CM2104 – Computational Mathematics – Plotting Circles - C21050251 – Theodor Baur

Task 1

In order to initialize tasks 2-6 the user must press "Run construction". If the user has a text file with correctly formatted points located in the directory, then they can run task 2-6 from preselected points. There are 6 checkboxes that can hide and show various geometric elements on the canvas by changing the visibility of the *plot()* value.

Task 2

In order to operate this task requires 2 sets of points, the centre and circumference of the first circle. If the task is run through the "Run construction" button then the user will select these points by clicking on the graph twice, otherwise they are selected from a text file. These two sets of points are plotted on the graph, they are then passed into the function *firstCircle*. This returns a set of points on the circle to plot, and the radius by setting the *r* to the distance between the two sets of points and using the parametric circle equations $x = x_0+r\cos(\theta)$ and $y = y_0+r\cos(\theta)$ to get the sets of points to plot.

Task 3

A new set of points are acquired from the user and plotted, representing the centre of the second circle. The second centre points and the centre and radius of the first circle are passed into *secondCircle*. This function finds the distance between the first and second circle's centre points and subtracts the first circle's radius to find the second circle's radius. The second radius is used to find a set of x and y points using parametric circle equations, then the x and y points are plotted.

Task 4

A new set of two points are acquired from the user and plotted, representing the closest point on the closest circle to each one and are plotted on the graph. A variable called *circlePointsArray* needs to be created. This contains the centre and radius of the first and second circle, with each circle on a different row in the form [xCenter, yCenter, r; xCenter, yCenter, r]. *circlePointsArray* and the new set of points are passed into *closestPointCircle*. For each row the function finds the parametric equations for the line that passes through the centre of the current circle and the new set of points chosen. The line equation, and radius and centre of the circle are passed through *circle_imp_line_par_int_2d* (Burkardt 2005), returning the intersection points which are added to an array.

For each point in the array, the distance between the intersection point and the initial user selected points are calculated and the intersection with the smallest distance is returned and plotted. This is repeated for another set of user selected points.

If the two intersection points lie on the same circle, then a prompt is shown to run the construction again from the start to work correctly. This works by finding the distance between the intersection points and the centre points of each circle. If the distance between the first circle's centre and the intersection point is the same as the first circle's centre and the second intersection point or if the same is true with the second circle's centre, then the error will be displayed.

A bisector between the first and second circle's centre is drawn. The two centre points are passed through *drawBisector*. This function finds the midpoints by adding the two x points together and halving them, then repeating this for the y points. The parametric equation for the line that passes through both centres is found by passing the centre points through *line_exp2par_2d* (Burkardt 2005). The bisector's parametric equation is then calculated by swapping the calculated f and g values and multiplying the new g value by -1. The x_0 and y_0 value are set to the midpoint and a set of points along this line are calculated, returned, and plotted.

A new set of two points are acquired from the user and plotted, these points are used to find the closest point on the newly plotted bisector. The new points and the set of bisector values are passed into *closestPointLine.* The first and second pair of x and y values are passed into *line_exp2par_2d* (Burkardt 2005) to find the parametric equation of the bisector. A new line equation that is

perpendicular to the bisector and passes through the user points is made by inverting the bisector line by swapping the f and g values and multiplying the new g value by -1. The x_0 and y_0 values for the second line are set to the new user points and both lines are passed through *lines_par_int_2d* (Burkardt 2005), returning the intersection point which is then plotted

Two new bisectors are created and plotted using the same *drawBisector* function. The first bisector is between points 1 and 2, the closest points on each circle's circumference to a user selected point. The second bisector is between points 1 and 3, 3 being the closest point on the first bisector to a user selected point. The intersection between these new bisectors is calculated and plotted by passing the list of values for the two new bisectors into *lineIntersection*. This function finds the parametric equations for both bisectors by passing the first 2 pairs of x and y values of each bisector into *line_exp2par_2d* (Burkardt 2005). It then finds the intersection point by passing the parametric equations into *lines_par_int_2d* (Burkardt 2005). The intersection is plotted and used to represent the centre of a third circle where either point 1, 2 or 3 can be used as a point on the circumference. The third circle centre and a circumference point are passed through *firstCircle* to get a range of values that are found on the third circle. The values are then plotted.

Task 5

The intersections between the first circle and the third circle, and the second circle and the third circle are plotted. This is done by passing the radius and centre point for each circle into *circles_imp_int_2d* (Burkardt 2005). The points are then plotted.

Task 6

A transformation that puts the centre of the third circle on the origin and the first bisector parallel to the x axis must be created. To form the translation matrix a 3x3 homogenous coordinates matrix is used, where dx = 0 – circle 3 x value, and dy = 0 - circle 3 y value. To form the rotation matrix, the list of values used to plot bisector 1 are passed into *rotateBisectorMatrix*. This function calculates the angle between the bisector and the x axis using trigonometry by finding the difference between each consecutive x and y value, then using each difference as a triangle side length. The theta value is found by multiplying the angle by -1 and is inserted into a 3x3 homogenous coordinate rotation matrix. Both matrices are multiplied to create a combined matrix, all 3 matrices are printed to the command line.

To apply the transformation to all points, the x value and y value for each point is put into the form [x;y;1] and multiplied by the combined matrix and replotted. For each geometric element, each value in the list of x and y values is put into the same format and multiplied by the combined matrix and replotted.

Task 7

The user is prompted by a question dialog box, asking if they would like to save their inputted values to a text file. While the program has been executing, if the user clicks on the graph to select their points, the points chosen are automatically appended into an array, if the user wants to save their points they select a location and file name, then this is appended into a text file.

If the user wants to run the construction again using values from this text file, then they need to click the "Load .txt file" to start. The user then selects the directory and file name and saves the points, provided the x and y points are separated by a space and each pair separated by a row.

Task 8

After the user has finished running the construction up to Task 7, they have the option to rotate all elements on the axis using a dial. The initial angle of the elements is set to 0 on initialisation, then with every change, the difference between the initial angle and the selected angle is calculated and applied to a rotation matrix. The initial value is then set to the selected angle.

If the user wants to take a screenshot of the window then they can click the "Screenshot" button. This captures a screenshot and saves it in the directory with the name "screenshot_yymmdd_HHMMSS".

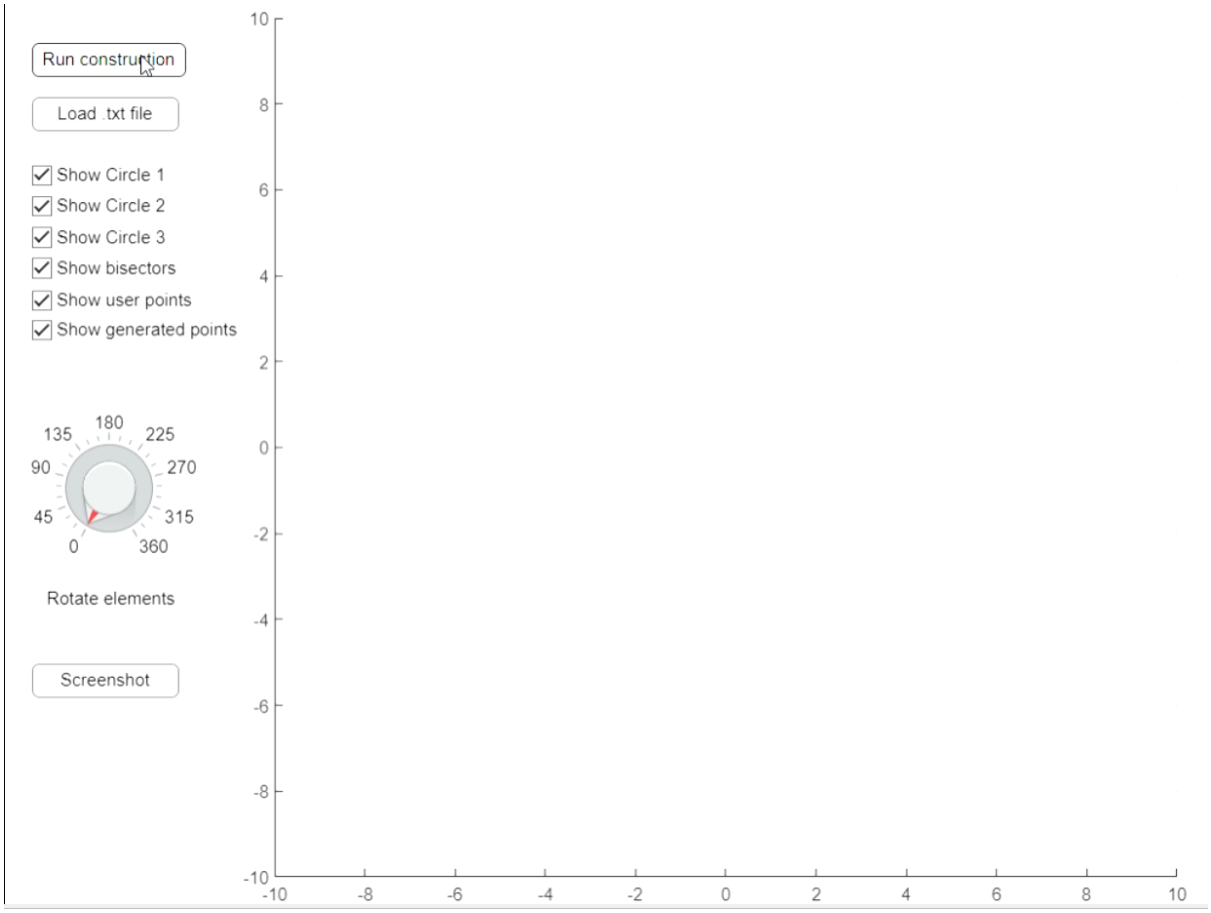

Figure 1: "Run construction" button clicked

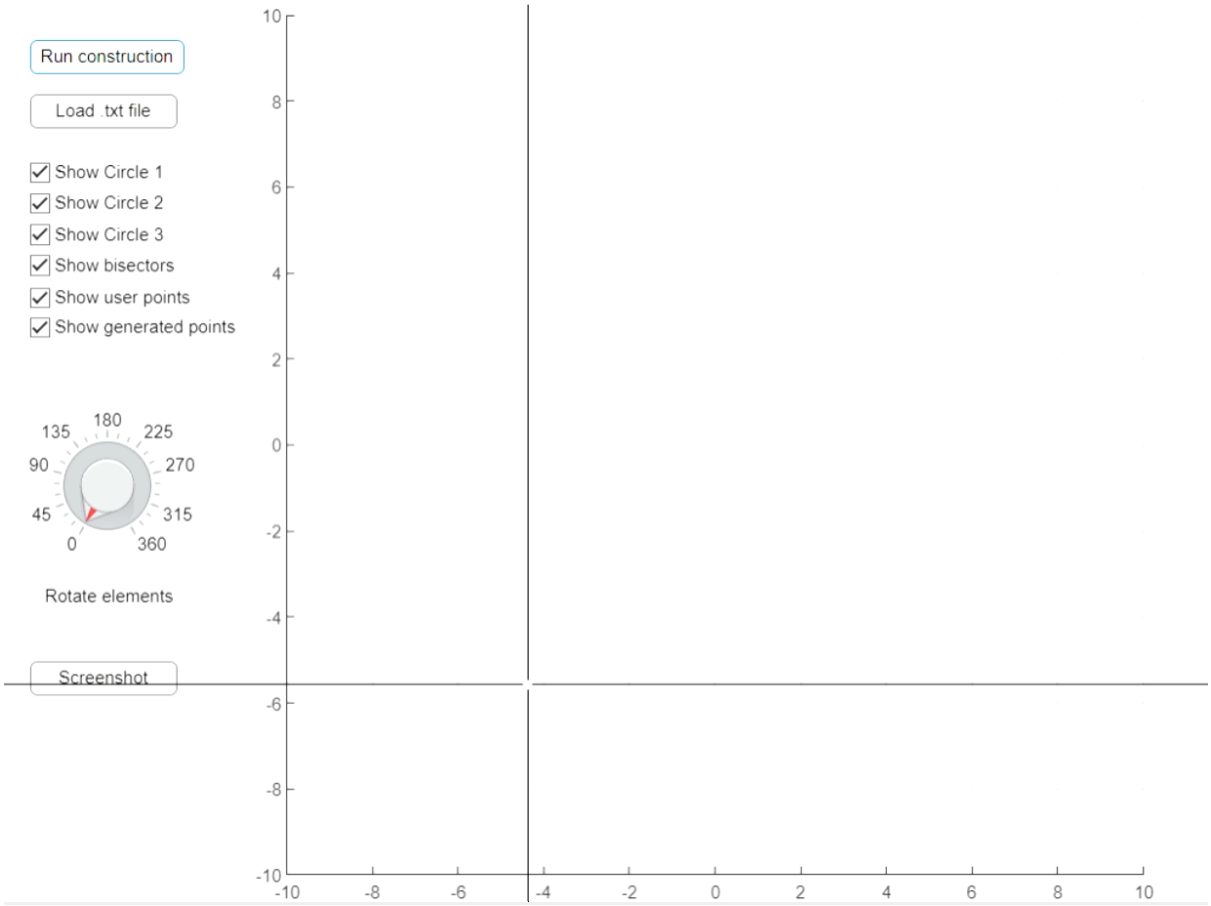

Figure 2: User selects centre of first circle

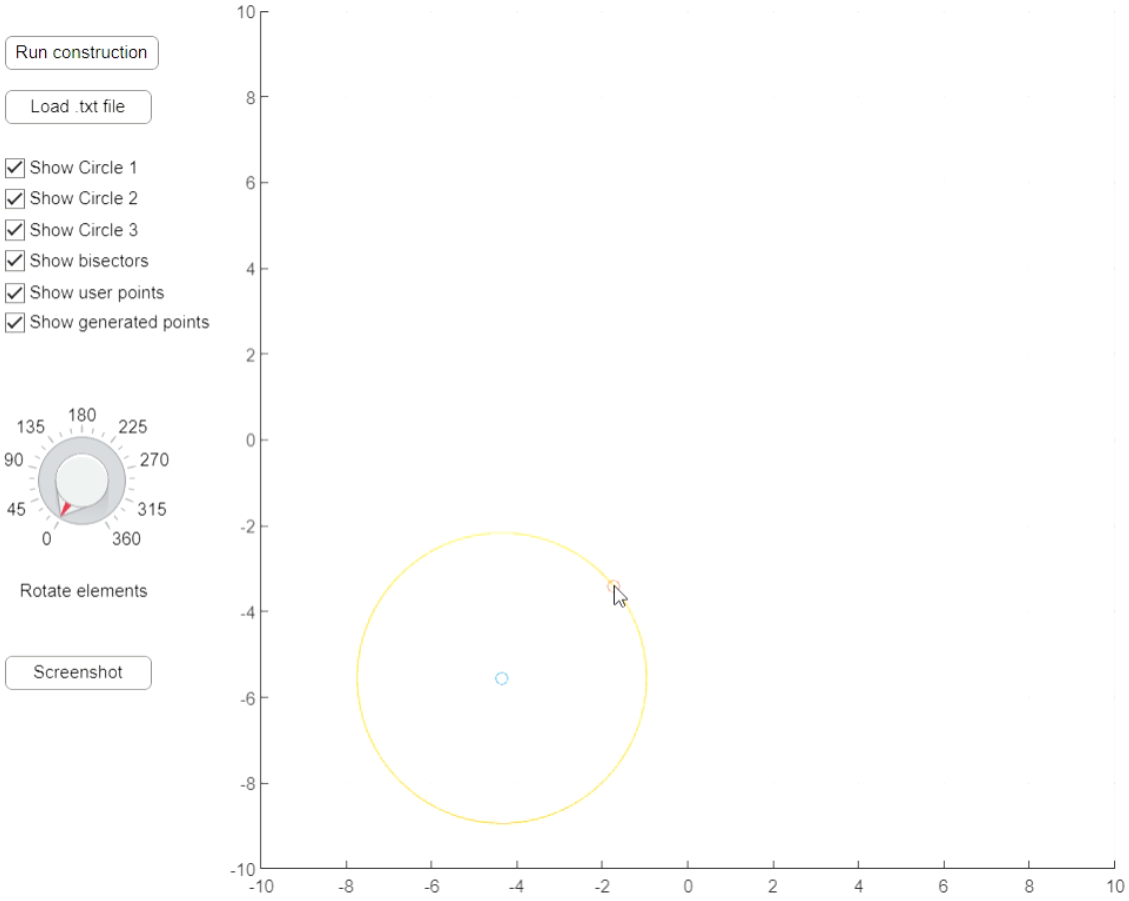

Figure 3: User selects circumference of first circle

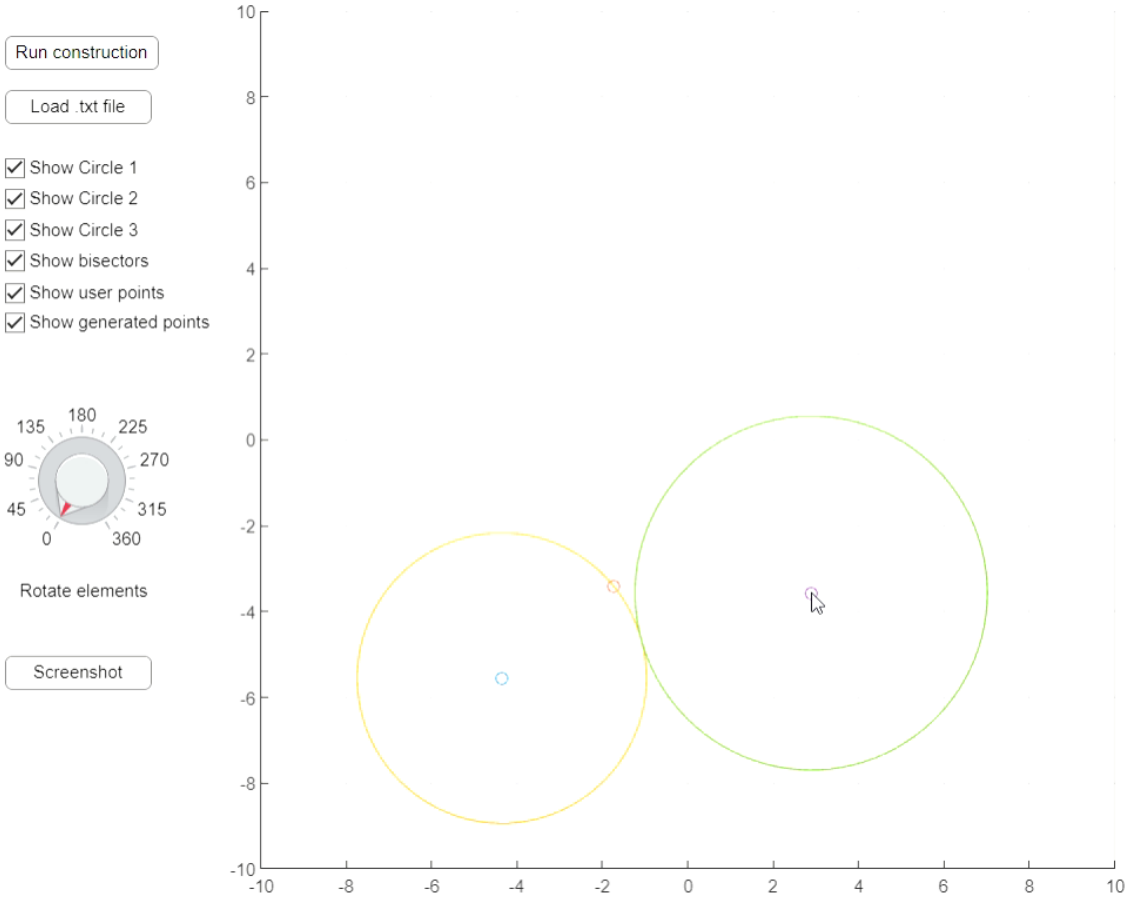

Figure 4: User selects centre of second circle

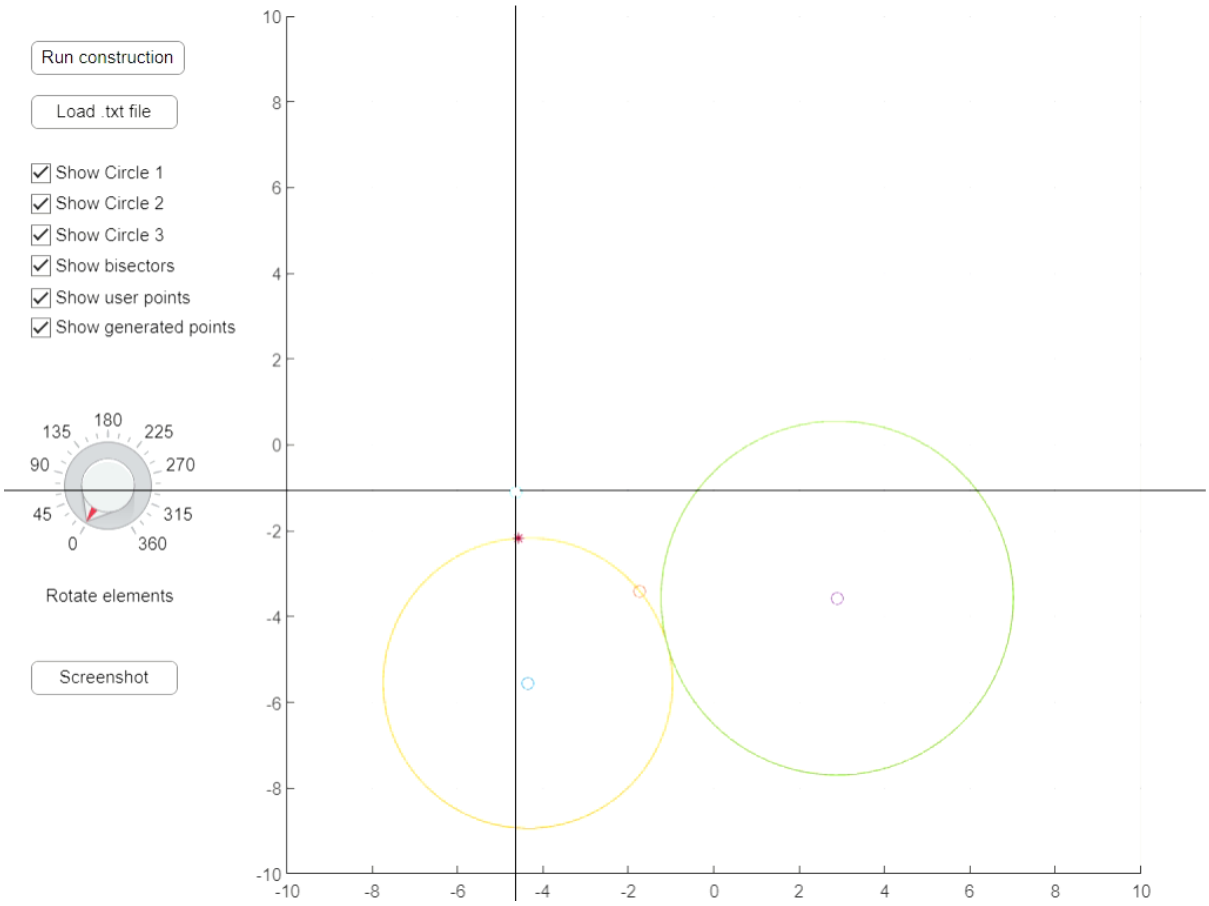

Figure 5: User selects point closest to point on one circle

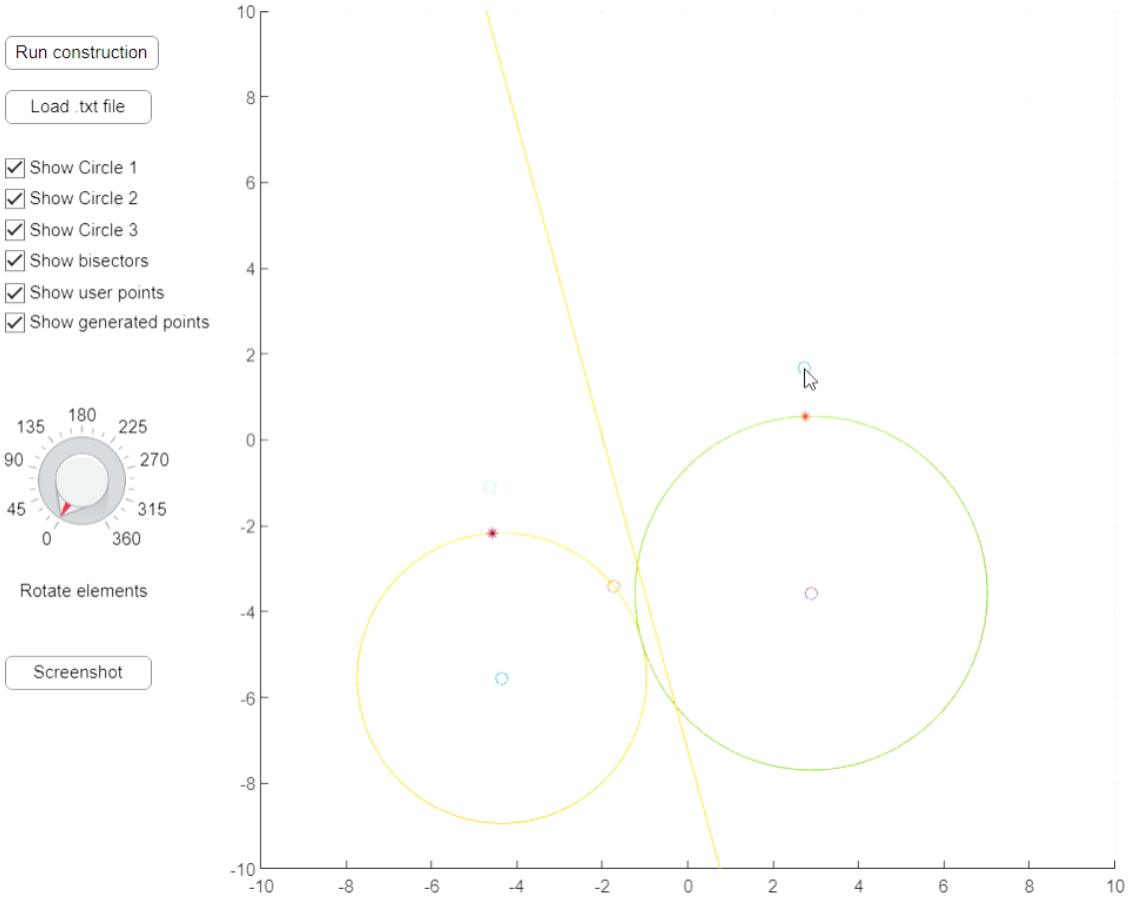

Figure 6: User selects point closest to point on other circle

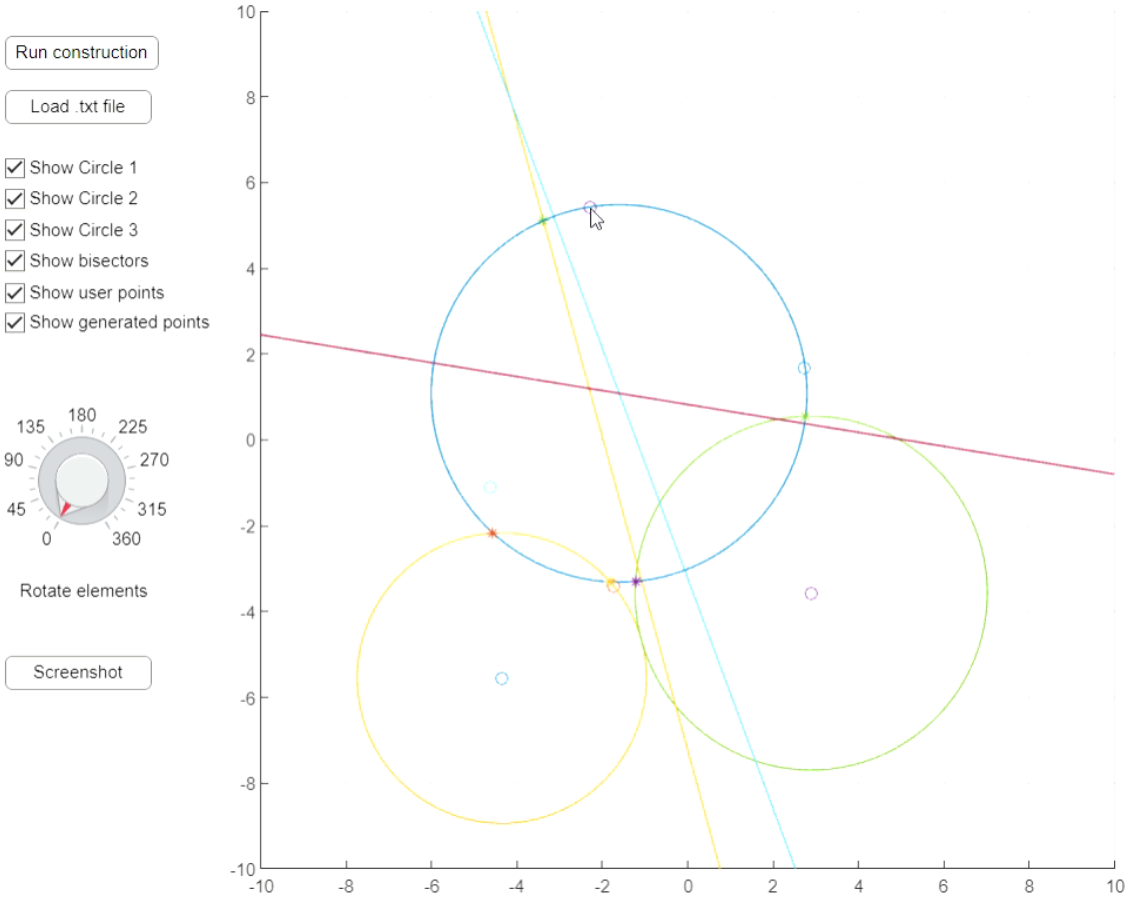

Figure 7: User selects point closest to point on bisector

```
transformThirdCircleOriginMatrix =

    1.0000         0    1.4593
         0    1.0000   -3.2564
         0         0    1.0000


rotationMatrix =

    0.4293   -0.9032         0
    0.9032    0.4293         0
         0         0    1.0000


combinedMatrix =

    0.4293   -0.9032    3.5675
    0.9032    0.4293   -0.0799
         0         0    1.0000
```

Figure 8: Transformation matrices printed to command line

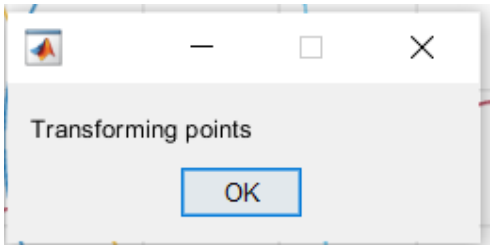

Figure 9: Prompt telling user that the elements are being transformed

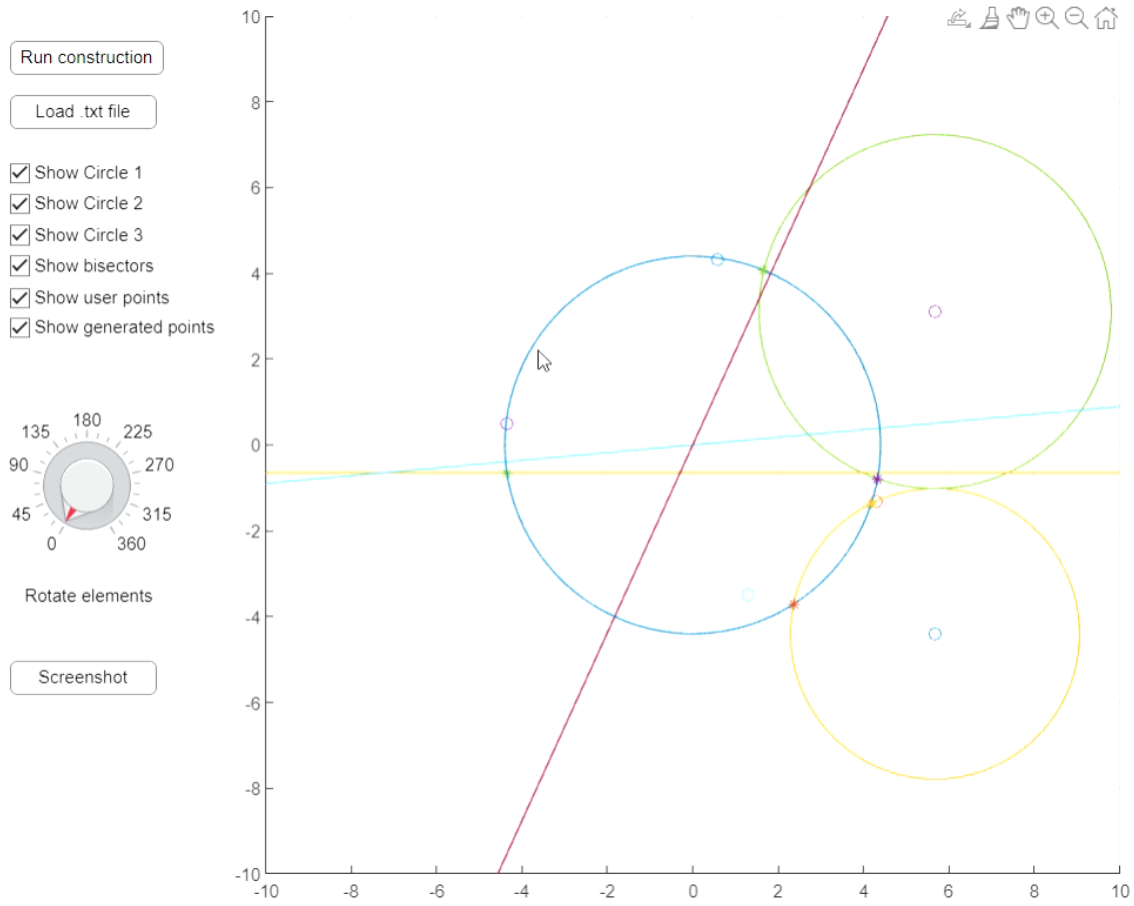

Figure 10: Transformed elements

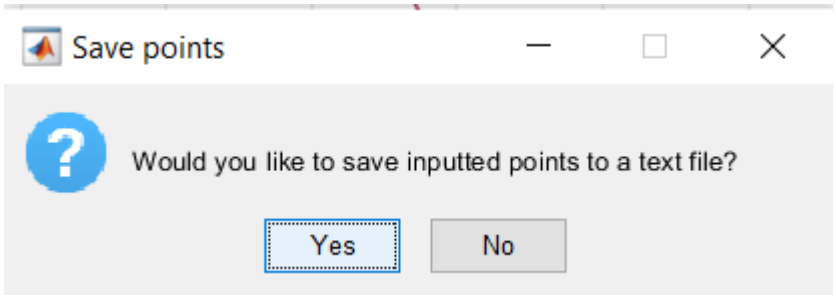

Figure 11: User asked if they would like to save points to file

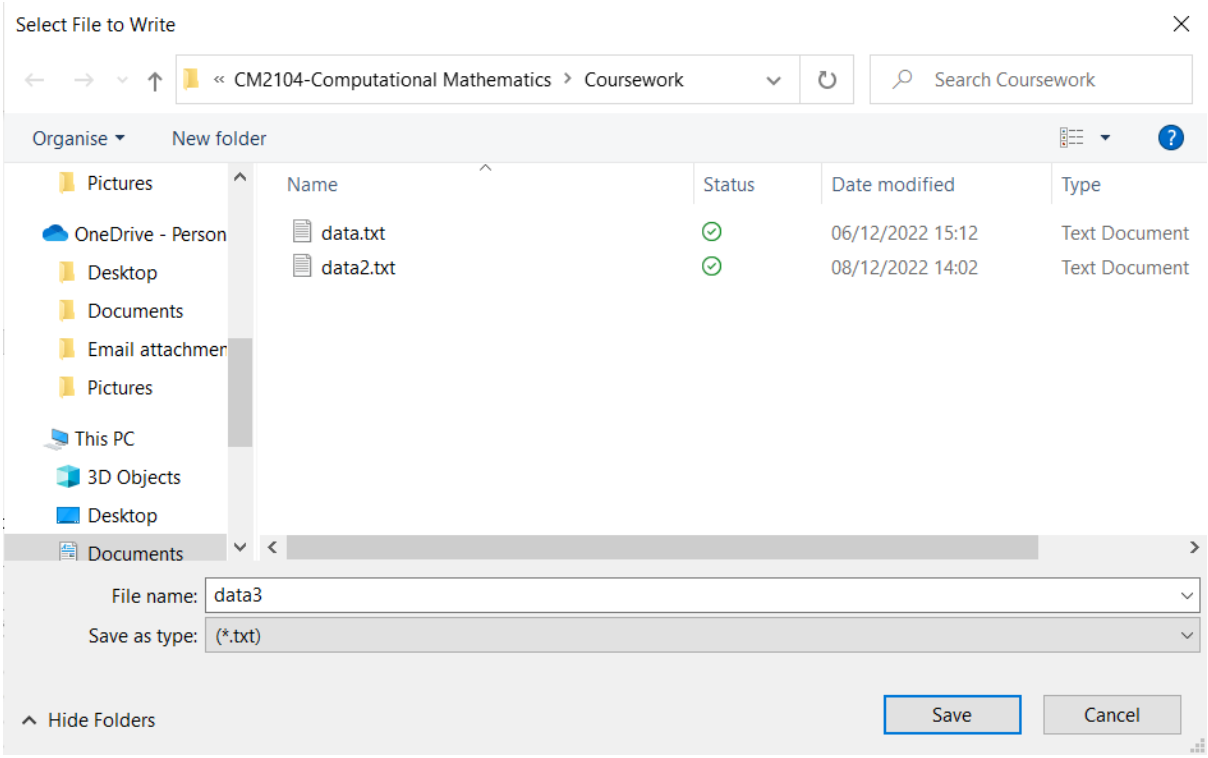

Figure 12: User saving points as "data3.txt"

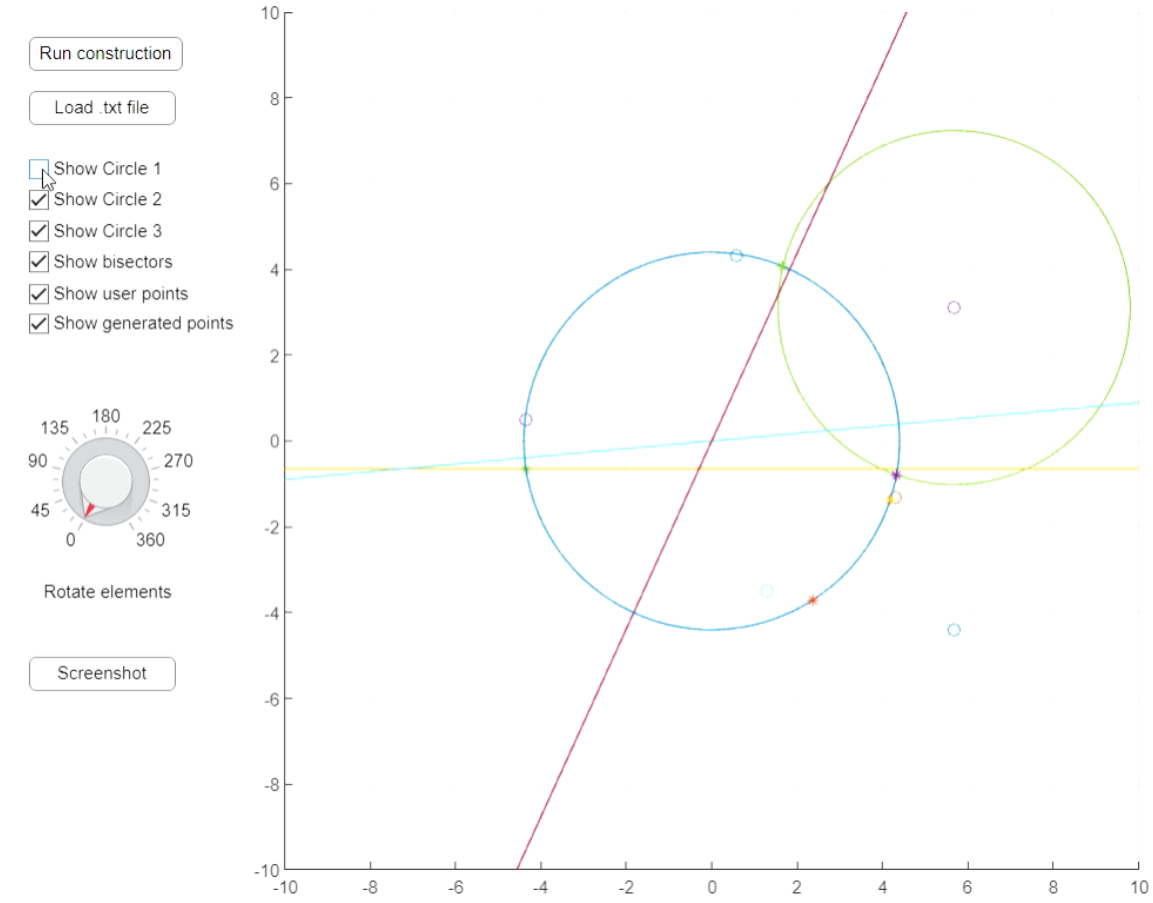

Figure 13: User hides Circle 1

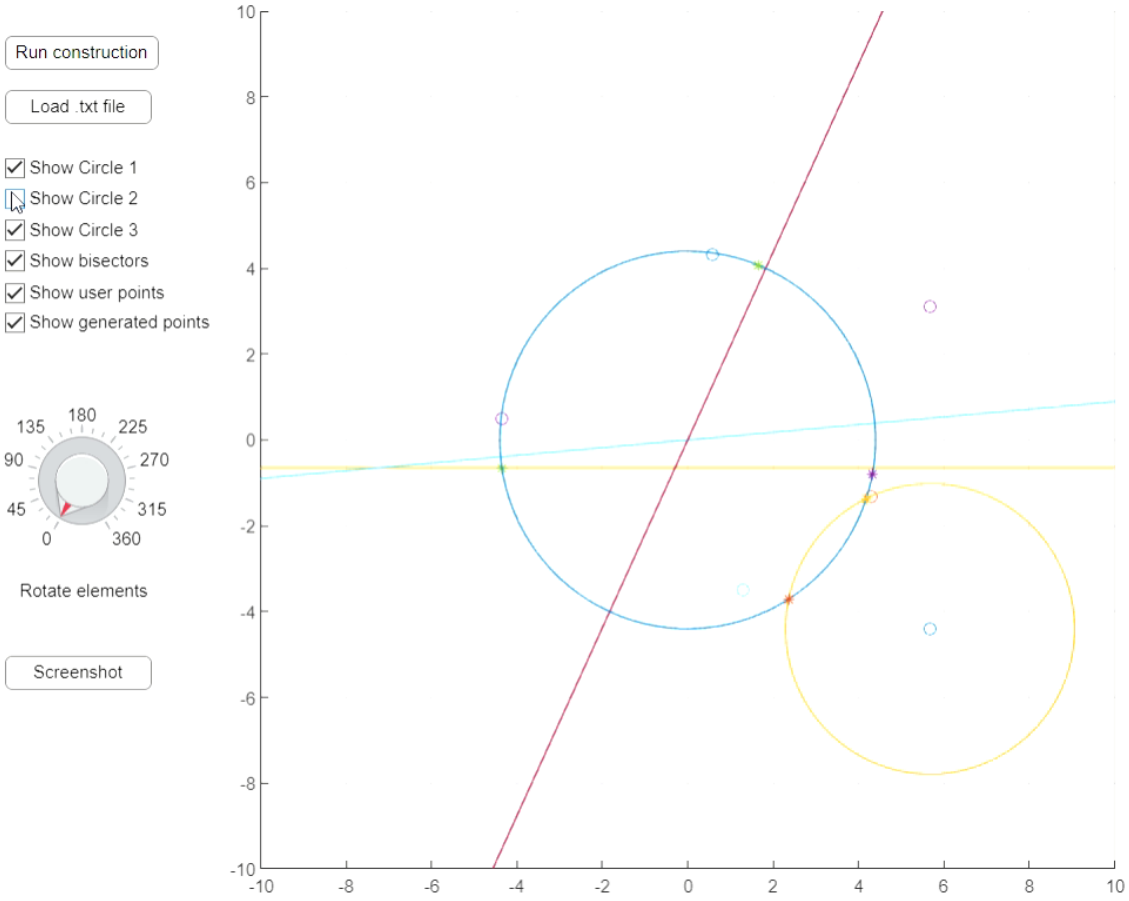

Figure 14: User hides Circle 2

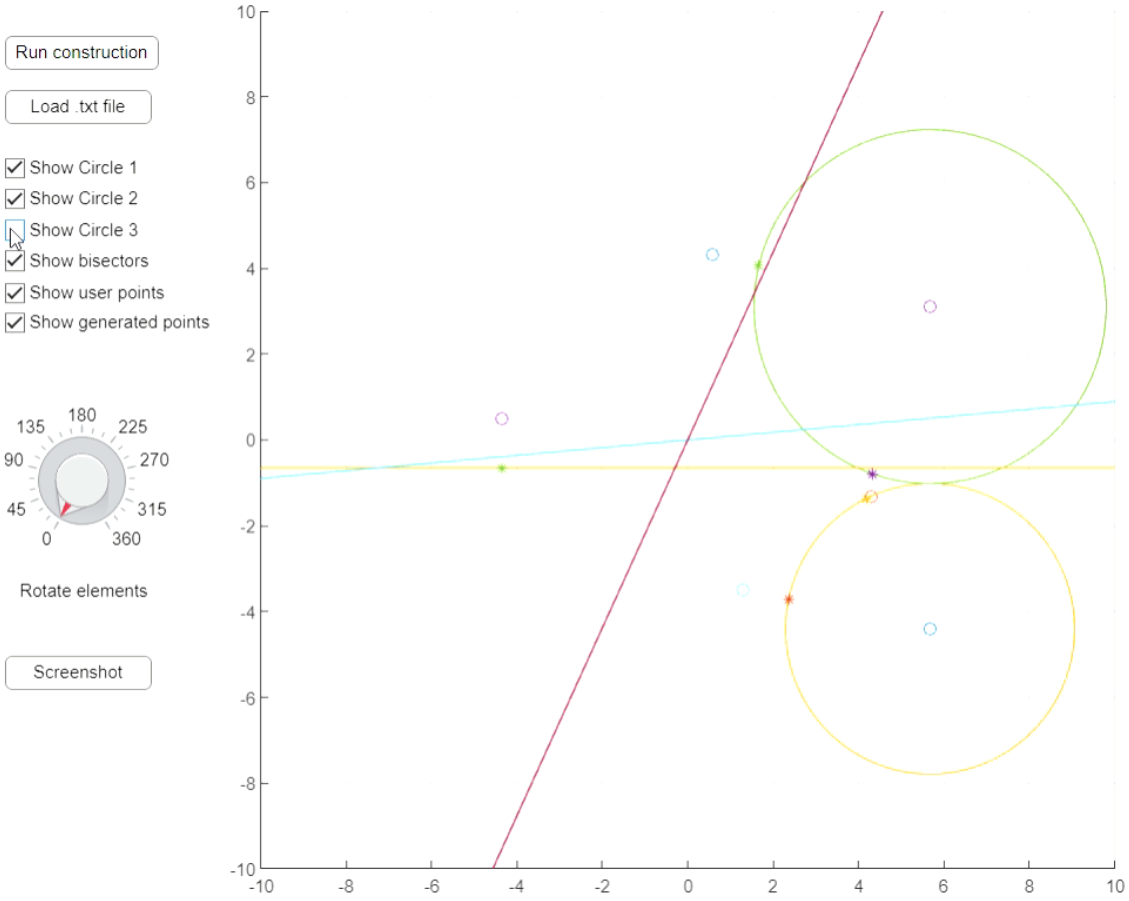

Figure 15: User hides Circle 3

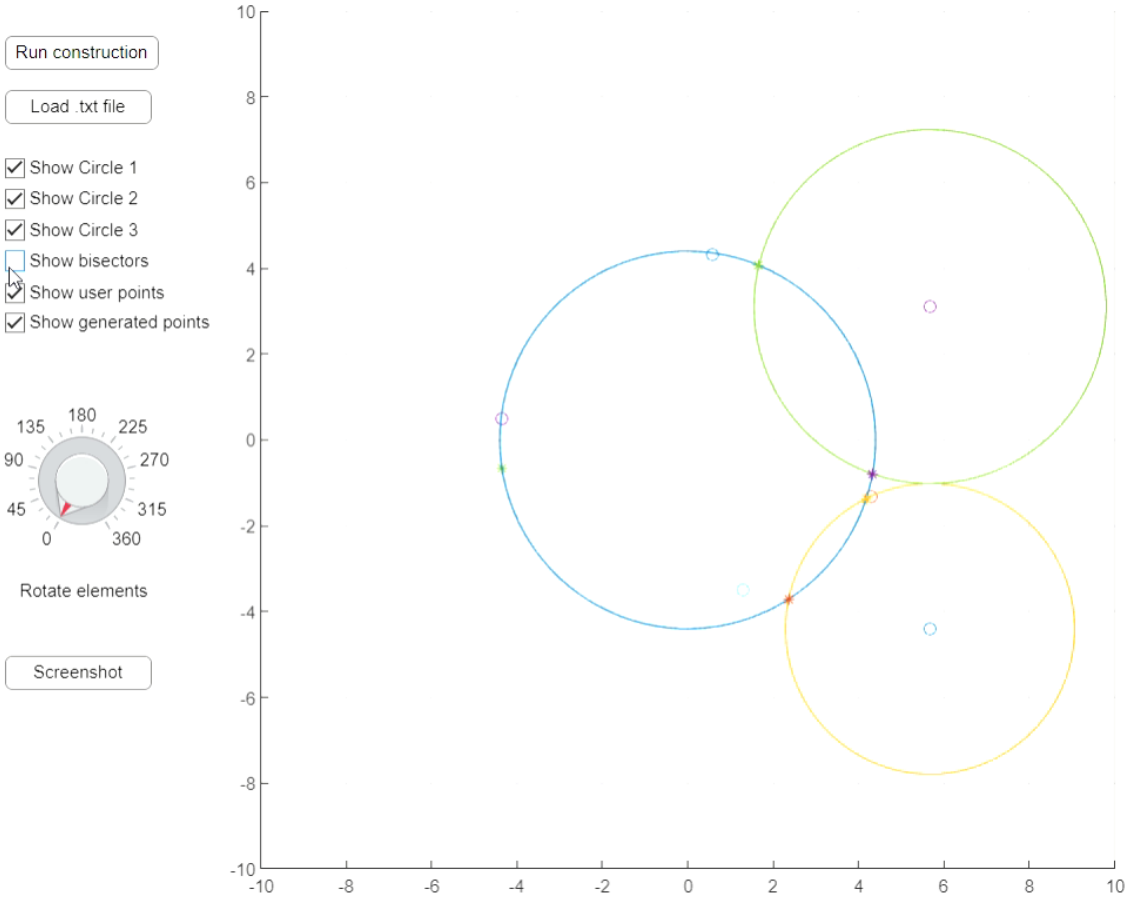

Figure 16: User hides bisectors

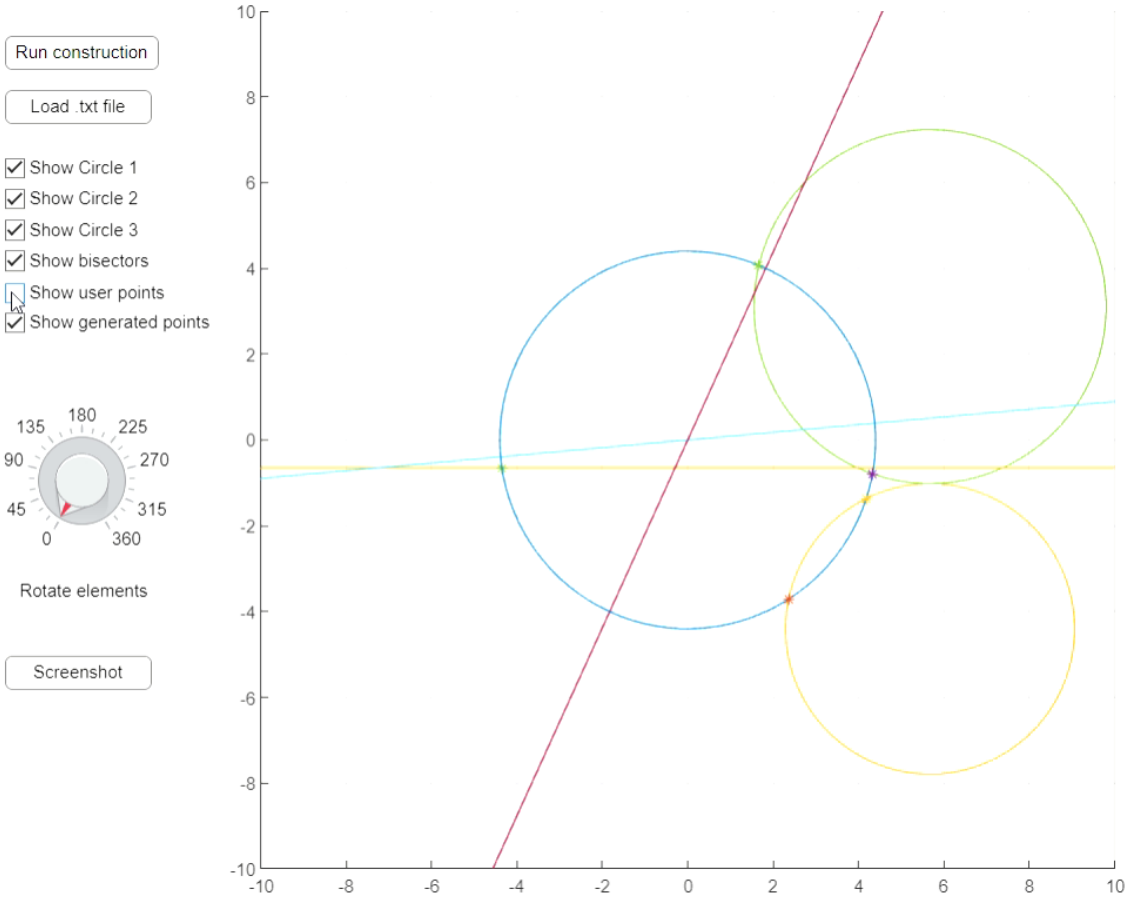

Figure 17: User hides user points

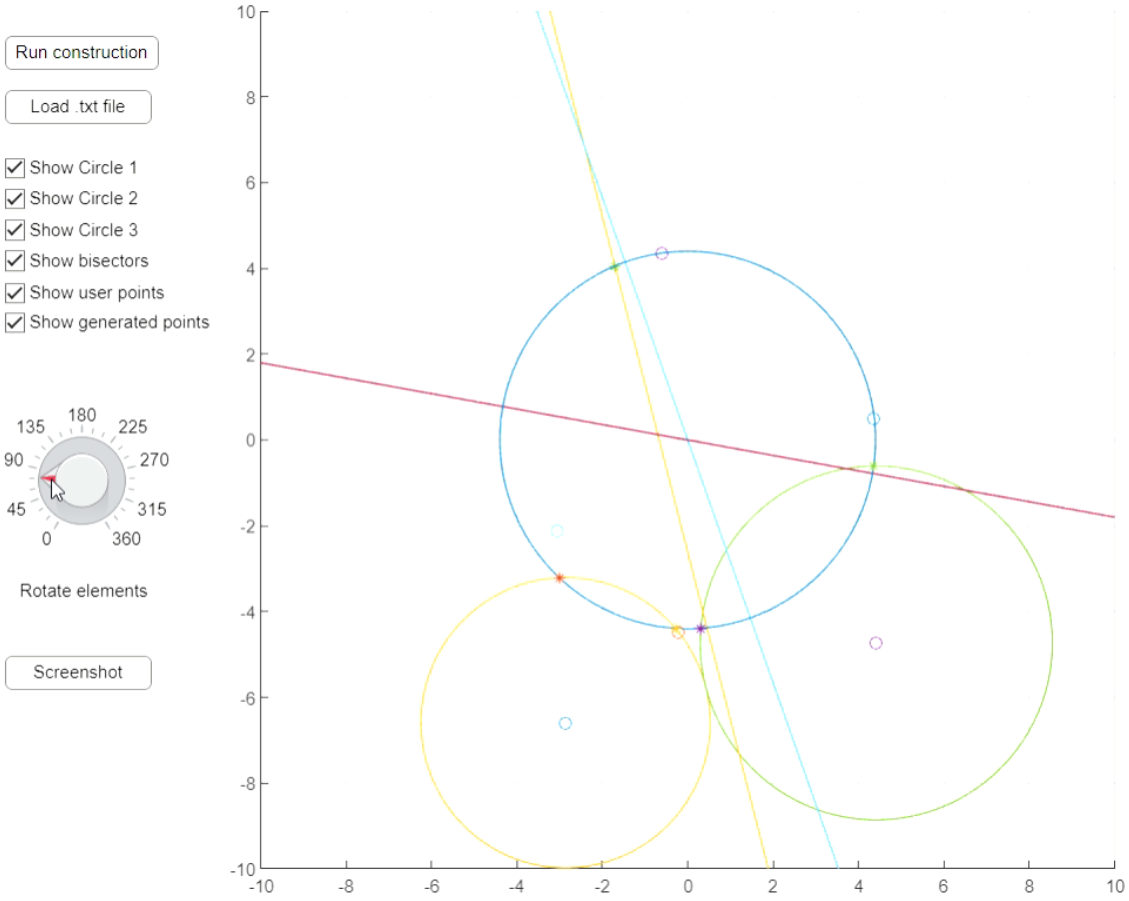

Figure 18: User hides generated points

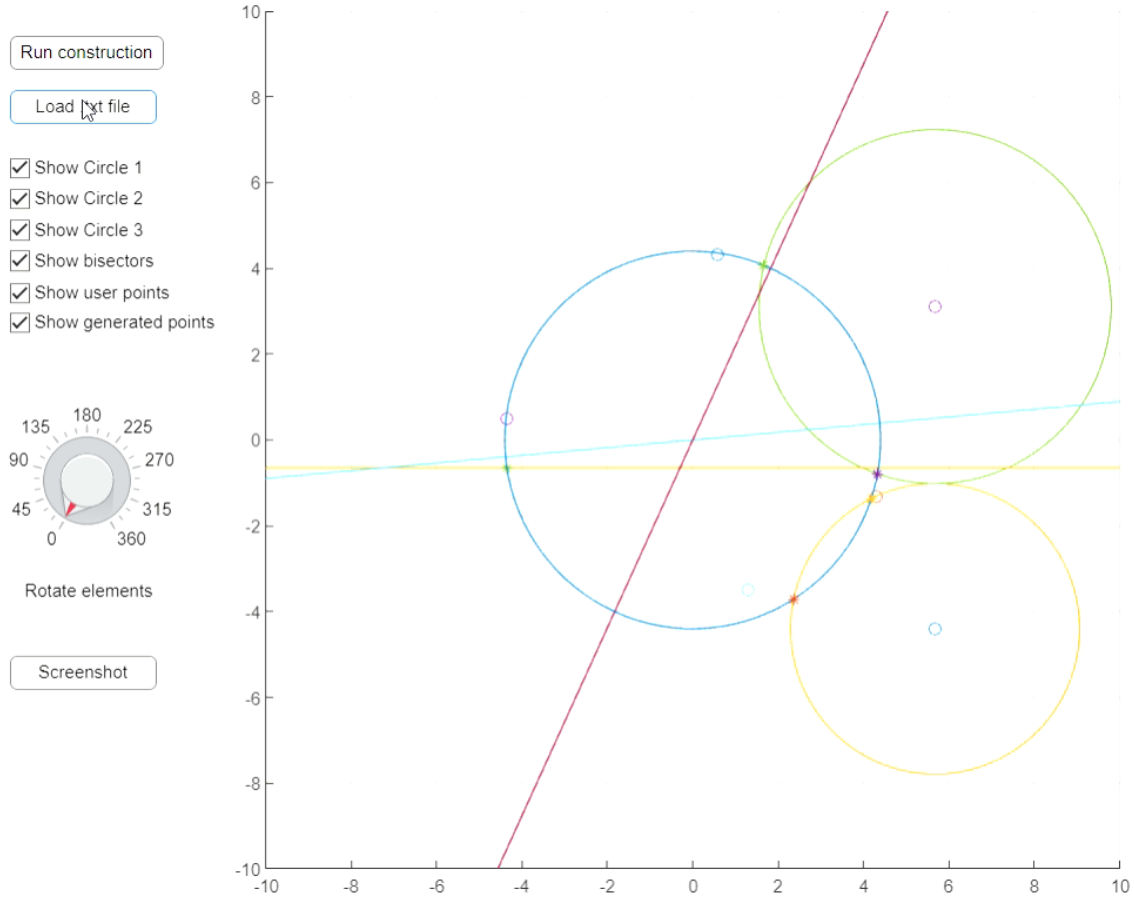

Figure 19: User clicks "Load .txt file"

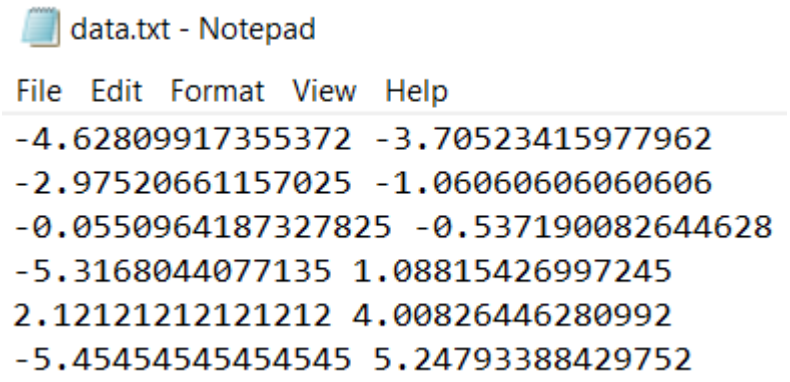

Figure 20: Coordinates in "data.txt"

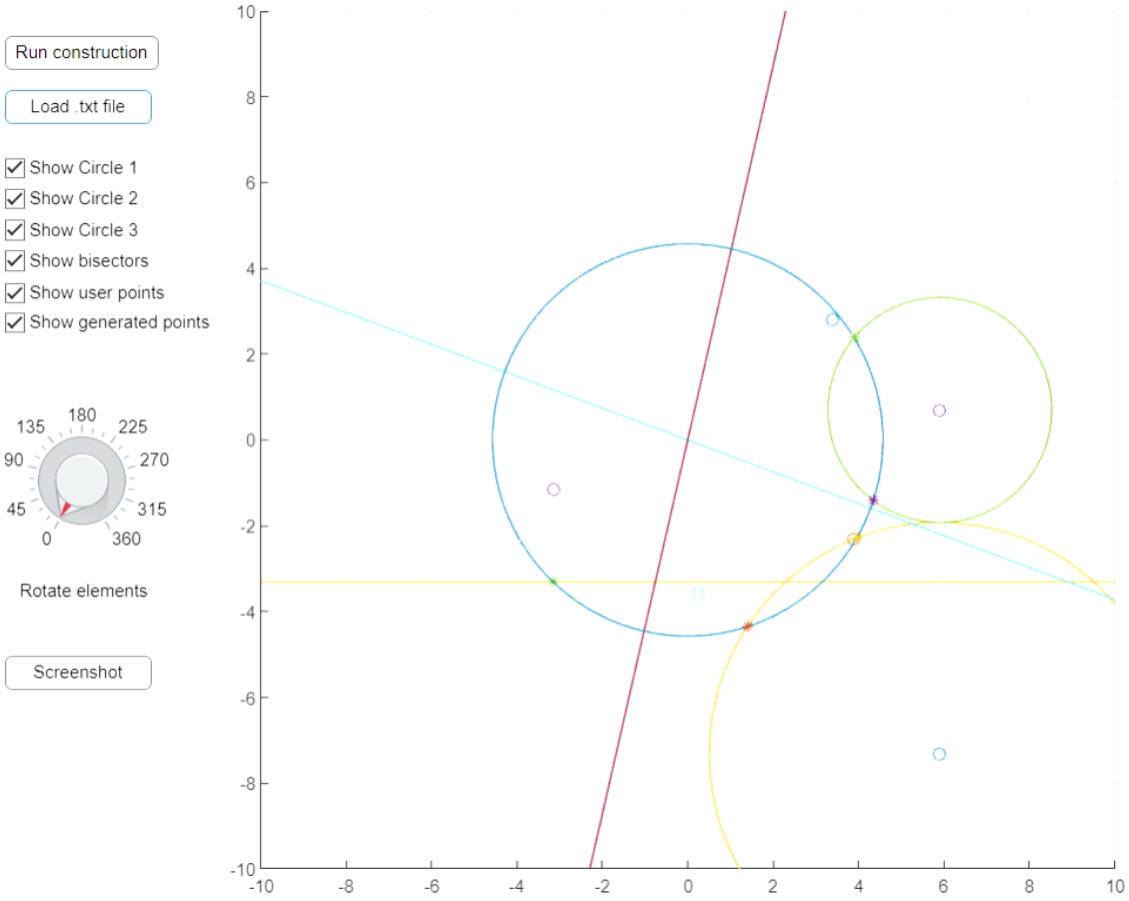

Figure 21: "data.txt" loaded

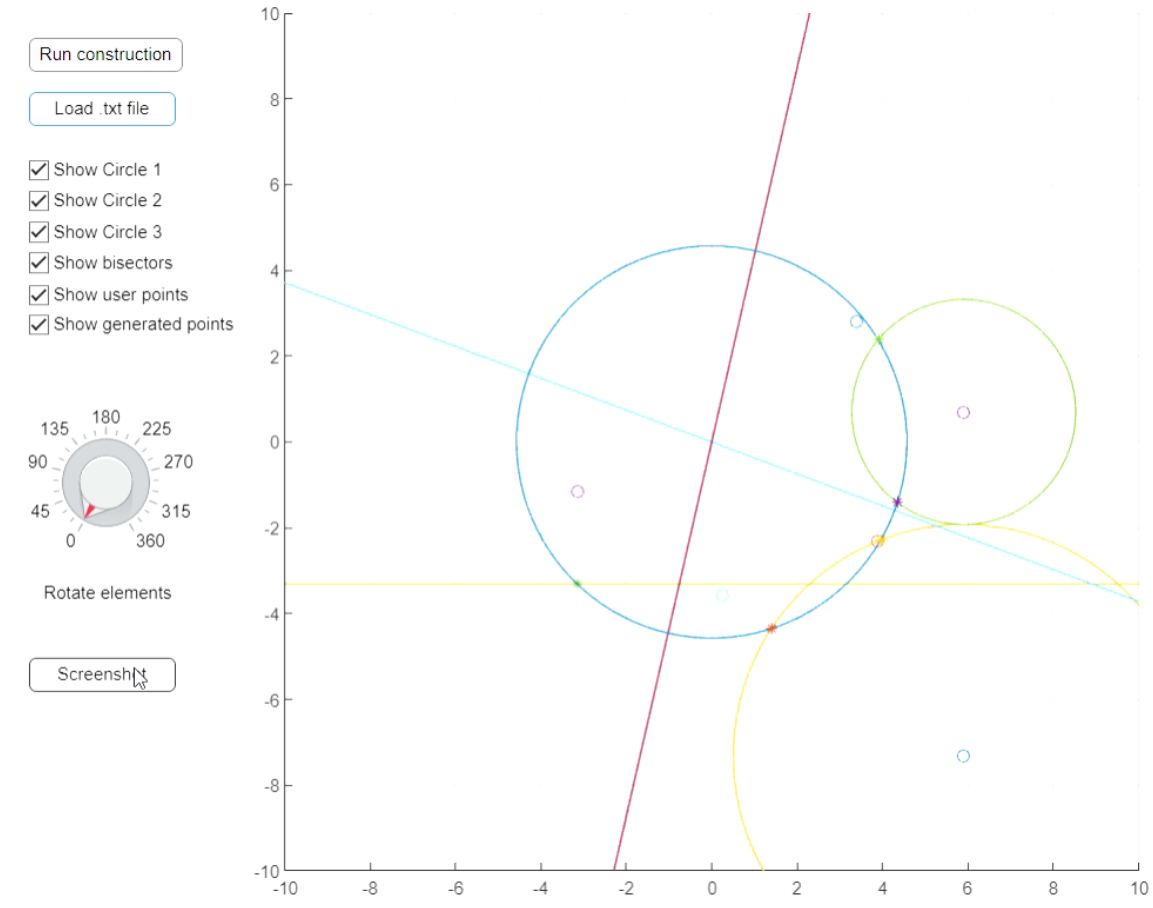

Figure 22: User clicks "Screenshot" button

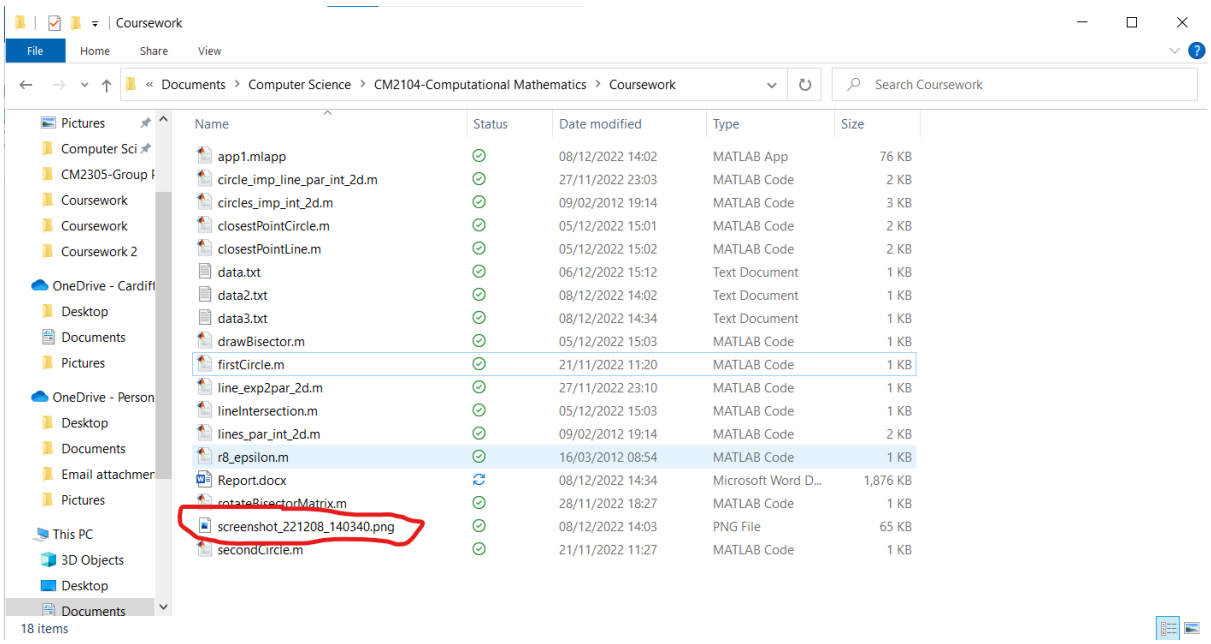

Figure 23: Screenshot saved in directory

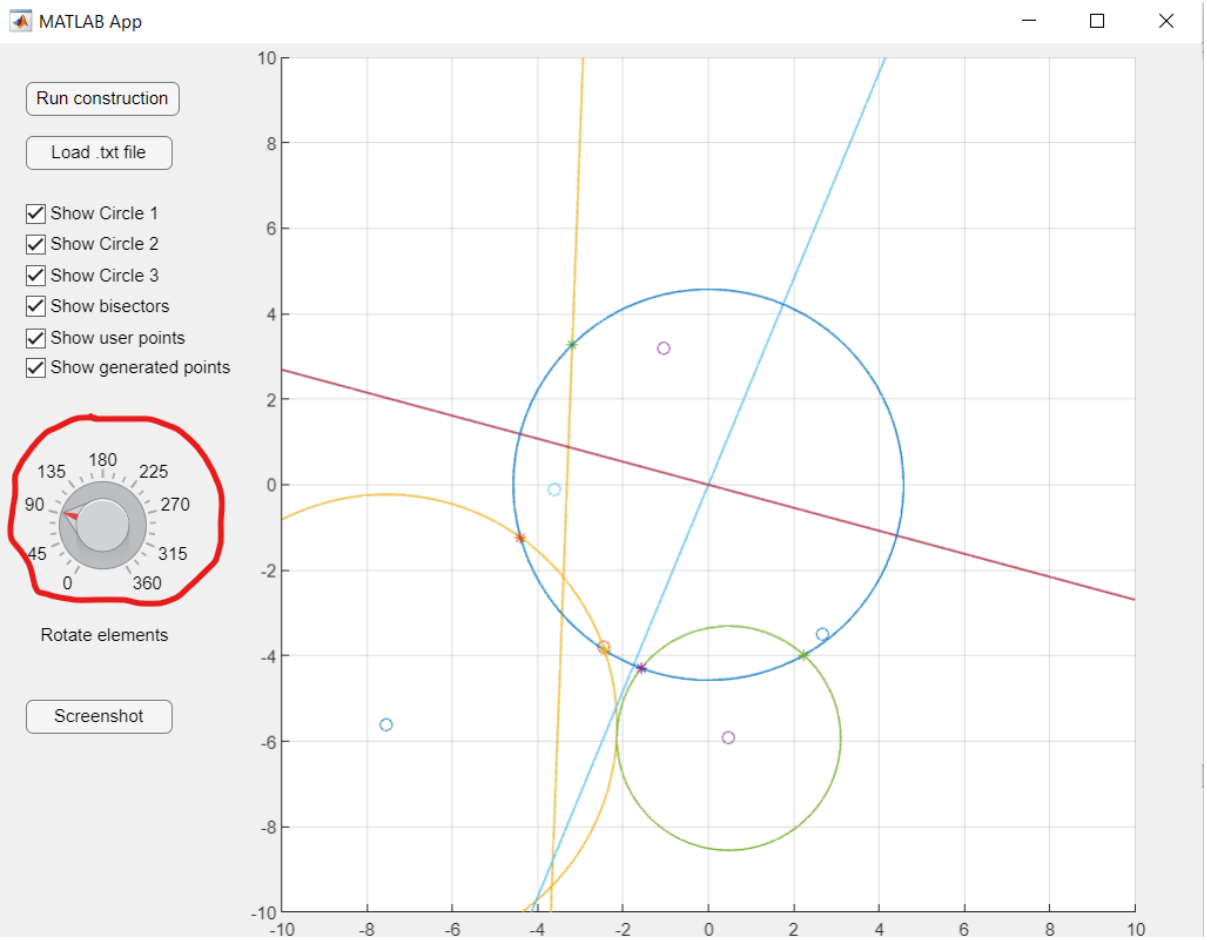

Figure 24: Elements rotated by 90 degrees using rotation dial

Burkardt, J. 2005. *circle_imp_line_par_int_2d source code.* [Source code]. Available at: https://people.sc.fsu.edu/~jburkardt/m_src/geometry/geometry.html [Accessed: 05 December 2022]

Burkardt, J. 2005. *circles_imp_int_2d source code.* [Source code]. Available at: https://people.sc.fsu.edu/~jburkardt/m_src/geometry/geometry.html [Accessed: 05 December 2022]

Burkardt, J. 2005. *line_exp2par_2d source code.* [Source code]. Available at: https://people.sc.fsu.edu/~jburkardt/m_src/geometry/geometry.html [Accessed: 05 December 2022]

Burkardt, J. 2005. *lines_par_int_2d source code.* [Source code]. Available at: https://people.sc.fsu.edu/~jburkardt/m_src/geometry/geometry.html [Accessed: 05 December 2022]