Research assignment

IT essentials 1ACS

Matt Boeren 1ACS02
r0932402

CAMPUS

Geel

Technology
Elektronics-ICT / Applied informatics

IT essentials

Course unit: IT essentials

Educational activity: IT essentials

First tier

Academic year 2022-2023

Word in advance

My subject is the drawing program I made for my self build CNC penplotter. I was going to do this eighter way but when the assignment came for the research paper I thought about dining it for a school project. It came out and it was doable within the death line.

The programming of the program went fairly smooth except for some problems. The writing of my paper on the other hand didn’t go as smooth as I hoped. I had a hard time finding a balance between a detailed and understandable paper. I think at the end I found a good balance between the two. Another reason why writing the paper didn’t go as smooth is because it is the first paper I wrote in English.

Table of contents

Content

List of illustrations

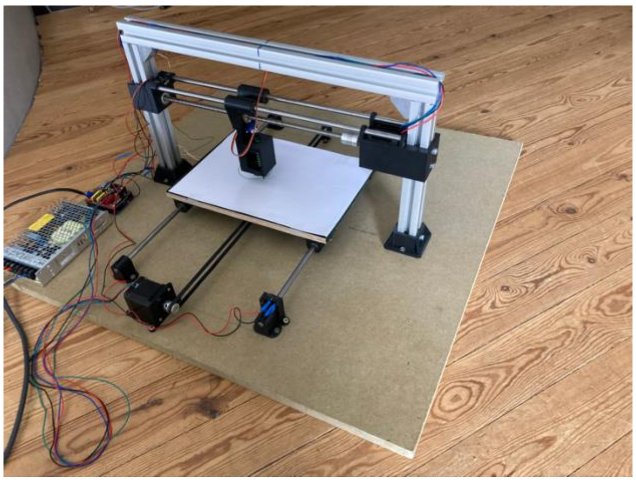
Figure 1: My CNC penplotter

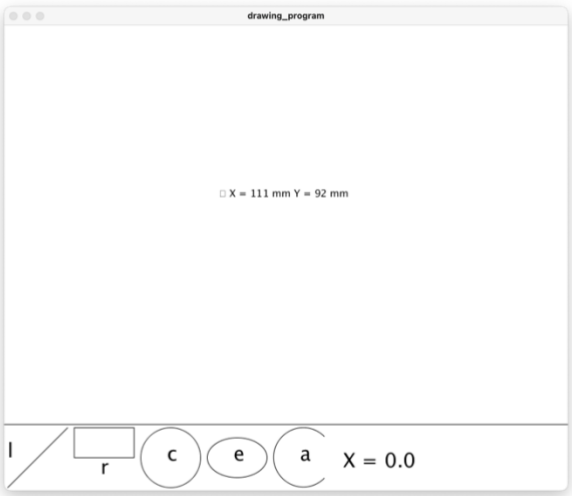

Figure 2: Layout of the program

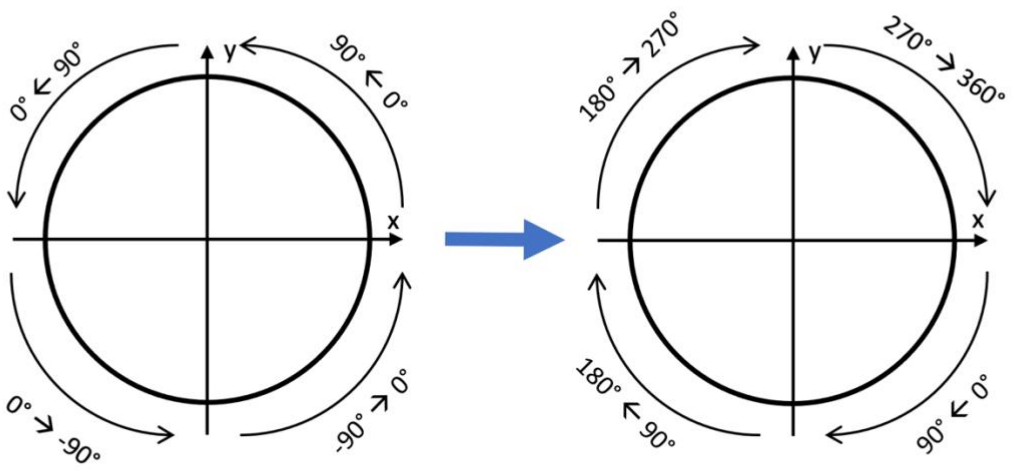

Figure 3: Angle conversion

1 Introduction to my drawing program

In my last year of high school I build and programmed a CNC penplotter for my integrated project (see figure 1). A CNC penplotter is a machine that can draw figures on a piece of paper. I programmed it to draw basic figures. The basic figures were a line, a rectangle, an ellipse, a circle and a circle arc.

Then the user had to enter a command to draw the figures. For My research paper I made a drawing program so the user could draw a picture using the basic figures on a computer before the machine draws it on the paper.

In my integrated project I went in depth about how the machine draws the figures so in my research paper I am only going to talk about the drawing program.

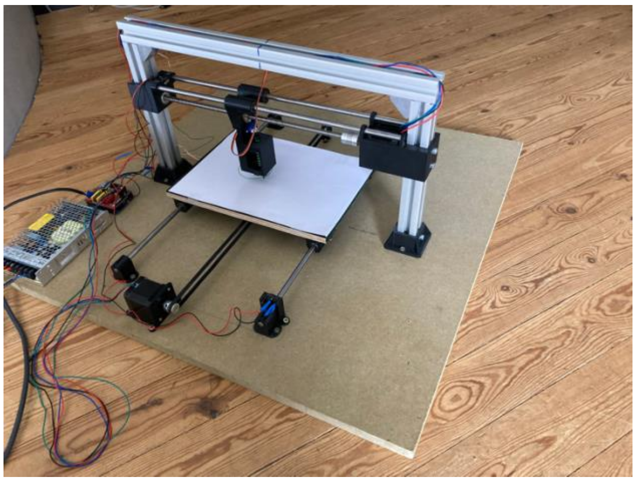

Figure 1: My CNC penplotter

2 Basic structure

I coded my drawing program in java using the processing environment. I choose to use processing because it is an environment made for graphics and it can easily be connected to an Arduino microcontroller. An Arduino microcontroller can be used to program small electronics and is the microcontroller I used to build my CNC penplotter.

The program works by first selecting a shape. You can do this by the keyboard shortcuts or the buttons below the canvas where you can draw (see Figure 2). After that you can draw the shape on the computer by pressing the mouse or giving in the coordinate of the point you should press with the mouse.

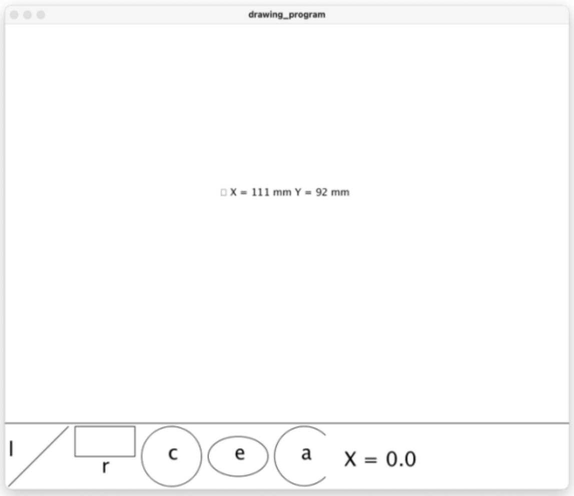

Figure 2: Layout of the program

To code the drawing program I used object-oriented programming. Within object-oriented programming the code is built from objects. For each object you write a class with the functions that the object should do. This is like a blueprint for the object. If the class is made within the main code multiple objects of the same kind can be created and for all of those all the functions within the class can be used. In my code I have a class for every type of shape. For example, I wrote a class for drawing a circle. Within the class are the functions to create a circle. Within the main code of the program I can call this functions per circle you create. Those circles and other shapes are stored in an arraylist. With an arraylist you can add objects to it or remove objects as much as you want. In contrast of using a normal array where you give your list a certain length and can only use the objects within the given length.

After drawing the picture you want the picture can be uploaded by plugging the Arduino into the computer and starting the serial communication by pressing enter. Then the shapes will be sent to Arduino one by one. If the Arduino gets a shape it will draw it and wait until the next one get sent by the drawing program.
The Arduino expects measurements for distance in millimeters but in my program the measurements are in pixels. My drawable canvas on the computer is 848 pixels by 600 pixels to represent an A4 paper with the dimensions of 297 mm by 210 mm. This means that one pixel is equal to 0.35 mm. This is calculated by the following formula $\frac{210mm}{600\,pixels} = 0.35\ \frac{mm}{pixel}$.

Within the main code of the drawing program I used the void setup(), the void draw() loop and two events that are built into processing. The first one is the event void keyPressed(). The code within the void setup() will only execute once before launching the program. Within the void setup() things are coded that only needs to be done ones like initializing the size of the canvas. The void draw() is a loop that wil loop until the program is quit. The code within the void keyPressed() event will only be executed if the user pressed a key on the keyboard. The other event I used is the void mousePressed() event. The code within this event will only be executed if the user presses the mouse. The events used are programmed outside the void draw() loop.

3 Shapes classes

3.1 Line class

```
class Line {
  PVector startPoint, endPoint;
  void createLine1(PVector mousePoint1){
    startPoint = mousePoint1;
  }

  void createLine2(PVector mousePoint2){
    endPoint = mousePoint2;
  }

  void drawLine1(){
    line(startPoint.x, startPoint.y, mouseX, mouseY);
  }

  void drawLine2(){
    line(startPoint.x, startPoint.y, endPoint.x, endPoint.y);
  }

  String sendLine(){
    String lineInformation = "L " + str(int(startPoint.x * pixToMm)) + " " +
str(int(startPoint.y * pixToMm)) + " " + str(int(endPoint.x * pixToMm)) + " "
+ str(int(endPoint.y * pixToMm));

    return lineInformation;
  }
}
```

Above you can see how I programmed the class for a line. The line is drawn by pressing the mouse twice or giving in two coordinates with the keyboard (see 5 Number input). The class starts with declaring two variables of the type PVector. A PVector is a variable that stores two values. A x and a y value. Those values can be used by putting .x or .y after the variable name. This variable type is ideal for storing a coordinate. The two PVectors that I use within the class are startPoint and endPoint. Those are the start point and the endpoint of the line.

The first function within the class is void createLine1(PVector mousePoint1). If we want to execute this function in the main program we need to send a variable with it. More specific the coordinate of the start point. Before the name of the function is a void this means this part of code will only be executed and will not send data back. The beginning and end of a function get defined by the curly brackets after the declaration of the function and the end of the function. Within the function the start point of the line will be set equal to mousePoint1.

With the void drawLine1() function will a line get drawn from the start point until the current position of the mouse. The mouseX and mouseY variables are built in variables within processing. The mouseX variable represents the x-coordinate of the mouse and the mouseY variable represents the y-coordinate of the mouse.

When the user has decided where they want to end the line by clicking the mouse for the second time or giving in the coordinate with the keyboard the function void createLine2(PVector mousePoint2) will be executed. Within the function the endpoint of the line will be set equal to mousePoint2.

When the start- and endpoint are defined the line can get drawn permanent by the void drawLine2() function. Within this function the program will draw a line from the startpoint until the endpoint.

When the user has finished drawing the figure they want the data of the line needs to be send over to the machine. The machine for a line expects a command in the following format: “L X1 Y1 X2 Y2”. To do this I programmed a function with the return type string. The code will create a string starting with a capital L to indicate that the machine needs to draw a line. The coordinates within the program are in pixels but the penplotter needs a measurement in millimeters. So before the drawing program sends the data to the penplotter the coordinates needs to be recalculated to millimeters. This is done by multiplying the coordinate by 0.35 (see 2 basic structure). This value did I store in a global variable named pixToMm. Global variables can be used true the whole program. This will leave the code with a value of the data type float. Floats are used to store decimal numbers. But the penplotter expects a value of the data type integer. Integers are used to store whole numbers. So the coordinates first needs to be converted to an integer and after that to a string because serial communication runs by strings. Within the Arduino program this will be converted to back to an integer.

When the complete string is made the program will return the string to the main program (see 6 Serial communication).

3.2 Rectangle class

```
class Rectangle{
  PVector corner1, corner2;
  void createRectangle1(PVector mousePoint1){
    corner1 = mousePoint1;
  }

  void createRectangle2(PVector mousePoint2){
    corner2 = mousePoint2;
  }

  void drawRectangle1(){
    noFill();
    rectMode(CORNERS);
    rect(corner1.x, corner1.y, mouseX, mouseY);
  }

  void drawRectangle2(){
    noFill();
    rectMode(CORNERS);
    rect(corner1.x, corner1.y, corner2.x, corner2.y);
  }

  String sendRectangle(){

    String rectangleInformation = "R " + str(int(corner1.x * pixToMm)) + " "
+ str(int(corner1.y * pixToMm)) + " " + str(int(corner2.x * pixToMm)) + " " +
str(int(corner2.y * pixToMm));

  return rectangleInformation;
  }
}
```

Above you can see the class for drawing a rectangle. The rectangle is drawn by pressing the mouse twice or giving in two coordinates with the keyboard (see 5 Number input). This will be the coordinates of two opposite corners of the rectangle.

The class starts with declaring two variables of the type PVector named corner1 and corner2. This two variables will store the coordinates of the two opposite corners.

The first function within the class is the void createRectangle1(PVector mousePoint1) within this function the coordinate of the first corner will be set equal to mousePoint1.

When the coordinate of the first corner is defined the drawRectangle1() function can be executed. Within this function a rectangle will be drawn between the coordinate of the first corner and the coordinate of the mouse by using the mouseX and mouseY variables. Within this function the noFill(); an rectMode(CORNERS); functions are used. Those functions are built in functions in the processing environment. With the noFill(); function the processing environment knows the filling of the shape needs to be transparent. By default it will be the last filling used or if no filling was used white.

By default the processing environment draws rectangles by defining the coordinate of the top left corner and a width and a height. But by using the rectMode(CORNERS); function processing knows the following coordinates within the rect() function are opposite corners and not the coordinates of the top left corner and a width and a height.

When the user knows where they want to draw a rectangle by pressing the mouse for a second time or giving in the second coordinate with the keyboard the function void createRectangle2(PVector mousePoint2) will execute and make the coordinate of the second corner equal to mousePoint2.

When the coordinate of both corners are defined the rectangle can be drawn on the screen permanently with the drawRectangle2() function. Within this function the build in noFill(); and rectMode(CENTER); are also used for the same reasons as within the first draw function. Within this function the rectangle will be drawn between the two opposite corners given in by the user.

When the user finished drawing the figure they want the data need to be sent to the penplotter. For a rectangle the penplotter expects a command in the following format "R X1 Y1 X2 Y2". The capital R is to indicate that the penplotter needs to draw a rectangle the X1 Y1 after it is the coordinate of the first corner. The X2 Y2 is the coordinate of the second corner. Just like the in the line class the coordinates will be first multiplied by 0.35 (pixToMm variable) and converted to an integer. After that they get converted to a string to send over to the penplotter.

3.3 Circle class

```
class Circle {
  PVector midPoint;
  float radius;

  void createCircle1(PVector mousePoint1){
    midPoint = mousePoint1;
  }

  void createCircle2(PVector mousePoint2){
    radius = dist(midPoint.x, midPoint.y, mousePoint2.x, mousePoint2.y);
  }

  void drawCircle1(){
    noFill();
    circle(midPoint.x, midPoint.y, 2*dist(midPoint.x, midPoint.y, mouseX,
mouseY));
  }

  void drawCircle2(){
    noFill();
    circle(midPoint.x, midPoint.y, 2*radius);
  }

  String sendCircle(){

    String circleInformation = "C " + str(int(midPoint.x * pixToMm)) + " " +
str(int(midPoint.y * pixToMm)) + " " + str(int(radius * pixToMm));

    return circleInformation;
  }
}
```

Above you can see the class for drawing a circle. A circle can be drawn by pressing the mouse two times or giving in two coordinates with the keyboard
(see 5 Number input). The first mouse press or the first given in coordinate will be for the coordinate of the middle point. The distance between the middle point and the second mouse press or given in coordinate will be the radius of the circle.

Within the class I start by declaring two variables one of the type PVector named midPoint and one of the type float named radius. The midpoint variable will be set to the middle point of the circle and the variable named radius will be set to the radius of the circle.

The first function in the class is the void createCircle1(PVector mousePoint1) within this class will the middle point of the circle be set equal to mousePoint1.

When the middle point of the circle is set a temporary circle can be drawn by using the function void drawCircle1(). This function starts with the build in noFill(); function to make sure the circle is transparent. After that a circle can be drawn with the set middle point and a radius that is equal to the distance between the middle point and the current position of the mouse. Processing has a build in function for this named dist. In this function you can give two coordinates and the function will return the distance between them. A circle in processing gets drawn by the circle function that takes the coordinate of the middle point and the diameter. So to draw a circle with a radius equal to the distance between the middle point and the position of the mouse the distance needs to multiplied by two.

When the user has decided the defined radius of the circle by pressing the mouse for a second time or giving in the second coordinate the void createCircle2(mousePoint2) will run. Within this function the radius of the circle will be set equal to the distance between the middle point and mousePoint2.

When the radius is set the circle can get drawn permanently by running the void drawCircle2() function. This function starts with the build in noFill() function to make sure the circle is transparent and after that a circle will be drawn with a middle point equal to midPoint and a diameter that is equal to two times the radius.

When the user is finished drawing the figure they want the data can be send over to the penplotter. For a circle the penplotter expects a command in the following format: “C X1 Y1 R”. The capital C is to indicate that the penplotter needs to draw a circle. The X1 Y1 after it are the coordinate of the middle point and the R is the radius. The coordinate of the middle point and the radius are expressed in pixels so those gets first converted to millimeters by multiplying them with 0.35 or the global variable pixToMm. After the conversion to millimeters they will get converted to an integer to work with the penplotter and get converted to a string for the serial communication.

3.4 Ellipse class

```
class Ellipse {
  PVector midPoint;
  float Width, Height;

  void createEllipse1(PVector mousePoint1){
    midPoint = mousePoint1;
  }

  void createEllipse2(PVector mousePoint2){
    Width = abs(mousePoint2.x - midPoint.x)*2;
    Height = abs(mousePoint2.y - midPoint.y)*2;
  }

  void drawEllipse1(){
    float Width1 = abs(mouseX - midPoint.x)*2;
    float Height1 = abs(mouseY - midPoint.y)*2;

    noFill();
    ellipse(midPoint.x, midPoint.y, Width1, Height1);
  }

  void drawEllipse2(){
    noFill();
    ellipse(midPoint.x, midPoint.y, Width, Height);
  }

  String sendEllipse(){

    String ellipseInformation = "E " + str(int(midPoint.x * pixToMm)) + " " +
str(int(midPoint.y * pixToMm)) + " " + str(int(Width * pixToMm)) + " " +
str(int(Height * pixToMm));

    return ellipseInformation;
  }
}
```

Above you can see the class for drawing ellipses. Within the class I declare three variables one of the type PVector to declare the coordinate of the middle point of and two variables of the type float to declare the width and height of the ellipse.

The first function within the class is the void createEllipse(PVector mousePoint1). Within this function the coordinate of the middle point will be set equal to the coordinate of mousePoint1.

When the middle point is set a temporary ellipse can be drawn by using the void drawEllipse1() function. Within this function I started with declaring two local variables of the type float. Local variables can only be used in the function they are declared. Those two variables are named width1 and height1. The variables store a temporary width and height of the ellipse. The width is calculated by subtracting the x-coordinate of the midpoint by the x-coordinate of the mouse. This can be either positive or negative but we need the absolute value of the subtraction. Processing has the build in abs() function to do that. There is one more problem with this the difference in distance between the x-coordinates of the mouse and the middle point is only half of the width we want. So the difference in distance needs to be multiplied by two. The height is calculated the same but with the y-coordinates of the mouse and the middle point. When the temporary width and height are calculated the temporary ellipse can be drawn. The program starts with the noFill() function to make sure the ellipse is transparent. To draw an ellipse in processing you can use the build in function ellipse(). This function needs a coordinate for the middle point of the ellipse and a width and a height. So to draw the temporary ellipse within the function will be the coordinate of the middle Point and the temporary width and height.

When the user has decided where to draw the ellipse by pressing the mouse a second time or giving in the second coordinate with the keyboard. The void createEllipse2(PVector mousePoint2) function will be executed. Within this function the permanent width and height will be calculated in the same way as the temporary width and height with the only difference that mousePoint2 is used instead of the current coordinate of the mouse.

When the permanent height and width are calculated the ellipse can be drawn permanently by running the void drawEllipse2() function. This function starts with the build in function noFill() to make the ellipse transparent. After that the ellipse will be drawn with the middle point and the permanent width and height.

When the user is finished drawing the figure they want the data of the ellipse can be send to the penplotter. The penplotter expects a command in the following format: “E X1 Y1 W H”. The capital E indicates that the penplotter needs to draw an ellipse. The X1 Y1 are the coordinate of the middlepoint and W is the width and H is the height. To send the to the penplotter the coordinate of the middle point, the width and the height that are expressed in pixels needs to be converted to millimeters by multiplying them by 0.35 or the pixToMm variable. After that they needs to be converted to an integer and after that to a string for the serial communication.

3.5 Circle arc class

```
class CircleArc{
  PVector midPoint;
  float startAngle, endAngle, radius;

  void createCircleArc1(PVector mousePoint1){
    midPoint = mousePoint1;
  }

  void createCircleArc2(PVector mousePoint2){
    radius = dist(midPoint.x, midPoint.y, mousePoint2.x, mousePoint2.y);

    if(mousePoint2.x < midPoint.x){
      startAngle = PI + asin((midPoint.y - mousePoint2.y)/radius);
    }
    else{
      if(mousePoint2.y > midPoint.y){
        startAngle = -asin((midPoint.y - mousePoint2.y)/radius);
      }
      if(mousePoint2.y <= midPoint.y){
        startAngle = TWO_PI - asin((midPoint.y - mousePoint2.y)/radius);
      }
    }
  }

  void createCircleArc3(PVector mousePoint3){
    if(mousePoint3.x < midPoint.x){
      endAngle = PI + asin((midPoint.y - mousePoint3.y)/radius);
    }
    else{
      if(mousePoint3.y > midPoint.y){
        endAngle = -asin((midPoint.y - mousePoint3.y)/radius);
      }
      if(mousePoint3.y <= midPoint.y){
        endAngle = TWO_PI - asin((midPoint.y - mousePoint3.y)/radius);
      }
    }
  }

  void drawCircleArc1(){
    noFill();
    line(midPoint.x, midPoint.y, mouseX, mouseY);
  }
  void drawCircleArc2(){
    float tempEndAngle = 0;

    if(mouseX < midPoint.x){
      tempEndAngle = PI + asin((midPoint.y - mouseY)/radius);
    }
    else{
      if(mouseY > midPoint.y){
        tempEndAngle = -asin((midPoint.y - mouseY)/radius)
      }
      if(mouseY <= midPoint.y){
        tempEndAngle = TWO_PI - asin((midPoint.y - mouseY)/radius);
      }



    }
    noFill();
    if(startAngle <= tempEndAngle){
      arc(midPoint.x, midPoint.y, radius*2, radius*2, startAngle,
tempEndAngle);
    }
    else{
      arc(midPoint.x, midPoint.y, radius*2, radius*2, startAngle - TWO_PI,
tempEndAngle);
    }
  }

  void drawCircleArc3(){
    noFill();
    if(startAngle <= endAngle){
      arc(midPoint.x, midPoint.y, radius*2, radius*2, startAngle, endAngle);
    }
    else{
      arc(midPoint.x, midPoint.y, radius*2, radius*2, startAngle - TWO_PI,
endAngle);
    }
  }


  String sendCircleArc(){

    String circleArcInformation = "A " + str(int(midPoint.x * pixToMm)) + " "
+ str(int(midPoint.y * pixToMm)) + " " + str(int(radius * pixToMm)) + " " +
str(startAngle/PI) + " " + str(endAngle/PI);

    return circleArcInformation;
  }
}
```

Above and on the previous pages you can see the class for drawing a circle arc. Circle arcs are drawn by pressing the mouse three times or giving in three coordinates with the keyboard (see 5 Number input). The first coordinate is for the middle point of the circle arc. The second is for the radius and the start angle and the third one for the end angle. The circle arc class starts with declaring four variables. One of the type PVector and three of the type float. The one PVector is to store the middle point of the circle. In the other variables I store the radius and the start angle and end angle.

The first function within the circle arc class is the void createCircleArc1(PVector mousePoint1) within this function the middle point will be set equal to mousePoint1.

If the middle point is set the function void drawCircleArc1() can be executed. Within this function a temporary line will be drawn from the middle point until the coordinates of the mouse. The line is drawn to indicate the radius of the circle arc.

If the user has decided what the radius and the start angle should be by pressing the mouse or giving in the second coordinate with the keyboard the function void createCircleArc2(PVector mousePoint2) can be executed. This function starts by calculating the radius of the circle arc. Which is equal to the distance between the middle point and mousePoint2. This can be calculated using the build in dist() function. When the radius is calculated the angle between the positive x-axis through the middle point of the circle arc. The middle point will be the zero point of the axis. The angle between them can be calculated by using the build in asin() function. This will calculate the arcsinus of a sinus. So within the asin() function I need to calculate the sinus of mousePoint2. This is done by dividing the difference of the y-coordinates of the middle point and mousePoint2 by the radius. This will not be the angle you need to draw a circle arc. The angle in the first quadrant will go up from 0° to 90° or from 0 rad to $\frac{\pi}{2}$ rad. In the second quadrant the angle will go from 90° back to 0° or from $\frac{\pi}{2}$ rad back to 0 rad. In the third quadrant the angle will go from 0° to -90° or from 0 rad to -$\frac{\pi}{2}$ rad. In the fourth quadrant the angle will go from -90° to 0° or from -$\frac{\pi}{2}$ rad to 0 rad. The angle you need to draw a circle arc is an angle within the range from 0° until 360° or from 0 rad to 2π rad. Mind that the angle will turn the opposite way than the unit circle. For example: the angle 45° or $\frac{\pi}{4}$ rad will lay in the fourth quadrant and not in the first quadrant (see picture 3). This is because the y-axis in computer vision runs down instead of up. The higher the y-coordinate the lower the point will be. Mind that a circle arc always will be drawn clockwise. To convert the angle from the arcsinus to the angle you need to draw a circle arc can be done by checking in which quadrant mousePoint2 lays. If it lays in quadrant two or three the conversion will be equal to the sum of the angle from the arcsinus and π or 180°. The angle from the arcsinus will always be in radians so you need to add π. In the first quadrant the angle you need will be equal to the difference between 2π and the angle from the arcsinus. In the fourth quadrant the angle you need will be equal to the negative angle from the arcsinus.

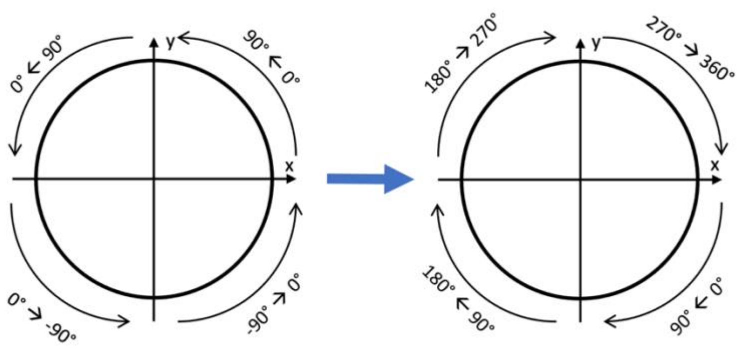

Figure 3: Angle conversion

If the radius and the start angle are set a temporary circle arc can be drawn. To do this the function void drawCircleArc2() can be run. Within this function a temporary end angle will be defined and calculated in the same way as the start angle but using the current coordinate mouse instead of mousePoint2. When that angle is calculated a temporary circle arc can be drawn. A circle arc can be drawn with the build in arc function in processing. This function needs the coordinates of the middle point the height and width, a start angle and an end angle. There is a height and a width because the arc in processing is based on an ellipse. Those two variables in my program will be equal to the diameter of the circle or two times the radius. When you use the arc function the start angle needs to be smaller than the end angle. When an arc goes through the x-axis through the middle point this will not be the case. This can be solved by splitting the drawing of the arc up in two different cases one where the start angle is smaller than the end angle and the circle arc can be drawn normally or one where the start angle is bigger. When the start angle is bigger this can be resolved by subtracting 2π from the temporary angle. Before the splitting up the build in function noFill() will be executed to make sure the circle arc is transparent.

When the user has decided where they want to draw the circle arc permanently the function void createCirlceArc3(PVector mousePoint3) can be executed. Within this function the end angle will be calculated in the same way as the start angle and the temporary angle but with mousePoint3.

When the end angle is set the permanent circle arc can be drawn by running the void drawCircleArc3() function. The function starts with the build in noFill() function to make sure the circle arc is transparent. After that the drawing will be split up in the same to cases as with the temporary arc but this time it is with the permanent end angle.

When the user is finished drawing the figure they want the data of the circle can be send to the penplotter. The penplotter expects a command in the following format: “A X1 Y1 R SA EA”. The capital A is to indicate that the penplotter needs to draw a circle arc. the X1 Y1 are the coordinate of the middle point. The SA is the start angle and the EA is the end angle. The measurements in pixels like the coordinate of the middlepoint and the radius needs to be converted to the millimeters first by multiplying them by 0.35 or pixToMm. After that they get converted to an integer and to a string.

The penplotter expresses the angels as the multiplier of the π if the angle is expressed in radians. So before sending the angles to penplotter they needs to be divided by π. The penplotter expects this values to be floats so they can get converted to strings directly after the division. The conversion to strings is necessary for the serial communication.

4 Creating and drawing the shapes on the screen

```
//Lines
ArrayList<Line> lines;

int circleArcMousePresses = 0;

boolean firstMousePress = false;
char command;

void setup(){
  size(848,700);

  //lines
  lines = new ArrayList<Line>();
}

void draw(){
  background(255);

  //lines
  if((command == 'l') && (firstMousePress == true)){
    lines.get(lines.size() - 1).drawLine1();
    for(int i = 0; i < lines.size() - 1; i++){
    lines.get(i).drawLine2();
    }
  }
  else{
    for(int i = 0; i < lines.size(); i++){
      lines.get(i).drawLine2();
    }
  }

  //circleArcs
  if((command == 'a') && (circleArcMousePresses == 1)){
    circleArcs.get(circleArcs.size() - 1).drawCircleArc1();
    for(int i = 0; i < circleArcs.size() - 1; i++){
      circleArcs.get(i).drawCircleArc3();
    }
  }
  else{
    if((command == 'a') && (circleArcMousePresses == 2)){
      circleArcs.get(circleArcs.size() - 1).drawCircleArc2();
      for(int i = 0; i < circleArcs.size() - 1; i++){
        circleArcs.get(i).drawCircleArc3();
      }
    }
    else{
      for(int i = 0; i < circleArcs.size(); i++){
        circleArcs.get(i).drawCircleArc3();
      }
    }
  }
}
```

```
void keyPressed(){
  //lines
  if(key == 'l'){
    command = 'l';
  }

  if((command == 'l') && (keyCode == BACKSPACE) && (lines.size() > 0) &&
(input == false)){
    lines.remove(lines.size() - 1);
  }
}
```

Above and on the previous page you can see the code for creating and drawing the shapes. Mind that only the line and special cases for the circle arc is shown. The other shapes are like the line.

4.1 Selecting the shape

4.1.1 Shortcuts

The first method the user can use to select the shape they want is by using the keyboard shortcuts. In table 1 you can see which key responds to which shape. The code to check which of those keys is pressed is programmed in the void keyPressed() event. Within this event the word key will be the key that is pressed. When one of the keys is pressed that is linked to a shape a variable named command will be set to the key that is pressed. The variable command is of the type char. The variable type char is used to store one ASCII character.

Pressed key	Shape
l	Line
r	Rectangle
c	Circle
e	Ellipse
a	Circle arc

Table 1: pressed keys and their linked shape

4.1.2 Buttons

The second method the user can use to select the shape they want is by pressing one of the buttons below the drawable canvas. When the mouse is pressed the first check that will happen is that the mouse is in the drawable canvas. The drawable canvas is from 0 to 600 pixels in the y direction. Everything from 600 to 700 pixels will be the user interface with the buttons. When the mouse is pressed in the drawable canvas the user wants to draw a shape and a new shape will be created if the variable command is equal to one of the shapes keys. When the mouse is pressed outside of the drawable canvas the user is probably going to press a button. So the program checks where in the x direction the mouse is located. When it is between 0 to 100 pixels the user has pressed the line button. When it is between 100 to 200 pixels the user has pressed the rectangle button. When it is between 200 to 300 pixels the user has pressed the circle button. When it is between 300 and 400 pixels the user has pressed the ellipse button. When it is between 400 and 500 pixels the user has pressed the circle arc button. In these cases the command variable will be set to the key of the shape.

4.2 Creating or deleting a shape

To store all the shapes that are going to be created I used an arraylist for every shape. Arraylist start out empty but objects can be added or deleted at every time.

To create a shape the user must select the shape they want to draw and after press the mouse the times its needed to draw that shape or giving in the coordinates the times is needed. Because there are two ways to create a shape I have written a function for it. The function starts with checking that it is still plausible to create a shape. This is done with a global variable named upload. The variable is of the type Boolean so it can only store a true (1) or false (0) value. If the value is false and the figure the user has drawn is not uploading the user can still create new shapes. After that a check runs to see which shape needs to be created after that a check runs to see if it is the first step of creating the shape. For all shapes unless the circle arc happens this with a Boolean variable named firstMousePress. If this is false a new object will be added to the list of shapes of that specific shape and the object will be declared as that shape. After that the first create function of that shapes class will be executed with the coordinate of the mouse at the moment it is pressed or with the coordinate the user entered. After that firstMousePress will be set to true. When the user wants to set the second coordinate by pressing the mouse a second time or giving in the second coordinate the same function will be executed but no object will be added to the list of shapes. The function will run the second create function of that shape and put the firstMousePress value to false.

This is for all of the shapes except for the circle arc because this shape needs three mousepresses to be drawn. So I made a separated variable to count the mouse presses named circleArcMousePresses. This is a variable of the type int and will count up during the creating process. On the first mouse press a new object will be added to the circle arcs list. But on the second mouse press the second create function of the object will be executed and the count will go up by one. On the third mouse press the third create function of the circle arc will run and the count will be set back to zero.

If the user wants to delete a shape it is plausible by activating the shape they want to delete and pressing backspace. The shape that is drawn last of that kind will be deleted.

4.3 Draw the shape

If the variable firstMousePress is true the temporary shape of the shape selected should be drawn. So to draw the shapes starts with the check of that. If that is the case I start by drawing the temporary shape of the last shape in the list of that shape by running the first draw function of that shape. After that I loop through the rest of the shapes left in the list and draw them permanently by running their second draw function. If the firstMousePress is false or the shape is not selected I will loop true the all the shapes in that list and draw them permanently by running their second draw function. The above is true for all shapes except for the circle arc because it needs three mouse presses. If the circle arc is active and the circleArcMousePresses variable is equal to 1 I will draw the temporary line by running the drawCircleArc1() function for the last circle arc. After that I will loop true the other circle arcs to draw them permanently by running their drawCircleArc3() function for the rest of them. If the circleArcMousePresses variable is equal to 2 the temporary circle arc will be drawn by running the drawCircleArc2() function for the last circle arc. After that I will loop true the other circle arcs to draw them permanently by running their drawCircleArc3() function for the rest of them. If the circle Arc shape is not selected or the circleArcMousePresses variable is equal to zero I will loop through all the circle arcs and draw them permanently by running their drawCircleArc3() function.

5 Number input

```
//number input variables
int number = 0;

boolean input = false;
boolean firstEnter = false;
float buffer;

void keyPressed(){
  //number input
  int intKey = int(key);
  int digit = 0;

  if((keyCode == BACKSPACE) && (input == true)){
    number = int(number/10);
  }

  if((intKey >= 48) && (intKey <= 57)){
    input = true;
    digit = intKey - 48;
    number = number * 10;
    number += digit;
  }

  if((keyCode == ENTER) && (input == true)){
    if(firstEnter == false){
      buffer = number;
      buffer = buffer/pixToMm;
      number = 0;
      firstEnter = true;
    }
    else{
      PVector coordinate;
      coordinate = new PVector(buffer, number/pixToMm);
      createFigure(coordinate);
      buffer = 0;
      number = 0;
      input = false;
      firstEnter = false;
    }
  }
}
```

My drawing program can be used by giving in a coordinate where the mouse press should be. Input from the keyboard in processing is done by reading in the ASCII character that has been pressed. This character can be converted to the ASCII number linked to the character. For digits this will be in the range from 48 until 57 with 48 being equal to zero and 57 being equal to nine. From 48 the count will go up by one to get the next digit. So 49 will be equal to 1 and so on. Due to this the ASCII number can easily be converted to the digit by subtracting 48 from the ASCII number.

The code to do this is written within the void keyPressed() event. Within the event two new variables of the type int will be declared one will be set equal to the ASCII number of the key and the other one will be used to store the digit. Within that event there will be checked if the ASCII of the pressed key is in the range of 48 until 57. If that is the case a boolean variable named input will be set to true to indicate that the user is typing in a number. This is done so the enter and backspace key can have two purposes. If the variable input is set the variable digit will be set equal to the subtraction of the ASCII number and 48. After that the global variable of the type integer named number will be multiplied by 10 to make place for a new digit. After that the given in digit will be added to the number.

If the user want to delete a digit from the number than is that plausible by pressing the backspace key. When this key is pressed and input is true the variable number will be divided by 10 and converted to the type integer. Because integers are made to store whole numbers will this get rid of the last digit. This check is coded above the check of the pressed keys ASCII number is between 48 and 57 because the ASCII number of backspace equal is to eight and this interfered with the number. When the user pressed eight and it was below the code would remove a digit instead of adding the digit.

When the user finished typing the x-coordinate they can submit it by pressing enter. When the input variable is true there will be checked if it is the first time that the user pressed enter by checking if the global variable of the type boolean named firstEnter is false. When this is the case the user want to submit the x-coordinate. A global variable of the type float named buffer will be set equal to the number. After that the buffer variable will be converted to pixels by dividing it with 0.35 or the global variable pixToMm. After that the variable number will be set equal to zero and the variable firstEnter will be set to true. When it is the second time the user presses enter and the variable input is equal to true the figure can be created by making a new PVector with the two coordinates. The y-coordinate needs to get converted to pixels first by dividing it by 0.35 or the global variable pixToMm. After that a figure can be created by running the createFigure() function with the PVector created with the coordinate. After that the variables number and buffer can be set to 0. After that the variables input and firstEnter can be set to false.

6 Serial communication

```
boolean upload = false;
int count = -1;

//serial communication
import processing.serial.*;
Serial myPort;

boolean machineReady = false;

void serialEvent(){
  machineReady = true;
  count += 1;
}

void draw(){
  //upload
  if((upload == true) && (machineReady == true)){
    int circleArcCount = lines.size() + rectangles.size() + circles.size() +
ellipses.size();
    int ellipseCount = lines.size() + rectangles.size() + circles.size();
    int circleCount = lines.size() + rectangles.size();

    if((count >= circleArcCount) && (count < (circleArcCount +
circleArcs.size()))){

      myPort.write(circleArcs.get(count - circleArcCount).sendCircleArc());
    }

    if((count >= ellipseCount) && (count < (ellipseCount +
ellipses.size()))){

      myPort.write(ellipses.get(count - ellipseCount).sendEllipse());
    }

    if((count >= circleCount) && (count < (circleCount + circles.size()))){

      myPort.write(circles.get(count - circleCount).sendCircle());
    }

    if((count >= lines.size()) && (count < (lines.size() +
rectangles.size()))){

      myPort.write(rectangles.get(count - lines.size()).sendRectangle());
    }

    if(count < lines.size()){
      myPort.write(lines.get(count).sendLine());
    }

    machineReady = false;
  }
}
```

When the user is finished drawing the figure they want the figure can be drawn by the penplotter. To do this all the data of the shapes need to be transferred to the penplotter. A standard Processing program can't communicate with an Arduino. To do this a library named processing.serial.* needs to be imported. After the library is imported a serial port named myPort gets created.

The user can indicate that the figure is done by pressing the enter key if they are not giving in data of a coordinate. When this happens the serial connection between the drawing program and the penplotter can be made. Also the global variable of the type boolean named upload will be set true so the rest of the program knows that the data can be transferred. The penplotter sends out a signal to the program when it is ready to receive data. This signal can the program read by using the void serialEvent() event. The code within this event will only be executed when there is a serial event on the serial port. Within this code a global variable of the type boolean named machineReady will be set to true to indicate that the next shape can be send to the penplotter. To keep track of which shape needs to be send to the penplotter a global variable of the type integer named count will count up by one every time a serial event happens. Note that the variable count starts out as -1 so the first shape will be indexed 0.

Within the void draw() loop a check will be happen if the variables upload and machineReady are true. When this is the case the next shape can be transferred to the penplotter. The shapes will be transferred in the following order: lines → rectangles → circles → ellipses → circle arcs. To know how high the count needs to be to start sending a specific shape three local variables will be created of the type integer and named circleCount, ellipseCount and circleArcCount. These variables will be set equal to the sum of the sizes of the arraylists of the shapes before them. Lines will be drawn first so they start at zero and rectangles will start after lines.

After the calculation of the shapes starting count a check can happens to see in which range the count is. The range of each shape will start from that shapes count and end at the sum of that shapes count and the size of their arraylist. If the count is within the range of a shape these shapes data will be transferred to the penplotter. This happens by getting the shape at the count minus the shapes count and running the send command of that shape. This will return a string with the data and this data will be written to the penlotter. After this action the variable machineReady will be set equal to false. When the machine is done drawing the shape it will send back a signal and the variable will be set back to true within the serialEvent().

7 Conclusion

I did It. I made my drawing program and made my paper within the time of the death line although it was a tight timing for such a project.

I learned that it is important to choose a project of the right size for the right paper. When I was writing this paper I got the feeling that the subject was to big for the exercise.

The programming went fairly smooth with the exceptions of some faults but I got to a working program quick. Writing the paper itself went not so smooth it was hard for me to decide how much in detail I had to write it but I think I found a good balance. Another thing was that this was my first paper I have written in English.

At the end I am happy with the way my paper turned out.

8 YouTube video

Link → https://www.youtube.com/watch?v=T-VG_1N2yLA