CPEG 585 - Assignment 4

Develop a matrix equation for minimizing the cost in image registration (without shear) when the correspondences between the points in two images I1 and I2 are known. Summation indicates the sum over all points in a shape.

$$\text{Cost} = \sum (I_1 - T(I_2))^2 = \sum \left(\begin{pmatrix} x1 \\ Y1 \end{pmatrix} - \left(\begin{pmatrix} a & b \\ -b & a \end{pmatrix} \begin{pmatrix} x2 \\ y2 \end{pmatrix} + \begin{pmatrix} t1 \\ t2 \end{pmatrix} \right) \right)^2 = \sum \begin{pmatrix} x_1 - (ax_2 + by_2 + t_1) \\ y_1 - (-bx_2 + ay_2 + t_2) \end{pmatrix}^2$$

$$\text{Cost} = \sum (x_1 - (ax_2 + by_2 + t_1))^2 + (y_1 - (-bx_2 + ay_2 + t_2))^2$$

a) To find the optimal transformation that will align image 2 to image 1, take the partial derivatives of the above cost with respect to a, b, t1 and t2 and set these to 0. Express the four resulting equations in matrix form.

$\partial C/\partial a = 0$

$\partial C/\partial b = 0$

$\partial C/\partial t_1 = 0$

$\partial C/\partial t_2 = 0$

For example, the first equation will appear as:

$$\partial C/\partial a = -2x_2 (x_1 - ax_2 - by_2 - t_1) - 2y_2(y_1 + bx_2 - ay_2 - t_2) = 0$$
$$-2x_1x_2 + 2ax_2^2 + 2bx_2y_2 + 2x_2t_1 - 2y_1y_2 - 2bx_2y_2 + 2ay_2^2 + 2y_2t_2 = 0$$

$$(2x_2^2 + 2y_2^2)a + 0\,b + 2x_2t_1 + 2y_2t_2 = 2x_1x_2 + 2y_1y_2$$

In matrix form, it can be written as

$$\sum \begin{pmatrix} 2x_2^2 + 2y_2^2 & 0 & 2x_2 & 2y_2 \\ & & & \\ & & & \\ & & & \end{pmatrix} \begin{pmatrix} a \\ b \\ t_1 \\ t_2 \end{pmatrix} = \sum \begin{pmatrix} 2x_1x_2 + 2y_1y_2 \\ \\ \\ \\ \end{pmatrix}$$

Complete the above matrix equation for all partial derives.

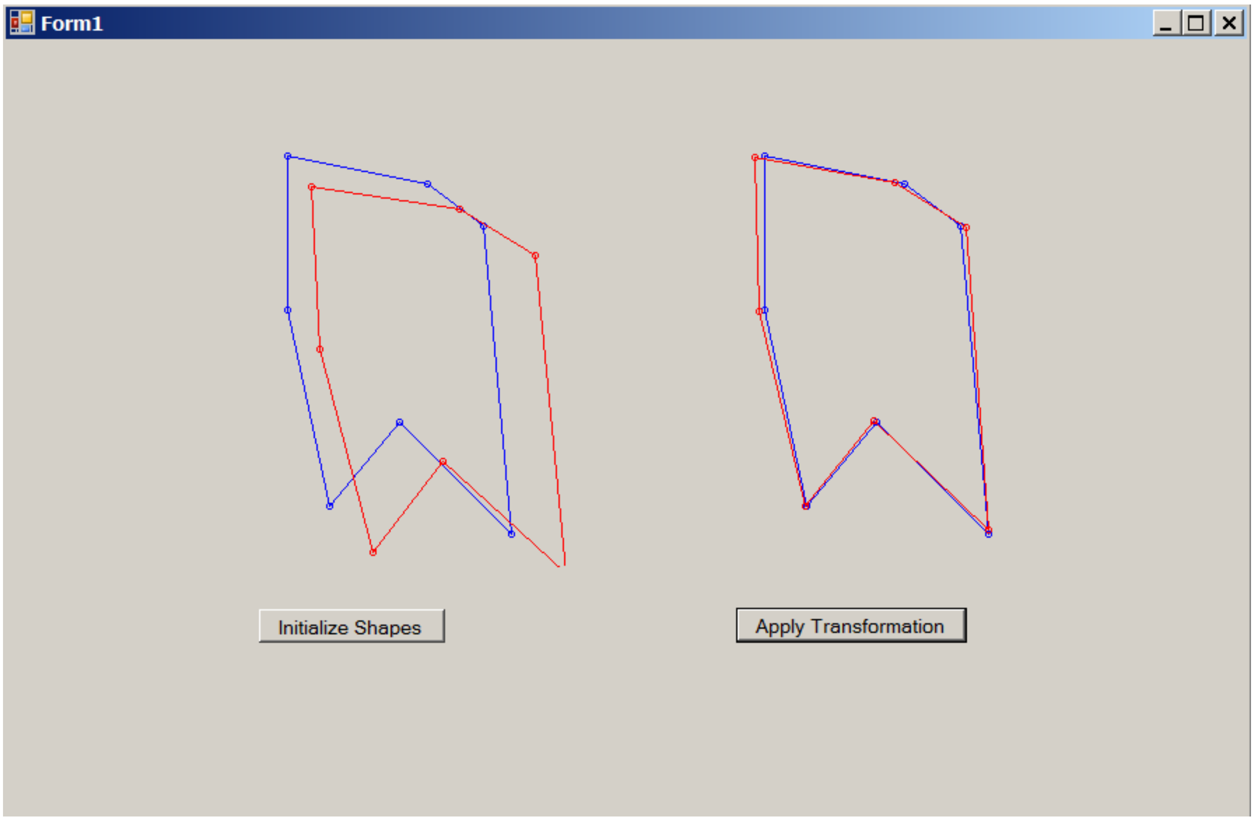
b) Test the above registration of two images by creating a set of points corresponding to two shapes, and then by applying the above matrix equation you will be able to register the I2 image with I1. For example, in C#, your test case code may appear as:

```
List<Point> Shape1 = new List<Point>();
List<Point> Shape2 = new List<Point>();

  private void btnInitializeShapes_Click(object sender, EventArgs e)
        {
            Shape1.Clear();
            Shape2.Clear();
            Point p1a = new Point(20, 30);
            Point p2a = new Point(120, 50);
            Point p3a = new Point(160, 80);
            Point p4a = new Point(180, 300);
            Point p5a = new Point(100, 220);
            Point p6a = new Point(50, 280);
            Point p7a = new Point(20, 140);

            Shape1.Add(p1a);   Shape1.Add(p2a);
            Shape1.Add(p3a);   Shape1.Add(p4a);
            Shape1.Add(p5a);   Shape1.Add(p6a);
            Shape1.Add(p7a);

            Transformation T2 = new Transformation();
            T2.A = 1.05; T2.B = 0.05; T2.T1 = 15; T2.T2 = 22;
            Shape2 = ApplyTransformation(T2, Shape1);
            Shape2[2] = new Point(Shape2[2].X + 10, Shape2[2].Y + 3);// change one
point
            Pen pBlue = new Pen(Brushes.Blue, 1);
            Pen pRed = new Pen(Brushes.Red, 1);
            Graphics g = panShape1.CreateGraphics();
            DisplayShape(Shape1, pBlue, g);
            DisplayShape(Shape2, pRed, g);
            }
```

Transformation is a simple class with a, b, t1, and t2 fields in it. If you place two panels on the form with IDs of panShape1 and panShape2, then the code for DisplayShape appears as:

```
void DisplayShape(List<Point> Shp, Pen pen, Graphics g)
 {
     Point? prevPoint = null;  // nullable
     foreach (Point pt in Shp)
     {
         g.DrawEllipse(pen, new Rectangle(pt.X - 2, pt.Y - 2, 4, 4));
         if (prevPoint != null)
             g.DrawLine(pen, (Point)prevPoint, pt);
         prevPoint = pt;
     }
     g.DrawLine(pen, Shp[0], Shp[Shp.Count - 1]);
 }
```

After you complete the above test program for determining and applying the transformation needed for Shape2 so that it aligns with shapes 1, it will appear as:



The left set of pictures corresponds to two shapes before registration. Right set of pictures shows Shape1 and transformed shape 2 after applying the transformation determined from partial derivative equations on Shape 2.

Note that you can use Mapack matrix library to solve the matrix equation in part 1). The Mapack library provides a Matrix class with an Inverse matrix computation and overload of multiply operator. After you have created the sum over all points for the 4x4 matrix and the right hand 4x1 vector in part 1), you can compute the required transformation by:

```
Matrix Ainv = A.Inverse;
Matrix Res = Ainv * B;
```