Universidad Zaragoza
1542

Trabajo Fin de Grado

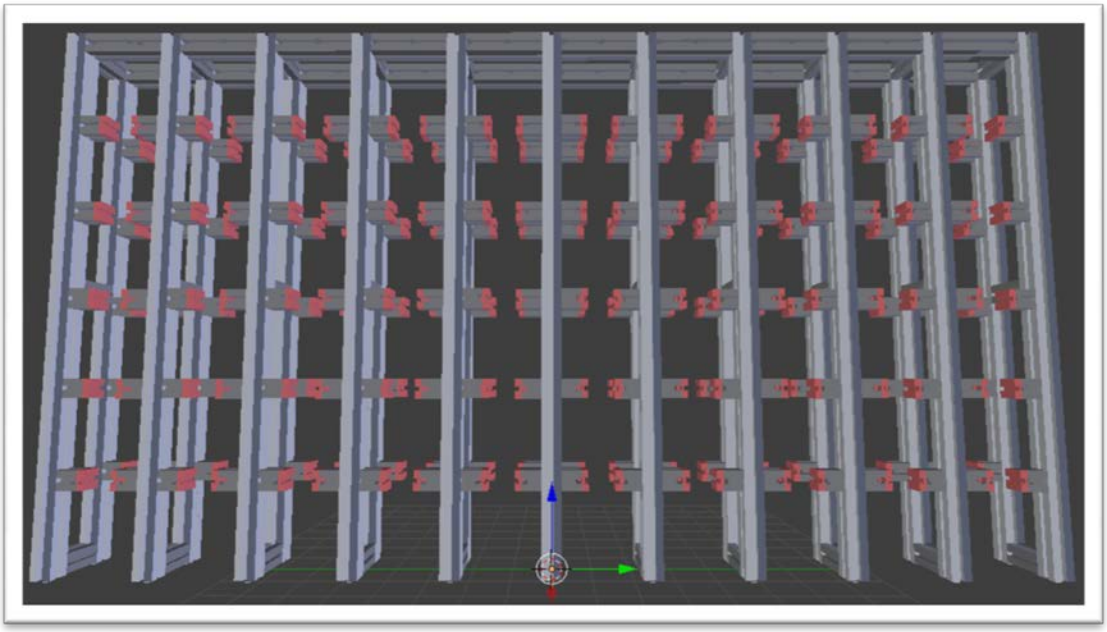
Habilitación del control remoto y visualización para el modelo de un sistema automatizado de almacenamiento y recuperación.

ANEXOS

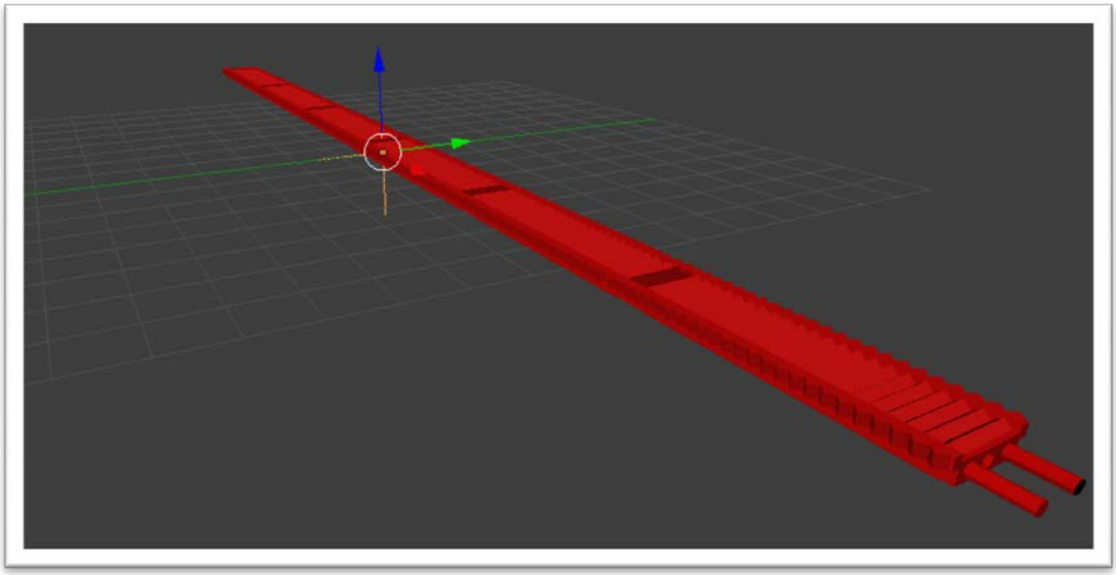
INDICE

1 Modelos 3D

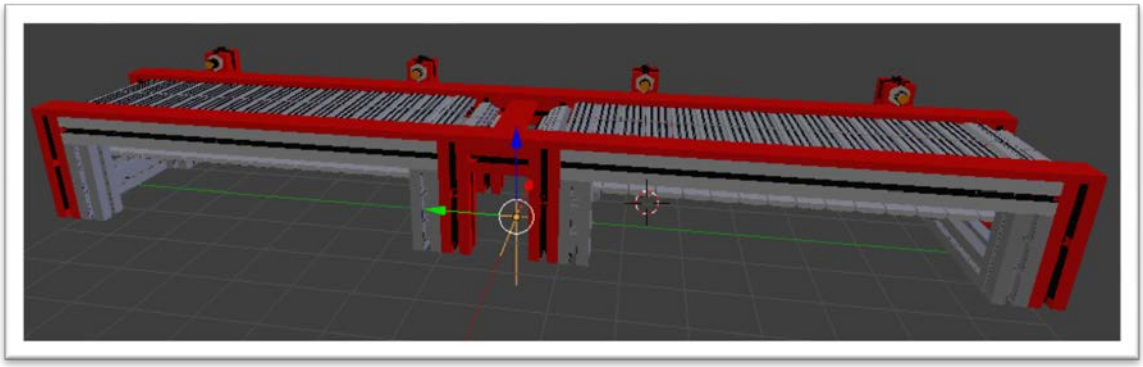
1.1 Rack



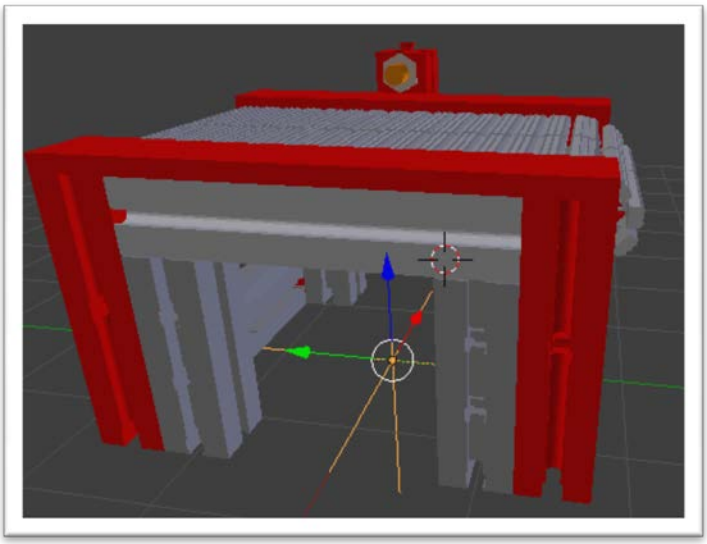
1.2 Railway



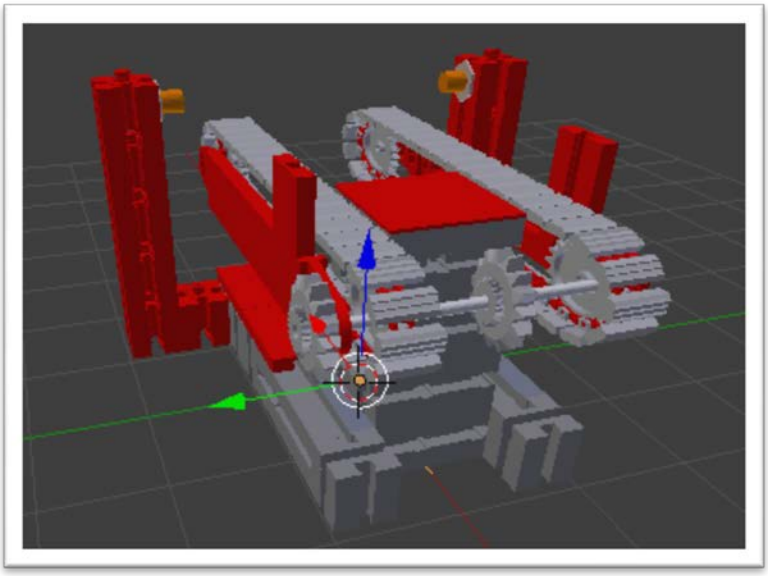
1.3 Machinery_long_belt



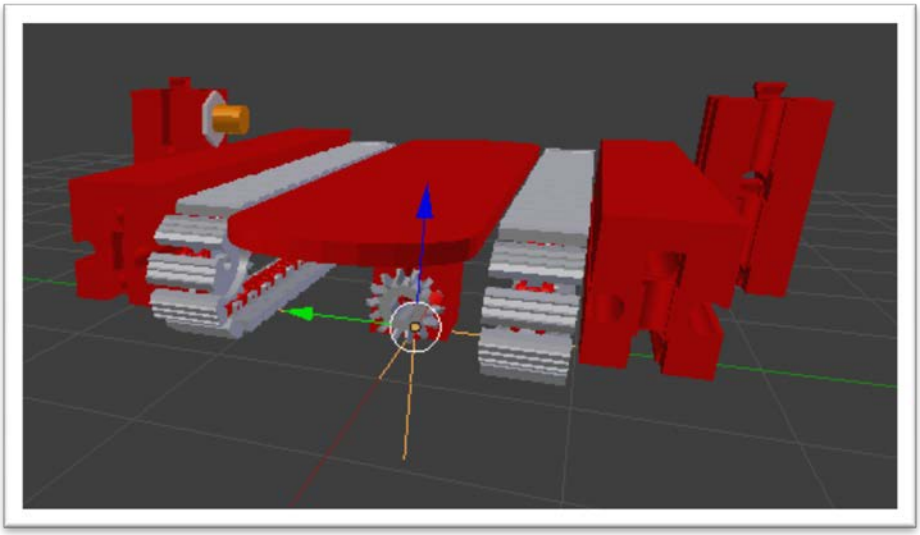
1.4 Machinery_short_belt



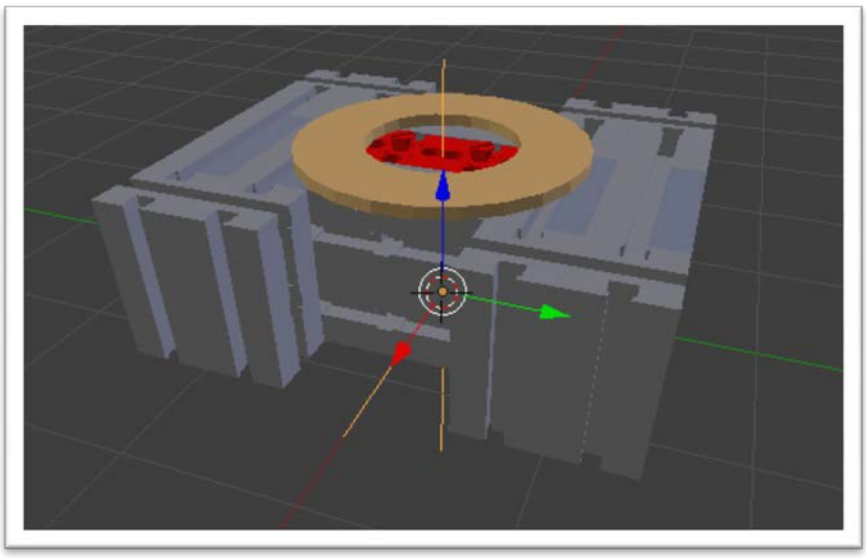
1.5 Machinery_elevator_connection



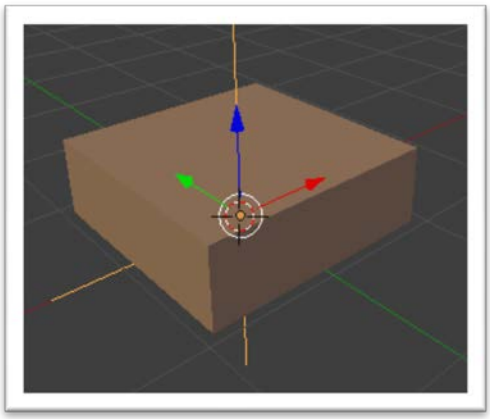
1.6 Machinery_rotatory_top



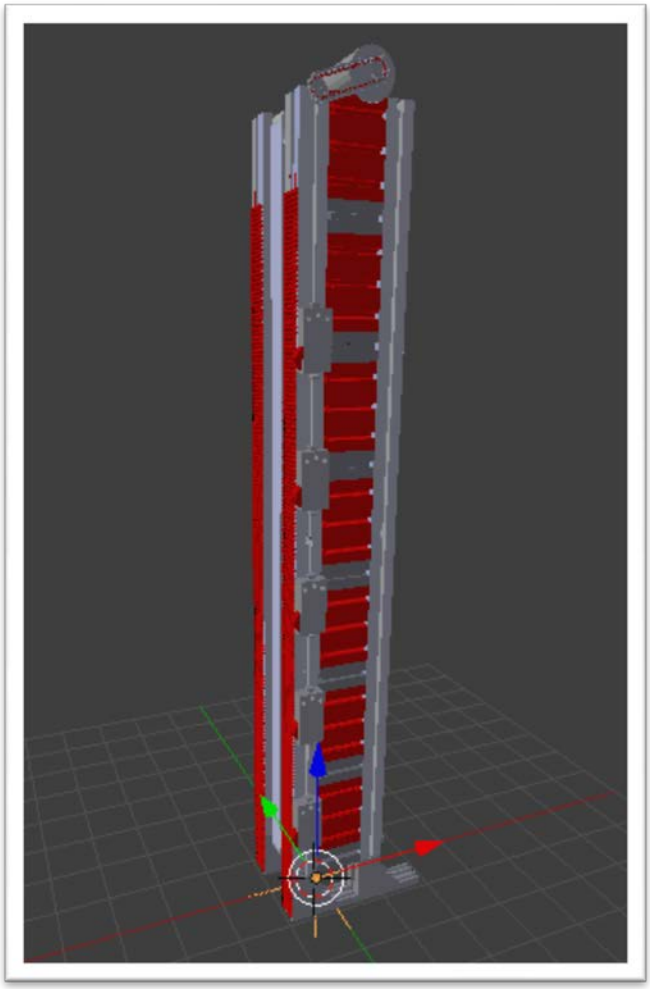
1.7 Machinery_rotatory_bottom



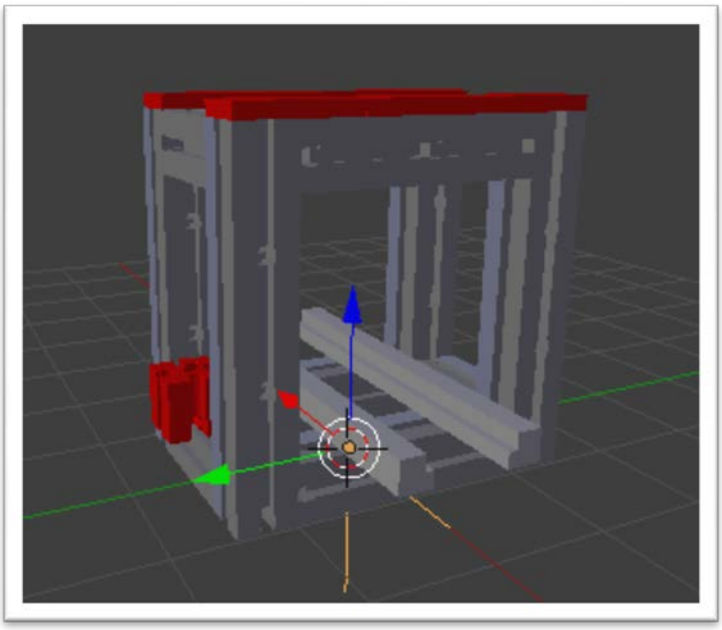
1.8 Box



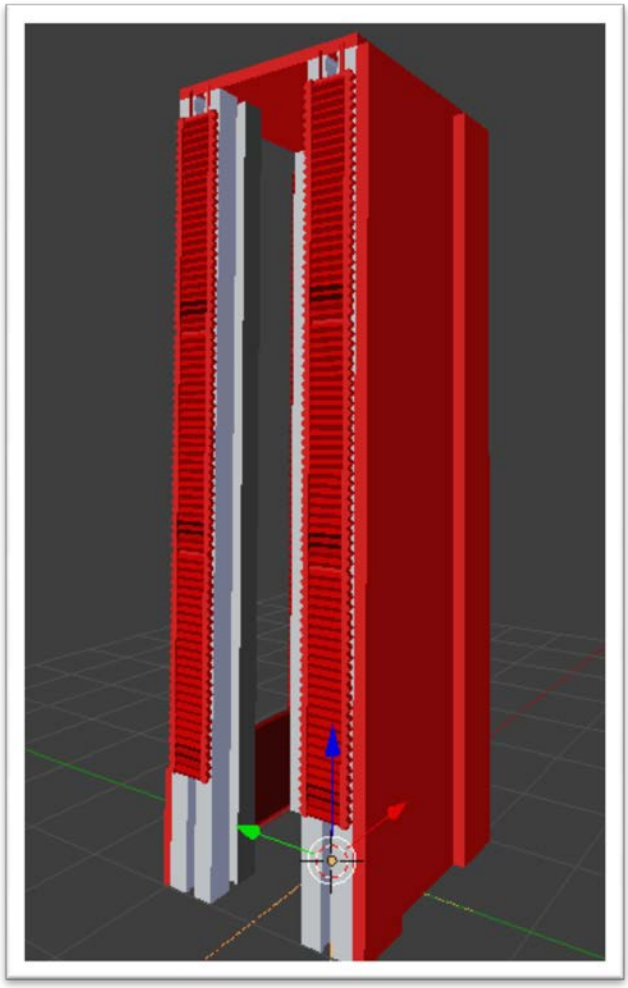
1.9 Elevator_tower



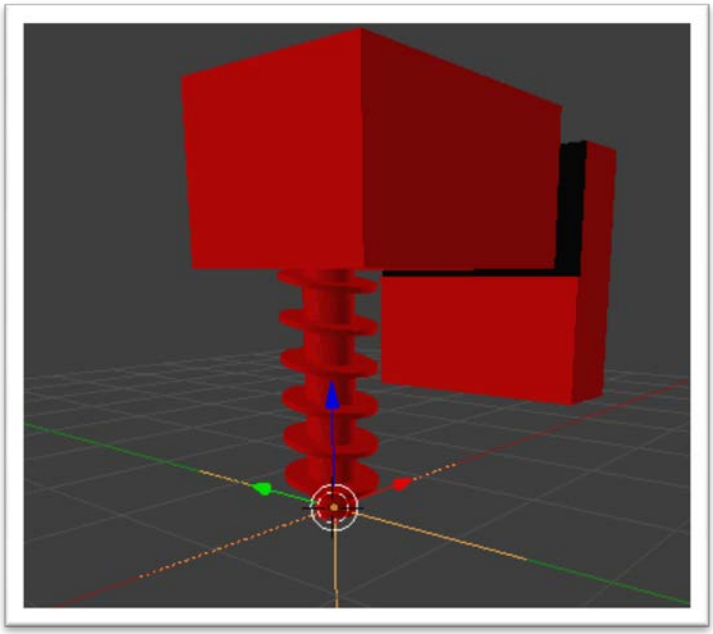
1.10 Elevator_cabin



1.11 Tool_machine_tower



1.12 Tool_machine_tool



2 Código Java

2.1 Class HRL_API

```java
package kodemat.hrl.client;
import java.util.HashMap;

/**
 *
 * @author marco
 */
public class HRL_API {
    // virtual map of possible locations in machines.
    public static HashMap <String, HRL_Component> sensorMap;
    // virtual map of possible locations in rack.
    public static HashMap <Integer, HRL_Component> rackMap;
    public static HRL_Scenario scenario;
    // list of names for boxes and their state
    public static HashMap <Integer, String> names;

    /**
    * this method initializes the variables and build the HRL scenario.
    * It must be called only once.
    */
    public static void start (){
        //create scenario
        scenario=new HRL_Scenario();
        //initialize sensorMap
        sensorMap = new HashMap();
        sensorMap.put("A",null);
        sensorMap.put("B",null);
        sensorMap.put("C",null);
        sensorMap.put("D",null);
        sensorMap.put("E",null);
        sensorMap.put("F",null);
        sensorMap.put("G",null);
        sensorMap.put("H",null);
        sensorMap.put("I",null);
        sensorMap.put("J",null);
        sensorMap.put("K",null);
        sensorMap.put("L",null);
        System.out.println("sensorMap initialized");
        //initialize rackMap
        rackMap = new HashMap();
        for(int i=0; i<=49; i++){
            rackMap.put(i,null);
        }
        System.out.println("rackMap initialized");
        //intitialize the names
        names = new HashMap();
        for(int i=1; i<=50; i++){
            names.put(i,"free");
        }
        System.out.println("names initialized");
    }

    /**
    * It gets the first free number of the HashMap "names".
    * @return the first free number.
    */
    private static int getFreeName(){
        for(int i=1; i<=names.size(); i++){
            if (names.get(i).equals("free")){
                names.put(i,"busy");
                return i;
            }
        }
        return 0;
    }

```

```
67      /**
68      * It turns free the number received in the HashMap "names".
69      * @param number the number to be established as free.
70      */
71      private static void releaseName(int number){
72          names.put(number, "free");
73      }
74
75      /**
76      * It sorts the boxes on the virtual map according to the state of the sensors.
77      * This method will call the relocation method to place the box in the correct place.
78      * @param sensorMapMachines array with the current state of the sensors.
79      */
80      public static void updateSensorMap (boolean[] sensorMapMachines){
81          //this loop checks the movevements of boxes.
82          for(int i=0; i<sensorMapMachines.length; i++){
83              //if sensor i off and box in that position
84              if((sensorMapMachines[i]==false) &&
85                      (sensorMap.get(Character.toString((char)(i+65)))!=null)){
86                  String sensor = Character.toString((char)(i+65));
87                  HRL_Component component= sensorMap.get(sensor);
88                  //elevator case
89                  if (sensor.equals("L") && scenario.elevatorRow != 5){
90                      sensorMap.remove("L");
91                      int cell = scenario.elevatorRow*10 + scenario.elevatorCol;
92                      rackMap.put(cell, component);
93                      component.setPositionIn(scenario.rack,scenario.elevatorRow,scenario.elevatorCol);
94                      System.out.println("Component stored");
95                  }else{
96                      for(int j=0; j<sensorMapMachines.length; j++){
97                          //if sensor j ON and that position empty
98                          if((sensorMapMachines[j]==true) &&
99                                  (sensorMap.get(Character.toString((char)(j+65)))==null)){
100                             if(movementAllowed(i,j)){
101                                 String key = Character.toString((char)(j+65));
102                                 sensorMap.put(key, component);
103                                 relocation(key);
104                                 sensorMap.remove(Character.toString((char)(i+65)));
105                                 System.out.println("Component "+component.model.name+" moved");
106                             }else{
107                                 sensorMap.remove(Character.toString((char)(i+65)));
108                                 releaseName(Integer.valueOf(component.model.name));
109                                 component.delete();
110                                 System.out.println("Component "+component.model.name+" deleted");
111                             }
112                         }
113                     }
114                 }
115             }
116         }
117
118         //this loop checks if new boxes have appeared.
119         for(int i=0; i<sensorMapMachines.length; i++){
120             if((sensorMapMachines[i]==true) &&
121                     (sensorMap.get(Character.toString((char)(i+65)))==null)){
122                 String sensor = Character.toString((char)(i+65));
123                 if (sensor.equals("L") && scenario.elevatorRow != 5){
124                     int cell = scenario.elevatorRow*10 + scenario.elevatorCol;
125                     HRL_Component component = rackMap.remove(cell);
126                     sensorMap.put("L", component);
127                     relocation("L");
128                     System.out.println("Component taken");
129                 }else{
130                     String name = Integer.toString(getFreeName());
131                     sensorMap.put(sensor, new HRL_Component(name,"BOX"));
132                     relocation(sensor);
133                     System.out.println("Component " + name + " created");
134                 }
135             }
136         }
137     }
138
```

```
139     /**
140     * It detects if a change of position has been done between neighboring positions.
141     * @param posIni one of the positions.
142     * @param posEnd the other position.
143     * @return true if the positions are next to each other.
144     */
145     private static boolean movementAllowed(int posIni, int posEnd){
146         if (posIni-posEnd==1 || posEnd-posIni==1){
147             if((posIni==3 && posEnd==4) || (posIni==3 && posEnd==4))
148                 return false;
149             return true;
150         }
151         if((posIni==3 && posEnd==11) || (posIni==11 && posEnd==3))
152                 return true;
153         if((posIni==1 && posEnd==4) || (posIni==4 && posEnd==1))
154                 return true;
155         return false;
156     }
157
158     /**
159     * It places the box in the scenario according to the location in "sensorMap".
160     * @param key contains the key to obtein the box to move.
161     */
162     private static void relocation(String key){
163         HRL_Component component = scenario.getHRL_ScenarioComponet(key);
164         if (component != null)
165             if (component.model.name.equals("machinery_long_belt")){
166                 switch (key) {
167                     case "E":
168                         sensorMap.get(key).setPositionIn(component,0);
169                         break;
170                     case "F":
171                         sensorMap.get(key).setPositionIn(component,1);;
172                         break;
173                     case "G":
174                         sensorMap.get(key).setPositionIn(component,2);
175                         break;
176                     case "H":
177                         sensorMap.get(key).setPositionIn(component,3);
178                         break;
179                 }
180             }else{
181                 sensorMap.get(key).setPositionIn(component);
182             }
183         else
184             System.out.println("Failure in relocation method. Component not found");
185     }
186
187     /**
188     * It handles the moveable components of the scenario.
189     * First it checks if the machines are already in the positions received.
190     * @param tool current position of the tool. Up or Down.
191     * @param rotatoryLeft orientation in degrees for that machine.
192     * @param rotatoryRight orientation in degrees for that machine.
193     */
194     public static void updateScenario (String tool, int rotatoryLeft, int rotatoryRight){
195         //checks the tool machine
196         if (tool.equals("up") && scenario.toolState.equals("down"))
197             scenario.toolUp();
198         if (tool.equals("down") && scenario.toolState.equals("up"))
199             scenario.toolDown();
200         if (rotatoryLeft >= 80 && scenario.rotatoryLeftState == 0)
201             scenario.rotatoryLeft(90);
202         if (rotatoryLeft <= 10 && scenario.rotatoryLeftState == 90)
203             scenario.rotatoryLeft(0);
204         if (rotatoryRight >= 80 && scenario.rotatoryRightState == 0)
205             scenario.rotatoryRight(90);
206         if (rotatoryLeft <= 10 && scenario.rotatoryRightState == 90)
207             scenario.rotatoryRight(0);
208     }
209
```

```
210     /**
211     * Moves the elevator by calling the correspondent method in class HRL_Scenario.
212     * First it checks if the elevator is already in that position.
213     * @param sensorMapElevatorCols Array of Column sensors.
214     * @param sensorMapElevatorRows Array of Row sensors.
215     */
216     public static void updateElevator(boolean[]sensorMapElevatorCols,boolean[]sensorMapElevatorRows){
217         for(int i=0; i<=9; i++){
218             if (sensorMapElevatorCols[i] == true && scenario.elevatorCol != i){
219                 scenario.moveElevator(scenario.elevatorRow, i);
220             }
221         }
222         for(int i=0; i<=5; i++){
223             if (sensorMapElevatorRows[i] == true && scenario.elevatorRow != i){
224                 scenario.moveElevator(i, scenario.elevatorCol);
225             }
226         }
227     }
228 
229     /**
230     * It receives the intitial state of boxes in the rack in order to update
231     * the map and place them into the respective cells.
232     * It should be called at the initialization by the SAP system.
233     * @param rack information of the state of the cells.
234     */
235     public void rackInitialState (boolean[] rack){
236         //creation of components into the virtual map of the rack
237         for(int i=0; i<rack.length; i++){
238             if (rack[i] == true){
239                 String name = Integer.toString(getFreeName());
240                 rackMap.put(i, new HRL_Component(name,"BOX"));
241             }
242         }
243         //placement of the components according to the virtual map
244         for(int i=0; i<rackMap.size(); i++){
245             HRL_Component component = rackMap.get(i);
246             if (component != null){
247                 int row = i/10;
248                 int col = i%10;
249                 component.setPositionIn(scenario.rack, row, col);
250             }
251         }
252     }
253 }
```

2.2 Class HRL_Component

```java
package kodemat.hrl.client;
import kodemat.visudata.visuComponents.VisuComponent;
import kodemat.visudata.VisuRotation;
import kodemat.visudata.VisuVector3f;

/**
 *
 * @author marco
 */

public class HRL_Component extends LayoutClientImpl implements SimulinkAPI{
    //It is the instace used to implement this class.
    public VisuComponent model;
    //This variable contains all mapped positions.
    public HRL_Positions positions = new HRL_Positions();

     /**
     * Constructor of the class "HRL_Component" Needs the name of the model we want and coordinates.
     * @param name the name of the compoenent we want.
     * @param x location in x axis.
     * @param z location in z axis.
     * @param y location in y axis.
     */
    public HRL_Component (String name, String designator, float x, float z, float y){
        this.model = this.createModel(name, "HRL/"+designator+".j3o");
        this.setNodeTranslation(model, new VisuVector3f(x, z, y));
    }

     /**
     * Constructor of the class "HRL_Component" Needs the name of the model we want.
     * Location in 0,0,0, by default.
     * @param name the name of the compoenent we want.
     */
    public HRL_Component (String name, String designator){
        this.model = this.createModel(name, "HRL/"+designator+".j3o");
        this.setNodeTranslation(model, new VisuVector3f(0, positions.row5, 0));
    }

    /**
     * It deletes the component created by this class.
     */
    @Override
    public void delete (){
        helper.deleteComponent(this.model.name);
    }

     /**
     * Establishes an orientation for the component.
     * @param x location in degrees around x axis
     * @param z location in degrees around z axis
     * @param y location in degrees around y axis
     */
    @Override
    public void setOrientation (float x, float z, float y) {
        this.setNodeRotation(model, new VisuRotation(x, z, y));
    }

    /**
     * Sets the component in a new point.
     * @param x location in x axis.
     * @param z location in z axis.
     * @param y location in y axis.
     */
    @Override
    public void setPosition (float x, float z, float y) {
        this.setNodeTranslation(model, new VisuVector3f(x, z, y));
    }

```

```
69      /**
70       * Takes the component to Long belt machine.
71       * @param component the component in which the object will be placed.
72       * @param sensor position inside the machine.
73       */
74      public void setPositionIn (HRL_Component component, int sensor){
75          if(component.model.name.equals("machinery_long_belt")){
76              if(sensor>=0 && sensor<4){
77                  this.childOf(component);
78                  this.model.setTranslation(positions.machinery_long_belt_positions[sensor]);
79                  this.model.setTranslation(positions.machinery_long_belt_positions[sensor]);
80              }else{
81                  System.out.println("Invalid argument.");
82                  System.out.println("Possible sensors are: 0, 1, 2, 3.");
83              }
84          }else{
85              System.out.println("Invalid argument.");
86              System.out.println("Only moving inside \"machinery_long_belt\" is possible);
87          }
88      }
89
90       /**
91       * Moves the component to a single sensor machine by changing its father.
92       * @param component the component in which the object will be placed.
93       */
94      public void setPositionIn (HRL_Component component){
95          this.childOf(component);
96          switch (component.model.name) {
97              case "machinery_rotatory_top_left":
98                  this.setPosition(0,positions.rotatory_machine_height,0);
99                  this.setPosition(0,positions.rotatory_machine_height,0);
100                 break;
101             case "machinery_rotatory_top_right":
102                 this.setPosition(0,positions.rotatory_machine_height,0);
103                 this.setPosition(0,positions.rotatory_machine_height,0);
104                 break;
105             case "elevator_cabin":
106                 this.setPosition(0,positions.cabin_offset_V*(-1),0);
107                 this.setPosition(0,positions.cabin_offset_V*(-1),0);
108                 break;
109             default:
110                 this.setPosition(0,positions.row5,0);
111                 this.setPosition(0,positions.row5,0);
112                 break;
113         }
114     }
115
116     /**
117      * Sets the object in a new cell of the rack.
118      * @param row row where to put the component, from 0 to 4.
119      * @param col column where to put the component, from 0 to 9.
120      */
121     public void setPositionIn (HRL_Component component, int row, int col){
122         if(row >= 0 && row <= 4 && col >= 0 && col<=9){
123             this.childOf(component);
124             model.setTranslation(positions.rack_positions[row][col]);
125             model.setTranslation(positions.rack_positions[row][col]);
126         } else {
127             System.out.println("Row values: 0-4");
128             System.out.println("Column values: 0-9");
129         }
130     }
131
132      /**
133      * Sets the object into the reference system of its father.
134      * @param father a different HRL_Component which will be the father.
135      */
136     @Override
137     public void childOf (HRL_Component father) {
138         addChildNode(this.model.getId(), father.model.getId());
139     }
```

2.3 Class HRL_Scenario

```
1  package kodemat.hrl.client;
2
3  /**
4   *
5   * @author marco
6   */
7  public class HRL_Scenario {
8      //This variable contains all mapped positions.
9      private HRL_Positions positions;
10     //List of components for the scenario.
11     public HRL_Component floor;
12     public HRL_Component rack;
13     public HRL_Component railway;
14     public HRL_Component elevator_tower;
15     public HRL_Component elevator_cabin;
16     public HRL_Component machinery_elevator_connection_left;
17     public HRL_Component machinery_elevator_connection_right;
18     public HRL_Component machinery_short_belt_left;
19     public HRL_Component machinery_short_belt_right;
20     public HRL_Component machinery_rotatory_bottom_left;
21     public HRL_Component machinery_rotatory_bottom_right;
22     public HRL_Component machinery_rotatory_top_left;
23     public HRL_Component machinery_rotatory_top_right;
24     public HRL_Component machinery_long_belt;
25     public HRL_Component machinery_short_belt_input;
26     public HRL_Component machinery_tool_machine_tower;
27     public HRL_Component machinery_tool_machine_tool;
28
29     //initial positions for the moveable components.
30     public int elevatorRow = 5;
31     public int elevatorCol = 0;
32     public String toolState = "up";
33     public int rotatoryLeftState = 0;
34     public int rotatoryRightState = 0;
35
36
37      /**
38      * Constructor of the class.
39      * It creates the components previously defined at their respective positions in the scenario.
40      * It will also establish the relationships father-son.
41      */
42     public HRL_Scenario (){
43         positions = new HRL_Positions();
44
45         floor = new HRL_Component("floor","FLOOR",0,0,0);
46         rack = new HRL_Component("rack","RACK",16,0,0);
47         railway = new HRL_Component("railway","RAILWAY",13,0,0);
48         elevator_tower = new HRL_Component("elevator_tower","ELEVATOR_TOWER",0,0,0);
49         elevator_cabin = new HRL_Component
     ("elevator_cabin","ELEVATOR_CABIN",0,1.3f,positions.cabin_offset_H);
50         machinery_elevator_connection_left = new HRL_Component
51          ("machinery_elevator_connection_left","MACHINERY_ELEVATOR_CONNECTION",10,0,positions.col0);
52         machinery_elevator_connection_right = new HRL_Component
53          ("machinery_elevator_connection_right","MACHINERY_ELEVATOR_CONNECTION",10,0,positions.col9);
54         machinery_short_belt_left = new HRL_Component
55                 ("machinery_short_belt_left","MACHINERY_SHORT_BELT",5,0,positions.col0);
56         machinery_short_belt_right = new HRL_Component
57                 ("machinery_short_belt_right","MACHINERY_SHORT_BELT",5,0,positions.col9);
58         machinery_rotatory_bottom_left = new HRL_Component
59                 ("machinery_rotatory_bottom_left","MACHINERY_ROTATORY_BOTTOM",0,0,positions.col0);
60         machinery_rotatory_bottom_right = new HRL_Component
61                 ("machinery_rotatory_bottom_right","MACHINERY_ROTATORY_BOTTOM",0,0,positions.col9);
62         machinery_rotatory_top_left = new HRL_Component
63                 ("machinery_rotatory_top_left","MACHINERY_ROTATORY_TOP",0,1,0);
64         machinery_rotatory_top_right = new HRL_Component
65                 ("machinery_rotatory_top_right","MACHINERY_ROTATORY_TOP",0,1,0);
66         machinery_long_belt = new HRL_Component
67                 ("machinery_long_belt","MACHINERY_LONG_BELT",0,0,0);
68         machinery_short_belt_input = new HRL_Component
69                 ("machinery_short_belt_input","MACHINERY_SHORT_BELT",0,0,positions.col0-5);
70         machinery_tool_machine_tower = new HRL_Component
71                 ("machinery_tool_machine_tower","TOOL_MACHINE_TOWER",3,0,positions.sensor0);
72         machinery_tool_machine_tool = new HRL_Component
73            ("machinery_tool_machine_tool","TOOL_MACHINE_TOOL",positions.tool_offset,positions.tool_up,0);
```

```
74
75          //Relationships father-son
76          elevator_cabin.childOf(elevator_tower);
77          elevator_tower.childOf(railway);
78          machinery_rotatory_top_left.childOf(machinery_rotatory_bottom_left);
79          machinery_rotatory_top_right.childOf(machinery_rotatory_bottom_right);
80          machinery_short_belt_left.setOrientation(0, 90, 0);
81          machinery_short_belt_right.setOrientation(0, 90, 0);
82          machinery_tool_machine_tool.childOf(machinery_tool_machine_tower);
83      }
84
85      /**
86      * Returns a scenario component according to the key received.
87      * @param key which represents one of the machines in the scenario.
88      * @return the component correspondant to the key.
89      */
90      public HRL_Component getHRL_ScenarioComponet(String key){
91          switch (key) {
92              case "A":
93                  return machinery_short_belt_input;
94              case "B":
95                  return machinery_rotatory_top_left;
96              case "C":
97                  return machinery_short_belt_left;
98              case "D":
99                  return machinery_elevator_connection_left;
100             case "E":
101                 return machinery_long_belt;
102             case "F":
103                 return machinery_long_belt;
104             case "G":
105                 return machinery_long_belt;
106             case "H":
107                 return machinery_long_belt;
108             case "I":
109                 return machinery_rotatory_top_right;
110             case "J":
111                 return machinery_short_belt_right;
112             case "K":
113                 return machinery_elevator_connection_right;
114             case "L":
115                 return elevator_cabin;
116         }
117         return null;
118     }
119
120     /**
121     * Moves the tool machine up
122     */
123     public void toolUp(){
124         this.machinery_tool_machine_tool.setPosition(positions.tool_offset, positions.tool_up, 0);
125         this.toolState = "up";
126     }
127
128     /**
129     * Moves the tool machine down
130     */
131     public void toolDown(){
132         this.machinery_tool_machine_tool.setPosition(positions.tool_offset, positions.tool_down, 0);
133         this.toolState = "down";
134     }
135
136     /**
137     * It turns the machine rotatoryLeft and saves the current orientation.
138     * @param degrees orientation for the machine.
139     */
140     public void rotatoryLeft (int degrees){
141         this.machinery_rotatory_top_left.setOrientation(0, degrees, 0);
142         this.rotatoryLeftState = degrees;
143     }
144
```

```
145     /**
146     * It turns the machine rotatoryRight and saves the current orientation.
147     * @param degrees orientation for the machine.
148     */
149     public void rotatoryRight (int degrees){
150         this.machinery_rotatory_top_right.setOrientation(0, degrees, 0);
151         this.rotatoryRightState = degrees;
152     }
153 
154     /**
155     * Moves the elevator.
156     * @param row location in row.
157     * @param col location in column.
158     */
159     public void moveElevator(int row, int col){
160         if(row >= 0 && row <= 5 && col >= 0 && col<=9){
161             this.elevator_tower.model.setTranslation(positions.elevator_positions[col]);
162             elevatorCol = col;
163             this.elevator_cabin.model.setTranslation(positions.cabin_positions[row]);
164             elevatorRow = row;
165         } else {
166             System.out.println("Row values: 0-5");
167             System.out.println("Column values: 0-9");
168         }
169     }
170 }
```

2.4 Class HRL_Positions

```
5  package kodemat.hrl.client;
6
7  import kodemat.visudata.VisuVector3f;
8
9  /**
10  *
11  * @author marco
12  */
13 public class HRL_Positions {
14     //Mapped locations for the different cells of the rack.
15     public final float row0 = 2.7f;
16     public final float row1 = 4.9f;
17     public final float row2 = 7.2f;
18     public final float row3 = 9.4f;
19     public final float row4 = 11.7f;
20     public final float row5 = 2.1f;   //this row defines the height of the transportation machines.
21     public final float rotatory_machine_height = 1.1f;  //height for the rotatory machine (upper part).
22     public final float col0 = -11.15f;
23     public final float col1 = -8.65f;
24     public final float col2 = -6.2f;
25     public final float col3 = -3.17f;
26     public final float col4 = -1.25f;
27     public final float col5 = 1.25f;
28     public final float col6 = 3.17f;
29     public final float col7 = 6.2f;
30     public final float col8 = 8.65f;
31     public final float col9 = 11.15f;
32
33     //Mapped locations for the different positions of the rack.
34     public final VisuVector3f[][] rack_positions =  { {new VisuVector3f(0f, row0, col0),
35      new VisuVector3f(0f, row0, col1), new VisuVector3f(0f, row0, col2), new VisuVector3f(0f, row0, col3),
36      new VisuVector3f(0f, row0, col4), new VisuVector3f(0f, row0, col5), new VisuVector3f(0f, row0, col6),
37      new VisuVector3f(0f, row0, col7),new VisuVector3f(0f, row0, col8), new VisuVector3f(0f, row0, col9)},
38
39                                                 {new VisuVector3f(0f, row1, col0),
40      new VisuVector3f(0f, row1, col1), new VisuVector3f(0f, row1, col2), new VisuVector3f(0f, row1, col3),
41      new VisuVector3f(0f, row1, col4), new VisuVector3f(0f, row1, col5), new VisuVector3f(0f, row1, col6),
42      new VisuVector3f(0f, row1, col7),new VisuVector3f(0f, row1, col8), new VisuVector3f(0f, row1, col9)},
43
44                                                 {new VisuVector3f(0f, row2, col0),
45      new VisuVector3f(0f, row2, col1), new VisuVector3f(0f, row2, col2), new VisuVector3f(0f, row2, col3),
46      new VisuVector3f(0f, row2, col4), new VisuVector3f(0f, row2, col5), new VisuVector3f(0f, row2, col6),
47      new VisuVector3f(0f, row2, col7),new VisuVector3f(0f, row2, col8), new VisuVector3f(0f, row2, col9)},
48
49                                                 {new VisuVector3f(0f, row3, col0),
50      new VisuVector3f(0f, row3, col1), new VisuVector3f(0f, row3, col2), new VisuVector3f(0f, row3, col3),
51      new VisuVector3f(0f, row3, col4), new VisuVector3f(0f, row3, col5), new VisuVector3f(0f, row3, col6),
52      new VisuVector3f(0f, row3, col7),new VisuVector3f(0f, row3, col8), new VisuVector3f(0f, row3, col9)},
53
54                                                 {new VisuVector3f(0f, row4, col0),
55      new VisuVector3f(0f, row4, col1), new VisuVector3f(0f, row4, col2), new VisuVector3f(0f, row4, col3),
56      new VisuVector3f(0f, row4, col4), new VisuVector3f(0f, row4, col5), new VisuVector3f(0f, row4, col6),
57      new VisuVector3f(0f, row4, col7),new VisuVector3f(0f, row4, col8),new VisuVector3f(0f, row4, col9)}};
58
59     //Mapped locations for the different positions of the elevator.
60     public final float elevator_offset = 1.5f; //it compensates the difference between cabin and tower.
61     public final VisuVector3f[] elevator_positions = {new VisuVector3f(0f, 0f, col0+elevator_offset),
62         new VisuVector3f(0f, 0f, col1+elevator_offset), new VisuVector3f(0f, 0f, col2+elevator_offset),
63         new VisuVector3f(0f, 0f, col3+elevator_offset), new VisuVector3f(0f, 0f, col4+elevator_offset),
64         new VisuVector3f(0f, 0f, col5+elevator_offset), new VisuVector3f(0f, 0f, col6+elevator_offset),
65         new VisuVector3f(0f, 0f, col7+elevator_offset), new VisuVector3f(0f, 0f, col8+elevator_offset),
66         new VisuVector3f(0f, 0f, col9+elevator_offset)};
67
```

```
68      //Mapped locations for the different positions of cabin.
69      public final float cabin_offset_V = -0.8f; //it compensates the vertical difference cabin-tower.
70      public final float cabin_offset_H = -1.5f; //it compensates the vertical difference cabin-tower.
71      public final VisuVector3f[]cabin_positions={new VisuVector3f(0f, row0+cabin_offset_V, cabin_offset_H),
72          new VisuVector3f(0f, row1+cabin_offset_V, cabin_offset_H),
73          new VisuVector3f(0f, row2+cabin_offset_V, cabin_offset_H),
74          new VisuVector3f(0f, row3+cabin_offset_V, cabin_offset_H),
75          new VisuVector3f(0f, row4+cabin_offset_V, cabin_offset_H),
76          new VisuVector3f(0f, row5+cabin_offset_V, cabin_offset_H)};
77
78      //Mapped locations for the different positions of machinery_long_belt.
79      public final float sensor0 = -6.7f;
80      public final float sensor1 = -2.2f;
81      public final float sensor2 = 2.2f;
82      public final float sensor3 = 6.7f;
83      public final VisuVector3f[] machinery_long_belt_positions = {new VisuVector3f(0f, row5, sensor0),
84          new VisuVector3f(0f, row5, sensor1),new VisuVector3f(0f, row5, sensor2),
85          new VisuVector3f(0f, row5, sensor3)};
86
87      //Mapped locations for tool machine.
88      public final float tool_offset = -3.3f;
89      public final float tool_up = 4;
90      public final float tool_down = 2;
91  }
```

3 Simulink function “Visualization”

```
function fcn(A,B,C,D,E,F,G,H,I,J,K,L,   C0,C1,C2,C3,C4,C5,C6,C7,C8,C9,
       R0,R1,R2,R3,R4,R5,   tDown,rotatoryLeft,rotatoryRight)

%enables javaMethod function.
coder.extrinsic('javaMethod')

%variables and initial states.
persistent STARTED;
tool='up';

%it ensures this method is called only once.
if isempty(STARTED)
       javaMethod('start','kodemat.hrl.client.HRL_API');
       STARTED=1;
end

%it sends the state of the sensors of the elevator.
javaMethod('updateElevator','kodemat.hrl.client.HRL_API',
       [C0,C1,C2,C3,C4,C5,C6,C7,C8,C9],[R0,R1,R2,R3,R4,R5]);

%it sends the state of the sensors of the machines.
javaMethod('updateSensorMap','kodemat.hrl.client.HRL_API',[A,B,C,D,E,F,G,H,I,J,K,L]);

%it checks the state of the tool machine
if (tDown==1)
       tool= 'down';
end

%it sends the state of the moveable elements of the machines.
javaMethod('updateScenario','kodemat.hrl.client.HRL_API',tool,rotatoryLeft,rotatoryRight);

end
```