LINKING WISE-SHOPFLOOR TO AN ABB IRB-140 ROBOT
Remote control, monitoring, and programming of an ABB robot IRC 5 through the internet

Bachelor Degree Project in Automation Engineering
Spring term 2010
22.5 ECTS

Bentabol Muñoz, Emilio
Bosque Ibáñez, Carlos
González Ruiz, Pedro
Hurtado de Mendoza, Jose Manuel
Ruiz Zúñiga, Enrique

Supervisors:
Givehchi Yazdi, Mohammad
Adamsson, Göran

Examiner:
Holm, Magnus
Abstract

The aim of this project is integrate the new robot IRB140 from ABB inside the application Wise ShopFloor (Web-based integrated sensor-driven e-ShopFloor) and the integration of a web camera inside the application as well. In order to integrate the ABB IRB140 inside the application, a Java 3D model has to be created, the kinematics and collision constrains have to be defined also and the GUI application modified to fit the virtual model and the camera inside the application. The user has to be able to jog the web camera and zoom it.

Changes in the server side have been done in order to introduce new functionalities such as the sessions management, the communication mechanism now is more general using Java inheritance.
Copyright Statement

Submitted by Emilio Bentabol Muñoz, Carlos Bosque Ibáñez, Pedro González Ruiz, Jose Manuel Hurtado de Mendoza and Enrique Ruiz Zúñiga as a dissertation of Bachelor Degree Engineer at the Skövde University, Skövde (Sweden).

We certify that all the material in this final project that is not our own work, has been identified.

Skövde, Sweden, 26th of May 2010.
Acknowledgement

First of all, we would like to thank to the University of Skövde for welcoming us and teaching us a great deal of knowledge which will be extremely useful in our future careers. We would like to specially thank Lihui Wang for giving us the opportunity to learn about the future of manufacturing systems.

We would also like to thank to Magnus Holm, Fedrick Oldefors, and Göran Adamsson for always being available to help us with any questions we had.

It is necessary to acknowledge the invaluable help of Mohammad Givehchi Yazdi; without him the project could not have reached all the prospects.

We could not finish this acknowledgment without mentioning Laura Benson. She has helped us out so much with our poor English Grammar and look through the whole report. The project would not be understandable without her.

Finally, we want to thank to our families; without their help, we would never be here in Sweden, and would never have gotten the chance to learn as much as we have in Sweden.
Table of Contents

Abstract ........................................................................................................................... ii
Copyright Statement ................................................................................................. ii
Acknowledgement ..................................................................................................... iii
Table of figures ........................................................................................................... vii
Introduction ................................................................................................................... 1
1.1. Wise ShopFloor ..................................................................................................... 1
1.2. Aim and objectives .............................................................................................. 3
2. Real-Time issue ........................................................................................................ 5
3. Technologies ............................................................................................................. 7
3.1.1. Java 3D ............................................................................................................... 8
3.1.2. Java Applets, Servlet and Pushlet ................................................................. 10
3.2. HTTP ................................................................................................................... 11
3.3. TCP/IP ................................................................................................................. 11
4. Work Execution ...................................................................................................... 12
4.1.1. Forward Kinematics ........................................................................................ 13
4.1.2. Types of Movement ........................................................................................ 18
4.2. Collision Detection .............................................................................................. 19
4.2.1. 3D Model Controlled By Robot Controller ...................................................... 20
4.2.2. Animation ......................................................................................................... 20
4.2.3. Virtual Control And off-Line Mode ................................................................. 22
4.2.4. How the Collision Detection System works ..................................................... 26
4.2.5. How to create a new Collision Detection Behavior ........................................ 27
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.5.</td>
<td>Collision Detection Process</td>
<td>28</td>
</tr>
<tr>
<td>4.3.</td>
<td>Server</td>
<td>29</td>
</tr>
<tr>
<td>5.</td>
<td>Camera</td>
<td>37</td>
</tr>
<tr>
<td>5.1.</td>
<td>Camera Choice</td>
<td>37</td>
</tr>
<tr>
<td>5.2.</td>
<td>Camera Java Program</td>
<td>39</td>
</tr>
<tr>
<td>6.</td>
<td>Robot</td>
<td>47</td>
</tr>
<tr>
<td>6.1.</td>
<td>Robot components</td>
<td>47</td>
</tr>
<tr>
<td>6.1.1.</td>
<td>Manipulator</td>
<td>47</td>
</tr>
<tr>
<td>6.1.2.</td>
<td>Controller</td>
<td>48</td>
</tr>
<tr>
<td>6.1.3.</td>
<td>FlexPendant</td>
<td>49</td>
</tr>
<tr>
<td>6.2.</td>
<td>Connecting the robot</td>
<td>50</td>
</tr>
<tr>
<td>6.3.</td>
<td>Rapid</td>
<td>50</td>
</tr>
<tr>
<td>6.3.1.</td>
<td>Sockets</td>
<td>52</td>
</tr>
<tr>
<td>6.3.2.</td>
<td>RobotWare</td>
<td>52</td>
</tr>
<tr>
<td>6.3.2.1.</td>
<td>Socket Messaging</td>
<td>53</td>
</tr>
<tr>
<td>6.3.2.2.</td>
<td>Multitasking</td>
<td>54</td>
</tr>
<tr>
<td>6.3.2.3.</td>
<td>PC Interface</td>
<td>55</td>
</tr>
<tr>
<td>6.3.3.</td>
<td>RobotStudio Online</td>
<td>55</td>
</tr>
<tr>
<td>6.4.</td>
<td>Robot programs</td>
<td>56</td>
</tr>
<tr>
<td>6.4.1.</td>
<td>Errors</td>
<td>61</td>
</tr>
<tr>
<td>6.4.2.</td>
<td>World Zones</td>
<td>61</td>
</tr>
<tr>
<td>6.5.</td>
<td>Other possible solution</td>
<td>62</td>
</tr>
<tr>
<td>7.</td>
<td>Future improvements</td>
<td>62</td>
</tr>
<tr>
<td>8.</td>
<td>Conclusion</td>
<td>63</td>
</tr>
<tr>
<td>9.</td>
<td>References</td>
<td>65</td>
</tr>
<tr>
<td>10.</td>
<td>Appendix</td>
<td>67</td>
</tr>
</tbody>
</table>
Annex 1. Connecting the robot ................................................................. 67
Annex II. Socket messaging ................................................................. 69
Annex III. Rapid programming .............................................................. 73
# Table of figures

- Figure 1. Main view Wise Shop Floor ................................................................. 2
- Figure 2. Wise ShopFloor Architecture ............................................................ 3
- Figure 3. Leaf nodes .......................................................................................... 9
- Figure 4. Robot Virtual model set it up. .............................................................. 12
- Figure 5. Transform matrix structure ............................................................... 13
- Figure 6. Kinematics example .......................................................................... 14
- Figure 7. Robot Java3D Structure ..................................................................... 15
- Figure 8. Virtual Control .................................................................................. 19
- Figure 9. Robot virtual model with the three boxes. .......................................... 23
- Figure 10. Robot Virtual model with combine bounding .................................... 24
- Figure 11. Robot Virtual Model transparent boxes ............................................ 25
- Figure 12. Main architecture of the communications .......................................... 30
- Figure 13. Classes Diagram for the communications ......................................... 33
- Figure 14. How old desing is adapt to the new MachineInterface desing .......... 35
- Figure 15. Communications ............................................................................. 36
- Figure 16. Applet Distribution ......................................................................... 40
- Figure 17. MJPEG structure file ....................................................................... 43
- Figure 18. Camera control panel ...................................................................... 44
- Figure 19. Security problem connection ............................................................ 45
- Figure 20. Scheme of client-servers distribution ............................................... 46
- Figure 21. ABB manipulator 140 ..................................................................... 48
- Figure 22. ABB IRC5 controller ...................................................................... 49
- Figure 23. ABB Flex Pendant ......................................................................... 50
- Figure 24. Illustration of socket communication ............................................. 54
- Figure 25. ABB RobotStudio virtual FlexPendant .......................................... 56
- Figure 26. The six axes of the manipulator ....................................................... 57
- Figure 27. ABB IRC5 controller connectivity .................................................... 67
- Figure 28. Two main communication ports on the controller Computer Unit .... 68
Introduction

In recent times, global business and the decentralization of organization of the companies has become an area with many possibilities because of the necessity of cross-border collaboration between companies distributed around the world.

During the last decades, the web has gained broad acceptance and has been used as a medium to share information and knowledge. It is a useful tool for developing collaborative applications, working groups and organizations due to its platform and operating system transparency and easy-to-use interface.

Despite various accomplishments to date, a shared collaborative system for real-time monitoring, remote control, off-site inspection and collaborative troubleshooting is still missing from the literature. Our approach of Wise-ShopFloor targets this area and strives to engage a dispersed working group in a collaborative environment, allowing team members (engineers, managers, operators, etc.) to share real-time information through our platform. [1]

The project Wise ShopFloor was started in 2004 and this project continues the purpose of the application integrating an ABB IRB140 and a camera web. For a further overview of the project it is necessary to read the previous report from 2006.

1.1. Wise ShopFloor

According to [1], the global market is changing continuously and is now the dynamic global market force for companies to decentralize. The goal of the companies is to stay competitive so the companies with distributed factories or divisions are demanding one way to intercommunicate among themselves or their suppliers in real-time. This goal is even more important for the companies which manufacture many types of products in small batches.
The suitable tool for this goal is the Wise-ShopFloor. It is a software application to provide users an intuitive web-based and sensor-driven environment for system implementation (See figure 1). This tool uses Java technology and internet, allowing the client to connect directly to the ShopFloor via Internet. This allows the client to monitor and control the Wise ShopFloor in real-time.

The Wise ShopFloor is an alternative to the camera-based systems, but the main and most important difference is the aim of the real-time. This is possible because of the use of 3D virtual models of the devices used in Wise-ShopFloor that do not need much data to implement their movements and functionalities. Camera-based systems send video through the Internet, but the main problem with this system is the amount of data that is sent. In the camera-based system, the amount of data is very large and the bandwidth is not enough to support the delivered data in real-time.

Figure 1. Main view Wise Shop Floor.
Since the data that is sent via the Wise-ShopFloor is reduced, all the commands sent and received by the robot can be implemented in real-time. The architecture of Wise-ShopFloor is illustrated in the figure 2.

![Wise ShopFloor Architecture](image)

**Figure 2. Wise ShopFloor Architecture [1]**

### 1.2. Aim and objectives

The requirements of the project can be summarized as integrate a web camera and the new model of robot ABB IRB 140 in the graphical user interface of the Wise-ShopFloor, to be able to monitoring and controlling it. This requirement can be split in many other categories, such as:

- Create a Java 3D model of the robot ABB IRB 140.

- Integrate the IRB 140 in the graphical user interface of the application.

- Identify and hook up a web camera that can be controlled remotely, including tilting, panning, zooming and on/off control from the graphical user interface.
- Create a graphical user interface in Java for all the functions of the web camera.

- Define kinematics and motion constraints of the virtual model of the robot.

- Monitor and control the movements of the robot via the graphical user interface.

- Develop a graphical user interface based on the existing one that shares the same set of information for robot monitoring and control.

- Consider a multi-user environment when developing the testbed.
2. Real-Time issue

The aim of this project is to control and monitor robots or other devices in real time using the web-based application Wise-ShopFloor. The real-time issue plays a very important role in this project; otherwise, implementing the Wise-ShopFloor would not make sense. Here there are three definitions of real-time [2]:

**Hard Real-Time (HRT):**
- Used in systems where incorrect operation may lead to catastrophic events.
- Errors in HRT systems can cause accidents or even death. Such systems are typically found in flight or train control systems.

**Soft Real-Time (SRT):**
- An error in a SRT system will not cause loss of property or life.
- SRT systems are not as safety-critical as HRT systems, and should not be used in a safety-critical situation, like a self-phone or a web cam.

**Firm Real-Time (FRT):**
- It is a sub-class of Soft Real-Time Systems.
- There are no benefits from late delivery of service.

Firm Real-Time is the nature of this project since if some information is delivered late, it will be useless or even will lead to delayed monitoring and control. It is only necessary to send and receive through Internet some numerical values with the degrees of the joints, so it can be done in a fast way. Further, missed or delayed information will not lead to big catastrophes as death or big money loss, for example a video game.

“The largely reduced network traffic makes real-time monitoring and control practical for users on slow hookups. In addition to real-time monitoring and control, the concept can also be extended and applied to remote diagnostics and off-site inspection” [3].

Our project cannot meet HRT requirements, basically because some parts of the Wise Shop Floor application runs on a non-real time operating system; communication is
performed with TCP/IP over a shared network and the controller sometimes has tasks to perform higher priority tasks. A minimum response time expected for the controlling and monitoring should be in order of 10-100 milliseconds.
3. Technologies

In this chapter, the different technologies that have been used in this project will be discussed.

3.1 Object Oriented Programming

Object-oriented programming (OOP) is a programming language model organized around "objects" rather than "actions" and data rather than logic. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data.

The programming challenge was seen as how to write the logic, not how to define the data. Object-oriented programming takes the view that what we really care about are the objects we want to manipulate rather than the logic required to manipulate them. Examples of objects range from human beings (described by name, address, and so forth) to buildings and floors (whose properties can be described and managed) down to the little widgets on your computer desktop (such as buttons and scroll bars). [32]

3.1. Java

The Java™ programming language is a general-purpose, concurrent, class-based, object-oriented language. It is designed to be simple enough that many programmers can achieve fluency in the language. The Java programming language is related to C and C++ but is organized rather differently, with a number of aspects of C and C++ omitted and a few ideas from other languages included.[31]

One of the advantages of Java is that you can run it on every computer which have installed JVM (Java Virtual Machine).
3.1.1. Java 3D

The Java 3D API is an application programming interface used for writing three-dimensional graphics applications and applets. It gives developers high-level constructs for creating and manipulating 3D geometry and for constructing the structures used in rendering that geometry. Application developers can describe very large virtual worlds using these constructs, which provide Java 3D with enough information to render these worlds efficiently.

Java 3D delivers Java's "write once, run anywhere" benefit to developers of 3D graphics applications. Java 3D is part of the JavaMedia suite of APIs, making it available on a wide range of platforms. It also integrates well with the Internet because applications and applets written using the Java 3D API have access to the entire set of Java classes.

The Java 3D API draws its ideas from existing graphics APIs and from new technologies. Java 3D's low-level graphics constructs synthesize the best ideas found in low-level APIs such as Direct3D, OpenGL, QuickDraw3D, and XGL. Similarly, its higher-level constructs synthesize the best ideas found in several scene graph-based systems. Java 3D introduces some concepts not commonly considered part of the graphics environment, such as 3D spatial sound. Java 3D's sound capabilities help to provide a more immersive experience for the user. [30]

Before describe what is a Transform3D or TransformGroup, it is necessary to highlight tome other concepts such as:

SceneGraph: A scene graph is a "tree" structure that contains data arranged in a hierarchical manner. The scene graph consists of parent nodes, child nodes, and data objects. The parent nodes, called Group nodes, organize and, in some cases, control how Java 3D interprets their descendants. Group nodes serve as the glue that holds a scene graph together. Child nodes can be either Group nodes or Leaf nodes. Leaf nodes have no children. They encode the core semantic elements of a scene graph— for example, what to draw (geometry), what to play (audio), how to illuminate objects (lights), or what code to execute (behaviours). Leaf nodes refer to data objects, called NodeComponent objects. NodeComponent objects are not scene graph nodes, but they
contain the data that Leaf nodes require, such as the geometry to draw or the sound sample to play. see figure [34].

Figure 3. Leaf nodes [33]

1. Virtual Universe: Java 3D defines the concept of a *virtual universe* as a three-dimensional space with an associated set of objects.[33]

2. BranchGroup: The BranchGroup serves as a pointer to the root of a scene graph branch; BranchGroup objects are the only objects that can be inserted into a Locale's set of objects. A subgraph, rooted by a BranchGroup node can be thought of as a compile unit.

-Transform3D

As is defined in [4], a generalized transform object represented internally as a 4x4 double-precision floating point matrix. The mathematical representation is row major, as in traditional matrix mathematics. A Transform3D is used to perform translations, rotations, and scaling and shear effects. The Transform 3D is used by methods such as rotation interpolator and so on.
-TransformGroup

According to [4], a group node containing a transform, the TransformGroup node specifies a single spatial transformation, via a Transform3D object, that can position, orient, and scale all of its children. The specified transformation must be affine. Further, if the TransformGroup node is used as an ancestor of a ViewPlatform node in the scene graph, the transformation must be congruent-only rotations, translations, and uniform scales are allowed in a direct path from a Locale to a ViewPlatform node. The transform groups form the tree structure of the robot.

3.1.2. Java Applets, Servlet and Pushlet

Java Applets is a code which has to be executed over a JVM and it can be incrusted into an HTML code. The applets are necessary in our project because the application will run over a web page.

The Java Servlet code is executed in the server. It is an object which is executed inside a server; and is especially designed to offer dynamic content from a server [4].

Pushlets are a mechanism used for multicasting messages along the clients. This framework is based on the HTTP protocol which is very well-known and reliable. According to Just van den Broecke [1], Pushlets provide an open stream-based communication between the server and the client, so it allows a web page to be periodically updated by the server without explicit requests from the client, as it happens in HTTP.

Its operation is quite simple. Every time a client wants to receive messages from the server, it sends a “subscription” message to a specific subject. On the other hand, the server “publishes” messages to some subjects as well. The Pushlets purpose is to make sure that all the messages that are published to a subject in the server and will eventually arrive to the subscribed clients. Although the main advantage of Pushlets is to allow distributing messages on to the clients, it also allows the clients to publish messages to the server. This gains importance when controlling the robot, since sending messages from the client to the robot, through the server, becomes essential.
The main reasons why Pushlets was chosen for the project include:

Server does not need to have information about the clients to distribute the message along the subscribed clients

Pushlets takes care of the communication, so the server only has to dispense the message to the framework and this will carry out the message distribution.

### 3.2. HTTP

HTTP (Hyper Text Transfer Protocol) is a standard client-server protocol. It works through request-response. Usually, server stores HTML files and images, which can be displayed by a standard browser, as Mozilla Firefox, and the client sends request for getting those files so they are shown in the browser. The protocol operation is very simple, when the client requests data; it opens a connection with the server and sends an HTTP request (which follows a specific format). This request embeds the path of the file the client wants to access in the server. When this request arrives to the server, it searches for the file the client is requesting and sends a response to the client with the file. After this, the connection is close. HTTP is based on TCP protocol and it opens and closes a TCP connection for every request-response communication between the client and the server.

### 3.3. TCP/IP

TCP/IP (Transmission Control Protocol) is a standard communications protocol. It works over IP protocol, and basically assures that every IP packet of information between two nodes in the network will arrive, even if the network fails. For that, it uses acknowledge (confirmation) packets, flow control and data exchange rate.
4. Work Execution

The first step in every project is the documentation. So, for the first one week and a half, the work was look for information about the different technologies, programming languages and knowledge which are necessary to perform the project. 4.1. Developing Virtual 3D Model

The virtual 3D model of the robot with direct kinematics was developed using Java 3D and Wise-ShopFloor model base objects. CAD models of robot components including links, motors and cables were imported from VRML format obtained from ABB website, it is showed in figure 3. Since the data type of the links was VRML1 and the employed Java3D VRML loader component supports only VRML2 format, a converter was needed to convert the CAD models from VRML1 to VRML2.

Figure 4. Robot Virtual model set it up.
4.1.1. Forward Kinematics

Forward Kinematics is a computation of the position and orientation of the robot's end effector as a function of its joint angles [5].

Given all of the manipulator's joint and link values (angles and lengths), what is the position and orientation of the hand?

The rotation matrix which calculates the rotation of one link around the different axis are:

Rotation around X axis =
\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & \cos \theta & -\sin \theta \\
0 & \sin \theta & \cos \theta \\
\end{bmatrix}
\]

Rotation around Y axis =
\[
\begin{bmatrix}
\cos \theta & 0 & \sin \theta \\
0 & 1 & 0 \\
-\sin \theta & 0 & \cos \theta \\
\end{bmatrix}
\]

Rotation around Z axis =
\[
\begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\]

Homogeneous Transform Matrix:

This matrix represents mathematically the pose of a frame relative to another frame.

\[\text{Figure 5. Transform matrix structure. [2]}\]
The robot can now be cinematically modeled by using the link transformations:

\[ T_n^0 = T_1 \cdot T_2 \cdot T_3 \ldots T_n \]

Where \( T_n^0 \) is the pose of the end effector relative to base.

Example

\[
\begin{align*}
\vec{x}_3 &= 
\begin{bmatrix}
\cos \theta_2 & -\sin \theta_2 & 0 & l_2 \\
\sin \theta_2 & \cos \theta_2 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix} \\
\vec{x}_2 &= 
\begin{bmatrix}
\cos \theta_1 & -\sin \theta_1 & 0 & l_1 \\
\sin \theta_1 & \cos \theta_1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix} \\
\vec{x}_1 &= 
\begin{bmatrix}
\cos \theta_0 & -\sin \theta_0 & 0 & l_0 \\
\sin \theta_0 & \cos \theta_0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\end{align*}
\]

Figure 6. Kinematics example. [2]

The robot uses this kinematics in order to calculate the position of the TCP, but in the IRB140 virtual model the kinematics are implemented as in Forward Kinematics.

The structure of our robot is a tree parent-child structure, all the TransformGroups are children of the TransformGroup stBase, and the TransformGroup of the link 1 is the parent of the TransformGroup of the link 2 and so on until the gripper. This means that in other to know the position, for instance, of the gripper all the Transform3D matrix which belongs to each TransformGroup have to be multiplied as in forward kinematics and the result will be a Transform3D which represents the position of the gripper. All these calculations will be calculated automatically by Java 3D.
Figure 7. Robot Java3D Structure
Since the rotation of the TransformGroup is around one axis which is parallel to the local coordinate system. The Wise-ShopFloor’s framework model base objects was extended with the class DynamicComponent2 which perform this rotation.

In addition in order to do that this steps have to be performed:

1- Move the component so that the axis of rotation is translated to its correspondent axis of the local coordination system.

2- Rotate the component the desire angles around the coordination system axis.

3- Translate back the component with a vector with the same size of previous translation and in opposite direction.

The equivalent of the matrix motion matrix is multiplication of the equivalent matrix of the above steps. Since the axis of rotation in the robot model has such conditions in its locate coordinate system.

This is one method which is inside DynamicComponent2 which performs the rotation about X. There is one method for the rotation about each axis:

```java
public void setRotationAxisParallelToXX(double rotationAxisY,
                                          double rotationAxisZ) {
    rotAxisX = null;
    rotAxisY = rotationAxisY;
    rotAxisZ = rotationAxisZ;
    rotAxisAlong = Transform3Dutil.AXIS_XX;
}
```

Since the movement of the animation is performed by the Java class RotationInterpolator and this class rotates in Y by default. DynamicComponent2 contains the method which provides a new Transform3D matrix that allows to change its coordinate system so that rotation around Y axis is identical to the one of the desire axis.
@Override
public void CreateInterpolator(float minValue, float maxValue) {

    super.CreateInterpolator(minValue, maxValue);

    if (this.interpolatorType == ROTATION_INTERPOLATOR) {
        //AxisAngle4d axis=null;new Matrix3d()
        if (this.rotAxisX == null)
            this.transform.set(Transform3Dutil.createTransform3D(0,
                rotAxisY,
                rotAxisZ,
                0, 0,
                Math.PI /
                2));

        else if (this.rotAxisY == null)
            this.transform.set(Transform3Dutil.createTransform3D(rotAxisX,
                0,
                rotAxisZ,
                0, 0,
                0)); //

        else if (this.rotAxisZ == null)
            this.transform.set(Transform3Dutil.createTransform3D(rotAxisX,
                rotAxisY,
                0,
                -Math.PI /
                2, 0,
                0)); //

        this.transform=Transform3Dutil.createTransformChangeCoordTransXZRotYY( rotAxisX, rotAxisZ, 0);

        else if (this.rotAxisZ == null)
            this.transform.set(Transform3Dutil.createTransform3D(rotAxisX,
                rotAxisY,
                0,
                -Math.PI /
                2, 0,
                0)); //

        this.transform=Transform3Dutil.createTransformChangeCoordTransXZRotYY( rotAxisX, rotAxisZ, -Math.PI/2);

        ((RotationInterpolator)this.interpolator).setTransformAxis(this.transform);
    }
}
4.1.2. Types of Movement

Previous to explaining the next steps of the project, it is necessary explain the different types of movements that can be performed by the virtual 3D animation of the robot. The three different modes to move the virtual robot: offline, animation, online control (Virtual Control or Real control). Using the offline mode the user can move the robot using jog buttons, see figure 4. This type of movement is to show how the robot can move inside the canvas while the robot is offline. In later chapters, it will be explained how the collision issue is avoided.

The animation model is just for exhibition, to show how the robot can move. The robot moves each joint from the maximum value to the minimum value joint. During the online control of the robot, the client connects with the server, which means the client is online. As it is shown in figure 4, the real robot moves according to the client’s request. At the same time, the robot’s 3D model is tracking the movements of the real robot. The collisions in this type of control are managed for the robot.

When the machine is not connected to the network and the user tries to connect it, the Wise-ShopFloor application shows a message to the user saying: “It is not possible to connect with the machine, Do you want Virtual Control?”, then the user can select this option. So now the user has a control browser of a virtual machine that he can control and see. If a second user wants to monitor the movements of the virtual machine done by the first user, then he can simply open a new browser and select virtual monitor of that virtual machine. Then if the first user moves the Virtual Control browser, the second user can also monitor the movement of the virtual machine from his own browser, as is showed in the figure 7.
4.2. Collision Detection

In this section it will be explained the solutions for the collision management, which depends on the type of movement used by the user of the 3D model.

In any 3D universe, collision detection system must exist. It will detect the collision between two or more objects and it allows desired actions as a roller back of the movement or any specific desired action by the user. Articulated models need the abilities of the collision detection to prevent the movement over other objects in the scene. The new collision detection system created in this project could be used in the future for others 3D models in Wise-ShopFloor to prevent collisions. For instance, collisions between two or more robots in the same shop floor, or a robot and fixed parts of its environment, such as cells or other machines inside its workspace.

As explained earlier in this report, there are four different kinds of controls for 3D model.

- 3D Model controlled by Robot Controller
- Animation
- Virtual Control
- Off-line model jogging

4.2.1. 3D Model Controlled By Robot Controller

When the user is controlling the robot in real time, Wise-ShopFloor receives the exact position of each link of the real robot from the controller, which administers the collision control in three different ways:

1. - By software, in which the controller knows the maximum and minimum default angles for each link and it is not going to allow to reach any angle out of that range.
2. - Mechanical limits, when it is reached, the robot movement is automatically stopped.
3. - Electric control, when one mechanical limit is reached, a voltage peak is created and consequently the robot movement is stopped.

These three conditions are necessary, but not enough to avoid the collisions within itself. For that purpose, two different ways can be used: to define a workspace in the robot in the same way as with the solution in collisions control for off line or virtual control. This option had to be rejected because, at present, this function is not available in robot controller. Therefore, the solution is to use the Motion supervision function, which together with workspace will be explained carefully in the World Zones section, in Robot program.

4.2.2. Animation

For this kind of control, the previous work of other students has been reused and has been adapted to the requirements of this project. Since the animation is used only to see in a main manner how the robot movements are, it will be used in exhibition mode only, so it is not necessary to have really accurate movements. The solution that has been used to control collisions is to limit the degrees of freedom of the axes two and three so when the user runs the animation the values of these axes are changed automatically.
@Override

    public void setAnimation(boolean flag) {
        // if the flag is true, then the joint limit is modified,
        // if it is false the default joint limits are used.
        dyJ2.CreateInterpolator(flag ? -65 : dyJ2.getNegStroke(), dyJ2.getPosStroke());
        super.setAnimation(flag);
    }
4.2.3. Virtual Control And off-Line Mode

The handling of the collisions for those modes of control is the same. When the user chooses virtual control or off-line, then it is needed to be a collision detection system that is more accurate than the previous ones. One that will be able to throw back the movements of the real robot with great precision.

A collision is detected in Java 3D when the bounding of two bodies starts to be in contact. The aim of this section is to explain how to define the bounding of an object which is able to collide in a 3D model and the specific solution for that project. The most important limitation of Java 3D is that only one collision can be managed at the same time accordingly until one collision is not finished another one is detected.

Working with collision detection in Java 3D, it is necessary to add bounding which contains the object for each part where is necessary to detect the collision. There are two different techniques to do that: SetBoundsAutocompute() and SetBounds(). The former is used for Java 3D to calculate the limits of the object automatically. This method works perfectly when the objects are simple geometric forms, like cubes or spheres. As a default setting, SetBoundsAutocompute() is activated for all the objects, but it can establish to false. The second technique is to use SetBounds(), in which the user can create a bounding of a desired size using different methods like BoundingBox() or BoundingSphere(). The only requirement for good performance is that the bound may be bigger than the object; if this is not the case, the program will use the bound calculated by Java 3D. Once the bounding is established, it can be used with collision detection purposes using SetCollisionBounds() and given as an argument to the previous bound.

The robot IRB140 is an articulated robot, it involves that from the beginning when the model is loaded in the applet the collisions are detected among the adjacent links. Since this moment it is impossible to detect another one collision.

The problem was solved with the function SetCollidable() which allows the establishment of one of the links of the robot as collidable or not. Due to the way in which the robot has been built, the collision will be produced between the Gripper
(StGripper), Joint 4 (dyJ4), base (StBase) or the floor (stFloor). The function SetCollidable(), together with Collision Behaviour, only tell the collision produced between Joint4 and Gripper to solve the problem.

It is really important to highlight that the SetCollidable method needs the parent of the entire robot’s elements to be established as true. In the opposite case, it will be established automatically as false and his children and the collision will not be detected among them.

Once the problem was solved, the next step was to create a bounding box, like it would be used in the function world zone in the real robot. This method would consist in a box which involves those parts of the robot that are exposed to be crashed by him; such as the link1, base and the platform. In order to that, a bounding base was created which covers the base and the floor. A new bounding for the base would be added instead of the bounding which was created by Java 3D automatically. In the same way, it would create three more bounding boxes, one for each link, which have to be watched: link1, link4 and Gripper as it is showed in the figure 8. These parts would be added if any of the bounding box crashes between each other and the program will know if a collision happened.

At first sight, this solution seemed the most suitable in order to solve the problem, but then a new problem presented itself. The bounding box should have covered the entire link1 but it was too big and restricted the robot’s movement too much, which didn’t allow the robot to have the same freedom of movements as the real one.
As a solution for this problem, it was thought to select the link 1 as collidable and extend the bounding of the base. As a result, this bounding covers the part of the link1 which is interesting for the collisions.

In order to prove this solution, the method combine() from the Bounds class was used. This method allows considering two bounds as one. The result was to reduce to the minimum expression of the bounding which covers the link1 (see figure 9), since the bound belongs to the base and the joint 1, it was already completely covered.

![Figure 10. Robot Virtual model with combine bounding.](image)

Unfortunately, it was not possible to make it work properly because Java 3D was not paying attention to this new extension of the base’s bounding for the collisions; therefore the gripper could collide with the link1.

Once this point was reached, two problems presented themselves:

- In order to establish a new bounding to an object, it has to be bigger than the object; but with this solution, the bounding was too big.
- The method combining bounds was not the expected behaviour and the collisions of link1 were not detected.

Then, the adopted solution imitated the behaviour of the bounding.

In order to do that, two boxes were added to the graphic scenarios which simulate the desired behaviour of the combined method and a third box, which replaced the link 4 and Gripper bounds (see figure 10). These boxes are geometric figures created by Java 3D. As result, a perfect automatic calculation of their bounds is achieved, which makes it possible to use and to solve the collisions if the boxes are located in the places on which the collisions might occur.

After that, it is just necessary to make the boxes invisible and the desired behaviour will be reached. In the real world, the same collision behaviour will be reached using the function world zone of the robot controller.

![Figure 11. Robot Virtual Model transparent boxes.](image)

Once the collisions are detected and it works properly, a Java 3D behaviour allows realization of the required actions once the mentioned collision it is detected.
In this project, a general system of detection collision has been created, which will be used in Wise-ShopFloor in the next models independently the type of machine.

For this project, the system realized a rollback movement to one position previous to the collision. This behaviour can be used as a standard for all of those models that just use Static Components and Dynamic Components. This means that if it is used in a model with components of the type Reactive Components, the position of those objects will not be refreshing to the previous position.

4.2.4. How the Collision Detection System works

The Collision Detection System is a new feature created for this version of the Wise-ShopFloor project.

The system allows to detect, to store, to inform and to make actions when two bodies are colliding.

The whole Collision Detection System involves six classes that work together. These classes are:

- DynamicComponent
- MachineType
- CollisionDetectionEventData
- CollisionDetectionBehaviour
- CollisionDetectionObservable
- CollisionResolverBehaviour

An overview of the working behaviour of the system could be summarized as the following:

As soon as a new model 3D want to be created, it should extend of MachineType class and it will be built of a chain of Models components. This version of the project includes a new method whereby the last thirty positions of the robot will be stored. With
this information already stored, there are three classes responsible if a collision has been detected CollisionDetectionBehaviour, CollisionDetectionEventData, and CollisionDetectionObservable. When this happens, CollisionResolverBehaviour will be called and it will be undoing the last movements until the collision finalizes.

4.2.5. How to create a new Collision Detection Behaviour

A new Collision Detection Behaviour should be created when an action is needed because two objects are in contact.

These actions could undo a movement, trigger an alarm, change the color of the objects which are colliding, print on the screen the objects collided, or another action needed.

To create a new object, CollisionDetectorBehavior is needed one Model Component and one schedulingBounds. This Model Component will be the one which is causing the collision itself and the schedulingBounds is a spatial boundary where the behavior will take place.

Example:

CollisionDetectorBehaviour myColDetGripper = new CollisionDetectorBehaviour(stGripper, Bounds);

The pattern followed in the design of the collision detection system has been the Observer Pattern.

After creating the new collision behaviour is needed to call AddCollisionDetectionBehaviour method, in which the behaviours will be added to a list of behaviours, the observers will be added to these behaviours and also the collision detection will be enabled or disabled.

Example:

this.AddCollisionDetectionBehaviour(myColDetGripper);
4.2.5. Collision Detection Process

Once a Collision Detection Behaviour is created, the system is ready to work. In this section, the relationship between the classes that form the system will be explained.

When a collision happens, CollisionDetectionBehaviour class will identify it and the process will start.

The process starts creating an object of CollisionDetectionObservable, this object will be responsible to inform all the Observers. An observer is a class which has been subscribed to the CollisionDetectionBehaviour. This means that these classes are interested in obtaining information about this collision.

collisionDetectorObservable = new CollisionDetectorObservable();

Every time a behaviour is created it has to be added to the Scene. In this case, the behaviour is added to the TransformGroup of the Model Component.

    component.getGroup().addChild(this);

Every behaviour must contain at least two methods: initialize() and processStimulus(). As is explained in Java 3D tutorial [1], the initialize method is invoked when the scene graph containing the behaviour class becomes live. The initialization method is responsible for setting the initial trigger event for the behaviour and setting the initial condition of the state variables for the behaviour. The trigger is specified as a WakeupCondition object, or a combination of WakeupCondition objects. The processStimulus method is invoked when the trigger event specified for the behaviour occurs. The processStimulus method is responsible for responding to the event.

For this system, the initialize method defines the trigger options for collision purposes, WakeupConditionEntry, WakeupConditionMovement, WakeupConditionExit.

In the processStimulus method, an instance of CollisionDetectionEventData is created. This object stores the trigger criteria and the model component which is causing the collision; this information is sent to the observer through the NotifyObservers method of collisionDetectorObservable class.
This NotifyObserver method will call the Update method of each observers which will set the object setIsCollided to true if the trigger event was WakeupConditionEntry or WakeupConditionMovement and to false when the trigger is WakeupConditionExit.

This version of the Collision Detection System is intended to undo the last movement by which the machine was colliding. When the object setIsCollided has the value true then CollisionResolverBehaviour is called. As all the behavior classes it has a method called processStimulus which will respond to the event, calling the rollebackMovement method from MachineType which will undo the last movement from the last stored position, this action will continuously made until the collision is finished and the object setIsCollided is set to false, which means the collision is finished and the collision detection system has finished its work.

### 4.3. Server

The content of this part has been extracted from the bachelor project report of Pedro González Ruiz, for more information consult the correspondent report. The server works as an interface between the client and the machine. It gets the incoming messages that come via Pushlets from the client and sends them to the robot through sockets.

The main characters in the Server are DSFAccessManager, SessionManager and MachineInterface.
MachineInterface represents a real machine and it provides a set of methods to connect, disconnect, send messages, start service communication and stop service communication. The services that machines can provide so far are Monitoring and Controlling.

DSFAccessManager is the gateway to the system, it gets the Pushlets messages from the client and, depending on the requests, it opens the corresponding communications with the machine. It stores a collection of MachineInterface to start the very communications, and another collection of clients sessions that is used by the SessionManager.

This project includes, for the first time, session management. This is performed using the SessionManager that is already provided in Pushlets. In order to adapt the SessionManager to our software, some features had to be added so that when a session expires, it closes, if necessary, the communication in DSFAccessManager. SessionManager is also responsible to send “heartbeat” messages to the clients to notice if a client has disconnected or crashed.

The communication performance has been improved in this project, making it more general and in principle, more efficient. The previous projects used one only thread to
perform the communications with the machine. This created a bottleneck when there were too many incoming and outcoming messages. Furthermore, to perform the communication with all the machines, a class called WhaliControl was used. This class was developed thinking of communicating with the robot WhaliMill, thus, using this class to perform the communication with every machine made the system very coupled to this machine, and also, not very general. What would happen if the commands of another robot were different? What if some robots do not even use sockets for communications?

For these reasons, the class MachineInterface was created. This class provides a set of methods to create connections, close connections, and, in general, communicate with a general machine independently of the mechanism used to send or receive the data.

In order to reduce the aforementioned bottleneck, the communication with the machines has been redesigned. Instead of using only one connection for sending and receiving the data, two are used. Incoming communications have been separated from outcoming communication. This makes the communication more reliable in case of failures, also loosening up the bottleneck.

In addition, within incoming communications in the server, there has been another separation. This one, regarding, threads of execution:

- One thread takes care of receiving data from the machine, through the incoming socket communication.
- Another thread is responsible for distributing messages along the clients using Pushlets.

Those two threads are running in parallel and are sleeping when there are no messages. The fact of separating the communication in two parallel executions makes the system more efficient, especially in the present time, so that multi-core processors are quite well-spread.

Regarding outcoming communications from the server, one thread has been used to send data to the machine. It was thought to use two threads at the beginning, one for sending data to the machine and another one for receiving Pushlets messages from the
clients. However, there was no point in the second one due to the synchronous nature of
the messages coming from the clients; messages do not arrive in a periodic way.
Therefore, there is only one thread that is asleep when there are no controlling
commands. This thread is woken up when new commands arrive (via Pushlets) and,
after transforming them, sends the data to the robot (through sockets).

The classes used to perform the communications with the robot are
DeviceDataReceiverInterface, DefaultDataReceiver, DeviceDataSenderInterface,
DefaultDataSender, DeviceDataSubscriberInterface, IRB140Monitor,
IRB140ControlAdapter and Packet.
Figure 13. Classes Diagram for the communications [6]
DeviceDataReceiverInterface is an interface that provides a set of methods for receiving data from the robot; no matter the mechanism used to get the data. Methods such as connect, startReceiving, stopReceiving are provided. DefaultDataReceiver implements that interface using socket to perform the communications and it gets packet of information from any machine that works with sockets, not only the robot ABB-IRB140.

DeviceDataSenderInterface is the analogue of DeviceDataSenderInterface for sending data to the robot. DefaultDataSender implements the former interface using sockets for the communication. It also can send packets of information to every machine that supports sockets, besides IRB140.

DeviceDataSubscriberInterface is an interface that provides a set of methods to distribute the messages along the clients independently of the mechanism used to distribute all that information. IRB140Monitor implements the latter interface using Pushlets for that. This class is bounded to the robot ABB-IRB140 due to the format of the messages that the buttons of the graphical interface send.

Finally, Packet is a class that represents a chunk of information coming from the robot. It has a header that is an integer to differentiate distinct packets.

The MachineInterface used to control and perform the communications with the robot ABB-IRB140 is MachineIRB140, and it contains instances of DefaultDataSender, DefaultDataReceiver and IRB140Monitor in order to communicate both with the machine and the clients.

As it was told before, MachineInterface is an interface that provides a set of methods for communicating with any machine or robot. However, the previous design did not adapt very well to this point, so the classes for communicating with WhaliMill and Tripod were the only ones that were being controlled before and were encapsulated in wrapper classes so they adapt to the new design. The following picture may help demonstrate this point.
Figure 14. How old design is adapt to the new MachineInterface design [6]
MachineControlInterface is the class responsible for getting controlling commands from the clients, through Pushlets. It is a class that was already created in the previous versions. It works as an adaptor for both real and virtual controlling.

MachineAbstract is a class that applies the new design, using the two communications and three threads. The class that represent ABB-IRB140 and all the classes that represent any other new robot should extend this class to make more efficient the communication with the machine.

On the other hand, in the figure can be seen the wrapper classes MachineWhali and MachineTripod. Both of them store inside a WhaliControl instance to communicate with the robot.

The server is between the parts of the application that undergo biggest changes. A new re-design has been applied in order to make the classes more general, easing future changes. Performance has also been another point when thinking the design, so it takes full advantages of the new technology processors.

Figure 15. Communications.
5. Camera

For this project it is required a special camera, not a normal webcam because it has to be possible to control certain movements of it, like pant till and zoom, so the kind of camera needed is a PTZ camera, that should be IP as well.

5.1. Camera Choice

After a research on internet about the cameras, communications between camera and server, movement of the camera, etc., a decision was made according to the specifications that were needed, It has to take an Internet Protocol (IP) camera that is Close Circuit Television (CCTV), which means that the camera uses internet protocol to establish the communication and send image data and control signals through Ethernet. These kinds of cameras have some advantages:

Cost advantages: Due to general-purpose IP networking equipment infrastructure, low cost of cabling installation (coaxial cables) and reduced space requirements.

Flexible image format: Compatible with an amount of image resolutions including standard analogical CCTV resolutions and Megapixel resolutions, Progressive scan allows dispose of still images is made in a better video quality. This is given for a fast moving target, improving the ability to choose the image size and resolution for each camera without using a specific hardware that converts the analog signal to digital one for storage on hard disks, it is important to choose the right codec as M-JPEG, MPEG-4, etc.

Extensible Network Infrastructure: Can be used an existing IP cabling structure, place the camera in every place because of the use of Wireless technology, Password lockout, can be added as cameras as you want in the circuit anytime.

On the other hand, there are many disadvantages with these kinds of cameras:

Higher initial cost per camera and less choice of manufacturers.

Lack of standards. Camera data should be deployed with a standard IP video recording solution to improve the compatibility of the software. But in this project the video
recording solution is the Java program, so it is not very important for the aim of the project.

High network bandwidth requirements.

Technical barrier: This kind of IP cameras has a complicated system of settings, IP address, DDNS, etc. This means that is needed a specialist to install it, but in the project will be configured automatically by DHCP, so it is not relevant for the project.

Lower dynamic range: i.e. reduced ability to cope with low light and high contrast scenes.

The main requirement for the camera is that it must be possible to move it and zoom the image; cameras covering this requirement are known as PTZ (Pan, Tilt, and Zoom) cameras.

Other important requirement include that the camera should have an API (Application Programming Interface) that will enable our Java application to communicate with the camera. An API is an interface implemented by a software program to enable interaction with other software, similar to the way a user interface facilitates interaction between humans and computers. APIs are implemented by applications, libraries and operating systems to determine the vocabulary and calling conventions the programmer should employ to use their services. It may include specifications for routines, data structures, object classes, and protocols used to communicate between the consumer and implementer of the API.

Most cameras on the market do not offer an API, but vendors offer their software solutions to interact with the camera.

However, Vivotek Cameras do offer an HTTP API that allows control of the camera through HTTP request; therefore, VIVOTEK is the camera vendor of the project. Moreover, in the webpage of the vendor there is a live demo of one PTZ camera. The live demo uses the HTTP API to show the image and offers a set of controls to manage the camera (using HTTP requests too). Thus the address of the live demo camera was used to embed the image from the live demo in our Java program as a testing
mechanism before buying the camera. The tests worked properly so the *Vivotek PZ7111/PZ7121 network camera* is the correct choice.

### 5.2. Camera Java Program

To make a program to control and show the video of the camera will be done in Java. The configuration of the program has three parts:

- **IP**: It is saved in ClientConfig.xml, config.xml is read with ClientConfig.xml.
- **User**: Is request to the user with a prompt.
- **Password**: Is request to the user with a prompt.

The webcam package is divided in different classes to have, in a good order, all the components of it. This division is:

- **CamController**: This class works like an interface between the real camera and our graphical user interface (CamView).
- **CamModel**: This class establishes the communication with the camera, represents the camera in our program.
- **CamView**: This class defines a JFrame with the necessary buttons to control a webcam. It's composed by three main Panels, Upper, Middle and Lower. Next picture shows this distribution.
Figure 16. Applet Distribution

- **CamViewInterface:** This is a Java Interface implemented by CamView. It defines several constants used along the webcam packet.

- **ImageApplet:** This class contains an applet where the image will be showed. It also has the methods needed to connect to the camera and read the MJPG stream, i.e. the image stream.

- **WebCamTab:** This class implements a main window in which there are included the CamView frame and the ImageApplet, run it in only one executable and has the tabs to change between the chat-room and the robot live vision.

Next image represents a schematic distribution of the classes used to do the camera software, it is possible to see how the different classes are connected between them and where are defined the different methods and variables used in the program.
Figure 17. Camera program Class Structure
TakeSnapshot is a servlet, and as part of the server it takes the configuration of the webcam (IP, user and password) from serverconfig.xml. TakeSnapshot is a class that executes the web server with the URL of the server/snapshot specified in web.xml which is where the servlet is configured.

The icons are in the program, there is a directory where the images are saved, so when one of them is used as an icon for a button, it has to call it to the direction where they were saved. All these icons are images saved as .gif with the same size (16x16 pixels) and the size of the button is adjusted to that size with a java function. Appendix X shows all the commented codes of the different classes.

The beginning of the program was ImageApplet. This class is able to see the image of the camera which the vendor uses as a demo in the webpage; the code opens up a window that implements the URL where the video is shown. It has to use a function in Java to log in also, so when the URL is called the log in is automatic and shows directly the video. Another interesting thing to comment about is that the camera sends a MJPEG stream, however our application uses the Java JPEGImageDecoder to create the visible image, so the first 4 lines and the last one of the table will be discarded. Therefore, it has to discard these lines in the following table, after which, the JPEG format is kept and refreshed to show it as a video.
For the GUI design, it is important to have taken into account that the controls have to take the smallest area as possible because the panels will be (video and control) placed in a tab with the Chat-room, so a good design is very important.

The figure 18 shows how the camera controls are distributed, each horizontal row is a panel and the arrow buttons are used to control the camera movement. The house button
is to put it in a default position and in the corners there are the snapshot, On/Off, and the zooming of the camera.

![Camera control panel](image)

*Figure 19. Camera control panel*

When writing the HTTP direction of the camera in ImageApplet, it is possible to see the video that the camera is sending. It is necessary to connect the panel to move the real camera, so CamController is written for that purpose. This class, using CamViewInterface as well, recognises which button is pressed by a simple if-else structure and show as a text which one, letting us see if the program really knows which one was pressed, then call to CamModel, where the HTTP has been written with the direction of the command pressed, sending it to the camera. Two programs are working separately, one for the image only, with its own main, and one for the GUI, also with its own main. Then it is necessary to implement a new class to put these two programs into one and insert the tablets and the final appearance of the live camera part.

TabsPanel class is defined for that, in which there are two tabs: One for the Chat-Room where it is going to implement the same chat that is already created in Wise-ShopFloor, and one for the camera, which is called webcamtab class, this is a panel divided in two parts using border-layout (if there are any empty parts, the centre one occupies it), the south one with the GUI and the centre one with the video. So it only implements the other classes in that one removing the two mains in ImageApplet and CamView and implementing it in TabsPanel. So, now executing TabsPanel is running the complete camera’s program with the control as well. To implement it in the Wise-ShopFloor program, a new class is created in Wise-ShopFloor where the whole camera program that was made separately will be copied. In J3DView, a panel model extra tool is used
to insert our tab in the window of Wise-ShopFloor and insert the chat in the empty space that was created for that purpose. Now the size of the image has to be adjusted to the size of the panel, which is smaller than the default image size. A log in and password are used to access the camera, separately of Wise-ShopFloor, so the first time the on/off button is pushed, a login is required. Before this, however, any of the buttons are available to push it.

When the snapshot button is pressed, TakeSnapshot opens a new browser where the picture will be viewed when the button is pressed.

Security problems

Applet is using security manager, so the programmer is not allowed to choose default authentic settings. There is no option to use the authentication Java method, instead it uses a base authentication request.

As the picture shows, the camera has its own server, but applets security does not allow that server to read and write files in the client, but it may be allowed to read and write files if the applet is loaded from the local file system using a URL of type “file”. The only solution is to read and write files on applet’s home webserver. This is the reason that this solution cannot be used.

The solution is to use servlets proxy, these will send the request to the servlet and this one send it to the camera, the same servlets are use to receive the response of the camera. There are three servlets proxy used, one is for videostream and the other two for commands to move the camera. The following figure 20 shows this distribution.
For the snapshot there are is a servlet, which is not proxy, which communicates with the servlet proxy; this one communicates at the same time with the camera instead of doing it from the server.

For a future solution in the security area, when the user logs in to the system, this will check it on the server instead of doing it on the client; thus the client only has to log in one time, at the beginning when it is opened Wise-shop floor.
6. Robot

In this chapter the characteristics of the new robot ABB IRB 140 included in the Wise-ShopFloor application are described and how the rapid programs of the controller are implemented.

6.1. Robot components

An industrial robot contains different electrical and mechanical devices which act together in a system. The controller contains its own operating system with specific software that dictates how the robot operates, moves and communicates. The robot, an essential part of the project, consists of these parts [2]:

1. Manipulator.
2. Controller.
3. FlexPendant.
4. Terminal element.

6.1.1. Manipulator

The manipulator is a mechanical structure of a group of rigid bodies actuating like links, connected by articulations or joints, see figure 21. This is the most visible part of the robot, since it determines the movements and, more generally, contains all the logic related to the robot controller. More important parts include the arm that ensures the mobility, the wrist that bestows dexterity and the end-effector or tool that performs the specific task of the robot. [7]
It is a mechanical arm of ABB, IRB model 140. The mechanical arm of 6 degrees of freedom has been on the market since 1999. There are several versions of this model, which, depending on the task to be undertaken, are optimized for more speed and are designed with tougher conditions, etc. In our case, we have the basic version with some additional options.

The IRB 140 is a mechanical arm that is “powerful and compact, with a unique combination of fast acceleration, large working area and high load capacity” [9]. The data seems to support those words, because the mechanical arm is around the 0.03 mm repeatability and it can withstand a load of up to 5 kg in a scope of 810 mm. [9]

### 6.1.2. Controller

The controller is the brain of the robot and is responsible for calculating the movements and transmitting the engine’s mechanical arm. In our case, the controller is an ABB, IRC5 model, it is showed in figure 22.
It consists of a traditional computer adapted to the needs of its function. Inside this computer runs an operating system designed by ABB called RobotWare. The IRC5 is well designed to support busses for I/O connections. Some powerful networking features are sensor interface functionality, remote disk access and socket messaging. [10]

### 6.1.3. FlexPendant

The FlexPendant is a device that incorporates IRC5 ABB controller. The device is used as a front end for the operator of the robot, and allows you to solve any problems which could occur in the normal operation of the robot, see figure 23. “The Flex Pendant is characterized by its clean and coloured touch screen-based design and 3D joystick for intuitive interaction”. [11]
In our project this device has been helpful and interesting because it is an extension of the controller that has all the commands to use it, with the benefits that an easy interface entails. We have used it to run the different tasks and set the configurations of the robot, programs and connections.

6.1.4. Terminal element

As a terminal arm, the robot has a pneumatic clamp mainly used to handle little objects. This pneumatic gripper is mounted on the plate that makes up the last joint of the robot. It receives the necessary air pressure and the electrical signals needed for its operation through the structure of the robot. For its actions, it has placed a solenoid which, through digital outputs of the controller, opens or closes the gripper.

6.2. Connecting the robot

This part of the chapter, how to connect the robot controller to the network, is explained technically in annex I.

6.3. Rapid

Rapid is a high-level programming language used to control ABB robots. It was introduced along with S4 Control System in 1994, superseding the ARLA programming language used by ABB. Rapid is the language that interprets the robot controller to
perform the actions of the mechanic arm and various computing functions. So it is an imperative language focused on the performance of movements in the robot and general control of the robot controller. [12]

Basic features of the language [12]:

- Routine parameters:
  - Procedures: These are used as subprograms of the main one.
  - Functions: These parameters return a value of a specific type. These are used like an argument of an instruction.
  - Trap routines: These are used to respond to interrupts.

- Logical and arithmetic expressions.
- Automatic error handling.
- Modular programs.
- Multitasking.

As the robot controller has its own operating system as well as the FlexPendant robot system, it was impossible to have absolute control of the robot from a machine outside the system. There were two possible software tools to solve the problem. On one hand, to include in the operating system of the robot optional software, named Socket Messaging, which allowed the opening in the band of the controller and the socket communication. This makes it possible to interact with an external PC, but in a very rudimentary way. [13]

On the other hand, another software, Webware SDK, allows communication between a PC and the operating system, RobotWare, of the controller but at a higher level than desired. It does not allow the execution of instructions directly and is, therefore, a monitoring tool rather than control. [14]

It is selected the first option because it does not need extra software and Sockets Messaging was enough to send and receive the data to monitor and control the robot.
The controller implements two RAPID programs concurrently using the option of RobotWare Multitasking, which will allow it to send data to the robot and receive data from it to control and monitor the movements. The two programs open a socket as a server and once connected, the client (in this case, the robot server written in Java), hears the requests it receives, and in terms of these running as requested. One program is intended to send the state data; this one is executed on the controller as a secondary task. The other task that is running, the main one, is the motion task and it is responsible for carrying out the movements of the robot when asked.

### 6.3.1. Sockets

A socket is a small piece of software that allows a program to communicate with another one by exchanging any data in a reliable way; either on the same computer or two different ones. Most communication protocols are based on socket connections. A socket is defined by an IP address, a protocol and a port. To have communication between two programs, they must know the data above each other. Subsequently, they need to be able to send information so the communication channel must be available. This communication is done using the TCP protocol with the option offered by the software IRC5 Socket Messaging. [15]

### 6.3.2. RobotWare

The RobotWare is the operating system that runs on the controller. It is a system developed by ABB for their drivers IRC5. The system is built around a core present in all distributions, on which different options can be installed (this is the word used by ABB that we could understand as an extension of functionality). The core contains the main logic for the execution of movements and other basic functionality, while additional options allow the extension of some functionality. There are options of all kinds depending of the different types of work that could be performance by the robot, like welding, painting, packaging.... In the next points are defined the main additional options used for this project [16].
6.3.2.1. Socket Messaging

Socket Messaging is a RobotWare option that permits the communication between the robot controller and a PC connected via Ethernet-based sockets. The purpose of the Socket Messaging option is to allow a program to exchange messages RAPID TCP/IP over a network. This utility sends and receives messages via the Ethernet channel permanent (which can be used simultaneously to other network traffic, e.g. for communication with the software RobotStudio Online).

Socket Messaging is a standard supported by operating systems such as UNIX or Microsoft Windows. One such communication can be divided into three phases [17]:

- Creating and closing the communication channel.
- Establishment of the communication session.
- Sending and receiving data.

More specifically, it allows open sockets on the controller which may be connected to other sockets of a remote PC or any device that has this capability, see figure 24. In this way, data can be exchanged between the controller and an external PC but in a very routine way.
6.3.2.2. Multitasking

The robot controller only allows running a concurrent program on the system, called a task in ABB terminology. There is, however, the additional option of Multitasking that allows the emulation of a concurrent system. This option allows the parallel execution of up to 20 tasks; but the manual states that it is pseudo parallelism. These tasks consist of a set of local modules. Two programs running in parallel have the possibility to be placed as background or foreground of another program. [18]

To use this function it is necessary to configure the robot with an extra task for each additional program. The tasks can be of three types: normal, static, or semistatic; for the motion task we need the normal option.

In each task constants, variables, and persistent are defined locally, but not global persistent. A persistent is global if it is not declared as local in the program. A global persistent can be used in all the tasks that it is declared if it has same name and type. The move instructions can only be used in the motion task. [18] [19] [20]
6.3.2.3. PC Interface

This is option of RobotWare that provides the communication interface between the robot and a remote PC. With this option it is possible to connect the controller of the robot to a local network to have access from a computer or directly to internet to have remote access to it. How to connect the controller to a local network or to internet is explained in the appendix I. It is used for [21]:

- Software integration using an OPC Server interface for a SCADA system, useful to monitor and control some functions of the process developed by the robot.
- Connect the robot with RobotStudio Online to interact with the controller through an Ethernet connection.
- Communicate with other ABB software, for example to use the multimove option.

6.3.3. RobotStudio Online

RobotStudio is a computer software that emulates a virtual controller of the ABB robot; it is the same software used by robots in the production. Robot Studio Online is made to simplify the installation and configuration of the controller software RobotWare used in the IRC5. [22] [23]

To create and develop programs, it is possible to use the ABB rapid robot language using the Teach Pendant or to use the Robot Studio planning and simulation software. Using this software, the programs are downloaded from the PC to the controller. “Events may be monitored and recorded, I/O signals added and configured and when completed all backed up on the PC” [23]. Then it is necessary to fine tune and test the programs. [22] [23]
Running the robot controller in automatic mode and using RobotStudio Online on the computer, the FlexPendant is almost not needed because the PC is transformed in a virtual teach pendant (see figure 25) with almost all the functionality of the real one [23].

6.4. Robot programs

There are two main rapid tasks running with two main programs to develop the monitoring and controlling of the robot. To obtain the status information of the robot for the monitoring task and then transmit it to the server, a specific rapid instruction is used, called CjointT. It is also possible to use another instruction, CRobT. Each returns a position of the robot, but in different ways. CJointT represents the position by the angle at which each joint is, see figure 26, while CRobT does it by coordinates X, Y, Z of TCP.

With a Rapid module with a cyclic timer we can log the positions during the program movement and save the data into a file each 0,033 seconds. It looks like this:

<table>
<thead>
<tr>
<th>Axis 1</th>
<th>Axis 2</th>
<th>Axis 3</th>
<th>Axis 4</th>
<th>Axis 5</th>
<th>Axis 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>74,9921</td>
<td>23,8943</td>
<td>22,606</td>
<td>43,7447</td>
<td>77,3445</td>
<td>44,4095</td>
</tr>
</tbody>
</table>
To use the function CJointT to read the current robot joint angles it is necessary to follow the next steps:
- Define the jointtarget data “joints” as VAR jointtarget joints;
- Read the current robot joints into “joints” data with the instruction joints:=CJointT( )
- Send the data to the server by sockets for the 3D simulation.

Here it is explained the monitoring program, the full programs are in the annex III.
Rapid Programming. This rapid program, implemented in the task of the robot TROB_2, executes the monitoring process of the robot. It has two procedures, the main one and the monitoring one. For this task, the sockets and rawbyte variables, the movement data type jointtarget and the numerical variables for the degrees of the six joints, are needed.

The program starts creating a socket, binding it to an IP address and a port number and then starts listening to incoming connections. After that, the program pointer stays in a “while” instruction accepting incoming connections, doing the monitoring procedure and closing the socket.
The monitoring procedure reads the current position of the joints with the function CJointT() and saves these in the variables of each one of the six joints. These variables are packed in a rawdata packet with the instruction PackRawBytes, together with a header and the size of the packet. Afterwards, this packet with the data of the robot is sent to the remote computer with the instruction SocketSend SocketOut\RawData. This procedure is repeated every 0.033 seconds for the monitoring with the java 3D model. In this procedure there is also an error handler that controls the ERR_SOCK_CLOSED error, that otherwise would stop the communications with the computer and would stop the rapid program. See annex III. Rapid Programming, monitoring program.

To control the robot movements, basically we proceed in a similar way as before but by changing the degrees of the variable of the joints that we want to move:

- Define the jointtarget data “angles” as VAR Jointtarget angles.

- While the program is not receiving any data from the computer the robot is stopped.

- Receive an array (rawbytes data type) from the client with the instructions to move the robot.

- Read the current robot joints into angles data with the instruction joints: =CJointT ()

- Depending of the header of the data received from the client side, it would be possible to modify the joints values by the client request or to load a program like a rapid module from the memory of the controller with specified commands and movements for the robot.

- Move to the updated position by the instruction MoveAbsJ\Conc, angles, v2000, fine, tool0;

Now this is explained carefully. The controlling program is implemented in the motion task of the robot T_ROB1. In this one the sockets are managed in the main procedure in the same way as in the monitoring program, using SocketCreate, SocketBind and SocketListen.
In a “while” instruction that’s always running in the main procedure, three commands are used, SocketAccept, go to controlling procedure to receive the data and move the robot and later socketclose. After this “while” it is executed an UNDO instruction that closes the communication socket when the program finishes. It is executed to be able to restart the communication when the rapid program starts again. There is a time out in the SocketAccept of 0.12 seconds, that throws an error, ERR_SOCK_TIMEOUT, while the program is not receiving any data from the computer. When this occurs the robot is stopped, the path of the movement is cleared, the procedure ResetLastJointMovementCommand is called and after that, the stop of the robot is resetted with the command Startmove.

That value of the time out, 0.12 seconds, was chosen because increasing it, the movements of the robot were smoother but the minimal amount of movement was increased; if this time out was decreased, the response of the robot was faster (doing smaller movements) but the movements were done with some steps.

The function of the procedure ResetLastJointMovementCommand is to change the values of the direction and last joint moved. When a packet is received the program continues. That error and the error ERR_SOCK_CLOSED are handled with the rapid handler in this controlling procedure.

The program starts setting the velocity of the TCP (Tool Centre Point) at 2000 mm/s and the velocity of reorientation of the TCP at 0.01 degrees/s with the command VAR speeddata vmedium := [ 2000, 0.01, 0, 0 ]. This is done in this way because when the joint six is turning, the TCP is displaced only a bit but the wrist that moves it is moved in a very fast way compared with the other joints, so we set the velocity of reorientation of TCP.

At the beginning of the main module we define a cylinder around the bed and the base of the robot where the TCP cannot go inside, to protect the robot of damaging himself, but this part of the code is not possible to run because the controller of the robot doesn’t have the option WordZones needed to use the instruction WZCylDef.

The instruction MotionSup is used to activate the motion supervision that provides the collision detection function. It is a rapid instruction to avoid the possible damage of the robot and its surrounding devices. It prevents the robot from damaging himself: when it
detects a force against the movement it goes back a bit and stops the movement immediately, it is even possible to stop the robot with the hand. This instruction is explained in annex III. Rapid programming.

The controlling procedure starts calling the procedure `ResetLastJointMovementCommand` that changes the values of the direction and the last joint moved. After that, a “while” instruction is used that receives the raw data packet with the data from the computer using the instruction `SocketReceive`. If a packet has been received, the first value is unpacked, which is the header, using the instruction `UnpackRawBytes`. This header - depending on its value - is implemented in order to have the opportunity in a future to control the robot in two different ways. The first one is controlling the robot changing the degrees of the joints joint by joint, so the procedure `MoveSingleJoint` is called passing to it the packet received from the PC like an argument. The second value that the header could have, is to order to load a module from the memory of the controller and run its predefined commands and movements.

After that process, the number of the joint to move and the desired direction are unpacked; the current degrees of the joints are read and saved in the variable `angles` using the function `CJointT()`. If the joint or the direction is different from the last movement, the joint ordered to move is changed by the function `setToLimit`.

The function `setToLimit` saves in the variable `anglesValue` the value of the limit of the range of the specified joint and direction. Afterwards, the procedure `StoreJointMovementCommand` saves the current number of joint and direction in the variables `lastJoint` and `lastDirection` and after that the procedure `lastJoint` is called from `MoveSingleJoint` that stores the changed value of the desired joint (`anglesValue`) in the joint target movement command “angles”.

The last procedure is “move” which receives the joint target, clears the path of the last movement and moves the robot to the ordered position using the instruction `MoveAbsJ` explained later. To see the rapid program and the specifications of the functions, data types and instructions see annex III. Rapid Programming.
6.4.1. Errors

To control the possible errors of the robot controller, like communication closed (error socket closed) or time out of the socket received (error socket timeout), that stopped the rapid program, we have to handle these using the rapid error handle. The events of “joint out or range” or “joint in wrong direction” were impossible to handle like the other errors. This is because these errors of the movement of joints are events instead errors. To fix this problem the movement that is ordered to each joint is to reach a few degrees less its limit in the positive or negative direction, and just before the joint reaches its limit, the movement is stopped, reset the path of the movement and started again if it is going in the correct direction.

6.4.2. World Zones

In a controlled environment such as ours, which provides for the use of robot for users with only a bit of prior training, this option can be very useful. This is because without experience of the user, the robot can become physically damaged. This can also be made consciously by malicious users.

Therefore, detection of collisions protects nearby objects as well as the robot of itself. In our environment, we have created a security zone to protect the bed on which the mechanical arm is mounted as well as the base of the robot. It could also be used to protect surrounding equipment. For this purpose, the function WZCylDef is used, which requires the RobotWare option World Zones. It defines a cylinder (or other shape) surrounding the bed and the lower part of the robot. Here is the rapid code that defines this collisions detector:

```
VAR shapedata volume;
CONST pos C2:= [0, 0, and 0];
CONST num R2:=300;
CONST num H2:=620;
WZCylDef\Inside, volume, C2, R2, H2;
```
There is another rapid instruction to avoid this possible damage of the robot and his surrounding devices: MotionSup (motion supervision), which takes care of the movements of the robot. It prevents the robot from damaging itself; when it feels a force against the movement, it goes back a bit and stops the movement immediately. It is even possible to stop the robot with the hand.

### 6.5. Other possible solution

Another solution to solve the communication problems of controlling and monitoring the robot through internet would be Webware SDK. It is commercial software distributed by ABB that allows connection between a PC and a controller of ABB. This is software to monitor drivers of robots that could have been in a production plant. With this software, applications are created to monitor and control some functions.

This SDK is made up of a set of ActiveX controls to be used only from platforms such as Microsoft Visual Basic or Visual C++. These controls communicate with what is called the Interlink, which is a small program that connects to the controller. In this way, you get access to the controller, and can obtain data of the robot, for example, the state of the joints or the state of the inputs as well as digital and analog outputs. You can also perform actions on the controller, such as start / stop the program or other work more oriented towards direct supervisory control.

The RobotStudio API (Application Program Interface) doesn't contain functionality for communicating directly with the controller. For this purpose, the PC SDK should be used, which is part of the Robot Application Builder product. With it you can upload programs, retrieve the program pointer and much more. It is possible to combine PC-SDK and RobotStudio API. [13] [25]

### 7. Future improvements

There are some possible improvements to implement in Wise Shop Floor.
To create a chat window in each webpage to be able to have communication between all the users of the system during controlling and monitoring modes.

For the robot control, it would be possible to add the option of controlling the robot using the coordinates of the TCP of the tool instead of using joint by joint movement. Also to modify the speed of the movements of the robot during the controlling mode.

When there is a collision detection in the real robot, the rapid programs are stopped. A task would be to move the program pointers to main and start to run the programs remotely.

It is possible to load a rapid module or program with specified instructions from the memory of the controller; so it would be possible to send a rapid module from the Wise Shop Floor application to the server and execute it in the controller.

In the camera field, to be able to change the size of the screen of the camera in the browser.

8. Conclusion

In the previous versions of the Wise ShopFloor, the goal was established as a Web based mechanism where multiple clients could monitor and control one machine, specifically, the milling machine. In this project the goal was introduce a new machine inside the application the model IRB140 and integrated a camera web inside the application.

When the project was handed to us, the software worked, but the source code used was barely understandable. It contained several bugs and the design of some parts was poor. In this project, some of those aspects have been improved. The way in which the virtual models are moved is now more general. The server has been entirely redesign and now it allows controlling several machines by several clients and, for the first time, the monitoring clients limit actually exists. Besides all this, some of the bugs have been solved, all new classes were well documented and there has been attempted to improve the old documentation.
Since Wise ShopFloor is an alternative for the camera based systems, the goal of this project is demonstrating the Real-Time or at least the soft Real-Time of the virtual model. It is a fact that the amount of data necessary to send an image is much bigger than the amount of data that it is necessary to send the information of the robot’s joints.
9. References


[18] Operating manual. IRC5 with FlexPendant.


[27] Operating manual IRC5 with FlexPendant.


10. Appendix

Annex 1. Connecting the robot

To connect the robot controller to the network, it is necessary that some options be installed in the controller, like PC Interface and Ethernet Services/FTP. [27]

It is necessary to X-Start the robot, then from the Boot Application menu, select Settings and then choose to use DHCP to have the robot acquire an IP address automatically from a DHCP Server on the network, or choose to assign a static IP address. In this project was used a static IP number. It was necessary to work with the network administrator to determine what an appropriate static IP address and which subnet and gateway settings should be. Once the settings are entered, press OK twice.
and press Select System to pick your system. Then press OK and Restart Controller. [27]

It is important to connect the Ethernet cable to the LAN port (B), see figure 12, and not to the service port (A) for the connection with the network. The service port also acts as a DHCP server, so if the cable is connected to the corporate LAN, it may start serving up IP addresses to other computers on the same network.

![Figure 29. Two main communication ports on the controller Computer Unit](image)

It is preferred to have the robot on the same subnet because some applications do not automatically search and connect to robot across other subnets. For example, if the PC is addressed as 192.168.100.x and the robot is addressed 192.168.150.x, the software will not locate the robot when it is browsed for the first time, or if the PC is restarted, communications will not be automatically re-established upon start-up. [29]
Annex II. Socket messaging

A socket is a small piece of software which allows a program communicates with another one exchanging any data in a reliable way, either on the same computer or two different ones. Most communication protocols explained later are based on sockets connections, i.e. using sockets for their transactions, on developing new functionality more complex. These are the functions that we have used in the rapid program for controlling and monitoring the robot. [15] [17] [19]

-SocketAccept is an instruction to accept incoming connection requests. SocketAccept can only be used for server applications. [15]

Example

VAR socketdev server_socket;
VAR socketdev client_socket;
...
SocketCreate server_socket;
SocketBind server_socket,"192.125.100.1", 1025;
SocketListen server_socket;
SocketAccept server_socket, client_socket;

It is created a server socket and bound to the port 1025 on the network address 192.125.100.1. After the execution of SocketListen the server socket starts to listen for incoming connections on that port and address. SocketAccept waits for incoming connections, accepts the connection request, and returns a client socket for the established connection. [15]

Program execution: The server socket will wait for any incoming connection requests. When accepting the incoming connection request the instruction is ready and the returned client socket is by default connected and can be used in SocketSend and SocketReceive instructions. [15]

-SocketCreate is used to create a new socket for connection based communication. [15]
The socket messaging is of stream type protocol TCP/IP with delivery guarantee. Both server and client application can be developed. Datagram protocol UDP/IP with broadcast is not supported. The instruction creates a new socket device.

The socket must not already be in use and it is in use between SocketCreate and SocketClose. [15]

- **SocketConnect** is used to connect the socket to a remote computer in a client application. [15]

The socket tries to connect to the remote computer on the specified address and port. The program execution will wait until the connection is established, failed, or a timeout occurs. [15]

- **SocketSend** is used to send data to a remote computer. SocketSend can be used both for client and server applications. [15]

Arguments [15]:

SocketSend Socket [ \Str ] | [ \RawData ] | [ \Data ] [ \NoOfBytes ]

Socket: Data type: socketdev.

In client application the socket to send from must already be created and connected.

In server application the socket to send to must already be accepted.

[ \Str ]: Data type: string.

It is the string to send to the remote computer.

[ \RawData ]: Data type: rawbytes

The rawbytes data to send to the remote computer.

[ \Data ]: Data type: array of byte

The data in the byte array to send to the remote computer. Only one of the option parameters \Str, \RawData, or \Data can be used at the same time.
\[\text{NoOfBytes}\]: Data type: num

If this argument is specified only this number of bytes will be sent to the remote computer. The call to SocketSend will fail if \text{NoOfBytes} is larger than the actual number of bytes in the data structure to send. If this argument is not specified then the whole data structure (valid part of rawbytes) will be sent to the remote computer. The size of the data to send is limited to 1024 bytes.

\textbf{-SocketReceive} is used for receiving data from a remote computer. SocketReceive can be used both for client and server applications.

Arguments [15]:

\begin{align*}
\text{SocketReceive} & \quad \text{Socket} \quad [\text{Str}] \quad | \quad [\text{RawData}] \quad | \quad [\text{Data}] \quad [\text{ReadNoOfBytes}] \quad [\text{NoRecBytes}] \quad [\text{Time}] \\
\text{Socket:} & \quad \text{Data type: socketdev} \\
\text{In a client application where the socket receives the data, the socket must already be created and connected. In a server application where the socket receives the data, the socket must already be accepted.} \\
\text{[Str]:} & \quad \text{Data type: string} \\
\text{The variable in which the received string data should be stored. Max. number of characters 80 can be handled.} \\
\text{[RawData]:} & \quad \text{Data type: rawbytes} \\
\text{The variable in which the received rawbytes data should be stored. Max. number of rawbytes 1024 can be handled.} \\
\text{[Data]:} & \quad \text{Data type: array of byte} \\
\text{The variable in which the received byte data should be stored. Max. number of byte 1024 can be handled. Only one of the optional parameters \text{Str}, \text{RawData}, and \text{Data} can be used at the same time.}
\end{align*}
[ `ReadNoOfBytes` ]: Data type: num

The number of bytes to read. The minimum value of bytes to read is 1, and the maximum amount is the value of the size of the data type used, i.e. 80 bytes if using a variable of the data type string.

If communicating with a client that always sends a fixed number of bytes, this optional parameter can be used to specify that the same amount of bytes should be read for each SocketReceive instruction.

If the sender sends RawData, the receiver needs to specify that 4 bytes should be received for each rawbytes sent.

**-SocketClose** is used when a socket connection is no longer going to be used.

After a socket has been closed it cannot be used in any socket call except SocketCreate.

The socket will be closed and its allocated resources will be released. Any socket can be closed at any time. The socket can not be used after closing. However it can be reused for a new connection after a call to SocketCreate. [15]

**-SocketBind** is used to bind a socket to the specified server IP-address and port number. SocketBind can only be used for server applications.

The server socket is bound to the specified server port and IP-address.

An error is generated if the specified port is already in use. [15]

**-SocketListen** is used to start listening for incoming connections, i.e. start acting as a server. SocketListen can only used for server applications.

The server socket starts listening for incoming connections. When the instruction is ready the socket is ready to accept an incoming connection. [15]
Annex III. Rapid programming

This contains the two rapid programs used in the different tasks of the controller. One is for the monitoring process of the robot and the other one is for the motion task to receive data from the client. This annex also includes the basic instructions, functions and data types used in the programs.

- Monitoring program:

```plaintext
MODULE Monitor
Var socketdev ServerSocket;
Var socketdev SocketOut;
Var jointtarget joints;
Var num header:=1;
Var num joint1:=0;
Var num joint2:=0;
Var num joint3:=0;
Var num joint4:=0;
Var num joint5:=0;
Var num joint6:=0;
Var rawbytes anglesData;
Var num size:=24;

PROC main()
    SocketCreate ServerSocket;
    SocketBind ServerSocket, "192.168.1.250", 2048;
    SocketListen ServerSocket;
    WHILE TRUE DO
        SocketAccept ServerSocket, SocketOut\Time:=WAIT_MAX;
        Monitoring SocketOut;
        socketclose SocketOut;
    ENDWHILE
ENDPROC

PROC Monitoring(Var socketdev SocketOut)
    WHILE TRUE DO
        joints := CJointT();
        joint1:=joints.robax.rax_1;
        joint2:=joints.robax.rax_2;
        joint3:=joints.robax.rax_3;
        joint4:=joints.robax.rax_4;
        joint5:=joints.robax.rax_5;
        joint6:=joints.robax.rax_6;
```

This is the program to control the robot. It is implemented in the motion task of the robot T_ROB1. In this one the sockets are managed in the main procedure in the same way as in the monitoring program, using SocketCreate, SocketBind and SocketListen.

- **Controlling program:**

```plaintext

MODULE Control
    VAR socketdev ServerSocket;
    VAR socketdev SocketIn;
    VAR rawbytes packet;
    VAR num header;
    VAR num lastJoint;
    VAR num lastDirection;
    VAR bool packetReceived;
    VAR speeddata vmedium := [ 2000, 0.01, 0, 0 ];
    PROC main()
        //It is necessary the WorldZone option.
        !VAR shapedata volume;
        !CONST pos C2 := [0.0, 0.0];
        !CONST num R2 := 300;
```
!CONST num H2:=620;
!WZCylDef \Inside, volume, C2, R2, H2;

SocketCreate ServerSocket;
SocketBind ServerSocket, "192.168.1.250", 2050;
SocketListen ServerSocket;
VelSet 2, 2000;

MotionSup \On \Tunevalue:=50;

WHILE TRUE DO
  SocketAccept ServerSocket, SocketIn\Time:=WAIT_MAX;
  Controlling SocketIn;
  socketclose SocketIn;
ENDWHILE
ENDPROC

PROC Controlling(Var socketdev SocketIn)
  ResetLastJointMovementCommand;

  WHILE TRUE DO
    packetReceived:=TRUE;
    SocketReceive SocketIn\RawData:=packet\Time:=0.12;
    IF packetReceived=TRUE THEN
      UnpackRawBytes packet, 1, header\IntX:=DINT;
      IF header=3 THEN MoveSingleJoint packet;
      ELSEIF header=2 THEN Moveangles packet;
      ENDIF
    ENDIF
  ENDWHILE

ERROR
IF ERR_SOCK_CLOSED = ERRNO THEN
  StopMove \Quick;
  ClearPath;
  RETURN;
ELSEIF ERR_SOCK_TIMEOUT = ERRNO THEN
  StopMove \Quick;
  ClearPath;
  StartMove;
  ResetLastJointMovementCommand;
  packetReceived:=FALSE;
  TRYNEXT;
ELSE
PROC MoveSingleJoint(VAR rawbytes packet)

VAR num joint:=0;
VAR num direction:=0;
VAR jointtarget angles;
VAR num angleValue:=0;

UnpackRawBytes packet, 5, joint\IntX:=DINT;
UnpackRawBytes packet, 9, direction\IntX:=DINT;

angles:=CJointT();

IF joint=lastJoint AND direction=lastDirection  THEN
   RETURN;
ENDIF

angleValue := setToLimit(joint, direction);

StoreJointMovementCommand joint, direction;

changeAngle angles, joint, angleValue;

move angles;

ENDPROC

PROC ResetLastJointMovementCommand()
   StoreJointMovementCommand -1, -2000;
ENDPROC

PROC StoreJointMovementCommand(num jointNumber, num direction)
   lastJoint:=jointNumber;
   lastDirection:=direction;
ENDPROC

FUNC num setToLimit (num joint, num direction)

VAR num res := 0;
IF direction=2 THEN
   IF joint=1 THEN res := 175;
   ELSEIF joint=2 THEN res := 105;
   ELSEIF joint=3 THEN res := 45;
   ELSEIF joint=4 THEN res := 195;
   ELSEIF joint=5 THEN res := 110;
   ELSEIF joint=6 THEN res := 395;
ENDIF
ELSEIF direction=1 THEN
    IF joint=1 THEN res := -175;
    ELSEIF joint=2 THEN res := -85;
    ELSEIF joint=3 THEN res := -225;
    ELSEIF joint=4 THEN res := -195;
    ELSEIF joint=5 THEN res := -110;
    ELSEIF joint=6 THEN res := -395;
ENDIF
ENDIF
RETURN res;
ENDFUNC

PROC changeAngle (INOUT jointtarget angles, num joint, num value)
    IF joint=1 THEN angles.robax.rax_1:=value;
    ELSEIF joint=2 THEN angles.robax.rax_2:=value;
    ELSEIF joint=3 THEN angles.robax.rax_3:=value;
    ELSEIF joint=4 THEN angles.robax.rax_4:=value;
    ELSEIF joint=5 THEN angles.robax.rax_5:=value;
    ELSEIF joint=6 THEN angles.robax.rax_6:=value;
ENDIF
ENDPROC

PROC move (jointtarget angles)
    ClearPath;
    MoveAbsJ.Conc, angles, v2000, fine, tool0;
ENDPROC
ENDMODULE

Functions [15]

- CJointT (Current Joint Target) is used to read the current angles of the robot axes and external axes.

- CRobT (Current Robot Target) is used to read the current position of a robot and external axes.

This function returns a robtarget value with position (x, y, z), orientation (q1 ... q4), robot axes configuration, and external axes position. If only the x, y, and z values of the robot TCP (pos) are to be read then use the function CPos instead.
Instructions [15]

- MoveAbsJ (Move Absolute Joint) is used to move the robot and external axes to an absolute position defined in axes positions.

The robot and external axes move to the destination position along a non-linear path. All axes reach the destination position at the same time.

This instruction can only be used in the main task T_ROB1 or, if in a MultiMove system, in Motion tasks.

The tool is moved to the destination absolute joint position with interpolation of the axis angles. This means that each axis is moved with constant axis velocity and that all axes reach the destination joint position at the same time, which results in a non-linear path.

Generally speaking, the TCP is moved at approximate programmed velocity. The tool is reoriented and the external axes are moved at the same time as the TCP moves. If the programmed velocity for reorientation or for the external axes cannot be attained, the velocity of the TCP will be reduced.

- PackRawBytes is used to pack the contents of variables of type num, dnum, byte, or string in a container of rawbytes. During program execution the data is packed from the variable of type anytype into the rawdata type container.

The length of valid bytes in the RawData variable is set to:

- (StartIndex + packed_number_of_bytes - 1)

- The length of valid bytes in the RawData variable is not changed if the complete packing operation is done inside the old length of valid bytes in the RawData variable.
- StopMove is used to stop robot and external axes movements and any belonging process temporarily. If the instruction StartMove is given then the movement and process resume. The argument ‘Quick stops the robot on the path as fast as possible.

- StartMove is used to resume robot, external axes movement and belonging process after the movement has been stopped.

- ClearPath clears the motion path on the current motion path level (base level or StorePath level).

This is to clear all the movement segments from any move instructions which have been executed in RAPID but not performed by the robot at the execution time of ClearPath. The robot must be in a stop point position or must be stopped with StopMove before the instruction ClearPath can be executed.

- UnpackRawBytes is used to unpack the data of a rawbytes container. It can content the data types byte, num, dnum or string.

- RAISE instruction creates an error in the program and calls the error handler in the routine.

This instruction can, for example, be used to jump back to a higher level in the structure of the program, e.g. to the error handler in the main routine if an error occurs at a lower level.

Program execution continues in the routine’s error handler. After the error handler has been executed in the program can be used:

  - Return: With this instruction the execution of the program continues with the routine that called the routine in question.
  - Raise: The program execution continues with the error handler of the routine that called the routine in question.

- WaitTime waits an amount of time or waits until the robot and external axes have come to a standstill. Program execution temporarily stops for the given amount of time. Interrupt handling and other similar functions, nevertheless, are still active.
Data types [15]

- Robjoint is used to define the position in degrees of the robot axes. Data of the type robjoint is used to store axis positions in degrees of the robot axis 1 to 6.

Axis position is defined as the rotation in degrees for the respective axis (arm) in a positive or negative direction from the axis calibration position.

- Jointtarget is used to define the position that the robot and the external axes will move to with the instruction MoveAbsJ. It is compounded of the following data types:

  - robax :  
    Data type: robjoint

    Axis position is defined as the rotation in degrees for the respective axis (arm) in a positive or negative direction from the axis calibration position.

  - extax:  
    Data type: extjoint

    The position is defined as follows for each individual axis (eax_a, eax_b ... eax_f). For rotating axes, the position is defined as the rotation in degrees from the calibration position. For linear axes, the position is defined as the distance in mm from the calibration position.

    So jointtarget has the following structure:

    < dataobject of jointtarget >
    < robax of robjoint >
    < rax_1 of num >
    < rax_2 of num >
    < rax_3 of num >
    < rax_4 of num >
    < rax_5 of num >
    < rax_6 of num >
    < extax of extjoint >
    < eax_a of num >
    < eax_b of num >
    < eax_c of num >
    < eax_d of num >
    < eax_e of num >
    < eax_f of num >
- Rawbytes is used as a general data container. It can be used for communication with I/O devices. Rawbytes data can be filled with any type of data - num, byte, string - by means of support instructions/functions. In any variable of rawbytes, the system also stores the current length of valid bytes. A rawbytes variable may contain 0 to 1024 bytes.

- Socketdev (socket device) is used to communicate with other computers on a network or between rapid tasks. The socket device is a handle to a communication link to another computer on a network.

- String is used for character strings. A character string consists of a number of characters (a maximum of 80) enclosed by quotation marks.