Goncalo Mendes

Computer Vision – Assignment 6

In this assignment, we applied two different methods of outlier removal, using Exhaustive Evaluation in the first part and the RANSAC algorithm in the second part. The difference in the application is that the exhaustive evaluation method is not suitable for large data sets as it would consume too much time as the RANSAC is more appropriate for bigger samples.

The goal is the same, find possible outliers in the data, or bad points that are not what is intended to be represented. That is something we have to deal with since any system can give us lower quality inputs in several occasions, making it important to be able to detect when we have a point that doesn’t belong to our set of points, for example.

For the first example, using the exhaustive evaluation method, we take 2 points at a time and we compute the cost of the model (transformation applied on the shape based on the components a,b,t1,t2. After, if taking those points increase the cost, we put them back, otherwise, we throw them out. We save the running costs so we are able to compare every time we attempt taking out a new pair of points. That process is repeated a certain amount of times until we find the best model.

Here is how it would look after computing one transformation with the initial shapes, and then with the model transformation after outlier removal.

After computing the first transformation, we get an error of about 4000 units, and after running the outlier removal, our transformation is noticeably better with an error of around 78 only, and the shapes being very close.

Source Code:

ICPTransformation class – with outlier removal method coded.

using Mapack;

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Outlier\_Removal\_1

{

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++)

{

int x1 = Shp1[i].X;

int x2 = Shp2[i].X;

int y1 = Shp1[i].Y;

int y2 = Shp2[i].Y;

A[0, 0] += 2 \* x2 \* x2 + 2 \* y2 \* y2;

A[1, 0] += 0;

A[2, 0] += 2 \* x2;

A[3, 0] += 2 \* y2;

A[0, 1] += 0;

A[1, 1] += 2 \* x2 \* x2 + 2 \* y2 \* y2;

A[2, 1] += 2 \* y2;

A[3, 1] += 2 \* x2 \* (-1);

A[0, 2] += 2 \* x2;

A[1, 2] += 2 \* y2;

A[2, 2] += 2;

A[3, 2] += 0;

A[0, 3] += 2 \* y2;

A[1, 3] += 2 \* x2 \* (-1);

A[2, 3] += 0;

A[3, 3] += 2;

B[0, 0] += 2 \* x2 \* x1 + 2 \* y1 \* y2;

B[1, 0] += 2 \* x1 \* y2 - 2 \* x2 \* y1;

B[2, 0] += 2 \* x1;

B[3, 0] += 2 \* y1;

}

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;

}

public static int OutlineRemoval(List<Point> P1List, List<Point> P2List, Transformation T)

{

double max = 0;

int Outlier\_Index = 0;

double[] cost = new double[P1List.Count];

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

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

{

List<Point> list1 = new List<Point>();

List<Point> list2 = new List<Point>();

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[i] = (P1List[i].X - xprime) \* (P1List[i].X - xprime) +

(P1List[i].Y - yprime) \* (P1List[i].Y - yprime);

if (cost[i] > max)

{

max = cost[i];

Outlier\_Index = i;

}

}

return Outlier\_Index;

}

}

}

Form 1 – Calling method and using computations from class ICPTransformation

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;

using static Outlier\_Removal\_1.ICPTransformation;

namespace Outlier\_Removal\_1

{

public partial class Form1 : Form

{

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

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

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

Transformation T = new Transformation();

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e) // Initialise Shapes Button

{

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 = panShape1.CreateGraphics();

DisplayShape(Shape1, pBlue, g);

DisplayShape(Shape2, pRed, g);

}

private void button2\_Click(object sender, EventArgs e) // Apply Transformation Button

{

T = ICPTransformation.ComputeTransformation(Shape1, Shape2);

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

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

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

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

Graphics g = panShape2.CreateGraphics();

DisplayShape(Shape1, pBlue, g);

DisplayShape(Shape2T, pRed, g);

}

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) // Outlier Removal Button

{

List<Point> ResList1 = new List<Point>();

List<Point> ResList2 = new List<Point>();

int Outlier\_Index = OutlineRemoval(Shape1, Shape2, T);

int counter = 0;

while (counter < Shape1.Count - 1)

{

if (counter == Outlier\_Index) continue;

ResList1.Add(Shape1[counter]);

ResList2.Add(Shape2[counter]);

counter++;

}

Transformation Tfinal = ICPTransformation.ComputeTransformation(ResList1, ResList2);

List<Point> Shape2T = ApplyTransformation(Tfinal, ResList2);

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

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

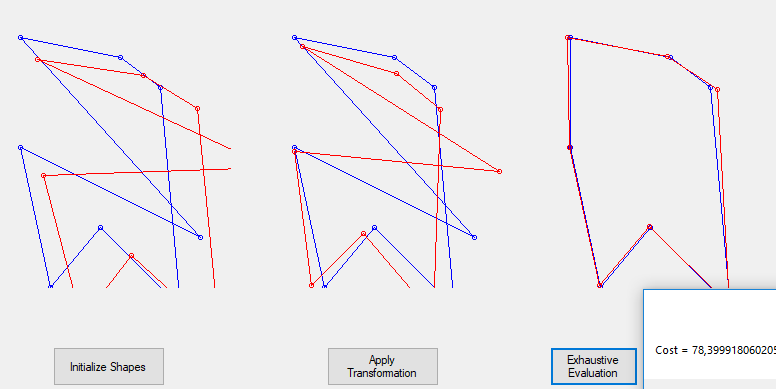
Graphics g = panShape3.CreateGraphics();

DisplayShape(ResList1, pBlue, g);

DisplayShape(Shape2T, pRed, g);

MessageBox.Show("Cost = " + ICPTransformation.ComputeCost(ResList1, ResList2, Tfinal).ToString());

}



For the second part, we implemented the RANSAC algorithm to have a more complex and effective way of removing outliers in case we are handling a large set of data points. Doing exhaustive evaluation for large sets is not the best option as we have to think of the trade-off between running time and the results we want.

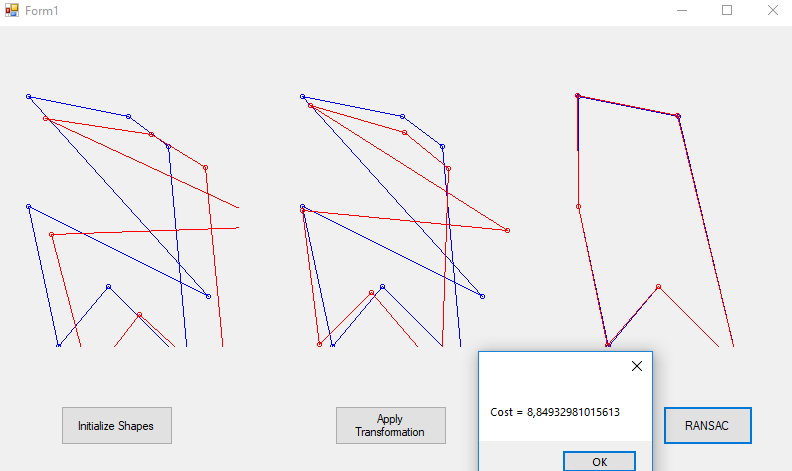
The RANSAC algorithm uses a technique called random sampling. It takes a random sample of a considerably short size within the whole dataset, and it computes the error for that sample. Now, with this basis sample, we know take a random pair of points and we add them to the sample under test. To do this, in our example, we take a random index of our list of initial shapes. With this pair of points added to our sample we are going to compute a new model and check if its cost is below a certain pre-determined threshold. If it is below, then it means we can keep the point, if not we should remove the point and move on to try a new random pair of points in the model.

After doing this, we check if by adding this pair of points, the sample has reached a certain pre-determined size since we started the process. If it has added points up to the considered acceptable size, we compute the final model transformation.

We repeat this process a pre-determined amount of times which will depend on the size of our data, and the precision we need.

Below are the results, which visually gets even better results, as we can notice the shapes are even closer together, nearly coinciding, as the previous method but with a lower error. Also, if the dataset was very large, which is not the case, then we would notice bigger visual differences in the sketches post transformation and the cost difference would be even bigger.

As we can see below, we get the error very close to 8 and the shapes overlap.



Below is the method for the RANSAC algorithm and the button to run it off the Form 1.

RANSAC RansacOutlierRemoval()

{

RANSAC rsc = new RANSAC();

List<Point> tmp1 = new List<Point>();

List<Point> tmp2 = new List<Point>();

int attempts=0;

int k = 30; // Actual number of attempts configured.

int threshold = 5; // Threshold measured on top of the cost, above which we will not add the new point to model.

int d = 4; // Minimum number of points to accept proposed model.

int sample = 3; // Initial random points at the start of the process.

double cost = int.MaxValue;

double attempt\_error;

int[] rand\_ind = new int[sample];

while (attempts<k)

{

double running\_cost;

Point tmp\_random1 = new Point();

Point tmp\_random2 = new Point();

Transformation R = new Transformation();

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

{

Random r = new Random();

int randomIndex = r.Next(Shape1.Count);

while (rand\_ind.Contains(randomIndex))

{

randomIndex = r.Next(Shape1.Count);

}

tmp\_random1 = Shape1[randomIndex];

tmp\_random2 = Shape2[randomIndex];

tmp1.Add(tmp\_random1);

tmp2.Add(tmp\_random2);

rand\_ind[i] = randomIndex;

}

R = ICPTransformation.ComputeTransformation(tmp1, tmp2);

running\_cost = ICPTransformation.ComputeCost(tmp1, tmp2, R);

List<Point> running\_consensus\_set1 = new List<Point>(tmp1);

List<Point> running\_consensus\_set2 = new List<Point>(tmp2);

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

{

if (rand\_ind.Contains(j)) continue; // random index has been used and point is on model list already.

else // New random index - Add point to list and compute cost of hypothetical model.

{

//tmp1.Add(Shape1[j]);

//tmp2.Add(Shape2[j]);

running\_consensus\_set1.Add(Shape1[j]);

running\_consensus\_set2.Add(Shape2[j]);

Transformation tmp\_model = ICPTransformation.ComputeTransformation(running\_consensus\_set1, running\_consensus\_set2);

double tmp\_cost = ICPTransformation.ComputeCost(running\_consensus\_set1, running\_consensus\_set2, tmp\_model);

if (Math.Abs(running\_cost - tmp\_cost) >= threshold)

{

running\_consensus\_set1.RemoveAt(running\_consensus\_set1.Count - 1);

running\_consensus\_set2.RemoveAt(running\_consensus\_set2.Count - 1);

}

}

}

if (running\_consensus\_set1.Count > d)

{

Transformation transf\_model = ICPTransformation.ComputeTransformation(running\_consensus\_set1, running\_consensus\_set2);

attempt\_error = ICPTransformation.ComputeCost(running\_consensus\_set1, running\_consensus\_set2, transf\_model);

if (rsc.best\_error != running\_cost) rsc.best\_error = running\_cost;

if ( attempt\_error < cost ) // First iteration will go in here, as cost is set to high value.

{

cost = attempt\_error;

rsc.best\_consensus1 = running\_consensus\_set1;

rsc.best\_consensus2 = running\_consensus\_set2;

rsc.best\_model = transf\_model;

rsc.best\_error = attempt\_error;

}

}

attempts = attempts + 1;

}

return rsc;

}

private void button4\_Click(object sender, EventArgs e) // RANSAC Outlier Removal

{

RANSAC rsc = new RANSAC();

rsc = RansacOutlierRemoval();

List<Point> res1 = new List<Point>();

List<Point> res2 = new List<Point>();

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

if(rsc.best\_consensus1.Contains(Shape1[i])) {

res1.Add(Shape1[i]);

res2.Add(Shape2[i]);

}

}

List<Point> result\_shape2 = ApplyTransformation(rsc.best\_model, res2);

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

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

Graphics g = panShape3.CreateGraphics();

DisplayShape(res1, pBlue, g);

DisplayShape(result\_shape2, pRed, g);

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

}

}

}