**CPEG 585 – COMPUTER VISION**

**HOMEWORK 5**

**Name:** Ivan Sangines Escrig

**ID#:** 968606

**Instructor:** Dr. Mahmood

**Date:** March 5, 2019

**TABLE OF CONTENT**

[**INTRODUCTION**](#_Toc446970371) 3

[**SCREENSHOTS:**](#_Toc446970373) 4

[**SOURCE CODE:**](#_Toc446970374) 6

**[CONCLUSION:](#_Toc446970375)**11

**INTRODUCTION**

The purpose of this assignment is to get familiar and understand better how to reduce the cost for an image registration when the correspondences between two images are known.

The cost is defined by the following formula:

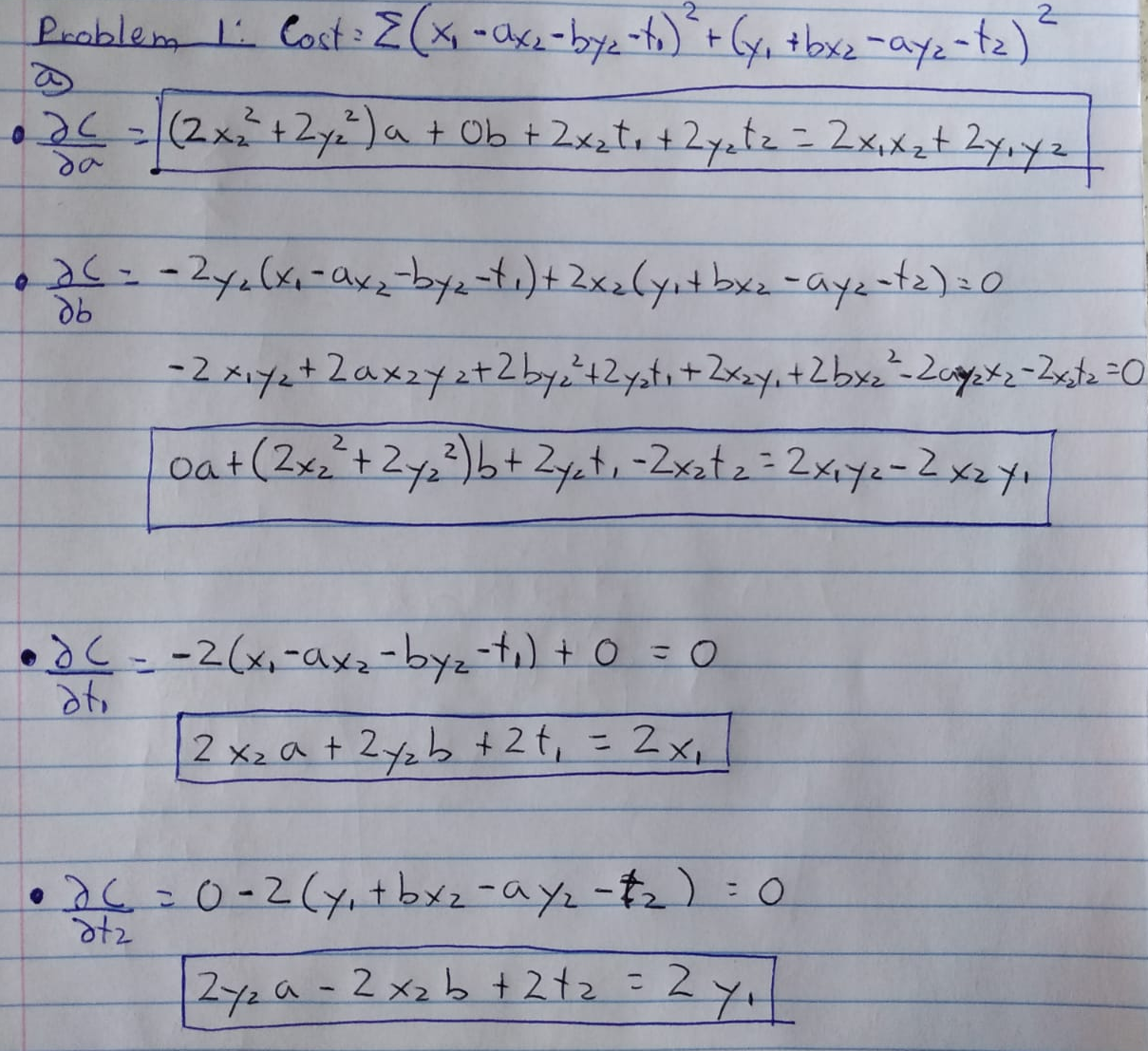
The first part of the assignment consists in finding the Matrix equations, which will be used to find the unknown parameters a, b, t1, t2 in order to be able to compare the two images with the least possible error. In order to find the Matrix, we will apply the technique Linear Least Squares Optimization to the partial derivatives of a, b, t1, t2.

On the other hand, the second part of this assignment consists in creating a shape and shifting it by some a, b, t, t2 values. Then, we will apply the Matrix in problem 1 to reshape the second shape and compare two shapes (they should be very similar since we are using linear of squares, the difference between shapes should be reduced to a value close to 0). In order to do all the Matrix computations, we will be using the Matrix class from the Mapack library.

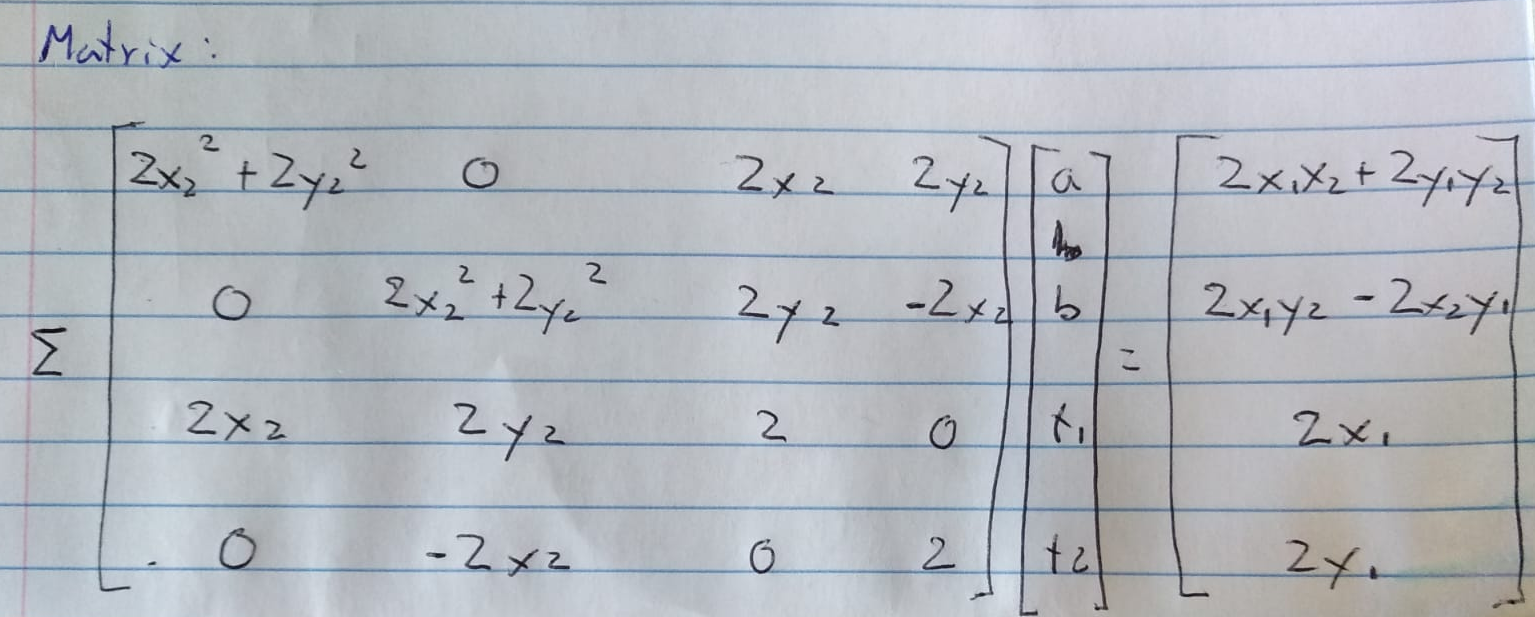
**SCREEN SHOTS:**

**Problem 1:**

Derivatives:

****

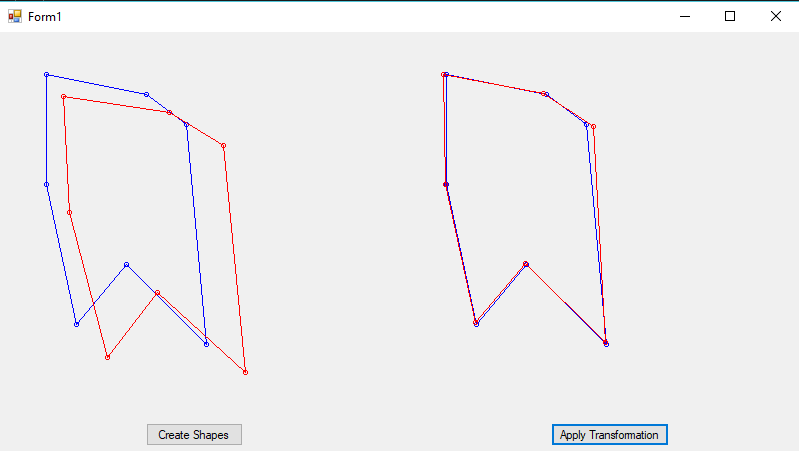
Final System of equations:



**Problem 2:**

On the following screenshot, as we can see on the drawings on the left, the initial shape is drawn in blue. A new shape in red is generated by shifting the initial image with chosen a, b, t, t2.

The right side of the image shows the results after applying the correct transformation to the shape 2. In order to find the correct transformation, I applied the linear of squares technique to the cost function. I used the matrix obtained, after applying the linear of squares technique, to find the specific a, b, t, t2 that will make the second shape as similar as possible to the first one



**SOURCE CODE:**

**Problem 2:**

Transformation Class (used to store unknown parameters):

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ImageRegistration

{

class Transformation

{

public double A { get; set; }

public double B { get; set; }

public double T1 { get; set; }

public double T2 { get; set; }

}

}

Form:

using Mapack;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ImageRegistration

{

public partial class Form1 : Form

{

//Lists with the points for each shape

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

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

Transformation T = new Transformation();

Matrix cost\_matrix = new Matrix(4, 4);

Matrix result = new Matrix(4, 1);

public Form1()

{

InitializeComponent();

}

//Creating Shapes

private void create\_btn\_Click\_1(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);

T.A = 1.05;

T.B = 0.05;

T.T1 = 15;

T.T2 = 22;

Shape2 = ApplyTransformation(T, 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 = panel1.CreateGraphics();

DisplayShape(Shape1, pBlue, g);

DisplayShape(Shape2, pRed, g);

}

List<Point> ApplyTransformation(Transformation t, List<Point> Shape)

{

List<Point> transformed = new List<Point>();

foreach (Point pt in Shape)

{

Point temp = transformPoint(t, pt);

transformed.Add(temp);

}

return transformed;

}

//Applying transformation to the points

Point transformPoint (Transformation tr, Point p)

{

Matrix param = new Matrix(2, 2);

param[0, 0] = tr.A;

param[0, 1] = tr.B;

param[1, 0] = tr.B \* -1;

param[1, 1] = tr.A;

Matrix tcomponents = new Matrix(2, 1);

tcomponents[0, 0] = tr.T1;

tcomponents[1, 0] = tr.T2;

Matrix points = new Matrix(2, 1);

points[0, 0] = p.X;

points[1, 0] = p.Y;

Matrix resM = new Matrix(2, 1);

resM = param\*points+tcomponents;

Point transformedPoint = new Point((int)resM[0, 0], (int)resM[1, 0]);

return transformedPoint;

}

//Drawing the shapes

void DisplayShape(List<Point> Shape, Pen pen, Graphics g)

{

Point? prevPoint = null; // nullable

foreach (Point pt in Shape)

{

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, Shape[0], Shape[Shape.Count - 1]);

}

private void button2\_Click(object sender, EventArgs e)

{

//Computing all the sumations in order to generate the matrix

for (int i=0; i<Shape1.Count; i++)

{

cost\_matrix[0, 0] += 2 \* Shape2[i].X \* Shape2[i].X + 2 \* Shape2[i].Y \* Shape2[i].Y;

//cost\_matrix[0, 1] += 0;

cost\_matrix[0, 2] += 2 \* Shape2[i].X;

cost\_matrix[0, 3] += 2 \* Shape2[i].Y;

//cost\_matrix[1, 0] += 0;

cost\_matrix[1, 1] += 2 \* Shape2[i].X \* Shape2[i].X + 2 \* Shape2[i].Y \* Shape2[i].Y;

cost\_matrix[1, 2] += 2 \* Shape2[i].Y;

cost\_matrix[1, 3] -= 2 \* Shape2[i].X ;

cost\_matrix[2, 0] += 2 \* Shape2[i].X;

cost\_matrix[2, 1] += 2 \* Shape2[i].Y;

cost\_matrix[2, 2] += 2;

//cost\_matrix[2, 3] += 0;

cost\_matrix[3, 0] += 2 \* Shape2[i].Y;

cost\_matrix[3, 1] -= 2 \* Shape2[i].X ;

//cost\_matrix[3, 2] += 0;

cost\_matrix[3, 3] += 2;

result[0, 0] += 2 \* Shape1[i].X \* Shape2[i].X + 2 \* Shape1[i].Y \* Shape2[i].Y;

result[1, 0] += 2 \* Shape1[i].X \* Shape2[i].Y - 2 \* Shape2[i].X \* Shape1[i].Y;

result[2, 0] += 2 \* Shape1[i].X;

result[3, 0] += 2 \* Shape1[i].Y;

}

Matrix Costinv = cost\_matrix.Inverse;

Matrix final\_res = Costinv\*result;

//Creating new transformation instance with the new parameters

Transformation myT = new Transformation();

myT.A = final\_res[0, 0];

myT.B = final\_res[1, 0];

myT.T1 = final\_res[2, 0];

myT.T2 = final\_res[3, 0];

//Reshaping Shape2 to compare with shape1

List<Point> transformed\_Shape2 = ApplyTransformation(myT,Shape2);

Pen pBlue = new Pen(Brushes.Blue, 1);

Pen pRed = new Pen(Brushes.Red, 1);

Graphics g = panel2.CreateGraphics();

DisplayShape(Shape1, pBlue, g);

DisplayShape(transformed\_Shape2, pRed, g);

}

}

}

**Conclusion:**

After concluding this assignment, I have to say I understand better how to develop the Matrix equation for minimizing the cost in image registrations by applying the linear of squares technique to the Cost function.