TAMPERE UNIVERSITY OF TECHNOLOGY

HÉCTOR GARCÍA PÁJARO

A 3D REAL-TIME MONITORING SYSTEM FOR A PRODUCTION LINE

MASTER OF SCIENCE THESIS

Examiner: Professor José L. Martínez Lastra

Examiner and topic approved in the

Automation, Mechanical and Materials

Engineering Faculty Council meeting on

09 May 2012

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
GARCÍA PÁJARO, HÉCTOR: A 3D Real-Time Monitoring System for a Production Line
Master of Science Thesis, 62 pages, 14 appended pages
June 2012
Keywords: Monitoring, 3D, Visualization of production lines, web manufacturing, digital manufacturing, web-based monitoring, assets monitoring, 3D modelling.

It is hypothesized than a transparent view of the factory floor, in real time, will provide grounds for achieving a better control over the factory assets, including maintenance activities and scheduling optimization. The main goal of this research is to provide a Real-Time 3D Monitoring of manufacturing systems by capitalizing the early 3D model created for simulation proposes and the eventualized information offered by the Factory Information Systems in the shape of web services via DPWS. In addition the monitoring application should be accessible via a web-based application allowing multi-viewers on multi-platform, including mobile ones.

The thesis discusses the background of Factory Monitoring and Visualization Systems, including 3D Modeling and Animation, and Web-based Applications. Then a robotized assembly line located at FAST-Lab. is presented for been used as the test bed of the implementation by the designed System. CATIA and DELMIA Software model the 3D view of the line assets. A method for importing to Unity3d Software is also developed and documented. The Unity Game engine produces the animation of the 3D models. Then, the Application is published as a web-based application running under Unity browser plugin. Lastly, an interface for the communication of the 3D virtual world and the line Sensors and Data Acquisition application is presented in order to achieve the Real-Time triggering of the events driven the 3D virtual representation of the assembly line.

PREFACE

This Master Thesis was carried out during my Erasmus Exchange program at FAST (Factory Automation Systems and Technology) laboratory in Tampere University of Technology.

It represents the fulfilment of my university studies, and furthermore, the end of a piece of my life as a student and the beginning of a new one. It is the work of a year-long experience in Finland, an amazing one that I will keep forever in my mind and heart.

I would like to thank Prof. José Lastra the opportunity given to carry out this thesis at FAST Laboratory. I would like to show also my gratitude also to Prof. José Antonio Pérez, for encouraging me to take this step in my career. Thanks also to supervisor Jani Jokinen for his advice.

I would like to show my gratitude to Luis Gonzalez, for all the advice and support in order to achieve the results of this thesis.

I would also like to be grateful to my FAST colleagues, for sharing this wonderful experience.

I don’t want to forget acknowledgments to my parents, for all their support. Thanks also to my friends and family, for their companionship and friendship. And special thanks also to all those people who incited me to grow, to learn, to share and to live.

Tampere. 18, May 2012.

HÉCTOR GARCÍA PÁJARO

A te.

“In 40 years’ time, I will not remember whether I arrived first or second. But .I will remember the emotions that I experienced”

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

API	Application Programming Interface
B-Rep	Boundary Representation
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CAM	Computer Aided Manufacturing
CAMX	Computer Aided Manufacturing using XML
CGM	CATIA Geometric Modeller
CSG	Constructive Solid Geometry
CSS	Cascade Style Sheets
DCC	Digital Creation Content
ERP	Enterprise Resource Planning
FAST	Factory Automation and Systems Technology
GUI	Graphical User Interface
HMI	Human Machine Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IGES	Initial Graphics Exchange Specification
MIT	Massachusetts Institute of Technology
NFC	Near Field Communication
NURBS	Non-uniform rational basis spline
PERT	Program Evaluation and Review Technique
PLC	Programmable Logic Controller
PLM	Product Lifecycle Management
RTU	Remote Terminal Unit
SCADA	Supervisory Control And Data Acquisition
STEP	Standard for the Exchange of Product
STL	STereoLithography
TUT	Tampere University of Technology
URL	Uniform Resource Locator
VRML	(Virtual Reality Modelling Language
XML	eXtensible Markup Language

1. INTRODUCTION

The purpose of this chapter is to initiate the present thesis settling the reader into the frame of reference for the thesis subject.

Section 1.1 constitutes a first approach and puts the thesis topic into context. Building on these foundations, section 1.2 defines the thesis problem and explain the need of working out it. Sections from 1.3 to 1.5 set respectively the objectives, methodology and assumptions and limitations for the thesis. Finally, section 1.6 outlines the thesis.

1.1 Presentation of the Thesis topic

The manufacturing industry has gone into new challenges during the last years. Market globalisation has brought global business decentralization and manufacturing operations outsourcing. Therefore companies need decentralized tools for factory, product and supply chain management and for teams' remote cooperation. Everyday more and more, engineering tasks such as product design, factory supervision and control or resources maintenance need to be done with decentralized multi-agent tools. [Chen et al., 2002]

Manufacturing companies need to leverage the supply chain and the partners' network operation, in order to drive down costs by means of increasing human resources productivity.

Factory information has to flow all around the world, reaching every target in spite of the barriers of geographical distance. Products are manufactured all over the world in large supply chains that include outsourced partners networks. Thus, collaboration means between companies and partners need to be outsourced from the company internal resources.

This all comes to being a challenging task, with Internet providing an enormous capacity for putting together long-distance company resources and providing a mean for effective remote teams' collaboration. [WANG 2010]

Companies have been moving for the last decade into the world of e-manufacturing. The goal being to build a distributed manufacturing environment and network that can provide fast, decentralized and real-time solutions for the everyday challenges of the engineering company activities. [de Albuquerque & Lelièvre-Berna 1998]

Because all of this, web applications have spread in the last years around all domains of engineering, both in academic and commercial fields, supporting and connecting working groups and organizations.

Furthermore, factory information has become a fundamental matter of every manufacturing facility. The amount of available data at the factory floor level has grown in line with the technology development. This has risen up a new troublesome issue, which is the presentation of all these data for humans in an effective and efficient way.

Nowadays, widespread monitoring systems collect a huge amount of raw data at the sensor level which is not handled in view of presenting easily understandable information. Proposals for visualization of monitoring systems have been developed in the industry in mainly two different trends. On the one hand, SCADA systems including graphics, charts and 2D drawings and schemes have been created, resulting in applications with poor graphics and deficient information. On the other hand, there have been attempts to create more rich applications based on Java3d or other similar technologies; this approach has not been truly successful because it leads to very complex and weighty application, with a development and updating cost that does not compensate for its features.

Space-time based processes, such as product assembly in a production lines, make up an environment in which assets situational awareness comes to be an important advantage. In such processes, assets need to be located and identified at every moment. A monitoring systems that can present the collected data as valuable visual information in a 3D time-dependant model helps to leverage the supervision and control process by means of providing a fastly comprehensive, intuitive interface.

Therefore, this thesis discusses and suggests a proposal for the building of a 3D Real-Time Monitoring System of a production line using the commercial UNITY 3D engine, as well as studying the implementation of a use case at FASTory Line at Tampere University of Technology.

1.2 Problem description and justification

The increasing amount of acquired data at sensor level and the expansion of information technologies demand to explore new paths for the visualization and navigation of the acquired data.

Nowadays, most monitoring systems deal with 2D visualization in a specific workstation, be it a HMI display attached to a PLC or another controller, be it an independent PC. Data that is presented to the human target as 2D graphics misleads information and is abstractedly shown. 3D graphics have the advantage of being more intuitive and being able to present more intuitive information, this translating in a faster and a better information understanding. [Feldhorst 2010] In addition, because of the obfuscation of information when working with a big bunch of raw data and 2D visualization, it is impossible to get an overview of the factory global state.

3D visual information of production lines is especially valuable for manufacturing companies that work in geographically decentralized environments or in outsourced networks of partners. In these situations, information receivers do not have a direct access to the physical manufacturing plant, and information must be as reliable as possible, obeying position and time constraints so that the viewer can get a comprehensive description of the production line situation. In those environments or partners networks, also line state, modifications on its working or product flow along the line need to be observed instantly no matter where the observer is located in the easiest, fastest, most comprehensive and most reliable way.

3D visualization offers the best performance when managing large amounts of data or when trying to provide a general overview of a production line. [Agrusa et al., 2009]

Hazardous or time critical environments, in which the capacity of figuring out the plant situation in a fast and intuitive way comes to be indispensable, are another field in which 3D visual, real-time information play an important role. In order to get efficient and fast answers to production problems, information needs to be presented in a most intuitive way as possible.

Other concern is that at factory floor level, workers and operators need to be provided with the best possible intuitive information. Resources or products state is not always easy to find out at a glance in many production lines. Getting a fast and well-structured overview of the process and plant situation, respecting location, spatial and time constraints helps to the identification of conflicts and their causes.

Some applications have worked on the camera-based approach for the building of 3D monitoring systems. However, the bandwidth consumption involves a limitation in the development of web-based real-time applications [Wang et al., 2011]. In addition, camera recording monitoring systems don’t support some features which are available in 3D computer modelled applications such as zoom in and out, viewpoint navigation or perspective views.

Another boost for the use of 3D graphics is that they have become available at low cost during the last years. However, in the majority of cases, the development of a 3D graphical user interface for monitoring system is still higher than the equivalent 2D system. [Sano et al., 2011]

The proposed solution in the present thesis involves reusing the computer CAD models of the production line equipment that were built originally for design purposes, in the new monitoring system. Thus, a 3D monitoring system that is highly reliable can be created in a shorter time than the development of a system using a conventional graphical libraries or technologies such as Java3d.

In addition, as it was previously discussed, current manufacturing environments make companies to work in distributed systems, built by networks of teams or partners. Therefore, the monitoring system should ideally run as a web-based application.

The fact that it can run on the browser, without any specific software installation requirements allows the same application to target different platforms, thus being accessible not only everywhere, but by everyone.

1.3 Objectives

The objective of this thesis is building an application for the 3D Real-Time Monitoring of an assembly line on behalf of the visualization of factory assets information.

This means that the system has to present the monitored data in a way that a view of the global production line is provided. Through 3D visualization, the amount of useful information given is increased, thus reducing the amount of necessary data collected. In addition, the application shall take into account the location and spatial relationships of the factory assets they shall be faithfully reproduced in the visualization, improving the accuracy of the given information. The application development shall also pay special attention to build an intuitive Graphical User Interface. Reduce the amount of data that must be observed at

Furthermore, the developed application should be platform independent and accessible world-wide, this leading to a web published application accessible through any browser.

Finally, assets and line status have to be presented observing time relationships, because of this; data retrieving should be done on real-time.

1.4 Methodology

Review and discussion of state of the art on 3D modelling and visualization, 3d monitoring systems and web applications.

An overview about 3D visualization of production lines and monitoring systems is given. In addition, basic fundamentals about 3D modelling and different system development tools are discussed.

Study of Unity3d game engine soundness and its integration as a valid tool for engineering applications.

Later, it is discussed the Unity 3D content development engine, and it is studied its suitability for developing the required 3D monitoring system.

Test bed presentation for the 3D Monitoring System.

FASTory Line, located in the FAST (Factory Automation and Systems Technology) Laboratory at TUT (Tampere University of Technology), is introduced and its state, technologies and tools are examined

Use case presentation and implementation of a 3D Real-Time Monitoring System.

Finally, the 3D monitoring system that constitutes the proposal of this thesis is developed. CAD models for the line equipment are imported to Unity 3D engine and the animation scripts are written in JavaScript language. The application is connected to the Line Data Acquisition System so that the equipment status is updated in real time.

1.5 Assumptions and Limitations.

Due to the fact that the assembly line used as test bed for the implementation of this thesis proposal has already been taking part in several other projects there are some considerations to be made.

First of all, it's assumed that CAD files with 3D models for the objects of the production line are, till some extent, already available.

In addition, it also has to be considered that line data acquisition, including sensor and PLC communication, are out of the scope of this thesis. However, an overview of the data acquisition, its treatment and its pushing to the 3D Monitoring application is presented in Appendix 1 at the reader's disposal.

Finally, it is also assumed that this thesis proposal development is focused in the technological, and commercial issues such as application attack vulnerability are overlooked.

1.6 Thesis Outline

This thesis, with the exception of the current chapter which is used as an introduction, is organised as follows.

First of all, Chapter 2 presents the theoretical background and a technology review for the thesis topic.

Afterwards, Chapter 3 focuses on the development of a use case and its implementation on the chosen test bed.

Finally, Chapter 4 puts forward the conclusions and results developed during the thesis work. Possible future work and extended applications of the current proposal are discussed in this chapter as well.

2. THEORETICAL BACKGROUND

This chapter discusses the subject concerning this thesis topic. Section 2.1 introduces the reader into the 3D visualization of production lines, tools and applications are discussed. Next, Section 2.2 elaborates on 3D computer graphics modelling, some commercial software applications are presented. Later on, Section 2.3 discusses the issue of assets monitoring and situational awareness. With this in mind, section 2.4 reviews 3D monitoring systems. Besides, Section 2.5 elaborates on the main tool employed to carry out this thesis, game engines. Finally, Section 2.6 discusses web applications.

2.1 3D Visualization of Production Lines

As it was stated in the introductory chapter of the present thesis, 3D visualization of production lines offers a number of advantages over traditional visualization methods based on charts, graphs, 2D models, colour codes or text labels.

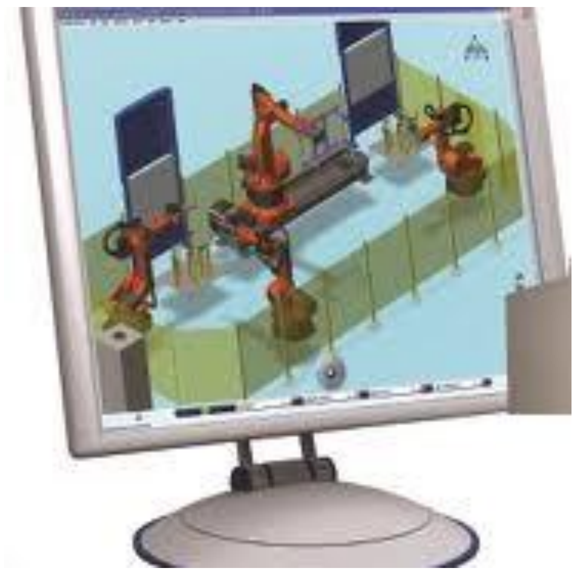


Figure 2.1. ***Digital model of a production line in a PC. [modified from 3ds.com]***

3D visualization of production lines can be implemented in a number of applications. Below is presented a brief review of some possibilities offered by this technology.

2.1.1 Applications

Design Validation

The building of digital models allows companies to examine and validate a product design or manufacturing solution while it is being developed at a lower cost than other alternatives. [Agrusa et al., 2009]

In addition to the validation of assembly lines design, 3d visualization of production lines allows to validate procedures or methods. When building a 3D visualization of an operational procedure or working method, the designers can verify and validate them. The 3D visualization provides can also help to optimize the operation methods in order to reduce wasting times or operations. The 3D visualization of production lines goes beyond other tools like diagrams or graphs representation of the line operations provide much more information and in a clearer way, and therefore, troublesome issues can be more often find and corrected for the procedure validation. [Beuthel et al., 2002]

3D virtual models of an assembly line provide a cost-effective way to validate design ideas and to accelerate the development process of a product. The cost of virtual prototypes is far less than real mock-ups, and its flexibility and reusability are much higher.

They are also much more reliably than 2D representations of line layouts, or graph based models of the factory process.

At the same time that is designed, and before the final physical production line is installed, a digital prototype of it allows manufacturers reviewing its real performance. The original design can be created, validated and optimized since the first step of the system development in a 3D virtual model. Its performance and characteristics can be visualized in advance to its implementation to make sure that it achieves the initial design specifications before the final physical version of the system is implemented. [Agrusa et al., 2009]

Not only can the production line design be validated, but also operational procedures or manufacturing methods. They are patterns/guidelines of what to do and how to do it. Before deciding to execute them in the manufacturing process, they can be validated in a 3D digital model of the factory.

Furthermore, a completely new manufacturing process or modifications of an existing one can be validated in a 3D virtual production line.

Generally, any design, operational procedure or manufacturing operation of a production line can be validated in a 3D virtual model with great advantages. If the production line has not been implemented yet, they can be validated in advance, saving valuable time in the development process. Had the production line been a reality, validating through a 3D digital model would still offer the advantage of not needing to make use of the resources of the production line, with the cost that it means.

Simulation

Some decades ago, 3D simulation of manufacturing processes started to be a standard in the industry. In the 1980es car manufacturers started applying it mainly for robotics workcells.

Robotics was the main application for 3D simulation. This was due to the fact that robots are complex 3D resources and concepts like robot reachability or resource sharing are very important in the domain of robotics workcells, Therefore 3D simulation is a very valuable solution in the development process of every production process where there are robotic workcells involved. Later on, with the introduction of robots offline programming, the power of robotics 3D simulation had a new boost that turned it in almost an indispensable tool for every medium-size manufacturing company.

3D production line simulation has a number of applications. First of all, it allows estimating space requirements. In addition, assets movements can be analysed in detail, as well as the workflow of products or resources. Besides, collision and reachability of robots and other machinery can be studied easily. Furthermore, joints speed or acceleration and other equipment movement parameters can be tested.

3D simulation provides several advantages. For instance, it allows analysing risky scenarios that could damage the real system if they were analysed in the real system. Also, it makes possible to visually identify errors that other way would not appear until the implementation phase. One other advantage, common to any computer simulation is that it is possible to simulate any operation at a much faster pace. Therefore, long duration processes can be simulated in seconds. For instance, conveyors, robots and similar device can work at speeds much higher during the simulations. Obviously, offline simulation does not disturb the real manufacturing line so that it can go on operating in its usual way, as it was previously said about operations validation. Finally, different layouts and their consequences can be easily tested in a 3D simulation.

Figure 2.2. ***Robot simulation with DELMIA software. [3ds.com]***

As examples of simulation tools, we can mention DELMIA, by Dassault Systemes, 3D Create by Visual Components or Enterprise Dynamics.

These tools provide an environment to build a digital factory and perform a number of simulations.

First of all, the factory layout can be simulated. A 3D model of the factory can be implemented building the 3D equipment models in detain or using predefined models provided by the software, depending on the level of detail that it wants to be reached for the visualization and the time available for it. Once the model is built, the software allows the designer to use different views or walk or fly through the factory environment to inspect it. The mock-up arrangement can be modified as many times as the designer wants till the final solution is achieved.

In addition, systems and processes in the factory can be simulated. For instance a 3D simulation enables working on material flow optimization. Furthermore, 3D simulation is particularly relevant in Flexible Manufacturing Systems (FMS) or in processes involving animated resources. One of these cases is robotic workcells, in which offline programming environments can be integrated with simulation to test the robots programs.

Last but not least, simulations tools usually include the ability to integrate human models in the simulations in order to analyse manual operations, workers safety or ergonomics.

Training Environments

3D visualization is the main and vital component of training environments. The building of a 3D virtual replica of a real working environment, such as a production line, in order to carry out workers training offers many advantages.

For instance, the workers can assimilate in an easier way the working conditions, reducing the workers training period and improving they future performance in the real production line. [Beuthel et al., 2002]

Besides, without a 3D training environment, the workers training needs to be executed in the real production line, making impossible that it is available at the same time for product manufacturing, and therefore involving a loss of production capacity.

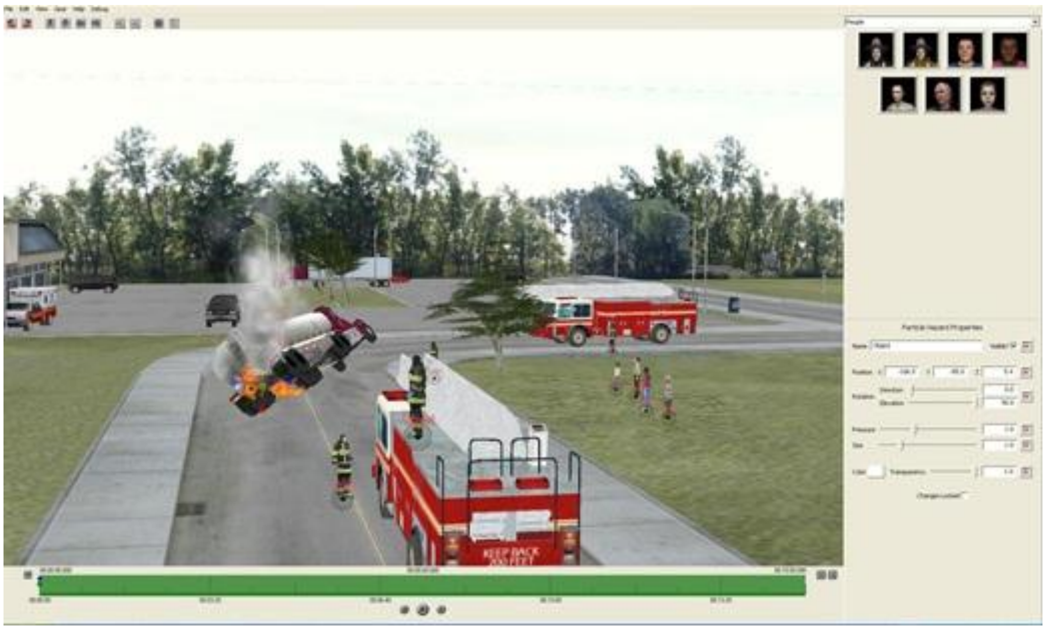


Figure 2.3. ***Code3d virtual training for emergency responders. [panda3d.com]***

Another advantage of 3D virtual training environments is that the workers training can be performed at any place. Not needing the real production line, a computer and the 3D training application are enough to accomplish it.

Furthermore, in those hazardous working operations that need it, a 3D training environment can be used to perform a last-minute review of the methods and operation that shall be carried out by the worker, minimizing the risks of the process.

As a result, it can be said that 3D animations can be created simulating any working environment, such as factories, inaccessible locations or machinery, or hazardous conditions for the training of working personnel, reducing costs and improving the results when compared to real conditions or other simulated training.

Examples such as CliniSpace™ show the potential of 3D virtual training environments. CliniSpace™ is a training environment for healthcare staff that offers a reliable 3D reproduction of the usual work scenario for the practice and learning of medical teams.

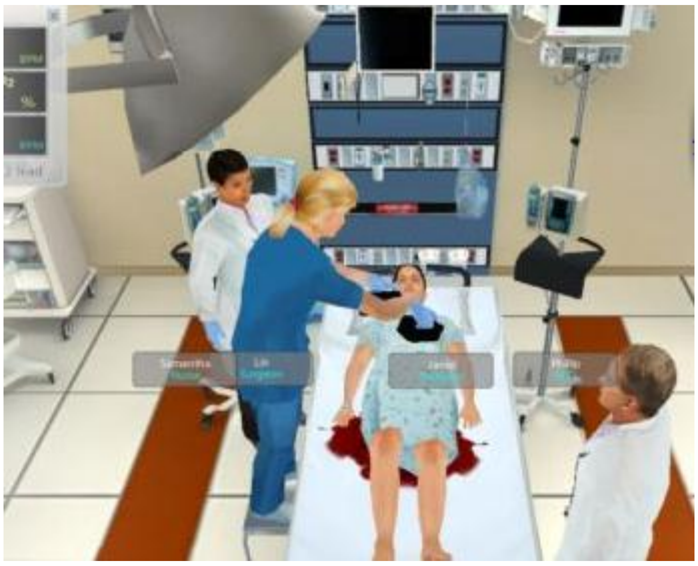


Figure 2.4. ***Clinispace learning environment with Unity. [clinispace.com]***

Communication and Marketing

3D visualization increases the repercussion and influence of any design proposal. Companies may raise their marketing success of a product or service by presenting 3D good-looking visualizations of it. A high-quality well-rendered animation of a process or object empowers the company's corporate image and makes their solution much more appealing for the client.

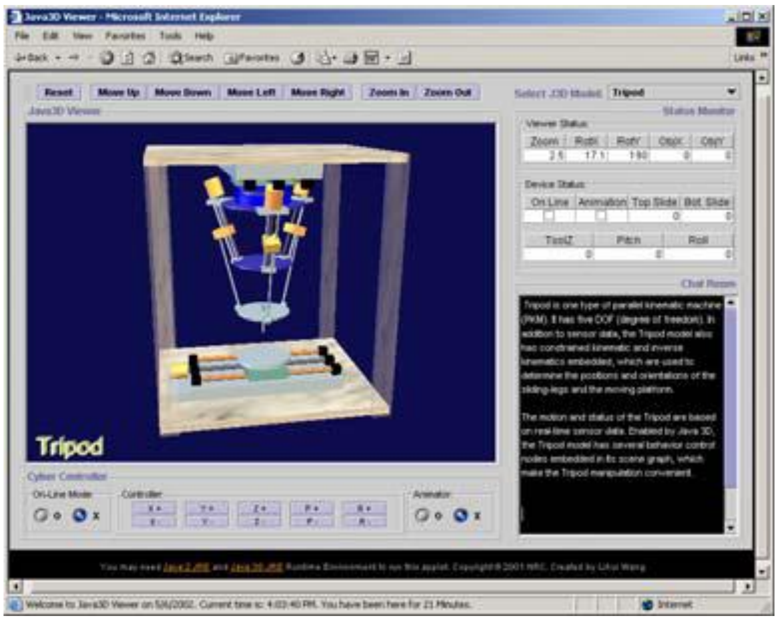
Equipment and devices sellers can show how the product or the service will work integrated in a production line, providing a realistic simulation of the final solution and visualizing how it would benefit the client.

In addition, manufacturing companies can show their production line processes to potential clients. They can provide a visualization of all the operations that would be executed and how they would work in order to get the quality required by the client. When a 3D visualization is provided, many doubts about the provider technological and management abilities and success probabilities are dispelled.

Also handling equipment companies take the most of 3D line visualizations ”3D simulations allow us to see how our tailored systems will function, before we actually build them” says Janne Konttila, Export Manager of Orfer Oy, accompany providing packaging and palletizing solutions to the food industry, ”For us, simulation is primarily a marketing tool. We have noticed that simulation helps us make things concrete for the customer on a completely new level; it's much easier to convince the customer when he can see a simulation of their new solution with their own eyes,” [visualcomponents.com 2012]

3D Monitoring Systems

Another application of 3D visualization of production lines, and the one that constitutes the topic of the present thesis is the building of 3D monitoring systems.



Tripod is one type of parallel kinematic machine (PKM). It has five DOF (degree of freedom). In addition to sensor data, the Tripod model also has constrained kinematic and inverse kinematics embedded, which are used to determine the positions and orientations of the sliding-legs and the moving platform.

The motion and status of the Tripod are based on real-time sensor data. Enabled by Java 3D, the Tripod model has several behavior control nodes embedded in its scene graph, which make the Tripod manipulation convenient.

Copyright© 2001 NRC. Created by Lihui Wang

Figure 2.5. ***Web-Based Remote Manipulation and Monitoring. [Zhang&Wang, 2005]***

As this application is the one of this thesis proposal, further and more detailed discussion is carried out in Section 2.4.

2.2 3D Computer Graphics

It is commonly accepted that 3D computer graphics started during the 1960s with the development of Ivan Sutherland's Sketchpad Software as part of his Ph.D. Thesis at MIT.

Since then, computer graphics evolved in line with computer processing capabilities. Only large corporations, mainly from the aerospace, defence, aircraft and automotive industries, could afford the cost of the technology at that time. CAD (Computer Aided Design) software was developed by the companies themselves as in-house solutions to match their own design requirement. [Bertolini et al., 1995]

As technology became accessible, CAD software spread to smaller industries and the business market grew. Therefore, CAD solutions development was outsourced from manufacturing companies building new business units. CAD developers turned into large corporation and CAD software included more functionalities as the computing capacity of workstations rose.

The decade of 1990s view a parallel development of 3D computer graphics, as 3D they spread widely both in the fields of gaming and multimedia.

However, because of its high computational cost and complexity 3D computer graphics have not arrived yet to all domains of engineering. One of those application fields which have remained reluctant to its adoption is the one of the monitoring systems, as it has already been stated.

Figure 2.6. ***Sketchpad at Massachusetts Institute of Technology. [mit.edu]***

In practice, two main different technologies exist for the creation of 3D models. On the one hand, Computer Aided Design (CAD) aims to the creation of engineering-purpose models. On the other hand, Digital Content Creation (DCC) or simply “3D modelling and animation” aims to the creation of artistic outcome.

We say there are two technologies because CAD and DCC differ in the use of different tools (software), methods (modelling procedure), and tasks (engineering product design and artistic creation).

Visualization and rendering quality of CAD software is not as high as those of 3D DCC tools, on the other hand CAD software features functionalities that are out of the scope of 3D DCC software aimed to engineering product development.

These two different 3d modelling and developing environments do not live one isolated of the other. For instance, most products are designed in CAD for its production, but they are also designed with DCC software for its marketing.

Productivity prays for the integration of both technologies in order to realize a smooth collaborative environment of the product development stages and teams.

However, this is not always achieved nowadays, and formats, software and files inconsistencies are still common in many companies.

For instance, many software applications dedicated to the building of HMI (Human Machine Interface) applications lack integration with CAD software. Hence, designers and engineers do not work in contact with each other, duplicating tools and work, decreasing flexibility and increasing development times and costs. [Agrusa et al., 2009]

2.2.1 3D Modelling

In computer graphics, a 3D model is a mathematical representation of an object. There are several methodologies for the building of this representation, leading to different 3D modelling tools and technologies.

There are mainly three different types of geometrical modelling that are presented below.

Line or Wireframe Modelling is the simplest form of geometrical representation. The object is defined by edges and vertices, although fine for 2D draughts, resultant 3D models are highly incomprehensible.

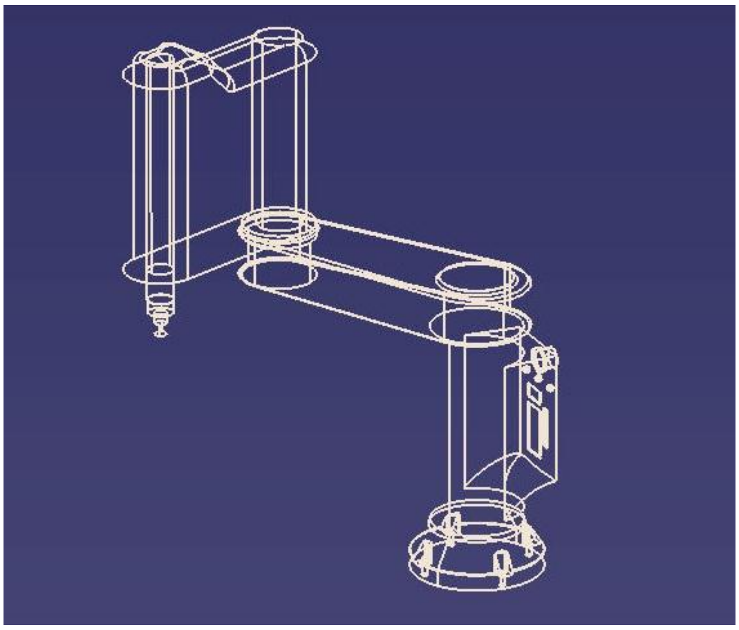


Figure 2.7. ***Wireframe model of a SCARA robot.***

Surface Modelling uses surfaces, in addition to vertices and edges for the definition of the model, it is particularly useful to build organic models or objects such as body car or aircraft panels, ship propellers or fan blades.

It provides a unique and non-ambiguous visualization of the object, something that does not happen with wireframe modelling. But mainly, it enables the designer to render or shade the object surface in order to create a realistic and understandable visualization of the model.

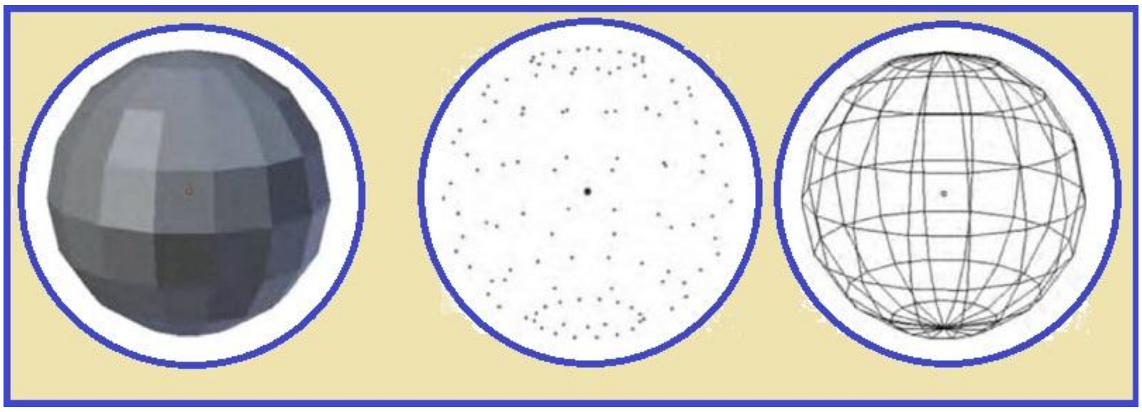


Figure 2.8. ***Sphere simple surface model and its vertices and edges components. Modified from [Flavell, 2010].***

One of the most common implementation of Surface modelling is the definition of a parametric surface by Bezier curves, cubic splines, B-splines or NURBS (Non-Uniform Rational Basis Spline).

Solid Modelling adds also the mass between the surfaces to the definition of the object, thus including volumetric information to the object’s model. There are two types of methods for solid modelling.

B-Rep (Boundary Representation) method defines the objects volume by its borders. The boundary is a collection of surfaces linked between them that define the limit between the solid volume and the outside. Operations in B-Rep method include extrusion, chamfer, blending or drafting.

CSG (Constructive Solid Geometry) uses primitives and Boolean operations to combine them in order to model the real object. Boolean operations include union, difference or intersection.

2.2.2 CAD (Computer Aided Design)

CAD stands for Computer-Aided-Design. CAD software enables to use computer to carry out the design process of a product.

CAD built models can be employed to visualize an idea or imaginary design, to build a digital prototype of a design to find errors or to perform simulation of the design product. In addition CAD models are used as the interface between the design and the manufacturing of the product, drafts or CNC code in CAM applications can be obtained from the CAD software for the product manufacturing.

<u>CAD Modelling</u>

CAD software is conceptually made up of a GUI (Graphical User Interface) that interacts with mainly B-Rep models through a geometric modelling kernel.

B-Rep method offers more flexibility, power and potential than constructive solid geometry for the representation of 3D models. Although CSG was more easily implemented initially, nowadays all CAD geometric modelling kernel are mainly B-Rep based. Examples include Parasolid by Siemens PLM Software, or ACIS and CGM by Dassault's subsidiary, Spatial.

Hybrid modelling

Premium CAD software tools include freeform surfacing, that enables creating more advanced surface designs complementing solid B-rep modelling.

Most widespread method for freeform surface definition is NURBS because of the flexibility and accuracy that this method offers.

Parametric modelling

CAD software features parametric modelling, which is a methodology that uses parameters to define a model.

The advantage is that parameters can be modified at any stage of the design process updating the whole model design very easily. For instance, parameters can be dimensions or relationships between features or parts, if an error in the design is detected in a dimension, that parameter can be modified and the whole model is corrected without needing to build it again from scratch or re-working on the design model.

Parametric modelling is a very powerful tool, but requires having a structured and well planned model, because modification of one parameter can affect others.

History edition

Most common CAD software applications build a feature history tree, recording all parameters and relationships of the design in the order that they were executed.

This is used as a modelling operations procedure, so that any change in any of the tree recordings will update the model.

2.2.3 DCC or Virtual Reality

DCC stands for Digital Content Creation and it is a term used to call the development of computer 3D graphics, animation, audio, video, images … 3D graphics software can be used to develop animated films, 3D games or other interactive 3D applications.

DCC software applications are aimed to game or artistic developers; therefore their user interface is aimed to provide flexibility and intuitive tools for the designer. Because artistic designs are based on intuition and ideas, they require fast, immediate functions to model them.

A modelling method that requires constraints or a strict structure is not feasible and coherent with the means of the designer. Tools to build the model are focused on the definition of the model appearance and not in engineering parameters like constraints, dimensions, mass or tolerances.

Differences with CAD software also reach the software modelling architecture. Because the computational cost of solid modelling is too high compared with surface modelling and its accuracy and other advantages are not needed, DCC modelling is based on surface modelling.

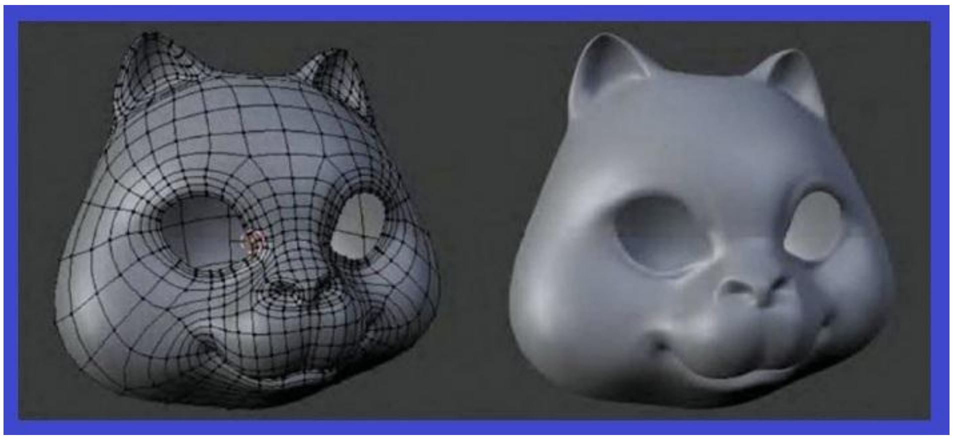


Figure 2.9. ***3D DCC modelling example. [Flavell, 2010].***

There are several ways to implement this surface modelling methods. A way to define the surface is needed and the most common technique is polygonal modelling, in which the surfaces modelling the object are represented by an approximation of polygons. These polygons build a 3D mesh. 3D DCC software also uses other methods like NURBS or subdivision surfaces. NURBS are interesting for organic model or complex surface models; however, they are slower when they have to be renderer.

In addition to build the model, 3D DCC technology includes other aspects like lighting or texture definition.

Lighting is defined by the scene camera and the lights position. The purpose of the lighting has to be showing the scene so that it can be wholly understand in a reliable way, thus extreme white and black lighting areas are to be avoided, and to give depth. In addition the lighting has to provide the capability of giving depth to the built scene through shading.

Textures assist to establish how a certain 3D model looks like. Textures affect the object surface characteristics, like colour or bumpiness.

The texture of an object can be defined through procedural textures or UV mapping. A procedural texture is carried into effect by an algorithm implemented by the 3D DCC software tools. UV mapping defines the texture manually by means of unwrapping the surface of the 3D model and attaching a predefined image to the corresponding unwrapped surface. The result is a more detailed and accurate texture, which is translated into a better model appearance.

2.2.4 CATIA

CATIA is a software suite in the 3D PLM (Product Lifecycle Management) domain. It provides solutions not only for CAD, but also for the whole product lifecycle, integrating CAM (Computer Aided Manufacturing) or CAE (Computer aided Engineering) functionalities.

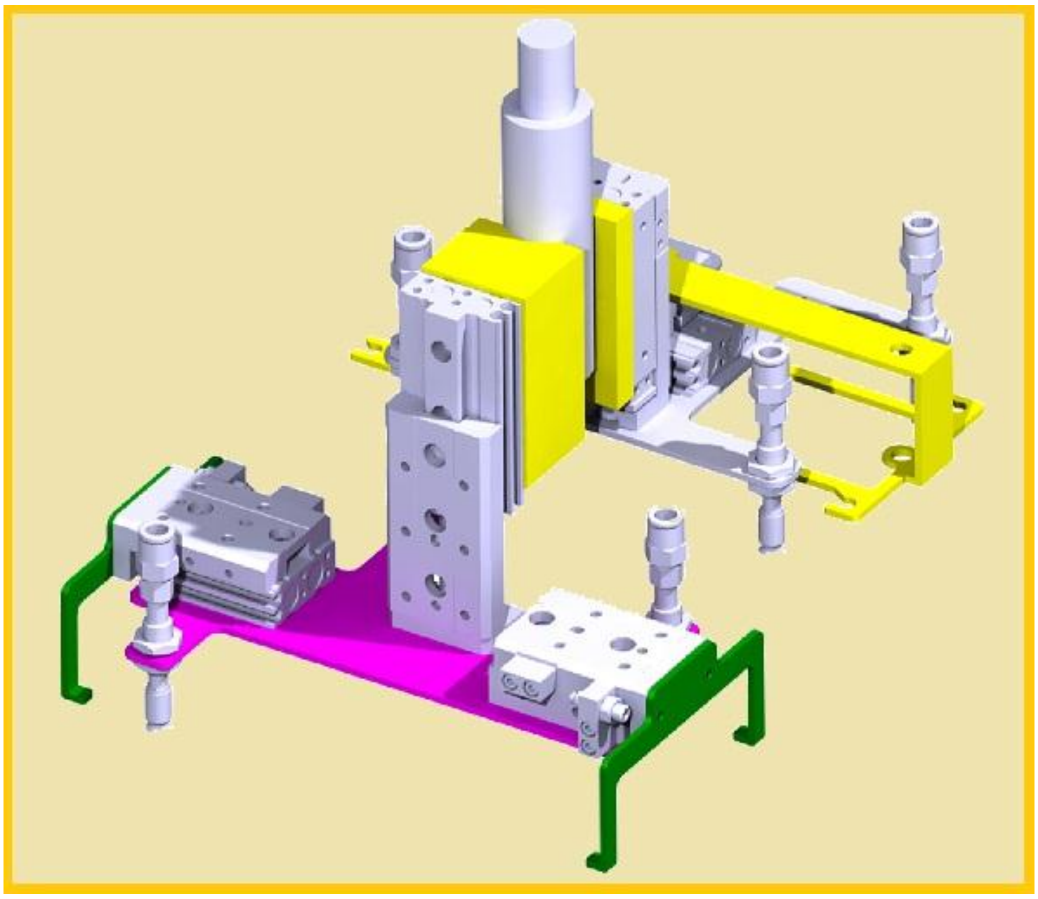


Figure 2.10. ***Robot gripper modelled in CATIA.***

CATIA development was started by French aircraft manufacturer Dassault Aviation at the end of the 1970s. In 1981, Dassault Sytemes was created as a new branch of Dassault group. Its purpose was developing CATIA software as a solution for different industries and as an independent business, an agreement was signed with IBM for its sale and support worldwide. The first customers to adopt CATIA were car and aircraft companies (Daimler-Benz, BMW, Snecma or Grumman). In 1986 Boeing choses CATIA as its CAD/CAM solution, which supposed the main boost for the software popularity, In 1993, Version 4 is launched, featuring Parametric design. In 1997, the acquisition of Deneb causes the creation of Delmia, addressing the production and manufacturing industries. In 1998, CATIA Version 5 is launched, being a complete rewritten version of CATIA and featuring a new architecture to support Product Life Management (PLM). In 2008, CATIA V6 appeared focusing on collaborative tools for PLM.

CATIA Part Design

CATIA Part Design is the workbench dedicated to the mechanical design of single parts.

It provides all the tools for creating the desired part. The usual modelling process starts by building a 2D sketch and give it 3D volume through operations of extrusion. Pockets, shaft, groover, rib, stiffener and multi-sections solid are other modelling functions.

In addition, functions like chamfer, edge fillet, draft angle, shell or thread enable the designer to detail the model. Points, lines and planes can be created as auxiliary geometric entities.

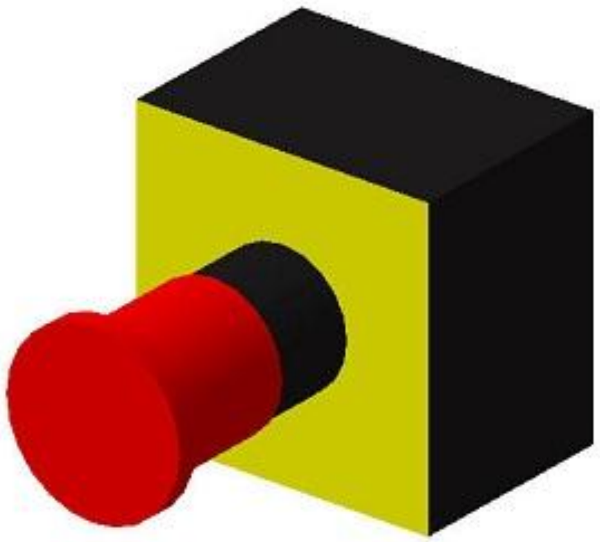


Figure 2.11. ***Simple emergency stop button part designed with CATIA.***

Constraints allow establishing the elements dimensions and relationships between elements.

Besides Part Design, CATIA offers other workbenches for parts modelling like Wireframe and Surface Design, Drafting or Sketching.

CATIA Assembly Design

Several parts may be gathered together to form an assembly. CATIA Assembly Design workbench has the necessary functionalities to relate the parts and define the assembly.

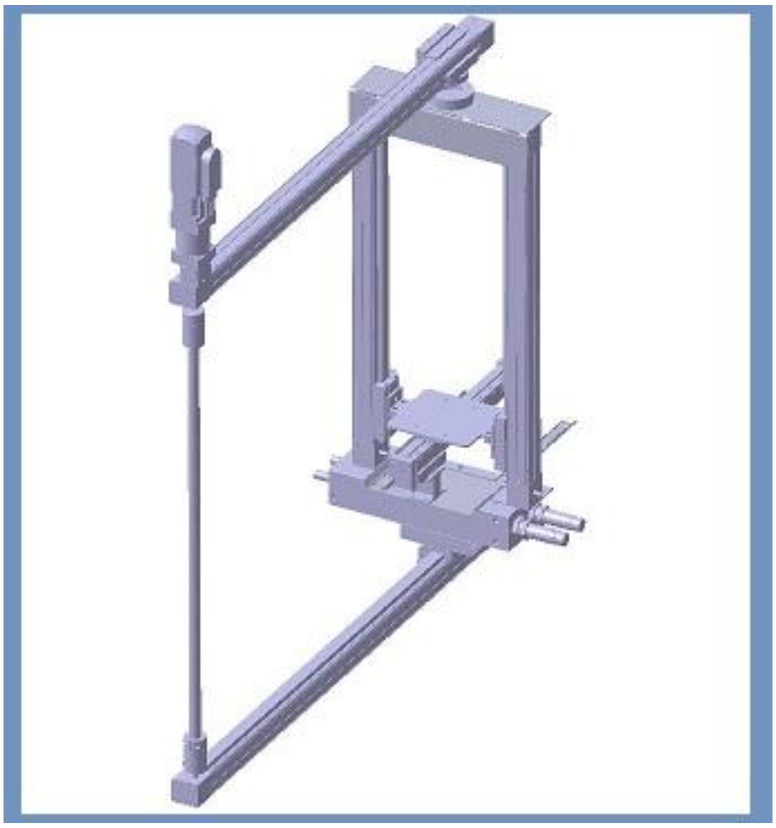


Figure 2.12. ***Buffer robot assembly designed with CATIA.***

CATIA files export

CATIA files can be exported to a number of different formats. As the more relevant ones, we can mention STEP and IGES as CAD format standards, STL for Rapid Prototyping or VRML and 3dxml.

CATIA Geometric Modelling Kernel

CATIA V5 introduced a new modelling kernel, the so called Convergence Geometric Modeller. It was released in 1999, and it meant a major innovation in CATIA series of products.

It is a B-Rep (Boundary Representation) modeller; it builds b-rep geometry and topology model. It features wire, surface and solid modelling functionalities. It supports both history-based and direct modelling editing.

Since 2011 it is not the modelling kernel exclusively for CATIA anymore, as it is licensed to other CAD vendors and application developers by Dassault Systems child subsidiary Spatial, who already created ACIS modelling kernel in 1989 and licensed it subsequently.

2.2.5 Blender

Blender is a 3D computer graphics software with applications in the area of 3D design, animations and visual effects creation or interactive 3D applications such as video games. In words of the own Blender Foundation, “Blender is the free open source 3D content creation suite, available for all major operating systems under the GNU General Public License.” [blender.org, 2012]

Blender features 3D objects modelling, rendering, UV unwrapping, texturing, video editing and a physics and a game engine. It is open-source and one of its strong points is its large amount of users that make up an online support community for development and resources.

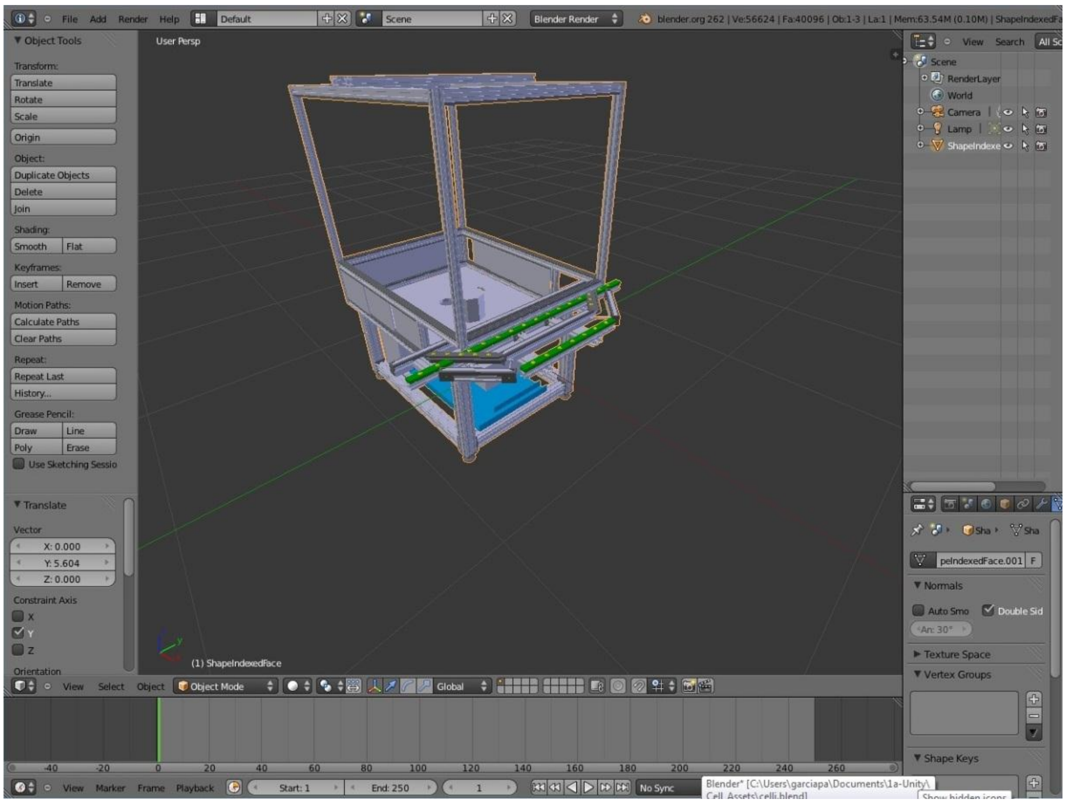


Figure 2.13. ***Blender Graphical User Interface.***

Blender software package was an in-house solution of NeoGeo animation studio. When the studio went into financial trouble, they sold the software to the Blender community for the price of 100 000 euro. Then Blender was released as open source software with a GNU license.

Since that moment, its user community has not stopped to grow and the software has gone on developing and adding new features and improvements.

Nowadays Blender constitutes a real alternative to other 3D DCC software packages such as Autodesk Maya, Autodesk 3ds Max or NewTek LightWave.

Blender is considered to have a slow learning curve and being a complex software to use. One of its peculiarities is that it includes a keyboard shortcut for most of its functionalities, which usually makes the users disorientated when starting to use it.

Blender Modelling:

Blender models are built up by a primitive mesh and stackable modifiers that alter and modify the original mesh in non-destructive operations.

Blender supports a variety of 3D object types including polygon meshes, NURBS surface or Bezier and B-spline curves. It offers as well mesh modelling based on vertex, edge and/or face selection and editing.

Python scripts may customize tools and gadgets.

Blender Files

It is common that Blender users have to work in an environment in which data interchangeability is essential. It is quite common that Blender is not the only software solution used in a project, thus it includes the ability to work in the following formats.

- .blend files, as the native file extension, also used for images, sounds or fonts. It also supports compression, digital signatures, encryption, forwards/backwards compatibility and can be used as a library to link to from other .blend files
- Import and export capability to a number of formats such as 3D Studio, COLLADA, FBX Export, DXF, Wavefront OBJ, DirectX, Lightwave, STL or VRML.
- support for other 2D formats like JPG, PNG, AVI and Quicktime GIF, or MOV (Windows and Mac OS X)

2.3 Assets Situational Awareness and Monitoring

In manufacturing companies, assets visual monitoring offers control through all the production process. This contributes to improve the factory control, making easier to take real-time decisions such as route planning or robots operation type in Flexible Manufacturing Systems (FMS).

Manufacturing assets position is particularly favourable when a manufacturing process has travelling assets, for instance pallets moving through a production line.

If an assets monitoring system is implemented in such a line, when some problematic issue or malfunction appears, the assets location is known and immediate trouble-solving decisions can be taken in order to recover the original state of the factory. In addition, assets monitoring allows finding out the past locations and working conditions of the assets so that the failure causes can be identified.

Finally, assets situational awareness contributes to production tracking and tracing.

2.4 3D Monitoring Systems

Given the benefits that 3D monitoring systems offer when compared with 2D visualizations, several attempts to efficiently build 3D monitoring systems have been done.

Most widespread 3D monitoring systems can be classified under two categories.

On the one hand, Java3D offers a 3D graphics API (Application Programming Interface) running on the Java platform over either OpenGL or Direct3D. Any real object and its behaviour can be defined and modelled with Java3D, and then it can be driven by sensor data coming from the real device. [WANG 2011]

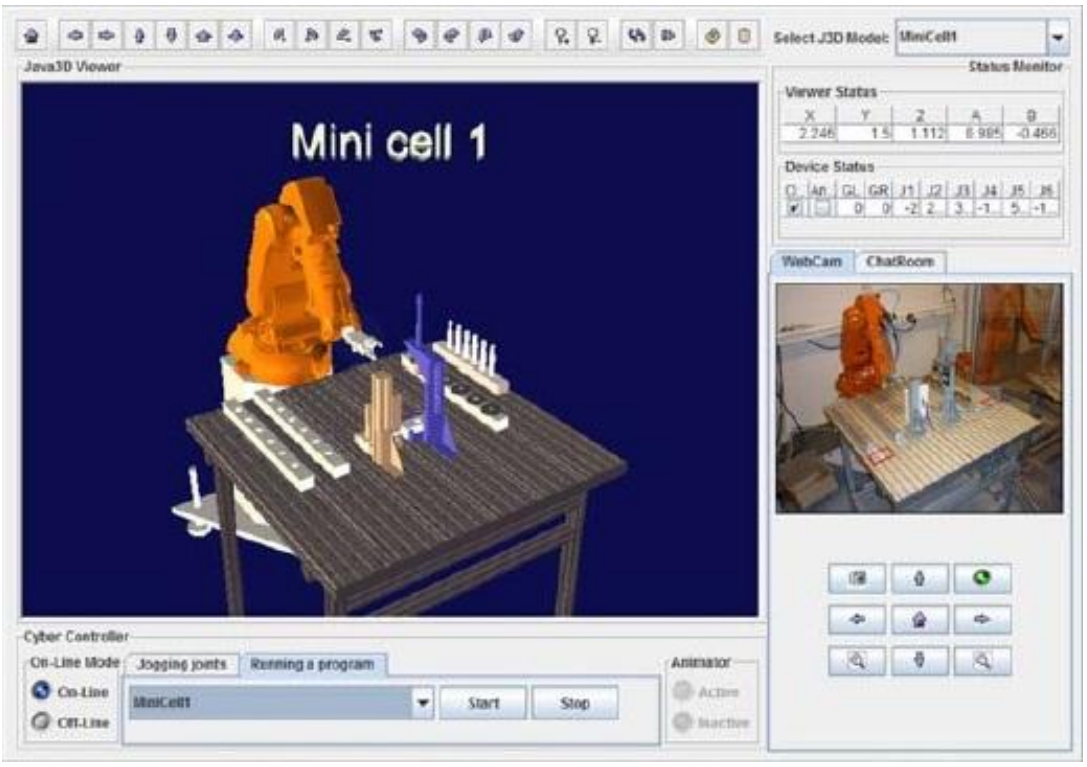


Figure 2.14. ***A Java3D monitoring system, [Wang, 2011]***

Although valid, this approach adds complexity to the development process. This meaning that cost is increased and flexibility and modularity are reduced till the point of affecting the system viability.

On the other hand, video monitoring systems use a camera, or a number of them, to constantly record a view of the production line that can be broadcasted to any computer or similar device.

Bandwidth is usually a critical parameter in these systems. When the monitored system is vast or complex as to require the installation of several cameras, like for instance a normal production line, bandwidth consumption is dramatically increased and access to the whole system visualization through the Internet cannot be guaranteed. Even in the case of one camera systems, quality of image is not usually as reliable as needed.

The increase of the number of cameras entails other troublesome issues such as system cost or maintenance.

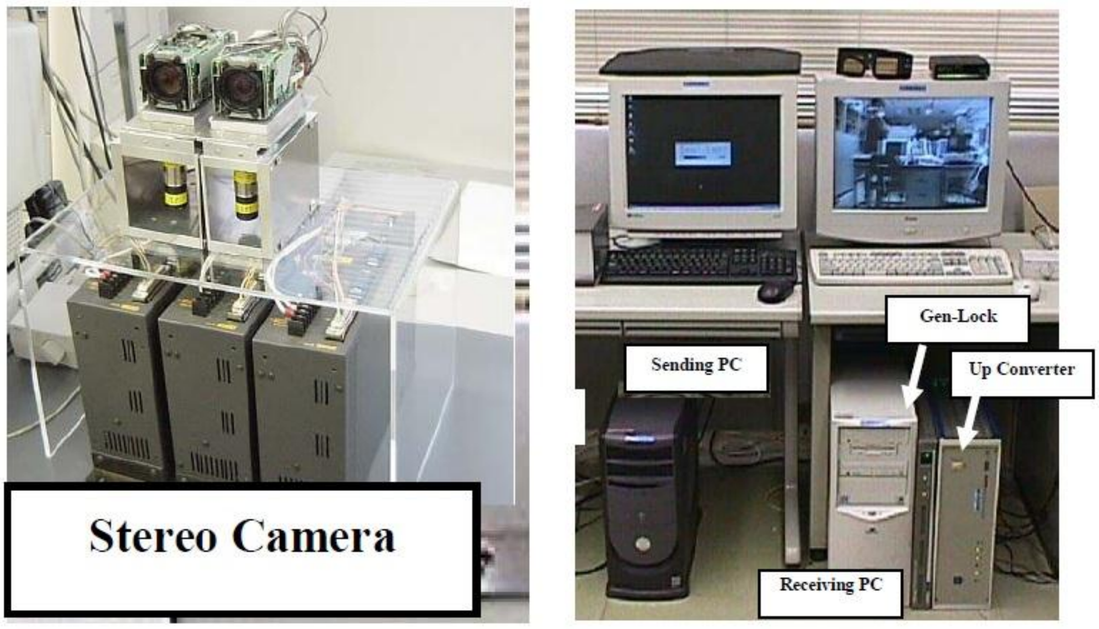


Figure 2.15. ***A camera Remote Stereo Vision monitoring system, [Takeno et al., 2008]***

Further application fields

Besides manufacturing, there are other fields of application for 3D monitoring systems.

As it was already previously stated in Chapter 2.3, assets monitoring is particularly relevant in those processes in which the assets are moving or travelling along the facility and their position is decisive in order to ensure the proper operation of the system. Thus, 3D monitoring systems have applications in airport or harbour traffic and operations monitoring, control and management area.

Asset awareness is also important in open environments, and 3D monitoring systems combined with GIS (Geographical Information System) are useful for the monitoring of staff or vehicles working in outdoors tasks.

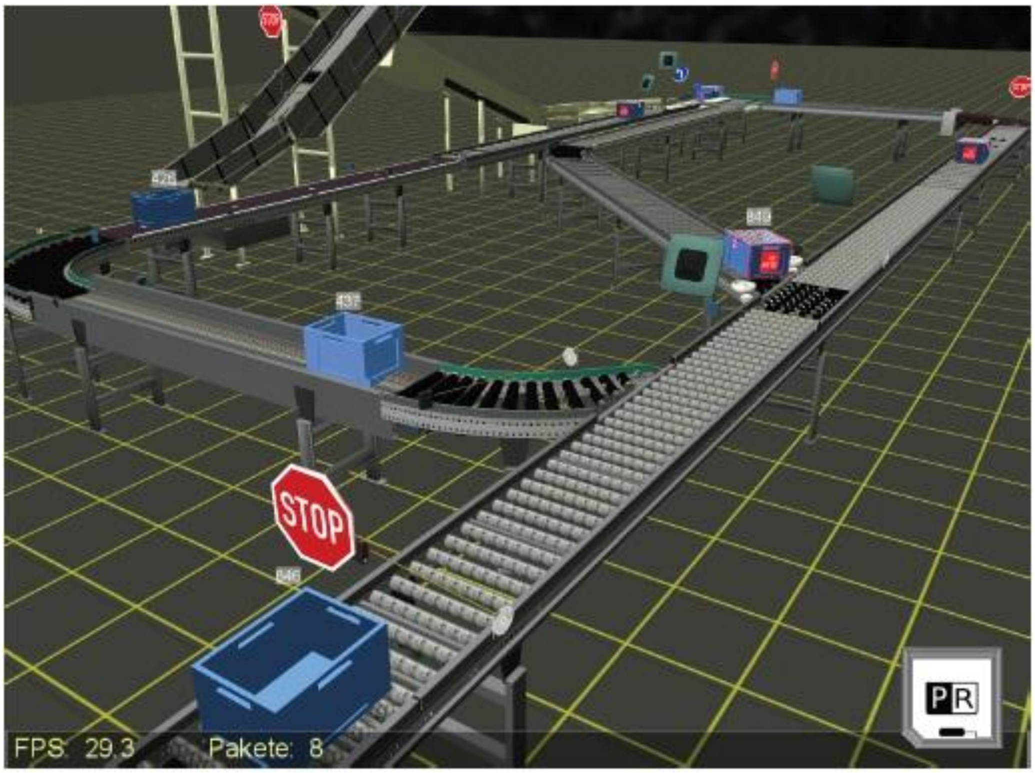


Figure 2.16. ***3D monitoring system of material flow. [Feldhorst et al., 2002]***

In addition, 3D monitoring systems are especially advantageous in those cases in which a 3D graphical representation is indispensable to visualize the system arrangement. For instance building monitoring is part of every intelligent building system in the new developing field of building automation. In that case, a 3D virtual model of the facility provides countless advantages for the monitoring system output interpretation.

2.5 Game Engines

A game engine is a software framework mainly intended for the creation of video games or similar applications. Game engines include, most commonly, a rendering engine for 3D graphics, a physics engine for the simulation of physics laws, and sound, animation, scripting and application publishing capabilities.

How a game engine works may vary a lot from one to another. Some may need to have the 3D models imported from a different 3D modelling application, while others may have 3D objects modelling capabilities or they may even be embedded into 3D modelling and animation software frameworks.

As it could be thought of such a powerful tool, game engines can have many more applications than the only development of a videogame. First of all, 3D games may provide tools to represent and manage any 3D environment. Visualization and rendering tools are also available. Another point not to be misled is that most game engines are prepared and can be adapted to communicate and to integrate with other software and development tools. [Harrop&Armitage 2006]

One of these possible non-game applications is using their graphics and physics engine for the simulation of engineering applications, although they are not as accurate as specific purpose technical software, the results can be acceptable depending on the application.

Physics engine may be useful for the creation of engineering simulations, if accuracy of results is inside the acceptable tolerance of the given problem.

Another application out of the ordinary for game engines relies in the simulation of virtual training environments. For instance, flight simulators can be used as training applications for the pilots to get familiar with the flying environment and the physics reactions of the plane. In this same field of application, 3d animations can be created simulating any working environment, such as factories, inaccessible locations or machinery, or hazardous conditions for the training of working personnel. [Craighead 2008]

Last but not least, since it constitutes the topic of the present thesis, another possible application is the building of factory visualization. Within a game engine it is possible to model all assets and products of a factory as well as animating them. Thereby, if the information shown in the application is selected with control and supervision purposes and periodically updated, 3D visual monitoring systems can be produced. If communications protocols allow it, real-time specifications can also be achieved.

2.5.1 Unity 3D

As stated by its developer, Unity Technologies, “”Unity is a feature rich, fully integrated development engine for the creation of interactive 3D content”. [unity3d.com]

The Unity editor run originally on OSX, and could publish desktop applications running on both Mac and Windows. However, in spring of 2009 Unity launched its Windows version and since then Unity definitely turned into a cross delivery platform tool. [Blackman 2011]

Applications developed with Unity may target multiple platforms, such as Mac OS, Windows, Web Player, Android or iOS.

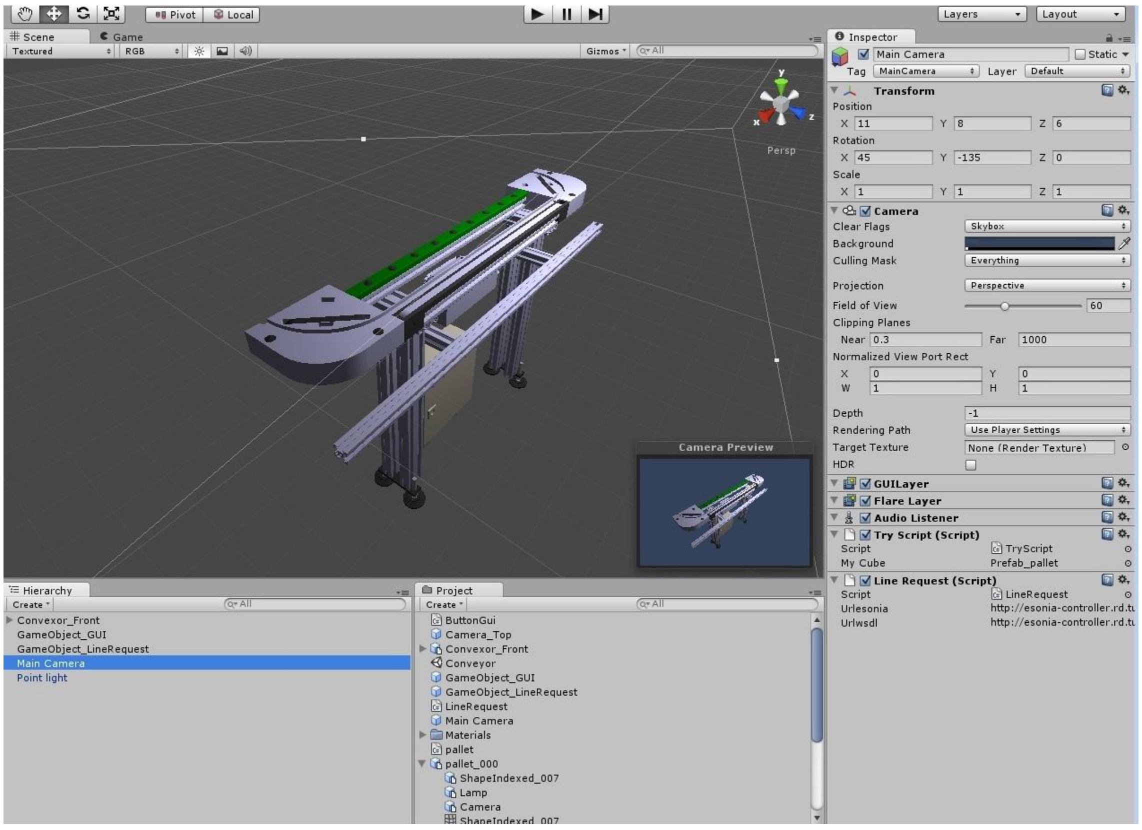
Unity editor is focused on virtual world building tools, featuring an asset-oriented architecture that improves the application building process. Every component or asset of the application has a virtual representation as an object and can be manipulated in the editor. Likewise, building a Unity application basically consists in creating a 3D scene in a similar way to a 3D DCC application and managing it through programming scripts.

Unity provides a complete documentation for the applications development. It includes a user manual, examples and tutorials and reference guides. Unity also has a large community of online active users that provides support through different online tools, like a wiki, forums and UnityAnswers site.

Another of Unity advantages is that it is a low-cost solution when compared with other game engines [Craighead et al., 2008]. This makes an ideal balance for users between features and functionality with purchase cost. [Blackman 2011]

<u>Unity Editor</u>

Unity's editor is undoubtedly the simplest and effortless to use among the most common widespread Game Engines. It features a hierarchical tree view and a drag-and-drop handling of game objects that facilitate the application building by means of providing visual tools. [Craighead et al., 2008]



```
Main Camera                         Static
Tag  MainCamera        Layer  Default
Transform
Position   X 11    Y 8     Z 6
Rotation   X 45    Y -135  Z 0
Scale      X 1     Y 1     Z 1
Camera
Clear Flags               Skybox
Background
Culling Mask              Everything
Projection                Perspective
Field of View             60
Clipping Planes
  Near 0.3                Far 1000
Normalized View Port Rect
  X 0                     Y 0
  W 1                     H 1
Depth                     -1
Rendering Path            Use Player Settings
Target Texture            None (Render Texture)
HDR
GUILayer
Flare Layer
Audio Listener
Try Script (Script)
  Script                  TryScript
  My Cube                 Prefab_pallet
Line Request (Script)
  Script                  LineRequest
  Urlesonia               http://esonia-controller.rd.tu
  Urlwsdl                 http://esonia-controller.rd.tu
```

```
Hierarchy
Convexor_Front
GameObject_GUI
GameObject_LineRequest
Main Camera
Point light
```

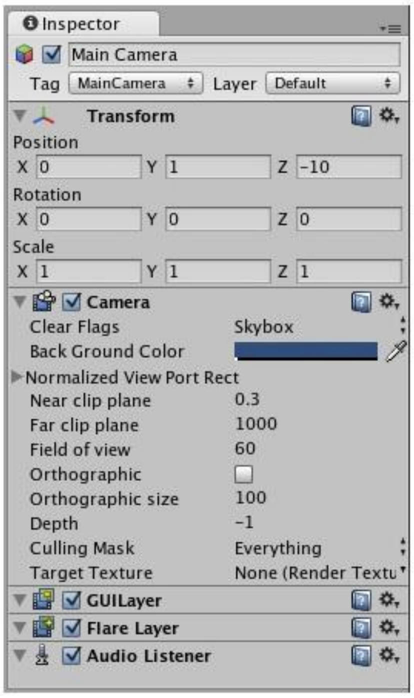
```
Project
ButtonGui
Camera_Top
Convexor_Front
Conveyor
GameObject_GUI
GameObject_LineRequest
LineRequest
Main Camera
Materials
pallet
pallet_000
  ShapeIndexed_007
  Lamp
  Camera
  ShapeIndexed_007
```

Figure 2.17. ***Unity 3D editor view.***

The editor allows scene assets preview, and moreover, the game can be played and tested in editor's mode, including platform emulation.

Every game or application created is organized as a self-contained project that is composed of different scenes, assets, scripts and any other resources needed by the application.

A scene is made up different Game Objects. Game objects can be 3D assets, cameras, lights, etc. Game objects include components, like the transform component to set the game object's position, physics components like colliders or joints, scripts, etc. Every component defines a property or characteristic of the Game Object. They are not a valid entity on themselves, but have to be part of a Game Object.



```
Inspector
☑ Main Camera
Tag MainCamera   Layer Default
Transform
Position
X 0    Y 1    Z -10
Rotation
X 0    Y 0    Z 0
Scale
X 1    Y 1    Z 1
☑ Camera
  Clear Flags            Skybox
  Back Ground Color
▶Normalized View Port Rect
  Near clip plane        0.3
  Far clip plane         1000
  Field of view          60
  Orthographic           ☐
  Orthographic size      100
  Depth                  -1
  Culling Mask           Everything
  Target Texture         None (Render Textu
☑ GUILayer
☑ Flare Layer
☑ Audio Listener
```

Figure 2.18. ***Unity Inspector view of Camera default Game Object***

Unity scripts are Components of the scene, and thus they have to be attached to a Game Object. They can be written in JavaScript-like language (also called Unityscript), C# or Boo (a Python dialect).

Scripts have three different functionalities, defining the Game Objects Behaviour, handling data or creating Graphical User Interfaces.

Unity Publishing

Unity is a multi-platform tool. This means that during the application development process, several different platforms can be targeted. This is possible because Unity abstracts many of the platform differences enabling to target different platforms from one single code. "#ifdef" tag can be used to write platform-dependant code. Unity also allows defining different texture compression and resolution settings for each platform. One step further, the editor is able to emulate all the target platforms, permitting to test the application in them parallelly to their building.

Unity and Unity Pro license allow to create desktop applications for both Windows and Mac, as well as web applications running under the Unity web Player, which works as a browser plug-in similarly to Adobe Flash. [Craighead et al., 2008]

Unity Web Player is available both for Windows (Internet Explorer, Mozilla Firefox, Chrome, Opera and Safari browsers) and Mac OS X (Safari, Firefox, Chrome and Camino) operative systems. [Unity Technologies 2011]

Unity iOS and iOS Pro license allow the development of applications targeting iPad, iPhone and iPod Touch devices.

Unity Android and Android Pro versions permit the publishing of applications running on Android operating system for mobile devices.

Furthermore, Unity also allows to produce applications for Wii, PS3 and Xbox 360 consoles.

2.5.2 Unity Soundness for the development of a 3D Monitoring System

Although mainly aimed to the development of videogames, Unity can have other fields of application as a platform for the development of any 3D interactive content application.

However, the development of a 3D monitoring system, like most engineering applications, has to satisfy a number of requirements.

Thus, its soundness to develop an application like the one that constitutes this thesis proposal must be analysed previously to the application creation.

The matter is whether Unity provides the tools and capabilities to achieve the application performance requirements specified in Section 1.3.

First of all, Unity should allow us to build a 3D virtual representation of the production line reusing the CAD existing models for the line equipment and devices. Although an intermediate application (Blender) is needed in order to convert the files format and carry out some modelling refinement, the models can be moved to Unity editor.

Animation of the models can be done in Unity with the help of scripts attached to the objects that drive the assets dynamical behaviour.

Second, the resulting output of the 3D monitoring system should be accessible and visualized through the Internet. This is easily fulfilled thanks to Unity ability to publish web applications that can be effortlessly executed in the Unity Web Player browser plug-in.

At last, Unity has to allow the monitoring application to communicate with the production line data acquisition system for the retrieving of the events information that drives the monitoring system. Unity Web Player implements some restrictions on accessing data outside of Unity's application when compared with Unity desktop applications. This restrictions affect the use of the WWW class and the .NET API (Application Programming Interface) for the usage of sockets. However, communication with the data acquisition system is still possible and, as it is later shown, it works fine.

2.6 Web Applications

The Web has turned into the preferred medium for data, information and knowledge sharing during the last years. Nowadays, companies based on the web their collaborative tools, so that geographically spread groups that have to work together can reach the same information in real time no matter of their location. [Zhang & Wang, 2005]

Web applications can be accessed all over the Internet or over an intranet network with the only need of a web browser. Main keys for web applications are the fact of being within reach of every far location through the network and that they don't need any software installation by the user, avoiding platform compatibility and updating and maintenance problematic issues.

How they work

When the client visits the relevant webpage, the application is downloaded from the server using HTTP protocol.

The web application is downloaded at every session that is started by the browser client. Thus, the user does not need to fulfil any tasks to update or maintain the application.

Then, the web app is executed in the client browser, in order to make it possible, a web player is usually needed.

A web player is a light piece of software that provides a framework to execute and display the downloaded application. A web player works as a browser plug-in, extending the browser potential without requiring user interaction. Once the web player is downloaded for the first time, it is embedded in the web player and versions updating is usually managed automatically.

Adobe Flash, Java Applets and Microsoft Silverlight are some of the most widespread platforms for web applications. In the field of video games and 3D interactive content, Unity web player also plays an important role.

Web applications seldom run alone, and usually are part of a wider website. One of the most common solutions for web applications publishing is that the web player in which they are executed is embedded in the HTML document constituting the webpage.

In that way, when a user visits the relevant webpage referenced by its URL, the whole HTML document is downloaded to client browser. The HTML document contains a script that loads the web player and the application file hosted from in the server, which is then, as it was previously stated, executed in the client's browser.

Unity Web Player and browser communication

3D applications development with Unity platform has gained popularity during the last years because of a number of reasons. Arguably one of them is its simplicity for publishing web applications.

Unity Web Player features almost 100 million installs and, since it supports Java WebStart, it can be installed with a single mouse click without requiring administrator privileges by the user. Thanks to implementing Native Client technology, it can even run on Google Chrome browser automatically without install. [unity3d.com]

Web applications developed with Unity can communicate with the HTML document in which the web player is embedded. More precisely, scripts of the webpage can call functions defined in the Unity application running inside the web player, and vice versa, functions in the webpage scripts can be called from the Unity code.

2.6.1 Web Documents

Hypertext Markup Language (HTML) is a markup language used to format web documents. HTML elements or tags define the document structure, which is interpreted by the web browser.

Each HTML element has an opening and a closing tag; the text between both tags is formatted according to the tag's browser interpretation or to the related style sheet. HTML 4.0 standardized that the formatting parameters affecting an HTML element's appearance should be defined in a specific style sheets instead. [Schafer 2005]

An HTML document includes a heading section, delimited by the `<head>` tag. It contains the title, meta information, and, most commonly, the document's scripts. The body section includes the proper content of the document.

The HTML document contains the semantic content and structure, but the document's appearance is defined by an external or embedded style sheet file, usually written in CSS (Cascading Style Sheets) language.

JavaScript is a scripting language primarily designed to add dynamic performance to web pages.

Scripts are inserted into HTML pages by using the `<script>` element. The script code can be embedded in the HTML document or referenced from an external script document. [Zakas 2012]

However, JavaScript is not only used in web documents scripting. Since its birth, it has gained popularity and has become a programming language used in other fields of applications. [White 2009]

3. TEST BED AND USE CASE PRESENTATION

In this chapter it will be presented the use case that constitutes the proposal of the current thesis, the development of a web application for a production line 3D real time monitoring system.

First section of the chapter will present the test bed used for its implementation. Test bed is FASTory Line at FAST Lab at Tampere University of Technology.

Later on, we elaborate on the system design requirements of this use case based on the theoretical background and industrial habits formerly presented. In addition, it is discussed the integration of the 3D real-time monitoring system with the line data acquisition system. Finally, there is some discussion about a FASTory Line visual simulation built with Delmia.

3.1 FASTory line

FASTory line is a pallet-based production line made up by ten robotic workstations; eight assembly stations, one loading station and one simple conveyor. The workstations are formed by a conveyor and a robotic cell. Pallets travel all along the conveyors carrying the assembly components and the robots perform the needed operation at each cell.

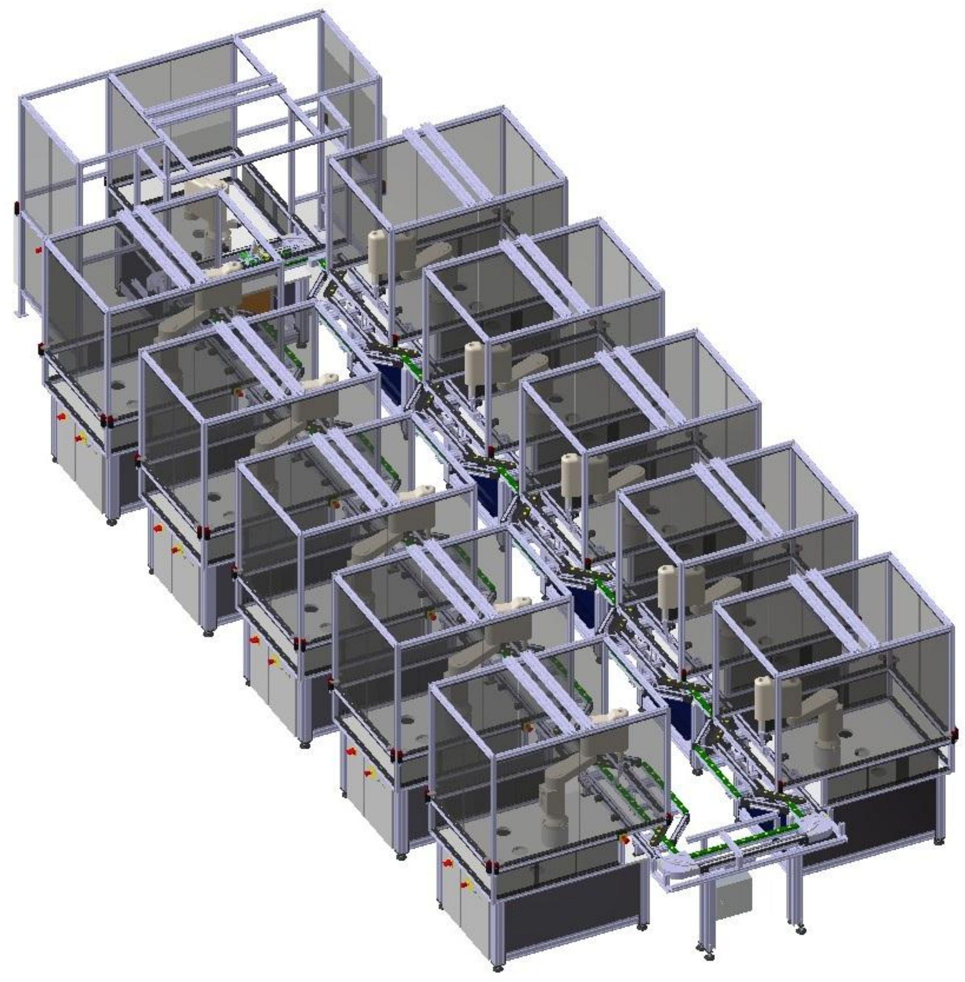


Figure 3.1. ***FASTory Line.***

The workstation arrangement forms a closed loop in a rectangular shape. Five operations workstation are placed in the longer edges of the rectangle, the loading workstation and a conveyor make up the tops of the rectangle.

Pallets travel continuously along all the workstations. When all the product assembly operations are finished, the product is unloaded from the pallet in the loading station and new components are loaded on the pallet.

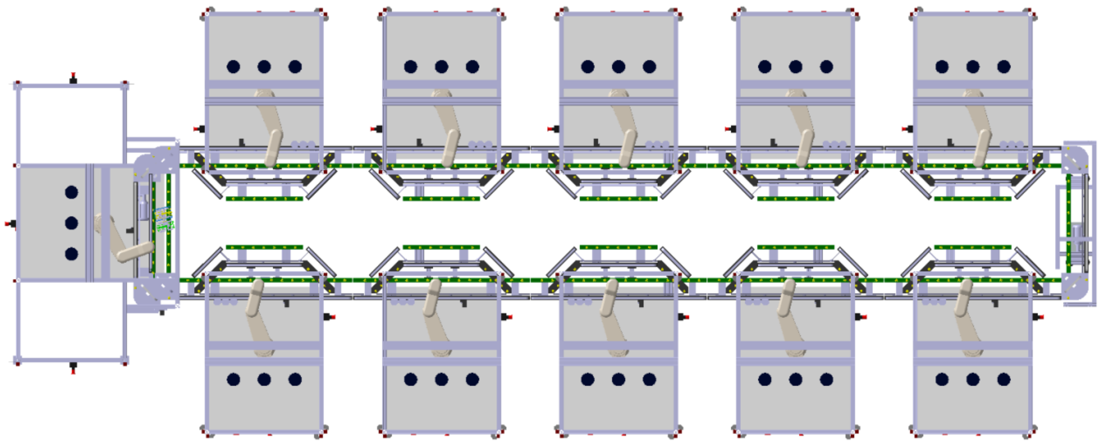


Figure 3.2. ***FASTory Line top view. workstations layout.***

The workstation conveyor includes a bypass to divert the pallets traffic when they don’t need any robot operation at the given workstation.

The workstation robotic cell contains a SCARA robot that performs the assembly operation needed by the product components travelling on the pallet.

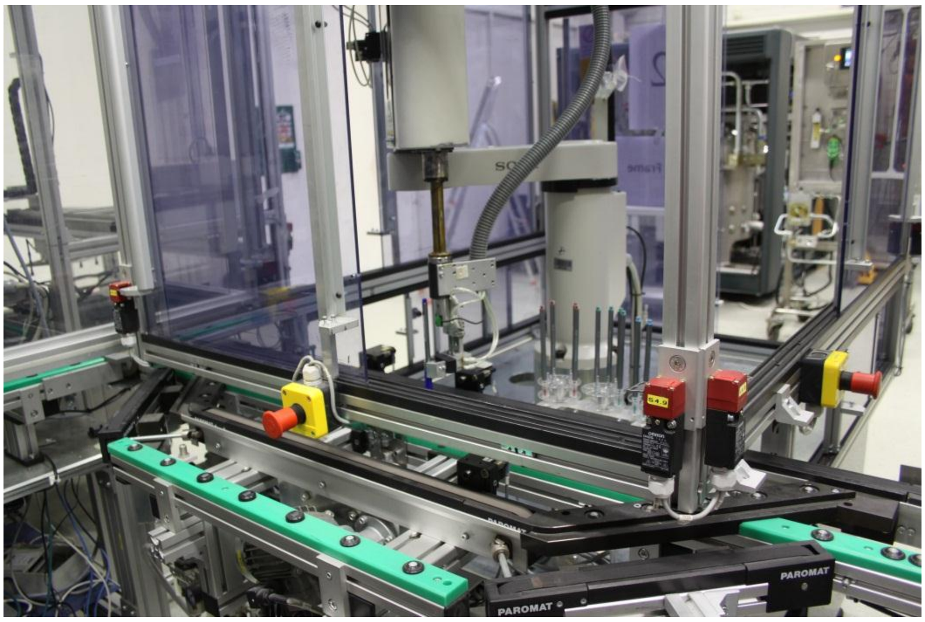


Figure 3.3. ***Workstation detailed photograph at FASTory Line.***

There is one stopper with a NFC reader at the entrance point of every workstation that checks the pallet NFC tag. When a pallet arrives there, it is identified and the control logic of the line decides if it is sent through the bypass or it is sent to the robotic cell main conveyor.

At the end of the bypass conveyor there is another stopper, where the pallet waits till the workstation exit area of the conveyor is free of pallets coming from the robot cell and therefore the pallet coming from the bypass can be reintegrated into the normal pallet flow.

Pallets travelling through the main conveyor have two stoppers in their way. One stopper holds on the pallet at the assembly point for the robot operation- The other one, placed some centimetres before, stops the next incoming pallet waiting for the robot operation area to be free.

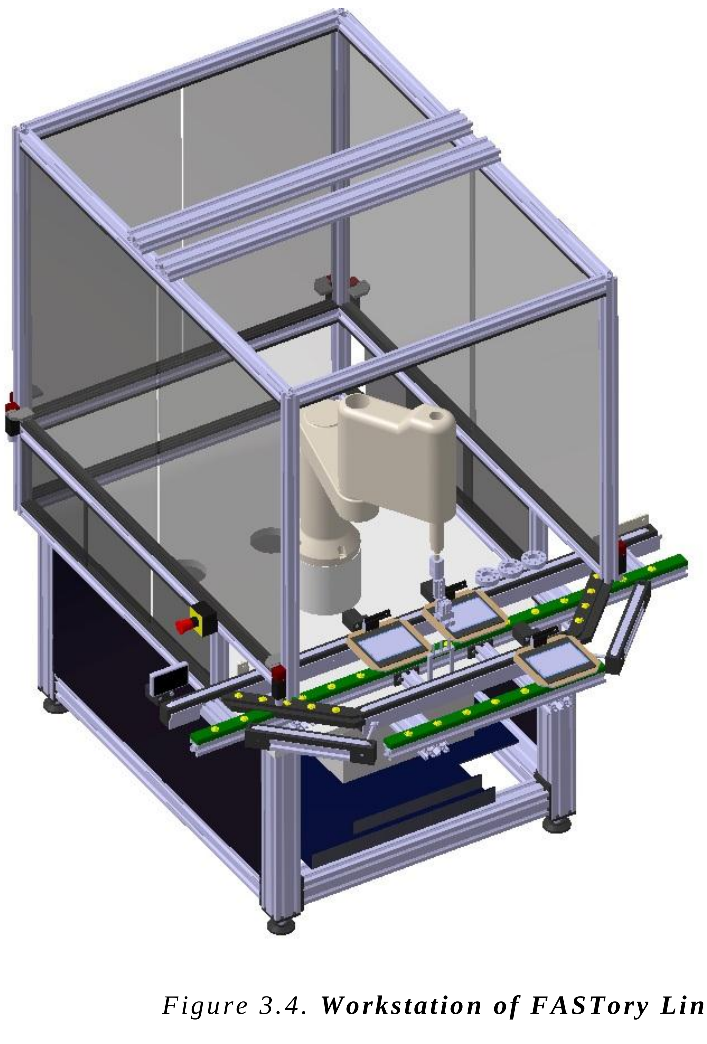


Figure 3.4. ***Workstation of FASTory Line.***

Workstation conveyors are 160 centimetres long and they are powered by asynchronous electric motors. The four stoppers are a make up by a pneumatic cylinder that when is activated it moves into the pallets path, avoiding their movement forwards and making them slip on the conveyor's belt.

The belt is always powered by the electric motor and never stops working. When the pallet has to go on with its movement, the control system retracts the stoppers cylinder and the pallet finds its way free.

This stopper and belt operation resembles the working order of a Power and Free conveyor, it enables that the belt is turning while one pallet is held at the stopper, this makes possible that one single pallet can be stopped in a conveyor while other pallets in the same conveyor are moving and provides more flexibility to the workstation operation.

The production line purpose is to simulate assembly operations. Robots draw mobile phone components (frame, keyboard and screen) on paper templates travelling on the line pallets. There is a wide range of shapes and colours for the robots operations, adding complexity to the line.

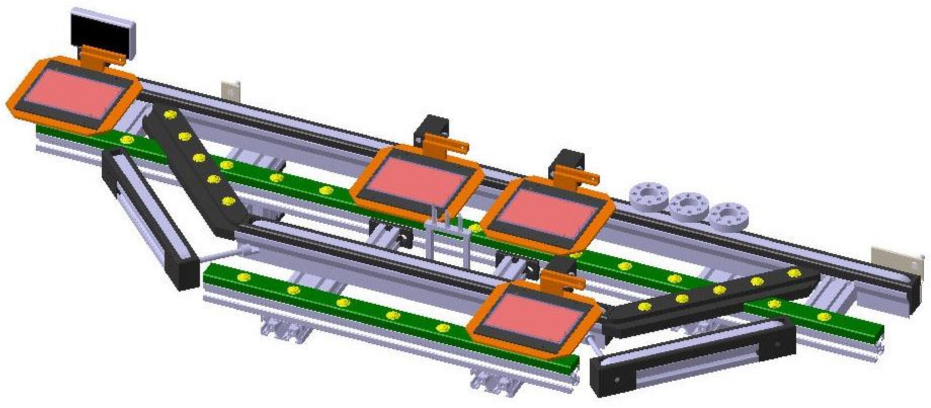


Figure 3.5. ***Conveyor and stoppers location.***

<u>Line communications</u>

Line controllers are smart remote terminal units (RTU). Commercial devices are Inico S1000 model; They can communicate among them and with high level applications through Web Services messaging through IPv6 protocol on Industrial Ethernet. Communication with the robots controllers is done through RS232 serial port.

FASTory line "can be monitored remotely by publishing information through internet without major modifications to the system. It suffices to enable a router with Internet connection. This is possible due to the fact that Web Services are transported over HTTP and can easily travel through standard network appliances" [Gonzalez et al., 12].

3.2 Use case presentation

3.2.1 Design Specifications

The 3D Real-Time Monitoring System that constitutes the proposal of the present thesis is expected to achieve a number of specifications.

Firstly, the 3D Monitoring System is expected to run as a Unity developed application. Unity software has been previously introduced in chapter 2 of this thesis as part of the theoretical background.

Second, the system should show a 3D reliable model of the assembly line. 3D models for the FASTory line resources and products will be reused from previous CATIA designs. Necessary modifications or updates of these models shall be done.

Third, the system should be able to fully integrate with the line data acquisition application. The 3D monitoring System will be event-driven, the messages arriving from the data acquisition system will be fetched in a browser script and they will trigger the status update of the line devices in the Unity application.

In addition, the resulting application should run on the web, so that it can be accessible "from any device, from anywhere".

Finally, real-time specifications requirements should be taken into account during all the application development.

3.2.2 System Integration

The 3D Real-Time Monitoring System that constitutes the proposal of the present thesis works connecting with the production line data acquisition system.

In practice, interaction is limited to a function call by the data acquisition system. The function, named 'EventRec()', is part of a JavaScript script of the 3D Monitoring System. The function call is used as a trigger for the events in the 3D Monitoring System, as well as carrying the line information related to the given event.

It is not the purpose of this thesis to elaborate on the production line data acquisition system. However, in order to present the use case and provide a better understanding of the 3D Monitoring System for the reader, a brief overview of the whole data acquisition system is provided in this section.

The data acquisition system merely sends the data from the production line to the web browser, from where it is retrieved by the web app that constitutes the 3D Real-time Monitoring System.

Data is sent without any manipulation besides than format conversion, so that line data is available as pure as possible for further treatment once it is retrieved by the 3D Monitoring App.

The following diagram shows at a glance an overview of the data flow.

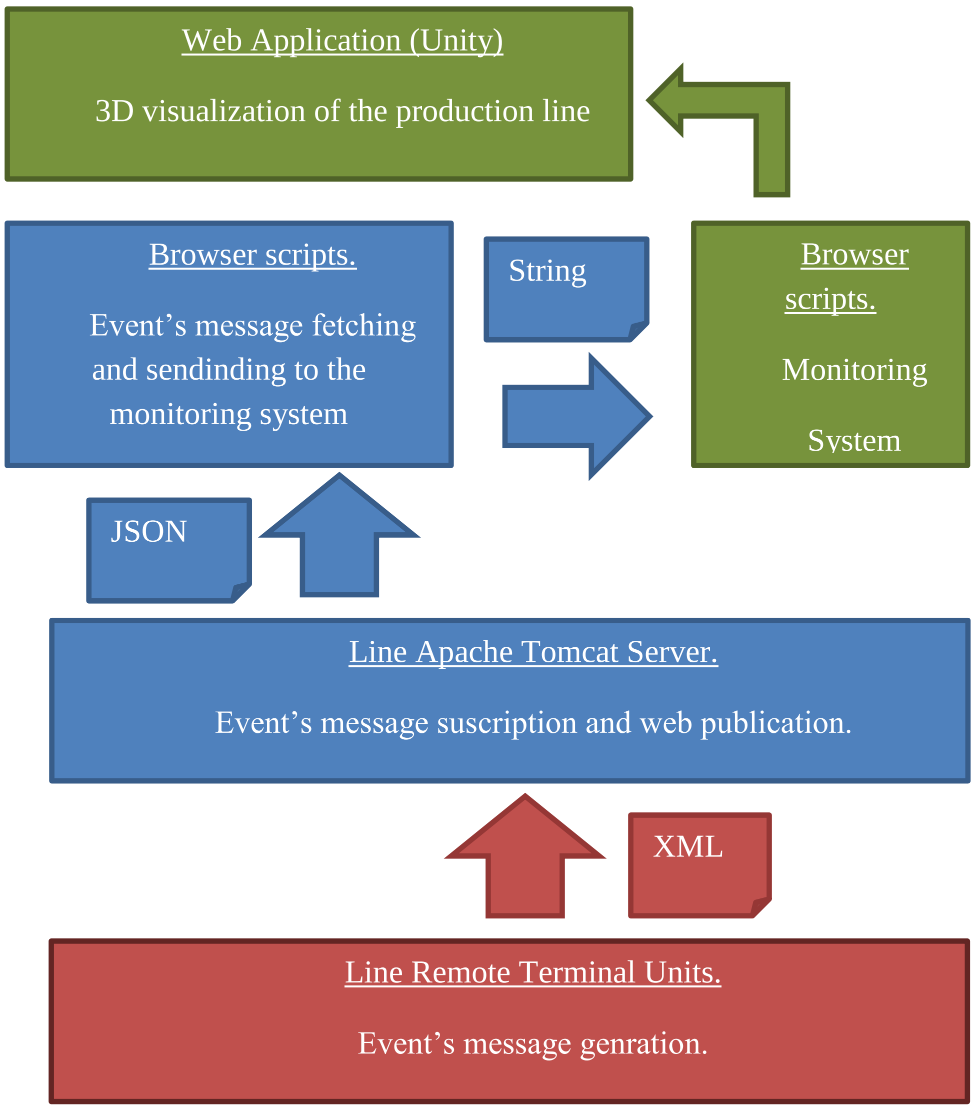

Web Application (Unity)
3D visualization of the production line
Browser scripts.
Event's message fetching and sendinding to the monitoring system
String
Browser scripts.
Monitoring System
JSON
Line Apache Tomcat Server.
Event's message suscription and web publication.
XML
Line Remote Terminal Units.
Event's message genration.

Figure 3.6. ***Data Flow. Red: line control system. Blue: Data Acquisition System. Green: 3D Real-time Monitoring System.***

3.2.3 Background on FASTory Line Simulation

This section describes a straightforward approach to FASTory line visual simulation with DELMIA software.

This approach is based on the background content for visual simulation of production lines discussed in section 2 of chapter 2.

It is used as an introduction to all the work subsequently reviewed and which makes up the proper content of the thesis.

Delmia simulation is a useful tool to introduce this present thesis. This is due to the fact that it allows skipping in a f some problematic issues such as the CAD model integration into Unity, since Delmia is fully integrated with CATIA, sharing interface and 3D objects models files. In addition, animation of components of the simulation is done avoiding scripting, simplifying again the work.

However, the Delmia capacities do not match the system design requirements, such as web app publishing. Therefore, we remind again to the reader that this Delmia Visual Simulation of Fastory Line must be considered as an approach to the thesis proposal, and never itself as a solution.

Delmia

Delmia V5 is a Digital Manufacturing software that is used by manufacturing companies to define a factory virtual model and design, plan, visualize, analyse and simulate all production processes related to the entire lifecycle of a product.

Delmia targets different areas within the manufacturing field, providing the tools that make possible the building of a whole virtual factory.

It is produced by French software developer Dassault Systèmes. Since version V5, it runs on CATIA V5 engine, sharing the graphical user interface as well.

The following figure shows Delmia GUI (Graphical User Interface). On the left side we can observe the history tree features for the current model. On the top, we found Delmia’s main menu. On the right side, we found the tools belonging to the current workbench. On the right upper corner of the model’s view we can see the compass, used to modify the point of view of the current object. Finally, top down we find other generic tools.

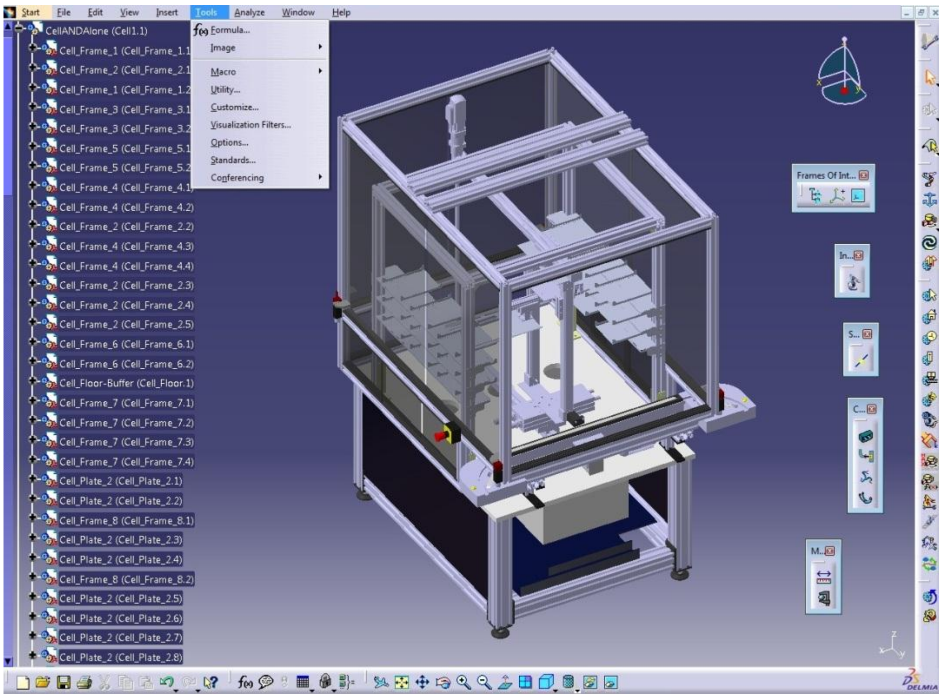


Figure 3.7. ***Delmia Graphical User Interface, example.***

<u>Digital manufacturing</u>

Product increasing complexity and optimization and continuous improvement of processes driven by market requirements lead to the need of new tools for the design, planning and simulation of the production process.

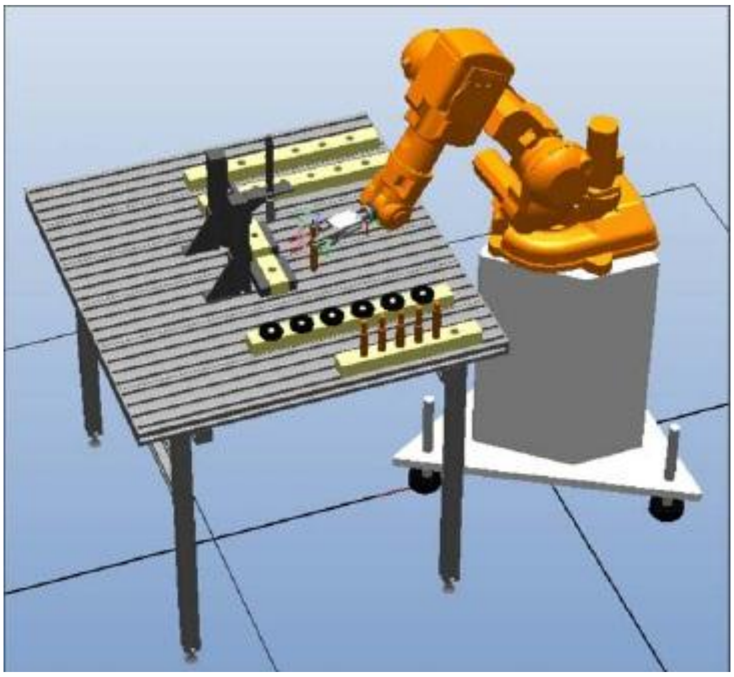


Figure 3.8. ***Digital model of a robotic assembly process. [Wang, 2011]***

Tools like physical prototypes or mock-ups of factory or manufacturing workstations layouts are out of the question. Production stops for testing or validation of new methods, equipment, layouts or processes are no longer acceptable and considered out of date because of the economical waste that they entail.

ERP (Enterprise Resource Planning) software offers limited capabilities, since it aims to the control and management of existing production processes. This underestimates the importance of innovation and the need of change an update of production processes.

Therefore, the building of a factory virtual model that can support the design, analysis and simulation of the production processes is a need.

Digital Manufacturing systems integrate different solutions to assist in the manufacturing process

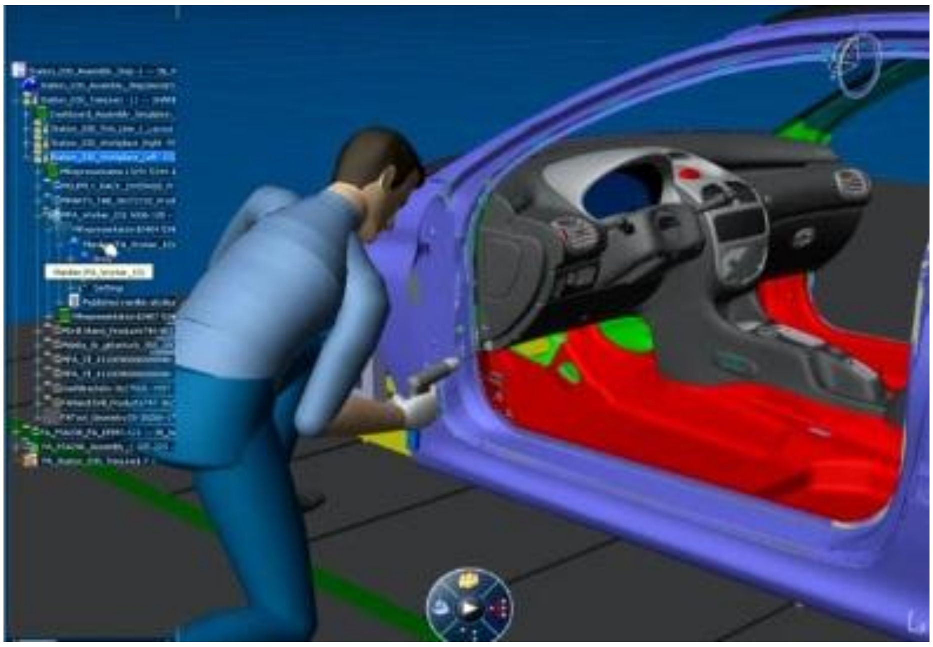


Figure 3.9. ***Digital Manufacturing example [3ds.com]***

Digital Manufacturing capabilities include the creation of visual simulations of production lines. This has positive effects on the evaluation of space requirements and other spatial relationships, the estimation of the robots and other machines reachability, the analysis of possible equipment collisions, obstructions or interferences, the assessment of the resources movement parameters (such as the robot joints or conveyors motors speeds). And in general, the 3D visualization of a production line provides a very intuitive scene of all the process leveraging the understanding of the whole line work.

<u>FASTory Line and DELMIA approach</u>

All parts and assemblies that make up the 3D model of the FASTory line were previously designed with CATIA V5. Models are imported into DELMIA, which uses the same files than CATIA for the parts and models design and even some shared workbenches.

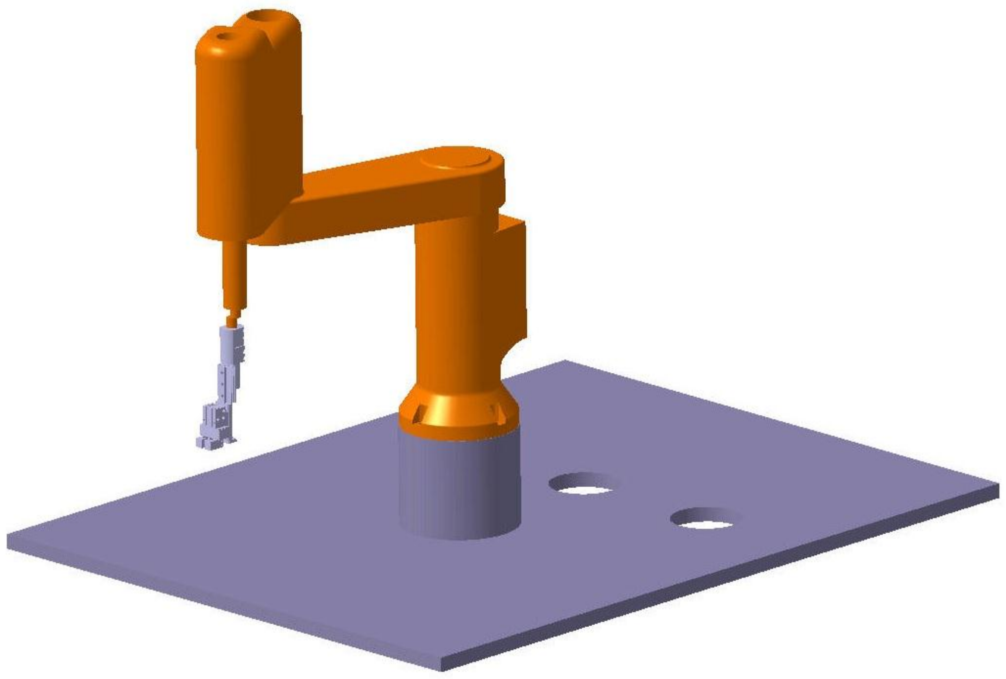


Figure 3.10. ***Robot Modelling with DELMIA for FASTory Line***

We used the ‘Device Building’ workbench, integrated in the ‘Resource Detailing’ solutions block, to define the robot animation properties and mechanisms, such as establishing the joints and the home positions.

The ‘Device Task Definition’ workbench allowed us to specify the operations carried out by the robots. Tags are placed in the consecutive positions reached by the robot tool point then we set the movement parameters for the robot tags transition.

In another group of solutions, the ‘Digital Process for Manufacturing’ block, the ‘Assembly Process Simulation Workbench’ was used in order to simulate the whole line running.

The structure of the simulation is defined by the Process, Product and Resource Tree. It contains a list with all the objects of each type that take part of the global simulation.

The resources are the robotic workstations, which are inserted into the simulation following the real line arrangement.

The products are the pallets, which are inserted in arbitrary initial points along the line conveyors.

There are two kinds of processes, robot tasks and pallet movements. Robot tasks were previously defined in the ‘Device Task Definition Workbench’, so they just need to be selected and assigned to a task process. Pallet movement processes are defined in this same workbench, setting the path and the time or speed for the pallet displacement. Sequencing and coordination of the different processes is done using Gantt or PERT charts.

Finally, the visual simulation of the line working can be executed and exported as a video file.

4. USE CASE IMPLEMENTATION

This section elaborates on the present thesis proposal development.

First section describes the line equipment modelling in CAD software, format conversion and import of the files into Unity software.

Secondly, the next section explains the process of building the scene emulating the factory environment in Unity 3D software.

Last part focuses on the publishing of the built application and its accessibility.

4.1.1 CAD to Unity assets import

3D virtual models of the assembly line equipment are called “assets” under Unity software terminology. Unity doesn’t offer 3D modelling capabilities; therefore assets need to be imported from an external application.

Unity game engine supports the import of assets from several 3D modelling formats, thus, objects or animations created from different Digital Content Creation applications such as Maya, 3ds Max, Cinema 4D, Cheetah3D or Blender can be easily imported into Unity. Native import for Unity is fbx format, and all other formats are automatically converted into fbx for its import.

Because in this thesis, as in almost every engineering application, 3D modelling is done with a CAD application, import of assets into unity requires some extra flow steps. CAD formats differ from 3D artistic modelling formats by its nature, so conversion between them is not as simple.

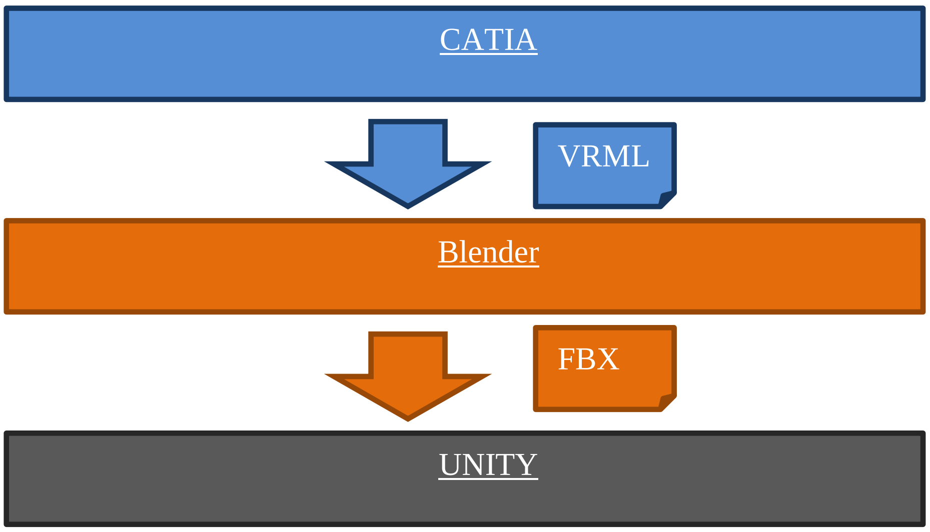


*Figure 4.1.***CAD to Unity models format conversion.**

Factory assets and products are designed in Dassault System CAD software: CATIA. A method to convert this files CAD into 3d DCC files is required. In the current thesis this was performed as follows.

The solution chosen was exporting CAD files as WRL format, importing them in Blender software, doing some minor refinement operations, exporting them as fbx format and finally importing them into Unity.

Although CAD and DCC software firms usually promise the integration of both technologies by means of ability of import and export capability to a wide range of formats, it is not so common to have a direct format integration between two different software. In addition, the format conversion involved in the import/export process is not as accurate as some times it may be figured out.

That is why in this proposal Blender is used as an intermediate format conversion tool between CATIA and Unity and why it is not only that, but also a tool more model definition refinement.

<u>CAD Assets modelling and export</u>

Assembly line equipment is modelled using Dassault System solution for product design CATIA V5.

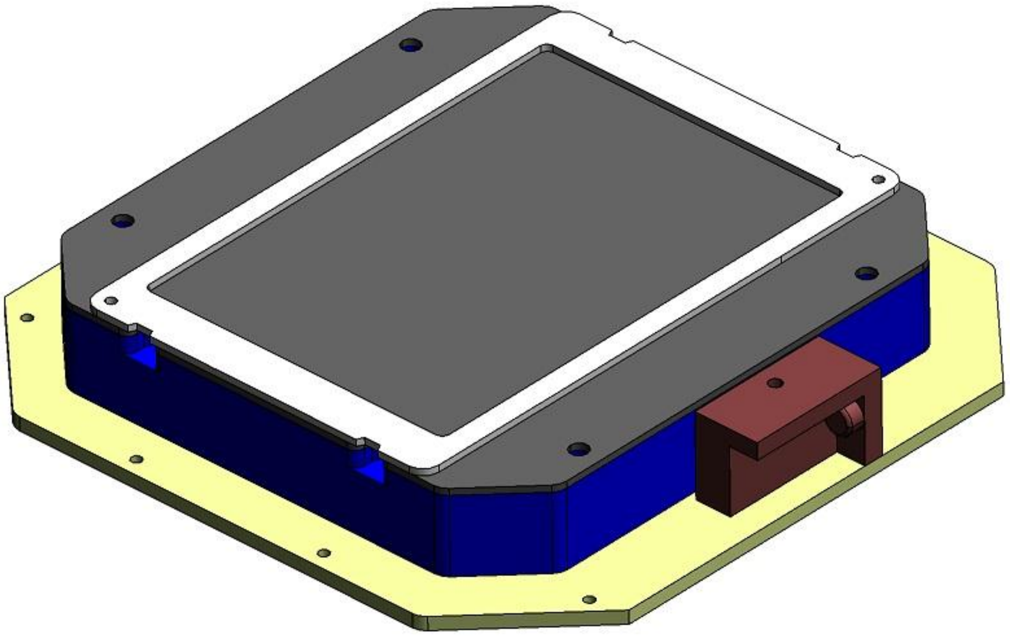


Figure 4.2. ***CAD model of one of the FASTory line pallets.***

Files need to be imported subsequently into Blender software. Among the CATIA export and Blender import supported formats, compatibility is only possible using VRML format.

Thus, CATIA assemblies are converted to single parts and exported as VRML format (.wrl extension).

In this export process some issues arise. Firstly, file size grows being one order of magnitude higher. And in addition, tessellation when exporting of the models means that the 3d model is changed, being defined with a new method, although the result apparently is the same, the mathematical model of the object is different. This new model is built automatically by the software, although quite satisfactory, the result isn't good enough and leads to some future trouble.

Blender assets import, refinement and export

VRML files are imported into Blender software in order to perform some modifications and for future export to a Unity compatible format.

First of the modifications is mesh refinement. Because the model was automatically built when exported from CATIA it has some definition errors. These may be unimportant at this point, but are clear when performing some kind of rendering later.

Unity assets import

Assets are imported into Unity 3D software and they are placed automatically in the assets folder. All assets manipulation must be done from Unity editor and not from Windows browser. Unity supports automatic update of the modifications performed in the original files with the initial 3D modelling software.

4.1.2 Unity scene building

Unity presents an asset-oriented architecture; this means that assets and not scripting are the basis for the creation of a game scene.

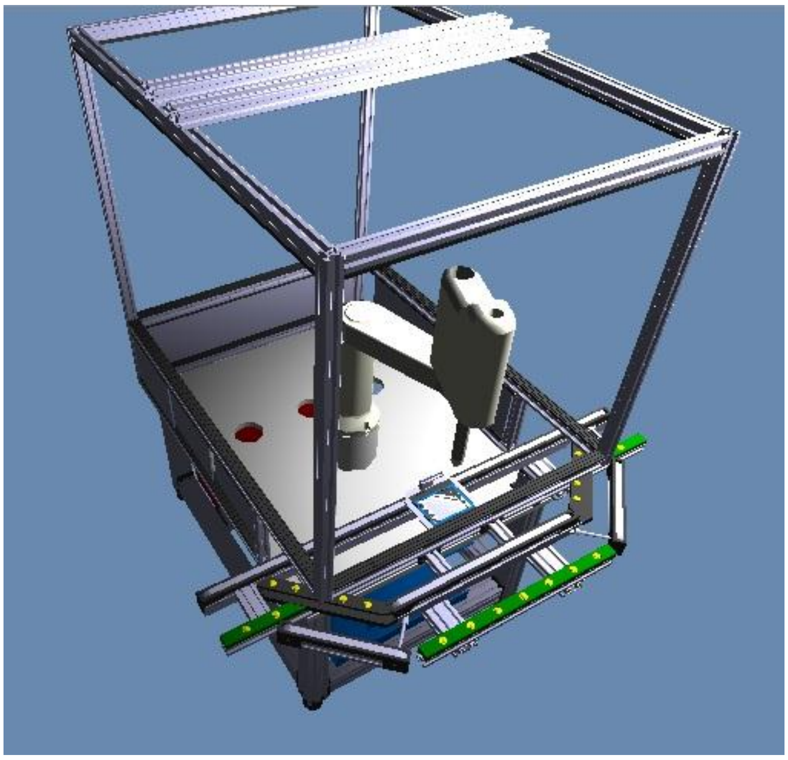


Figure 4.3. ***FASTory workstation in Unity.***

For simplifying and introductory purposes it can be said that unity scene creation is based on the import of 3D assets and the control and setting of their components and properties through scripting.

Once that assets are imported into Unity editor, simply by clicking and dragging them from the Project View to the Hierarchy or Scene View, assets are included in the Scene as Game Objects.

Game Objects are then placed around the scene according to the real factory arrangement. Unity Game Objects include a vital component for this purpose, the Transform component. It sets the position, rotation and scale of every object according to the three axis of the coordinate system.

Once that objects are placed around the scene, it is time to animate them. A script is added to every object in the scene, this script determines the object's behaviour. Scripting language chosen for this present thesis is JavaScript, in its Unity native version.

Unity scripting includes some original functions that allow the developer to animate and control the properties and components of the "game" assets.

The picture below shows the events used to drive the monitoring system. When a pallet leaves a stopper (points 1, 2, 3 and 4), a message is sent from the line to the monitoring system with the event relevant data.

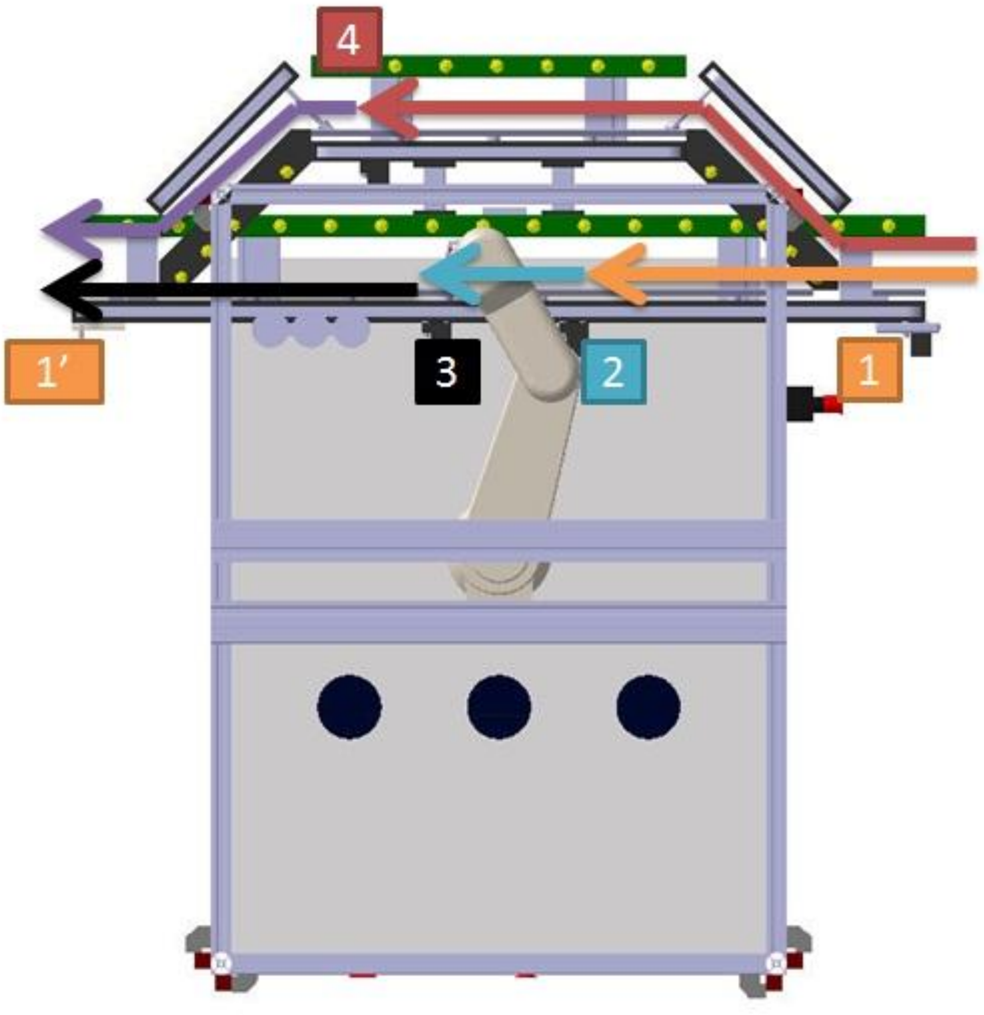


Figure 4.4. ***Pallets movement events in the workstations.***

Once the message is retrieved by the Unity application, then the corresponding pallet is animated in Unity moving it from the relevant production line stopper till the next stopper, where it remains till the pertinent message to move it forward arrives.

The arrows show the pallet flow in the workstation conveyor from one stopper till the next one. The message contains all the information needed to identify the pallet, the workstation, the conveyor, the departure stopper and the destination stopper, as well as the event timestamp.

Messages are produced according to CAMX (Computer Aided Manufacturing using XML) standard. It is used to specify the data exchange and communications among manufacturing equipment and applications.

The picture below show the original event message, produced in the line controllers, which contains the information to trigger the monitoring system events evolution.

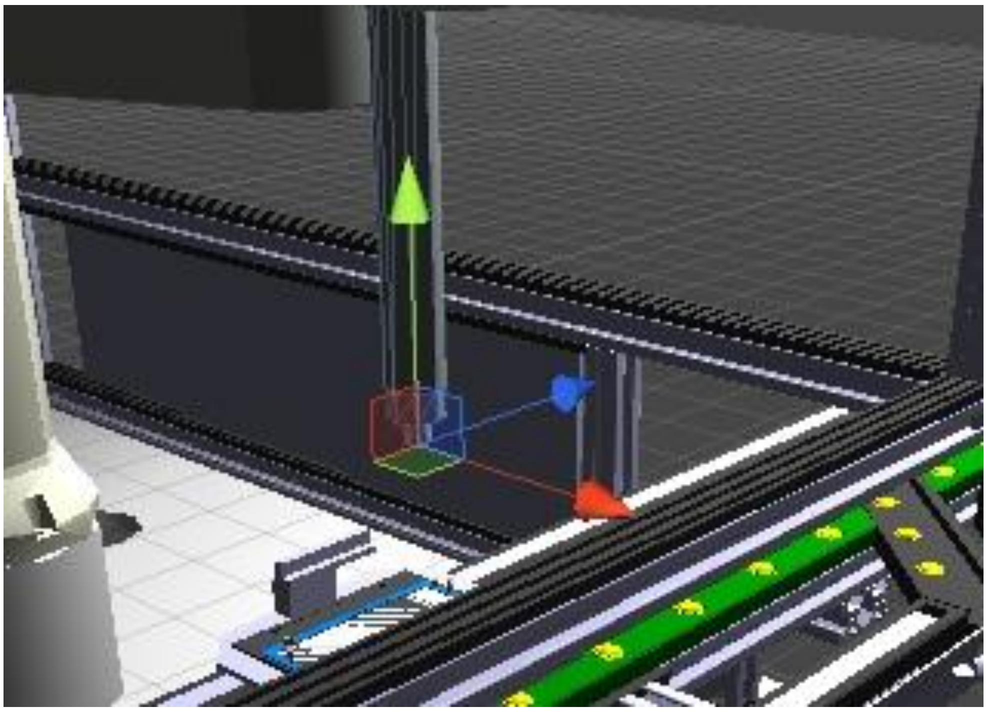


Figure 4.5. ***Transform Component view in the editor.***

The pallet movement along the assembly line conveyors is controlled defining the pallet coordinates through its Transform component.

Assets have a defined position on its rest state. When an event from the line is received indicating an evolution of its state, the position is updated until the next rest state.

Pallets rest state positions are the conveyor stoppers. When the corresponding message is received, the position is updated moving the pallet till the destination stopper.

In the same way, the robot movement is controlled defining its parts position at all times. For modularity purpose, it was decided that every part of the robot constitutes an asset on its own.

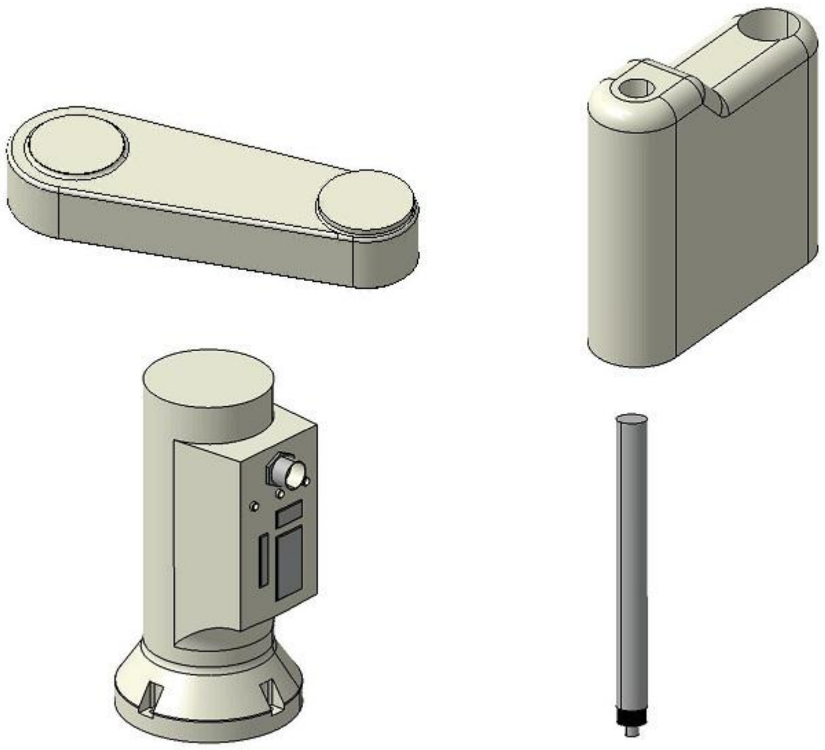


Figure 4.6. ***Robot parts imported into Unity editor.***

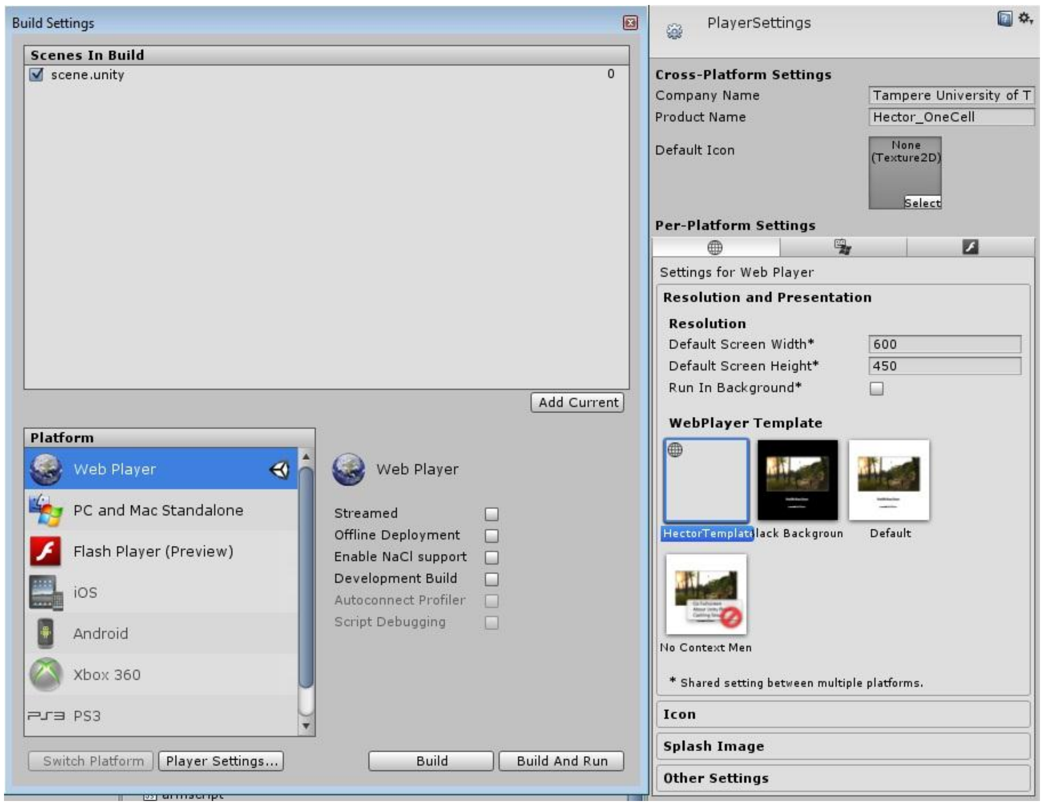
In the same way than the pallets scripting, when a message from the assembly line indicating the start of the movement of the robot is received, the robot parts scripts are updated triggering the animation of the robot. The robot parts execute the defined movement and return to their rest state.

```
if (run==true){

var objbase : GameObject;

objbase = GameObject.Find("base");

var objbasetransform = objbase.transform;


if (ida==true){

if (transform.position.z < 7.15 && (transform.position.z > 6.34)){

transform.position.z = objbasetransform.position.z - crank *
Mathf.Sin(5.70 + omega*Time.time);

transform.position.x = objbasetransform.position.x + crank *
Mathf.Cos(5.70 + omega*Time.time);

...
```

Figure 4.7. Example of the robot animation code.

4.1.3 Web application publishing from Unity

The proposed application is running on the web under the Unity web player. For the execution of the application, Unity web player has to be installed as a browser plugin, Unity 3 supports Java WebStart so end-users can download and install the web player with a single mouse click in a few seconds.



Build Settings

Scenes In Build

☑ scene.unity 0

Platform: Web Player (selected), PC and Mac Standalone, Flash Player (Preview), iOS, Android, Xbox 360, PS3

Web Player

Streamed	☐
Offline Deployment	☐
Enable NaCl support	☐
Development Build	☐
Autoconnect Profiler	☐
Script Debugging	☐

PlayerSettings

Cross-Platform Settings

Company Name	Tampere University of T
Product Name	Hector_OneCell
Default Icon	None (Texture2D)

Per-Platform Settings

Settings for Web Player

Resolution and Presentation

Resolution

Default Screen Width*	600
Default Screen Height*	450
Run In Background*	☐

WebPlayer Template: HectorTemplate (selected), Black Background, Default, No Context Menu

* Shared setting between multiple platforms.

Figure 4.8. ***Unity application publishing.***

When the application is built from the Unity editor, we get a unity3d file and html file. The unity3d file is the built application itself, the html document is the file accessed by the end-user as a normal website. It embeds the unity web player for the streaming of the application from the unity3d file to the browser.

Unity content is loaded in the browser by the Unity Web Player plugin. The HTML code includes a call to a script called UnityObject that facilitates the communication the Unity Web Player. This script also enables easy Web Player installation.

```
<script type="text/javascript"
src="http://webplayer.unity3d.com/download_webplayer-
3.x/3.0/uo/UnityObject.js"></script>
```

Once this script is loaded, the global unityObject variable can now be used to embed Unity content, this is done by invoking the embedUnity method that accepts several parameters.

```
<script type="text/javascript">

<!--

function GetUnity() {

if (typeof unityObject != "undefined") {

return unityObject.getObjectById("unityPlayer");

}

return null;

}

if (typeof unityObject != "undefined") {

unityObject.embedUnity("unityPlayer", "WebPlayer.unity3d", 600, 450);

}

</script>
```

Figure 4.9. ***Unity scripts in the HTML document***

The first the parameters of the embedUnity method specifies the id of the HTML element that will be replaced by Unity content, being the most common a <div> section, then the HTML placeholder is placed inside the <body> section of the document and the content is already embedded into the html file. The method has some other customizable parameters, including the file path for the unity3d data file to be displayed at the web player and the size of the web player in the html document.

```
<div id="unityPlayer">

<div class="missing">

<a href="http://unity3d.com/webplayer/" title="Unity Web Player.
Install now!">

<img alt="Unity Web Player. Install now!"
src="http://webplayer.unity3d.com/installation/getunity.png"
width="193" height="63" />

</a>

</div>

</div>
```

Figure 4.10. ***HTML code to place Unity Web Player in the Body of the document***

Several templates are available for the HTML document, and also own-built html documents with CSS style sheets can be used. For this thesis proposal, an on-the-purpose html template was created.

4.1.4 Unity-browser-server interface

Unity provides support for the web player plugin and the html website communication, therefore, unity functions (including parameters) can be called from the html document through embedded JavaScript code and vice-versa, script functions from the HTML document can be called in Unity.

This communication allows retrieving the assembly line information for the assets monitoring. Data acquisition from the assembly line is uncoupled from the visual monitoring system. It is not the purpose of this thesis to work on that issue, therefore it is assumed that the data is available as json objects in a provided URL. This modular architecture makes the visualization system completely independent of the data acquisition.

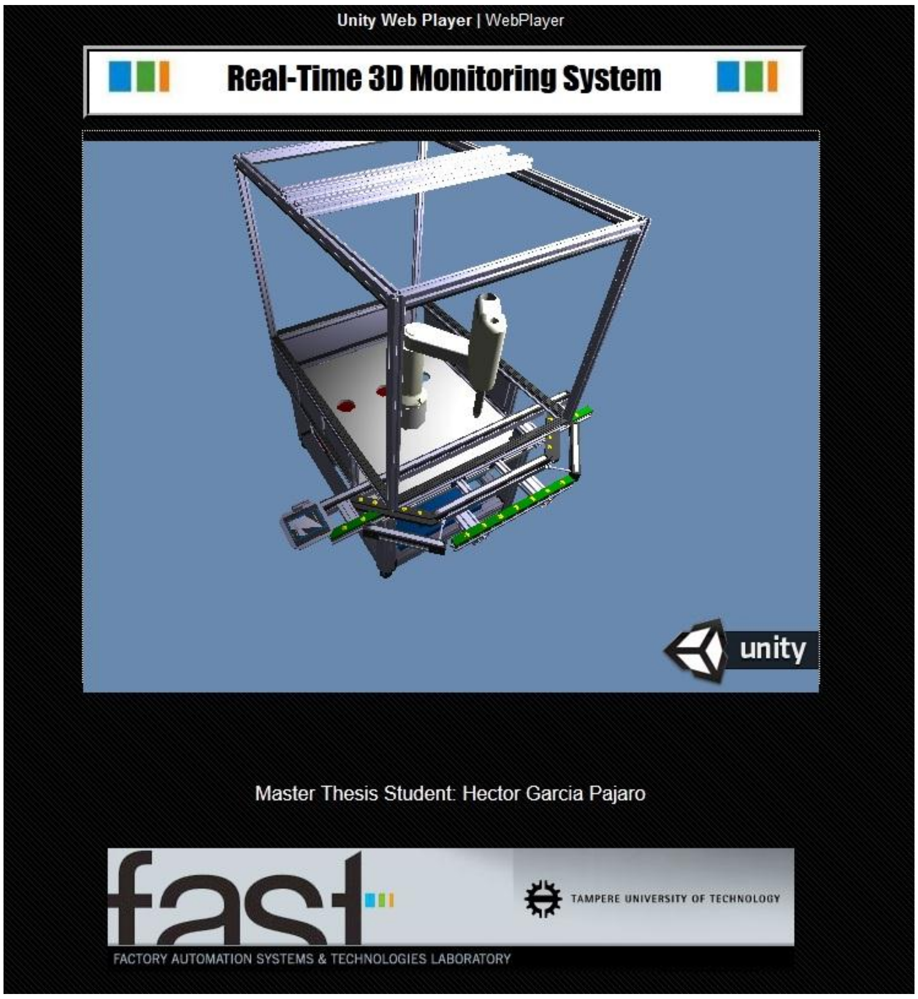

Unity Web Player | WebPlayer
Real-Time 3D Monitoring System
unity
Master Thesis Student: Hector Garcia Pajaro
fast
TAMPERE UNIVERSITY OF TECHNOLOGY
FACTORY AUTOMATION SYSTEMS & TECHNOLOGIES LABORATORY

Figure 4.11. ***Visualization of the web application, example for on cell.***

In order to get the assembly line data, a JavaScript script is embedded into the html document. It contains a function that calls the URL in which this data is published from the line every time that there is a new event notification. Data format is a json object. In order to facilitate Unity communication and for efficiency reasons, the script fetches only the last json objects whenever there is a different event, since the even notification is done periodically, the same data may arrive in different messages several times. The json object is parsed and the data is stored into a string variable which is send to unity, by calling a function inside ‘unityAPI’ script. From this ‘unity API’ script, the string variable is read and the needed variables of every asset script are updated in order to refresh the asset state.

5. CONCLUSIONS

This chapter elaborates on the conclusions that are inferred from this thesis proposal development. It discusses the success in reaching the objectives set out in the introductory Chapter 1 and it analyses the final result. Finally, new directions, further extension and future work ideas for the actual 3D monitoring system are suggested.

This visual web-based technology allows companies users to access production lines and factory assets real-time monitoring from all over the world with the only need of a web browser.

With this monitoring system, employees that are not in the factory location can get process information. In addition to workers, marketing and sales staff can show to their clients the manufacturing capabilities of the company, by means of providing them with a trustworthy, realistic and real-time visualization of the production processes.

Company suppliers can access in real-time visual information of the factory status, thus improving the process quality due to a better suppliers network integration ire order to achieve lean and JIT manufacturing.

All this comes to facilitate achieving the goals desired by any manufacturing company, like increasing productivity, improving information flow and enhancing the product and process quality.

Furthermore, the current proposal developed in this thesis offers a number of advantages. For the user, it allows to access the application from any platform without needing to install, update or maintain any software. A simple click to execute the web browser allows them to run the application.

For the manufacturing company, the developed program provides an open architecture that enables integration with other applications or tools of the company. In addition, the application architecture allows the application to be easily updated to reproduce future factory or process modifications. Besides, it is independent of the technologies used on the line. For instance, in the test bed used for the thesis proposal, the change of the control system architecture does not have an effect on the working of the 3D real-time monitoring application.

5.1 System review, results and achievements

The principal goal of the present thesis has been the development of a web application whose purpose is to visualize a 3D real-time monitoring system for a production line. In spite of offering a number of advantages over 2D common SCADA systems and video monitoring systems, 3D monitoring systems are not as widespread as they could be in the manufacturing industry. This is due to the fact that they involve a more complex development process; as a result their implementation is not always viable because of the time and cost implicated in the elaboration.

Therefore, this thesis has presented a novel approach to the issue, aiming to the elaboration of a methodology to build 3D monitoring systems reusing the CAD models for the production line equipment, that were produced during the design process. Although format conversion is always a troublesome issue, the models could be imported into Unity 3D engine, after some previous refinement.

Unity is a tool to produce 3D content applications; although it is aimed to the video game and the movie industry it provides useful solution for the building of engineering applications.

Once the 3D models are imported, Unity allows us to manage their behaviour and properties through scripting. The production line that is used as the test bed for this thesis proposal is an event based system. Therefore, we defined the relevant events, such as pallets movements or robot operations in the Unity editor with JavaScript language scripts. Then we can communicate the Unity developed application with the data acquisition system of the production line, so that standardized CAMX (Computer Aided Manufacturing using XML) messages created by the line RTUs (Remote Terminal Unit) can be sent to the monitoring system by the data acquisition system. Once the messages are fetched by the monitoring application, they trigger the corresponding event updating the behaviour scripts to the current line state.

Unity allows the developer to publish the application targeting the web platform. The application is hosted in the web server and is loaded by the relevant webpage running under the Unity Web Player. This enables the user to utilize the 3D monitoring system without the need of installing any software on their computer or carrying out updating or execution tasks. In addition, the 3D Monitoring system can be accessed with any computer, no matter of the platform particularities and from everywhere in any moment. The 3D monitoring system can not only run at the factory floor, but in any computer in any specific location on the globe.

We can say that the building of a 3D real-time monitoring system is not a straightforward task. During its development some troublesome issues appeared.

First of all, CAD models cannot be imported directly into Unity. CATIA does not provide an export format that is compatible with Unity import formats. Therefore, we used Blender as an intermediate solution to achieve format compatibility.

However, not only file format is a problem, but things go beyond it. The files format conversion implies a modelling method change. The modelling procedure is substantially different for CAD and 3D DCC applications, consequently the resulting

3D models after the export to VRML format are not defined in a proper way, with an overconstrained mesh that contains small bugs and inaccuracy errors. Although they are not big mistakes and apparently at the first glance they are working fine, they are detrimental to the visualization rendering performance. This makes necessary the use of polygon reduction or decimation methods in order to improve the mesh definition and achieve the desired result.

This CAD models reutilization enables to create reliable 3D monitoring applications in a more straightforward and simple way than in the case that common 3D graphics libraries or APIs are used. All this is translated in saving a lot of time and reducing the cost for the development process.

5.2 Future work

The present thesis has brought to the reader's view the enormous potential that 3D real-time monitoring systems can offer. They are a growing an expanding field on which arguably will play most manufacturing companies during the coming years. The proposal that is suggested in this thesis offers a lot of possibilities for further development and expansion.

On behalf of optimization, several actions can be taken. First of all, the improving of the application rendering performance would make possible to provide a more user friendly interface, as well as using it for marketing purposes. In the same way, robot animation reliability can be improved and various readjustments could be carried out. Finally, some techniques can be employed to optimize bandwidth consumption of the current application.

But most important guidelines for future work rely on extending the capabilities of the presented application. Some of them are suggested below:

3D assets visualization interface may present as well other information common to monitoring systems such as energy consumption, production rates, or operations executed to the product, alarms state or maintenance parameters. This information can be visualized through text labels, charts, 2D graphs or colours codes integrated in the same window than the present application or creating different selectable interfaces.

In order to maximize the benefits of 3D visualization, more fixed viewpoints can be added to the current proposal. Furthermore, the inclusion of camera motion controls would allow the user to navigate freely around the 3D scene. Pan, rotate and zoom controls can be easily implanted. In addition, it can be added the functionality of seeking resources or products in real-time selecting them from a list or by name identifier to get a closer overview.

One natural extension, empowered by Unity multi-platform targeting tools, would be the expansion of the monitoring application to Android and iOS mobile operative systems. This can be easily done extending Unity license grants and realizing minor code modifications.

Furthermore, the technology employed enables us to have at our disposal a flexible architecture for the built application. The monitoring application is an event driven system, but it must not be like this by imperative. For instance, the test bed used for the use case, FASTory line at Tampere University of Technology, implements a pallets localization system through wireless sensors and artificial intelligence algorithms. The data that produced by this system, that is, coordinates of the pallets at every time, can replace or complement the current data acquisition system.

Finally, in coordination with the data acquisition system it could be implemented a data base or another system to store relevant past information, so that in case of failure or any anomaly in the production line normal working order, the virtual system can be driven backwards to the critical timestamp and it can be then executed to visualize what had happened.

REFERENCES

[3ds.com, b]	http://a1.media.3ds.com/fileadmin/VIDEOS/Thumbnails/DELMIA-V6-Automotive-Final-Assembly-400x300.jpg
[3ds.com,]	http://a3.media.3ds.com/fileadmin/PRODUCTS/DELMIA/IMG/DELMIA_RoboticComm_400x267.png
[ashlar.com]	http://www.ashlar.com/gallery/electronics/images/elec037.jpg
[Bertoline et al., 1995]	Bertoline, Gary R.. Wiebe, Eric N.. Miller, CraigL.. Nasma, Leonard O.. “Engineering Graphics Communication”. Chicago : Irwin, 1995. ISBN: 0-256-11418-8 0-256-21309-7
[Beuthel et al., 2002]	Beuthel, C., Dai, F., Kolb, E., & Kruse, E. (2002). “3D visualization for the monitoring and control of airport operations.” *Control Engineering Practice*, *10*(April), 1315-1320
[Blackman et al., 2011]	Blackman, Sue. Beginning 3D Game Development with Unity: The World's Most Widely Used Multi-Platform Game Engine. Apress, © 2011. Books24x7. Web. May 8, 2012
[blender.org, 2012]	http://www.blender.org
[Chen et al., 2002]	Lienjing Chen, P. Bender, P. Renton, T. El-Wardany, Integrated Virtual Manufacturing Systems for Process Optimisation and Monitoring, CIRP Annals - Manufacturing Technology, Volume 51, Issue 1, 2002, Pages 409-412, ISSN 0007-8506, 10.1016/S0007-8506(07)61548-0.
[clinispace.com]	http://www.clinispace.com/images/content_hero.png
[Craighead et al., 2008]	J. Craighead, J. Burke, R. Murphy. "Using the Unity Game Engine to Develop SARGE: A Case Study". Proceedings of the 2008 Simulation Workshop at the International Conference on Intelligent Robots and Systems (IROS 2008). Sept. 2008
[de Albuquerque &Lelièvre-Berna, 1998]	M.P. de Albuquerque, E. Lelièvre-Berna, Remote monitoring over the Internet, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, Volume 412, Issue 1, 21 June 1998, Pages 140-145, ISSN 0168-9002, 10.1016/S0168-9002(98)00294-0

[Feldhorst et al., 2010] Feldhorst, S.; Fiedler, M.; Heinemann, M.; ten Hompel, M.; Krumm, H.; , "Event-based 3D-monitoring of material flow systems in real-time," *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on* , vol., no., pp.195-200, 13-16 July 2010 doi: 10.1109/INDIN.2010.5549433

[Flavell, 2010] Flavell, Lance. "Chapter 3 - Modeling". Beginning Blender: Open Source 3D Modeling, Animation, and Game Design. Springer. © 2010. Books24x7.

[Gonzalez et al., 2012] [Gonzalez 2012] Luis E. Gonzalez Moctezuna, Jani Jokinen, Corina Postelnicu, Jose L. Martinez Lastra. ”Retrofitting a Factory Automation System to Address Market Needs and Societal Changes.” Industrial Informatics (INDIN), 2012 10th IEEE International Conference.

[Harrop & Armitage, 2006] W. Harrop, G. Armitage, "Real-Time Collaborative Network Monitoring and Control Using 3D Game Engines for Representation and Interaction," in *VizSEC'06 Workshop on Visualization for Computer Security, Virginia, USA, October-November 2006.*

[mit.edu] http://mstatic.mit.edu/nom150/items/Sketchpad_01.jpg

[panda3d.com] http://www.panda3d.org/randomscreens/ssg-code3d/code3D01.jpg

[Russ et al., 2009] Russ Agrusa, Valeria G. Mazza, and Roberto Penso. 2009. Advanced 3D visualization for manufacturing and facility controls. In Proceedings of the 2nd conference on Human System Interactions (HSI'09). IEEE Press, Piscataway, NJ, USA, 453-459.

[Sano et al., 2011] Sano, T., Nakata, H. and Sugimoto, A. (2011), A generation and customization method for constructing a 3D GUI for SCADA. Electron. Comm. Jpn., 94: 53–63. doi: 10.1002/ecj.10221

[Schafer et al., 2005] Schafer, Steve. "Chapter 1 - The Basics of HTML". Web Standards Programmer's Reference: HTML, CSS, JavaScript, Perl, Python, and PHP. Wrox Press. © 2005. Books24x7. <http://common.books24x7.com/toc.aspx?bookid=11777> (accessed May 14, 2012)

[Takeno et al., 2008] Takeno, J.; Enaka, T.; Sato, H.; , "A 3D online monitoring system," *Sensing Technology, 2008. ICST 2008. 3rd International Conference on* , vol., no., pp.318-323, Nov. 30 2008-Dec. 3 2008 doi: 10.1109/ICSENST.2008.4757121

[unity3d.com, 2012]	http://www.unity3d.com
[visualcomponents.com, 2012]	http://download.visualcomponents.net/www/Documents/Success_Stories/Orfer_EN.pdf
[Wang et al., 2003]	Lihui Wang, Ryan Sams, Marcel Verner, Fengfeng Xi, Integrating Java 3D model and sensor data for remote monitoring and control, Robotics and Computer-Integrated Manufacturing, Volume 19, Issues 1–2, February–April 2003, Pages 13-19, ISSN 0736-5845, 10.1016/S0736-5845(02)00058-3.
[Wang et al., 2011]	Lihui Wang, Mohammad Givehchi, Göran Adamson, Magnus Holm, A sensor-driven 3D model-based approach to remote real-time monitoring, CIRP Annals - Manufacturing Technology, Volume 60, Issue 1, 2011, Pages 493-496, ISSN 0007-8506, 10.1016/j.cirp.2011.03.034.
[Wang, 2010]	Wang, Lihui. 2010. A Novel Collaborative Planning Approach for Digital Manufacturing. Book Title: Proceedings of the 6th CIRP-Sponsored International Conference on Digital Enterprise Technology. Pages 939-955. Volume 66. Publisher: Springer Berlin / Heidelberg.
[White, 2009]	White, Alexei. "Chapter 1 - Introduction to JavaScript". JavaScript Programmer's Reference. Wrox Press. © 2009. Books24x7.
[Zakas, 2012]	Zakas, Nicholas C.. "Professional - JavaScript for Web Developers, Third Edition". Professional JavaScript for Web Developers, Third Edition. Wrox Press. © 2012. Books24x7.
[Zhang&Wang, 2005]	Dan Zhang and Lihui Wang. 2005. Web-based remote manipulation in advanced manufacturing system. In *Proceedings of the IEEE EEE05 international workshop on Business services networks* (BSN '05). IEEE Press, Piscataway, NJ, USA, 15-15.

APPENDIX 1: ANIMATION SCRIPTS

Browser API

```
function getmsg (paldata:String){
          var pall : GameObject;
          pall = GameObject.Find("pallet_000");
          pallscript = pall.GetComponent("palletscript");
          var pallet=paldata.Split("~"[0]);
          pallscript.posit=pallet[5];
          pallscript.tozone=pallet[3];
          positi=pallet[5];
          pallscript.toLane=pallet[3];
          pallscript.newmessage(positi);
}
```

Pallet Script

```
function Start () {
          transform.position = Vector3(2.21, 2.23, -3.5);
          transform.eulerAngles = Vector3(0, 0, 0);
}
function Update () {
          if (posit=="1"){
                    if (tozone=="4"){
                              if (transform.position.z < -2){
                                        transform.Translate( 0, 0,
speed * Time.deltaTime, Space.World);
                                        transform.Rotate( 0, 0, 0,
Space.Self);
                              }//turn right
```

```
                                if ((transform.eulerAngles.y < 45) &&
(transform.position.z < 1)){

                                           transform.Translate( 0, 0,
speed * Time.deltaTime, Space.World);

                                           transform.Rotate( 0,
rot*Time.deltaTime, 0, Space.Self);

                                }//straight 45

                                if ((transform.eulerAngles.y >= 45) &&
(transform.position.z < 1.3)){

                                           transform.Translate(
(speed+0.2) * Time.deltaTime, 0, speed * Time.deltaTime,
Space.World);

                                           transform.Rotate( 0, 0, 0,
Space.Self);

                                }//turn left

                                if ((transform.position.z >= 1.3) &&
(transform.eulerAngles.y > 2)){

                                           transform.Translate( 0, 0,
speed * Time.deltaTime, Space.World);

                                           transform.Rotate( 0, -
rot*Time.deltaTime, 0, Space.Self);

                                }// straight 0

                                if ((transform.eulerAngles.y <= 2) &&
(transform.position.z < 7.5) && (transform.position.z > 1.5)){

                                           transform.Translate( 0, 0,
speed * Time.deltaTime, Space.World);

                                           transform.Rotate( 0, 0, 0,
Space.Self);

                                }


                     } else{

                                if (transform.position.z < 2.5){

                                           transform.Translate( 0, 0,
speed * Time.deltaTime, Space.World);

                                           transform.Rotate( 0, 0, 0,
Space.Self);

                                }

                     }

          } else if (posit=="2"){

                     if (transform.position.z < 5){
```

```
                                transform.Translate( 0, 0, speed *
Time.deltaTime, Space.World);

                                transform.Rotate( 0, 0, 0,
Space.Self);

                    }

          } else if (posit=="3"){

                    if (transform.position.z <13.5){

                                transform.Translate( 0, 0, speed *
Time.deltaTime, Space.World);

                                transform.Rotate( 0, 0, 0,
Space.Self);

                    }

          } else if (posit=="4"){

                    if (transform.position.z < 8){

                                transform.Translate( 0, 0, speed *
Time.deltaTime, Space.World);

                                transform.Rotate( 0, 0, 0,
Space.Self);

                    }

                    if ((transform.position.z >= 8) &&
(transform.position.z < 8.5)){

                                transform.Translate( (-speed) *
Time.deltaTime, 0, (speed+0.3) * Time.deltaTime, Space.World);

                                transform.Rotate( 0, -
rot*Time.deltaTime, 0, Space.Self);

                    }

                    if ((transform.position.z >= 8.5) &&
(transform.eulerAngles.y > 315)&& (transform.position.z < 11)){

                                transform.Translate( (-speed) *
Time.deltaTime, 0, (speed+0.3) * Time.deltaTime, Space.World);

                                transform.Rotate( 0, -
rot*Time.deltaTime, 0, Space.Self);

                    }

                    if ((transform.eulerAngles.y <= 315) &&
(transform.position.z < 11.5) && (transform.eulerAngles.y >= 10)){

                                transform.Translate( (-speed) *
Time.deltaTime, 0, speed * Time.deltaTime, Space.World);

                                transform.Rotate( 0, 0, 0,
Space.Self);

                    }
                    if ((transform.position.z>= 11.5) &&
(transform.position.z<= 12.5) ){

                                          transform.Translate( 0, 0,
speed * Time.deltaTime, Space.World);

                                          transform.Rotate( 0, 45 *
Time.deltaTime, 0, Space.Self);

                    }

                    if ((transform.position.z>= 12.5) &&
(transform.position.z<= 13.5) ){

                                          transform.Translate( 0, 0,
speed * Time.deltaTime, Space.World);

                                          transform.Rotate( 0, 0, 0,
Space.Self);

                    }

                              if (transform.position.z < 2.5){

                                          transform.Translate( 0, 0,
speed * Time.deltaTime, Space.World);

                                          transform.Rotate( 0, 0, 0,
Space.Self);

                              }

          }

}



function newmessage (positi:String){

          if ((positi=="1")){

                    transform.position = Vector3(2.21, 2.23, -3.5);

                    transform.eulerAngles = Vector3(0, 0, 0);

          }

                    else if (positi=="2"){

                              transform.position = Vector3(2.21,
2.23, 2.5);

                              transform.eulerAngles = Vector3(0, 0,
0);

                    }

                    else if (positi=="3"){

                              transform.position = Vector3(2.21,
2.23, 5);
```

```
                                transform.eulerAngles = Vector3(0, 0,
0);
                    }
                    else if (positi=="4"){
                                transform.position = Vector3(5.25,
2.23, 7.5);
                                transform.eulerAngles = Vector3(0, 0,
0);
                    }
}
```

Robot Arm 1 Script

```
function Start () {
         transform.eulerAngles.y=330;
}
function Update () {
         if (run==true){
                    if (ida==true){
                                //movement from 1 to 2
                                if((robotbutton==true) &&
transform.eulerAngles.y<345){
                                           transform.RotateAround
(pivotpoint, Vector3.up, 5 * Time.deltaTime);
                                }
                                //movement from 2 to 3
                                if((robotbutton==true) &&
transform.eulerAngles.y>345 &&  transform.eulerAngles.y<355){
                                           transform.RotateAround
(pivotpoint, Vector3.up, 7 * Time.deltaTime);
                                }
                    }
                    //movement from 3 to 4
                    if((robotbutton==true) &&
(transform.eulerAngles.y>355) && (ida==true)){
                                ida=false;
                    }

                    if((robotbutton==true) &&
(transform.eulerAngles.y>340) && (ida==false)){

                              transform.RotateAround (pivotpoint,
Vector3.up, (-5.3) * Time.deltaTime);

                    }

                    //movement from 4 to 1

                    if((robotbutton==true) &&
(transform.eulerAngles.y<340) && (transform.eulerAngles.y>330) &&
(ida==false)){

                              transform.RotateAround (pivotpoint,
Vector3.up, (-5.3) * Time.deltaTime);

                    }

          }

}
```

Robot Arm 2 Script

```
function Start () {

          transform.position=Vector3(-0.155,6.10, 7.1385);

          transform.eulerAngles.y=35;

          var objbase : GameObject;

          objbase = GameObject.Find("base");

          var objbasetransform = objbase.transform;

}
```

function Update () {

```
          if (run==true){

                    var objbase : GameObject;

                    objbase = GameObject.Find("base");

                    var objbasetransform = objbase.transform;


                    if (ida==true){

                              //movement from 1 to 2

                              if (transform.position.z < 7.15 &&
(transform.position.z > 6.34)){
```

```
                                        transform.position.z =
objbasetransform.position.z - crank * Mathf.Sin(5.70 +
omega*Time.time);

                                        transform.position.x =
objbasetransform.position.x + crank * Mathf.Cos(5.70 +
omega*Time.time);


         transform.Rotate(Vector3.up, 1.5*Time.deltaTime,
Space.Self);



                              }

                              //movement from 2 to 3. initial angle:
6.021

                              if (transform.position.z < 6.34 &&
(transform.position.z > 5.8)){

                                        transform.position.z =
objbasetransform.position.z - crank * Mathf.Sin(5.70 +
omega*Time.time + 0.052*accel*(Time.time)*(Time.time));

                                        transform.position.x =
objbasetransform.position.x + crank * Mathf.Cos(5.70 +
omega*Time.time + 0.052*accel*(Time.time)*(Time.time));


         transform.Rotate(Vector3.up, -7.5*Time.deltaTime,
Space.Self);

                                        accel= accel + 0.001;

                                        //var giro:float= ((5.70 +
omega*Time.time)/3.14*180)-360;// angle (degrees) for the arm2 since
the start of the movement

                                        }

                    }



                    //movement from 3 to 4,         initial angle:
6.0705

                    if ((transform.position.z < 5.8) && (ida==true)){

                              ida=false;

                    }

                    if ((transform.position.z < 6.65) &&
(ida==false)){

                              // newtime reference

                              if (i<1){

                                        newtime=Time.time;
```

```
                                          i=i+1;

                                          Debug.Log("i: " + i);

                                          Debug.Log("newtime: " +
newtime);

                               }

                               //update  timer

                               timer=Time.time-newtime;

                               Debug.Log("newtime: " + newtime);

                               Debug.Log("Time.time: " + Time.time);

                               Debug.Log("timer: " + timer);

                               Debug.Log("Time.deltaTime: " +
Time.deltaTime);

                               Debug.Log("i: " + i);

                               //update position

                               transform.position.z =
objbasetransform.position.z - 3.55 * Mathf.Sin(6.1105 - 0.08*timer);

                               transform.position.x =
objbasetransform.position.x + 3.55 * Mathf.Cos(6.1105 - 0.08*timer);

                               transform.Rotate(Vector3.up, (-
2.3)*Time.deltaTime, Space.Self);

                               Debug.Log("position.z : " +
transform.position.z);

                               Debug.Log("position.x : " +
transform.position.x);

                     }

                     //movement from 4 to 1

                     if ((transform.position.z > 6.65)
&&(transform.position.z < 7.15) && (ida==false)){

                               timer=Time.time-newtime;

                               transform.position.z =
objbasetransform.position.z - 3.55 * Mathf.Sin(6.1105 - 0.08*timer);

                               transform.position.x =
objbasetransform.position.x + 3.55 * Mathf.Cos(6.1105 - 0.08*timer);

                               transform.Rotate(Vector3.up,
7*Time.deltaTime, Space.Self);

                     }

          }
```

Robot Actuator Script

```
var c:int=0;

var run:boolean=false;


function Start () {

         transform.position=Vector3(1.945,4.00,5.69);

}


function Update () {

         if (run==true){

                    var speed:float=0.4;

                    if (c<1){

                               //movement from 1 to 2

                               if (transform.position.z > 4.6 &&
(transform.position.x < 1.95)){

                                          transform.Translate(0,0,-
speed*Time.deltaTime,Space.World);

                               }

                               //movement from 2 to 3

                               if (transform.position.x < 2.6 &&
(transform.position.z < 4.6)){


         transform.Translate(speed*Time.deltaTime,0,0,Space.World);

                               }

                               //movement from 3 to 4

                               if (transform.position.z < 5.7 &&
(transform.position.x > 2.6)){


         transform.Translate(0,0,speed*Time.deltaTime,Space.World);

                               }

                               //movement from 4 to 1

                               if (transform.position.x > 1.95 &&
(transform.position.z > 5.7)){

                                          transform.Translate(-
speed*Time.deltaTime,0,0,Space.World);

                               }
```

```
                                        if (transform.position.x < 1.95 &&
(transform.position.z > 5.7)){
                                                      c=c+1;
                                        }
                          }
             }
}
```

APPENDIX 2: FASTORY LINE DRAWINGS

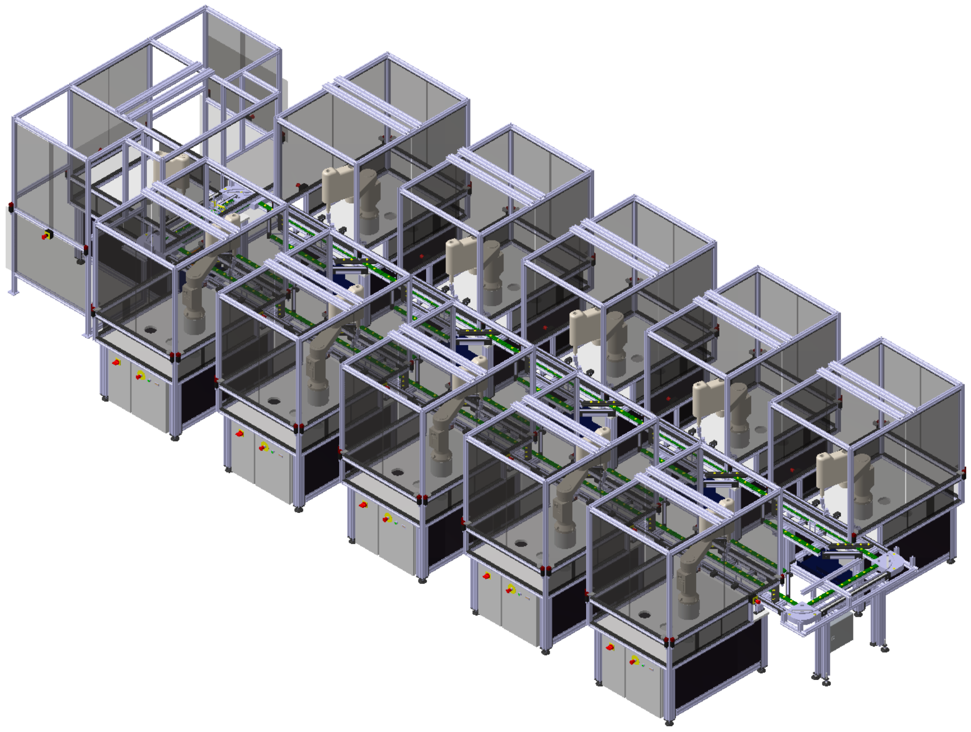


Figure A2.1. FASTory Line, isometric view

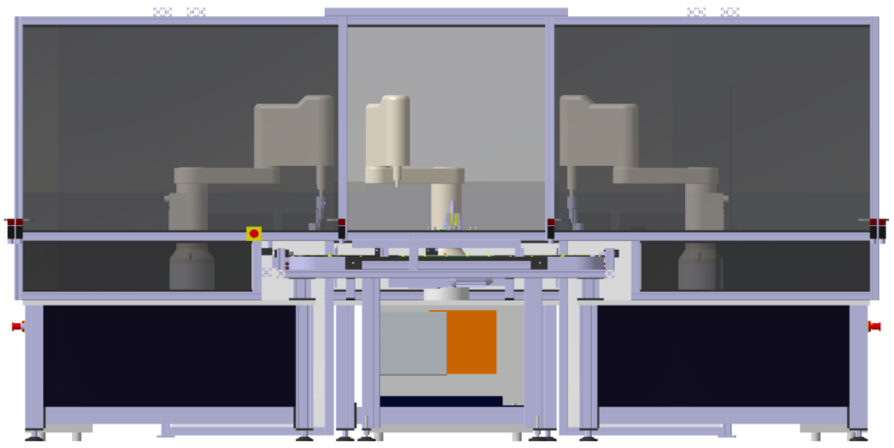


Figure A2.2. FASTory Line, front view

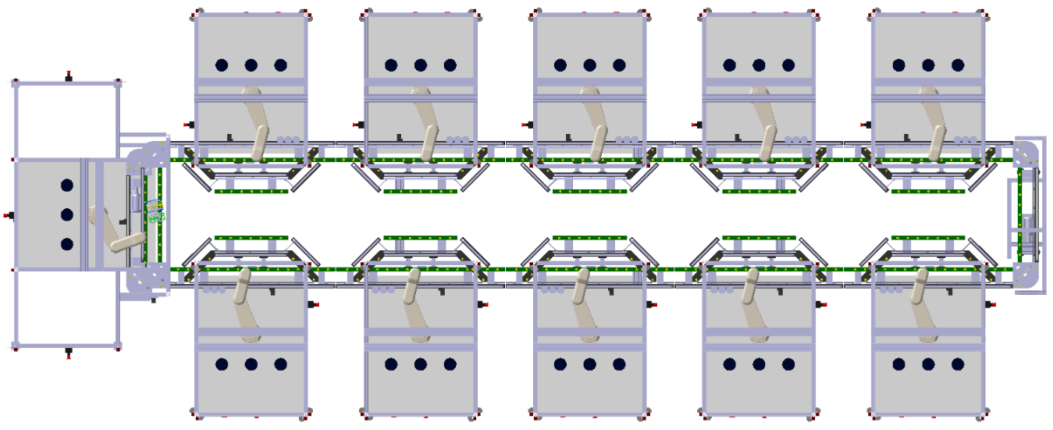


Figure A2.3. FASTory Line, top view

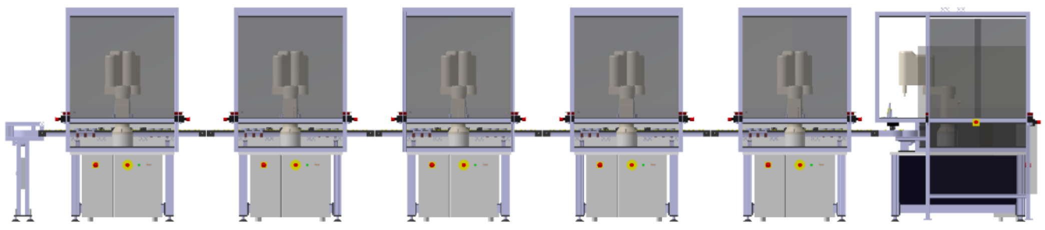


Figure A2.4. FASTory Line, side view

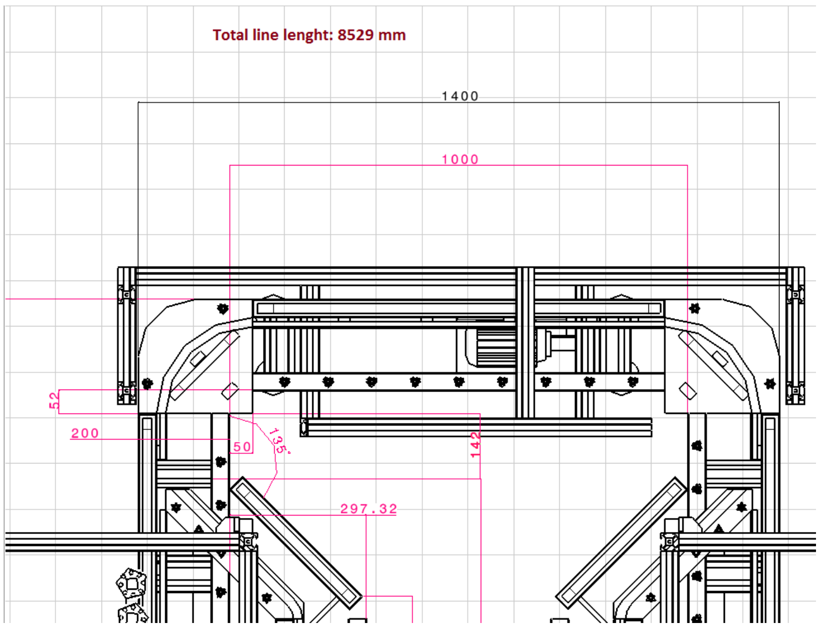


Figure A2.5. FASTory Line, detail and dimesions

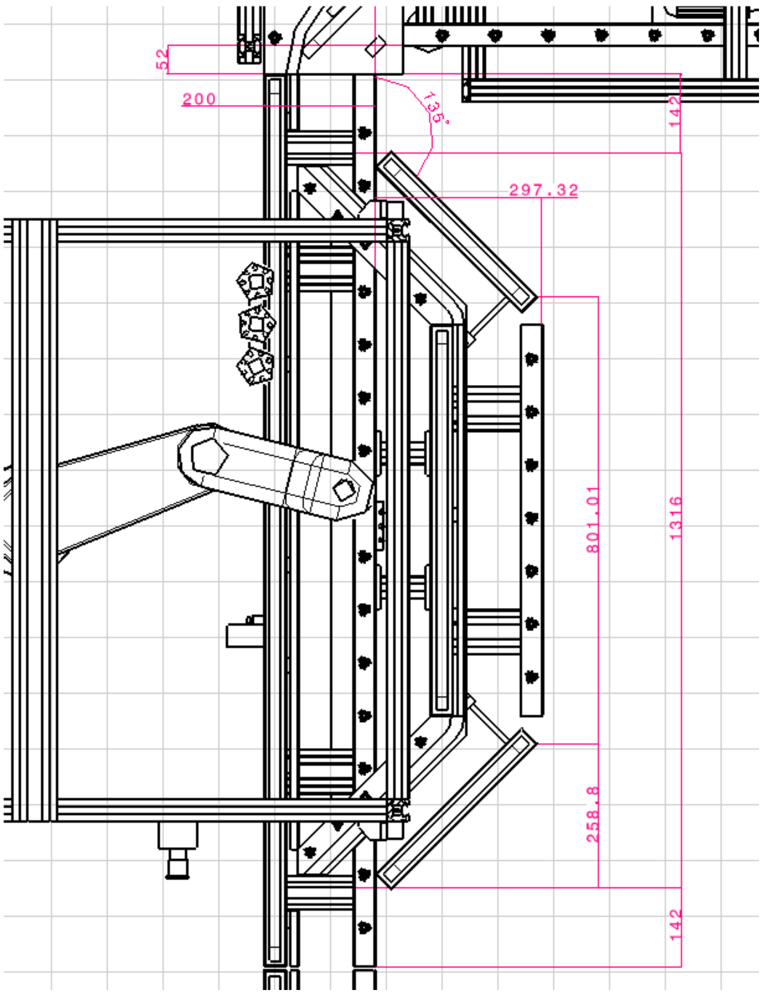


Figure A2.6. FASTory Line, detail and dimesions of the cells

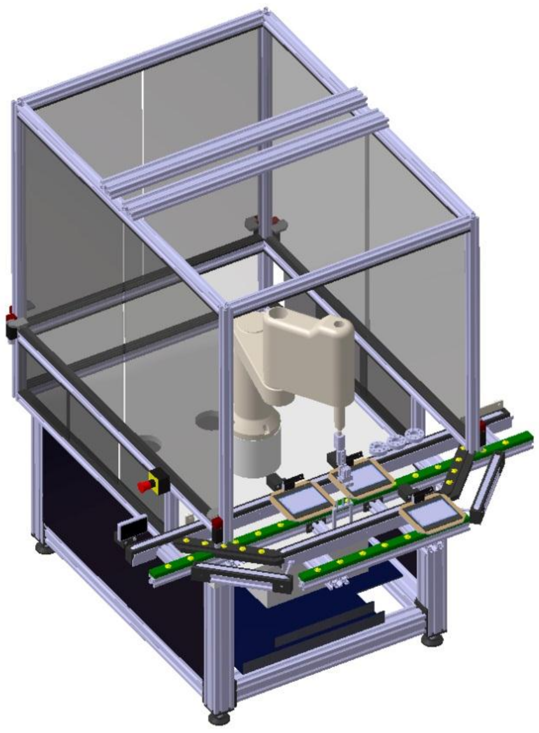


Figure A2.7. FASTory Workstation, isometric view

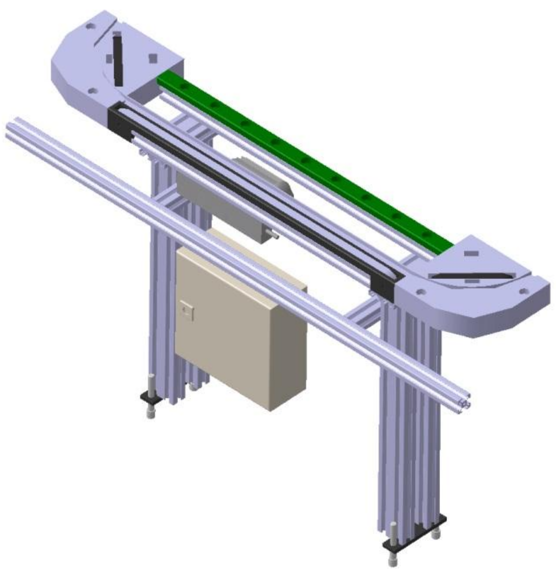


Figure A2.8. FASTory Conveyor, isometric view

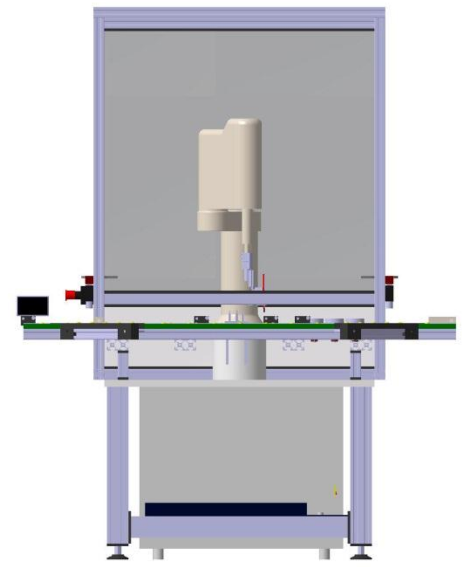


Figure A2.9. FASTory ***Workstation, front view***

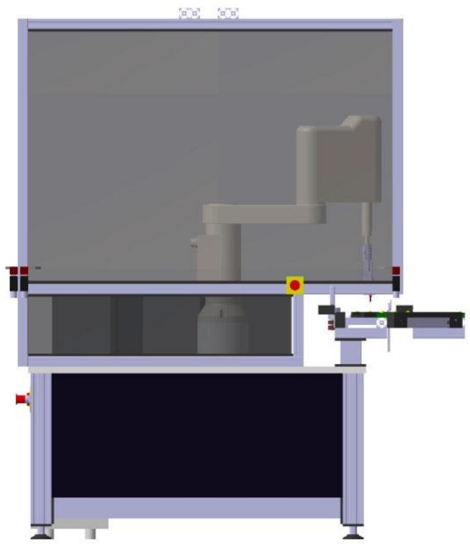


Figure A2.10. FASTory Workstation, side view

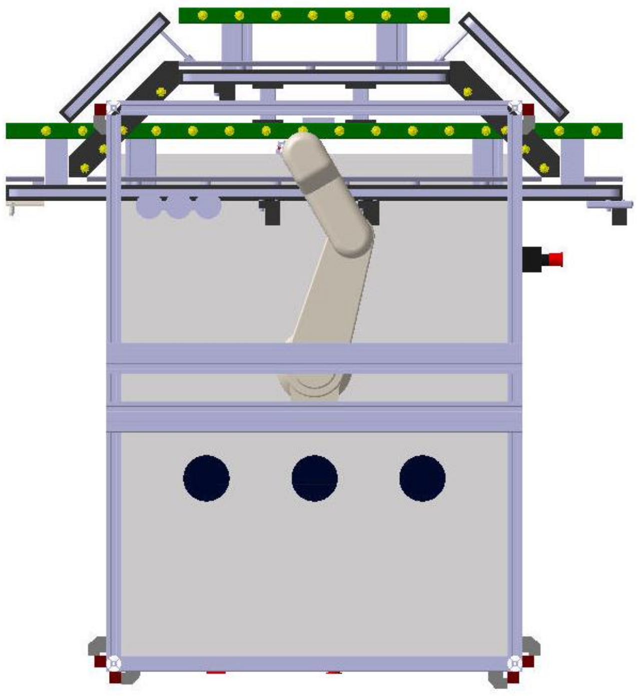


Figure A2.11. FASTory Workstation, top view