ALGORITHIM PROJECT

Mentor

DR. Mirna Al-Shetairy

2021170611	SC	هنا عبد الرازق فوزي عبد الرازق	93
2022170060	CS	اسلام عمرو عبدالعزيز السيد	
2022170259	CS	علي بسام المصري	
2022170282	CS	عمر محمد سعيد محمد	
2022170425	CS	مصطفى محمد مجدى مصطفى	

Graph construction code

```
public static List<Edge> BuildEdges(RGBPixel[,] img, Func<RGBPixel,
byte> channelSelector)
{


    int height = ImageOperations.GetHeight(img);
    int width = ImageOperations.GetWidth(img);
    var edges = new List<Edge>(height * width *4 );

    int[] dx = { -1, -1, -1, 0, 0, 1, 1, 1 };
    int[] dy = { -1, 0, 1, -1, 1, -1, 0, 1 };

      //

    for (int y = 0; y < height; y++)
    {
        for (int x = 0; x < width; x++)
        {
            int node = y * width + x;
            byte color = channelSelector(img[y, x]);

            for (int k = 0; k < dx.Length; k++)
            {
                int nx = x + dx[k];
                int ny = y + dy[k];
                if (nx < 0 || ny < 0 || nx >= width || ny >= height)
                    continue;



                int node2 = ny * width + nx;

                if (node2 < node) continue;
                byte color2 = channelSelector(img[ny, nx]);
                int weight = Math.Abs(color - color2);


                edges.Add(new Edge(node, node2, weight));
            }
        }
    }

    return edges;
}
```

Image segmentation code

```
public class Segmenter
{
private readonly RGBPixel[,] img;
private readonly int width;
private readonly int height;
private readonly long k;


public Segmenter(RGBPixel[,] image, long kValue)
{
img = image;
height = ImageOperations.GetHeight(image);
width = ImageOperations.GetWidth(image);
k = kValue;
}

private int[,] RunMonoChannel(Func<RGBPixel, byte> selector)
{

int total = width * height;
var edges = GraphBuilder.BuildEdges(img,selector);
edges.Sort((a, b) => a.Weight.CompareTo(b.Weight));
var dsu = new DisjointSet(total);

foreach (var e in edges)
{
int a = dsu.FindLeader(e.U);
int b = dsu.FindLeader(e.V);
if (a == b) continue;

double ta = k / dsu.GetSize(a);
double tb = k / dsu.GetSize(b);
double ma = dsu.InternalDiff(a) + ta;
double mb =dsu.InternalDiff(b) + tb;

if (e.Weight <=Math.Min(ma, mb))
{
dsu.Union(a, b, e.Weight);
}

}

var leaders = new int[height, width];


Parallel.For(0, height, y =>
{
```

```
for (int x = 0; x < width; x++)
{
int id = y * width + x;
leaders[y, x] = dsu.FindLeader(id);
}
});
return leaders;
}


int comp;
public int[,] RunColor()
{
// Segmentation on R, G, B
int[,] lr = null, lg = null, lb = null;


Parallel.Invoke(
() => { lr = RunMonoChannel(p => p.red); },
() => { lg = RunMonoChannel(p => p.green); },
() => { lb = RunMonoChannel(p => p.blue); }
);

int total = width * height;

var finalDsu = new DisjointSet(total);

int[] dx = { 1, 1, -1, -1, 0, 1, 0, -1 };
int[] dy = { 1, -1, 1, -1, 1, 0, -1, 0 };

for (int y = 0; y < height; y++)
{
for (int x = 0; x < width; x++)
{

int id = y * width + x;
for (int i = 0; i < dx.Length; i++)
{
int nx = x + dx[i], ny = y + dy[i];
if (nx < 0 || ny < 0 || nx >= width || ny >= height) continue;

int nid = ny * width + nx;
if (lr[y, x] == lr[ny, nx] &&
lg[y, x] == lg[ny, nx] &&
lb[y, x] == lb[ny, nx])
{
finalDsu.Union(id, nid, 0);
}}}
```

```
var finalLeaders = new int[height, width];


Parallel.For(0, height, y =>
{
for (int x = 0; x < width; x++)
{
int id = y * width + x;
finalLeaders[y, x] =finalDsu.FindLeader(id);
}
});
return finalLeaders;
}
public (int count, List<int> sizes) GetStats(int[,] leaders)
{
var freq = new Dictionary<int, int>();
int h = leaders.GetLength(0);
int w = leaders.GetLength(1);

for (int y = 0; y < h; y++)
for (int x = 0; x < w; x++)
{
int leader = leaders[y, x];
if (!freq.ContainsKey(leader)) freq[leader] = 0;
freq[leader]++;
}


var sizes = new List<int>(freq.Values);
sizes.Sort((a, b) => b.CompareTo(a));
return (sizes.Count, sizes);
}
```

Helper Function for Image segmentation (DisjointSet)

```
public DisjointSet(int n)
{
componants = n;
parent = new int[n];
groupSize = new int[n];
internalDiff = new double[n];

for (int i = 0; i < n; i++)
{
parent[i] = i;
groupSize[i] = 1;
internalDiff[i] = 0.0;
}
}

public int GetComponants()
{
return componants;
}
public int FindLeader(int x)
{
if (parent[x] == x) return x;

return parent[x] = FindLeader(parent[x]);
}

public void Union(int x, int y, double weight)
{
int leader1 = FindLeader(x);
int leader2 = FindLeader(y);

if (leader1 == leader2) return;

if (GetSize(leader1)<GetSize(leader2))
{
//swap
(leader2, leader1) = (leader1, leader2);
}

parent[leader2] = leader1;
groupSize[leader1] += groupSize[leader2];

internalDiff[leader1] = Math.Max(Math.Max(internalDiff[leader1],
internalDiff[leader2]), weight);
componants--;
}
public int GetSize(int x);

return groupSize[FindLeader(x)];
}
public double InternalDiff(int x)
{
return internalDiff[FindLeader(x)];
}
```

Helper Function for Image segmentation (Edge)

```
struct Edge
{
    public int U, V;
    public int Weight;
    public Edge(int u, int v, int w) { U = u; V = v; Weight = w; }
}
```

Image segmentation visualization code

```csharp
public RGBPixel[,] Colorize(int[,] leaders)
{
var rnd = new Random();
var colors = new Dictionary<int, Color>();

int h = leaders.GetLength(0);
int w = leaders.GetLength(1);



RGBPixel[,] mat = new RGBPixel[h, w];

for (int y = 0; y < h; y++)
for (int x = 0; x < w; x++)
{
int leader = leaders[y, x];
if (!colors.ContainsKey(leader))
colors[leader] = Color.FromArgb(rnd.Next(256), rnd.Next(256),
rnd.Next(256));



mat[y, x].red = colors[leader].R;
mat[y,x].green = colors[leader].G;
mat[y,x].blue = colors[leader].B;
}
return mat;
}
```

ANALYSIS

Graph construction code

```
public static List<Edge> BuildEdges(RGBPixel[,] img, Func<RGBPixel,
byte> channelSelector)
{
      // O(1) FOR ALL

    int height = ImageOperations.GetHeight(img);
    int width = ImageOperations.GetWidth(img);
    var edges = new List<Edge>(height * width *4 );

    int[] dx = { -1, -1, -1, 0, 0, 1, 1, 1 };
    int[] dy = { -1, 0, 1, -1, 1, -1, 0, 1 };

      //

    for (int y = 0; y < height; y++)
    {
        for (int x = 0; x < width; x++)
        {
    // TOTAL COMPLIXITY FOR LOOPS O(N) ; N = pixels
            int node = y * width + x;
            byte color = channelSelector(img[y, x]); // o(1)

            for (int k = 0; k < dx.Length; k++) // o(1)
            {
                int nx = x + dx[k]; // o(1)
                int ny = y + dy[k]; // o(1)
                if (nx < 0 || ny < 0 || nx >= width || ny >= height)
                    continue; // o(1)



                int node2 = ny * width + nx; // o(1)

                if (node2 < node) continue; // o(1)
                byte color2 = channelSelector(img[ny, nx]); // o(1)
                int weight = Math.Abs(color - color2); // o(1)


                edges.Add(new Edge(node, node2, weight)); // o(1)
            }
        }
    }

    return edges;
}
// TOTAL COMPIXTY O(N)
```

Image segmentation code

```
public class Segmenter
{   // O(1)
private readonly RGBPixel[,] img;
private readonly int width;
private readonly int height;
private readonly long k;


public Segmenter(RGBPixel[,] image, long kValue)
{
// O(1)
img = image;
height = ImageOperations.GetHeight(image);
width = ImageOperations.GetWidth(image);
k = kValue;
}

private int[,] RunMonoChannel(Func<RGBPixel, byte> selector) // n log n
{

int total = width * height; // O(1)
var edges = GraphBuilder.BuildEdges(img,selector); //n
edges.Sort((a, b) => a.Weight.CompareTo(b.Weight)); // n log n
var dsu = new DisjointSet(total); //n

foreach (var e in edges) //n
{
int a = dsu.FindLeader(e.U); //log n
int b = dsu.FindLeader(e.V); //log n
if (a == b) continue; // 1

// O(1)
double ta = k / dsu.GetSize(a);
double tb = k / dsu.GetSize(b);
double ma = dsu.InternalDiff(a) + ta;
double mb =dsu.InternalDiff(b) + tb;

if (e.Weight <=Math.Min(ma, mb))
{
dsu.Union(a, b, e.Weight);
}

}

var leaders = new int[height, width]; // O(N)

//nlogn for loop
```

```
Parallel.For(0, height, y =>
{
for (int x = 0; x < width; x++)
{
int id = y * width + x;
leaders[y, x] = dsu.FindLeader(id); // log N
}
});
return leaders;
}


int comp;
public int[,] RunColor()
{
// Segmentation on R, G, B
int[,] lr = null, lg = null, lb = null;


Parallel.Invoke( // n log n per each which is total NlogN
() => { lr = RunMonoChannel(p => p.red); },
() => { lg = RunMonoChannel(p => p.green); },
() => { lb = RunMonoChannel(p => p.blue); }
);


//var l = RunCombinedRGB();
int total = width * height;

var finalDsu = new DisjointSet(total); // n

int[] dx = { 1, 1, -1, -1, 0, 1, 0, -1 };
int[] dy = { 1, -1, 1, -1, 1, 0, -1, 0 };

for (int y = 0; y < height; y++)
{
for (int x = 0; x < width; x++)
{
/// BOTH LOOPS O(N)
int id = y * width + x;
for (int i = 0; i < dx.Length; i++) // O(1)
{
int nx = x + dx[i], ny = y + dy[i];
if (nx < 0 || ny < 0 || nx >= width || ny >= height) continue;

int nid = ny * width + nx;
if (lr[y, x] == lr[ny, nx] &&
lg[y, x] == lg[ny, nx] &&
lb[y, x] == lb[ny, nx])
{
finalDsu.Union(id, nid, 0); // log n
}


}
}
/// TOTAL N LOG (N)
}



var finalLeaders = new int[height, width]; // N

//nlogn for loop

Parallel.For(0, height, y =>
{
for (int x = 0; x < width; x++)
{
int id = y * width + x;
finalLeaders[y, x] =finalDsu.FindLeader(id); // log n
}
});
return finalLeaders;
}
public (int count, List<int> sizes) GetStats(int[,] leaders)
{
var freq = new Dictionary<int, int>(); // O(1)
int h = leaders.GetLength(0);
int w = leaders.GetLength(1);

for (int y = 0; y < h; y++)
for (int x = 0; x < w; x++)
{
int leader = leaders[y, x];
if (!freq.ContainsKey(leader)) freq[leader] = 0;
freq[leader]++;
}
// Total loop complexity: O(n), where n = h * w


var sizes = new List<int>(freq.Values); // O(s), s = number of unique
leaders (segments)
sizes.Sort((a, b) => b.CompareTo(a)); // s log s
return (sizes.Count, sizes); // O(1)
}
// TOTAL COMPLIXITY O(n + s log s)
```

Helper Function for Image segmentation (DisjointSet)

```
public DisjointSet(int n)
{
componants = n; // O(1)
parent = new int[n];// O(N)
groupSize = new int[n];// O(N)
internalDiff = new double[n];// O(N)

for (int i = 0; i < n; i++) // O(N)
{
parent[i] = i;
groupSize[i] = 1;
internalDiff[i] = 0.0;
}
}

public int GetComponants() //O(1)
{
return componants;
}
public int FindLeader(int x) //O(Log N)
{
if (parent[x] == x) return x;

return parent[x] = FindLeader(parent[x]);//O(Log N)
}

public void Union(int x, int y, double weight)
{
int leader1 = FindLeader(x); //O(Log N)
int leader2 = FindLeader(y); //O(Log N)

if (leader1 == leader2) return; // O(1)

if (GetSize(leader1)<GetSize(leader2))
{
//swap
(leader2, leader1) = (leader1, leader2); // O(1)
}

parent[leader2] = leader1;// O(1)
groupSize[leader1] += groupSize[leader2];// O(1)

internalDiff[leader1] = Math.Max(Math.Max(internalDiff[leader1],
internalDiff[leader2]), weight);// O(1)
componants--;// O(1)
}
public int GetSize(int x)//O(Log N)
{
return groupSize[FindLeader(x)];//O(Log N)
}
public double InternalDiff(int x)//O(Log N)
{
return internalDiff[FindLeader(x)];//O(Log N)
}
```

Helper Function for Image segmentation (Edge)

```
struct Edge // O(1)
{
    public int U, V;
    public int Weight;
    public Edge(int u, int v, int w) { U = u; V = v; Weight = w; }
}
```

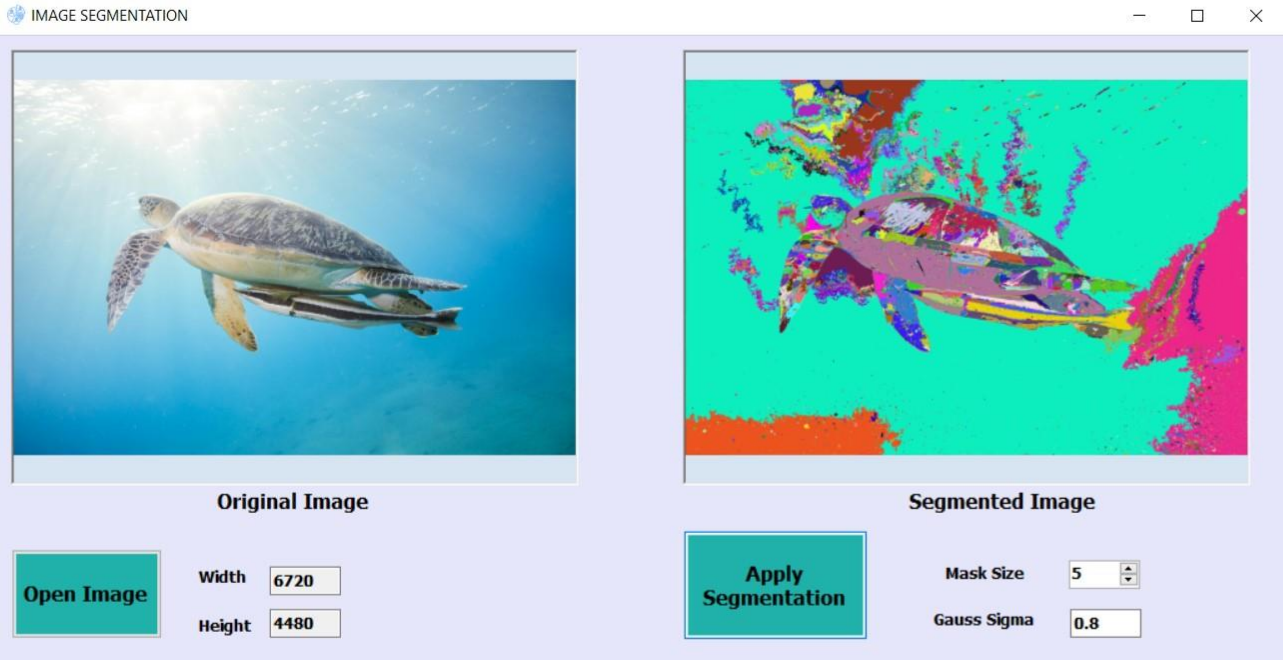
Image segmentation visualization code

```
public RGBPixel[,] Colorize(int[,] leaders)
{
var rnd = new Random(); // O(1)
var colors = new Dictionary<int, Color>(); // O(1)

int h = leaders.GetLength(0); // O(1)
int w = leaders.GetLength(1); // O(1)



RGBPixel[,] mat = new RGBPixel[h, w]; // n

for (int y = 0; y < h; y++)
for (int x = 0; x < w; x++)
{
int leader = leaders[y, x]; // O(1)
if (!colors.ContainsKey(leader))
colors[leader] = Color.FromArgb(rnd.Next(256), rnd.Next(256),
rnd.Next(256));
// O(s), s = number of unique leaders (segments)



mat[y, x].red = colors[leader].R;  // O(1)
mat[y,x].green = colors[leader].G; // O(1)
mat[y,x].blue = colors[leader].B; // O(1)
}
return mat;

// TOTAL COMPLIXITY O(N)
}
```

Running code

```
private void btnGaussSmooth_Click(object sender, EventArgs e)
{
double sigma = double.Parse(txtGaussSigma.Text);
int maskSize = (int)nudMaskSize.Value ;
ImageMatrix = ImageOperations.GaussianFilter1D(ImageMatrix, maskSize,
sigma);


Stopwatch timer = Stopwatch.StartNew();


var segmenter = new Segmenter(ImageMatrix, 30000); //O(1)
int[,] leaders = segmenter.RunColor(); // N Log N

var ImageMatrix2 = segmenter.Colorize(leaders); // O(N)

var (count, sizes) = segmenter.GetStats(leaders); // O(n + s log s)
timer.Stop();

long time = timer.ElapsedMilliseconds;

Debug.WriteLine("TIME:" + time);


ImageOperations.DisplayImage(ImageMatrix2, pictureBox2);




string outputPath = @"D:\Algorithims project\Image-
Segmentation\ImageSegmentation\ImageSegmentation\MyOutput.txt";

using (var sw = new StreamWriter(outputPath, false)) // O(leaders) ;
{
sw.WriteLine(count);

foreach (var s in sizes)
sw.WriteLine(s);
}


SaveFileDialog saveFileDialog1 = new SaveFileDialog();
saveFileDialog1.Filter = "bmp files (*.bmp)|*.bmp|All files (*.*)|*.*";
saveFileDialog1.RestoreDirectory = true;
if (saveFileDialog1.ShowDialog() == DialogResult.OK)
{
pictureBox2.Image.Save(saveFileDialog1.FileName, ImageFormat.Bmp);}}
```

OUTPUT SAMPLES

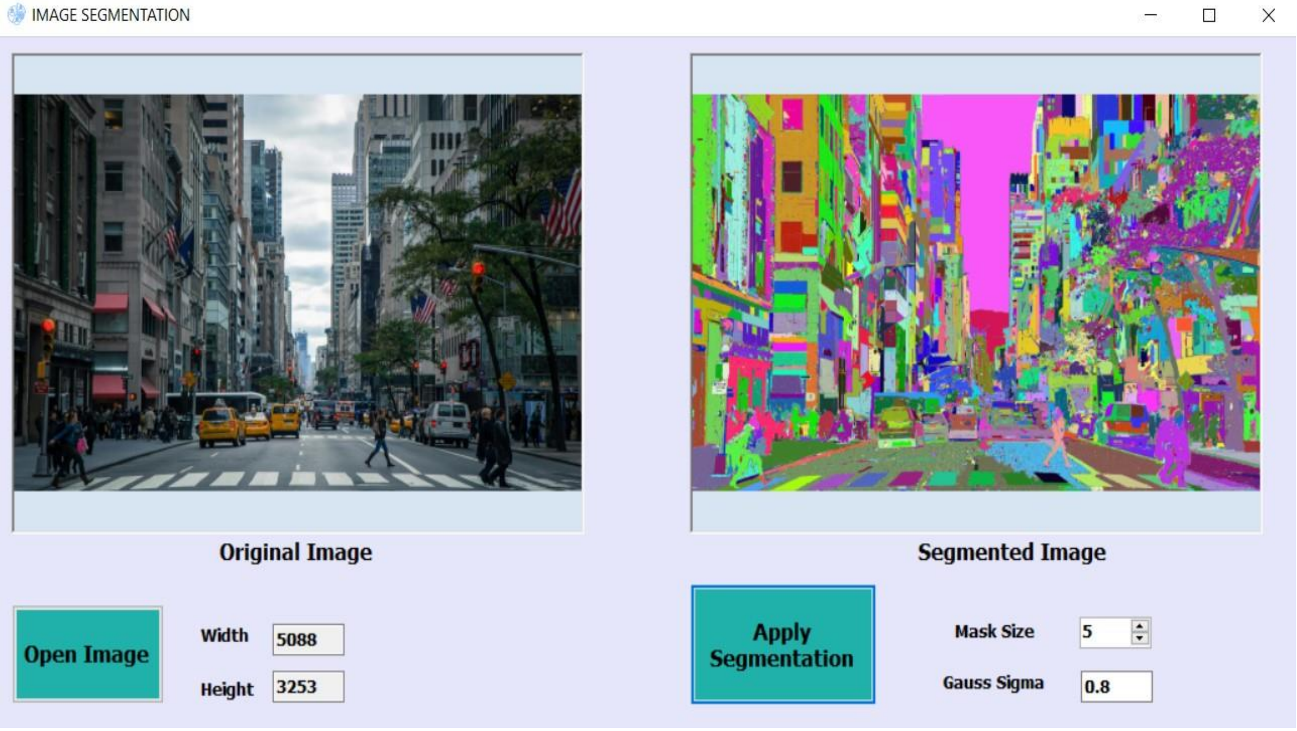


Width 6720

Height 4480

Mask Size 5

Gauss Sigma 0.8

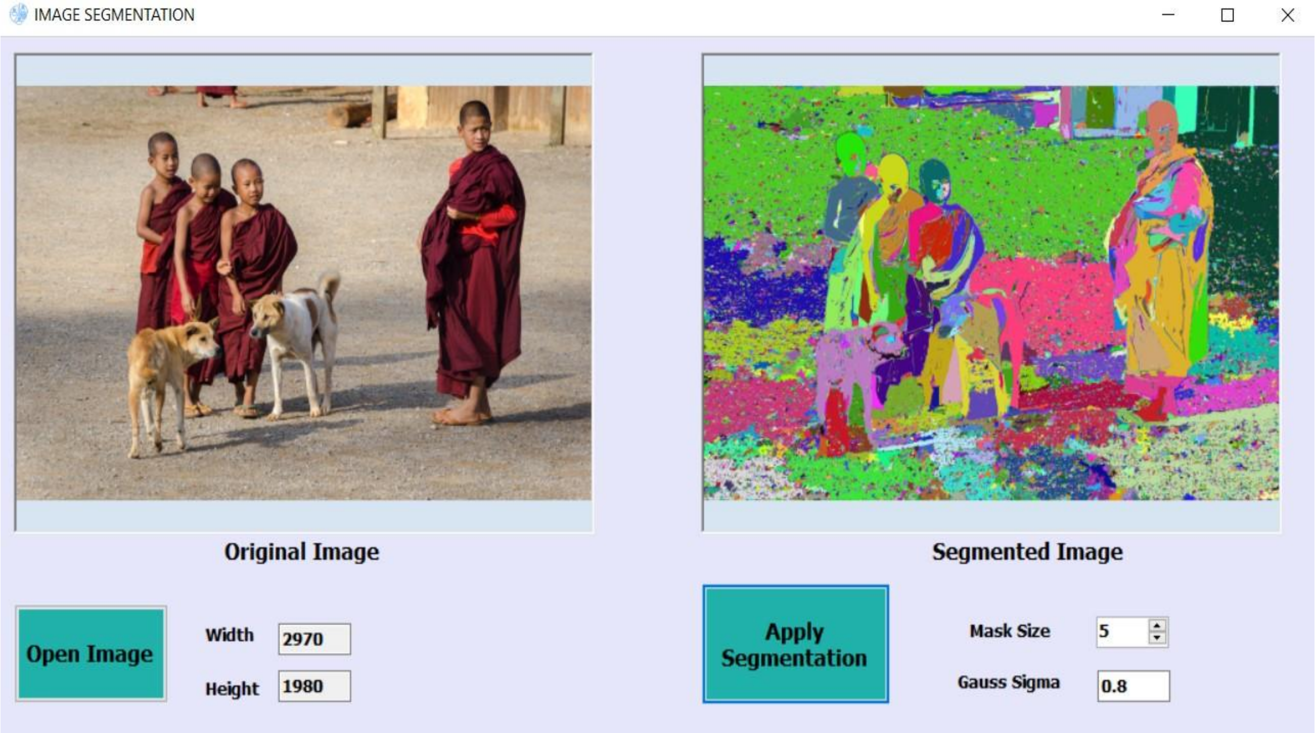


Width 5088

Height 3253

Mask Size 5

Gauss Sigma 0.8

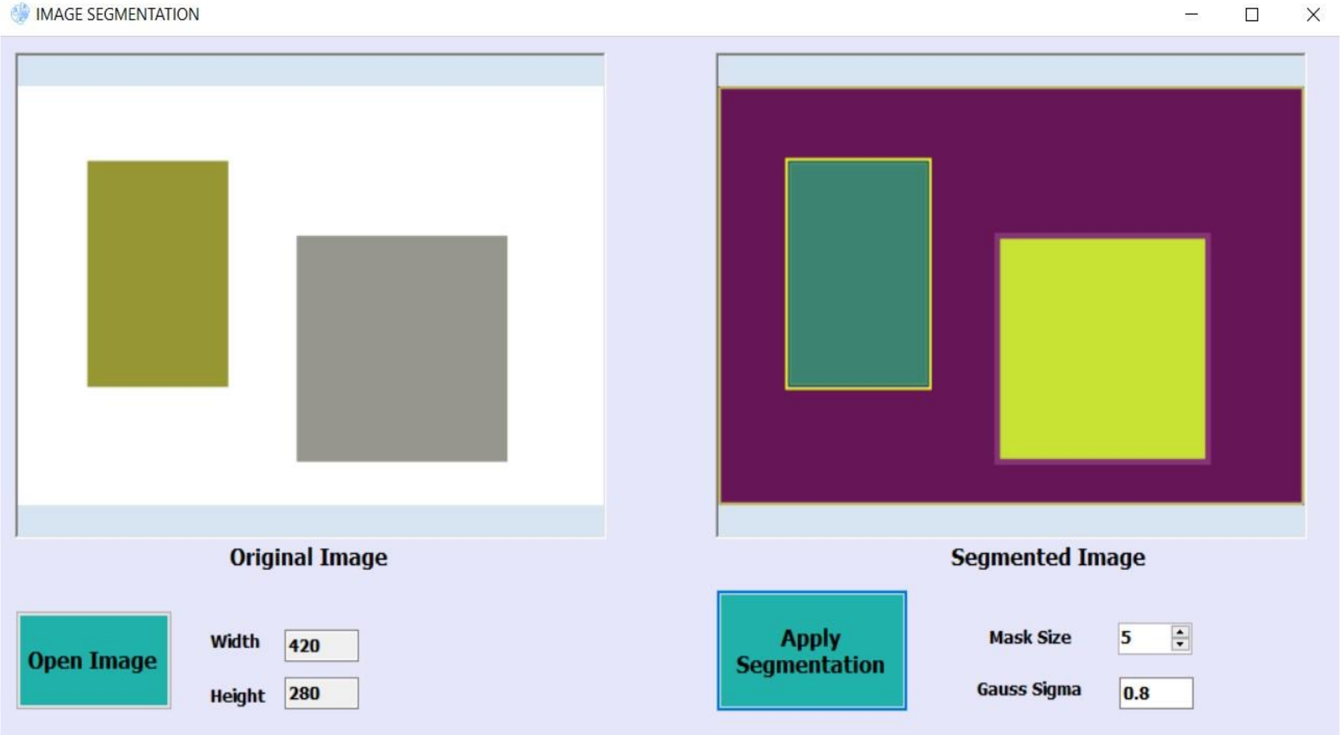


Width 2970

Height 1980

Mask Size 5

Gauss Sigma 0.8



Width 420

Height 280

Mask Size 5

Gauss Sigma 0.8