7] for assigning closeness values in activities with criteria other than product flow. The diagram obtained is presented the Figure 8.



Value	CLOSENESS	No. of Ratings
A	Absolutely Necessary	1
E	Especially Important	0
I	Important	5
O	Ordinary closeness	0
U	Unimportant	3
X	**Not desirable**	1

Code	REASON
1	Product flow
2	Noise disturbance
3	Safety
4	Workflow

Figure 8 Activity relationship diagram for the case study

Each of the five elements in the diagram represents the element of the same name in the layout (see Figure 6. Top view of the workstation in IPS). Most of the elements' desired closeness values are of normal importance (I) due to the product flow reason: the flywheel cover moves from one element to another, so it is desirable that they are as close as possible, but not critical. Other relations are unimportant (U), since their closeness does not have any impact: the carrier and the manikin, for example, do not have any kind of interaction.

One relation is of absolutely necessary closeness (A): the manikin and the engine must be as close as possible, since the manikin has to work in the engine in order to assemble the flywheel cover once it has been put into place. And finally, one relation has a not desirable closeness (X): the manikin and the IRB, despite it being a collaborative workstation, should be as far as possible from each other due to safety reasons, since while the manikin is working on assembling the cover if the robot is too close a collision could take place, resulting in injuries.

After the KPIs were defined, the design method proposed in by Ore et al. in [6], [32] was utilized to define the number of total simulations to be performed in order to design the workstation and propose an "optimal" solution. Table 2 summarizes the results of applying the method to this case: 7 variables have been identified, and each of them has been defined as either a constant or having a number of different alternatives.

Table 2. Definition of variables for the case study

Variable	Nº of alternatives	Constant
Robot variant	-	X
Robot position	6	-
Robot gripper	-	X
Carrier position	4	-
Hand-over position	20	-
Silicon machine position	4	-
Engine & pillar position	-	X
Manikin family	-	X

- The robot variant variable reflects the possibility of using different types of robots in the workstation design; for this case, it has been defined as a constant, since only one model is utilized (KUKA KR 210 R2700 PRIME [44]).
- The robot position variable indicates the different positions that the IRB can be displaced at in the scene from its original location; a total of 6 alternatives have been proposed: two displacements in the X axis, two in the Y axis, and two in the Z axis.
- The robot gripper variable alludes to the gripper element that is attached to the head of the IRB in order to grab the product, and the different models that could be used for it. In this case, it has been fixed as a constant, since only one model of the gripper is used.
- The carrier position, similar to the robot position, has four possible alternatives of displacement: two in the X axis and two in the Y axis.
- The hand-over position in which the manikin starts to hand-guide the robot is a critical point, since it will be the most impactful one in the ergonomic score. In this case, and given the limitations of the Lua API that will be further discussed in point 5.1, the hand-over position will be a fixed point in the X-Y plane. Thus, the 20 alternatives proposed in the table will be of small incremental movements in the Z axis.

- The silicon machine position, as in the case of the carrier, has four displacement positions: two in the X axis and two in the Y axis.
- The engine and pillar position variables have been fixed as a constant, since it would not be feasible to change their current position in the real workstation environment.
- Finally, the manikin family variable reflects the possibility of using different families from different anthropometric databases. In this case, it is a constant, with the data for the family utilized provided by the Swedish anthropometric database [45].

The combination of all the variables and their number of alternatives would give a total of $6x4x20x4 = 1920$ simulations, which is too large an amount for the scope of this thesis. For this reason, and given that the only variable affecting the ergonomic assessment is the hand-over position, it was decided to simulate those 20 alternatives separately in order to find an optimal ergonomic score, and once that point is fixed the rest of the $6x4x4 = 96$ simulations would be used to compare different layouts regarding the other two KPIs, processing time and layout score.

4 RESULTS

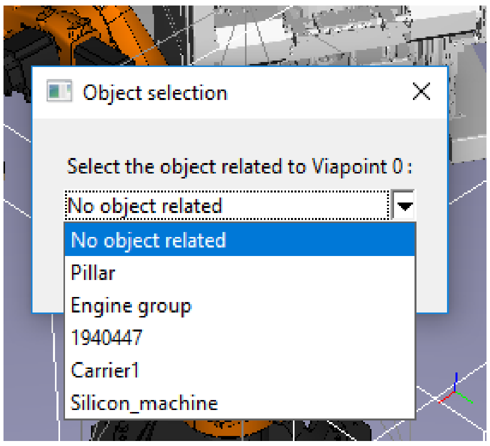
In this chapter are presented the results obtained with the method described in the previous chapter.

4.1 LUA scripts

A total of three main scripts have been developed by the author, based on the requirements for improvements that were established in the beginning of the thesis, and limited by the available functionalities and time constraints of the project. Their aim is to enable the user to perform layout changes in an existing simulation scene and automatically recompute the operation sequence and generate a new simulation reflecting the alterations made. The three scripts have been key named as: Initial Script, Free-move Script and Automated Script.

All three scripts are generalized within the possibilities of the API, so that they may be applicable to a wide range of different scenarios, and not only the case study performed in this thesis. Thus, the scripts will require input from the user in order to account for information that can not be directly gathered from the existing scene, e.g., the object associated with each position or the starting position of the IRB.

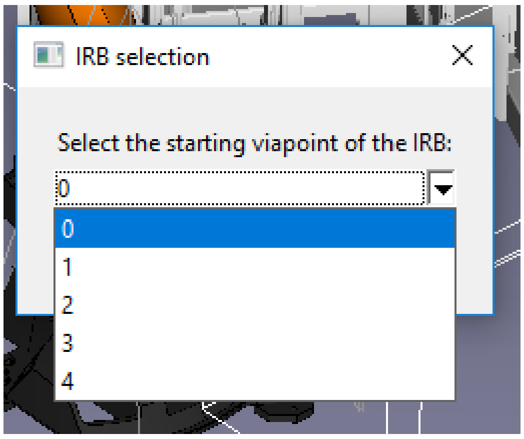
- **Initial Script:** this script must be the first one executed when starting the optimization of a new workstation. It reads the existing scene and saves the following information in vectors: the active objects in the scene tree and their positions, the existing body path planning and all its viapoints, the IRB mode and its position (both TCP position and robot base position), and the Operation Sequence actors and their sequence of actions.
  Once all this information has been gathered, since the goal of the scripts is to be partially generalized for different cases it will be required for the user to input some information (see Figure 9). After having read the viapoints of the path plan and the active objects in the scene, the script will prompt the user to select a related object to each of the viapoints (or establish that there are no objects related). This enables the user to define what is the sequence of movements of the product through objects, instead of viapoints. In the case study example, the viapoint 0 would be associated with the Carrier object, the viapoint 1 with the Silicon machine, and the rest of the viapoints would have no object related to them.
  Finally, the script prompts the user to select the starting viapoint of the IRB TCP in order to account for cases in which the IRB grabs the product in the middle of a sequence, instead of at the beginning (as in the case study).
  After the user has inputted the required information, the script proceeds to calculate and store in vectors the relative position of every active object to their associated viapoint (as well as the TCP). This information will be used by the other two scripts in order to correctly reposition the viapoints in the scene after any objects have been moved.
  Once the script has finished running, the user is free to move objects around and use the Free-move script or run the Automated script for an automatic movement approach.


Object selection

Select the object related to Viapoint 0 :

No object related

- No object related
- Pillar
- Engine group
- 1940447
- Carrier1
- Silicon_machine


IRB selection

Select the starting viapoint of the IRB:

0

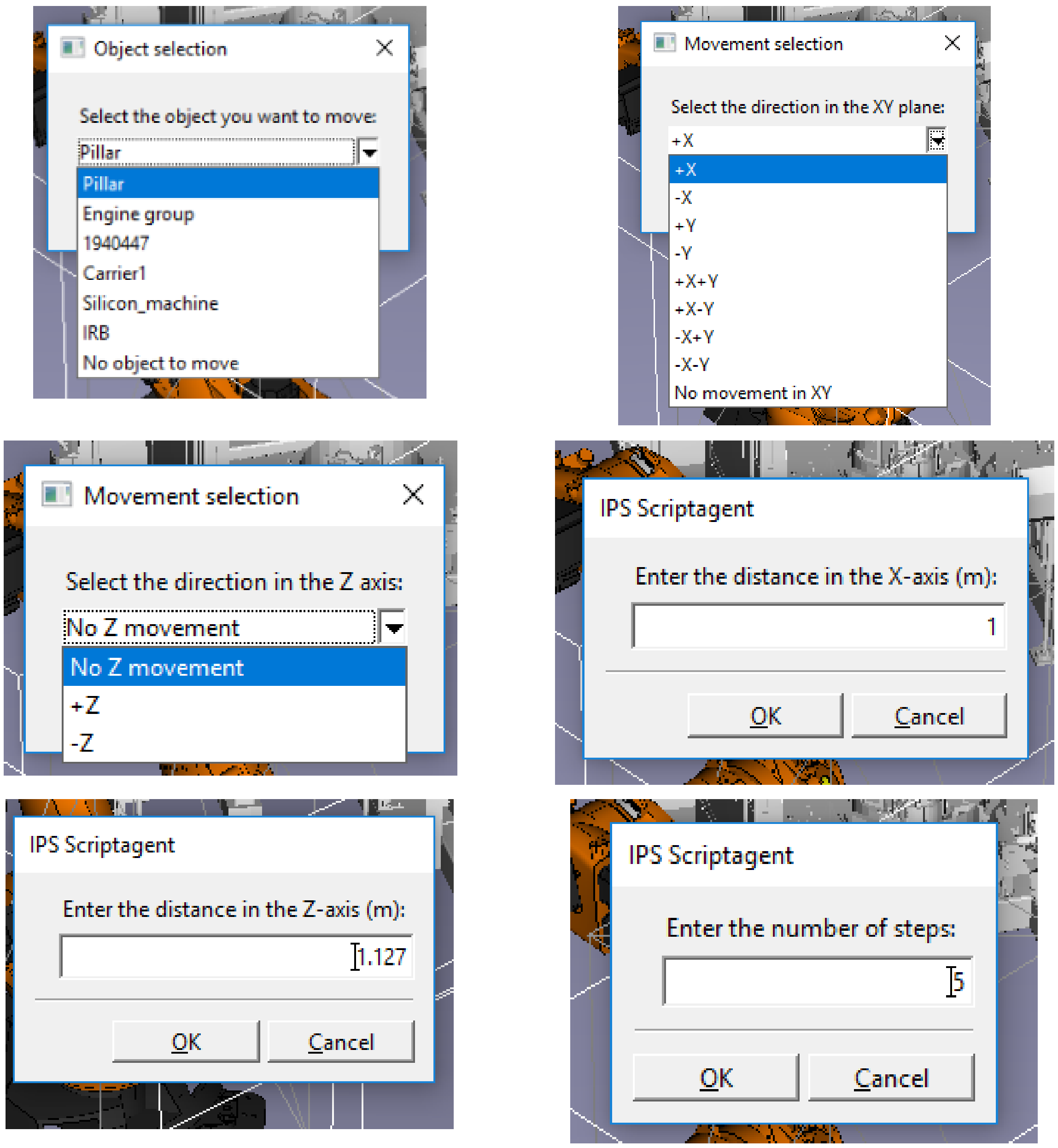
- 0
- 1
- 2
- 3
- 4

Figure 9. Example of user inputs in the Initial Lua Script

- **Free-move script:** this script may be used at any moment after the initial script has been run. The user should start by moving objects to their new desired positions and run the script once they have been placed. A safeguard has been implemented in the script, which will send an alert to the user if any of the objects have been moved out of range of the IRB, and cancel the execution of the rest of the script.
  This script does not require any user input, since all the information can be gathered from the scene and the previously saved data. It will read the scene and retrieve the active objects and their new positions; then, applying the data stored by the initial script regarding relative positioning, it will calculate the new position of the viapoints and reposition them so that they keep the same relative positioning as in the beginning. Then, it will proceed to compute the new Layout score (point 3.1) and store it in a vector for later use. After this, it will automatically run the new path planning, save the movement segments to the scene, and replace the existing movement actions of the product actor in the Operation Sequence by the new movements. Finally, it will execute the updated Operation Sequence, generating a new simulation replay with the processing time, and run the Ergonomic evaluation to obtain the .csv files that are used to compute the RULA score in MATLAB.
  The goal of this script is to provide the user with the possibility of swiftly making significant changes to the existing layout and exploring different options by moving objects to completely different positions.

- **Automated script:** this script may be used at any moment after the initial script has been run. Its goal is to provide a “fine tuning” approach to the layout design of the workstation by allowing the user to select an object to move automatically in a 3-D direction for a determinate distance in a certain number of steps.
  This script will require several inputs from the user in order to establish the conditions for the automated move (see Figure 10). Once the script is run, it will read the active objects tree and immediately prompt the user to select an object to perform the automated move on. After an object has been chosen, an option will be offered to the user to select in which directions of the X-Y plane should the automated move be performed; after that, the user will then select the direction in the Z axis. With the 3-D directions of movements established, the user is given the option to enter the distances (in meters) that the object should be displaced in each of the selected axis, and finally, input the number of steps that have to be performed.
  Once the user has finished inputting all the information, the script will start by calculating the distances that the selected object has to be moved in each of the iterations by dividing the total distance by the number of steps. Then, the script will displace the object by that distance, and recalculate and reposition the viapoints of the path planning (with the information stored by the Initial script regarding relative positioning of objects and

viapoints) in a similar way to the Free-move script. After the viapoints have been relocated, it will compute the new Layout score (point 3.1) and store it in a vector for later use. Then, it will automatically run the new path planning, generate the new movements and replace the existing ones in the Operation Sequence, execute the sequence to generate a new simulation and Ergonomic evaluation, and re-do the same process for each iteration until a number of simulations equal to the number of steps are achieved.


Object selection

Select the object you want to move: Pillar

Movement selection

Select the direction in the XY plane: +X

Movement selection

Select the direction in the Z axis: No Z movement

IPS Scriptagent

Enter the distance in the X-axis (m): 1

IPS Scriptagent

Enter the distance in the Z-axis (m): 1.127

IPS Scriptagent

Enter the number of steps: 5

Figure 10. Example of the selection process for user inputs in the Automated Lua Script

A fourth minor script has been developed, with the purpose of displaying the stored values in the Layout scores vector, so that the user does not need to keep track of the value after each individual simulation.

4.2 Case solution

After the scripts had been developed, the case study was used to test their feasibility and effectiveness in the case of a theoretical HRC workstation design. The methodology for the layout design of the case study is explained in point 3.1.

The Automated script has been used to displace the hand-over position along the Z axis in 20 different steps, from the initial position of 650 mm to a final position of 1600 mm, with a simulation being performed every 50 mm. These are the results obtained regarding the RULA score:

Table 3. RULA scores for the hand-over positions

Z Position (mm)	RULA score	Z Position (mm)	RULA score
650	3,379693	1150	3,261409
700	3,389751	1200	3,260751
750	3,405278	1250	3,259542
800	3,369042	1300	3,337237
850	3,385779	1350	3,319593
900	3,275278	1400	3,317369
950	3,279472	1450	3,314775
1000	3,201278	1500	3,315726
1050	3,275561	1550	3,311185
1100	3,275764	1600	3,316944

This RULA score is calculated as an average of the sum of scores of each of the 10 manikins (5 males and 5 females). An optimal solution is obtained for the Z position of 1000 mm; this will be further elaborated in point 5.2.

After an optimal hand-over position has been determined, a combination of the Automated move and Free-move script is used to perform the 96 simulations defined in point 3.1. The obtained results regarding processing time and layout score are presented in Table 5 and charted in Figure 11. The code figures allude to the positioning of an object relative to their original location, according to Table 4. From left to right, each of the three number is associated to the objects IRB, Silicon machine, and Carrier respectively. Thus, the code 412 would refer to the layout in which the IRB is displaced in the positive Y axis from its original position, the Silicon machine is displaced in the negative X axis, and the Carrier is displaced in the positive X axis.

The magnitude of each of the displacements is of 0.2 meters in the indicated direction, with the codes 5 and 6 being 0.2 and 0.4 meters in the positive Z axis, respectively.

Table 4. Code number and relative position assigned

Code:	**1**	**2**	**3**	**4**	**5**	**6**
Relative position:	-X	+X	-Y	+Y	+Z	+2Z

Several layout codes can be observed to have processing times and scores of 0; these are cases in which one or more of the objects in the scene were out of reach of the IRB after being displaced, thus making it impossible to run the simulation.

The concept of Pareto-optimal solutions was used [49] [6] to create a trade-off curve between layout score and processing times. Depending on the preferences of importance for the decision makers, any of the solutions that are part of the pareto frontier (highlighted in Table 5) may be chosen as an "optimal" solution. Results are further discussed in point 5.2.

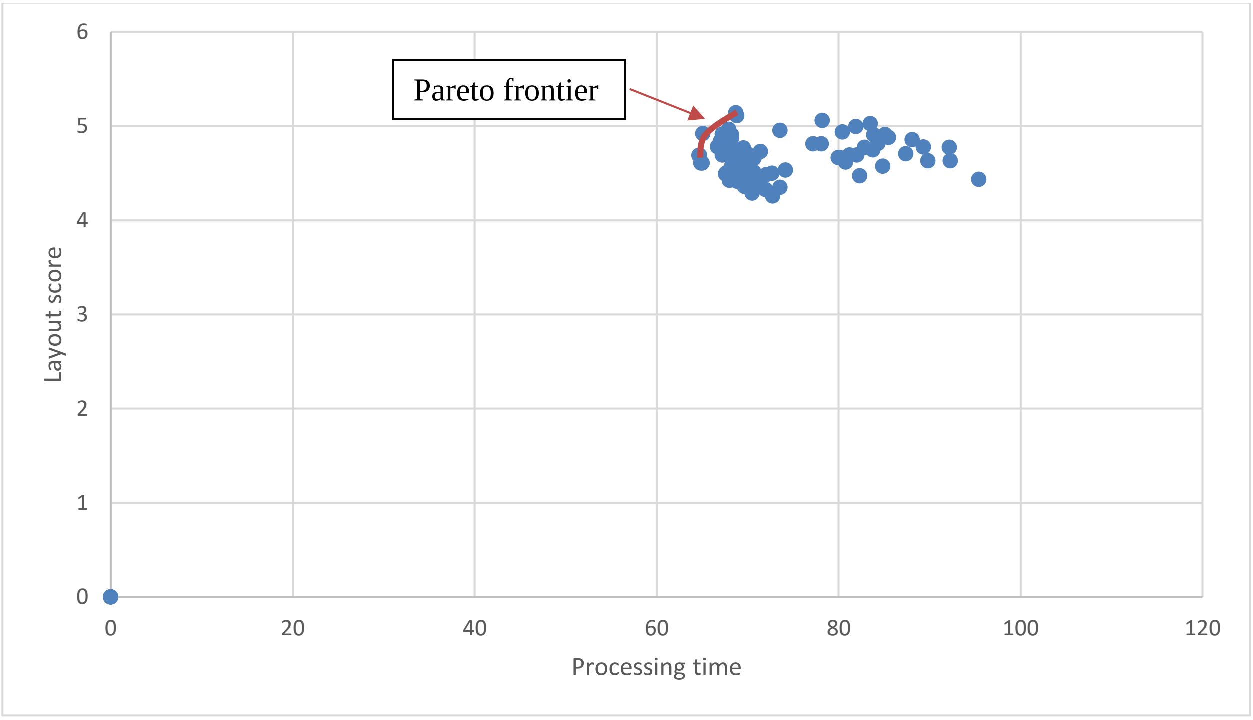


Figure 11. Chart with the plotted results of the case solution, displaying the Pareto Frontier

Table 5. Results of the 96 simulations for the case study

Code	Time (s)	L. Score
111	69,8935	4.585
112	66,6906	4.78
113	70,1634	4.61
114	68,6829	4.753
121	87,3705	4.706
122	83,8813	4.905
123	83,7489	4.748
124	88,0755	4.855
131	89,3096	4.777
132	81,8844	4.996
133	84,3063	4.814
134	73,5503	4.954
141	70,6638	4.512
142	69,6847	4.695
143	68,2387	4.545
144	70,6748	4.655
211	0	0
212	68,6488	4.568
213	68,8519	4.415
214	70,0953	4.559
221	0	0
222	67,2369	4.691
223	69,3097	4.55
224	69,3595	4.658
231	0	0
232	69,5305	4.767
233	69,9004	4.601
234	68,9564	4.741
241	0	0
242	67,5647	4.493
243	69,6372	4.36
244	70,1285	4.471
311	71,9864	4.327
312	67,7236	4.505
313	73,5546	4.348
314	72,0618	4.485
321	95,3873	4.435
322	80,7559	4.616
323	82,3327	4.473
324	84,8349	4.573
331	72,6621	4.5
332	70,0172	4.702
333	74,1421	4.533
334	69,8959	4.666
341	72,7549	4.257
342	68,0095	4.423
343	70,4978	4.287
344	71,2733	4.391
411	69,149	4.742
412	65,0734	**4.921**
413	67,4314	4.757
414	68,2286	4.908
421	85,4815	4.879
422	78,2054	5.06
423	85,0823	4.91
424	83,4873	5.025
431	80,4263	4.937
432	68,7017	**5.14**
433	67,9424	4.964
434	68,7903	5.111
441	70,7861	4.678
442	67,0303	4.845
443	71,4246	4.731
444	67,2093	4.914
511	68,9629	4.509
512	**64,6402**	**4.689**
513	68,8326	4.528
514	68,2307	4.672
521	92,2686	4.63
522	78,0863	4.813
523	80,1342	4.665
524	92,1586	4.773
531	81,198	4.693
532	67,5337	4.896
533	68,1475	4.723
534	68,2125	4.863
541	70,2779	4.442
542	64,8737	4.609
543	68,2774	4.469
544	68,2623	4.58
611	69,044	4.509
612	64,7205	4.689
613	68,996	4.528
614	68,2698	4.672
621	89,8246	4.63
622	77,1945	4.813
623	79,9635	4.665
624	82,842	4.773
631	82,0146	4.693
632	67,5341	4.896
633	68,1468	4.723
634	67,1765	4.863
641	69,2785	4.442
642	65,0041	4.609
643	67,9199	4.469
644	68,2624	4.58

5 DISCUSSION AND CONCLUSIONS

In this chapter, a discussion of the results and the conclusions that the author has drawn during the thesis are presented. The conclusions are based from the analysis with the intention to answer the formulation of questions that was presented in Chapter 1.

5.1 Discussion of the LUA scripts

RQ1: How can simulation software for design of human–industrial robot collaboration workstations be improved through programming of Lua scripts?

The first research question is answered by the developed Lua scripts presented in point 4.1. Three main scripts have been developed: an Initial script to gather information from the scene and receive input parameters from the user; a Free-move script that allows the user to quickly check different layout designs in a broad manner; and an Automated script that enables the user to automatically perform consecutive movements in order to fine tune the layout design of a workstation.

The scripts developed enable the user to perform simulations in a swift, efficient, automated and simplified way in comparison to the traditional method. The scripts reduce the need of manual work to a minimum, only requiring the user to input some parameters to decide which type of movements to perform in the layout design, whereas with the previous procedure it would take a large amount of time and effort to perform even a single simulation after changing the layout.

The time required to execute a simulation with a new layout using the new scripts in estimated to be less than one sixth of the time required with the traditional method; furthermore, most of that time is due to the software's own calculation times for the path planning, with little to no manual labour required.

Besides, with the addition of the Automated script, it is now possible for the user to perform a large number of simulations consecutively without the need of interacting with the program, potentially saving tens or hundreds of hours by running the program without supervision.

Moreover, the scripts can also be used to run the program as a "black box", enabling users to utilize the software without the need of being adepts at it, requiring only the basic knowledge to move objects in the scene and run the scripts; this would, for example, allow layout designers to use the program in order to design HRC workstations without the need of having to learn first all the complexities that it entails.

Finally, since the scripts have been partially generalized, they may be used in different scenarios other than the case study of this thesis, with the trade-off being the need of the user to input several parameters. This partial generalization of the scripts enables them to be used in scenes that fulfil the following requirements:

- The scene must have a single Rigid Body Path Planning that defines the product movement, since it is not possible for the Lua API to incorporate Robot motions into the operation sequence. The possibility of handling more than one body path planning was considered, but deemed not necessary for most cases.
- The scene must contain a single IRB.
- The scene may have only one Manikin Family, since the current version of the software does not allow to have more than a single family as actors in an operation sequence.
- The scene must have a fixed hand-over position in the X-Y plane. This is due to the fact that it is not currently possible through the Lua API to retrieve the necessary information to automatically set a new position for the manikin to move to.

These constraints are given by the currently existing Lua API functionalities, as expressed in the delimitations of the thesis in point 1.3 and establish room for future work.

5.2 Discussion of the case solution

RQ2: How can a human–industrial robot collaborative workstations be optimised in an efficient way using the improved software?

The second research question is answered by the solution of the case study presented in point 4.2 Following the guidelines established in the case study approach (point 3.1), a two-step procedure has been followed to offer an "optimal" solution for the case.

In the first place, the most favourable height from the ergonomics perspective for the hand-over position has been found by using the Automated script to run the defined number of simulations consecutively (Table 3). As it can be seen in the table, all the values for the RULA score are quite similar; this is arguably due to the method used for computing the score, in which the average of the family of 10 manikins (5 males and 5 females, representing 95% of the population spectrum [45]) is calculated, which causes values for the score to not vary significantly since positions that have a lower ergonomic score for people with smaller height may have a higher score for higher people, and vice versa. However, the position of Z=1000 mm has the lowest RULA average value by a notable margin, and thus it is safe to consider it as the optimal position from the ergonomics perspective.

After having defined the optimal height for the hand-over position, the second step was to run the 96 simulations established in the case study and analyse the processing times and layout scores in order to find an "optimal" solution through a Pareto frontier multi-objective optimization [49] [4].

As displayed in the graph in Figure 11 and highlighted in *Table 5*, three points comprise the Pareto frontier: the ones corresponding to codes 412, 432 and 512. Any of these positions could be considered an "optimal" one, and the best one should be selected depending on the preferences of the importance of processing time over layout score.

All the simulations could be performed efficiently with the use of the developed scripts, saving a large amount of time; however, some issues were found during the testing, which give room for improvements and future work:

- When running the automated script to find the optimal height of the hand-over position, no more than 5 simulation steps could be run consecutively; when running a sixth one, an error would cause the script to be interrupted. This issue made it necessary to run the 21 required simulations in 5 different usages of the Automated script (5, 5, 5, 5 and 1 steps respectively), resetting the program after each run of the script.
- Most of the manual work was dedicated to collecting the data from the Ergonomic evaluation for calculating the RULA score. The .csv files of each simulation containing the joint values for all manikins had to be transferred to MATLAB, where a different script was used to calculate the score; afterwards, this score had to be manually inputted in an excel file to keep track of the scores for each simulation.
- The processing times of each simulation had to be manually inputted into an excel file.
- When retrieving the information about the layout score, a short script was used to display the score calculated in each of the simulations that had been run; this information had then to be manually inputted into an excel file together with the processing times.

Once these issues are tackled, it will be possible to perform the simulations with little to no manual work required, which will mark a significant improvement in the optimisation and layout design of HRC workstations with this software.

5.3 Conclusions

The general conclusion from the research performed is that it is possible to improve existing software to simulate, visualise, evaluate and optimise HRC workstations through the use of Lua scripts. With these scripts it is possible to efficiently design the layout of future HRC assembly workstations by performing large amount of simulations in a reduced time and in an automated way.

The main academic contribution of the thesis are the scripts themselves and the new methodology developed by using them to design future HRC workstations. Thanks to the scripts, it is possible to run a large number of simulations, achieving a greater degree of accuracy in the evaluation and comparison of different alternatives in the layout design process, saving time and resources; furthermore, with the addition of the Automated script it is also possible to perform those simulations consecutively without the need of human interaction, which is a remarkable improvement over the current simulation manual procedures.

The scripts are also the main part of the industrial contribution. Through them HRC workstations layout can be designed and improved in an efficient way early in the production design process. However, a number of issues need to be resolved before it can have a major impact in the industries. One is that, even if the scripts have been partly generalized to adapt to different workstations, it can still only cover a limited range of HRC station possibilities; the scripts would need to be able to be utilized in a wider variety of situations for them to be utilized in industry. One other obstacle is the maturity of the software, which has to increase through further development in order to make the required improvements to the scripts, as is discussed in the recommendations and future work.

6 RECOMMENDATIONS AND FUTURE WORK

In this chapter, recommendations on improvements for the solution and future work in this field are presented.

As discussed in Chapter 5.1, several issues and possible improvements to be made to the software in development and the Lua API have been detected:

- The Lua API should include functionalities to enable the full script automation of the robot path planning tool and its integration in the operation sequence. With this, it would be possible to handle more than one product's motions, as well as create robot movements without having the product attached.
- The IRB object should be considered as a solid object for calculating collision avoidance in path planning. This would eliminate the need of using a "robot box" when using the rigid body path planner.
- The operation sequence should be able to handle more than one family of manikins, to represent the cases in which there is more than one human collaborating with the robot in the HRC station.
- Additional functionalities should be added to the Lua API to be able to handle the movements of the manikins, so that it can automatically create move actions to new positions, removing the need for the hand-over position to be static.
- The error that causes the Automated script to fail execution when running more than five consecutive steps should be solved.
- The RULA ergonomic assessment could be incorporated into the software, so that there is no need for an external program to calculate the score.
- The layout score calculations methodology could be added into the software, removing the need to implement it into the script.
- The data transfer method could be improved, by making it possible to automatically export the processing times, RULA score and layout score to a datasheet file.

Aside from these improvements into the demonstrator software, further work should be put into the scripts, making them more generalized by including different possibilities of HRC workstations: more than one IRB collaborating with more than one human, different objects moving at a same time, a manikin moving to different positions that change with the layout, etc.

Furthermore, as presented in the introductory point 1.1, future work in the project with relation to the smart factory and Industry 4.0 concepts ([50], [51]) could be performed by focusing in the information feedback from the real-time HRC stations and its histogram and how this information may be used as inputs in the design process of the workstations.

7 REFERENCES

[1] C. Murphy and T. Perera, “The definition of simulation and its role within an aerospace company,” *Simulation Practice and Theory,* vol. 9, pp. 273-291, 2002.

[2] D. Mourtzis, N. Papakostas, D. Mavrikios, S. Makris and K. Alexopoulos, “The role of simulation in digital manufacturing: applications and outlook,” *International journal of Computer Integrated Manufacturing,* vol. 28, pp. 3-24, 2015.

[3] F. Ore, L. Hanson, N. Delfs and M. Wiktorsson, “Human industrial robot collaboration - development and application of simulation software,” *International Journal of Human Factors Modelling and Simulation,* vol. 5, no. 2, pp. 164-185, 2015.

[4] F. Ore, “Human - Industrial Robot Collaboration: Simulation, Visualisation and Optimisation of future assembly workstations,” Mälardalen University Press Thesis, Västerås, Sweden, 2015.

[5] IPS AB, [Online]. Available: http://www.fcc.chalmers.se/software/ips/ips-imma/. [Accessed 20 May 2018].

[6] F. Ore, L. Hanson and M. Wiktorsson, “Method for design of human-industrial robot collaboration workstations,” in *27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM 2017*, Modena, Italy, 2017.

[7] R. Muther and L. Hales, Systematic Layout Planning, Fourth Edition, Marietta, USA: Management & Industrial Research Publications, 2015.

[8] L. Fritzsche, “Ergonomics risk assessment with digital human models in car assembly: Simulation versus real life,” *Human Factors and Ergonomics in Manufacturing & Service Industries,* vol. 20, no. 4, pp. 287-299, 2010.

[9] M. F. Zaeh and M. Prasch, “Systematic workplace and assembly redesign for aging workforces,” *Production Engineering,* vol. 1, no. 1, pp. 57-64, 2007.

[10] S. Oberer-Treitz, T. Dietz and A. Verl, “Safety in industrial applications: From fixed fences to direct interaction,” in *44th International Symposium on Robotics, ISR*, Seoul, Korea, 2013.

[11] G. Reinhart, R. Spillner and Y. Shen, “Apporaches of Applying Human-Robot-Interaction-Technologies to Assist Workers with Musculoskeletal Disorders in Production,” in *Intelligent Robotics and Applications, 5th International Conference, ICIRA, Proceedings, Part II,* Montreal, Canada, 2012.

[12] J. Krüger, T. Lien and A. Verl, “A Cooperation of human and machines in assembly lines, Keynote paper,” *CIRP Annals - Manufacturing Technology,* vol. 58, pp. 628-646, 2009.

[13] J. T. C. Tan, F. Duan, Y. Zhang, K. Watanabe, R. Kato and T. Arai, “Human-robot collaboration in cellular manufacturing: Design and development,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2009.

[14] I. Aaltonen, T. Salmi and I. Marstio, *Refining Levels of Collaboration to Support the Design and Evaluation of Human-Robot Interaction in the Manufacturing Industry,* Stockholm, Sweden: paper approved for the 51st CIRP Conference on Manufacturing Systems (CIRP CMS 2018), 2018.

[15] J. Krüger, B. Nickolay, P. Heyer and G. Seliger, “Image based 3D Surveillance for flexible Man-Robot-Cooperation,” *CIRP Annals - Manufacturing technology,* vol. 54, pp. 19-222, 2005.

[16] A. Stopp, T. Baldauf, R. Hantsche, S. Horstmann, S. Kristensen, F. Lohnert, C. Priem and B. Ruscher, “The manufacturing assistant: Safe, interactive teaching of operation

sequences," in *11th IEEE International Workshop on Robot and Human Interactive Communication,* Berlin, Germany, 2002.

[17] H. Kagermann, W. Wahlster and J. Helbig, "Recommendations for Implementing the Strategic Initiative Industrie 4.0: Securing the Future of German Manufacturing Industry; Final Report of the Industrie 4.0 Working Group," Acatech, Berlin, Germany, 2013.

[18] M. A. Goodrich and A. C. Schultz, "Human-Robot Interaction: A Survey," *Foundations and Trends in Human-Computer Interaction,* vol. 1, no. 3, pp. 203-275, 2008.

[19] S. Walther and T. Guhl, "Classification of physical human-robot interaction scenarios to identify relevant requirements," in *Proceedings of ISR/Robotik 2014, 41st International Symposium on Robotics,* 2014.

[20] ISO (2011b) ISO 10218-2:2011, Robots and robotic devices - Safety requirements for industrial robots - Part 2: Robot systems and integration, Geneva, Switzerland: International Organization for Standardisation.

[21] J. Fryman, "Updating the Industrial Robot Safety Standard," in *Proceedings of ISR/Robotik 2014; 41st International Symposium on Robotics,* Munich, Germany, 2014.

[22] ISO (2011a) ISO 10218-1:2011, Robots and robotic devices - Safety requirements for industrial robots - Part 1: Robots, Geneva, Switzerland: International Organization for Standardisation.

[23] M. Vasic and A. Billard, "Safety Issues in Human-Robot Interactions," in *Procedings of the 2013 IEEE International Conference on Robotics and Automation, ICRA,* 2013.

[24] M. Bortolini, M. Gamberi, F. Pilati and A. Regattieri, *Automatic Assessment of the Ergonomic Risk for Manual Manufacturing and Assembly Activities Through Optical Motion Capture Technology,* Stockholm, Sweden: paper approved for the 51st CIRP Conference on Manufacturing Systems (CIRP CMS 2018), 2018.

[25] L. Wang, B. Schmidt and A. Y. C. Nee, "Vision-guided active collision avoidance for human-robot collaborations," *Manufacturing Letters,* vol. 1, no. 1, pp. 5-8, 2013.

[26] P. Bobka, T. Germann, J. K. Heyn, R. Gerbers, F. Dietrich and K. and Dröder, "Simulation Platform to Investigate Safe Operation of Human-Robot Collaboration Systems," *Procedia CIRP,* vol. 44, pp. 187-192, 2016.

[27] J. T. C. Tan, F. Duan, R. Kato and T. Arai, "Safety Strategy for Human-Robot Collaboration: Design and Development in Cellular Manufacturing," *Advanced Robotics,* vol. 24, no. 10, pp. 839-860, 2012.

[28] F. Pini, F. Leali and M. Ansalomi, "A systematic approach to the engineering design of a HRC workcell for bio-medical product assembly," in *IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA),* Luxembourg, 2015.

[29] G. Pahl, W. Beitz, J. Feldhusen and K. Grote, Engineering Design: A Systematic Approach, London, UK: Springer-Verlag, 2007.

[30] F. Chen, K. Sekiyama, J. Huang, B. Sun, H. Sasaki and T. Fukuda, "An assembly strategy scheduling method for human and robot coordinated cell manufacturing," *International Journal of Intelligent Computing and Cybernetics,* vol. 4, pp. 487-510, 2011.

[31] P. Tsarouchi, A.-S. Matthaiakis, S. Makris and G. Chryssolouris, "On a human-robot collaboration in an assembly cell," *International Journal of Computer Integrated Manufacturing,* pp. 1-10, 2016.

[32] F. Ore, B. R. Vemula, L. Hanson and M. Wiktorson, "Human-Industrial Robot Collaboration: Application of Simulation Software for Workstation Optimisation," *Procedia CIRP,* vol. 44, pp. 181-186, 2016.

[33] IPS AB, [Online]. Available: http://industrialpathsolutions.se/. [Accessed 20 May 2018].

[34] FCC, [Online]. Available: http://www.fcc.chalmers.se/software/ips/. [Accessed 13 May 2018].

[35] IPS AB, [Online]. Available: http://industrialpathsolutions.se/ips-rigid-body-path-planner. [Accessed 20 May 2018].

[36] IPS AB, [Online]. Available: http://industrialpathsolutions.se/ips-robot-optimization. [Accessed 20 May 2018].

[37] J. Laring, M. Forsman, R. Kadefors and R. Örtengren, "MTM-based ergonomic workload analysis," *International Journal of Industrial Ergonomics,* vol. 30, no. 3, pp. 135-148, 2002.

[38] L. McAtamney and E. Corlett, "RULA: a survey method for the investigation of work-related upper limb disorders," *Applied Ergonomics,* vol. 24, pp. 91-99, 1993.

[39] R. Ierusalimschy, Programming in lua, Second Edition, Rio de Janeiro, Brazil, 2006.

[40] R. Ierusalimschy, L. H. De Figueiredo and W. Celes Filho, "Lua-an extensible extension language," *Softw., Pract. Exper.,* vol. 26, no. 6, pp. 635-652, 1996.

[41] Lua, [Online]. Available: https://www.lua.org/about.html. [Accessed 12 May 2018].

[42] P. Mårdberg, J. Carlson, R. Bohlin, N. Delfs, S. Gustafsson, D. Högberg and L. Hanson, "Using a formal high-level language and an automated manikin to automatically generate assembly instructions," *International Journal of Human Factors Modeling and Simulation,* vol. 4, pp. 233-249, 2014.

[43] V. V. Unhelkar, H. C. Siu and J. A. Shah, "Comparative Performance of Human and Mobile Robotic Assistants in Collaborative Fetch-and-Deliver Tasks," in *HRI '14: Proceedings of the 9th ACM/IEEE International Conference on Human-Robot Interaction,* 2014.

[44] Kuka, [Online]. Available: https://www.kuka.com/en-in/products/robotics-systems/industrial-robots/kr-quantec-prime. [Accessed 16 May 2018].

[45] L. Hanson, L. Sperling, G. Gard, S. Ipsen and C. Olivares Vergara, "Swedish anthropometrics for product and workplace design," *Applied Ergonomics,* vol. 40, no. 4, pp. 797-806, 2009.

[46] A. R. Hevner, "A Three Cycle View of Design Science Research," *Scandinavian Journal of Information Systems,* vol. 19, no. 2, pp. 87-92, 2007.

[47] A. R. Hevner, S. T. March, J. Park and S. Ram, "Design Science in Information Systems Research," *MIS Quarterly,* vol. 28, no. 1, pp. 75-105, 2004.

[48] F. Ore, L. Hanson, M. Wiktorsson and E. Yvonne, "Automation constraints in human-industrial robot collaborative workstation design," in *7th Swedish Production Symposium (SPS) 2016,* Lund, Sweden, 2016.

[49] E. M. Kasprzak and K. E. Lewis, "An Approach to Facilitate Decision Tradeoffs in Pareto Solution Sets," *Journal of Engineering Valuation and Cost Analysis,* vol. 3, no. 1, pp. 173-187, 2000.

[50] S. Wang, J. Wan, D. Zhang, D. Li and C. Zhang, "Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination," *Computer Networks,* vol. 101, pp. 158-168, 2016.

[51] W. Kühn, "Digital factory: Simulation Enhancing the Product and Production Engineering Process," in *Proceedings of the Winter Simulation Conference, WSC 06*, Monterey, US, 2006.

APPENDIX A: COMPLETE SCRIPT CODE

This appendix includes the complete code of the three main Lua scripts developed by the author.

```
function calculateAngle(rotvector) --Function to calculate the angle given a rotation matrix R from
a transf3 object
	shift = 10 ^ 4   --Rounds the angle to the 4th decimal
	resultsin = math.floor( rotvector['r2x']*shift + 0.5 ) / shift
	resultcos = math.floor( rotvector['r1x']*shift + 0.5 ) / shift
	local angle=math.atan2(resultsin,resultcos);  --Calculates the angle with the inverse of the
tangent
return angle
end
function calculateDistance(vector1,vector2)  --Function to calculate the distance vector between
two points; returns module and angle
	distx=vector1['tx']-vector2['tx'];
	if distx<0 then 		--The sign of the relative x position has to be used to adjust the angle
in the pp2 script
		dirx=-1;
		else
		dirx=1;
	end
	disty=vector1['ty']-vector2['ty'];
	local ang=math.atan2(disty,distx);	--Calculates the angle that forms the distance vector
between the two points
	local dist=math.sqrt(distx^2 + disty^2);
	return dist, ang, dirx
end
--Declaration of variables
	options=StringVector()
	irbpos=StringVector()
	layoutscore=NumberVector()
	object={};	coords={};	viapoints={};	relatedobject={};
	dist={};	relang={};	dir={};		posz={};
	ang1={};	ang2={};	diff_ang={};
	options:push_back("No object related");
--Go through the active objects tree
	root = Ips.getActiveObjectsRoot();
	child = root:getFirstChild();
	belowRoot=root:getNextSibling();
	obj=child;
	i=1;
while not(obj == nil) and not (obj:equals(belowRoot)) do
	if (obj:getType() == "RigidBodyObject") then
		object[i]=obj:toRigidBodyObject();		--Store rigid bodies in the
object vector
		coords[i]=object[i]:getFrameInWorld();		--Store the coordinates of the
rigid body in the coords vector
		options:push_back(obj:getLabel());		--Add the name of the object
to the options vector
		i=i+1;
	end
```

```
        obj=obj:getObjectBelow();
end
--Get the IRB object and its coordinates
        root=Ips.getMechanismRoot();
        child = root:getFirstChild();
        belowRoot=root:getNextSibling();
        obj=child;
while not(obj == nil) and not (obj:getLabel()=="Simulations") do
        if (obj:getType() == "IRBObject") then
                obj=obj:toIRBObject();
                coordirb=obj:getTCPInWorld();
        end
        obj=obj:getObjectBelow();
end
--Get the rigid body planning object and its viapoints
        root = Ips.getProcessRoot();
        pp = root:getFirstChild();
        pp=pp:toRigidBodyPathPlanning();
        nviapoints=pp:getNumViaPoints();
--Store the viapoints in a vector
        viapoints[0]=pp:getStart();
        if nviapoints>0 then
                for i=1,nviapoints do
                        viapoints[i]=pp:getViaPoint(i-1);
                end
        end
        viapoints[nviapoints+1]=pp:getGoal();
        totalpoints=nviapoints+2;
--Display a drop down list for the user to select the object related to each viapoint
for j=1, totalpoints  do
choice=Ips.inputDropDownList("Object selection","Select the object related to Viapoint "..(j-1).."
:", options);
        for i=1, table.maxn(object) do
        if choice==i then
                relatedobject[j-1]=i;    --For each viapoint saves in a vector what object is related
to it
                end
        end
end
--Goes through the related objects vector and calculates the relative spatial position of the viapoint
and its related object
for i=0, totalpoints-1 do
if relatedobject[i]~=nil then    --Only use viapoints that have an object related
        dist[i],relang[i],dir[i]=calculateDistance(viapoints[i],coords[relatedobject[i]]);
        ang1[i]=calculateAngle(coords[relatedobject[i]])
        ang2[i]=calculateAngle(viapoints[i])
        diff_ang[i]=ang2[i]-ang1[i];
        posz[i]=viapoints[i]['tz']-coords[relatedobject[i]]['tz'];
        end
end
--Asks the user to select the starting point of the IRB
for i=0, totalpoints-1 do
```

```
        irbpos:push_back(i)
end
irbstart=Ips.inputDropDownList("IRB selection","Select the starting viapoint of the IRB:",
irbpos);

--Calculates the relative spatial position of the IRB and the input viapoint
distirb,relangirb,dirirb=calculateDistance(coordirb,viapoints[irbstart]);
ang1_irb=calculateAngle(viapoints[irbstart])
ang2_irb=calculateAngle(coordirb)
diff_angirb=ang2_irb-ang1_irb;
poszirb=coordirb['tz']-viapoints[irbstart]['tz'];
print("PP1 ran succesfully - You can move objects around now.")
--Disables the reseting of variables when the script finishes, so that they can be used in other scripts
Script.resetStateWhenFinished(false);

function rotMatrix(anglestart,anglediff)        --Calculates the new rotation matrix R from an start
angle and an angle increment
        local angle=anglestart+anglediff;
        r1=Vector3d(math.cos(angle),-math.sin(angle),0);
        r2=Vector3d(math.sin(angle),math.cos(angle),0);
        r3=Vector3d(0,0,1);
        local R=Rot3(r1,r2,r3);
        return R
end
function transVector(startpos,distance,angle,posz,dir)        --Calculates the translation vector
with a starting position, distance and angle
        distx=distance*math.cos(angle);        --The dir parameter is used to adjust the sign for the
relative position of the object in the x axis
        disty=distance*math.sin(angle);
        local t=Vector3d(startpos['tx']+distx,startpos['ty']+disty,startpos['tz']+posz);
        return t
end
function calculateAngle(rotvector)    --Function to calculate the angle given a rotation matrix R
from a transf3 object
        local shift = 10 ^ 4
   resultsin = math.floor( rotvector['r2x']*shift + 0.5 ) / shift
        resultcos = math.floor( rotvector['r1x']*shift + 0.5 ) / shift
        local angle=math.atan2(resultsin,resultcos);
return angle
end
function calculateDistance(obj1,obj2)        --Calculates the distance between two objects in the
X-Y plane
        local distx=obj1['tx']-obj2['tx'];
        local disty=obj1['ty']-obj2['ty'];
        local calculatedDistance=math.sqrt(distx^2+disty^2);
return calculatedDistance
end
function calculateScore(distance,relation)
        if relation==0 then
                return 0
        end
        local score=1/(distance*relation/100);
        return score
end
--Get the Robot box
        root = Ips.getGeometryRoot();
        child = root:getFirstChild();
        belowRoot=root:getNextSibling();
        obj=child;
        while not(obj==nil) and not (obj:equals(belowRoot)) do
                obj=obj:getObjectBelow();
                if (obj:getLabel() == "Geometry Group 3") then
                        irbbox=obj:toPositionedTreeObject();
                end
        end
--Offer option to select object to move

        --Get the active rigid bodies
        optionsmove=StringVector();
        root = Ips.getActiveObjectsRoot();
        child = root:getFirstChild();
        belowRoot=root:getNextSibling();
        obj=child;
        i=1;
while not(obj == nil) and not (obj:equals(belowRoot)) do
        obj=obj:getObjectBelow();
        if (obj:getType() == "RigidBodyObject") then
                object[i]=obj:toRigidBodyObject();              --Store rigid bodies in the
object vector
                coords[i]=object[i]:getFrameInWorld();          --Store the coordinates of the
rigid body in the coords vector
                optionsmove:push_back(obj:getLabel());                  --Add the name of the
object to the options vector
                i=i+1;
        end
end
optionsmove:push_back("IRB");
optionsmove:push_back("No object to move"); --Add a not move anything option
choice1=Ips.inputDropDownList("Object selection","Select the object you want to move:",
optionsmove);
for i=1, table.maxn(object) do
        if choice1==i-1 then
                objecttomove=i; --Saves the index of the object to move
                end
end
if choice1==(table.maxn(object)+1) then
 do return end  --End the script if the option not to move anything is chosen
end
--Offer option to select type of movement
optionsmovetype=Vector({'+X','-X','+Y','-Y','+X+Y','+X-Y','-X+Y','-X-Y','No movement in
XY'})
choice2=Ips.inputDropDownList("Movement selection","Select the direction in the XY plane:",
optionsmovetype);
optionsmovetype=Vector({'No Z movement','+Z','-Z'})
```

```
choice3=Ips.inputDropDownList("Movement selection","Select the direction in the Z axis:", optionsmovetype);
choice4=0; choice5=0; choice6=0;
--Offer option to select distance
if choice2==0 or choice2==1 or choice2==4 or choice2==5 or choice2==6 or choice2==7 then
        choice4=Ips.inputNumberWithDefault("Enter the distance in the X-axis (m): ", 1.0);
end
if choice2==2 or choice2==3 or choice2==4 or choice2==5 or choice2==6 or choice2==7 then
        choice5=Ips.inputNumberWithDefault("Enter the distance in the Y-axis (m): ", 1.0);
end
if choice3==1 or choice3==2 then
        choice6=Ips.inputNumberWithDefault("Enter the distance in the Z-axis (m): ", 1.0);
end
choice7=Ips.inputNumberWithDefault("Enter the number of steps: ", 5.0);
iterations=math.floor(choice7);
stepdistancex=choice4/iterations;
stepdistancey=choice5/iterations;
stepdistancez=choice6/iterations;
for w=1, iterations do
        w=w+1;
--Declaration of variables
        rotation={};    total_rot={};
        transf_t={};    transf_R={};   transf_T={};

--Performs a cleanup of motion simulations
        root=Ips.getSimulationsRoot();
        child=root:getFirstChild();
        belowRoot=root:getNextSibling();
        obj=child;
        while not(obj == nil) and not(obj:getLabel()=="Measures")do
                if (obj:getType() == "RBMotionSimulation") then
                        child=obj:getNextSibling();
                        Ips.deleteTreeObject(obj);
                end
                obj=child;
        end
--Get the IRB mechanism group and retrieve its current coordinates
        root = Ips.getMechanismRoot();
        child = root:getFirstChild();
        belowRoot=root:getNextSibling();
        obj=child;
        while not(obj==nil) and not (obj:equals(belowRoot)) do
                obj=obj:getObjectBelow();
                if (obj:getLabel() == "IRB") then
                        irbbase=obj:getParent();
                        irbbase=irbbase:toPositionedTreeObject();
                        irbbasecoords=irbbase:getTWorld();
                end
        end
--Go through the active objects tree to retrieve the rigid bodies and current coordinates
        root = Ips.getActiveObjectsRoot();
        child = root:getFirstChild();
```

```
	belowRoot=root:getNextSibling();
	obj=child;
	i=1;
	while not(obj == nil) and not (obj:getLabel()=="Mechanisms") do
		obj=obj:getObjectBelow();
		if (obj:getType() == "RigidBodyObject") then
			object[i]=obj:toRigidBodyObject();			--Store rigid bodies in
the object vector
			coords[i]=object[i]:getFrameInWorld();			--Store		the
coordinates of the rigid body in the coords vector
			i=i+1;
		end
	end
--Get the IRB object and its mode
	root=Ips.getMechanismRoot();
	child = root:getFirstChild();
	belowRoot=root:getNextSibling();
	obj=child;
	while not(obj == nil) and not (obj:getLabel()=="Simulations") do
		obj=obj:getObjectBelow();
		if (obj:getType() == "IRBObject") then
			irb=obj:toIRBObject();
			mode=irb:getIKinMode();
		end
	end
--Move the object to a new position
	if choice1==table.maxn(object) then --If the object is the IRB
		if w<=(iterations+1) then
			if choice3==0 then
			elseif choice3==1 then
	irbbasecoords['tz']=irbbasecoords['tz']+stepdistancez;
				irbbase:setTWorld(irbbasecoords);
			elseif choice3==2 then
				irbbasecoords['tz']=irbbasecoords['tz']-stepdistancez;
				irbbase:setTWorld(irbbasecoords);
			end
			if choice2==0 then
	irbbasecoords['tx']=irbbasecoords['tx']+stepdistancex;
				irbbase:setTWorld(irbbasecoords);
			elseif choice2==1 then
				irbbasecoords['tx']=irbbasecoords['tx']-stepdistancex;
				irbbase:setTWorld(irbbasecoords);
			elseif choice2==2 then
	irbbasecoords['ty']=irbbasecoords['ty']+stepdistancey;
				irbbase:setTWorld(irbbasecoords);
			elseif choice2==3 then
				irbbasecoords['ty']=irbbasecoords['ty']-stepdistancey;
				irbbase:setTWorld(irbbasecoords);
			elseif choice2==4 then
	irbbasecoords['tx']=irbbasecoords['tx']+stepdistancex;
	irbbasecoords['ty']=irbbasecoords['ty']+stepdistancey;
				irbbase:setTWorld(irbbasecoords);
```

```
                    elseif choice2==5 then
irbbasecoords['tx']=irbbasecoords['tx']+stepdistancex;
                              irbbasecoords['ty']=irbbasecoords['ty']-stepdistancey;
                              irbbase:setTWorld(irbbasecoords);
                    elseif choice2==6 then
                              irbbasecoords['tx']=irbbasecoords['tx']-stepdistancex;
irbbasecoords['ty']=irbbasecoords['ty']+stepdistancey;
                              irbbase:setTWorld(irbbasecoords);
                    elseif choice2==7 then
                              irbbasecoords['tx']=irbbasecoords['tx']-stepdistancex;
                              irbbasecoords['ty']=irbbasecoords['ty']-stepdistancey;
                              irbbase:setTWorld(irbbasecoords);
                    end
          end
elseif choice1~=table.maxn(object) then
          if w<=(iterations+1) then
                    if choice3==0 then
                    elseif choice3==1 then
coords[objecttomove]['tz']=coords[objecttomove]['tz']+stepdistancez;
object[objecttomove]:setFrameInWorld(coords[objecttomove]);
                    elseif choice3==2 then
coords[objecttomove]['tz']=coords[objecttomove]['tz']-stepdistancez;
object[objecttomove]:setFrameInWorld(coords[objecttomove]);
                    end
                    if choice2==0 then
coords[objecttomove]['tx']=coords[objecttomove]['tx']+stepdistancex;
object[objecttomove]:setFrameInWorld(coords[objecttomove]);
                    elseif choice2==1 then
coords[objecttomove]['tx']=coords[objecttomove]['tx']-stepdistancex;
object[objecttomove]:setFrameInWorld(coords[objecttomove]);
                    elseif choice2==2 then
coords[objecttomove]['ty']=coords[objecttomove]['ty']+stepdistancey;
object[objecttomove]:setFrameInWorld(coords[objecttomove]);
                    elseif choice2==3 then
coords[objecttomove]['ty']=coords[objecttomove]['ty']-stepdistancey;
object[objecttomove]:setFrameInWorld(coords[objecttomove]);
                    elseif choice2==4 then
coords[objecttomove]['tx']=coords[objecttomove]['tx']+stepdistancex;
coords[objecttomove]['ty']=coords[objecttomove]['ty']+stepdistancey;
object[objecttomove]:setFrameInWorld(coords[objecttomove]);
                    elseif choice2==5 then
coords[objecttomove]['tx']=coords[objecttomove]['tx']+stepdistancex;
coords[objecttomove]['ty']=coords[objecttomove]['ty']-stepdistancey;
object[objecttomove]:setFrameInWorld(coords[objecttomove]);
                    elseif choice2==6 then
coords[objecttomove]['tx']=coords[objecttomove]['tx']-stepdistancex;
coords[objecttomove]['ty']=coords[objecttomove]['ty']+stepdistancey;
object[objecttomove]:setFrameInWorld(coords[objecttomove]);
                    elseif choice2==7 then
coords[objecttomove]['tx']=coords[objecttomove]['tx']-stepdistancex;
coords[objecttomove]['ty']=coords[objecttomove]['ty']-stepdistancey;
object[objecttomove]:setFrameInWorld(coords[objecttomove]);
```

```
                        end
                end
        end
--Get the rigid body planning object and its attached rigid body
        root = Ips.getProcessRoot();
        pp = root:getFirstChild();
        pp=pp:toRigidBodyPathPlanning();
        rigbod=pp:getRigidBody();
--For each viapoint with an object linked, calculate the new position of the viapoint based on the
new position of the object
        for i=0, totalpoints-1 do
                if relatedobject[i]~=nil then    --Only use viapoints that have an object related
                        ang2[i]=calculateAngle(coords[relatedobject[i]]);    --New angle of the
object in world coordinates
                        rotation[i]=ang2[i]-ang1[i];    --Calculates the rotation of the object from
the initial position in world coordinates
                        total_rot[i]=rotation[i]+relang[i];        --Adds the rotation of the object to the
relative angle between object and viapoint
        transf_t[i]=transVector(coords[relatedobject[i]],dist[i],total_rot[i],posz[i],dir[i])    --Calls
the transVector function
        transf_R[i]=rotMatrix(calculateAngle(coords[relatedobject[i]]),diff_ang[i])
        --Calls the rotMatrix function
                        transf_T[i]=Transf3(transf_R[i], transf_t[i]); --Creates the Transf3 element
with the spatial coordinates
--Places the viapoint in the new calculated coordinates
                        if i==0 then
                                pp:setStart(transf_T[i]);
                                rigbod:setFrameInWorld(transf_T[i]);
                        elseif i==(totalpoints-1) then
                                pp:setGoal(transf_T[i])
                        else
                                pp:setViaPoint(i-1,transf_T[i]);
                        end
                end
        end
--Updates the viapoints vector with the new positions of the viapoints
        viapoints[0]=pp:getStart();
        if nviapoints>0 then
                for i=1,nviapoints do
                        viapoints[i]=pp:getViaPoint(i-1);
                end
        end
        viapoints[nviapoints+1]=pp:getGoal();
--Calculates the new starting position of the IRB
        ang2_irb=calculateAngle(viapoints[irbstart]);
        rotationirb=ang2_irb-ang1_irb;
        total_rotirb=rotationirb+relangirb;
--Checks if the IRB is in range of the viapoints
for i=0, totalpoints-1 do
transf_tirb=transVector(viapoints[i],distirb,total_rotirb,poszirb,dirirb)
transf_Rirb=rotMatrix(calculateAngle(viapoints[i]),diff_angirb)
transf_Tirb=Transf3(transf_Rirb, transf_tirb);
```

```
transf_Tirb['r3z']=-1;                                  --Sets the Z direction to negative
transf_Tirb['r1y']=transf_Tirb['r1y']*-1        --Changes the direction of the Y axis
transf_Tirb['r2y']=transf_Tirb['r2y']*-1        --Changes the direciton of the Y axis
if irb:setFacePlateInWorld(transf_Tirb,mode,false,0,0,0)==false then
        Ips.alert("Viapoint "..i.." out of IRB range")
        do return end
end
end
--Places the IRB faceplate in the new starting position
transf_tirb=transVector(viapoints[irbstart],distirb,total_rotirb,poszirb,dirirb)
transf_Rirb=rotMatrix(calculateAngle(viapoints[irbstart]),diff_angirb)
transf_Tirb=Transf3(transf_Rirb, transf_tirb);
transf_Tirb['r3z']=-1;                                  --Sets the Z direction to negative
transf_Tirb['r1y']=transf_Tirb['r1y']*-1        --Changes the direction of the Y axis
transf_Tirb['r2y']=transf_Tirb['r2y']*-1        --Changes the direciton of the Y axis
irb:setFacePlateInWorld(transf_Tirb,mode,false,0,0,0)
--Places the IRB box in the new position of the IRB base
        --irbbox:getTWorld();
        --DO THE LAYOUT SCORE CALCULATION
--coords[1] is the pillar
--coords[2] is the Engine Group
--coords[3] is the Flywheel cover
--coords[4] is the Carrier
--coords[5] is the Silicon_machine
--irbbasecoords is the IRB
--viapoints[2] is the manikin
        A=100;          E=75;  I=50;   O=25;  U=0;   X=-100;                 score={};
        totalscore=0;
        score[0]=calculateScore(calculateDistance(coords[4],coords[5]),I);
        score[1]=calculateScore(calculateDistance(coords[4],coords[2]),U);
        score[2]=calculateScore(calculateDistance(coords[4],irbbasecoords),I);
        score[3]=calculateScore(calculateDistance(coords[4],viapoints[2]),U);
        score[4]=calculateScore(calculateDistance(coords[5],coords[2]),I);
        score[5]=calculateScore(calculateDistance(coords[5],irbbasecoords),I);
        score[6]=calculateScore(calculateDistance(coords[5],viapoints[2]),U);
        score[7]=calculateScore(calculateDistance(coords[2],irbbasecoords),I);
        score[8]=calculateScore(calculateDistance(coords[2],viapoints[2]),A);
        score[9]=calculateScore(calculateDistance(irbbasecoords,viapoints[2]),X);
        for i=0,9 do
        totalscore=totalscore+score[i];
        end
        layoutscore:push_back(totalscore);
        print(tostring(totalscore));
--3rd Set the planning box for the path planning
        planningbox=pp:setAutoBox();
        planningbox.zmax=1.6; --Set maximum height to not get out of the robots range
        pp:setPlanningBox(planningbox);
--4th do the path planning with the new points and perform one smoothing on it
        pp:planPaths();
        pp:smooth();
        --pp:smooth();
        --pp:smooth();
        --pp:smooth();
        local smoothedMotionSimulation = pp:pushToScene()
--5th set the velocity of the simulation to the maximum velocity of the robot (0.25)
        root = Ips.getSimulationsRoot();
        child = root:getFirstChild();
        belowRoot=root:getNextSibling();
        obj=child;
        while not(obj == nil) and not (obj:getLabel()=="Measures") do
                obj=obj:getObjectBelow();
                if (obj:getType() == "RBMotion") then
                        motion=obj:toRigidBodyMotion();
                        motion:setLocked(false);
                        local nWaypoints = motion:getNumWayPoints()
                        for i=0,nWaypoints-1 do
                                wpoint=motion:getWayPoint(i);
                                wpoint:setVelocity(0.25);
                        end
                        motion:setLocked(true);
                end
        end
        print("PP2 ran succesfully")
--1st Obtain the number of segments (number of viapoints + 1), and the rigid body used in the path
planning
        root = Ips.getProcessRoot();
        pp = root:getFirstChild();
        pp=pp:toRigidBodyPathPlanning();
        nviapoints=pp:getNumViaPoints();
        totalsegments=nviapoints+1;
        rigbod=pp:getRigidBody();
--Obtain the Operation Sequence tree object
        root=Ips.getProcessRoot();
        child=root:getFirstChild();
        belowRoot=root:getNextSibling();
        obj=child;
        while not(obj == nil) and not(obj:getType()=="OperationSequence")do
                obj=obj:getObjectBelow();
        end

        if (obj:getType() == "OperationSequence") then
                opseq=obj:toOperationSequence();
        end
--2nd Create vectors with the actions of the different actors
        actor=ActorVector();
        actor=opseq:getActors();
        nactors=actor:size();
        for i=0,nactors-1 do
                if actor[i]:getSgObject()==nil then
                        irbactions=ActionVector();
                        irbactions=opseq:getActorActions(actor[i]);
                elseif actor[i]:getSgObject():getLabel()==rigbod:getLabel() then
                        body=actor[i];
                        bodyactions=ActionVector();
```

```
                    bodyactions=opseq:getActorActions(body);
                    --Clean up the current actions of the rigid body
                    for i=0,bodyactions:size()-1 do
                            opseq:removeAction(bodyactions[i]);
                    end
            elseif actor[i]:getSgObject():getType()=="SGFamily" then
                    manikinactions=ActionVector();
                    manikinactions=opseq:getActorActions(actor[i]);
            end
    end
--3rd Create the new object's "follow motion" actions
    root=Ips.getSimulationsRoot();
    child=root:getFirstChild();
    belowRoot=root:getNextSibling();
    obj=child;
    i=1;
    segments={};
    --Save the motions in a vector
    while not(obj == nil) and not(obj:getLabel()=="Measures")do
            obj=obj:getObjectBelow();
            if (obj:getType() == "RBMotion") then
                    segments[i]=obj:toRigidBodyMotion();
                    i=i+1;
            end
    end
    --Turn the vector into actions
    for i=1, totalsegments do
            opseq:CreateActiveObjectFollow(body,segments[i]);
    end
    bodyactions=opseq:getActorActions(body);
--4th Establish precedence constraints
    bodyactions[0]:addPrecedenceAction(irbactions[0]);
    manikinactions[0]:addPrecedenceAction(bodyactions[1]);
    bodyactions[2]:addPrecedenceAction(manikinactions[1]);
    irbactions[1]:addPrecedenceAction(bodyactions[2]);
    bodyactions[3]:addPrecedenceAction(irbactions[1]);
    manikinactions[2]:addPrecedenceAction(bodyactions[3]);
--Set the current states as start states
    for i=0,nactors-1 do
            actor[i]:setCurrentStateAsStart();
    end
--5th Execute the sequence to generate the replay
    replay=opseq:executeSequence();
    tim=replay:getFinalTime();
--6th Perform the ergonomics analysis
    --replay:computeErgonomicScore("Demo",0,tim);

end
function rotMatrix(anglestart,anglediff)        --Calculates the new rotation matrix R from an start
angle and an angle increment
    local angle=anglestart+anglediff;
    r1=Vector3d(math.cos(angle),-math.sin(angle),0);
```

```
        r2=Vector3d(math.sin(angle),math.cos(angle),0);
        r3=Vector3d(0,0,1);
        local R=Rot3(r1,r2,r3);
        return R
end
function transVector(startpos,distance,angle,posz,dir)      --Calculates the translation vector
with a starting position, distance and angle
        distx=distance*math.cos(angle);     --The dir parameter is used to adjust the sign for the
relative position of the object in the x axis
        disty=distance*math.sin(angle);
        local t=Vector3d(startpos['tx']+distx,startpos['ty']+disty,startpos['tz']+posz);
        return t
end
function calculateAngle(rotvector)   --Function to calculate the angle given a rotation matrix R
from a transf3 object
        local shift = 10 ^ 4
   resultsin = math.floor( rotvector['r2x']*shift + 0.5 ) / shift
        resultcos = math.floor( rotvector['r1x']*shift + 0.5 ) / shift
        local angle=math.atan2(resultsin,resultcos);
return angle
end
function calculateDistance(obj1,obj2)       --Calculates the distance between two objects in the
X-Y plane
        local distx=obj1['tx']-obj2['tx'];
        local disty=obj1['ty']-obj2['ty'];
        local calculatedDistance=math.sqrt(distx^2+disty^2);
return calculatedDistance
end
function calculateScore(distance,relation)
        if relation==0 then
                return 0
        end
        local score=1/(distance*relation/100);
        return score
end
--Get the Robot box
        root = Ips.getGeometryRoot();
        child = root:getFirstChild();
        belowRoot=root:getNextSibling();
        obj=child;
        while not(obj==nil) and not (obj:equals(belowRoot)) do
                obj=obj:getObjectBelow();
                if (obj:getLabel() == "Geometry Group 3") then
                        irbbox=obj:toPositionedTreeObject();
                end
        end

--Declaration of variables
rotation={};   total_rot={};
transf_t={};   transf_R={};   transf_T={};
--Performs a cleanup of motion simulations
        root=Ips.getSimulationsRoot();
        child=root:getFirstChild();
        belowRoot=root:getNextSibling();
        obj=child;
        while not(obj == nil) and not(obj:getLabel()=="Measures")do
                if (obj:getType() == "RBMotionSimulation") then
                        child=obj:getNextSibling();
                        Ips.deleteTreeObject(obj);
                end
                obj=child;
        end
--Get the IRB mechanism group and retrieve its current coordinates
        root = Ips.getMechanismRoot();
        child = root:getFirstChild();
        belowRoot=root:getNextSibling();
        obj=child;
        while not(obj==nil) and not (obj:equals(belowRoot)) do
                obj=obj:getObjectBelow();
                if (obj:getLabel() == "IRB") then
                        irbbase=obj:getParent();
                        irbbase=irbbase:toPositionedTreeObject();
                        irbbasecoords=irbbase:getTWorld();
                end
        end
--Go through the active objects tree to retrieve the rigid bodies and current coordinates
        root = Ips.getActiveObjectsRoot();
        child = root:getFirstChild();
        belowRoot=root:getNextSibling();
        obj=child;
        i=1;
while not(obj == nil) and not (obj:getLabel()=="Mechanisms") do
        obj=obj:getObjectBelow();
        if (obj:getType() == "RigidBodyObject") then
                object[i]=obj:toRigidBodyObject();              --Store rigid bodies in the
object vector
                coords[i]=object[i]:getFrameInWorld();          --Store the coordinates of the
rigid body in the coords vector
                i=i+1;
        end
end
--Get the IRB object and its mode
        root=Ips.getMechanismRoot();
        child = root:getFirstChild();
        belowRoot=root:getNextSibling();
        obj=child;
while not(obj == nil) and not (obj:getLabel()=="Simulations") do
        obj=obj:getObjectBelow();
        if (obj:getType() == "IRBObject") then
                irb=obj:toIRBObject();
                mode=irb:getIKinMode();
        end
end
--Get the rigid body planning object and its attached rigid body
```

```
        root = Ips.getProcessRoot();
        pp = root:getFirstChild();
        pp=pp:toRigidBodyPathPlanning();
        rigbod=pp:getRigidBody();
--For each viapoint with an object linked, calculate the new position of the viapoint based on the
new position of the object
for i=0, totalpoints-1 do
if relatedobject[i]~=nil then   --Only use viapoints that have an object related
        ang2[i]=calculateAngle(coords[relatedobject[i]]);   --New angle of the object in world
coordinates
        rotation[i]=ang2[i]-ang1[i];  --Calculates the rotation of the object from the initial
position in world coordinates
        total_rot[i]=rotation[i]+relang[i];     --Adds the rotation of the object to the relative angle
between object and viapoint
        transf_t[i]=transVector(coords[relatedobject[i]],dist[i],total_rot[i],posz[i],dir[i])   --Calls
the transVector function
        transf_R[i]=rotMatrix(calculateAngle(coords[relatedobject[i]]),diff_ang[i])
        --Calls the rotMatrix function
        transf_T[i]=Transf3(transf_R[i], transf_t[i]); --Creates the Transf3 element with the spatial
coordinates
        --Places the viapoint in the new calculated coordinates
        if i==0 then
                pp:setStart(transf_T[i]);
                rigbod:setFrameInWorld(transf_T[i]);
                elseif i==(totalpoints-1) then
                pp:setGoal(transf_T[i])
                else
                pp:setViaPoint(i-1,transf_T[i]);
        end
end
end
--Updates the viapoints vector with the new positions of the viapoints
viapoints[0]=pp:getStart();
if nviapoints>0 then
        for i=1,nviapoints do
                viapoints[i]=pp:getViaPoint(i-1);
        end
end
viapoints[nviapoints+1]=pp:getGoal();
--Calculates the new starting position of the IRB
ang2_irb=calculateAngle(viapoints[irbstart]);
rotationirb=ang2_irb-ang1_irb;
total_rotirb=rotationirb+relangirb;
--Checks if the IRB is in range of the viapoints
for i=0, totalpoints-1 do
transf_tirb=transVector(viapoints[i],distirb,total_rotirb,poszirb,dirirb)
transf_Rirb=rotMatrix(calculateAngle(viapoints[i]),diff_angirb)
transf_Tirb=Transf3(transf_Rirb, transf_tirb);
transf_Tirb['r3z']=-1;                          --Sets the Z direction to negative
transf_Tirb['r1y']=transf_Tirb['r1y']*-1        --Changes the direction of the Y axis
transf_Tirb['r2y']=transf_Tirb['r2y']*-1        --Changes the direciton of the Y axis
if irb:setFacePlateInWorld(transf_Tirb,mode,false,0,0,0)==false then
```

```
        Ips.alert("Viapoint "..i.." out of IRB range")
        do return end
end
end
--Places the IRB faceplate in the new starting position
transf_tirb=transVector(viapoints[irbstart],distirb,total_rotirb,poszirb,dirirb)
transf_Rirb=rotMatrix(calculateAngle(viapoints[irbstart]),diff_angirb)
transf_Tirb=Transf3(transf_Rirb, transf_tirb);
transf_Tirb['r3z']=-1;                                  --Sets the Z direction to negative
transf_Tirb['r1y']=transf_Tirb['r1y']*-1        --Changes the direction of the Y axis
transf_Tirb['r2y']=transf_Tirb['r2y']*-1        --Changes the direciton of the Y axis
irb:setFacePlateInWorld(transf_Tirb,mode,false,0,0,0)
--Places the IRB box in the new position of the IRB base
        --irbboxcoords=irbbox:getTWorld();
                --DO THE LAYOUT SCORE CALCULATION
--coords[1] is the pillar
--coords[2] is the Engine Group
--coords[3] is the Flywheel cover
--coords[4] is the Carrier
--coords[5] is the Silicon_machine
--irbbasecoords is the IRB
--viapoints[2] is the manikin
        A=100;          E=75;  I=50;   O=25;  U=0;    X=-100;                 score={};
        totalscore=0;
        score[0]=calculateScore(calculateDistance(coords[4],coords[5]),I);
        score[1]=calculateScore(calculateDistance(coords[4],coords[2]),U);
        score[2]=calculateScore(calculateDistance(coords[4],irbbasecoords),I);
        score[3]=calculateScore(calculateDistance(coords[4],viapoints[2]),U);
        score[4]=calculateScore(calculateDistance(coords[5],coords[2]),I);
        score[5]=calculateScore(calculateDistance(coords[5],irbbasecoords),I);
        score[6]=calculateScore(calculateDistance(coords[5],viapoints[2]),U);
        score[7]=calculateScore(calculateDistance(coords[2],irbbasecoords),I);
        score[8]=calculateScore(calculateDistance(coords[2],viapoints[2]),A);
        score[9]=calculateScore(calculateDistance(irbbasecoords,viapoints[2]),X);
        for i=0,9 do
        totalscore=totalscore+score[i];
        end
        layoutscore:push_back(totalscore);
        print(tostring(totalscore));
--3rd Set the planning box for the path planning
planningbox=pp:setAutoBox();
planningbox.zmax=1.6; --Set maximum height to not get out of the robots range
pp:setPlanningBox(planningbox);
--4th do the path planning with the new points and perform one smoothing on it
pp:planPaths();
pp:smooth();
--pp:smooth();
--pp:smooth();
--pp:smooth();
local smoothedMotionSimulation = pp:pushToScene()
--5th set the velocity of the simulation to the maximum velocity of the robot (0.25)
root = Ips.getSimulationsRoot();
```

```
        child = root:getFirstChild();
        belowRoot=root:getNextSibling();
        obj=child;
while not(obj == nil) and not (obj:getLabel()=="Measures") do
        obj=obj:getObjectBelow();
        if (obj:getType() == "RBMotion") then
                motion=obj:toRigidBodyMotion();
                motion:setLocked(false);
                local nWaypoints = motion:getNumWayPoints()
                for i=0,nWaypoints-1 do
                        wpoint=motion:getWayPoint(i);
                        wpoint:setVelocity(0.25);
                end
                motion:setLocked(true);
        end
end
print("PP2 ran succesfully")
--1st Obtain the number of segments (number of viapoints + 1), and the rigid body used in the path
planning
        root = Ips.getProcessRoot();
        pp = root:getFirstChild();
        pp=pp:toRigidBodyPathPlanning();
        nviapoints=pp:getNumViaPoints();
        totalsegments=nviapoints+1;
        rigbod=pp:getRigidBody();
--Obtain the Operation Sequence tree object
        root=Ips.getProcessRoot();
        child=root:getFirstChild();
        belowRoot=root:getNextSibling();
        obj=child;
while not(obj == nil) and not(obj:getType()=="OperationSequence")do
        obj=obj:getObjectBelow();
end
        if (obj:getType() == "OperationSequence") then
                opseq=obj:toOperationSequence();
        end
--2nd Create vectors with the actions of the different actors
        actor=ActorVector();
        actor=opseq:getActors();
        nactors=actor:size();
        for i=0,nactors-1 do
                if actor[i]:getSgObject()==nil then
                        irbactions=ActionVector();
                        irbactions=opseq:getActorActions(actor[i]);
                elseif actor[i]:getSgObject():getLabel()==rigbod:getLabel() then
                        body=actor[i];
                        bodyactions=ActionVector();
                        bodyactions=opseq:getActorActions(body);
                        --Clean up the current actions of the rigid body
                        for i=0,bodyactions:size()-1 do
                                opseq:removeAction(bodyactions[i]);
                        end
                elseif actor[i]:getSgObject():getType()=="SGFamily" then
                        manikinactions=ActionVector();
                        manikinactions=opseq:getActorActions(actor[i]);
                end
        end
--3rd Create the new object's "follow motion" actions
        root=Ips.getSimulationsRoot();
        child=root:getFirstChild();
        belowRoot=root:getNextSibling();
        obj=child;
        i=1;
        segments={};
        --Save the motions in a vector
        while not(obj == nil) and not(obj:getLabel()=="Measures")do
                obj=obj:getObjectBelow();
                if (obj:getType() == "RBMotion") then
                        segments[i]=obj:toRigidBodyMotion();
                        i=i+1;
                end
        end
        --Turn the vector into actions
        for i=1, totalsegments do
                opseq:CreateActiveObjectFollow(body,segments[i]);
        end
        bodyactions=opseq:getActorActions(body);
--4th Establish precedence constraints
        bodyactions[0]:addPrecedenceAction(irbactions[0]);
        manikinactions[0]:addPrecedenceAction(bodyactions[1]);
        bodyactions[2]:addPrecedenceAction(manikinactions[1]);
        irbactions[1]:addPrecedenceAction(bodyactions[2]);
        bodyactions[3]:addPrecedenceAction(irbactions[1]);
        manikinactions[2]:addPrecedenceAction(bodyactions[3]);
        precac=bodyactions[0]:getPrecedenceAction();
        print(tostring(precac));
        print(tostring(precac[0]:getType()));
--Set the current states as start states
        for i=0,nactors-1 do
                actor[i]:setCurrentStateAsStart();
        end
--5th Execute the sequence to generate the replay
        replay=opseq:executeSequence();
        tim=replay:getFinalTime();
--6th Perform the ergonomics analysis
        --replay:computeErgonomicScore("Demo",0,tim);
```