**Programación en C#**

**A1. Imagen con Primer Filtro**

float TH, TL, Sigma;

int MaskSize;

TH = (float)Convert.ToDouble(TxtTH.Text); //gradiente máximo

TL = (float)Convert.ToDouble(TxtTL.Text); //gradiente mínimo

MaskSize = Convert.ToInt32(TxtGMask.Text); //valor de filtrado

Sigma = (float)Convert.ToDouble(TxtSigma.Text); //constante de filtrado

CannyData = new Canny((Bitmap)pictureBox1.Image, TH, TL, MaskSize, Sigma); //selección de imagen a ser procesada

pictureBox2.Image = CannyData.DisplayImage(CannyData.FilteredImage); //mostrar imagen en el espacio del picturebox2

**A2. Código de Canny**

float TH, TL, Sigma;

int MaskSize;

TH = (float)Convert.ToDouble(TxtTH.Text); //gradiente máximo

TL = (float)Convert.ToDouble(TxtTL.Text); //gradiente mínimo

MaskSize = Convert.ToInt32(TxtGMask.Text); //valor de filtrado

Sigma = (float)Convert.ToDouble(TxtSigma.Text); //constante de filtrado

CannyData = new Canny((Bitmap)pictureBox1.Image, TH, TL, MaskSize, Sigma); //selección de imagen a ser procesada

pictureBox2.Image = CannyData.DisplayImage(CannyData.GNH); //mostrar imagen en el espacio del picturebox2

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Drawing.Imaging;

using System.Threading;

namespace S7\_net\_example

{

class Canny

{

// static TimeSpan waitTime2 = new TimeSpan(0, 0, 2);

public int Width, Height;

public Bitmap Obj;

public int[,] GreyImage;

//Gaussian Kernel Data

int[,] GaussianKernel;

int KernelWeight;

int KernelSize = 5;

float Sigma = 1; // for N=2 Sigma =0.85 N=5 Sigma =1, N=9 Sigma = 2 2\*Sigma = (int)N/2

//Canny Edge Detection Parameters

float MaxHysteresisThresh, MinHysteresisThresh;

public float[,] DerivativeX;

public float[,] DerivativeY;

public int[,] FilteredImage;

public float[,] Gradient;

public float[,] NonMax;

public int[,] PostHysteresis;

int[,] EdgePoints;

public float[,] GNH;

public float[,] GNL;

public int[,] EdgeMap;

public int[,] VisitedMap;

public Canny(Bitmap Input)

{

// Gaussian and Canny Parameters

MaxHysteresisThresh = 20F;

MinHysteresisThresh = 10F;

Obj = Input;

Width = Obj.Width;

Height = Obj.Height;

EdgeMap = new int[Width, Height];

VisitedMap = new int[Width, Height];

ReadImage();

DetectCannyEdges();

return;

}

public Canny(Bitmap Input, float Th, float Tl)

{

// Gaussian and Canny Parameters

MaxHysteresisThresh = Th;

MinHysteresisThresh = Tl;

Obj = Input;

Width = Obj.Width;

Height = Obj.Height;

EdgeMap = new int[Width, Height];

VisitedMap = new int[Width, Height];

ReadImage();

DetectCannyEdges();

return;

}

public Canny(Bitmap Input, float Th, float Tl, int GaussianMaskSize, float SigmaforGaussianKernel)

{

// Gaussian and Canny Parameters

MaxHysteresisThresh = Th;

MinHysteresisThresh = Tl;

KernelSize = GaussianMaskSize;

Sigma = SigmaforGaussianKernel;

Obj = Input;

Width = Obj.Width;

Height = Obj.Height;

EdgeMap = new int[Width, Height];

VisitedMap = new int[Width, Height];

ReadImage();

DetectCannyEdges();

return;

}

public Bitmap DisplayImage()

{

int i, j;

Bitmap image = new Bitmap(Obj.Width, Obj.Height);

BitmapData bitmapData1 = image.LockBits(new Rectangle(0, 0, Obj.Width, Obj.Height),

ImageLockMode.ReadOnly, PixelFormat.Format32bppArgb);

unsafe

{

byte\* imagePointer1 = (byte\*)bitmapData1.Scan0;

for (i = 0; i < bitmapData1.Height; i++)

{

for (j = 0; j < bitmapData1.Width; j++)

{

// write the logic implementation here

imagePointer1[0] = (byte)GreyImage[j, i];

imagePointer1[1] = (byte)GreyImage[j, i];

imagePointer1[2] = (byte)GreyImage[j, i];

imagePointer1[3] = (byte)255;

//4 bytes per pixel

imagePointer1 += 4;

}//end for j

//4 bytes per pixel

imagePointer1 += (bitmapData1.Stride - (bitmapData1.Width \* 4));

}//end for i

}//end unsafe

image.UnlockBits(bitmapData1);

return image;// col;

} // Display Grey Image

public Bitmap DisplayImage(float[,] GreyImage)

{

int i, j;

int W, H;

W = GreyImage.GetLength(0);

H = GreyImage.GetLength(1);

Bitmap image = new Bitmap(W, H);

BitmapData bitmapData1 = image.LockBits(new Rectangle(0, 0, W, H),

ImageLockMode.ReadOnly, PixelFormat.Format32bppArgb);

unsafe

{

byte\* imagePointer1 = (byte\*)bitmapData1.Scan0;

for (i = 0; i < bitmapData1.Height; i++)

{

for (j = 0; j < bitmapData1.Width; j++)

{

// write the logic implementation here

imagePointer1[0] = (byte)((int)(GreyImage[j, i]));

imagePointer1[1] = (byte)((int)(GreyImage[j, i]));

imagePointer1[2] = (byte)((int)(GreyImage[j, i]));

imagePointer1[3] = (byte)255;

//4 bytes per pixel

imagePointer1 += 4;

} //end for j

//4 bytes per pixel

imagePointer1 += (bitmapData1.Stride - (bitmapData1.Width \* 4));

}//End for i

}//end unsafe

image.UnlockBits(bitmapData1);

return image;// col;

} // Display Grey Imag

public Bitmap DisplayImage(int[,] GreyImage)

{

int i, j;

int W, H;

W = GreyImage.GetLength(0);

H = GreyImage.GetLength(1);

Bitmap image = new Bitmap(W, H);

BitmapData bitmapData1 = image.LockBits(new Rectangle(0, 0, W, H),

ImageLockMode.ReadOnly, PixelFormat.Format32bppArgb);

unsafe

{

byte\* imagePointer1 = (byte\*)bitmapData1.Scan0;

for (i = 0; i < bitmapData1.Height; i++)

{

for (j = 0; j < bitmapData1.Width; j++)

{

// write the logic implementation here

imagePointer1[0] = (byte)GreyImage[j, i];

imagePointer1[1] = (byte)GreyImage[j, i];

imagePointer1[2] = (byte)GreyImage[j, i];

imagePointer1[3] = (byte)255;

//4 bytes per pixel

imagePointer1 += 4;

} //end for j

//4 bytes per pixel

imagePointer1 += (bitmapData1.Stride - (bitmapData1.Width \* 4));

}//End for i

}//end unsafe

image.UnlockBits(bitmapData1);

return image;// col;

} // Display Grey Image

private void ReadImage()

{

int i, j;

GreyImage = new int[Obj.Width, Obj.Height]; //[Row,Column]

Bitmap image = Obj;

BitmapData bitmapData1 = image.LockBits(new Rectangle(0, 0, image.Width, image.Height),

ImageLockMode.ReadOnly, PixelFormat.Format32bppArgb);

unsafe

{

byte\* imagePointer1 = (byte\*)bitmapData1.Scan0;

for (i = 0; i < bitmapData1.Height; i++)

{

for (j = 0; j < bitmapData1.Width; j++)

{

GreyImage[j, i] = (int)((imagePointer1[0] + imagePointer1[1] + imagePointer1[2]) / 3.0);

//4 bytes per pixel

imagePointer1 += 4;

}//end for j

//4 bytes per pixel

imagePointer1 += bitmapData1.Stride - (bitmapData1.Width \* 4);

}//end for i

}//end unsafe

image.UnlockBits(bitmapData1);

return;

}

private void GenerateGaussianKernel(int N, float S, out int Weight)

{

float Sigma = S;

float pi;

pi = (float)Math.PI;

int i, j;

int SizeofKernel = N;

float[,] Kernel = new float[N, N];

GaussianKernel = new int[N, N];

float[,] OP = new float[N, N];

float D1, D2;

D1 = 1 / (2 \* pi \* Sigma \* Sigma);

D2 = 2 \* Sigma \* Sigma;

float min = 1000;

for (i = -SizeofKernel / 2; i <= SizeofKernel / 2; i++)

{

for (j = -SizeofKernel / 2; j <= SizeofKernel / 2; j++)

{

Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j] = ((1 / D1) \* (float)Math.Exp(-(i \* i + j \* j) / D2));

if (Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j] < min)

min = Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j];

}

}

int mult = (int)(1 / min);

int sum = 0;

if ((min > 0) && (min < 1))

{

for (i = -SizeofKernel / 2; i <= SizeofKernel / 2; i++)

{

for (j = -SizeofKernel / 2; j <= SizeofKernel / 2; j++)

{

Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j] = (float)Math.Round(Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j] \* mult, 0);

GaussianKernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j] = (int)Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j];

sum = sum + GaussianKernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j];

}

}

}

else

{

sum = 0;

for (i = -SizeofKernel / 2; i <= SizeofKernel / 2; i++)

{

for (j = -SizeofKernel / 2; j <= SizeofKernel / 2; j++)

{

Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j] = (float)Math.Round(Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j], 0);

GaussianKernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j] = (int)Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j];

sum = sum + GaussianKernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j];

}

}

}

//Normalizing kernel Weight

Weight = sum;

return;

}

private int[,] GaussianFilter(int[,] Data)

{

GenerateGaussianKernel(KernelSize, Sigma, out KernelWeight);

int[,] Output = new int[Width, Height];

int i, j, k, l;

int Limit = KernelSize / 2;

float Sum = 0;

Output = Data; // Removes Unwanted Data Omission due to kernel bias while convolution

for (i = Limit; i <= ((Width - 1) - Limit); i++)

{

for (j = Limit; j <= ((Height - 1) - Limit); j++)

{

Sum = 0;

for (k = -Limit; k <= Limit; k++)

{

for (l = -Limit; l <= Limit; l++)

{

Sum = Sum + ((float)Data[i + k, j + l] \* GaussianKernel[Limit + k, Limit + l]);

}

}

Output[i, j] = (int)(Math.Round(Sum / (float)KernelWeight));

}

}

return Output;

}

private float[,] Differentiate(int[,] Data, int[,] Filter)

{

int i, j, k, l, Fh, Fw;

Fw = Filter.GetLength(0);

Fh = Filter.GetLength(1);

float sum = 0;

float[,] Output = new float[Width, Height];

for (i = Fw / 2; i <= (Width - Fw / 2) - 1; i++)

{

for (j = Fh / 2; j <= (Height - Fh / 2) - 1; j++)

{

sum = 0;

for (k = -Fw / 2; k <= Fw / 2; k++)

{

for (l = -Fh / 2; l <= Fh / 2; l++)

{

sum = sum + Data[i + k, j + l] \* Filter[Fw / 2 + k, Fh / 2 + l];

}

}

Output[i, j] = sum;

}

}

return Output;

}

private void DetectCannyEdges()

{

Gradient = new float[Width, Height];

NonMax = new float[Width, Height];

PostHysteresis = new int[Width, Height];

DerivativeX = new float[Width, Height];

DerivativeY = new float[Width, Height];

//Gaussian Filter Input Image

FilteredImage = GaussianFilter(GreyImage);

//Sobel Masks

int[,] Dx = {{1,0,-1},

{1,0,-1},

{1,0,-1}};

int[,] Dy = {{1,1,1},

{0,0,0},

{-1,-1,-1}};

DerivativeX = Differentiate(FilteredImage, Dx);

DerivativeY = Differentiate(FilteredImage, Dy);

int i, j;

//Compute the gradient magnitude based on derivatives in x and y:

for (i = 0; i <= (Width - 1); i++)

{

for (j = 0; j <= (Height - 1); j++)

{

Gradient[i, j] = (float)Math.Sqrt((DerivativeX[i, j] \* DerivativeX[i, j]) + (DerivativeY[i, j] \* DerivativeY[i, j]));

}

}

// Perform Non maximum suppression:

// NonMax = Gradient;

for (i = 0; i <= (Width - 1); i++)

{

for (j = 0; j <= (Height - 1); j++)

{

NonMax[i, j] = Gradient[i, j];

}

}

int Limit = KernelSize / 2;

int r, c;

float Tangent;

for (i = Limit; i <= (Width - Limit) - 1; i++)

{

for (j = Limit; j <= (Height - Limit) - 1; j++)

{

if (DerivativeX[i, j] == 0)

Tangent = 90F;

else

Tangent = (float)(Math.Atan(DerivativeY[i, j] / DerivativeX[i, j]) \* 180 / Math.PI); //rad to degree

//Horizontal Edge

if (((-22.5 < Tangent) && (Tangent <= 22.5)) || ((157.5 < Tangent) && (Tangent <= -157.5)))

{

if ((Gradient[i, j] < Gradient[i, j + 1]) || (Gradient[i, j] < Gradient[i, j - 1]))

NonMax[i, j] = 0;

}

//Vertical Edge

if (((-112.5 < Tangent) && (Tangent <= -67.5)) || ((67.5 < Tangent) && (Tangent <= 112.5)))

{

if ((Gradient[i, j] < Gradient[i + 1, j]) || (Gradient[i, j] < Gradient[i - 1, j]))

NonMax[i, j] = 0;

}

//+45 Degree Edge

if (((-67.5 < Tangent) && (Tangent <= -22.5)) || ((112.5 < Tangent) && (Tangent <= 157.5)))

{

if ((Gradient[i, j] < Gradient[i + 1, j - 1]) || (Gradient[i, j] < Gradient[i - 1, j + 1]))

NonMax[i, j] = 0;

}

//-45 Degree Edge

if (((-157.5 < Tangent) && (Tangent <= -112.5)) || ((67.5 < Tangent) && (Tangent <= 22.5)))

{

if ((Gradient[i, j] < Gradient[i + 1, j + 1]) || (Gradient[i, j] < Gradient[i - 1, j - 1]))

NonMax[i, j] = 0;

}

}

}

//PostHysteresis = NonMax;

for (r = Limit; r <= (Width - Limit) - 1; r++)

{

for (c = Limit; c <= (Height - Limit) - 1; c++)

{

PostHysteresis[r, c] = (int)NonMax[r, c];

}

}

//Find Max and Min in Post Hysterisis

float min, max;

min = 100;

max = 0;

for (r = Limit; r <= (Width - Limit) - 1; r++)

for (c = Limit; c <= (Height - Limit) - 1; c++)

{

if (PostHysteresis[r, c] > max)

{

max = PostHysteresis[r, c];

}

if ((PostHysteresis[r, c] < min) && (PostHysteresis[r, c] > 0))

{

min = PostHysteresis[r, c];

}

}

GNH = new float[Width, Height];

GNL = new float[Width, Height]; ;

EdgePoints = new int[Width, Height];

for (r = Limit; r <= (Width - Limit) - 1; r++)

{

for (c = Limit; c <= (Height - Limit) - 1; c++)

{

if (PostHysteresis[r, c] >= MaxHysteresisThresh)

{

EdgePoints[r, c] = 1;

GNH[r, c] = 255;

}

if ((PostHysteresis[r, c] < MaxHysteresisThresh) && (PostHysteresis[r, c] >= MinHysteresisThresh))

{

EdgePoints[r, c] = 2;

GNL[r, c] = 255;

}

}

}

HysterisisThresholding(EdgePoints);

for (i = 0; i <= (Width - 1); i++)

for (j = 0; j <= (Height - 1); j++)

{

EdgeMap[i, j] = EdgeMap[i, j] \* 255;

}

return;

}

private void HysterisisThresholding(int[,] Edges)

{

int i, j;

int Limit = KernelSize / 2;

for (i = Limit; i <= (Width - 1) - Limit; i++)

for (j = Limit; j <= (Height - 1) - Limit; j++)

{

if (Edges[i, j] == 1)

{

EdgeMap[i, j] = 1;

}

}

for (i = Limit; i <= (Width - 1) - Limit; i++)

{

for (j = Limit; j <= (Height - 1) - Limit; j++)

{

if (Edges[i, j] == 1)

{

EdgeMap[i, j] = 1;

Travers(i, j);

VisitedMap[i, j] = 1;

}

}

}

return;

}

private void Travers(int X, int Y)

{

if (VisitedMap[X, Y] == 1)

{

return;

}

//1

if (EdgePoints[X + 1, Y] == 2)

{

EdgeMap[X + 1, Y] = 1;

VisitedMap[X + 1, Y] = 1;

Travers(X + 1, Y);

return;

}

//2

if (EdgePoints[X + 1, Y - 1] == 2)

{

EdgeMap[X + 1, Y - 1] = 1;

VisitedMap[X + 1, Y - 1] = 1;

Travers(X + 1, Y - 1);

return;

}

//3

if (EdgePoints[X, Y - 1] == 2)

{

EdgeMap[X, Y - 1] = 1;

VisitedMap[X, Y - 1] = 1;

Travers(X, Y - 1);

return;

}

//4

if (EdgePoints[X - 1, Y - 1] == 2)

{

EdgeMap[X - 1, Y - 1] = 1;

VisitedMap[X - 1, Y - 1] = 1;

Travers(X - 1, Y - 1);

return;

}

//5

if (EdgePoints[X - 1, Y] == 2)

{

EdgeMap[X - 1, Y] = 1;

VisitedMap[X - 1, Y] = 1;

Travers(X - 1, Y);

return;

}

//6

if (EdgePoints[X - 1, Y + 1] == 2)

{

EdgeMap[X - 1, Y + 1] = 1;

VisitedMap[X - 1, Y + 1] = 1;

Travers(X - 1, Y + 1);

return;

}

//7

if (EdgePoints[X, Y + 1] == 2)

{

EdgeMap[X, Y + 1] = 1;

VisitedMap[X, Y + 1] = 1;

Travers(X, Y + 1);

return;

}

//8

if (EdgePoints[X + 1, Y + 1] == 2)

{

EdgeMap[X + 1, Y + 1] = 1;

VisitedMap[X + 1, Y + 1] = 1;

Travers(X + 1, Y + 1);

return;

}

//VisitedMap[X, Y] = 1;

return;

}

//Canny Class Ends

}

}

**A3. Segmentación de Imagen**

//---------------conteo de pixeles--------------------------//

//-------------------------botella-------------------------//

//----------------tapa--------------------//

for (int i = 105; i <= 270; i++)

{

for (int j = 500; j <= 530; j++)

{

int a = b.GetPixel(i, j).ToArgb();

if (a.Equals(-1))

{

tapa = tapa + 1;

b.SetPixel(i, j, cambio1);

}

}

}

//----------------tapa--------------------//

for (int i = 110; i <= 270; i++)

{

for (int j = 625; j <= 950; j++)

{

int a = b.GetPixel(i, j).ToArgb();

if (a.Equals(-1))

{

blancos = blancos + 1;

b.SetPixel(i, j, cambio);

}

}

}

//-------------------------botella fin----------------------------------------//

//------------------------------botella 1--------------------------------------//

//----------------tapa1--------------------//

for (int i = 560; i <= 740; i++)

{

for (int j = 500; j <= 530; j++)

{

int a = b.GetPixel(i, j).ToArgb();

if (a.Equals(-1))

{

tapa1 = tapa1 + 1;

b.SetPixel(i, j, cambio1);

}

}

}

//----------------tapa1--------------------//

for (int i = 565; i <= 700; i++)

{

for (int j = 630; j <= 930; j++)

{

int a = b.GetPixel(i, j).ToArgb();

if (a.Equals(-1))

{

blancos1 = blancos1 + 1;

b.SetPixel(i, j, cambio);

}

}

}

//--------------------------------botella 1 fin-------------------------------//

//--------------------------botella 2-----------------------------------------//

//----------------tapa2--------------------//

for (int i = 1008; i <= 1150; i++)

{

for (int j = 485; j <= 530; j++)

{

int a = b.GetPixel(i, j).ToArgb();

if (a.Equals(-1))

{

tapa2 = tapa2 + 1;

b.SetPixel(i, j, cambio1);

}

}

}

//----------------tapa2--------------------//

for (int i = 1023; i <= 1135; i++)

{

for (int j = 635; j <= 925; j++)

{

int a = b.GetPixel(i, j).ToArgb();

if (a.Equals(-1))

{

blancos2 = blancos2 + 1;

b.SetPixel(i, j, cambio);

}

}

}

blancos4 = 0;

blancos5 = 0;

blancos6 = 0;

tapa4 = 0;

tapa5 = 0;

tapa6 = 0;

bot4 = 0;

bot5 = 0;

bot6 = 0;

//--------------------------botella 2fin-----------------------------------------//

//------------------------botella 4 ------------------------------------------//

//----------------tapa 4--------------------//

for (int i = 110; i <= 305; i++)

{

for (int j = 410; j <= 440; j++)

{

int d = c.GetPixel(i, j).ToArgb();

if (d.Equals(-1))

{

tapa4 = tapa4 + 1;

c.SetPixel(i, j, cambio1);

}

}

}

//----------------tapa 4--------------------//

for (int i = 130; i <= 300; i++)

{

for (int j = 520; j <= 840; j++)

{

int d = c.GetPixel(i, j).ToArgb();

if (d.Equals(-1))

{

blancos4 = blancos4 + 1;

c.SetPixel(i, j, cambio);

}

}

}

//--------------------------botella 4 fin--------------------------------------//

//----------------------------botella 5----------------------------------------//

//----------------tapa5--------------------//

for (int i = 540; i <= 715; i++)

{

for (int j = 410; j <= 445; j++)

{

int d = c.GetPixel(i, j).ToArgb();

if (d.Equals(-1))

{

tapa5 = tapa5 + 1;

c.SetPixel(i, j, cambio1);

}

}

}

//----------------tapa5--------------------//

for (int i = 520; i <= 707; i++)

{

for (int j = 515; j <= 840; j++)

{

int d = c.GetPixel(i, j).ToArgb();

if (d.Equals(-1))

{

blancos5 = blancos5 + 1;

c.SetPixel(i, j, cambio);

}

}

}

//---------------------------botella 5 fin-----------------------------------//

//---------------------------botella 6---------------------------------------//

//----------------tapa6--------------------//

for (int i = 985; i <= 1150; i++)

{

for (int j = 410; j <= 445; j++)

{

int d = c.GetPixel(i, j).ToArgb();

if (d.Equals(-1))

{

tapa6 = tapa6 + 1;

c.SetPixel(i, j, cambio1);

}

}

}

//----------------tapa6--------------------//

for (int i = 980; i <= 1130; i++)

{

for (int j = 530; j <= 810; j++)

{

int d = c.GetPixel(i, j).ToArgb();

if (d.Equals(-1))

{

blancos6 = blancos6 + 1;

c.SetPixel(i, j, cambio);

}

}

}

//---------------------------botella 6fin------------------------------------//

**A4. Extracción de Características**

//--------------------------------------botella-------------------------------//

//----------------tapa--------------------//

for (int i = 105; i <= 270; i++)

{

for (int j = 500; j <= 530; j++)

{

int a = b.GetPixel(i, j).ToArgb();

if (a.Equals(-1))

{

tapa = tapa + 1;

b.SetPixel(i, j, cambio1);

}

}

}

if (tapasup < tapa)

tapasup = tapa;

if (tapa != 0)

if (tapainf > tapa)

tapainf = tapa;

//----------------tapa--------------------//

for (int i = 110; i <= 270; i++)

{

for (int j = 625; j <= 950; j++)

{

int a = b.GetPixel(i, j).ToArgb();

if (a.Equals(-1))

{

blancos = blancos + 1;

b.SetPixel(i, j, cambio);

}

}

}

if (blanossup < blancos)

blanossup = blancos;

if (blancos != 0)

if (blancosinf > blancos)

blancosinf = blancos;

label1.Text = tapasup.ToString();

label2.Text = tapainf.ToString();

label3.Text = blanossup.ToString();

label4.Text = blancosinf.ToString();

//-------------------------------botella fin---------------------------------//

//-------------------------------botella 1-----------------------------------//

//----------------tapa1--------------------//

for (int i = 560; i <= 740; i++)

{

for (int j = 500; j <= 530; j++)

{

int a = b.GetPixel(i, j).ToArgb();

if (a.Equals(-1))

{

tapa1 = tapa1 + 1;

b.SetPixel(i, j, cambio1);

}

}

}

if (tapasup1 < tapa1)

tapasup1 = tapa1;

if (tapa1 != 0)

if (tapainf1 > tapa1)

tapainf1 = tapa1;

//----------------tapa1--------------------//

for (int i = 565; i <= 700; i++)

{

for (int j = 630; j <= 930; j++)

{

int a = b.GetPixel(i, j).ToArgb();

if (a.Equals(-1))

{

blancos1 = blancos1 + 1;

b.SetPixel(i, j, cambio);

}

}

}

if (blanossup1 < blancos1)

blanossup1 = blancos1;

if (blancos1 != 0)

if (blancosinf1 > blancos1)

blancosinf1 = blancos1;

label5.Text = tapasup1.ToString();

label6.Text = tapainf1.ToString();

label7.Text = blanossup1.ToString();

label8.Text = blancosinf1.ToString();

//-----------------botella 1 fin-------------------------------------------//

//------------------------------------botella 2-------------------------------//

//----------------tapa2--------------------//

for (int i = 1008; i <= 1150; i++)

{

for (int j = 485; j <= 530; j++)

{

int a = b.GetPixel(i, j).ToArgb();

if (a.Equals(-1))

{

tapa2 = tapa2 + 1;

b.SetPixel(i, j, cambio1);

}

}

}

if (tapasup2 < tapa2)

tapasup2 = tapa2;

if (tapa2 != 0)

if (tapainf2 > tapa2)

tapainf2 = tapa2;

//----------------tapa2--------------------//

for (int i = 1023; i <= 1165; i++)

{

for (int j = 635; j <= 925; j++)

{

int a = b.GetPixel(i, j).ToArgb();

if (a.Equals(-1))

{

blancos2 = blancos2 + 1;

b.SetPixel(i, j, cambio);

}

}

}

if (blanossup2 < blancos2)

blanossup2 = blancos2;

if (blancos2 != 0)

if (blancosinf2 > blancos2)

blancosinf2 = blancos2;

label9.Text = tapasup2.ToString();

label10.Text = tapainf2.ToString();

label11.Text = blanossup2.ToString();

label12.Text = blancosinf2.ToString();

//------------------------------botella 2fin----------------------------------//

//----------------------botella4 ---------------------------------------------//

//----------------tapa4--------------------//

for (int i = 110; i <= 305; i++)

{

for (int j = 410; j <= 440; j++)

{

int d = c.GetPixel(i, j).ToArgb();

if (d.Equals(-1))

{

tapa4 = tapa4 + 1;

c.SetPixel(i, j, cambio1);

}

}

}

if (tapasup4 < tapa4)

tapasup4 = tapa4;

if (tapa4 != 0)

if (tapainf4 > tapa4)

tapainf4 = tapa4;

//----------------tapa4--------------------//

for (int i = 130; i <= 300; i++)

{

for (int j = 520; j <= 840; j++)

{

int d = c.GetPixel(i, j).ToArgb();

if (d.Equals(-1))

{

blancos4 = blancos4 + 1;

c.SetPixel(i, j, cambio);

}

}

}

if (blanossup4 < blancos4)

blanossup4 = blancos4;

if (blancos4 != 0)

if (blancosinf4 > blancos4)

blancosinf4 = blancos4;

label13.Text = tapasup4.ToString();

label14.Text = tapainf4.ToString();

label15.Text = blanossup4.ToString();

label16.Text = blancosinf4.ToString();

//-------------------------------botella4 fin---------------------------------//

//--------------------------botella5-------------------------------------------//

//----------------tapa5--------------------//

for (int i = 540; i <= 715; i++)

{

for (int j = 410; j <= 445; j++)

{

int d = c.GetPixel(i, j).ToArgb();

if (d.Equals(-1))

{

tapa5 = tapa5 + 1;

c.SetPixel(i, j, cambio1);

}

}

}

if (tapasup5 < tapa5)

tapasup5 = tapa5;

if (tapa5 != 0)

if (tapainf5 > tapa5)

tapainf5 = tapa5;

//----------------tapa5--------------------//

for (int i = 520; i <= 707; i++)

{

for (int j = 515; j <= 840; j++)

{

int d = c.GetPixel(i, j).ToArgb();

// b.SetPixel(i, j, cambio);

if (d.Equals(-1))

{

blancos5 = blancos5 + 1;

c.SetPixel(i, j, cambio);

}

}

}

if (blanossup5 < blancos5)

blanossup5 = blancos5;

if (blancos5 != 0)

if (blancosinf5 > blancos5)

blancosinf5 = blancos5;

label17.Text = tapasup5.ToString();

label18.Text = tapainf5.ToString();

label19.Text = blanossup5.ToString();

label20.Text = blancosinf5.ToString();

//--------------------------botella5 fin--------------------------------------//

//------------------------botella6-------------------------------------//

//----------------tapa6--------------------//

for (int i = 985; i <= 1150; i++)

{

for (int j = 410; j <= 445; j++)

{

int d = c.GetPixel(i, j).ToArgb();

if (d.Equals(-1))

{

tapa6 = tapa6 + 1;

c.SetPixel(i, j, cambio1);

}

}

}

if (tapasup6 < tapa6)

tapasup6 = tapa6;

if (tapa6 != 0)

if (tapainf6 > tapa6)

tapainf6 = tapa6;

//----------------tapa6--------------------//

for (int i = 980; i <= 1130; i++)

{

for (int j = 530; j <= 810; j++)

{

int d = c.GetPixel(i, j).ToArgb();

if (d.Equals(-1))

{

blancos6 = blancos6 + 1;

c.SetPixel(i, j, cambio);

}

}

}

if (blanossup6 < blancos6)

blanossup6 = blancos6;

if (blancos6 != 0)

if (blancosinf6 > blancos6)

blancosinf6 = blancos6;

label21.Text = tapasup6.ToString();

label22.Text = tapainf6.ToString();

label23.Text = blanossup6.ToString();

label24.Text = blancosinf6.ToString();

//-----------------------------botella6 fin---------------------------------//

**A5. Conversión de variables del PLC**

using System;

using System.Globalization;

namespace S7.Net

{

/// <summary>

/// Conversion methods to convert from Siemens numeric format to C# and back

/// </summary>

public static class Conversion

{

/// <summary>

/// Converts a binary string to Int32 value

/// </summary>

/// <param name="txt"></param>

/// <returns></returns>

public static int BinStringToInt32(this string txt)

{

int cnt = 0;

int ret = 0;

for (cnt = txt.Length - 1; cnt >= 0; cnt += -1)

{

if (int.Parse(txt.Substring(cnt, 1)) == 1)

{

ret += (int)(Math.Pow(2, (txt.Length - 1 - cnt)));

}

}

return ret;

}

/// <summary>

/// Converts a binary string to a byte. Can return null.

/// </summary>

/// <param name="txt"></param>

/// <returns></returns>

public static byte? BinStringToByte(this string txt)

{

int cnt = 0;

int ret = 0;

if (txt.Length == 8)

{

for (cnt = 7; cnt >= 0; cnt += -1)

{

if (int.Parse(txt.Substring(cnt, 1)) == 1)

{

ret += (int)(Math.Pow(2, (txt.Length - 1 - cnt)));

}

}

return (byte)ret;

}

return null;

}

/// <summary>

/// Converts the value to a binary string

/// </summary>

/// <param name="value"></param>

/// <returns></returns>

public static string ValToBinString(this object value)

{

int cnt = 0;

int cnt2 = 0;

int x = 0;

string txt = "";

long longValue = 0;

try

{

if (value.GetType().Name.IndexOf("[]") < 0)

{

// ist nur ein Wert

switch (value.GetType().Name)

{

case "Byte":

x = 7;

longValue = (long)((byte)value);

break;

case "Int16":

x = 15;

longValue = (long)((Int16)value);

break;

case "Int32":

x = 31;

longValue = (long)((Int32)value);

break;

case "Int64":

x = 63;

longValue = (long)((Int64)value);

break;

default:

throw new Exception();

}

for (cnt = x; cnt >= 0; cnt += -1)

{

if (((Int64)longValue & (Int64)Math.Pow(2, cnt)) > 0)

txt += "1";

else

txt += "0";

}

}

else

{

// ist ein Array

switch (value.GetType().Name)

{

case "Byte[]":

x = 7;

byte[] ByteArr = (byte[])value;

for (cnt2 = 0; cnt2 <= ByteArr.Length - 1; cnt2++)

{

for (cnt = x; cnt >= 0; cnt += -1)

if ((ByteArr[cnt2] & (byte)Math.Pow(2, cnt)) > 0) txt += "1"; else txt += "0";

}

break;

case "Int16[]":

x = 15;

Int16[] Int16Arr = (Int16[])value;

for (cnt2 = 0; cnt2 <= Int16Arr.Length - 1; cnt2++)

{

for (cnt = x; cnt >= 0; cnt += -1)

if ((Int16Arr[cnt2] & (byte)Math.Pow(2, cnt)) > 0) txt += "1"; else txt += "0";

}

break;

case "Int32[]":

x = 31;

Int32[] Int32Arr = (Int32[])value;

for (cnt2 = 0; cnt2 <= Int32Arr.Length - 1; cnt2++)

{

for (cnt = x; cnt >= 0; cnt += -1)

if ((Int32Arr[cnt2] & (byte)Math.Pow(2, cnt)) > 0) txt += "1"; else txt += "0";

}

break;

case "Int64[]":

x = 63;

byte[] Int64Arr = (byte[])value;

for (cnt2 = 0; cnt2 <= Int64Arr.Length - 1; cnt2++)

{

for (cnt = x; cnt >= 0; cnt += -1)

if ((Int64Arr[cnt2] & (byte)Math.Pow(2, cnt)) > 0) txt += "1"; else txt += "0";

}

break;

default:

throw new Exception();

}

}

return txt;

}

catch

{

return "";

}

}

/// <summary>

/// Helper to get a bit value given a byte and the bit index.

/// Example: DB1.DBX0.5 -> var bytes = ReadBytes(DB1.DBW0); bool bit = bytes[0].SelectBit(5);

/// </summary>

/// <param name="data"></param>

/// <param name="bitPosition"></param>

/// <returns></returns>

public static bool SelectBit(this byte data, int bitPosition)

{

int mask = 1 << bitPosition;

int result = data & mask;

return (result != 0);

}

/// <summary>

/// Converts from ushort value to short value; it's used to retrieve negative values from words

/// </summary>

/// <param name="input"></param>

/// <returns></returns>

public static short ConvertToShort(this ushort input)

{

short output;

output = short.Parse(input.ToString("X"), NumberStyles.HexNumber);

return output;

}

/// <summary>

/// Converts from short value to ushort value; it's used to pass negative values to DWs

/// </summary>

/// <param name="input"></param>

/// <returns></returns>

public static ushort ConvertToUshort(this short input)

{

ushort output;

output = ushort.Parse(input.ToString("X"), NumberStyles.HexNumber);

return output;

}

/// <summary>

/// Converts from UInt32 value to Int32 value; it's used to retrieve negative values from DBDs

/// </summary>

/// <param name="input"></param>

/// <returns></returns>

public static Int32 ConvertToInt(this uint input)

{

int output;

output = int.Parse(input.ToString("X"), NumberStyles.HexNumber);

return output;

}

/// <summary>

/// Converts from Int32 value to UInt32 value; it's used to pass negative values to DBDs

/// </summary>

/// <param name="input"></param>

/// <returns></returns>

public static UInt32 ConvertToUInt(this int input)

{

uint output;

output = uint.Parse(input.ToString("X"), NumberStyles.HexNumber);

return output;

}

/// <summary>

/// Converts from double to DWord (DBD)

/// </summary>

/// <param name="input"></param>

/// <returns></returns>

public static UInt32 ConvertToUInt(this double input)

{

uint output;

output = S7.Net.Types.DWord.FromByteArray(S7.Net.Types.Double.ToByteArray(input));

return output;

}

/// <summary>

/// Converts from DWord (DBD) to double

/// </summary>

/// <param name="input"></param>

/// <returns></returns>

public static double ConvertToDouble(this uint input)

{

double output;

output = S7.Net.Types.Double.FromByteArray(S7.Net.Types.DWord.ToByteArray(input));

return output;

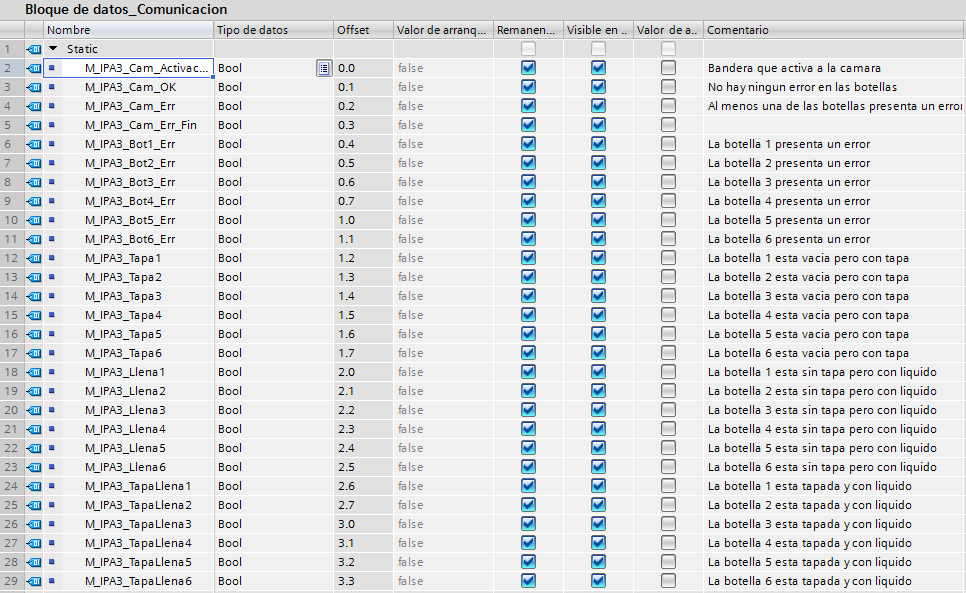
}

}

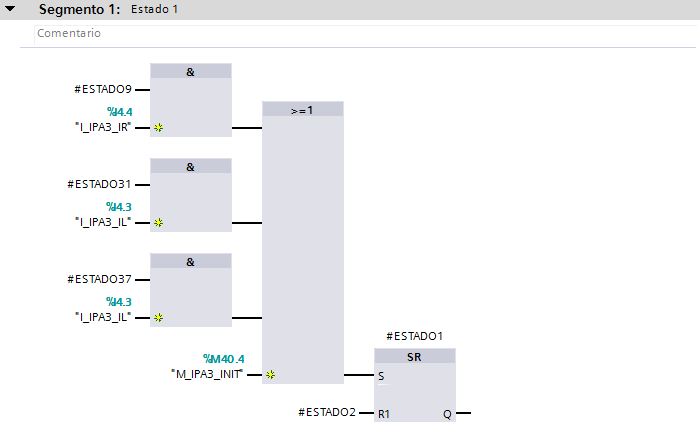
}

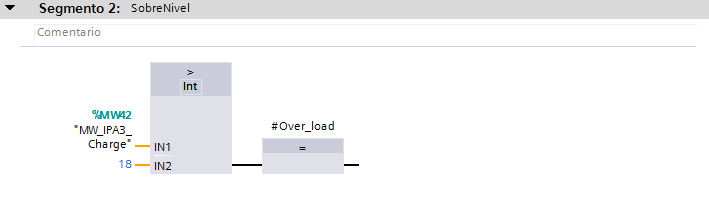
**Programación en TIA Portal**

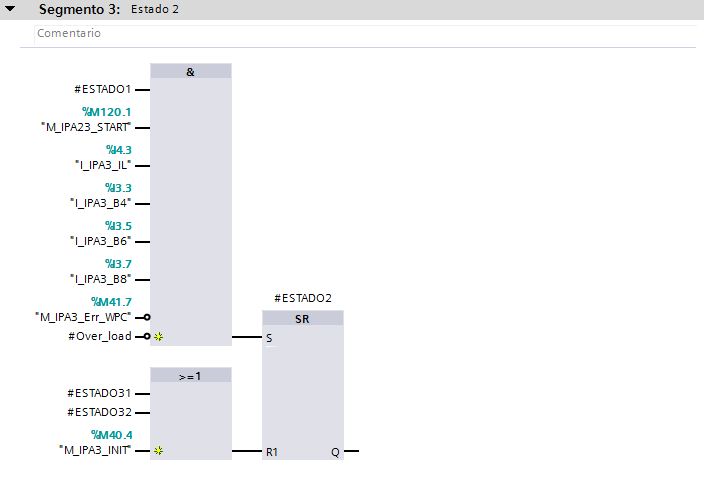
**A6. Bloque de comunicación**

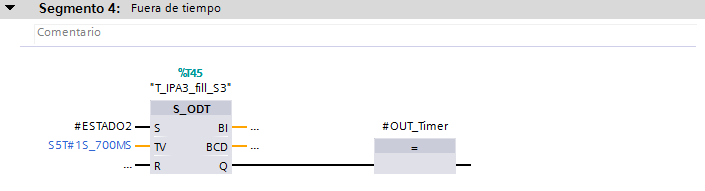
****

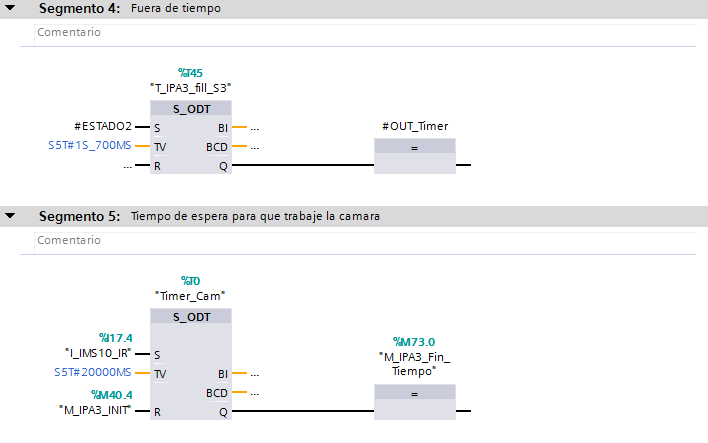
**A7. Bloque de transiciones**

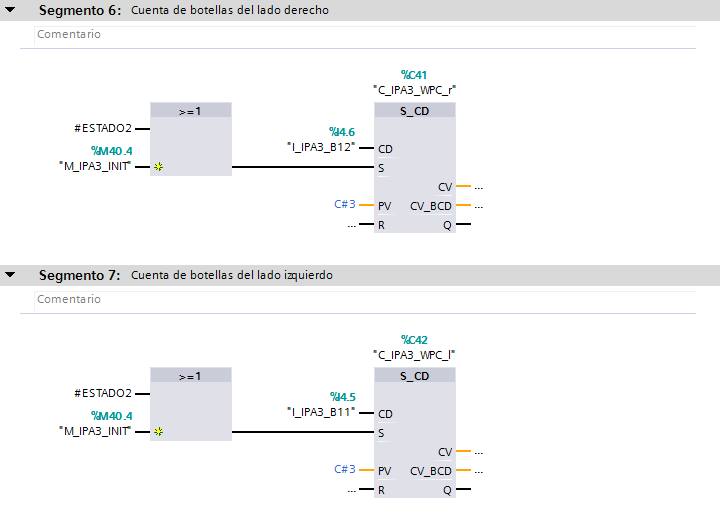
****

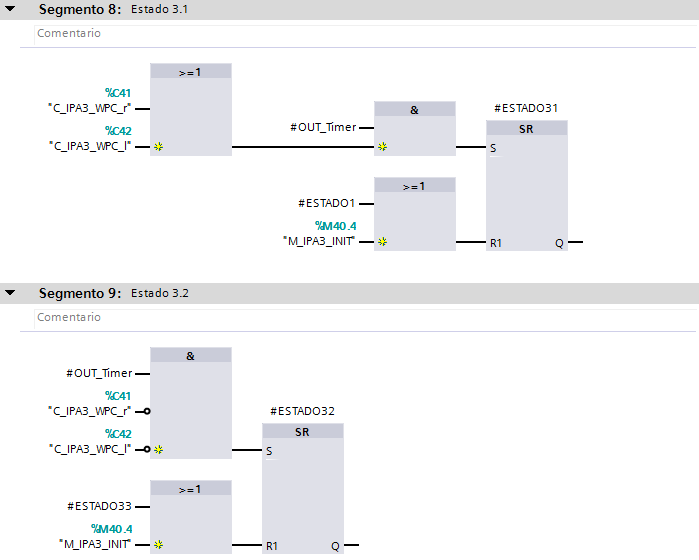
****

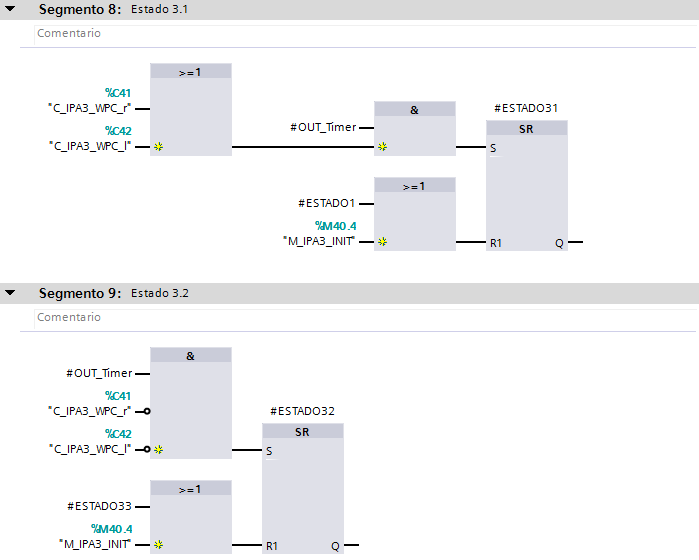
****

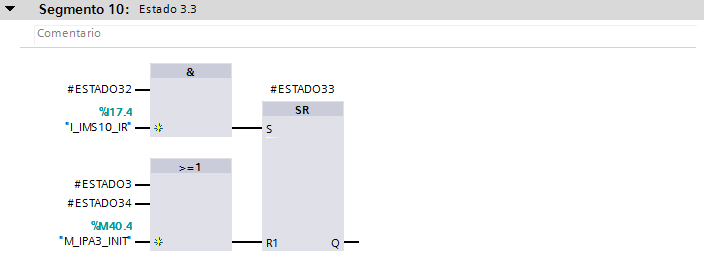
****

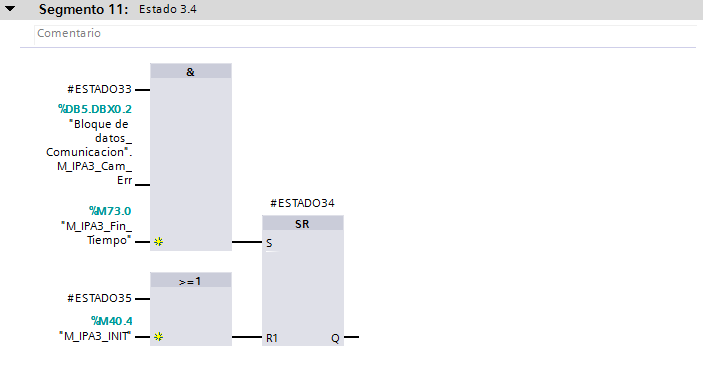
****

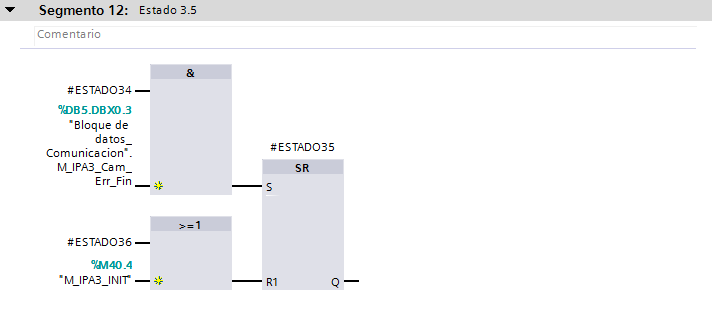
****

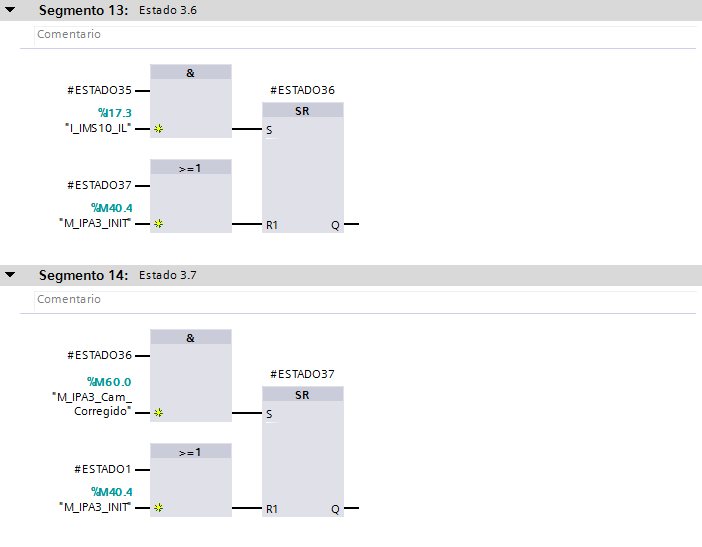
****

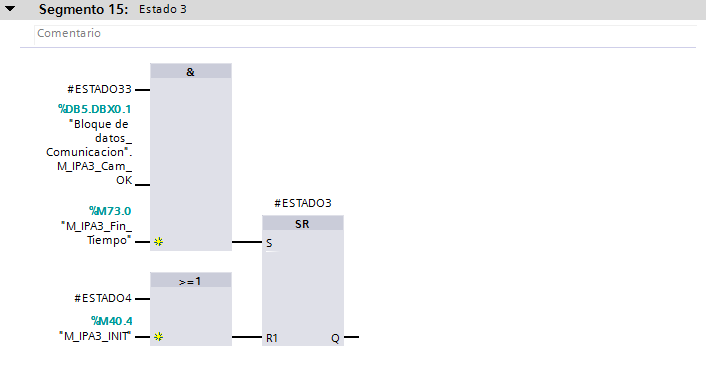
****

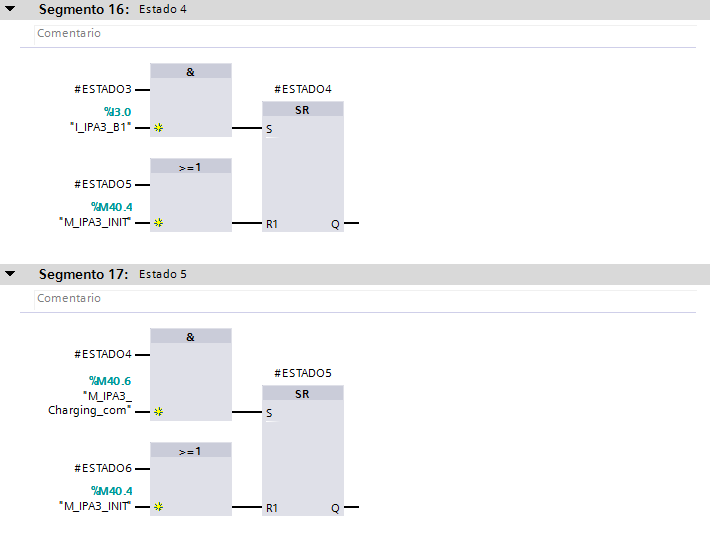
****

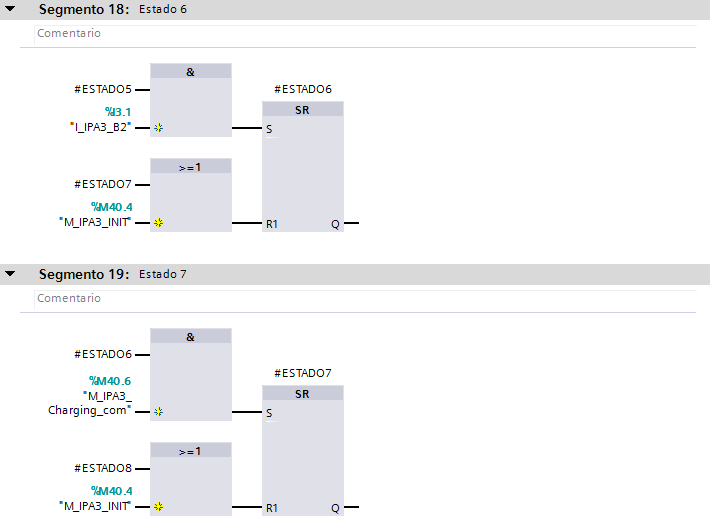
****

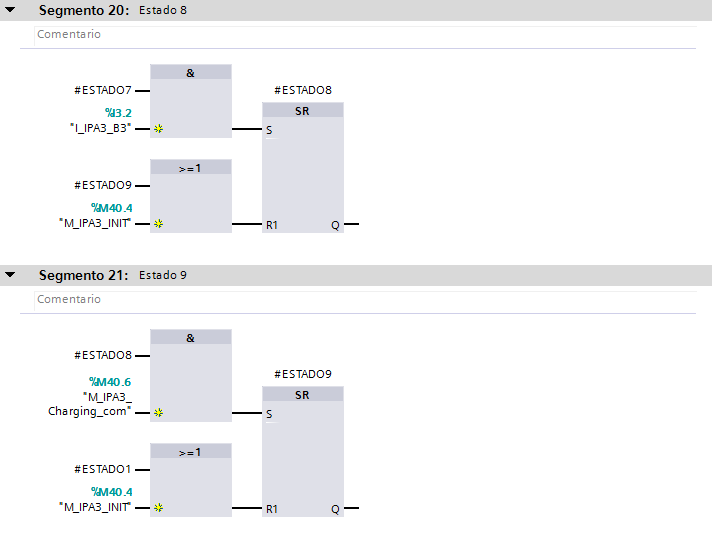
****

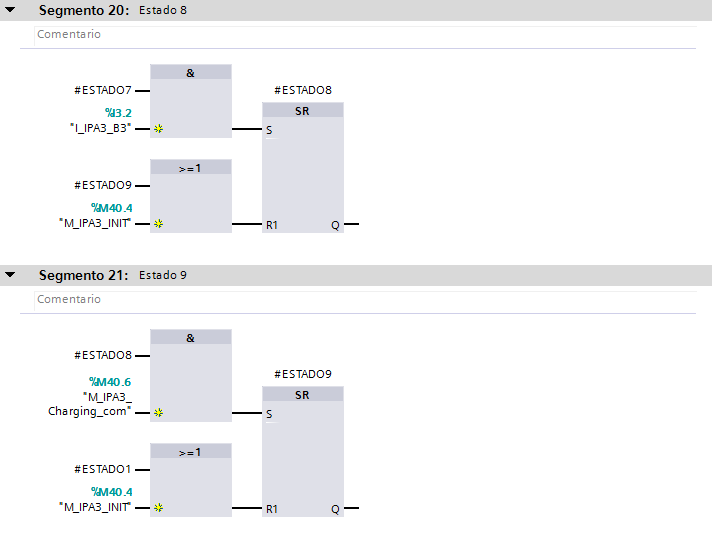
****

****

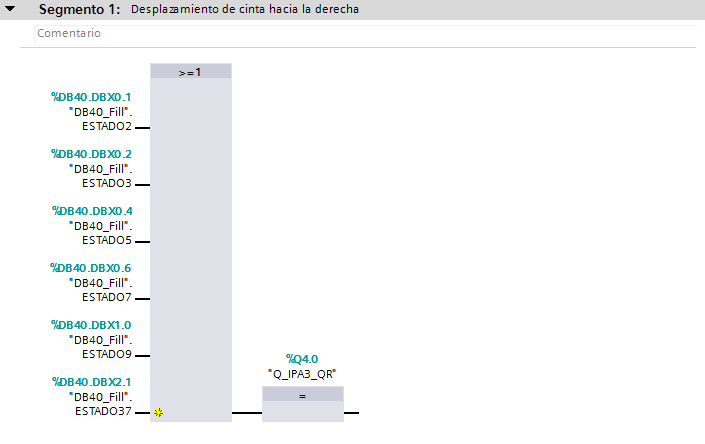
****

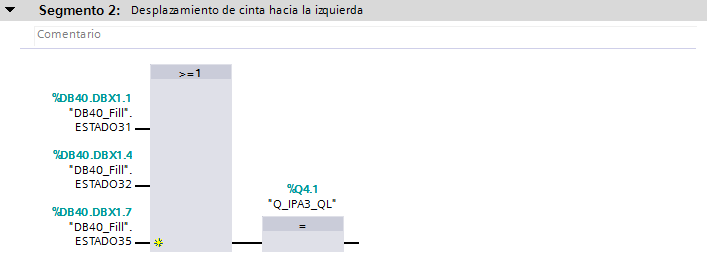
****

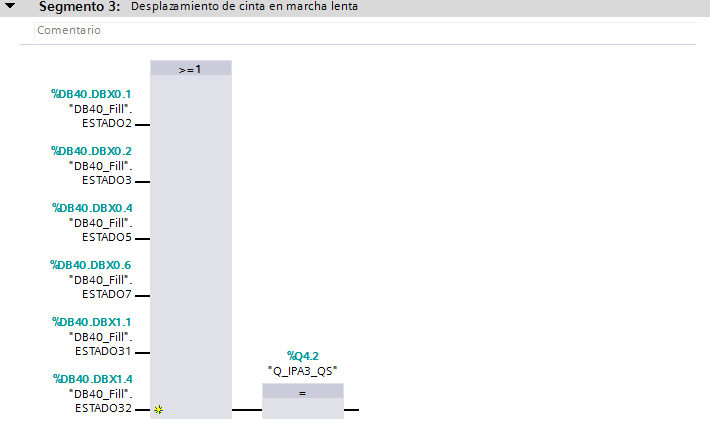
****

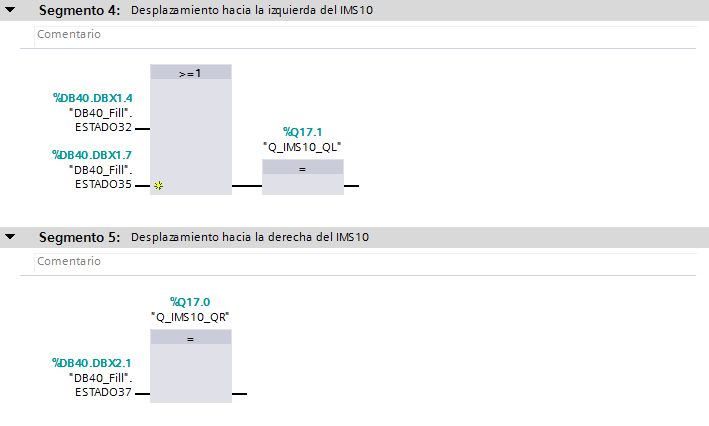
****

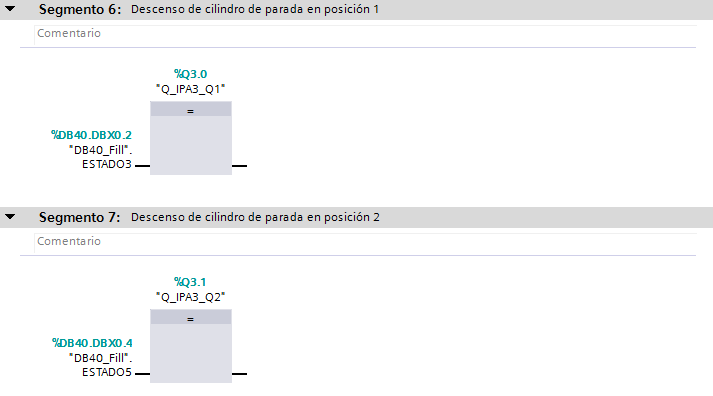
**A8. Bloque de acciones**

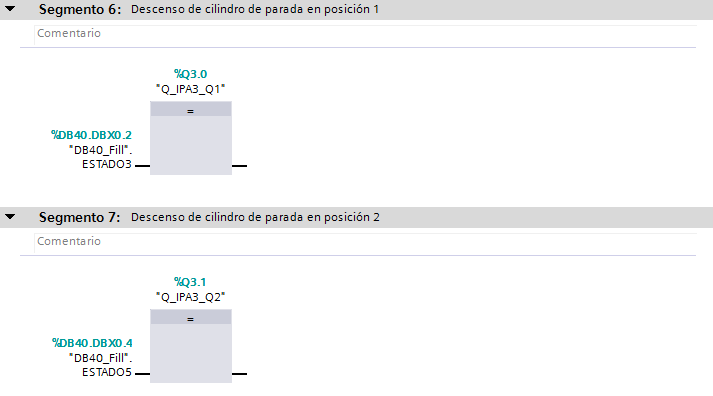
****

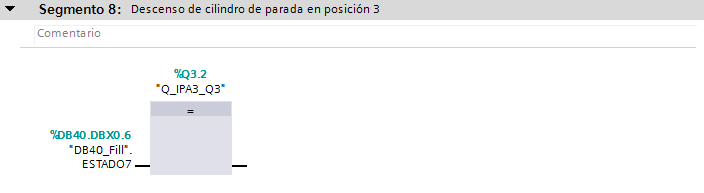
****

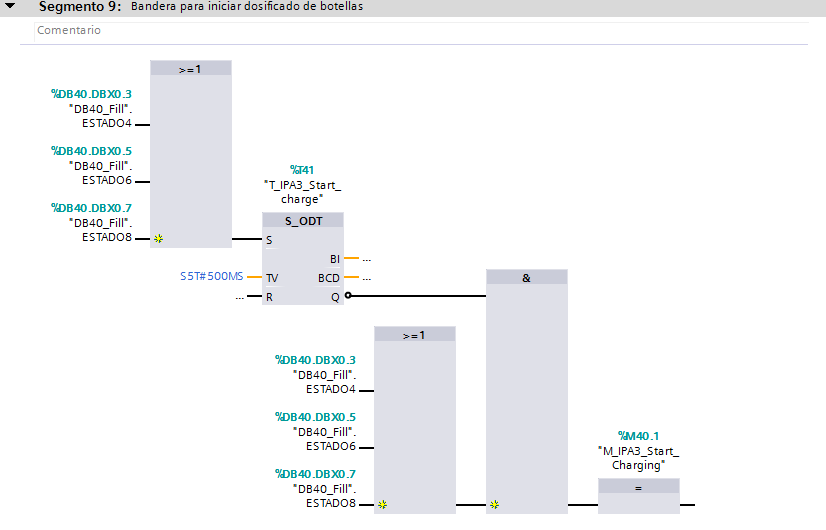
****

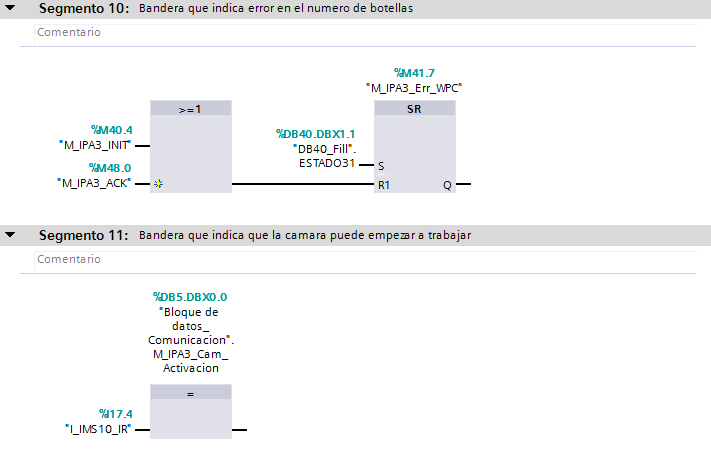
****

****

****

****

****

****