

Rob 1 Assignment 4, Group 1

Martin Moghadam, Martin Franzen
Kalle Grafström, Audrius Palisaitis, Robertas Jacauskas

January 20, 2010

Abstract

The task is milling letters in styrofoam using a Fanuc robot and RobWorkStudio. Two applications were developed to solve this task. The programs were developed in modules so that i.e. LetterFonts C# program could be substituted for another program to write letters with. It should just follow the Pathplanner plugins specifications that parse the text file from the LetterFonts program and translates the Cartesian coordinates to joint coordinates. Which can be simulated and uploaded to the Fanuc LR Mate 200i. The simulation in RobWorksStudio is performed before any upload to the robot is commenced to protect it from harm itself or the surroundings. When the robot receives an upload it performs the milling of the letters in the Styrofoam.

Contents

1	Introduction	2
2	Specifications of the Robot Arm	2
2.1	Dimensions of the two Models	4
2.2	Construction of the Cage	5
2.3	Construction of Joint Angles	7
2.4	Modeling the Tool Frame	10
2.5	Construction of XML File	12
2.6	Construction of Joints	12
3	Letter Font Application	13
3.1	Letter labeling application	16
3.2	Letter Font Application Extension	19
4	RobworkStudio plugin	19
4.1	Parser	20
4.1.1	Implementation	20
4.2	Path planning and joint angle solving	21

4.2.1 Implementation	21
5 Testing the System	22
6 Conclusion	22
References	22
7 Appendix	23

1 Introduction

This assignment is a part of the Rob1 course. As a starting point this assignment is supposed to give the students experience in developing small applications that make use of the topics learned so far in the course such as making plugins for RobWork, using the RobWork interface and additionally the students should interface the application to a physical robot and use a tool that perform some kind of interaction with the surroundings.

The assignment given to this group of students will make use of a Fanuc LR Mate 200i that is configured to physically carve in styrofoam with a milling tool mounted on the end joint of the robot.

The user of the robot should be able to type a sentence in a GUI with a user selected font and font size. From this GUI a task description that tells the robot the lines it has to follow to mill the letters in the foam, is created. The task description must then be translated into joint coordinates that will be uploaded to the Fanuc robot.

But before the joint coordinates is uploaded to the robot it must be simulated in RobWorkStudio to make sure that it doesn't damage itself, the milling tool or the surroundings. Some thoughts on how the the tool should be moved when it's airborne and when it is docked to the foam and maybe how and where to start and stop the milling tool. Another part of the assignment is to specify the work area where the foam is placed.

2 Specifications of the Robot Arm

It's necessary to verify the Fanuc LR Mate 200i specifications before any work on the RobWork plugins can commence, because the robot in the laboratory is a LR Mate 200iB and in RobWork it's a LR Mate200iC model. It is essential that the LR Mate model in RobWork is true to the real robot in the laboratory. The model in RobWork (LR Mate200iC) is depicted in the figure below left and the model in the laboratory (LR Mate200iB) is depicted below right. As can be seen from the figure, the models in some points are different however in other cases they are similar.



Figure 1: Fanuc Models, left: LR Mate 200iC, right: LR Mate 200iB

Items		LR Mate 200iC (RobWork)	LR Mate 200iB (Laboratory)
Axes		6	6
Payload – wrist (kg)		5	5
Reach (mm)		704	700
Repeatability		+0.02	+0.04
Interference Radius (mm)		181	180
Motion Range (degrees)	J1	340 (360 option)	320
	J2	200	185
	J3	388	316
	J4	380	380
	J5	240	240
	J6	720	720
Motion speed (degrees/s)	J1	350	180
	J2	350	180
	J3	400	225
	J4	450	400
	J5	450	330
	J6	720	480

Figure 2: Specifications of LR Mate 200iB and LR Mate 200iC

Find the similarity and difference points in the table above from which an analyze can be commenced. As can be seen the main point to analyze is the motion range that is a little bit different between the two models. But physically are the two robots similar with respect to physical length of the joint arms, it's important to get a correct physical representation of the robot in RobWorkStudio. Other points from the table are not so important for use, among this is payload, reach or motion speed, because they are more related to efficiency.

2.1 Dimensions of the two Models

By analyzing the datasheet of the two models it can be concluded that the physical appearance of the two Fanuc robots are similar in dimensions. They have six revolute joints and similar joint arm dimensions. Both models are depicted in the figure 1. Then DH-parameters can be determined using the datasheet and physically inspecting the robot in the laboratory to verify the dimensions.

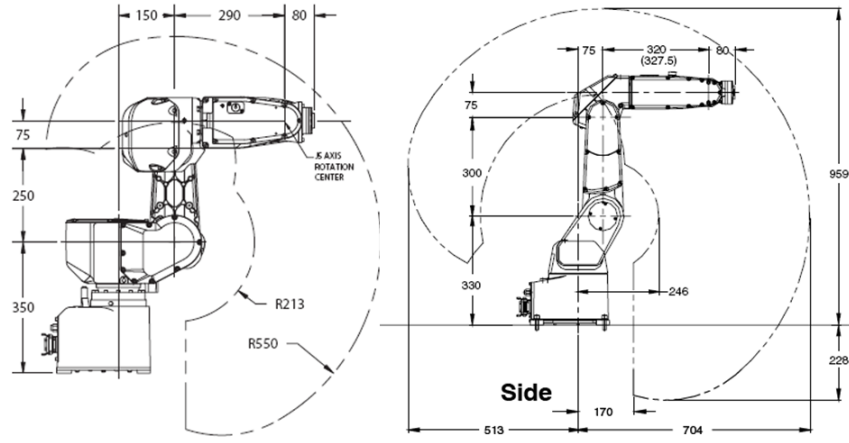


Figure 3: left: LR Mate 200iB (Laboratory), right: LR Mate 200iC (RobWork)

Before determining the DH-parameters, it's important to understand what the meaning of the parameters are, they are defined below:

- a_{i-1} – alpha is angle between Z_i and Z_{i+1} which is measured about axis X_i
- a_{i-1} – here distance in mm from Z_i and Z_{i+1} which is measured about axis X_i
- d_i – distance from X_{i-1} to X_i measured along axis Z_i
- θ_i – theta is angle between X_{i-1} and X_i measured along axis Z_i

To determine the DH-parameters we need to know the dimensions of both the models, they are shown in the figure 3. At the left LR Mate 200iB model which we use in the laboratory and on the right LR Mate 200iC model which we use only to simulate. The determined parameters are shown in the figure 4.

<i>Model</i>	LR Mate 200iB (Laboratory)				LR Mate 200iC (RobWork)			
<i>i</i>	a_{i-1}	a_{i-1}	d_i	θ_i	a_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1	0	0	0	θ_1
2	-90	150	0	θ_2	-90	75	0	θ_2
3	180	250	0	θ_3	180	300	0	θ_3
4	-90	75	290	θ_4	-90	75	320	θ_4
5	90	0	0	θ_5	90	0	0	θ_5
6	90	0	0	θ_6	90	0	0	θ_6

Figure 4: DH parameters of LR Mate 200iB and LR Mate 200iC

From figure4 it can be seen that the real model and the model which will be used for simulations is quite similar, however some links are different. However differences are not huge so a decision was made that the model can be used for simulation. The LR Mate 200iB was constructed by using some frames from and the some 3D model objects of the LR Mate 200iC. The model seen in RobWorkStudio are a mix of LR Mate 200iC and LR Mate 200iB, but the dimensions and DH-parameters are determined from the robot in the laboratory.

2.2 Construction of the Cage

After verification of the details, measuring the dimensions of where the robot is fixed were made. The LR Mate 200iB stands in the cage as shown in the figure 5 (here is LR Mate 200iC model). Dimensions of the cage measured and depicted by the RobWorkStudio were created using the XML file. Robot hand stands on the table and around the robot hand is a cage which guarantees the safety of the operator. Position of the LR Mate 200iB in the cage is in the left up corner (if watched from the front view).

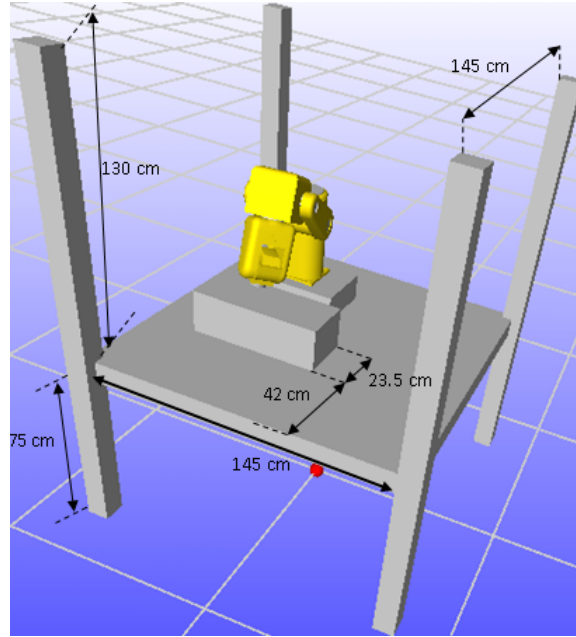


Figure 5: Front view of the LR Mate 200i

Figure 6 shows the robot position from top view and dimensions of the cage where LR Mate 200i stands. As can be seen the shape of the table is square, the real table width is a little bit bigger. A smaller is used because it makes it easier to avoid any damage to the cage. A defined smaller work space gives more safety.

Lengths were measured by using the 50 cm ruler. Of course a bigger one would have been better, so there are probably some errors in the measurements ± 2.5 cm. It's not guaranteed that all values are precise enough.

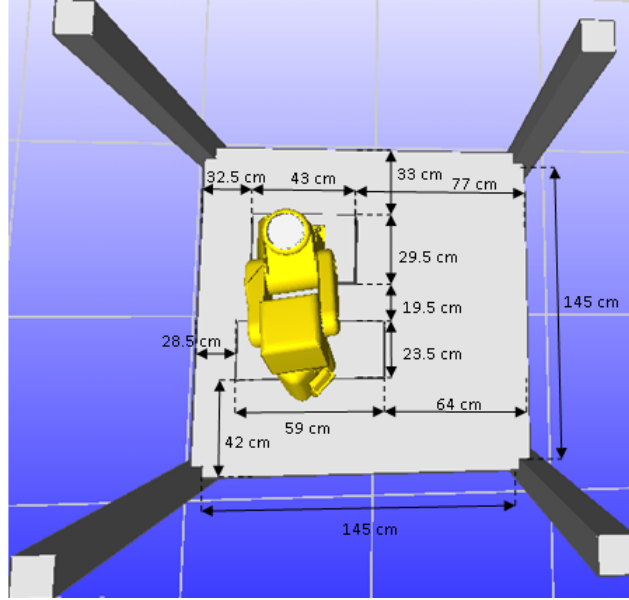


Figure 6: Top view of the LR Mate 200i

Regarding this, max theta angles can be determined. They will limit the movements of the joints. If this is not done, there is a risk in damaging the cage or robot.

2.3 Construction of Joint Angles

Joint angles means the interval of turning between min and max. By default values were set as shown in the figure 7, however a change of those do guarantees that the robot hand can't move too much. If these values are allowed and something is wrong in the state sequence there could be a risk of damage to the window, robot or even worse injure people around. So that is why a decision was made to limit the turning angle of the robot.

```

72 <PosLimit refjoint="Joint1" min="-180" max="180" />
73 <PosLimit refjoint="Joint2" min="-180" max="180" />
74 <PosLimit refjoint="Joint3" min="-180" max="180" />
75 <PosLimit refjoint="Joint4" min="-180" max="180" />
76 <PosLimit refjoint="Joint5" min="-180" max="180" />
77 <PosLimit refjoint="Joint6" min="-180" max="180" />

```

Figure 7: Description of LR Mate 200i in Xml (by default)

Before start an illustration of the robots position in the cage is needed to show

all measured dimensions. With all dimensions and joint positions determined, modeling of the turning angles of joint arms could be done. In the XML file each joint has min and max and is set by θ_{min} and θ_{max} here theta is angle limits of turning.

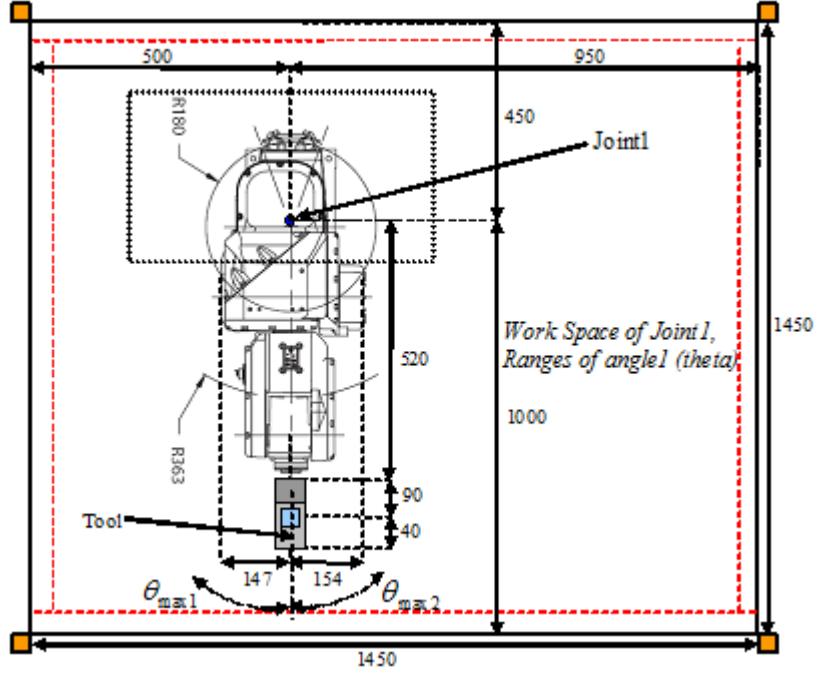


Figure 8: Work space of Joint1 (Top View)

According to figure 8 max movements or angles of joint1 can be calculated:

$$\theta_{min} = \theta_{max1}$$

$$\theta_{max} = \theta_{max2}$$

$$\theta_{max1} = \arctan\left(\frac{500 - x}{520 + 90 + 40}\right) \quad (1)$$

$$\theta_{max2} = \frac{\pi}{2} + \arctan\left(\frac{500 - x}{520 + 90 + 40}\right) \quad (2)$$

In equation 1 the value x is wrong. Depending on the shape of the robot x > 154 is used, x = 180 were chosen. Than max movement range of the joint 1 could be calculated:

$$\theta_{max1} = \arctan\left(\frac{500-180}{520+90+40}\right) = \arctan\left(\frac{320}{650}\right) \approx 26deg$$

$$\theta_{max1} = \frac{\pi}{2} + \arctan\left(\frac{500-180}{520+90+40}\right) = \frac{\pi}{2} + \arctan\left(\frac{320}{650}\right) \approx 116deg$$

Angles of theta can be illustrated from another view (view from left). Using formula 1 other angles can also be calculated, however we not all of them can be calculated, because the joints are dependent of each other and the here the model is only static.

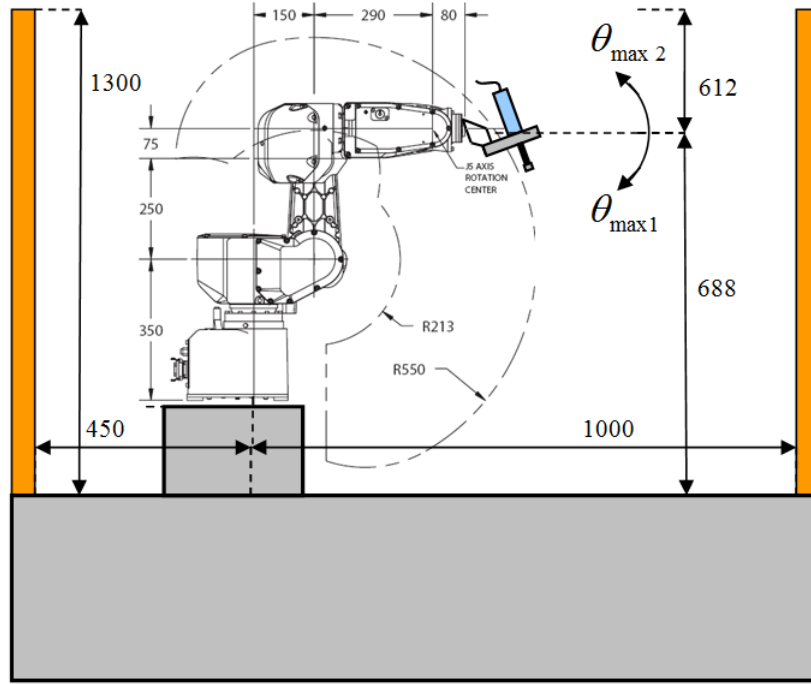


Figure 9: Work space (Left View)

The chosen calculated angles are shown in the figure 10. As can be seen, the angles are much smaller than they can be, because of the safety.

```

70 <PosLimit refjoint="Joint1" min="-26" max="116" /> >
71 <PosLimit refjoint="Joint2" min="-30" max="90" /> >
72 <PosLimit refjoint="Joint3" min="-120" max="0" /> >
73 <PosLimit refjoint="Joint4" min="-45" max="45" /> >
74 <PosLimit refjoint="Joint5" min="-60" max="60" /> >
75 <PosLimit refjoint="Joint6" min="-0" max="0" /> >

```

Figure 10: Description of LR Mate 200i in Xml

2.4 Modeling the Tool Frame

Requirement was to have few parameters which let us easily calibrate the tool frame (drill frame). Before that we need to measure the drill which shown in the figure 11 (not very nice). As we see from the figure 11, there are two unknowns a and b . Values of a and b let define the position of the drill. As we see than we changing b than we can shift to right or left and it depends mostly of the drill construction. Second one called a is height control of the drill and it possibly changed many times. For instance if we broken the drill or changed the new one.

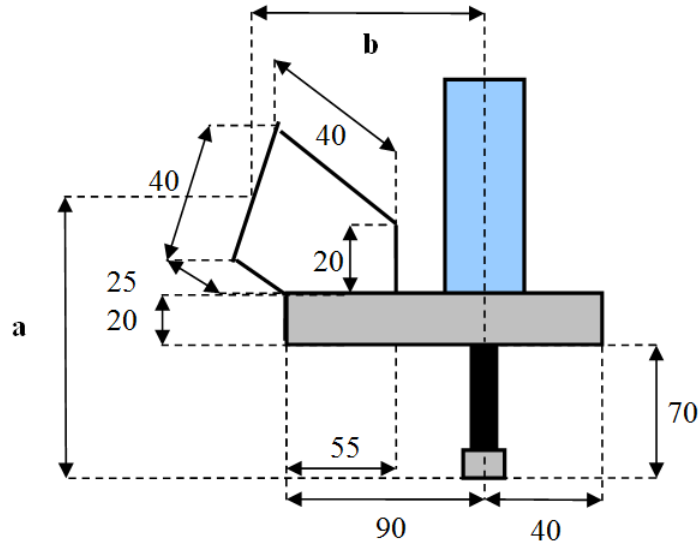


Figure 11: Dimensions of the drill (Tool)

Setting of the tool frame were done through an intermediate tool frame. The intermediate tool frame rotates the frame by 45 degrees, because of the drill construction. Then it's easily to calibrate the tool frame according to the created intermediate frame. As shown in the figure 12 there are two values a and b (for more details look at the figure 11) value b is the width of where drill stand and value a is height of the drill.

```

68 <Frame name="Intermediate_Tool_Frame" refframe="Joint6">
69   <RPY> 0 -45 0 </RPY> <Pos> 0 0 -0.08</Pos>
70 </Frame>
71 <Frame name="ToolFrame" refframe="Intermediate_Tool_Frame">
72   <RPY> 0 -90 0 </RPY> <Pos> -0.1 0 -0.1</Pos>
73 </Frame>
74

```

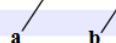


Figure 12: Dimensions of the drill (Tool)

The result of the intermediate tool frame and tool frames are shown in the figure 13. Configuration of it is shown in the XML file extract (figure 12). Using this any desired tool can easily be used. For instance if a longer drill or a tool with different dimensions than the XML file can be easily modified.

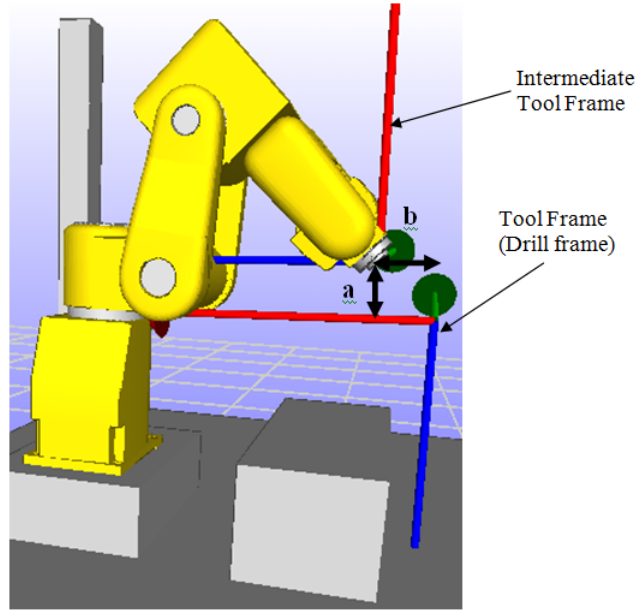


Figure 13: Tool frame (drill frame)

As we can see from the figure 13 tool frame can be easily modified. Tool frame is important for the future parts, like path planner. Path planner takes Cartesian coordinates from the letter model and moves tool frame regarding it. By moving tool frame all join values change and we need to parse them and send to real robot, but this discussion is out of this chapter scope.

2.5 Construction of XML File

Xml file were constructed taking this into consideration:

1. Base frame setup. Discussion where to set the base frame. It's chosen to put center down under the table (cage).
2. Dimensions of the cage. All dimensions were measured and shown in the chapter: "Construction of the cage". Regarding this, XML file has parameters which depict the cage.
3. Dimensions of the LR Mate 200iB. Using the dimensions of this the joints in the XML file were created.
4. Model graphics of the LR Mate 200iC. This 3D model were used to illustrate the simulated robot behavior.
5. Collision setup. There were added reference to a collision setup which indicates collisions of the LR Mate 200iC model.

2.6 Construction of Joints

LR Mate 200iB has six axes. Each axis is revolute, dimensions are given by the manufacturer. Using this, the creation of joints in XML can be done. There are two ways to do it:

1. Calculate positions of the joints by using the template of the Fanuc 200i model. This was given by RobWork, however the models are not quite the same.
2. Calculate DH parameters and using this to build the XML file with joints.

The first approach was used because in the assignment there was no description of how to do it. First approach were used for someone but parameters were not the same, so new ones were needed to be calculated. Before the start of constructing the joints, it was necessary to define where to put the base frame. From the base frame all other joints are constructed. There were some discussion of where to put the base frame. One opinion was to put the base frame at the one of the table corners, another one where robot center is and the last one down under at the center of the table. It was decided to choose the last one, because it was the easiest way to start constructing the table and the joints. Reason was that the base frame will stand independently to any of the objects around it. Second approach will be more useful, the reason is that it's easier to understand and adopt new changes to the robot. For instance if the length of the Robot arm changed. It's actually necessary to change DH parameters in the XML by entering the spots shown in figure 14.

```
<DHJoint name="Joint1" alpha="0" a="0" d="0" offset="-90" state="Active" />
<DHJoint name="Joint2" alpha="-90" a="0.15" d="0" offset="-90" state="Active" />
```

Figure 14: Example of configuration by DH parameters

3 Letter Font Application

This section presents the letter and font GUI application, the input comes from a user, and the user can choose a set of settings, the main settings are the font, font style and font size. This application writes a set of milling points to an output .txt file. The letter font model is implemented under vision (image processing), so all theory from image processing could be applied there. The purpose of this application is to return letters milling points, and these milling points would be used to performing milling (carving) by the robot arm Fanuc 200ib (figure 15).

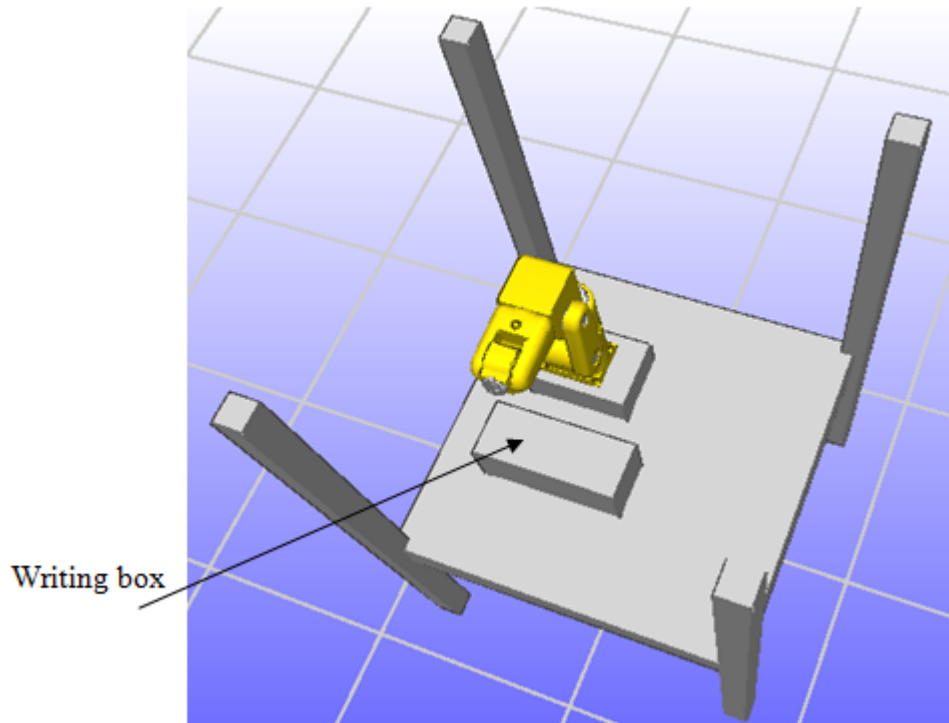


Figure 15: Writing box position

Letter font model is an important step for performing actual (real) letter carving in foam, another step would be inverse kinematics, which the path planner

uses for planning movements between frames. The applications main window is shown in figure 16, there are also provided a description of the application elements.

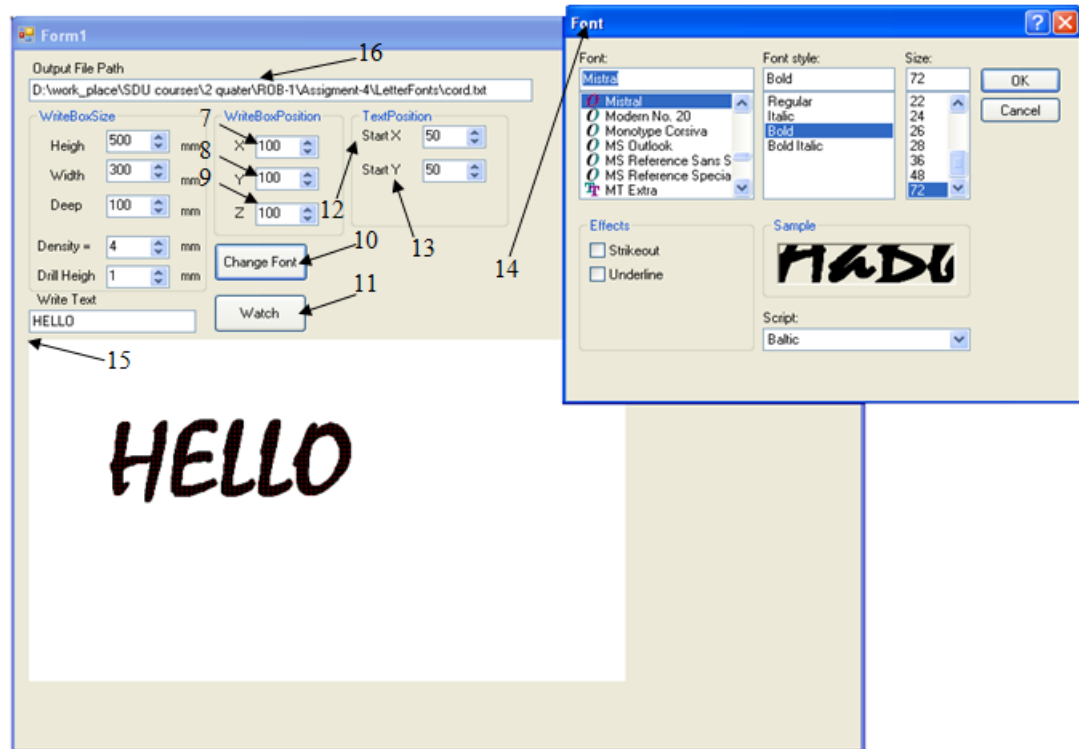


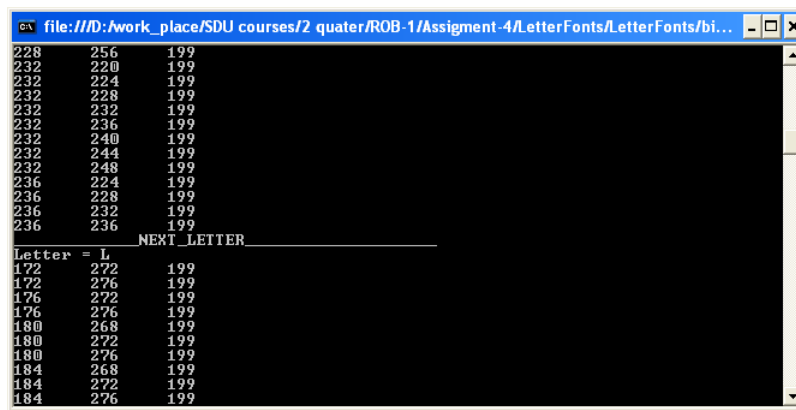
Figure 16: Letter font application

1. Sets writing box height
2. Sets writing box width
3. Sets writing box deep
4. Set the density of drilling points (red points)
5. Sets the drill height, in other words, how deep we want to drill
6. The place of letter or text
7. Set the writing box position according X coordinate (in Cartesian domain)
8. Set the writing box position according Y coordinate (in Cartesian domain)
9. Set the writing box position according Z coordinate (in Cartesian domain)
10. Call the font window

11. Display the text on writing box
12. Set the text position according X (in Cartesian domain) on writing box
13. Set the text position according Y (in Cartesian domain) on writing box
14. The font window, where we can the size, font, font style of text
15. The start position of writing box
16. The path of txt file, where all the milling points are stored

The text fields have been changed to make the understanding of the application easier, for an updated picture see figure 24 in appendix.

The default output path for the generated .txt file is the same directory as the application directory but can be changed to a user defined. The output file stores all the milling points, this information can also be seen in the C# application (figure 17). Second it is necessary to set the writing box position, because according to this position, is where the milling points start, and the other settings are optional. The density option is also written to the output file because it is used by the RobWorkStudio plugin to find the gaps in the letters and of course also to define how exact the milling tool will follow the letters outlines.



```

C:\ file:///D:/work_place/SDU courses/2 quater/ROB-1/Assignment-4/LetterFonts/LetterFonts/bi...
228 256 199
232 220 199
232 224 199
232 228 199
232 232 199
232 236 199
232 240 199
232 244 199
232 248 199
236 224 199
236 228 199
236 232 199
236 236 199
NEXT LETTER
Letter = L
172 272 199
172 276 199
176 272 199
176 276 199
180 268 199
180 272 199
180 276 199
184 268 199
184 272 199
184 276 199

```

Figure 17: Letter font application console

As it can be seen in figure 17 each letter will be milled until it is finished before it begins on a new one. If this weren't the case the milling tool would never know when a new letter had begun and wouldn't rise the tool to start at the new starting point.

3.1 Letter labeling application

The image is dealt with in byte mode and the image is represented like a figure 18. As we can see the image is composed of R-red, G-green, B-blue bytes and padding bytes. The only thing that is interesting are RGB bytes, the padding bytes describes how the image is stored in memory, and they do not have any influence of displaying the image. The programming becomes complex, because in order to jump to the next image row we need the count offset (which is equal to (stride - image width)), and in order to access the next pixel we need to jump three bytes.

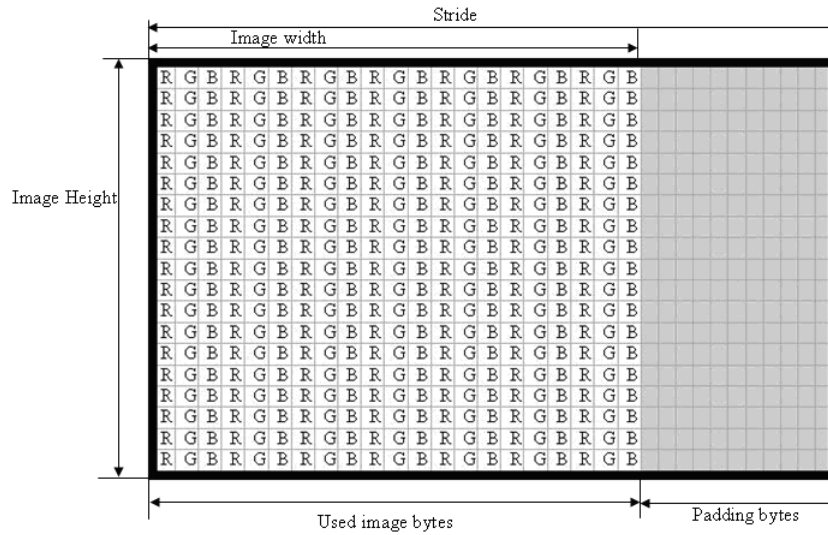


Figure 18: Image in byte mode

There are two images, one image is the original, and the other image is a clone of the original one. In the clone image each pixel bytes are accessed as an array, where the values are changed, it's done so, because it is faster, and easier to access to write values, rather than to create a huge array.

A 8-connectivity matrix were used see figure 19, there are a lot of options but this one were chosen, it is not too big and it is easy to manage this type of matrix (not like triangles or cross shape).

1	2	3
4	5	6
7	8	9

Figure 19: 9 pixel connectivity matrix

The connectivity matrix is used to take values from the original image and write the result to a clone image. Another reason of choosing 3x3 connectivity matrix is that we lose only the image's corner pixels. It was a possibility to choose 2x2 matrix, but the time of image processing would increase significantly, and the corner pixels would also be lost in the process.

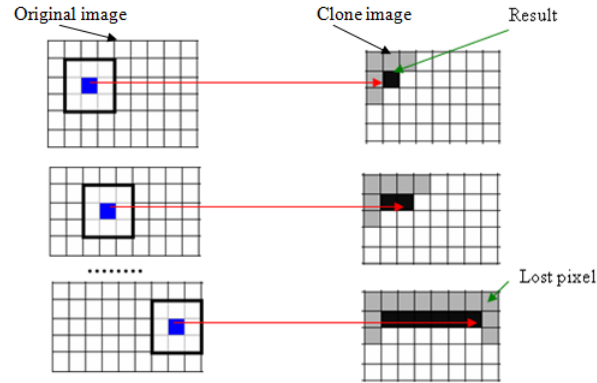


Figure 20: 9 pixels connectivity matrix over image

The letters (objects) labeling algorithm is shown in figure 20. At the beginning the image is converted to binary, it made it easier to deal with the information, non-white pixels were set to black.

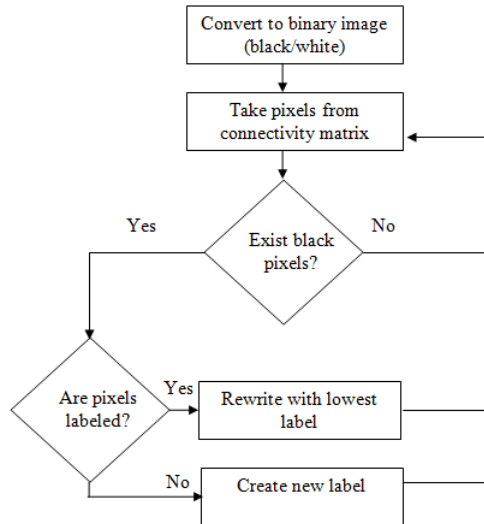


Figure 21: Letters labeling algorithm

When a binary image is used there are only two values: black which represent the letter (or object), and white which are assumed to be the background. In the next step, pixels from the connectivity matrix are looked at and a check if black pixels (which belongs to letter or object) exists are done. If there are no black pixels, a move of the connectivity matrix by one pixel in a particular direction and a repeat of the black pixel checking are done. But if there are black pixels, a check are performed in order to figure out if a new label (detected new object) needs to be created, or some of these pixels belongs to a previously detected letter (or object), and it is enough to rewrite the previous labels. The example of labeling text letters is shown in figure 22. We have a binary image this image is called "clone image" it is a copy of the original. By performing the letter labeling algorithm (figure 21) a detection of how many different letters (objects) there are can be done. The important thing is that the letter labeling algorithm are repeated twice, at first time it's arbitrarily chosen which corner of the image where the algorithm starts, for instance it's convenient to start in the top/left corner, when the algorithm repeats an opposite corner is chosen to start from, it could be the bottom/right one.

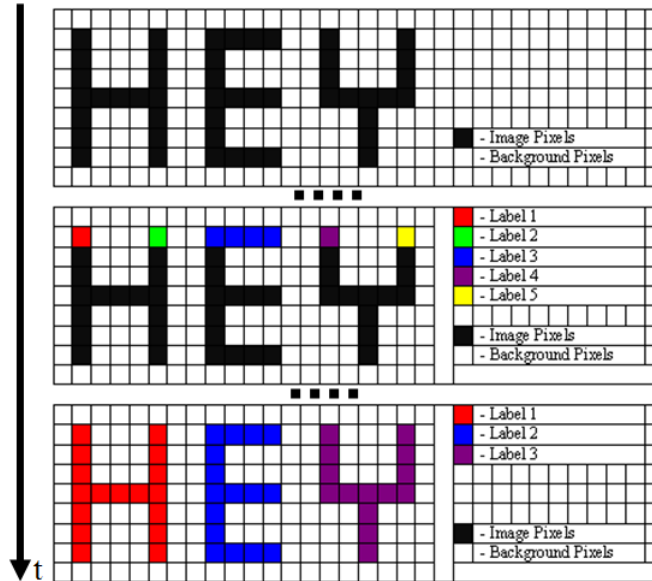


Figure 22: Letters labeling example

If the algorithm wouldn't start from the opposite corner, the algorithm would detect more letters (objects) then there exists, in such letters as U, Y, H, V, W... , because the connectivity matrix is too small, but if a connectivity matrix was chosen to be quite large, there are risks: i.e. detect two letters as one letter, loose more pixels (near image corners).

3.2 Letter Font Application Extension

At the moment the letter font application deals with text as an image, in other words, the letters are consider as an image (which can be saved, painted or color changed). It is possible to load an image directly (this feature is not implemented on final application, because the task was to deal with letters), so by using this application source code and knowing the basics of object oriented programming, it's easy to load any picture here as shown in figure 23.

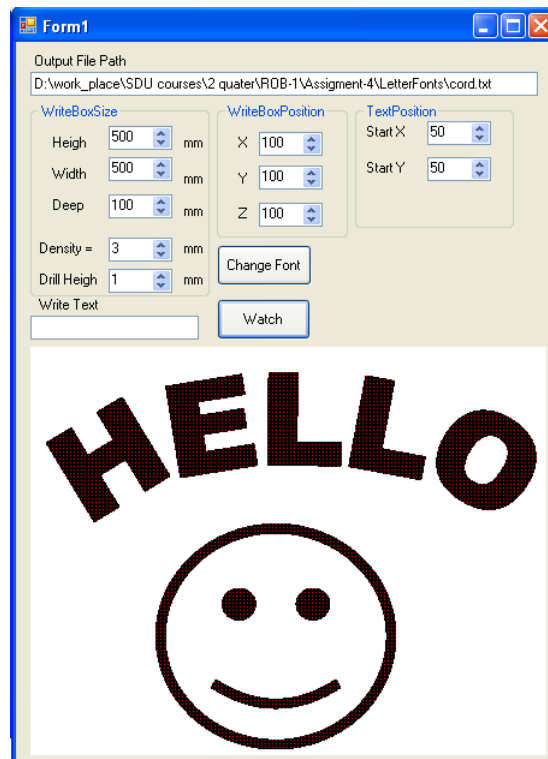


Figure 23: Letter font application

What is interesting is that more image processing knowledge could be applied here, such as edge detection, Hough transformation etc., and this mean that this letter font application is universal, we can easy make extensions.

4 RobworkStudio plugin

In addition to the Letter Font application a plugin for RobworkStudio has been created. The purpose of this plugin is to read Cartesian coordinates from a file

similar to the one created by the Letter Font program and then output both a RobworkStudio simulation file and a jnt-file for direct use with the Fanuc robot. In addition to the main functionality the plugin has a button to test the input coordinates, and examine if they correspond with the Fanuc robot workspace and outputs the joint values, the button is called "Test".

4.1 Parser

To parse an input file a list of criteria must be met:

- Txt file format. The parser only accepts files with the extension txt, even if the files are written in any form of clear text.
- Letter allocation. Coordinates are allocated into subgroups according to letters. The parser scans for the keyword *Letter = \$* to signify a new group of coordinates allocated to a specific letter. A letter can be any character or number except space.
- Letter separation. Letters are separated by the keyword *NEXT_LETTER* to signify the end of a group.
- Coordinate forming. Coordinates must be on individual lines, each component separated by a tabular. Coordinates are assumed to be integer (pixel) values corresponding to millimeter offsets on the individual axes from an origin specified by the plugin based on the selected robot device. The X axis specifies the height of text, 0 and negative being furthest from the robot. The Y axis specifies the width of the text, with 0 and negative to the left of the robot. The Z axis specifies the elevation of the toolhead when milling, with 0 at table level and positive values elevating the drill above it.
- Density specification. The toolhead will be lifted with a fixed value when moving between letters. This is to avoid carving connecting paths between them. To enable the plugin to lift the toolhead when milling complex letters or gaps in these, the density of coordinates must be specified in the file with the keyword *Density = \$*. The parser assumes that coordinate density is quadratic and therefore equal on both the X and Y axes.

If an input file respects this specification the parser will most likely parse the file successfully.

The actual implementation of the parser is described in the following section.

4.1.1 Implementation

The `sscanf` (formatted string reader) and `strstr` (locate substring) is used to parse the input file. Which is read by `ifstream` (input file stream). All parse

functionality is C portable, primarily because the programmer was familiar with these functionalities, and the functionality was appropriate for the task.

4.2 Path planning and joint angle solving

Despite its name, path planning is not a feature that is used at all in the final implementation of the plugin. The functionality is implemented based on the Robworks manual, but the overhead imposed by the planner is considerable and the gain was deemed insignificant in the end since mostly all coordinates are inside the target milling material and therefore either always or never collides with an object. In addition the paths between these coordinates are generally short and trivial, rendering the pathPlanner unnecessary. A function called *pathPlanner* exists in the project namespace but is never used and is only provided as an addon for future use.

The main part of the plugin handles the conversion from the parsed Cartesian coordinates into rotation values for each of the robots joints. These are determined by using inverse kinematics (henceforth written as IK). More specifically, one of the inbuilt iterative IK solvers called ResolvedRateSolver is used through the IK wrapper IKMetaSolver. The values are then stored in a vector array of configurations. This array is passed to a function that converts each configuration into a joint parameter configuration and printed to the output jnt file.

4.2.1 Implementation

The solver is called when the user presses the *generate* button which is only available if the parse button has previously been pressed and a valid file has been selected.

When the *generate* button is clicked, the State is updated to a State with the robot device in the pregenerated "home" configuration close to the working area. This is preferable when using the IK solver since it iteratively tries to solve the Cartesian coordinates based on the current State and therefore might reduce the total number of iterations needed to solve the problem.

It then iteratively solves each letter coordinate taken in relation to the workarea origin. During this it takes into consideration the distance from the previous coordinate to the current, and if this is larger than the density value of the letter coordinates, the tool head is lifted by inserting two additional configurations with increased Z-values.

When all coordinates has been solved or failed, the vector array of configurations is parsed to a function called *writeJNT*. This function uses `fprintf` to create a string with the joint values formatted in and time frame constant added. This is done for each configuration and creates a file in the application directory called `%devicename%.jnt`.

Additionally the configurations are converted to states and written to a replay file which allows the user to simulate the output path in RobWorkStudio. This file is also placed in the application directory and is called %device-name%.rwplay.

5 Testing the System

The first test of the LetterFont application and the RobWorkStudio plugin were simulated in RobWorkStudio to make sure that robot didn't make any errors. A little movie of that simulated test is put on the cd. The simulation went very well and the path file was uploaded to the robot to tweak some values to make sure that the height is correct so that the milling tool didn't start too far above or under the foam block. These dummy tests are important to do to make sure that the robot doesn't harm itself or its surroundings.

The second test on the actual robot was a success and was recorded and put on the cd but it runs at twice the actual speed of the Fanuc. The speed of the Fanuc was pretty slow because we use a constant time frame between points which makes the robot move extremely slow when points are close and very fast when distant. It would be more convenient to calculate a time frame that has constant speed between points independent of the distance between them.

6 Conclusion

The robot was properly simulated with RobWorkStudio. A C# application was developed to create milling points that are in Cartesian coordinates from a user written text. A plugin was developed for RobWorkStudio that create the path from the given points and converts the path to a format that the Fanuc robot can understand. The project group gained practical experience of working with a physical robot, and developing a plugin with RobWorkStudio.

Further improvements to our path planner plugin and LetterFont application were not implemented due to time constraints, but the improvements were discussed in this report.

References

- [1] John J. Craig. *Introduction to Robotics Mechanics and Control Third Edition*. Prentice Hall, 2005.

7 Appendix

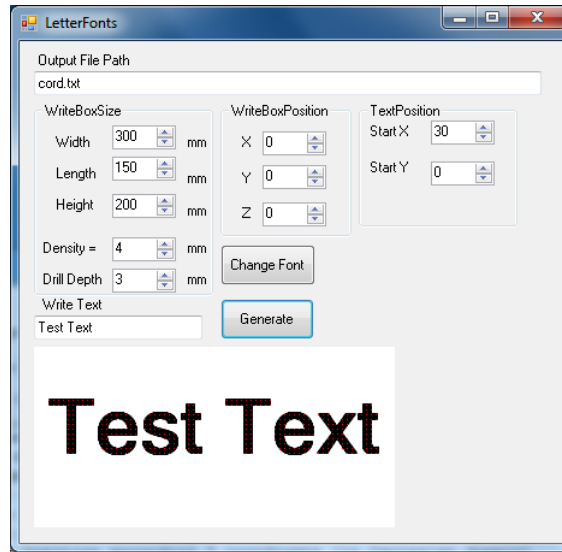


Figure 24: Letter font application console

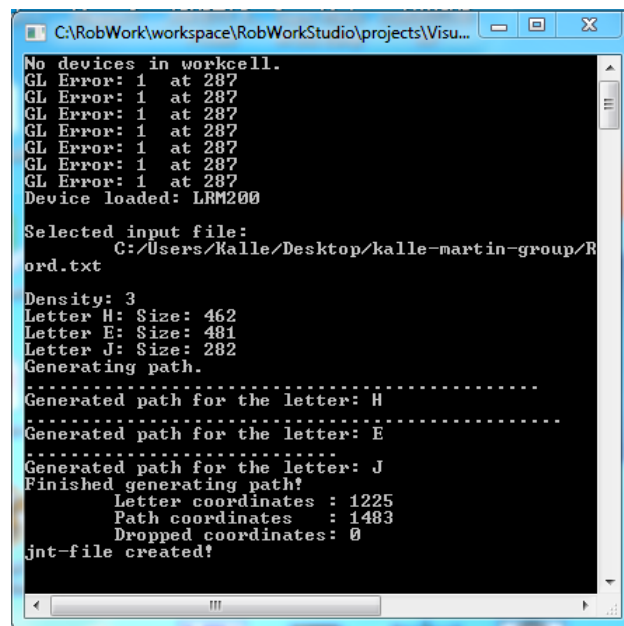


Figure 25: Screenshot of RobWorkStudio console output

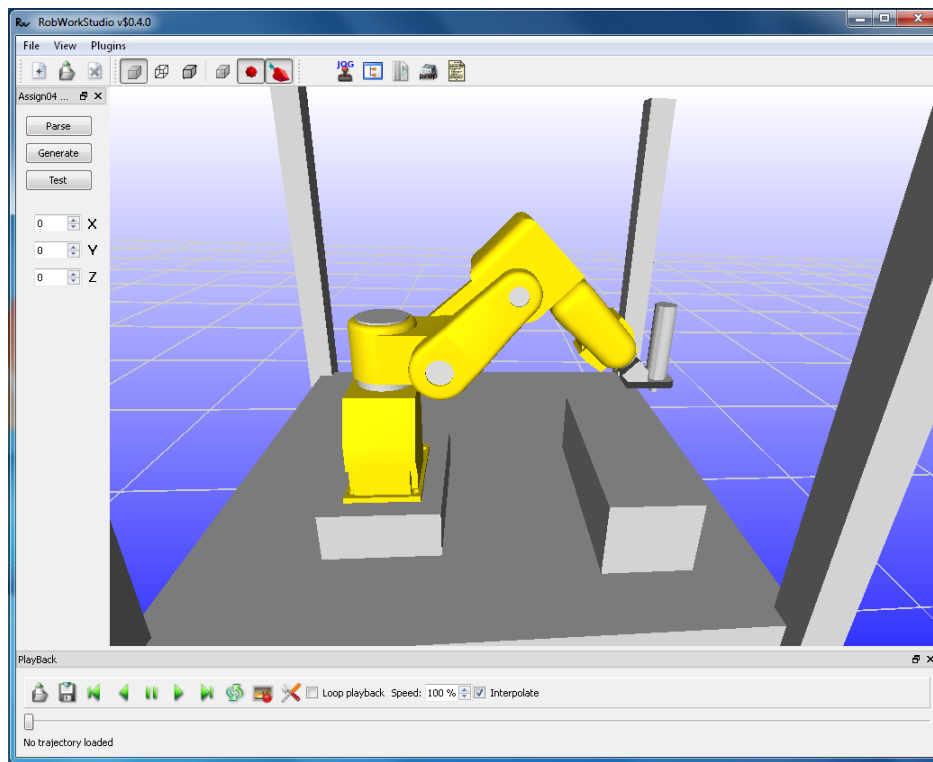


Figure 26: Screenshot of RobWorkStudio with the plugin