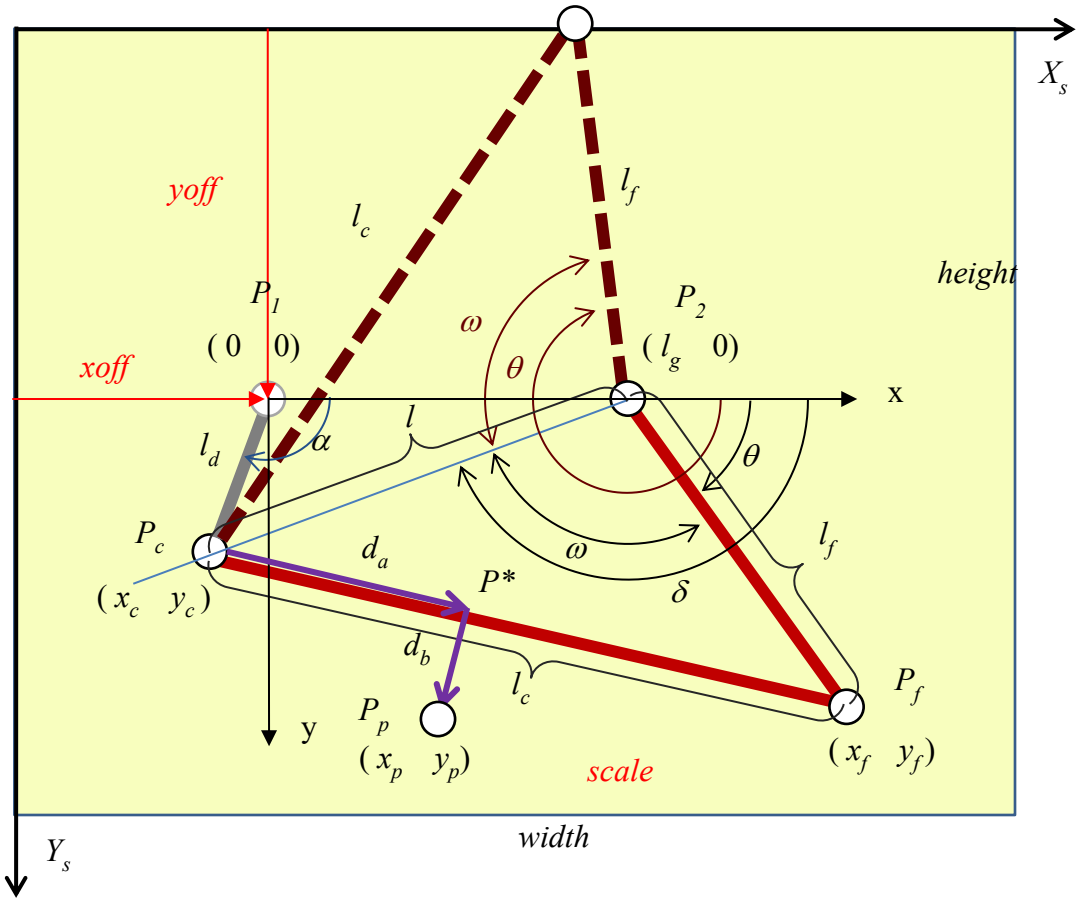
Homework Assignment 11 (2020)

Planar Four Bar Linkage

Create a Win application project named as **<yourID><yourName>Ass011**PlanarFourBarLinkage. Remember to rename your Form1.cs file to have a meaningful class name.

Add a class (namely, e.g., PlanarFourBarLinkage) to represent a four-bar linkage. The geometric notations are shown bellow. Your class should define data fields and public properties to let user specify the lengths of the four bars, configuration, and the location of point of interest; e.g., float Lg, Ld, Lc, Lf, alpha, Da, and Db, as shown in the figure. These data are characteristic data of a four-bar linkage.



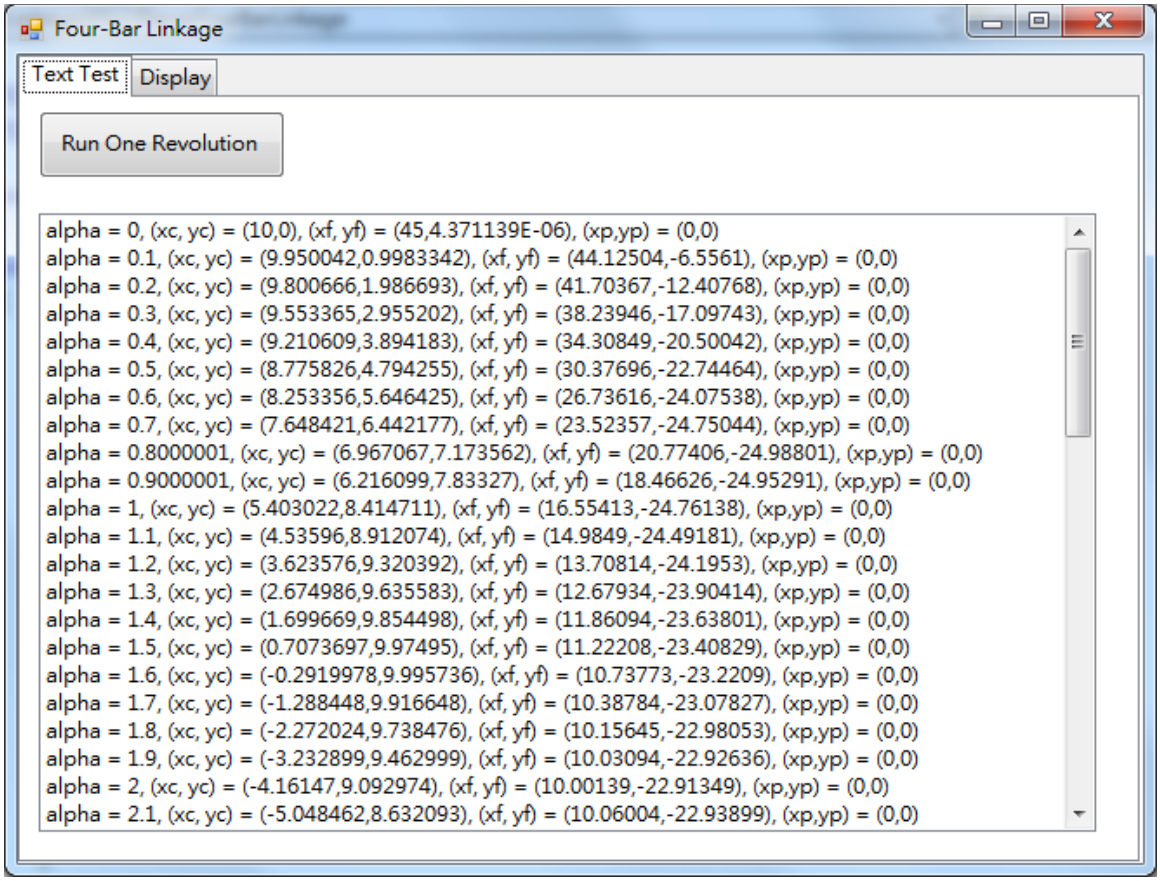
Part I

To display the linkage graphically, the coordinates of points P_1, P_2, P_c, P_f, and P_p should be calculated and updated by the class subject to the geometrical constraints. They are therefore defined as data fields of type `PointF` (defined in namespace `System.Drawing`), whose values should be updated by your class (where P_1 and P_2 remained fixed when α is changed).

Let coordinate system x-y is defined with origin located at P_1 and P_2 is located on the x axis. Define a public method (e.g., `public bool updateConfiguration( float newAlpha )`) that receive a new value of α and then updates the coordinates of points P_c,

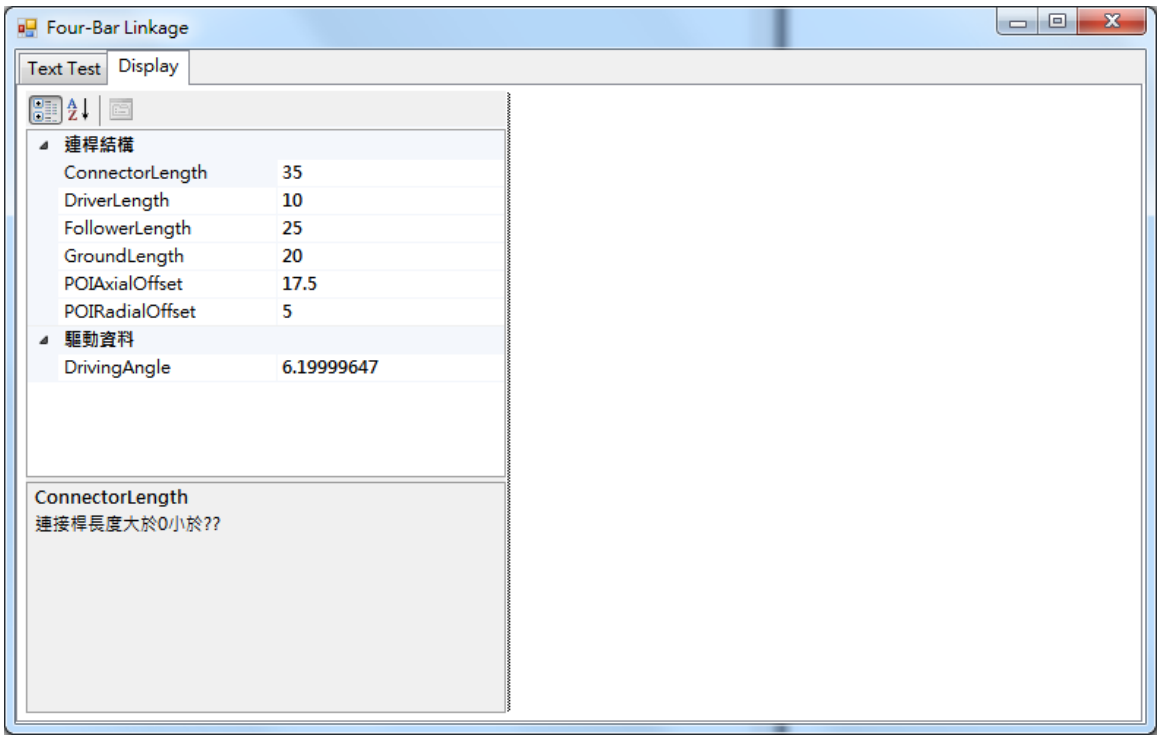
P_f, and P_p, if the value is OK; not violating the geometrical constraints. Notice that some values of **`newAlpha`** will not generate feasible configuration of the linkage. In this case, value of α should not be updated and the method returns **`false`** back to the caller.

In addition, provide a method that prints out the updated coordinates of points P_c, P_f, and P_p as a string and returns to the caller to display the configuration of the linkage. In your form, define an object of the linkage. Provide a button to automatically specify new values for α, from 0 to 2π, to repeatedly call the configuration update method (using **`for`**-statement). Provide a **`ListBox`** to display the updated coordinates of the linkage for each value of α.



Add two **`using`** statements: **`using System.ComponentModel;`** **`using System.Drawing;`** to your class file. Follow the instructions in the lab time to provide managed accessibility of private data fields with public properties. To take the advantage of property definitions, use a **`PropertyGrid`** UI object to display properties of the linkage for modification. Attributes (defined in namespace System.ComponentModel), such as [**`Category`**("xxx")], [**`DisplayName`**("yyy")], and [**`Description`**("zzz")] can be appended to properties to enhance readability and accessibility of the properties.

Public properties can be modified by the user interactively via suitable UI controls, such as an object of **`PropertyGrid`**. If a new value of a bar length property specified by the user is not be able to generate a feasible quadrilateral (四邊形), the bar length update is not allowed. Therefore, a helping function should be implemented to help guarding the property setting to maintain a feasible quadrilateral. Let the function be **`bool isQuadrilateralFeasible`**(). Implement this function correctly to check whether the given Lg, Ld, Lc, and Lf lengths can compose a feasible quadrilateral or not.




```
public class PlanarFourBarLinkage
{
    float Lg = 20.0f;
    float Ld = 10.0f;
    float Lc = 35.0f;

    [Category( "平面四連桿" )]
    [DisplayName( "Connector Length" )]
    [Description( "The length of the connector" )]
    public float ConnectorLength
    {
        get { return Lc; }
        set { Lc = value; }
    }

    float Lf = 25.0f;
    float Da = 17.5f;
    float Db = 5.0f;
    float alpha = 1.0f;

    PointF P1 = new PointF( 0.0f, 0.0f );
    PointF P2 = new PointF( );
    PointF Pc = new PointF( );
    PointF Pf = new PointF( );
    PointF Pp = new PointF( );

    public PlanarFourBarLinkage( )
    {
        updateConfiguration( 0.5f );
    }


    // Add this feasible quadrilateral (四邊形) check function on four
    // given lengths. And in length property setting, call this function
    // to check whether new length of a bar is ok or not. If not, do not
    // change the bar length.
```

```
    bool isQuadrilateralFeasible()
    {
           // For example:
           if (Lg > Ld + Lc + Lf) return false;
           // ...
           return true;
    }

    // attributes of GroundLength property
    [Category("平面四連桿"),Description("固定桿的長度")]
    public float GroundLength
    {
        get{  return Lg; }
        set
        {
            // Keep original value, in case new value is illegal
            float oldLg = Lg;
            Lg = value;
            if (!isQuadrilateralFeasible()) Lg = oldLg;
        }
    }

    public bool updateConfiguration( float newAlpha )
    {
        P2.X = Lg;
        P2.Y = 0.0f;
        Pc.X =  Ld * (float)Math.Cos( newAlpha );
        Pc.Y = Ld * (float)Math.Sin( newAlpha );
        float L;
        L = …
        if( … ) return false; // Infeasible configuration detected.
        // Configuration is OK. Update alpha.
        alpha = newAlpha;
        // Calculate Pf.X, Pf.Y
        double omega;
        double theta;
        //…
        // Use Math.Atan2(delta y,delta x)
        Pf.X = …;
        Pcf.Y= …
        float unitX, unitY;
        //…
        Pp.X = …;
        Pp.Y =…;
        return true;
    }

    public string getCoordinateString( )
    {
           return string.Format( "alpha = {0}, (xc, yc) = ({1},{2}), (xf,
yf) = ({3},{4}), (xp,yp) = ({5},{6})", alpha, … );
       // You can use interpolated string literal $"alpha={alpha}…"
    }

    int xoff, yoff;
    float scale;

    // Transform a point from linkage coordinates to
    // screen coordinates
```

```
    Point TransformPoint( PointF pt )
    {
        Point p = Point.Empty;
        p.X = (int) ( xoff + pt.X * scale );
        p.Y = (int) ( yoff + pt.Y * scale );
        return p;
    }

    // Draw the linkage within a rectangle of a graphics device
    public void DrawLinkage( Graphics g, Rectangle rect )
    {
       // Update xoffset, yoffset and scale with respect to the rectangle
       scale = rect.Width / ( Lc + Lf );
       xoff = (int)( Ld * scale * 1.1f );
       yoff = rect.Height / 2;
         // set screen points
       Point s1, s2, sc, sf, sp;
       s1 = TransformPoint(p1); s2 =…; sc= …; sf = …; sp = …;
         // Draw 5 lines
       g.DrawLine( Pens.Red, s1, sc );
        g.DrawLine( Pens.Red, s1, sc );
         …

    }

    public Point PointOfInterest
    {
        get { return TransformPoint(Pp); }
    }

}
```

The coordinates of the points resulting for a given driving angle α are derived as follows:

$$x_c = l_d \cos\alpha \qquad y_c = l_d \sin\alpha$$

$$\left(l_g - x_c\right)^2 + y_c{}^2 = l^2$$

$$\text{if } (l_f + l_c < l \vee l + l_c < l_f \vee l + l_f < l_c) \text{ no feasible configuration}$$

$$2 l_f l \cos\omega = l_f^2 + l^2 - l_c^2$$

$$\omega = \cos^{-1}\left(\frac{l_f^2 + l^2 - l_c^2}{2 l_f l}\right)$$

$$\theta = \delta \pm \omega = \tan^{-1}\left(\frac{y_c}{x_c - l_g}\right) \pm \omega$$

$$x_f = l_f \cos\theta + l_g \qquad y_f = l_f \sin\theta$$

$$x_p = ??d_a\,??d_b \qquad y_p = ??d_a\,??d_b$$

The derivation of the coordinates of P_p is your job. You should find unit vector from point P_c to P_f first. Then you can find coordinates of point P* using D_a. Note that vector from

point P* to P_p is perpendicular to the vector from P_c to P_f. Reuse the unit vector and D_b to find coordinates of P_p.

Part II

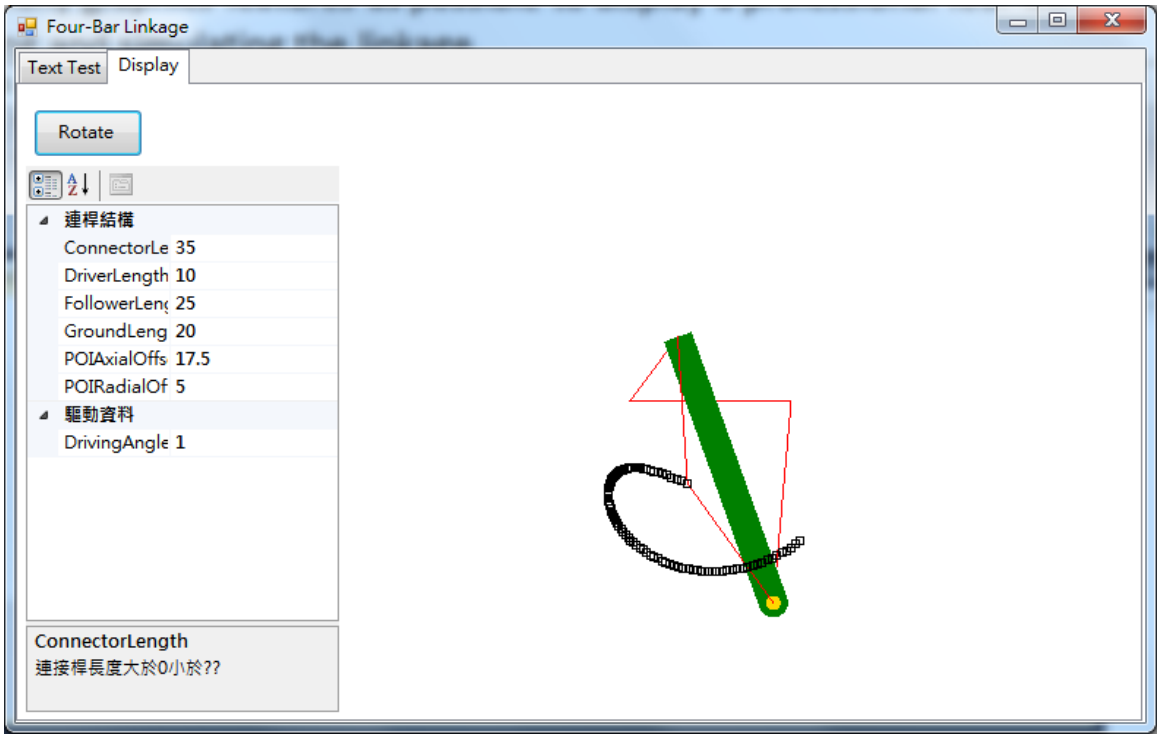
Enhance your assignment by adding graphics display and animation to your four-bar linkage. Explore as many graphics features as possible to display a professional four-bar linkage for learning and simulating the linkage. To show points of interest, in your form class add a dynamic list of points: **`List<Point> pois = new List<Point>();`**. Add a property to the FourBarLinkage class to return the current point of interest.

```
public Point PointOfInterest
{
    get { return TransformPoint(Pp); }
}
```

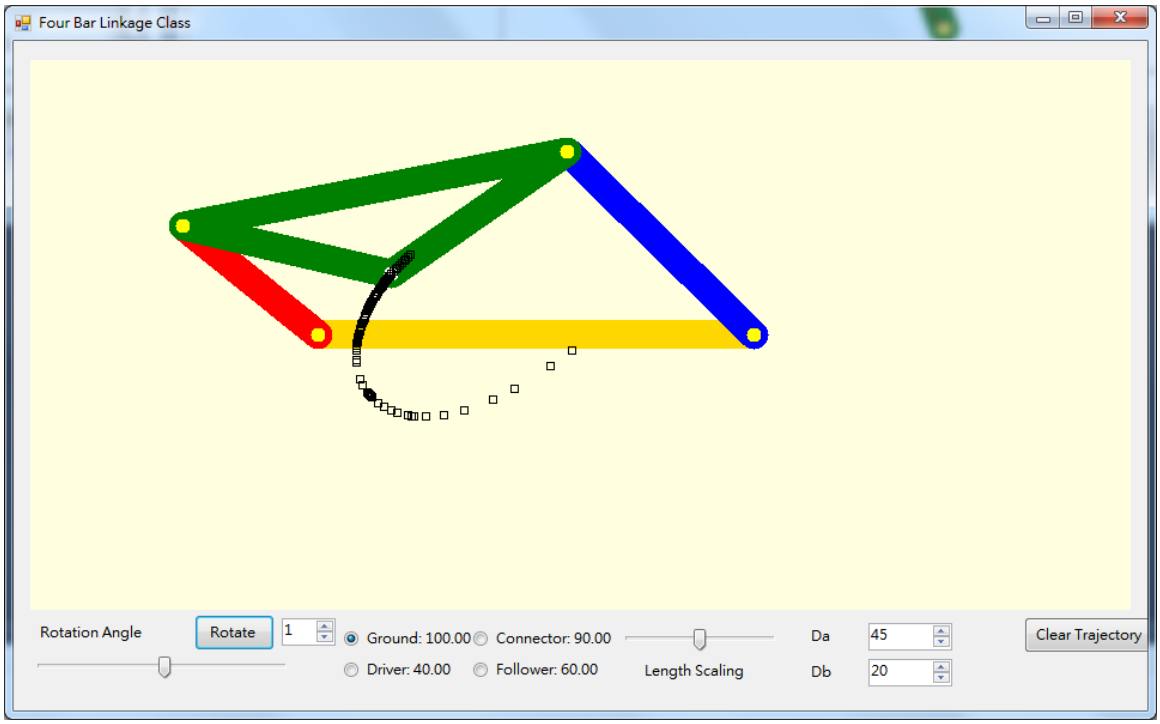
In timer tick event handling function we add the updated POI to the list. And in the Paint event handling method of the panel that display the linkage, draw the list of points of interest one after the other one. Provide Button to allow user to clear the points of interest. Useful methods of a **`List`**<> instance include: **`Add()`**, **`Clear()`**.

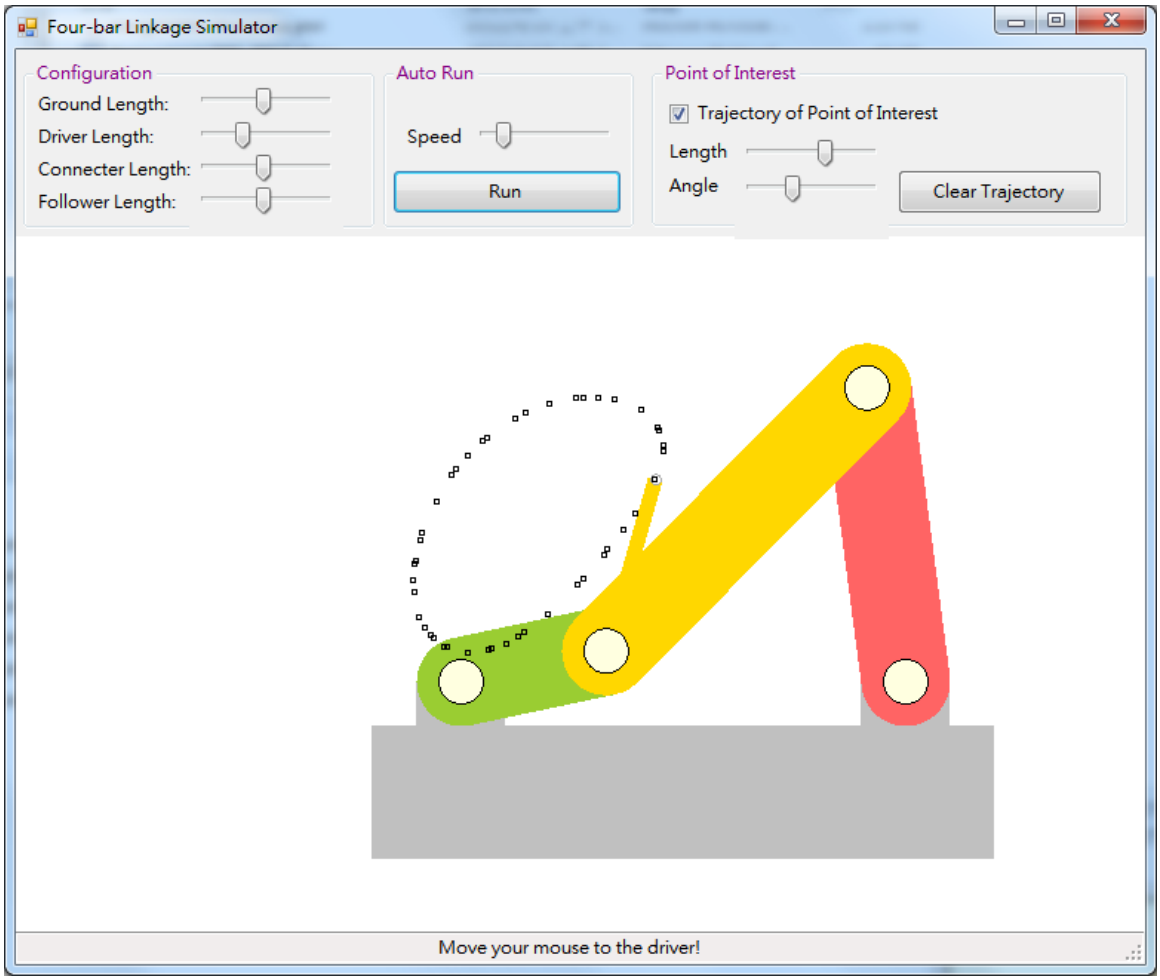
Try to add other UI controls to allow user manipulating the configure of the linkage. For example, adding mouse events (Down, Move, Up, etc.) to simulate the linkage driving or adding **`TrackBar`** controls to dynamically modify the bar lengths.

The following is a draft version of the application for your reference.



The follows are formal or professional versions for your reference.


Four Bar Linkage Class
Rotation Angle
Rotate
1
Ground: 100.00
Connector: 90.00
Driver: 40.00
Follower: 60.00
Length Scaling
Da
45
Db
20
Clear Trajectory


Four-bar Linkage Simulator
Configuration
Ground Length:
Driver Length:
Connecter Length:
Follower Length:
Auto Run
Speed
Run
Point of Interest
Trajectory of Point of Interest
Length
Angle
Clear Trajectory
Move your mouse to the driver!