**CPEG 585 – COMPUTER VISION**

**HOMEWORK 6**

**Name:** Ivan Sangines Escrig

**ID#:** 968606

**Instructor:** Dr. Mahmood

**Date:** March 12, 2019

**TABLE OF CONTENT**

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

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

[**SOURCE CODE:**](#_Toc446970374) 8

**[CONCLUSION:](#_Toc446970375)**15

**INTRODUCTION**

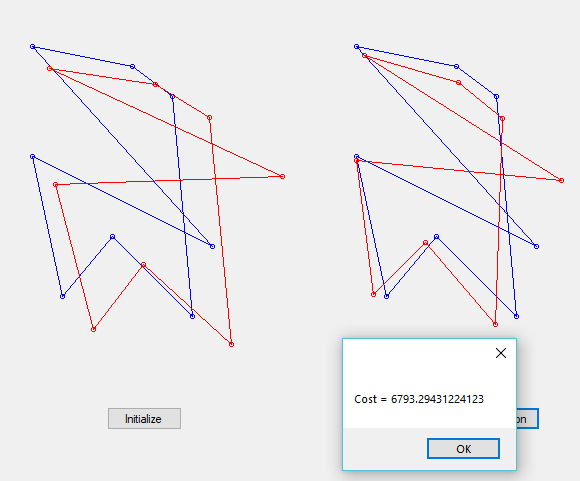
The purpose of this assignment is to get familiar and understand better how to reduce de noise or outliners from an image/shape. The first part we will be reducing the outliners using exhaustive evaluation, which consists in removing the points with more error on the shapes, we can decide what percentage of data to remove.

The second part of the assignment consists in understand better how to use RANSAC algorithm in order to remove noise from two different shapes. We will be using RANSAC in order to remove an outliner given 2 different shapes. However, it can be a very useful algorithm to use in image recognition knowing the correspondences.

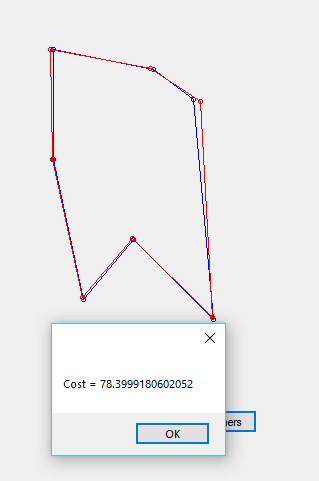
**SCREEN SHOTS:**

**Problem 1:**

In the first screenshot we can see how the shapes are transformed in order to readjust them as close as possible to each other. However, the outliners are making the error to be pretty high.

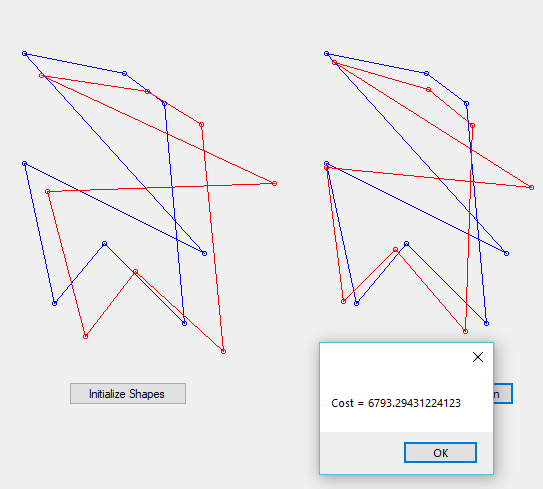
****

On the following screenshot we can see how we ended up eliminating the outliner making the shapes really similar to each other. We can see the exhaustive removal is very useful since we are reducing the error to 73 from the initial 6793.

****

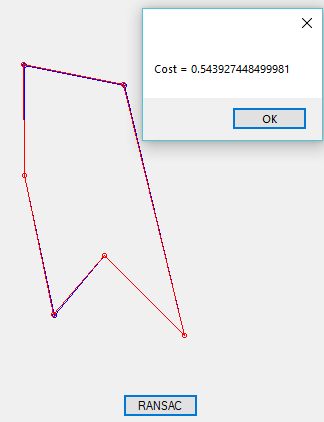
**Problem 2:**

As we can see on the following screenshot, the first step of the assignment is to create two different shapes with some outliners on it (panel 1). Then, we will apply the matrix equation for minimizing the cost, used on assignment 4. As we can see, this technique will not do a good job since the outliners are making us obtain a very big error (6793). In order to reduce the cost/error we will need to remove the outliners by applying the RANSAC algorithm.



The third pannel shows the result obtained after applying the RANSAC algorithm. As we can see, there are not 8 points anymore, that is because we removed the outliners (noise) that were distorsioning our results making it impossible to mach both shapes. Now we have a better model as we can see on the cost results, reducing it to 0.5439 much better than the previous 6793.

In order to apply the RANSAC algorithm, I am iterating 20 times to find the best result (since we just have 8 data points we do not need many iterations), and accepting any point that increments the error by less that 7 (treshold). Also, I define a valid model once we have more than 4 data points in the model.



**SOURCE CODE:**

**Problem 1:**

**Form:**

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 ExhaustiveRMVL

{

public partial class Form1 : Form

{

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

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

List<Point> Shape2Transformed = new List<Point>();

List<Point> Shape1Transformed = new List<Point>();

Transformation T;

public Form1()

{

InitializeComponent();

}

private void button1\_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

// add outliers to both shapes

Point ptOutlier1 = new Point(200, 230);

Shape1.Add(ptOutlier1);

Point ptOutLier2 = new Point(270, 160);

Shape2.Add(ptOutLier2);

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);

}

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

{

T = ICPTransformation.ComputeTransformation(Shape1, Shape2);

List<Point> Shape2T = ApplyTransformation(T, Shape2);

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

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

Graphics g = panel2.CreateGraphics();

DisplayShape(Shape1, pBlue, g);

DisplayShape(Shape2T, pRed, g);

MessageBox.Show("Cost = " + ICPTransformation.ComputeCost(Shape1, Shape2, T).ToString());

}

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]);

}

List<Point> ApplyTransformation(Transformation T, List<Point> shpList)

{

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

foreach (Point pt in shpList)

{

double xprime = T.A \* pt.X + T.B \* pt.Y + T.T1;

double yprime = T.B \* pt.X \* -1 + T.A \* pt.Y + T.T2;

Point pTrans = new Point((int)xprime, (int)yprime);

TList.Add(pTrans);

}

return TList;

}

private void button3\_Click(object sender, EventArgs e)

{

double[] cost = new double[Shape1.Count];

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

{

double xprime = T.A \* Shape2[i].X + T.B \* Shape2[i].Y + T.T1;

double yprime = -1 \* T.B \* Shape2[i].X + T.A \* Shape2[i].Y + T.T2;

cost[i] = (Shape1[i].X - xprime) \* (Shape1[i].X - xprime) + (Shape1[i].Y - yprime) \* (Shape1[i].Y - yprime);

}

int count = (int)(Shape1.Count \* 0.8) + 1;

int[] index = new int[count];

for (int j = 0; j < count; j++)//remove 20% of the points with the largest errors

{

double min = cost.Min();

index[j] = cost.ToList().IndexOf(min);

cost[index[j]] = cost.Max();

}

//Sort the indexes in order to NOT show a random shape

Array.Sort(index);

for (int i = 0; i < index.Length; i++)

{

Shape1Transformed.Add(Shape1[index[i]]);

Shape2Transformed.Add(Shape2[index[i]]);

}

Transformation T2 = ICPTransformation.ComputeTransformation(Shape1Transformed, Shape2Transformed);

List<Point> Shape2T = this.ApplyTransformation(T2, Shape2Transformed);

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

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

Graphics g = panel3.CreateGraphics();

DisplayShape(Shape1Transformed, pBlue, g);

DisplayShape(Shape2T, pRed, g);

MessageBox.Show("Cost = " + ICPTransformation.ComputeCost(Shape1Transformed, Shape2Transformed, T2).ToString());

}

}

}

**Problem 2:**

**Form:**

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 RANSAC

{

public partial class Form1 : Form

{

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

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

List<Point> Shape2Transformed = new List<Point>();

Ransac\_Algo ransac = new Ransac\_Algo();

public Form1()

{

InitializeComponent();

}

private void initialize\_btn\_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

// add outliers to both shapes

Point ptOutlier1 = new Point(200, 230);

Shape1.Add(ptOutlier1);

Point ptOutLier2 = new Point(270, 160);

Shape2.Add(ptOutLier2);

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

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

Graphics g = original\_panel.CreateGraphics();

DisplayShape(Shape1, pBlue, g);

DisplayShape(Shape2, pRed, g);

}

private void transformation\_btn\_Click(object sender, EventArgs e)

{

Transformation T = ICPTransformation.ComputeTransformation(Shape1, Shape2);

List<Point> Shape2T = ApplyTransformation(T, Shape2);

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

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

Graphics g = transformed\_panel.CreateGraphics();

DisplayShape(Shape1, pBlue, g);

DisplayShape(Shape2T, pRed, g);

MessageBox.Show("Cost = " + ICPTransformation.ComputeCost(Shape1, Shape2, T).ToString());

}

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]);

}

List<Point> ApplyTransformation(Transformation T, List<Point> shpList)

{

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

foreach (Point pt in shpList)

{

double xprime = T.A \* pt.X + T.B \* pt.Y + T.T1;

double yprime = T.B \* pt.X \* (-1) + T.A \* pt.Y + T.T2;

Point pTrans = new Point((int)xprime, (int)yprime);

TList.Add(pTrans);

}

return TList;

}

private void ransac\_btn\_Click(object sender, EventArgs e)

{

ransac = ApplyRansac();

//we need to sort the points, otherwise we will get a random figure

List<Point> sorted1 = new List<Point>();

List<Point> sorted2 = new List<Point>();

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

{

if (ransac.best\_consensus1.Contains(Shape1[j]))

{

sorted1.Add(Shape1[j]);

sorted2.Add(Shape2[j]);

}

}

List<Point> new\_Shape2 = ApplyTransformation(ransac.best\_model, sorted2);

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

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

Graphics g = ransac\_panel.CreateGraphics();

DisplayShape(sorted1, pBlue, g);

DisplayShape(new\_Shape2, pRed, g);

MessageBox.Show("Cost = " + ransac.best\_error);

}

Ransac\_Algo ApplyRansac()

{

Ransac\_Algo rans = new Ransac\_Algo();

int k = 20; //iterations for the algorithm

int t = 7; //treshold to determine if we accept the new point or not, if it adds less than 7 to the cost it is valid

int n = 3; //number of initial random point

int d = 4; //minimum data points to accept the model as valid

int iterations = 0; //counter of iterations done

/\*

Transformation best\_model = new Transformation();

List<Point> best\_consensus\_set1 = new List<Point>();

List<Point> best\_consensus\_set2 = new List<Point>();

Ransac\_Algo rans = new Ransac\_Algo();

\*/

double best\_error = int.MaxValue;

Random rand = new Random(); //used to select random indexes

while (iterations < k)

{

List<int> random\_index = new List<int>(); //list that will contain random indexes from shapes

List<Point> maybe\_inliners1 = new List<Point>();

List<Point> maybe\_inliners2 = new List<Point>();

Transformation maybe\_model = new Transformation();

double maybe\_cost = 0; //cost from the random points

//selecting random points (using indexes) from data

for (int i = 0; i < n; i++)

{

int index = rand.Next(Shape1.Count);

while (random\_index.Contains(index))

{

index = rand.Next(Shape1.Count);

}

random\_index.Add(index);

maybe\_inliners1.Add(Shape1[index]);

maybe\_inliners2.Add(Shape2[index]);

}

maybe\_model = ICPTransformation.ComputeTransformation(maybe\_inliners1, maybe\_inliners2); //computing model for random chosen points

maybe\_cost = ICPTransformation.ComputeCost(maybe\_inliners1, maybe\_inliners2, maybe\_model); //computing cost for random chosen points

List<Point> consensus\_set1 = new List<Point>(maybe\_inliners1);

List<Point> consensus\_set2 = new List<Point>(maybe\_inliners2);

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

{

if (!random\_index.Contains(i))

{

consensus\_set1.Add(Shape1[i]);

consensus\_set2.Add(Shape2[i]);

Transformation temp\_model = ICPTransformation.ComputeTransformation(consensus\_set1, consensus\_set2);

double temp\_cost = ICPTransformation.ComputeCost(consensus\_set1, consensus\_set2, temp\_model);

double dif = Math.Abs(maybe\_cost - temp\_cost);

if (dif >= t)

{

consensus\_set1.RemoveAt(consensus\_set1.Count - 1);

consensus\_set2.RemoveAt(consensus\_set2.Count - 1);

}

}

}

if (consensus\_set1.Count > d) //checking if consensus has enough points to make the model valid

{

Transformation better\_model = ICPTransformation.ComputeTransformation(consensus\_set1, consensus\_set2);

double this\_error = ICPTransformation.ComputeCost(consensus\_set1, consensus\_set2, better\_model);

if (this\_error < best\_error)

{

rans.best\_model = better\_model;

rans.best\_consensus1 = consensus\_set1;

rans.best\_consensus2 = consensus\_set2;

rans.best\_error = this\_error;

}

}

iterations++;

}

return rans;

}

}

}

**Ransac Class:**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RANSAC

{

class Ransac\_Algo

{

public Transformation best\_model { get; set; }

public List<Point> best\_consensus1 { get; set; }

public List<Point> best\_consensus2 { get; set; }

public double best\_error { get; set; }

}

}

**Transformation Class:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RANSAC

{

class Transformation

{

public double A { get; set; }

public double B { get; set; }

public double T1 { get; set; }

public double T2 { get; set; }

}

}

**ICPTransformation Class:**

using Mapack;

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RANSAC

{

class ICPTransformation

{

public static Transformation ComputeTransformation(List<Point> Shp1, List<Point> Shp2)

{

Matrix A = new Matrix(4, 4);

Matrix B = new Matrix(4, 1);

for (int i = 0; i < Shp1.Count; i++)

{

A[0, 0] += 2 \* Shp2[i].X \* Shp2[i].X + 2 \* Shp2[i].Y \* Shp2[i].Y;

A[0, 1] += 0;

A[0, 2] += 2 \* Shp2[i].X;

A[0, 3] += 2 \* Shp2[i].Y;

A[1, 0] += 0;

A[1, 1] += 2 \* Shp2[i].X \* Shp2[i].X + 2 \* Shp2[i].Y \* Shp2[i].Y;

A[1, 2] += 2 \* Shp2[i].Y;

A[1, 3] -= 2 \* Shp2[i].X;

A[2, 0] += 2 \* Shp2[i].X;

A[2, 1] += 2 \* Shp2[i].Y;

A[2, 2] += 2;

A[2, 3] += 0;

A[3, 0] += 2 \* Shp2[i].Y;

A[3, 1] -= 2 \* Shp2[i].X;

A[3, 2] += 0;

A[3, 3] += 2;

//A[3, 3] += -2;

B[0, 0] += 2 \* Shp1[i].X \* Shp2[i].X + 2 \* Shp1[i].Y \* Shp2[i].Y;

B[1, 0] += 2 \* Shp1[i].X \* Shp2[i].Y - 2 \* Shp2[i].X \* Shp1[i].Y;

B[2, 0] += 2 \* Shp1[i].X;

B[3, 0] += 2 \* Shp1[i].Y;

}

Matrix Ainv = A.Inverse;

Matrix Res = Ainv \* B;

Transformation T = new Transformation();

T.A = Res[0, 0];

T.B = Res[1, 0];

T.T1 = Res[2, 0];

T.T2 = Res[3, 0];

return T;

}

public static double ComputeCost(List<Point> P1List, List<Point> P2List, Transformation T)

{

double cost = 0;

for (int i = 0; i < P1List.Count; i++)

{

double xprime = T.A \* P2List[i].X + T.B \* P2List[i].Y + T.T1;

double yprime = -1 \* T.B \* P2List[i].X + T.A \* P2List[i].Y + T.T2;

cost += (P1List[i].X - xprime) \* (P1List[i].X - xprime) + (P1List[i].Y - yprime) \* (P1List[i].Y - yprime);

}

return cost;

}

}

}

**Conclusion:**

After concluding this assignment, I have to say I understand better how to eliminate noise or outliners from a shape or image. The first part of the assignment helped me to learn a simple technique that can be applied to small samples of data in order to remove outliners. This Exhaustive technique consists in calculating the cost of each point and remove the ones with more error to further apply the transformation to the new model (without outliners).

On the other hand, this assignment also helped me to understand better how the RANSAC algorithm works and the importance of removing outliners while trying to find correspondences in image registration. I now know how to apply RANSAC with the proper thresholds and minimum data values in order to obtain a much more accurate model for image registration to further apply the determined transformation. I was also able to realize how much better work can RANSAC do if we compare it with Exhaustive removal, we can easily it by comparing the error for both techniques RANSAC < 1 Exhaustive removal= 73.