**Lebanese University**

**Faculty of Sciences**

**Fanar Campus**

# Wood Board Cutting Application

**by**

Elie Atamech (53528)

**Supervised by**

Dr. Ralph El Khoury

**July 2019**

**ABSTRACT**

*This Project was chosen mainly to gain experience with IOT, and learn an algorithm that can be used in all sorts of applications, wanted to learn new things other than working with databases, API, and websites as already did many projects that were purely based on programming.*

## Table of Contents

# 1. Introduction

We aim to create a software (could be a Mobile App, or Web App, or Desktop App) that mainly helps a carpenter to cut rectangular wood board into small pieces according to his needs with minimal waste of wood.

We chose to build a Desktop Application, as this type of software is generally easy to install and configure and it only demands a decent computer as it does not take a lot of computing power. Plus, with a simple and friendly user interface this application is designed to be easy to use for people with basic experience.

In addition to the software part, in this project we will build a CNC (Computer Numerical Control) machine, that takes the information from the application, and draw marks on a one by one piece of wood, which further helps the carpenter in the cutting process.

Both the application and the CNC machine will run on algorithms. An algorithm to figure out the cuts that waste the minimal amount of wood for the first one, and another algorithm to pilot the machine in order to draw on the wood board.

For the algorithm that figures out the cuts, we will use a packing algorithm, as packing problems have found many industrial applications specially in wood or glass industries, where rectangular components have to be cut from large sheets of material.

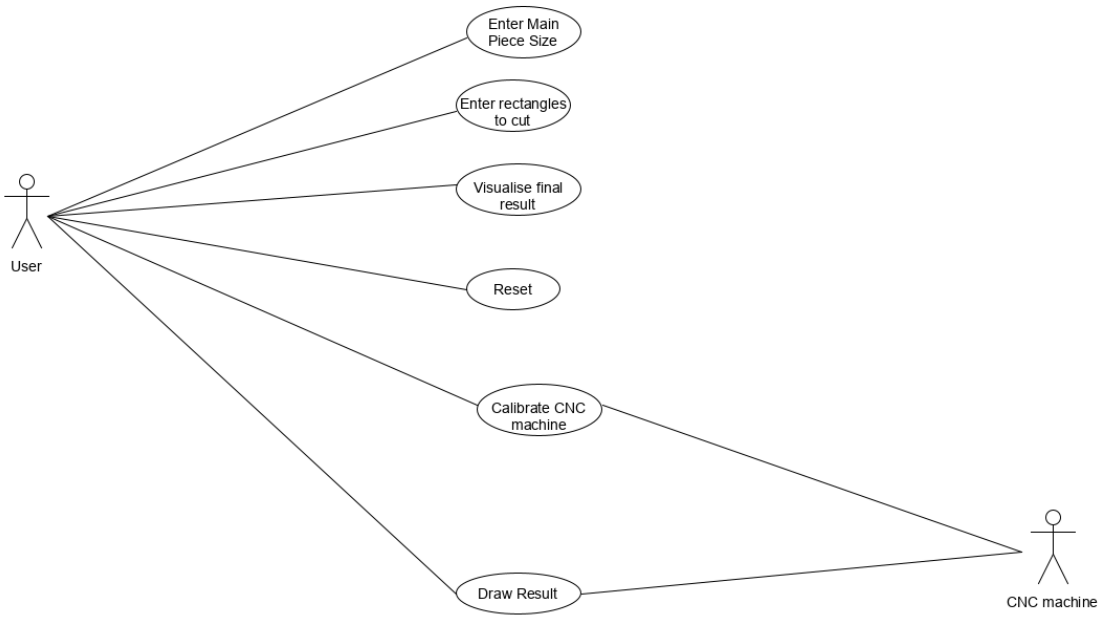
We can imagine that we have one big rectangular box, and we want to fit in it n smaller boxes. If we put the small boxes in the big one in a way that we made use of all the space possible and didn't waste any space, if we view them from the top, we can consider the big box as the big piece we want to cut, and the small boxes as the small pieces we want to have.

So, theses algorithm can give us a solution for our problem that is cutting with minimal waste of wood.

## 2. Desktop Application

This application is used as a user interface, to enter the main piece size (width, height), add the number of rectangles to cut and the size of each one.
After entering all the information, by clicking apply, we run the algorithm, see the result on a view panel, and finally we can click on CNC button to configure the machine, calibrate it and start drawing on the wood board.

### *2.1 Use Case*




**Enter Main Piece Size:**

The user must enter the size of the main wood board he wants to cut.

**Enter rectangles to cut:**

These are the rectangles pieces the user wants to cut from the main board.

He has to enter the size of each piece.

**Visualize result:**

Before drawing on the actual board, the user can visualize the drawing on a view panel.

**Reset:**

A button used to clear all reset all the data entered.

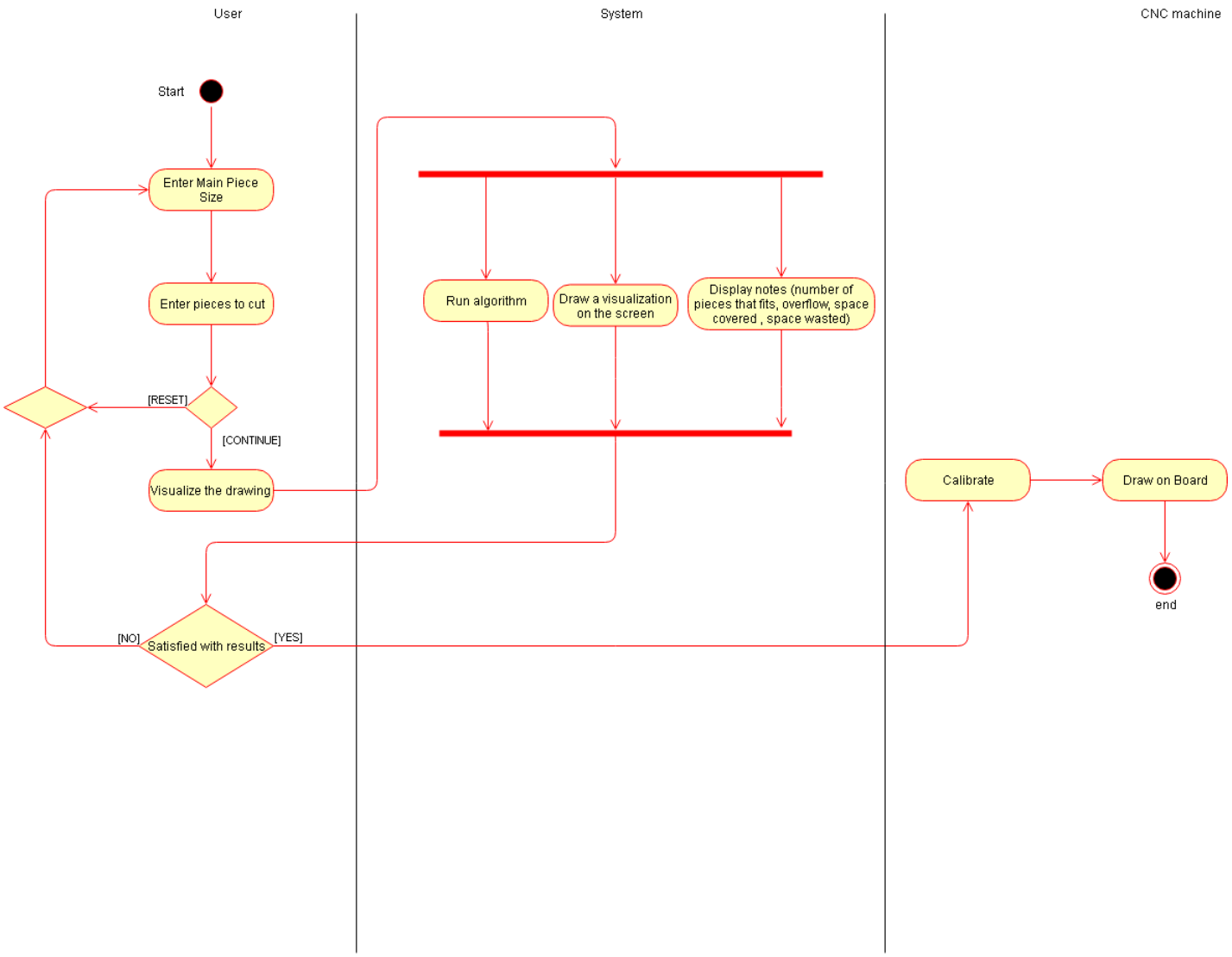
**Calibrate CNC machine:**

From the Application, the user can calibrate the CNC machine, and set its (0,0,0) position before drawing.

**Draw Result:**

After Testing the connection with the machine, can calibrating it, the user can click a "Draw" button on the application, and the CNC machine will start drawing.

## *2.2 Activity Diagram*



### *2.3 Implementation*

This application is created on the MVC design pattern that make it easy to update it in the future, like adding new option of shapes to cut rather than only rectangles (like triangle, circles, etc…), and implementing new algorithms.
Plus design patterns such as singleton were heavily used when writing the classes.

#### *2.3.1 Technical environment used*

This application was developped using java, in Eclipse Mars IDE.

The algorithm was originally written in C++, visualized with openGL, using CodeBlocks IDE.

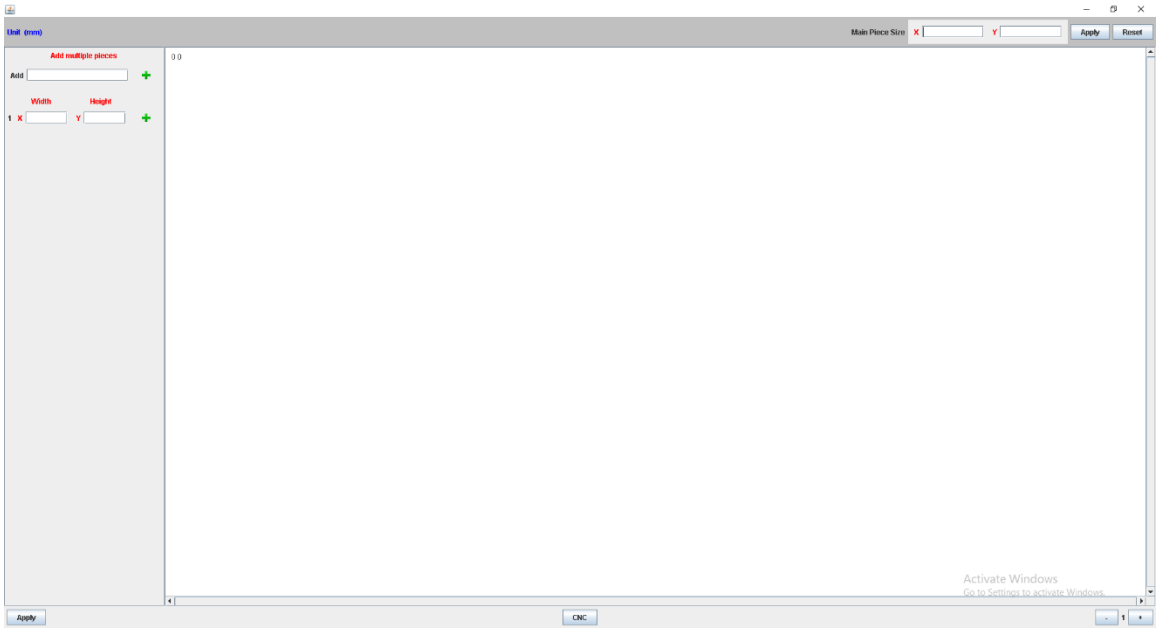
But later, I switched to java as it is more flexible for developing desktop application.

**The libraries used:**

- **Glut:** Used to visualize the result of the algorithm when written in C++
- **Java.awt, Javax.swing:** In order create the user interface
- **Processing.core, processing.serial:** To communicate with the microcontroller to pilot the CNC machine (discussed in details in the Communication section)

### *2.3.2 Screens description*

**1-**




This is the main window. This frame(jframe) is divided into four panels (jpanels) : TopPanel, BottomPanel, LeftPanel, RightPanel.

**Panels (View) classes used:**

Each panel is a different class, this makes it easy to modify each one and keeping a simple code.

Plus each one of them is a singleton class, which help a lot in object-oriented programming, where there are classes that don't need to exist in more than one instance.

**CoordPanel:**

Represent the whole : Main Piece Size X |_____| Y |_____|. This present in the TopPanel and RightPanel, and it is used to get the width and height of a piece.

**View Panel:**

This is the white panel in the middle, and it is where a visualization of the final result will be drawn.

**Top Panel:**

This is the panel at the top, where the user can enter the size of the main piece, and hit apply to draw the main piece, or click reset to restart the application

**Left Panel:**

This is where the user can enter as many pieces as he likes, with eacho one its width and height.

These are the pieces that he want to cut .

**Right Panel:**

This is where the result of the algorithm used to figure out the position of the pieces to cut will appear.

**Bottom Panel:**

This is the panel that the user can hit the applyB to draw the visualization on the ViewPanel, use to zoom in and out of this visualization, or go to the CNC window to configure the CNC machine and start drawing.

**Control Classes:**

**Settings:**
This singleton class is used to hold global variables used all over the application:
Float vpScaleFactor : used ehn drawing to the View Panel (explained in Screen description)
Zoomfactor : used when zooming in and out of the view panel. We simply multiply the position coordinates,, the width and the height of every rectangle by this value when the user change it.
Unit: The unit used (mm)

**Point:**
This is used to represent a (X,Y) coordinate:
Int x
Int y

**Square:**
This is used to holds information about a square:
Point pos
Int width
Int height
Int air

**HRalgorithm:**
This is the class where the algorithm is implemented (the algorithm is explained later in this report)

**ScaledImageIcon:**
This class is used to get import an image, scale it, and return it as an icon

**AddButton**

This class is used to from the green add buttons present in the top and left panel.

It uses the ScaledImageIcon class mentioned above, the get the Add.png image.

**EventSubscriber:**

This class is used to make the communication between classes easier.

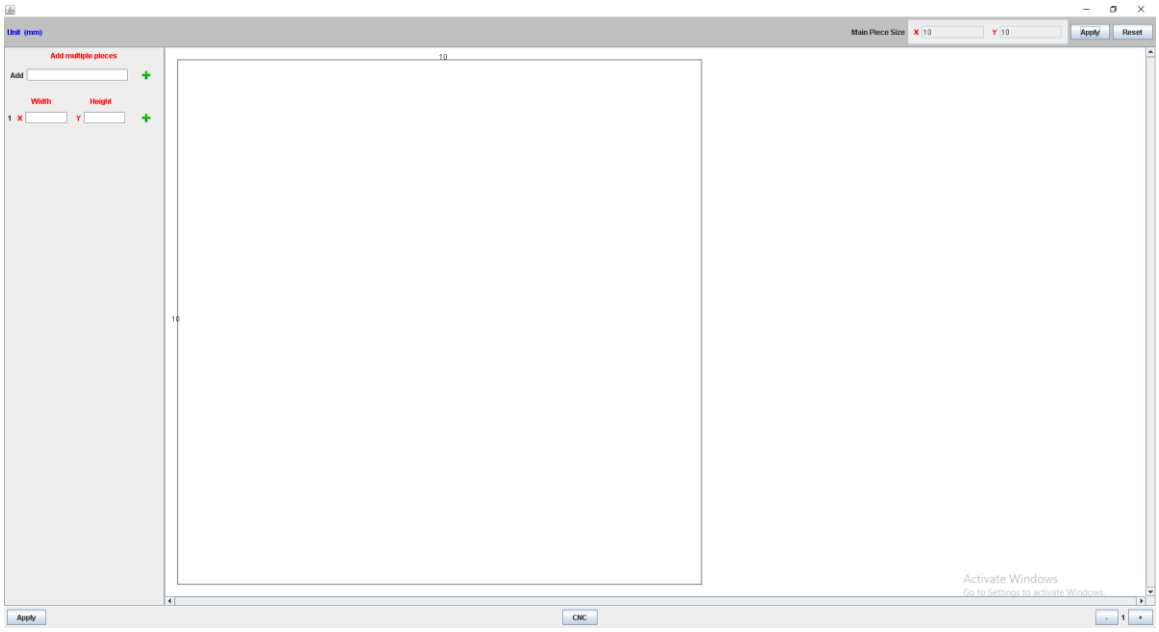
A class can create an event.

Other classes can subscribe to that event. When they do so, they must specify a function of their own that will run when that event start.

Finally, when that event start, every subscriber will execute the function that he specified.

To make this class the java Introspection and Reflection libraries were used.

**2-**




When the user enter the size of the main piece and hit apply, the main piece will be drawn in the ViewPanel.

Whatever the size of the main piece is, the longest side of it will be changed so it fit on the smallest side of the View Panel

Example: If:

(Vx,Vy) = (500,1000) with Vx : width of View Panel and Vy : height of the View Panel

(Px,Py) = (700,1500) with Px :width of main piece, and Py : height of the main piece

minV = max (Vx,Vy) = 500;

maxP = max(Px,Py) = 1500 ;

Settings.vpScaleFactor = minV/maxP = 500/1500 = 0.33

So now when draing the main piece we multiply its width and height by Settings.vpScaleFactor and we get :

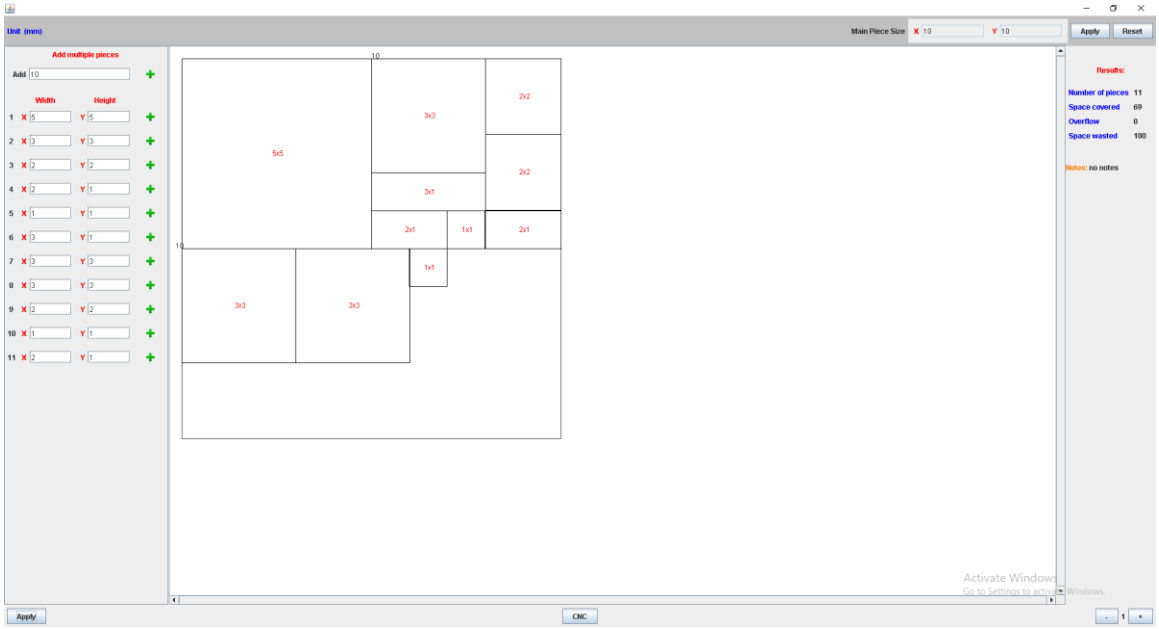
width_used+to_draw = Px * 0.33 = 231

height_used_to_draw = Py * 0.33 = 459

This way we never exceed the view panel margins.

And from now on, when drawing anuthing on the View Panel (the pieces to cut), we multiply its position coordinates, and its width and height by Settings.vpScalFactor, in order to keep the same distances and scale.

**3-**

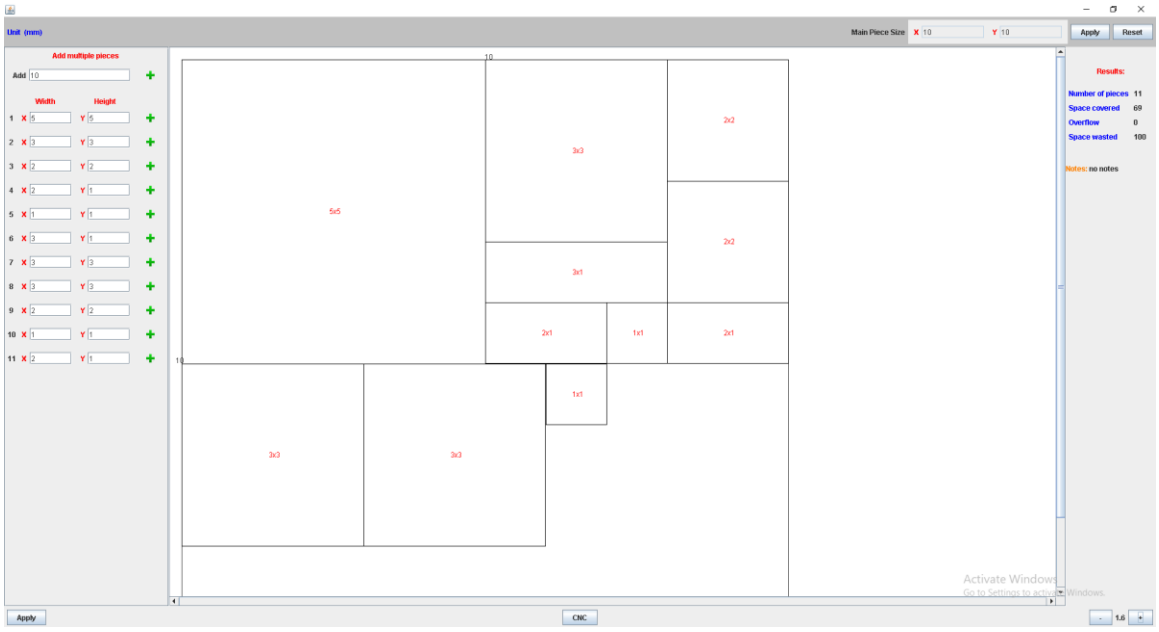



After the user enter all the pieces that he wants to cut, when he hit the Apply button on the bottom panel, the algorithm will run and a visualization of the pieces is drawn on the view panel. Written inside every rectangle, its width and height (width x height).

Also, the user can see the result of the algorithm in the right panel:

- Number of pieces
- Space wasted
- Overflow
- Space covered
- Note

**4-**




The user can zoom in and out of the view panel, by clicking on the plus or minus buttons on the bottom right of the window.

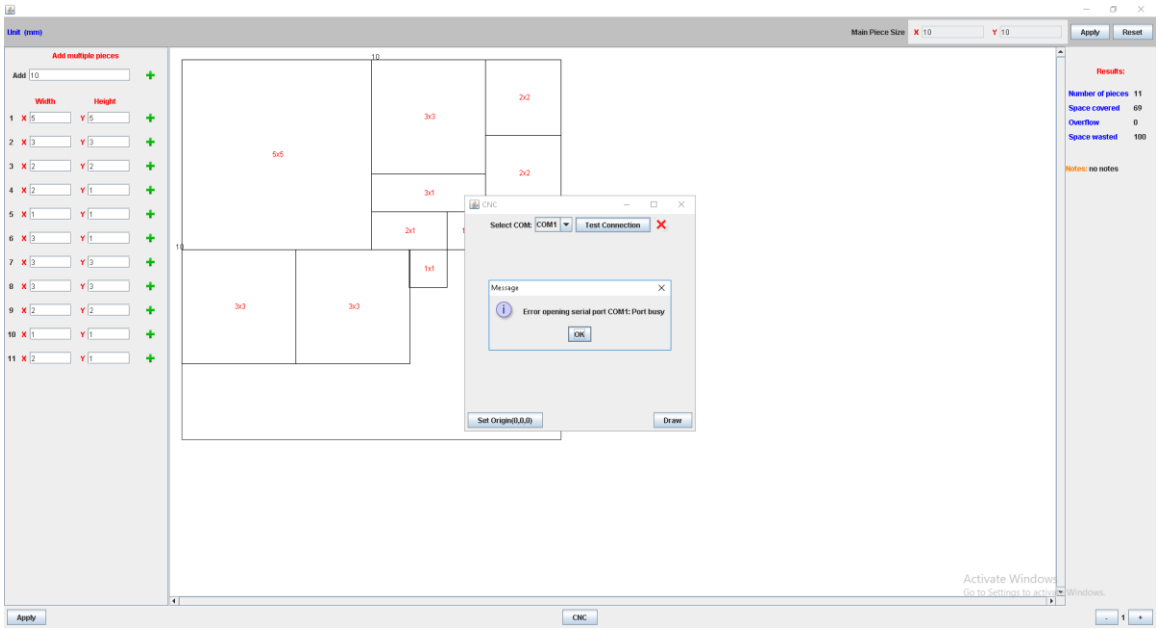
Zooming is mostly used when the main piece is big, and some pieces to cut are small.

When zooming in the Settings.zoomFactor variable will increase, and when zooming out it will decrease. And it will be originally 1.

Finally when drawing we alse multiply the po, width and height of everything by Settings.zeoomFactor

**5-**


Unit (mm)
Main Piece Size
Apply
Reset
Add multiple pieces
Width
Height
Results:
Number of pieces 11
Space covered 69
Overflow 0
Space wasted 100
Notes: no notes
CNC
Select COM: COM1
Test Connection
Set Origin(0,0,0)
Draw
Activate Windows
Apply
CNC


Unit (mm)
Main Piece Size
Apply
Reset
Add multiple pieces
Width
Height
Results:
Number of pieces 11
Space covered 69
Overflow 0
Space wasted 100
Notes: no notes
CNC
Select COM: COM1
Test Connection
Message
Error opening serial port COM1: Port busy
OK
Set Origin(0,0,0)
Draw
Activate Windows
Apply
CNC


When the user clicks on the “CNC” button in the bottom panel, a new window will appear.

This window is used to connect to the correct port that is connected to the microcontroller (Arduino) of the CNC machine, calibrate the CNC machine, setting its Origin (zero position), and start drawing.

When the COM port is correct and the connection to the CNC machine is established, the user can calibrate the machine by clicking on the arrows, which will make the machine move like he wants. Then he can set the Origin, and finally he can draw on the board by clicking the Draw button.

## View Classes:

### CNCWindow:

This is the View class of the CNC window

### Arduino:

This class is used to communicate between the application and the microcontroller that pilot the CNC machine.

The library processing is used

(Discussed in details later in this report)

**UpArrowButton - DownArrowButton – LeftArrowButton – RightArrowButton:**

Theses classes are used to form the arrows.

**FalseLabel – CorrectLabel:**

Used to create the correct icon and the false icon used to indicate if the COM port is correct.

# 3. Algorithm

The algorithm used int this application is based on heuristic strategies and a recursive structure. We will call it HR in this report.
The term heuristic is used for algorithms which finds a solution but does not guarantee that this is the best one. Sometimes they can find the best solution, but they are still called heuristic until this solution is proven to be the best.

## *3.1 Heuristic recursive (HR) algorithm*

This algorithm was taken from the article by Defu Zhang,Yan Kang, Ansheng Deng, published online on 26 February 2005.

Reference : https://www.inf.utfsm.cl/~mcriff/SP/articulo4.pdf

**Algorithm design:**

This algorithm follows a divide-and-conquer approach.

Divide-and-conquer algorithm follow three steps:

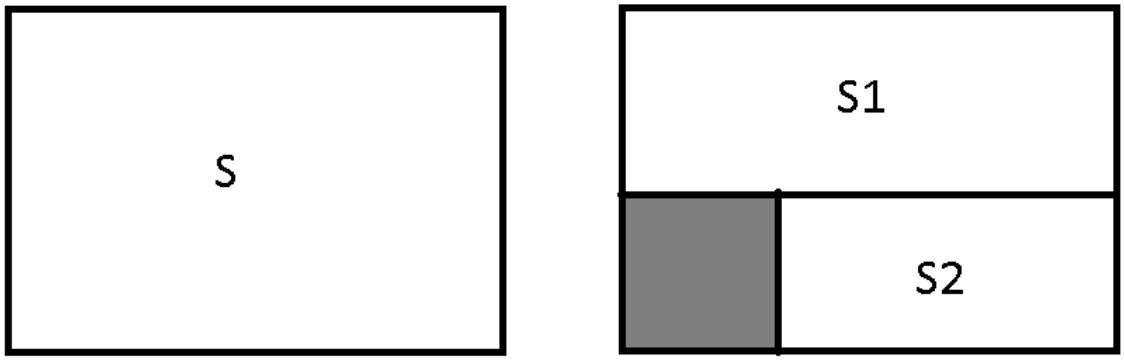
1- Divide the original problem into smaller subproblems that don't overlap.
2- Solve each subproblem recursively.
3- Combine to solutions of the subproblems to form the solution for the original problem

**Algorithm Explanation:**

1) Pack a rectangle in a space that it fits in, then divide this space into subspaces.

2) Pack each subspace recursively.

3) Combine the solutions to the subproblems (subspaces) to form the solution of the big problem (the rectangular wood piece).




S is the original space (original problem) what we want to pack.

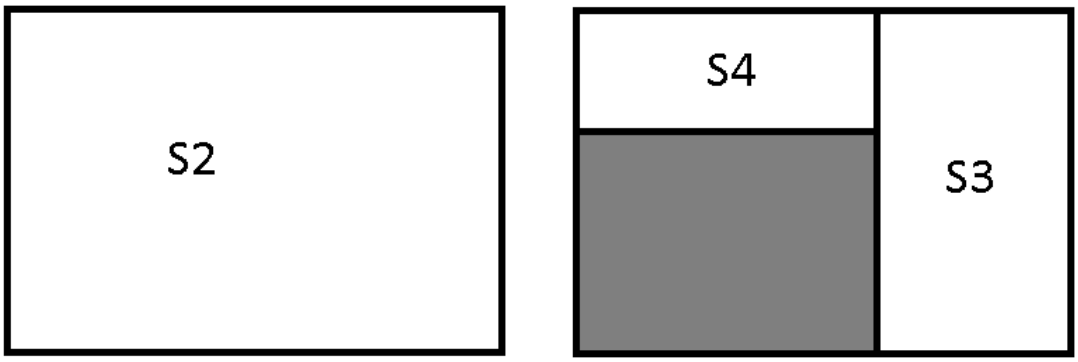
S(i) are the subspaces (subproblems) we got from dividing S, and S(i-1)

There are 2 type of spaces:

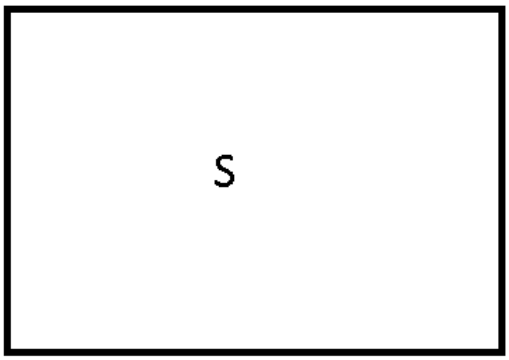
Bounded space: a space which we fixed its height to a certain value, like S2.

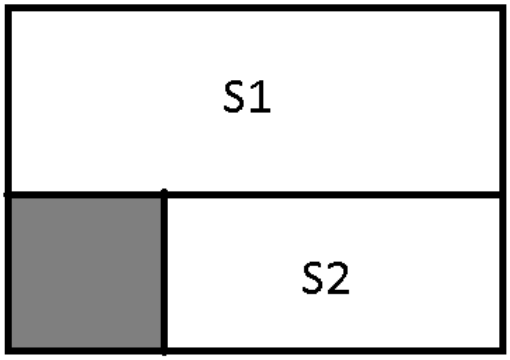
Unbounded space: a space which we didn't fix its height. So, reaching the top of the original space S with its height, like S1.

How we divide the space:




a) If the space S2 is bounded: we pack a rectangle in it, then divide it to 2 subspaces: S3 and S4 which are both bounded, then we recursively pack S3 and S4 starting with the one with the bigger area.




b) If the space S is unbounded, we pack a rectangle in it, and divide into 2 subspaces: S1 unbounded, and S2 bounded, then we recursively pack S2. We finally start packing S1, when there are no rectangles that can fit in S2 anymore.

So, we have 2 functions:

1- **BoundedPacking**: Used to pack bounded spaces.
It passes on all every rectangle once so its time complexity is: O(n).

```
BoundedPacking(bounded subspace S2)
If (no rectangle that fits in S2 found) return
Else {
    Select a rectangle R, pack it in S2
    Divide the unpacked space of S2 into 2 subspaces S3 and S4
    If (area(S3) > area(S4)) {
        BoundedPacking(S3)
        BoundedPacking(S4)
    }else {
        BoundedPacking(S4)
        BoundedPacking(S3)
    }
}
```

We should pack the rectangles that have the bigger area first, so a space will be left for smaller rectangles in the future.
So we sort the rectangles in decreasing order of their area.

2- **UnboundedPacking:** Used to pack unbounded spaces.
Also uses the BoundedPacking function to recursively pack the bounded subspaces that generates from dividing the unbounded subspaces.
And it will also be the function that we call to get the final result, as it enters a loop, packs the unbounded spaces, generates bounded subspaces, then recursively call BoundedPacking on these bounded subspaces, then calls itself again on the unbounded subspaces left until there are no rectangles left to be packed.

UnboundedPacking (unbounded space S)

If there are no unpacked rectangles left then return

Else {

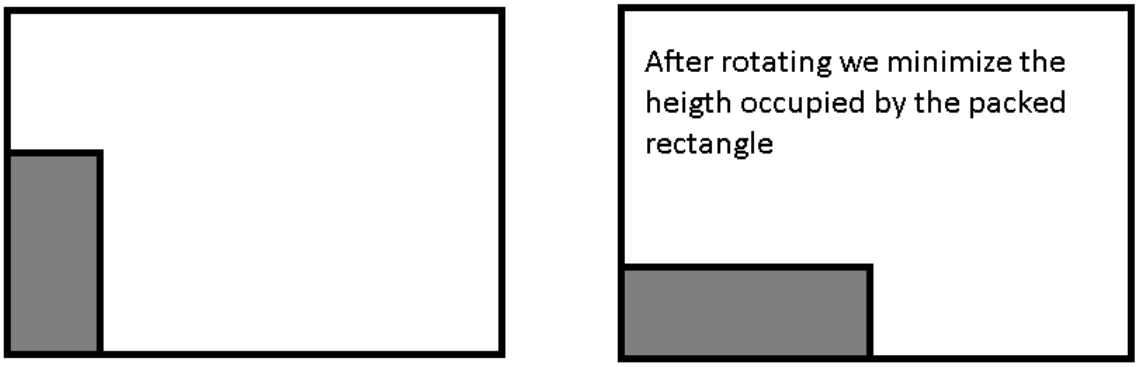
Select a rectangle that fits in S and pack it in S

Divide the unpacked space into unbounded subspace S1 and bounded subspace S2

BoundedPacking (S2)
UnboundedPacking (S1)

}

When we select a rectangle that fits in S, we rotate it so its longer side is lying on the bottom of the space then we see if it fits, this is so we reach a minimal height at the end.




Now we can call the pack function to run the algorithm.

### *3.2 Implementation*

## Square class:

Used to represent a square

```
3  import java.awt.*;
4
5  public class Square implements Comparable<Square> {
6
7      Point pos = new Point () ;
8
9      int width , height;
10     int air ;
11
12     public Square(int width, int height) {
13         this.width = width;
14         this.height = height;
15
16         air = width * height ;
17     }
18
19     public Square(Point pos , int width, int height) {
20         this.pos = pos;
21         this.width = width;
22         this.height = height;
23
24         air = width * height ;
25     }
26
27     public void SetPos ( int x , int y ){
28         pos.x = x ;
29         pos.y = y ;
30     }
31
32     public void Rotate90 () {
33         int temp = width ;
34         width = height ;
35         height = temp ;
36     }
37
38     public void Draw (Graphics g , Point pieceTopLeft){
39
40         int sx = Math.round(width * Settings.getSettings().vpScaleFactor * Settings.getSettings().zoomFactor) ;
41         int sy = Math.round(height * Settings.getSettings().vpScaleFactor * Settings.getSettings().zoomFactor);
42
43         // pos.x will be the shelf current width
44         int x = pieceTopLeft.x + Math.round(pos.x * Settings.getSettings().vpScaleFactor * Settings.getSettings().zoomFactor) ;
45         int y = pieceTopLeft.y + Math.round(pos.y * Settings.getSettings().vpScaleFactor * Settings.getSettings().zoomFactor) ;
46
47
48         g.setColor(Color.BLACK);
49         g.drawRect(x,y,sx,sy);
50
51         g.setColor(Color.RED);
52         g.drawString(width+"x"+height, x + Math.round(sx/2.0f) - 7 , y +Math.round(sy/2.0f) + 4 );
53     }
54
55     @Override
56     public int compareTo(Square x) {
57         if ( this.air > x.air ){
58             return 1 ;
59         }
60
61         return -1 ;
62     }
63
64 }
```

Variables:

Point pos: Point is a class that holds 2 variables: x, and y. It is used in this application to represent a position with coordinates x and y.

Integer air: so we can sort the rectangles in decreasing order of there area.

Functions:

Constructors.

Rotate90: this function is used to rotate the square: used when packing a rectangle in a space in order to reach minimal height like discussed earlier in the algorithm explanation section.

SetPos: setter class for the pos variable

Draw: Used to draw the square.

**HRalgorithm class:**

The class that holds and run the algorithm

```
import java.awt.Graphics;

class Space {
    Point pos ;
    int width ;
    int height ;

    int area ;

    boolean bounded ;

    ArrayList <Square> squares = new ArrayList <Square> ();

    public Space(Point pos, int width, int height,boolean _bounded) {
        this.pos = pos;
        this.width = width;
        this.height = height;
        this.bounded = _bounded;

        area = width * height ;
    }

    boolean canAdd (Square x){

        for ( int i = 0 ; i < 2 ; i ++){
            if ( x.width <= width && x.height <= height ){
                return true ;
            }

            x.Rotate90();

        }

        return false ;
    }

    public void Add (Square s){
        s.pos.x = pos.x;
        s.pos.y = pos.y;

        //this.squares.add(s);

        this.bounded = true ;
    }
```

```
49
50 }
51
52 public class HRalgorithm {
53
54     Space initialSpace ;
55
56     Point topLeft ;
57
58     int spaceCovered = 0;
59     int spaceWasted  ;
60     int totalNbPieces = 0;
61     int nbAddedPieces = 0;
62     int overflow = 0;
63
64     String notes ;
65
66     ArrayList <Space> shelfs = new ArrayList <Space> ();
67     ArrayList <Square> squares = new ArrayList <Square> ();
68
69     ArrayList <Square> newSquares = new ArrayList <Square> () ;
70
71     private Comparator <Square> comp = new Comparator<Square> () {
72         @Override
73         public int compare(Square a, Square b) {
74             if ( a.air < b.air ){
75                 return 1 ;
76             }
77             else return -1 ;
78         }
79     };
80
81     public HRalgorithm (Point _topLeft ,int width , int height){
82         this.topLeft = _topLeft;
83         initialSpace = new Space (new Point (0,0),width,height,false);
84
85         spaceWasted = width*height;
86
87         notes = "no notes";
88     }
89
90     public void setSquares (ArrayList <Square> _s){
91         squares = _s;
92         squares.sort(comp);
93
94         this.totalNbPieces = squares.size();
```

```
95      }
96
97      public void Run () {
98          UnboundedPacking(initialSpace);
99          this.overflow = this.totalNbPieces - this.nbAddedPieces;
100         if (overflow > 0){
101             this.notes = "There are "+overflow+" piece(s) not added, change the main piece width or height";
102         }else {
103             this.notes = "no notes";
104         }
105
106         RightPanel.rightPanel().UpdateResults("Heuristic Algorithm", nbAddedPieces, spaceCovered, overflow, spaceWasted, notes);
107     }
108
109     private void BoundedPacking (Space s2){
110
111         Square pickedSquare = null ;
112         int indexPickedSquare = 0;
113
114         for ( int i = 0 ; i < squares.size() ; i ++){
115             Square currentSquare = squares.get(i);
116
117             // Can add to Space s2
118             if (s2.canAdd(currentSquare)){
119                 System.out.println("R CA");
120                 pickedSquare = currentSquare ;
121                 indexPickedSquare = i;
122                 break ;
123             }
124
125         }
126
127         if (pickedSquare == null ){
128             // No square that fit was found
129             return ;
130         }else {
131             s2.Add(pickedSquare);
132             squares.remove(indexPickedSquare);
133
134             this.newSquares.add(pickedSquare);
135
136             this.nbAddedPieces ++;
137             this.spaceCovered += pickedSquare.air;
138
```

```
139             Point s3Pos = new Point (s2.pos.x,s2.pos.y+pickedSquare.height);
140             int s3Width = pickedSquare.width;
141             int s3Height = s2.height - pickedSquare.height;
142 
143             Point s4Pos = new Point (s2.pos.x + pickedSquare.width , s2.pos.y);
144             int s4Width = s2.width - pickedSquare.width;
145             int s4Height = s2.height;
146 
147             Space s3 = new Space (s3Pos , s3Width , s3Height , true);
148             Space s4 = new Space (s4Pos , s4Width , s4Height , true);
149 
150 
151             if (s3.area > s4.area) {
152                 BoundedPacking (s3);
153                 BoundedPacking (s4);
154             }else {
155                 BoundedPacking (s4);
156                 BoundedPacking (s3);
157             }
158         }
159 
160     }
161 
162     private void UnboundedPacking (Space s) {
163         if (squares.size() == 0) return ;
164             System.out.println("Size "+ + squares.size());
165             Square pickedSquare = null ;
166             int pickedIndex = 0 ;
167 
168             for ( int i = 0 ; i < squares.size() ; i ++ ){
169                 Square currentSquare = squares.get(i);
170 
171                 if ( currentSquare.width < currentSquare.height){
172                     //heuristic packing strategy --> pack it flat
173                     currentSquare.Rotate90();
174                 }
175                 if ( s.canAdd(currentSquare) ){
176                     System.out.println("P CA " + currentSquare.width);
177                     pickedSquare = currentSquare ;
178                     pickedIndex = i;
179                     break;
180                 }
181 
182 
183             }
184 
```

```
185             if (pickedSquare == null){
186                 // No square that fits was found
187                 return ;
188             }else {
189 
190                 s.Add(pickedSquare);
191                 squares.remove(pickedIndex);
192 
193                 this.newSquares.add(pickedSquare);
194 
195                 this.nbAddedPieces ++;
196                 this.spaceCovered += pickedSquare.air;
197 
198                 Point s1Pos = new Point (s.pos.x , s.pos.y + pickedSquare.height);
199                 int s1Width = s.width;
200                 int s1Height = s.height - pickedSquare.height;
201 
202                 Point s2Pos = new Point (s.pos.x + pickedSquare.width , s.pos.y);
203                 int s2Width = s.width - pickedSquare.width;
204                 int s2Height = pickedSquare.height;
205 
206                 Space s1 = new Space (s1Pos,s1Width,s1Height,false);
207                 Space s2 = new Space (s2Pos,s2Width,s2Height,true);
208 
209                 s = s1 ;
210 
211                 BoundedPacking (s2);
212 
213                 UnboundedPacking(s);
214 
215             }
216 
217 
218     }
219 
220     public ArrayList <Square> getNewSquares () {
221         return this.newSquares;
222     }
223 
224     public void Draw (Graphics g ){
225         for ( int i = 0 ; i < newSquares.size() ; i ++ ){
226             newSquares.get(i).Draw(g,topLeft);
227         }
228     }
229 
230 }
```

**Class Space:**

The class Space is used to represent a bounded or unbounded space.

Variables

It holds the variables pos, width, height, area.

The bool bounded and the ArrayList of squares are not used. They were just for testing.

Functions:

Constructor

boolean canAdd (Square x): we rotate the square x and see if there is a way it can fit in this space, otherwise we return false.

void Add (Square x): we change the square's x pos.x and pos.y to be the same as this space's pos.x and pos.y (We add the square to the space).

**Class HRalgorithm:**

This class is used to run the HR algorithm.

Variables:

It holds a space called initialSpace, 2 lists of squares: squares and newSquares.

The initialSpace is the original unbounded space that we want to pack. It represents the main wood piece that we want to cut and has the same width and height of that piece.

The other variables are used to display notes and for drawing the result.

The list called shelfs is not used, and was only for testing.

Functions:

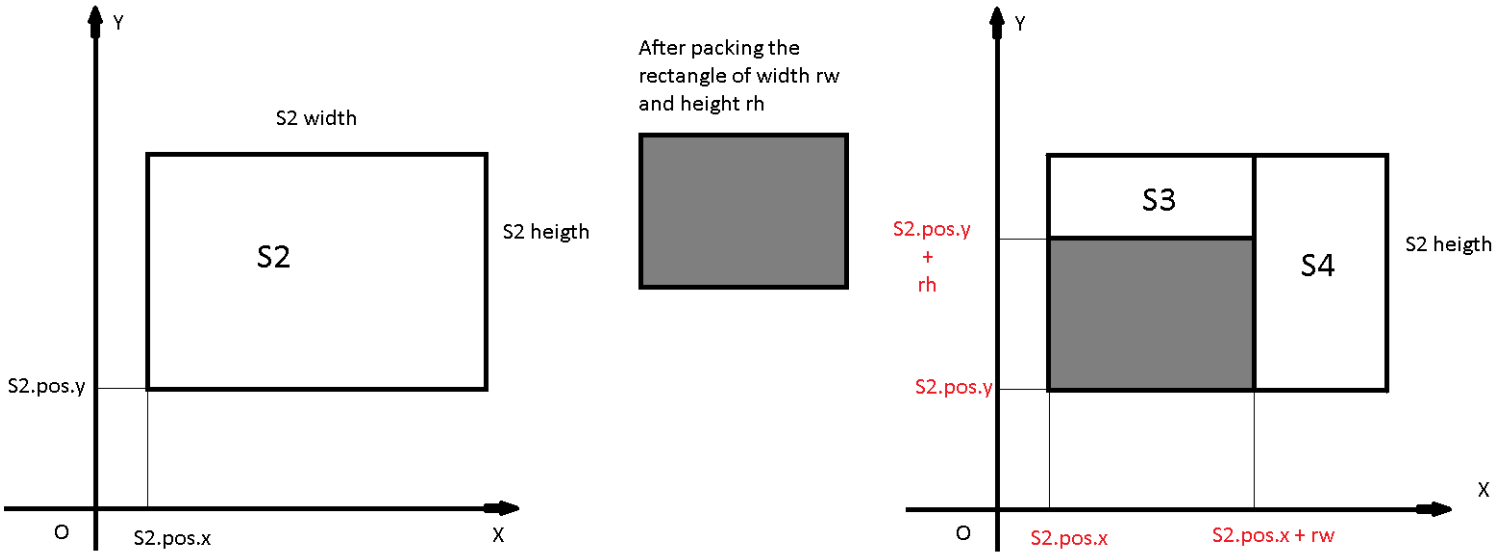
Constructor

Comparator <Square> comp: is used so we can sort the squares in decreasing order of their area.

Void SetSquares (List of squares _s): This is function is called from the bottom panel class when the user hits Apply. The bottom panel class takes the widths and heights that the user entered in the left panel, and create a square from each combination of width and height.

Finally the bottom panel class call this function to pass the created squares to the HRalgorithm class.

Void BoundedPacking: Used to pack a bounded space. Explained earlier in the algorithm explanation section.From line 139 to line 145 is how we divide a space into subspaces: we just create 2 spaces (subspace) and set their position, width, and height relative to the space S2 to match the figure below:



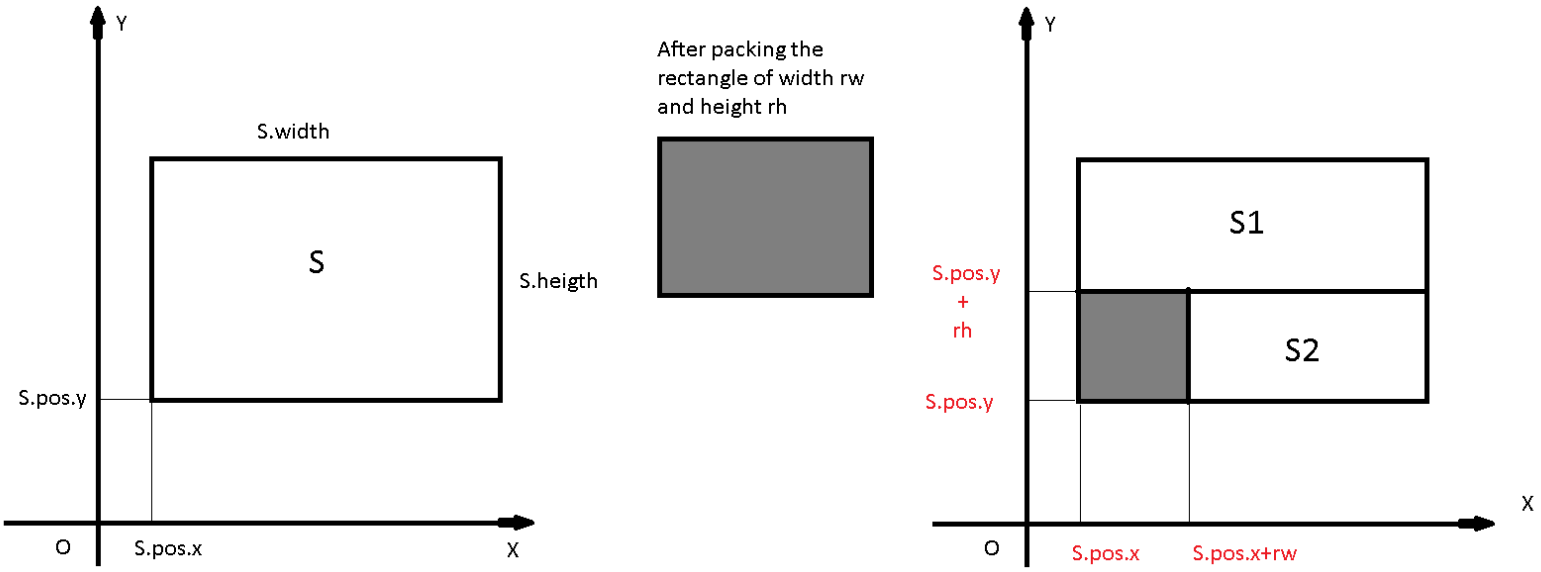

S3.pos.x = S2.pos.x
S3.pos.y = S2.pos.y + rh

S3.width = rw
S3.heigth = S2.heigth - rh

S4.pos.x = S2.pos.x + rw
S4.pos.y = S2.pos.y

S4.width = S2.width - rw
S4.height = S2.heigth

Voic UnboundedPacking: Used to pack an unbounded space. Explained earlier in the algorithm explanation section. From line 198 to 204 we divide and create the subspaces to match the figure below:




S2.pos.x = S.pos.x + rw
S2.pos.y = S.pos.y

S2.width = S.width - rw
S2.heigth = rh

S1.pos.x = S.pos.x
S1.pos.y = S.pos.y + rh

S1.width = S.width
S1.heigth = S.heigth - rh

In the BoundedPacking and UnboundedPacking functions, we pack a square by changing its pos, width, and height, and then we add it to the list of squares called newSquares that represent all the packed squares.

Void getNewSquares : to get the squares from the list newSquares.

Void Draw: Used to draw all the squares in the list newSquares (the packed squares)

Void Run: used to run the algorithm on the inbounded initial space.

### *3.3 Complexity*

With n being the number of the rectangles, we want to pack, the complexity of this algorithm is: O(n), because we pass once on each rectangle and we pack it.

The figure below taken from the article, is a comparison between the average running time in multiple test cases (C1, C2, C3, C4, C5, C6, C7) of this algorithm and GA+BLF and SA+BLF, which are two of the best packing algorithms:

| Problem category | Number of items: *n* | Optimal height | Object dimensions |
|---|---|---|---|
| C1 (C11, C12, C13) | 16 (C11, C13), 17 (C12) | 20 | 20 × 20 |
| C2 (C21, C22, C23) | 25 (C21, C22, C23) | 15 | 15 × 40 |
| C3 (C31, C32, C33) | 28 (C31, C33), 29 (C32) | 30 | 30 × 60 |
| C4 (C41, C42, C43) | 49 (C41, C42, C43) | 60 | 60 × 60 |
| C5 (C51, C52, C53) | 73 (C51, C52, C53) | 90 | 90 × 60 |
| C6 (C61, C62, C63) | 97 (C61, C62, C63) | 120 | 120 × 80 |
| C7 (C71, C72, C73) | 196 (C71, C73), 197 (C72) | 240 | 240 × 160 |

Average running time of GA+BLF, SA+BLF and HR (s)

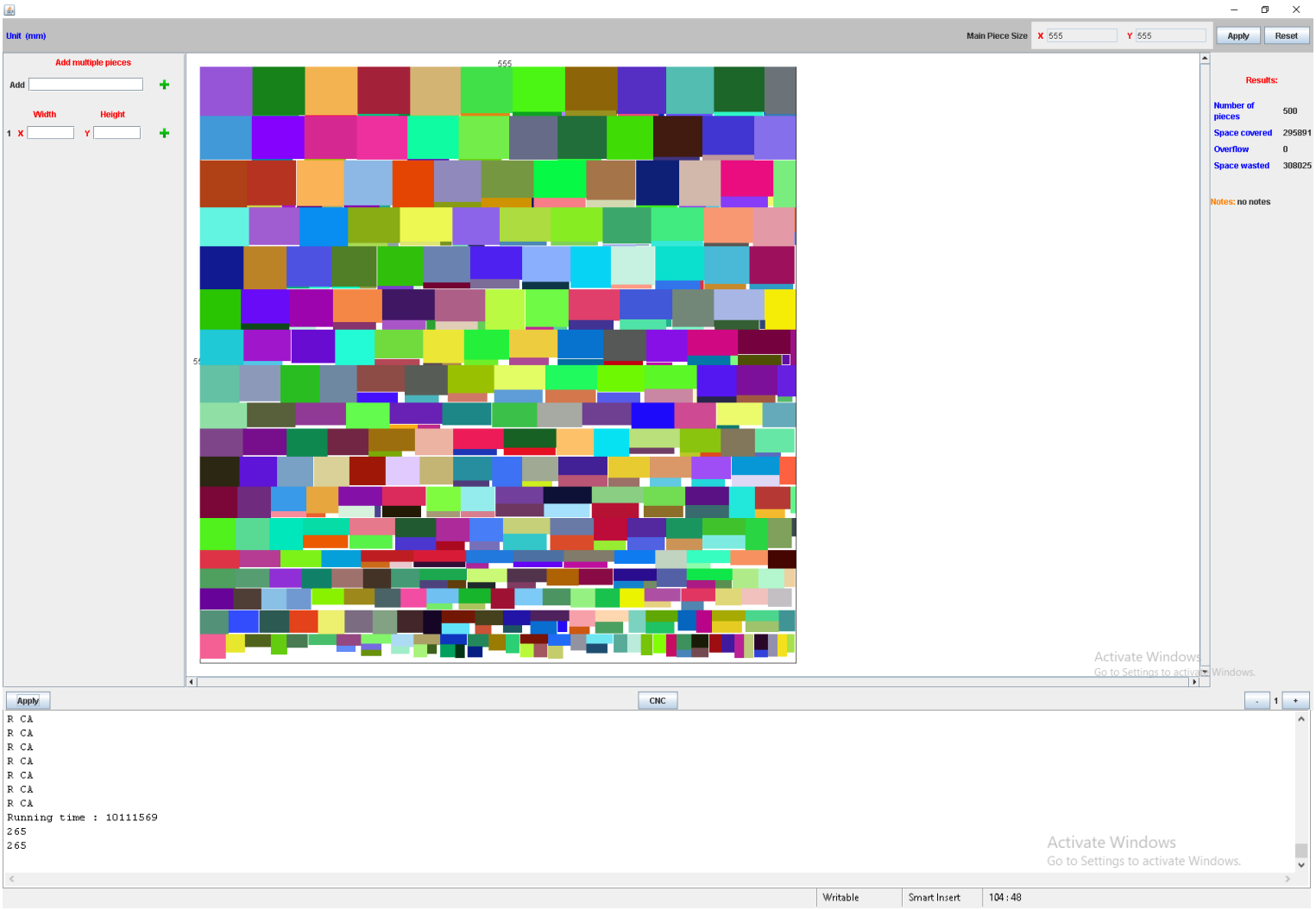
| | C1 | C2 | C3 | C4 | C5 | C6 | C7 | Average |
|---|---|---|---|---|---|---|---|---|
| GA+BLF | 4.61 | 9.22 | 13.83 | 59.93 | 165.96 | 396.46 | 3581.97 | 604.57 |
| SA+BLF | 3.227 | 11.064 | 18.44 | 152.13 | 530.15 | 1761.02 | 19274.41 | 3107.2 |
| HR | 0 | 0 | 0.03 | 0.14 | 0.69 | 2.21 | 36.07 | 5.59 |

Reference: https://www.inf.utfsm.cl/~mcriff/SP/articulo4.pdf

The picture below is the result of this algorithm running in this application after implementation. I tried to pack 500 pieces of wood of random width and height between 1 and 50 in a main piece which width and height are 555.

I changed the code so the squares are generated and not entered by the user, and to print in the console the time in nanoseconds which this algorithm took to finish.

Plus, every square is drawn with a random color (not white)




The algorithm took 10111569 nanoseconds = 10.111569 milliseconds

And overflow = 0, which means all the rectangles were packed

### *3.4 Conclusion*

This algorithm is very simple, and can solve the rectangular packing problem fast. It has a low complexity, and can solve the problem even for a very large input.
Plus, most of the times it outperforms two of the best packing algorithms when given a large input.

# 4. CNC Machine

## *4.1 Definition*

A Computer Numerical Control (CNC) at the most is nothing more than a computer-controlled machine.

When we talk about CNC machines, we refer to machines used in the industrial manufacturing world, which build the things we use every day, like packing machines used in food industries, wood cutters, etc…

These machines come in a variety of shapes and sizes. The high-end ones are very heavy and expensive. However, for small projects like this one where I need to build a DIY CNC machine, parts can be found at a low price in local electronic shops like: EKT Katrangi, which is the shop used to get all the parts used to build the machines.

## *4.2 Introduction*

In order to help the carpenter, cut the pieces of wood, this machine will be built to draw on that piece the outline where he should cut.

The build of this CNC machine is split into three parts:

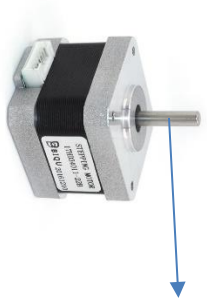
- **Design**
- **Build**
- **Programming**

Let's first understand the basics of what is needed to be known before we start building the machine.

## *4.3 Basics*

### 1- <u>Movement and electrical motors:</u>

An electrical motor is an electrical machine that when supplied with a certain voltage and current, converts the electrical energy into mechanical energy by rotating its shaft, which generate a force that we can use to apply on certain objects in order to move them.




Reference:
https://www.google.com/search?biw=1920&bih=969&tbm=isch&sa=1&ei=6Tg1XbDcAYKFmwWa5piACA&q=nema+17&oq=nema+17&gs_l=img.3..0i67j0l4j0i67j0l4.82271.82979..83615...0.0..0.166.900.0j7.....0....1..gws-wiz-img.WNtHubnAIrc&ved=0ahUKEwjw9Ibd1cfjAhWCwqYKHRozBoAQ4dUDCAY&uact=5#imgrc=bNru25pLnrSH_M:

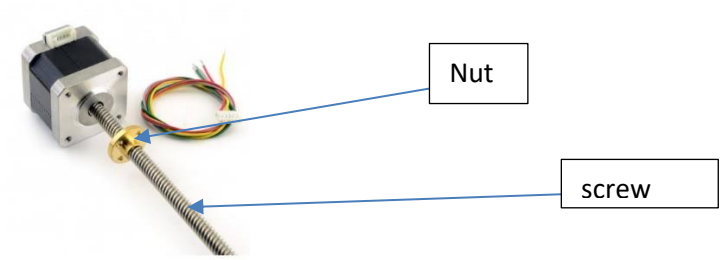
**There are 2 types of movement:**

**Rotary motion:** is turning around in circle, such as a wheel turning.

**Linear motion:** is moving in a straight line.

Originally, a motor will give us a rotary motion. We can transform this rotary motion to a linear motion in several ways, two of them are by using :

- Lead Screw:




Reference: https://www.diyelectronics.co.za/store/stepper-motors/481-nema-17-stepper-motor-with-leadscrew-tr8-480mm.html

A lead screw is a long screw connected to a nut that can turn freely on the screw. If we rotate the screw and restrain the screw from rotating, the screw will move along the screw.

So, we can use the motor to rotate the screw, and put the load we want to move on the screw.

- Timing Belt:

References:
https://www.google.com/search?q=timing+belt&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjBiI-R1cfjAhUuxqYKHZTbBpEQ_AUIESgB&biw=1920&bih=969#imgrc=8l-RIow7DFniHM:

https://www.norwegiancreations.com/2015/07/tutorial-calibrating-stepper-motor-machines-with-belts-and-pulleys/

A timing belt with teeth can be connect from one side to the motor shaft, and the fixed in a way that allow the belt to be rotated by the motor.
Finally, we can put the object we want to move on the timing belt, but this will bend the belt as it not a solid, that is why we use bearings.

## 2- **Bearings:**

A bearing is a device that reduces friction between moving parts, and it is used to enable rotational or linear movement, while reducing friction and handling stress.


References:
https://www.indiamart.com/proddetail/linear-bearing-lmh10uu-15816244755.html

https://www.amazon.com/Linear-Motion-Bearing-Closed-Metric/dp/B002BBH5Z4

This is a linear bearing that we can mount to a round shaft, and when put a heavy object on it, it can move smoothely on the round shaft althought it is holding weight, because it reduces the fritcion.
The weight have to be reasonable (not too heavy).

This type of bearing is called a **Linear Bearing Slide.**

So we can use bearings to hold the object, and use a lead screw or a timing belt to move the bearings since they are easier to move thatn the object.

## 3- Microcontrollers:

Like in computers, the processor handles all the computing tasks, in a CNC machine we use a microcontroller as the brain of the machine.
This microcontroller can output voltages.
It can be programmed, so we can output voltages when we want.
An appropriate electrical motor can be connected to this microcontroller, so we can give voltage to theses motors by simply programming the microcontroller.
We can transform the rotary motion of the motor to a linear motion as mentioned above.
These are the basics of creating a movement using a microcontroller.

## 4- Power Supplies:

These are simple power supplies that will provide power to the microcontrollers and the motors.

How to choose a power supply?
We have to check the current that the motors need.
If we have k motors, we can use a simple formula to determine the current needed from the power supply:

Current needed from power supply > 0.75 * (sum of current needed by each of the k motors)

### *4.4 Parts used*

**1- Stepper Motor Nema 17:**

There are many types of stepper motors.
A Stepper motor is used when we need to rotate with precision, as it rotates in steps.
When coding, we can rotate the stepper motor in steps, and depending on the motor, each step is a certain angle.

Micro stepping is used for precision in movement. As the name indicates we move in micro steps.

Ex: If:

Each step is 1.8 degrees rotation of the shaft
So, we need 360/1.8 = 200 steps to make 1 revolution (360 degrees turn)

If we enable 1/32 micro stepping, the motor will do 36 micro steps per step,

And it will rotate 0.05 degreed each step instead of 1.8, because each micro step is 1.8/32 = 0.05 degrees

And we need 200*32 = 6400 micro steps to make a full revolution.

So now we rotate 0.05degrees per step, which we can use to increase the precision when moving, specially when we are moving in mm.


Reference: https://www.dynamic.me/index.php?_route_=stepper-motor-nema-17-42bygh&search=nema%2017

The current that this stepper motor needs is : 1.7A

We are using 4 of these motors, so we need a minimum 5.1 A power supply.

**2- Timing belt – pulleys with teeth – timing pulleys without teeth**

Reference:
https://www.dynamic.me/index.php?route=product/search&search=pulley

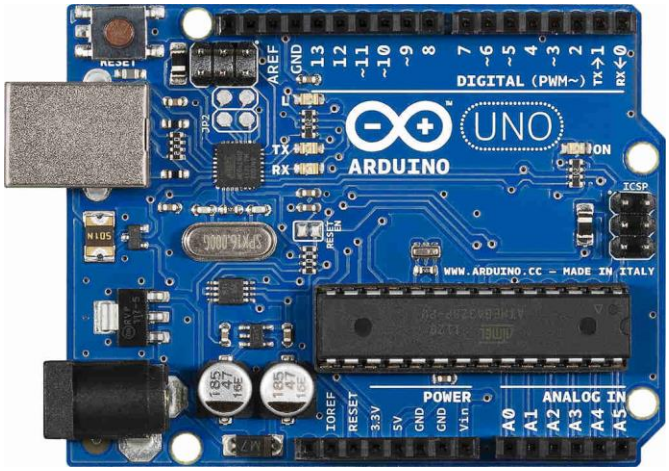
**3- Smooth Shaft (Act as support Axis X, Y or Z)**

Reference:
https://www.google.com/search?biw=1920&bih=969&tbm=isch&sa=1&ei=ozc1XavABcqymwXX2ZFY&q=smooth+shaft+axis&oq=smooth+shaft+axis&gs_l=img.3...38491.40564..40855...1.0..0.117.683.0j6.....0....1..gws-wiz-img.....0i30j0i24.qyNvbJ-1VH0&ved=0ahUKEwjrndHB1MfjAhVK2aYKHddsBAsQ4dUDCAY&uact=5#imgrc=wMP2VYkb-WJH7M:

## 4- Linear Bearing Slide Axis

Reference: https://www.amazon.com/Linear-Motion-Bearing-Closed-Metric/dp/B002BBH5Z4

## 5- Microcontroller:

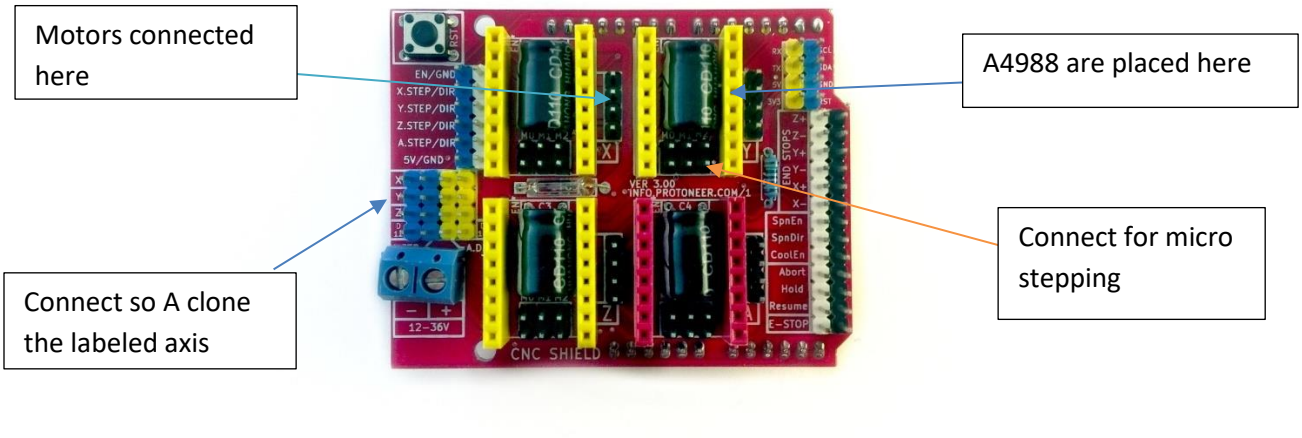



Reference : http://robotechshop.com/shop/arduino/arduino-board/arduino-uno-r3-china/?v=c951270e425b

The Arduino UNO is a microcontroller which we can plug it the comport with a serial-to-USB cable in order to program it.
The serial end is connected to the Arduino, and the usb end is connected to the computer.

C is the programming language used.

## 6- CNC Shield




Reference: http://www.alselectro.com/stepper-driver-cnc-shield.html

This is a shield that we can plug on the Arduino microcontroller, which makes it easier to control 4 stepper motors.

We need 4 stepper motor drivers called A4988 that we mount on that shield, then finally we connect to motor cables the 4 pins near the X Y Z and A labels.


Reference : https://createc3d.com/shop/en/electronics/329-buy-driver-a4988-offer-price.html

The machine will run using 4 motors: 2 to control the Y movement, 1 to control the X movement, and 1 to control the Z movement.

The motor labeled A on the CNC shield is used to clone one of the motors, and we need to clone the Y motor (because we need 2 motors to control the Y axis). To clone the Y motor, we connect the 2 blue pins on the right together, and the 2 yellow pins next to them together (We can use jumpers to connect them).

It's easier to connect the motor to this shield, than connecting the motor to the A4988 driver and then connecting that driver to the Arduino.

Now we plug this shield to Arduino, we give it power from a power supply (12V – 36V), we connect the motors to that shield, we connect the Arduino port to the computer, and finally we can start programming the Arduino, which mean controlling the motors with a code.

In this project, no libraries were used to control the motors (like the GRBL library written for the CNC shield).

## 7- Pen

This pen is uncapped, and fixed on the Z axis.
So, by moving the Z Axis we can make the pen touch the board (drawing), or not (not drawing)

### *4.5 Design*

This is the 3d design Model that this machine is built on.

I created this model using Design Spark which is a free program used to make mechanical designs.

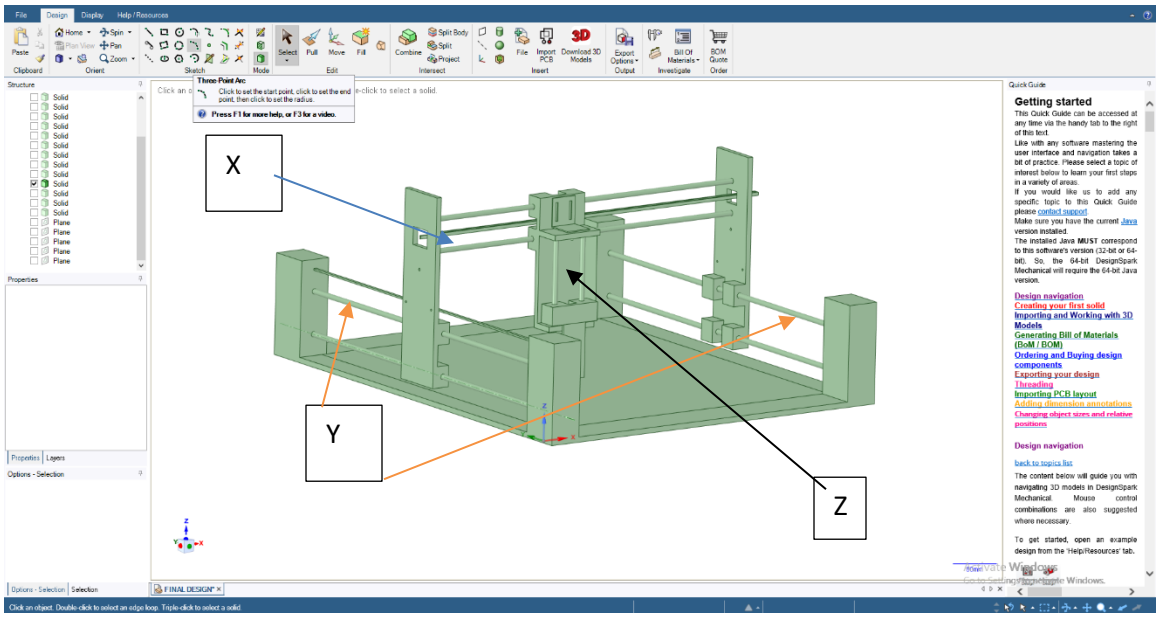
Axis are show in the figure below: 3 Axis (2 Y, 1 X, 1 Z)

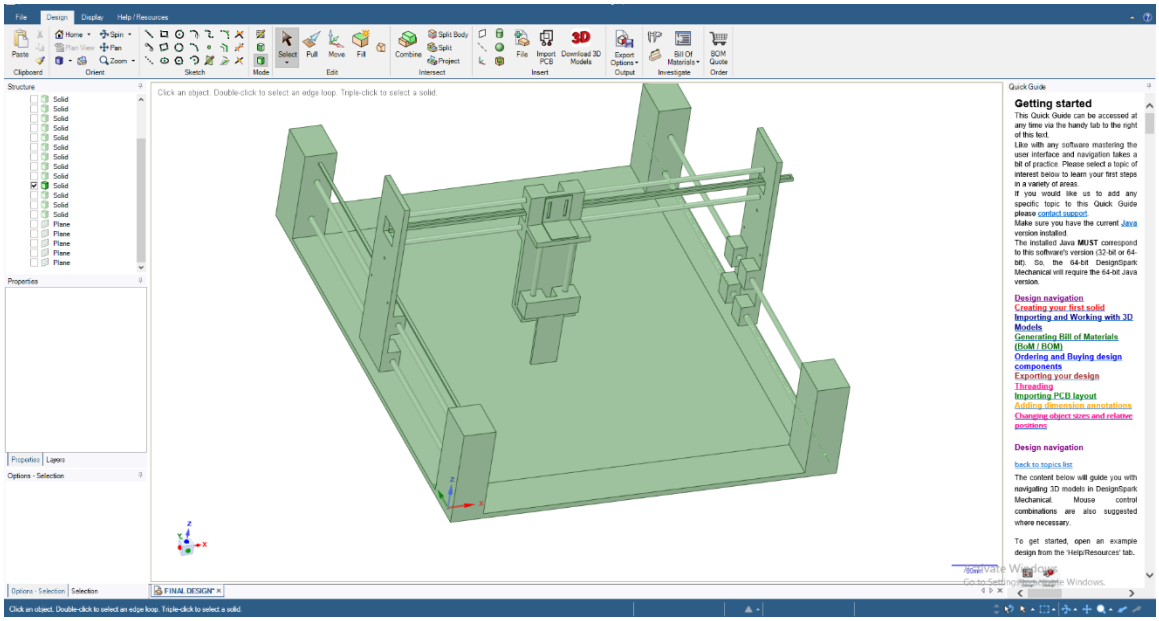
Each axis has a motor, so we need 4 motors

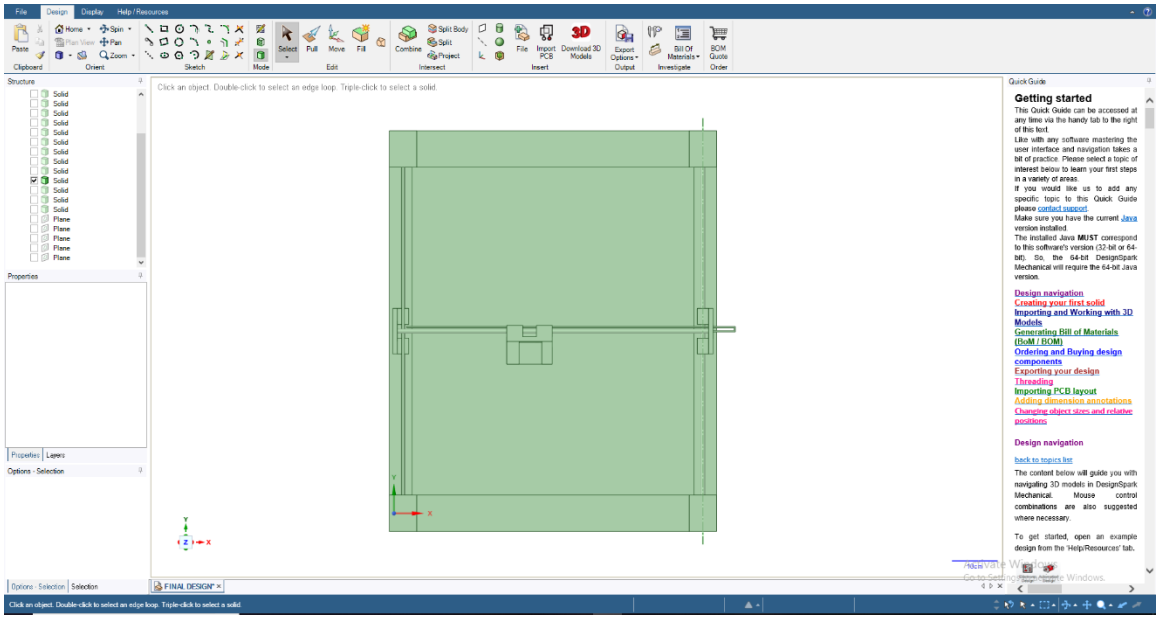
2 motors for the Y axis, spinning in opposite direction, so we can move up and down

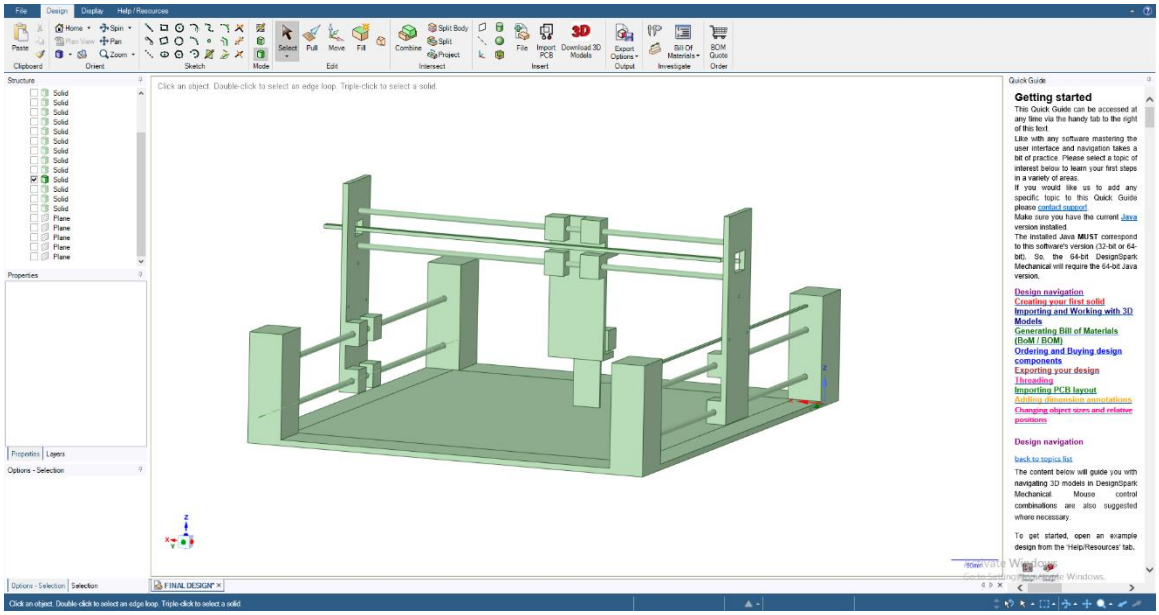
1 motor for the X axis to move on the X axis

1 motor for the Z axis to move the pen up and down



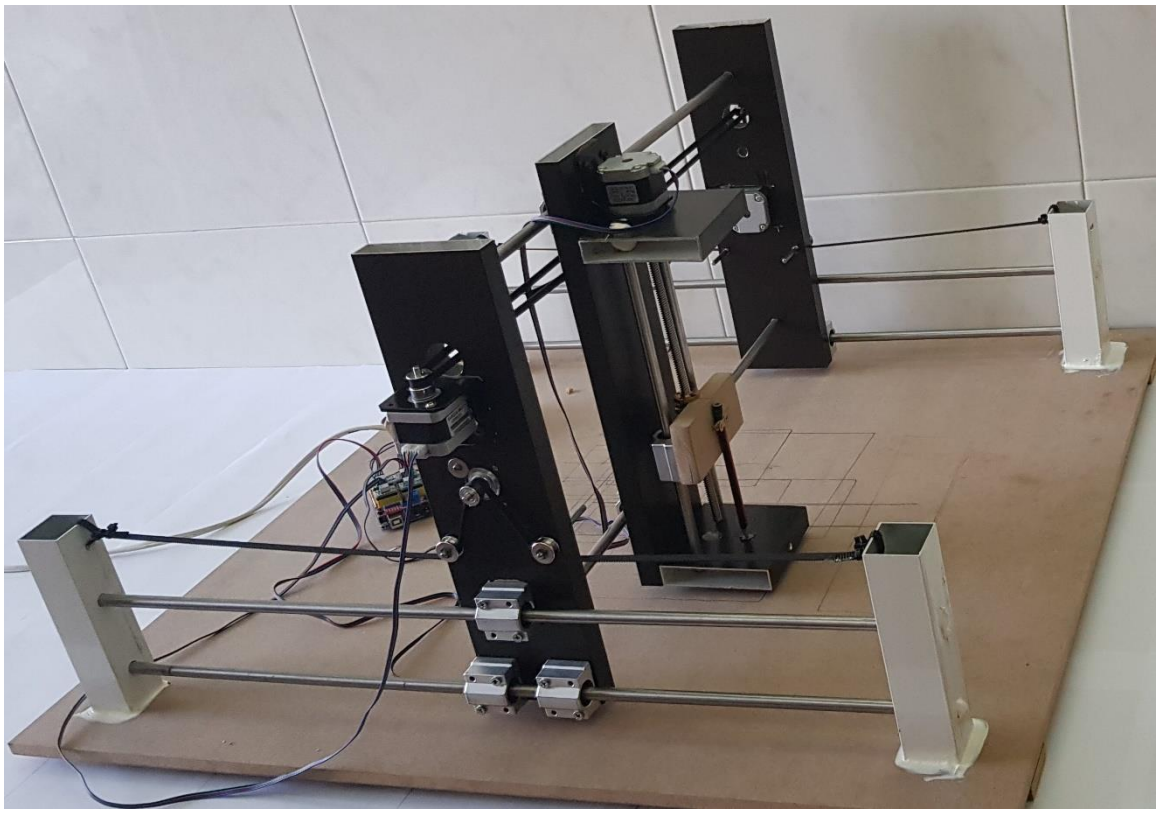

Click an object. Double-click to select an edge loop. Triple-click to select a solid.
Quick Guide
Getting started
Design navigation
Creating your first solid
Importing and Working with 3D Models
Generating Bill of Materials (BoM / BOM)
Ordering and Buying design components
Exporting your design
Threading
Importing PCB layout
Adding dimension annotations
Changing object sizes and relative positions
Design navigation
back to topics list
FINAL DESIGN*


Click an object. Double-click to select an edge loop. Triple-click to select a solid.
Quick Guide
Getting started
Design navigation
Creating your first solid
Importing and Working with 3D Models
Generating Bill of Materials (BoM / BOM)
Ordering and Buying design components
Exporting your design
Threading
Importing PCB layout
Adding dimension annotations
Changing object sizes and relative positions
Design navigation
back to topics list
FINAL DESIGN*

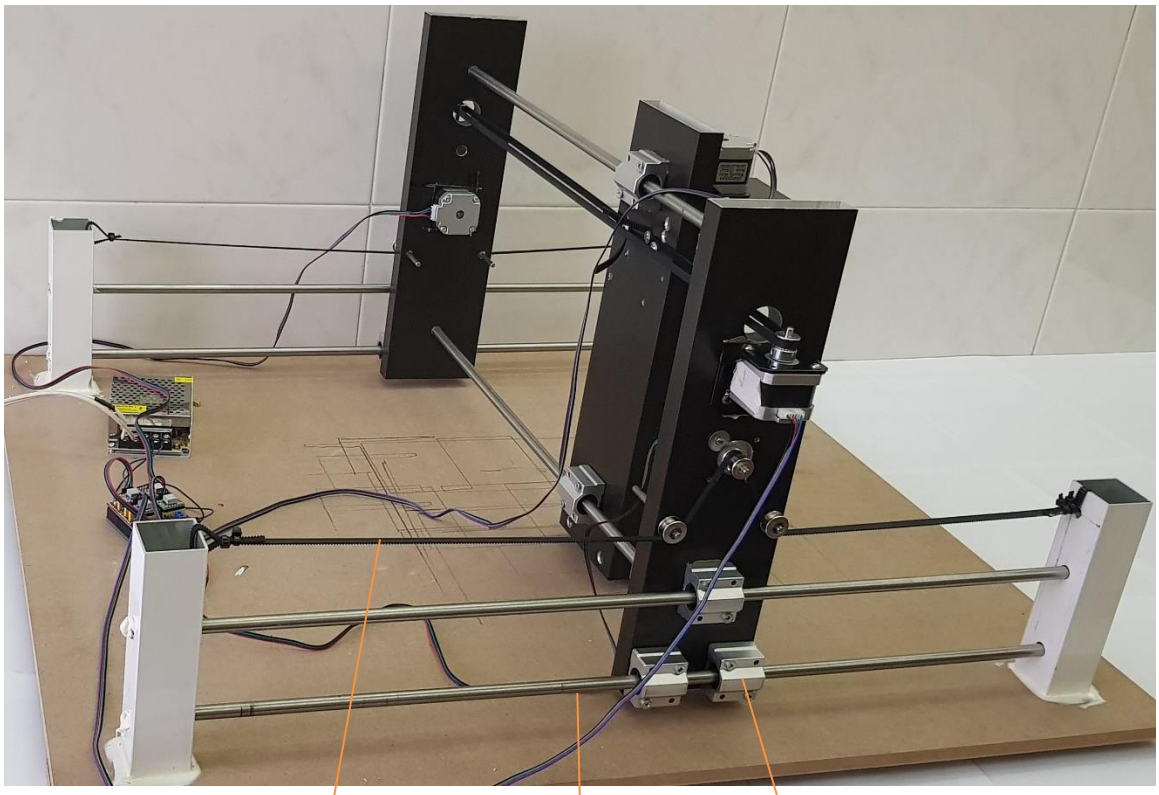

File
Design
Display
Help / Resources
Paste
Clipboard
Orient
Sketch
Mode
Select
Pull
Move
Fill
Edit
Combine
Split Body
Split
Project
Intersect
File
Import PCB
Download 3D Models
Insert
Export Options
Output
Bill Of Materials
Investigate
BOM Quote
Order
Structure
Properties
Click an object. Double-click to select an edge loop. Triple-click to select a solid.
Quick Guide
Getting started
This Quick Guide can be accessed at any time via the handy tab to the right of this text.
Like with any software mastering the user interface and navigation takes a bit of practice. Please select a topic of interest below to learn your first steps in a variety of areas.
If you would like us to add any specific topic to this Quick Guide please contact support.
Make sure you have the current Java version installed.
The installed Java MUST correspond to this software's version (32-bit or 64-bit). So, the 64-bit DesignSpark Mechanical will require the 64-bit Java version.
Design navigation
Creating your first solid
Importing and Working with 3D Models
Generating Bill of Materials (BoM / BOM)
Ordering and Buying design components
Exporting your design
Threading
Importing PCB layout
Adding dimension annotations
Changing object sizes and relative positions
Design navigation
back to topics list
The content below will guide you with navigating 3D models in DesignSpark Mechanical. Mouse control combinations are also suggested where necessary.
To get started, open an example design from the 'Help/Resources' tab.
FINAL DESIGN*
Click an object. Double-click to select an edge loop. Triple-click to select a solid

### *4.6 Machine Screenshots*

Front View:



Back view



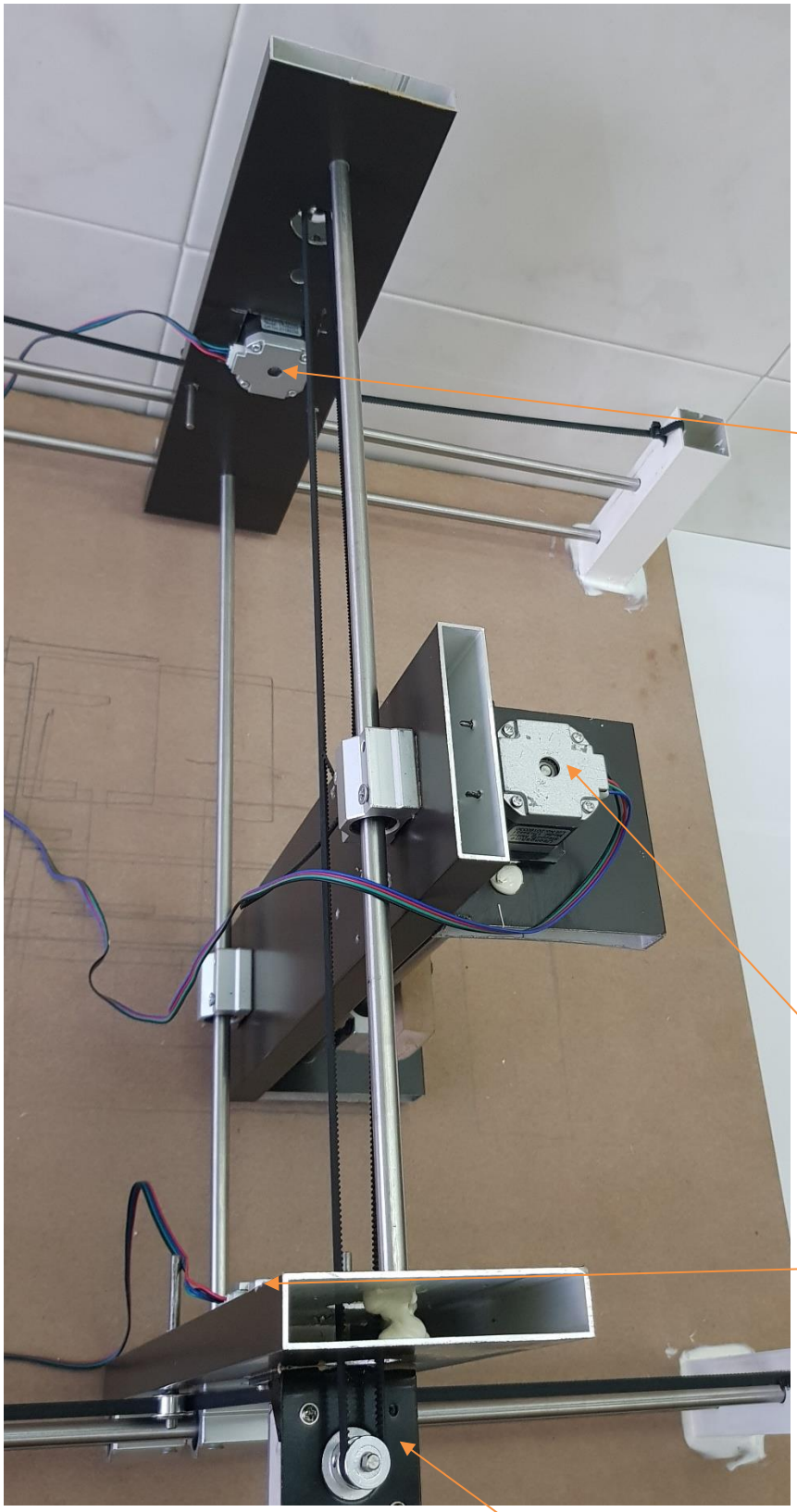
Y axis timing belt

Linear slide bearings

Smooth metallic shaft

It acts as support axis

Top view:



Y motors

(2 motors rotating in different

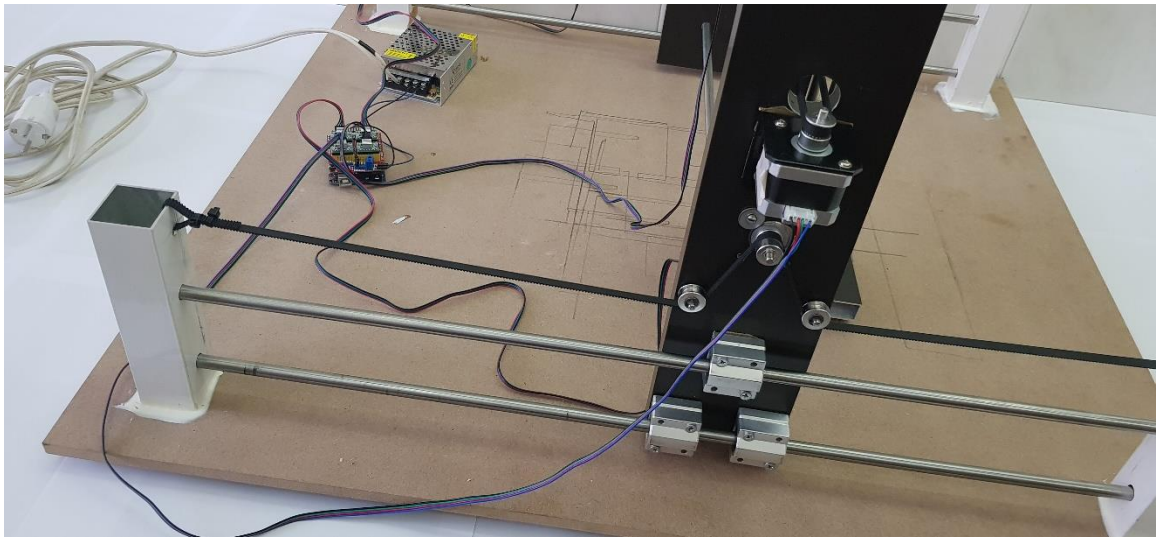
directions so we move in

one direct ion on the Y axis)

Z motor

X motor

Y Axis:

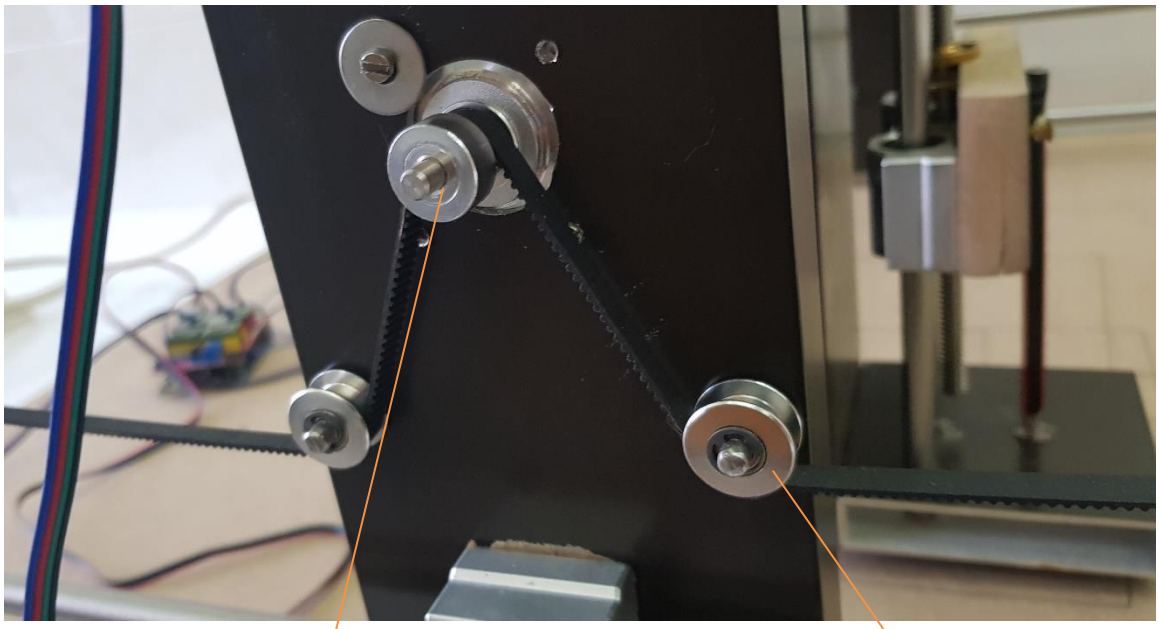


The timing belt is fixed on both ends.

The aluminum plate is attached to the bearings which can move smoothly on the 2 smooth metallic shafts.

The motor is attached to the other side of the aluminum plate, so its shaft comes out from this side.

This is done on the tow opposite aluminum plates of the Y axis



Timing pulley fixed to the shaft of the motor.

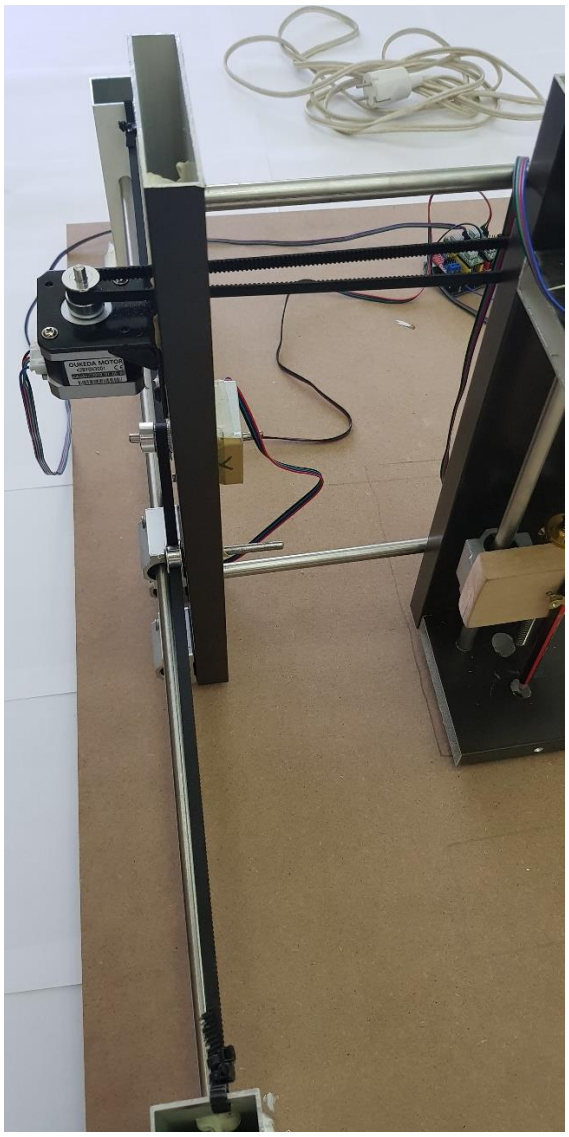
This is used to engage with the teeth of the timing belt

Bearings that can rotate smoothly

Now as the timing belt can’t move, when we rotate the motor the teeth of the timing pulley engage with the teeth of the timing belt, and cause the aluminum plate to move along the Y Axis.

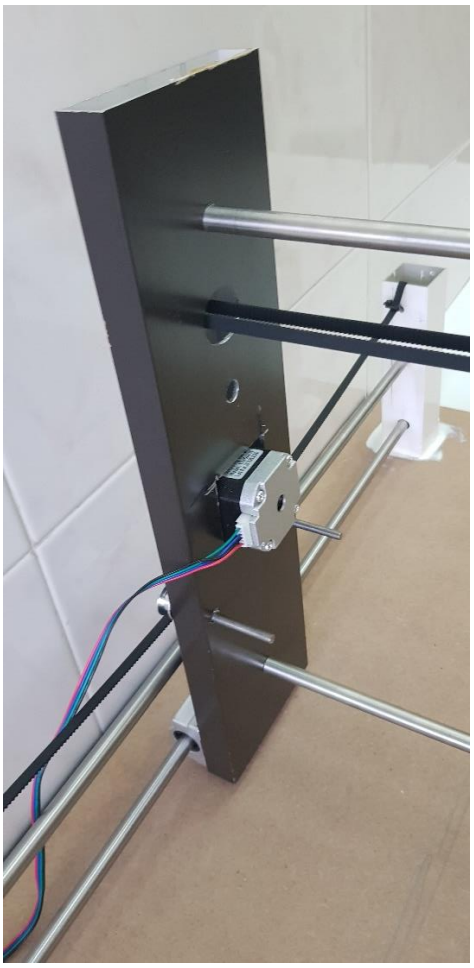
On the opposite aluminum plate the motor has to rotate in the opposite direction, so the two plates move in one direction.

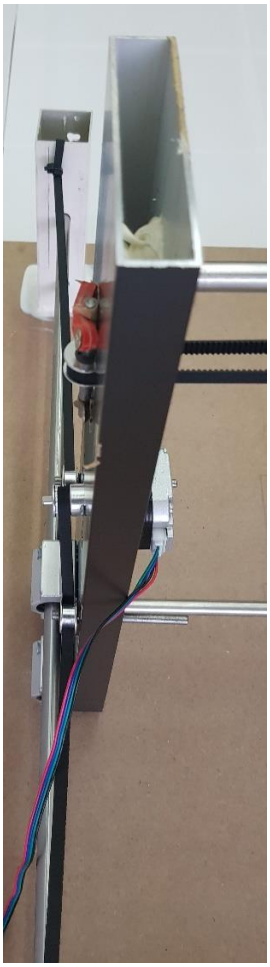
X Axis:



This motor placed on one of the Y aluminium plates, rotate to move the timing belt attached to the aluminium plate of the Z Axis.

On the opposite plate of the Y axis, the timing belt if rolled arround a bearing that can rotate soomthly like the ones used in the previous picture with the timing belt of the Y axis.

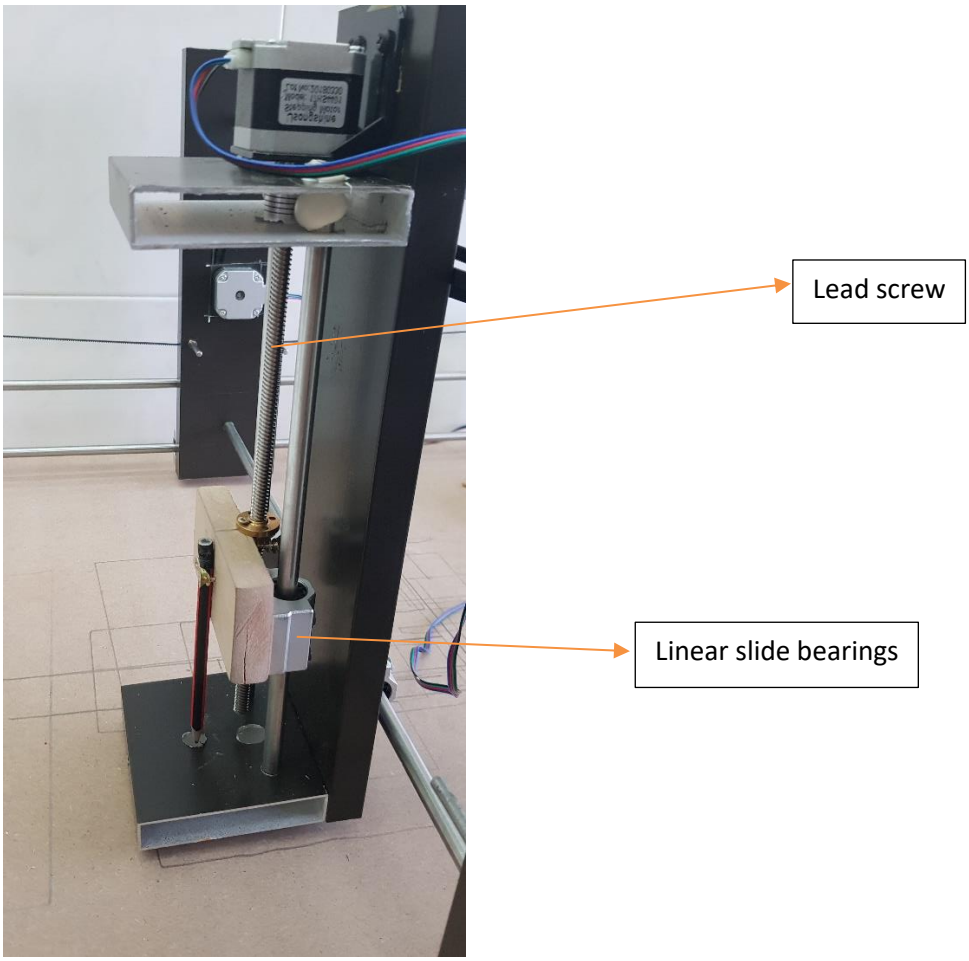




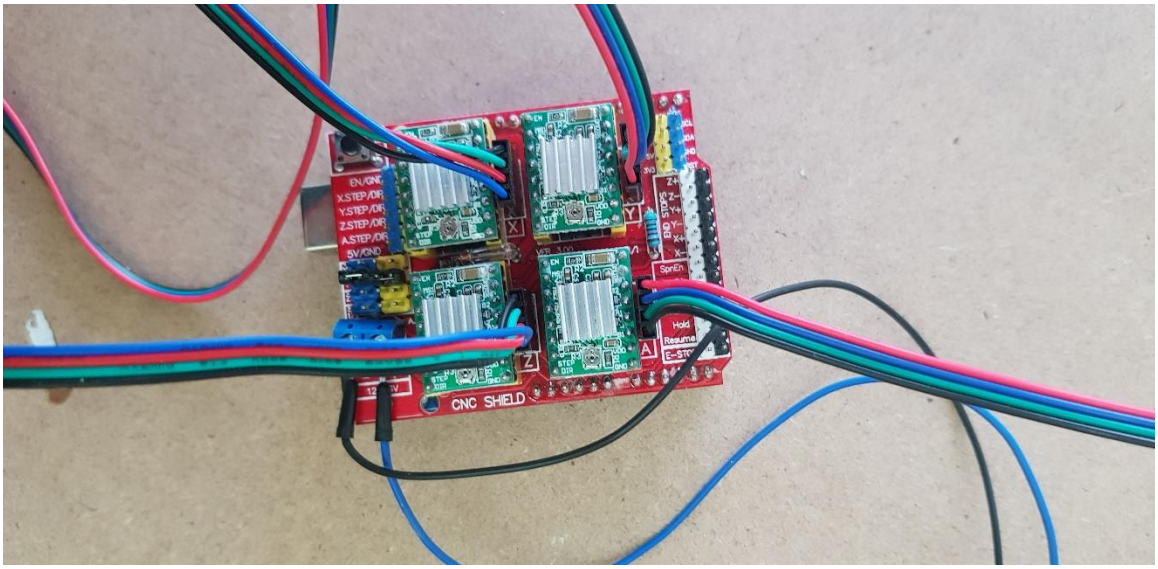
This is the opposite plate of the Y axis where we roll the timing belt of the X axis arround the bearing.

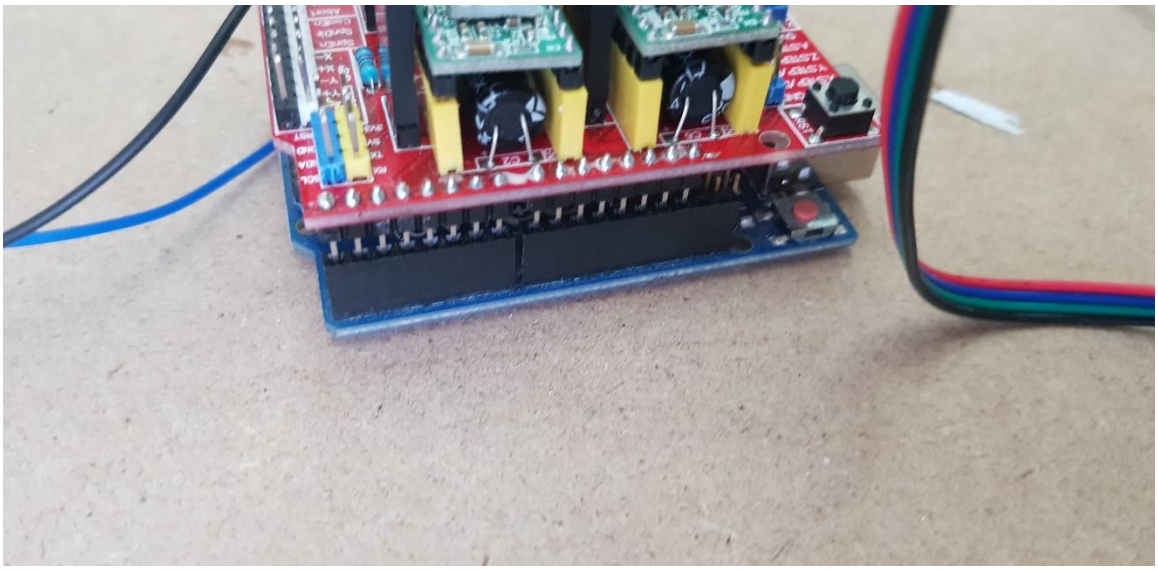
This shows how the timing belt is attached to the plate the Z Axis in order to move it when the motor rotates.

Z Axis:



CNC Shield:



It is connnected to the arduino, to the pwer supply.

The four motors are connected to this cnc Shield.

Power supply:



As explained, we need a 5A power supply to run the four motors we have.

We connect it to the CNC shield which need at least 12V.

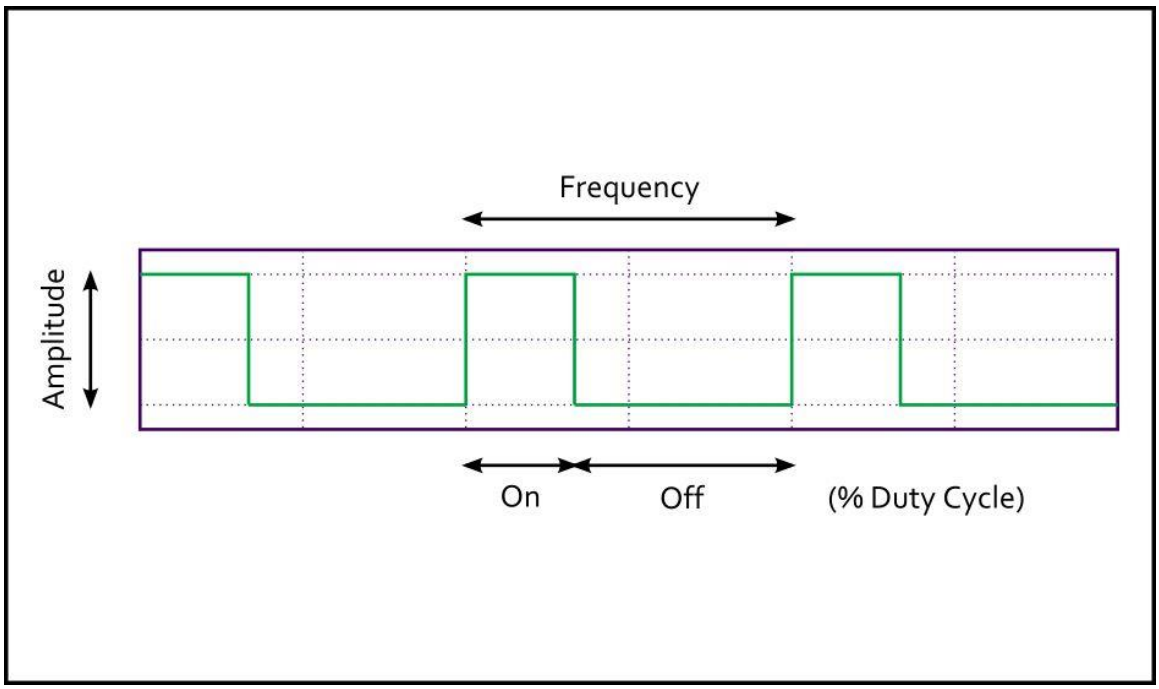
### *4.7 Programming*

Arduino IDE is used to write the code, that we upload on the Arduino by clicking a button on that IDE.

The base language is C.

#### 1- **Code to control the motors:**

We can rotate the stepper motor using PWM (pulse width modulation), which is a technique for creating digital square wave signal.




Reference: https://www.instructables.com/id/Arduino-Hardware-PWM-for-stepper-motor-drives/

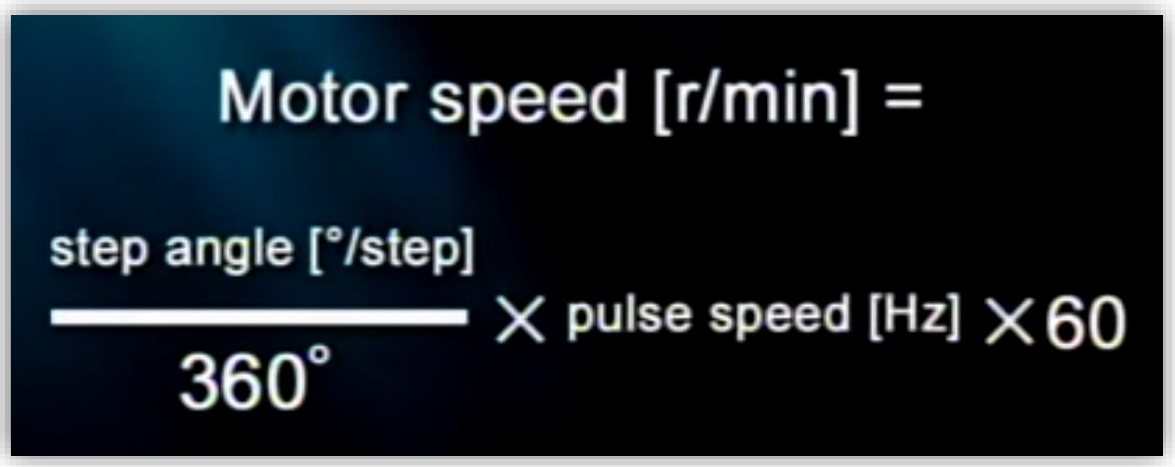
The amplitude is 5V, as this is the output voltage of the Arduino, and it is enough for a Nema 17 motor to rotate.

When providing these electric pulses to the motor, it will rotate with a certain speed.

Every pulse the motor will rotate one step.

We can change the speed of the motor by varying the frequency of the pulse signals.

We can obtain the motor speed with this formula:




Reference : https://www.youtube.com/watch?v=VCv4PeEWfzQ

Pulse Speed: frequency of the pulses

r/min: revolution per minute (360 rotation per minute)

We program the Arduino to send pulses to the pins where the motor is controlled.

This is code written in Arduino to send PWM with a pulse speed of 800:

```
for (int i = 0; i <n; i ++) {
 digitalWrite (motorPin, HIGH);
 delayMicroseconds (delayMS);
 digitalWrite (motorPin, LOW);
 delayMicroseconds (delayMS);
}
```

We will turn n steps, with a speed of 240r/m (the step angle of this the motor used is 1.8)

## 2- Libraries created:

Libraries were created when programming the Arduino, in order to make the final code short and easier to read.

Stp.h: Used to hold the function needed to control a stepper motor, like: Step(char dir, int nbSteps) which send PWM signals using the code mentioned above, plus the constructer where we specify the pins of the Arduino that are connected to the motor.

Reader.h and Write.h: Used to handle communication between the Arduino and the desktop application.
Writer write integers, characters, and arrays in a certain form: f
Reader read integers, characters, and arrays written in a certain form: f
The data is sent under this form f explained in Algorithm and CNC Communication section below.

## 5. Application and CNC Communication

Arduino is not like a USB device or another computer. Although it is connected to the USB port of the computer, it can't access the data storage and the memory of the computer. So, we cannot program the Arduino to read from a text file located on the computer.
But we can send serial data which is the process of sending data one bit at a time.

Most computer boards don't have serial ports on them, they only have USB ports, this is why Virtual COM Port setting is used.
Virtual COM Port is a setting within the Windows Operating System that allows software products to connect to USB devices using traditional serial (COM Port) connections while the plug on the back of the computer remains USB.

Library used in Eclipse Mars:

Processing: By default, java Eclipse Mars can't send Serial Data, this is why we used this library in order to send Serial data and communicate with Arduino.

Data is sent in a certain way, so the receiver (Arduino) can know what it actually means. For example, if we send an array, the data must be in this form:

a- 5- 1- 2- 3- 4- 6-

Where a denotes that this is an array, 5 its size, and the reset is the content of the array. The dash ' – ' is ignored as it indicates a move to next data.

So, everything we do when we are on the CNC Window of the application is sent in a similar way to the CNC machine, like when the user is calibrating the machine data is sent in this way: c- …. , indicating that the user is calibrating.

Finally, when we want to draw data is also sent in certain way.

The Reader and Writer libraries created for the Arduino are used to let this microcontroller read and write to the application.

The Arduino class created for the application is used to let it read and write data to the Arduino.

Writer library:

Writer.cpp | Reader.cpp

```
# include "Arduino.h"
# include "Writer.h"

 Writer::Writer() {}

void Writer::writeChar (char c){
     char s [3] = {c,'-','\0'};
     Serial.write(s);
}
```

Reader library:

Writer.cpp | Reader.cpp

```
# include "Arduino.h"
# include "Reader.h"
# include "Carray.h"

 Reader::Reader() {}

int Reader::readInt() {
    while (!Serial.available () ) {delay (500);}
    String s = Serial.readStringUntil('-');
    return s.toInt () ;
}

char Reader::readChar () {
    while (!Serial.available () ) {delay (500);}
    String s = Serial.readStringUntil ('-');
    return s[0];
}

Carray Reader::readArray() {
    int l = readInt();
    Carray a = Carray (l);

    for ( int i = 0 ; i < l ; i ++ ){
        a.Add(i,readInt ());
    }

    return a ;
}

void Reader::flash(int x){
   for ( int i = 0 ; i < x ; i ++ ){
     digitalWrite(13,HIGH);
     delay (500);
     digitalWrite(13,LOW);
     delay (500);
   }
}

void Reader::flash (Carray x ){
   for ( int i = 0 ; i < x.l ; i ++ ){
     flash (x.a[i]);
     delay (1000);
   }
}
```

Arduino class:

```
42
43      public  void WriteInt (int x){
44          if ( !isConnected ()) return ;
45
46          String s = String.valueOf(x)+'-';
47          serial.write(s);
48      }
49
50      public void WriteChar (char c){
51          if ( !isConnected () ) return ;
52
53          String s = String.valueOf(c)+'-';
54          serial.write(s);
55      }
56
57      public char readChar () {
58          if ( !isConnected () ) return 'e' ;
59
60          while (serial.available() == 0) {delay (500);}
61
62          String s = serial.readStringUntil('-');
63          return s.charAt(0) ;
64      }
65
66      public  void WriteArray (int a []){
67          if ( !isConnected ()) return ;
68
69          WriteInt (a.length);
70          for ( int i = 0 ; i < a.length ; i ++ ){
71              WriteInt(a[i]);
72          }
73      }
74
75  }
```

## 6. Drawing Algorithm

This algorithm is very simple, and it is uploaded to the Arduino in order to run the motors and move the pen along the X Y and Z axis.

When the user clicks the Draw button in the application, serial data is sent to Arduino containing information about all the rectangles: an array containing the pos width and height of each rectangle. This data is also sent in a certain way as discussed earlier.

Then the machine loop through all the squares, and draw each one.

```
void DrawSquare (Point pos , int width , int height ){
  Move(pos,false);
  Move (Point (currentPos.x+width,currentPos.y) ,true);
  Move (Point (currentPos.x,currentPos.y + height) ,true);
  Move (Point (currentPos.x - width,currentPos.y) ,true);
  Move (Point (currentPos.x,currentPos.y - height) ,true);
}
```

Point is a struct containing integer x and integer y.

This function takes the position of the square, its width and height.

Move is a function that take a Point target, and a Boolean which indicate if the pen has to be touching the wood board when we move.

When the application sends the array holding the information about the squares, Arduino will loop through this array and call this function DrawSquare on each square.

## 7. Conclusion

The project is about helping a carpenter cut shapes in a rectangular wooden board.

But the application created can be used in other applications as well, because of the algorithm that it uses. For example, in warehousing contexts, goods have to be placed on shelves, in newspaper paging articles and advertisements have to be arranged in pages, and in the shipping industry a bunch of objects of various sizes have to be shipped as many as possible in a larger container.

The MVC structure of the application makes it very easy to modify it in the future, and add new algorithms and new shapes that the carpenter can cut.

The code of the CNC allows it to move in diagonal form (not only right left up and down), which later give the option use to draw different shapes other than rectangles, by giving it the path it has to follow.

Finally, this machine may not be big enough to be used by a carpenter that need to cut on a large piece of wood, but by using the same algorithm and concept, with more powerful motors

## 8. References

Algorithm:

https://www.inf.utfsm.cl/~mcriff/SP/articulo4.pdf

The reference of each picture is written under it.