

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 1
YAPILAN İŞ	Kalibrasyon Teorisi Araştırma (1/2)	TARİH : 24.06.19

1 MONO KAMERA KALİBRASYONU

1.1 KALİBRASYON TEORİSİ

Kameralar dünyamızdaki 3 boyutlu noktaları 2 boyutlu noktalar olarak okur ve yansıtırlar. Bunun matematiksel olarak yaklaşımı 3 boyutlu uzaydan 2 boyutlu uzaya projeksiyondur. Bu projeksiyonu homojen koordinat sistemini kullanarak matrisler ile aşağıdaki gibi ifade edebiliriz.

$$C=AW$$

C : homojen kamera koordinatları

A : projeksiyon matrisi

W :homojen dünya koordinatları

Buradaki üç matristen ikisini biliyorsak üçüncü matrisi üretebiliriz. Zaten Kalibrasyon işlemi de dünya koordinatları ve kamera koordinatları bilindiği takdirde projeksiyon matrisinin üretilmesidir.

$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$x = \frac{C_1}{C_4}$$

$$y = \frac{C_2}{C_4}$$

x, y : kartezyen kamera koordinatları

Yukarıdaki matrislerin sonucunda aşağıdaki denklem sistemi elde edilir.

$$\begin{aligned} C_1 &= a_{11}X + a_{12}Y + a_{13}Z + a_{14} = C_4x \\ C_2 &= a_{21}X + a_{22}Y + a_{23}Z + a_{24} = C_4y \\ C_4 &= a_{41}X + a_{42}Y + a_{43}Z + a_{44} \end{aligned}$$

C_4 'ü ilk iki denklemde yerine yazıp düzenlersek aşağıdaki denklem sistemi elde edilir :

$$\begin{aligned} a_{11}X + a_{12}Y + a_{13}Z + a_{14} - a_{41}Xx - a_{42}Yx - a_{43}Zx - a_{44}x &= 0 \\ a_{21}X + a_{22}Y + a_{23}Z + a_{24} - a_{41}Xy - a_{42}Yy - a_{43}Zy - a_{44}y &= 0 \end{aligned}$$

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 2
YAPILAN İŞ	Kalibrasyon Teorisi Araştırma (2/2)	TARİH : 24.06.19

1 nokta için 2 denklem elde etmiş olduk. Projeksiyon matrisinde 12 tane değişken olduğundan en az 6 nokta kullanmamız lazım.

n tane nokta için ise bu denklem sistemini aşağıdaki gibi yazabiliriz.

$$\begin{aligned}
a_{11}X_1 + a_{12}Y_1 + a_{13}Z_1 + a_{14} - a_{41}X_1x_1 - a_{42}Y_1x_1 - a_{43}Z_1x_1 - a_{44}x_1 &= 0 \\
a_{21}X_1 + a_{22}Y_1 + a_{23}Z_1 + a_{24} - a_{41}X_1y_1 - a_{42}Y_1y_1 - a_{43}Z_1y_1 - a_{44}y_1 &= 0 \\
a_{11}X_2 + a_{12}Y_2 + a_{13}Z_2 + a_{14} - a_{41}X_2x_2 - a_{42}Y_2x_2 - a_{43}Z_2x_2 - a_{44}x_2 &= 0 \\
a_{21}X_2 + a_{22}Y_2 + a_{23}Z_2 + a_{24} - a_{41}X_2y_2 - a_{42}Y_2y_2 - a_{43}Z_2y_2 - a_{44}y_2 &= 0 \\
a_{11}X_3 + a_{12}Y_3 + a_{13}Z_3 + a_{14} - a_{41}X_3x_3 - a_{42}Y_3x_3 - a_{43}Z_3x_3 - a_{44}x_3 &= 0 \\
a_{21}X_3 + a_{22}Y_3 + a_{23}Z_3 + a_{24} - a_{41}X_3y_3 - a_{42}Y_3y_3 - a_{43}Z_3y_3 - a_{44}y_3 &= 0
\end{aligned}$$

:

$$\begin{aligned}
a_{11}X_n + a_{12}Y_n + a_{13}Z_n + a_{14} - a_{41}X_nx_n - a_{42}Y_nx_n - a_{43}Z_nx_n - a_{44}x_n &= 0 \\
a_{21}X_n + a_{22}Y_n + a_{23}Z_n + a_{24} - a_{41}X_ny_n - a_{42}Y_ny_n - a_{43}Z_ny_n - a_{44}y_n &= 0
\end{aligned}$$

Bu homojen bir denklem sistemidir ve birden fazla çözümü vardır. Bunlardan birinde a_{44} 'ün değeri 1'e eşit olacaktır. Bu çözümü elde etmek için a_{44} 'ün yerine 1 yazıp çözüyoruz.

$$\begin{aligned}
a_{11}X_1 + a_{12}Y_1 + a_{13}Z_1 + a_{14} - a_{41}X_1x_1 - a_{42}Y_1x_1 - a_{43}Z_1x_1 &= x_1 \\
a_{21}X_1 + a_{22}Y_1 + a_{23}Z_1 + a_{24} - a_{41}X_1y_1 - a_{42}Y_1y_1 - a_{43}Z_1y_1 &= y_1 \\
a_{11}X_2 + a_{12}Y_2 + a_{13}Z_2 + a_{14} - a_{41}X_2x_2 - a_{42}Y_2x_2 - a_{43}Z_2x_2 &= x_2 \\
a_{21}X_2 + a_{22}Y_2 + a_{23}Z_2 + a_{24} - a_{41}X_2y_2 - a_{42}Y_2y_2 - a_{43}Z_2y_2 &= y_2 \\
a_{11}X_3 + a_{12}Y_3 + a_{13}Z_3 + a_{14} - a_{41}X_3x_3 - a_{42}Y_3x_3 - a_{43}Z_3x_3 &= x_3 \\
a_{21}X_3 + a_{22}Y_3 + a_{23}Z_3 + a_{24} - a_{41}X_3y_3 - a_{42}Y_3y_3 - a_{43}Z_3y_3 &= y_3
\end{aligned}$$

:

$$\begin{aligned}
a_{11}X_n + a_{12}Y_n + a_{13}Z_n + a_{14} - a_{41}X_nx_n - a_{42}Y_nx_n - a_{43}Z_nx_n &= x_n \\
a_{21}X_n + a_{22}Y_n + a_{23}Z_n + a_{24} - a_{41}X_ny_n - a_{42}Y_ny_n - a_{43}Z_ny_n &= y_n
\end{aligned}$$

Yukarıdaki denklem sistemini matrise aktaralım.

$$\begin{bmatrix}
X_1Y_1Z_110000 - X_1x_1 - Y_1x_1 - Z_1x_1 \\
0000X_1Y_1Z_11 - X_1y_1 - Y_1y_1 - Z_1y_1 \\
X_2Y_2Z_210000 - X_2x_2 - Y_2x_2 - Z_2x_2 \\
0000X_2Y_2Z_21 - X_2y_2 - Y_2y_2 - Z_2y_2 \\
X_3Y_3Z_310000 - X_3x_3 - Y_3x_3 - Z_3x_3 \\
0000X_3Y_3Z_31 - X_3y_3 - Y_3y_3 - Z_3y_3 \\
\vdots \\
X_nY_nZ_n10000 - X_nx_n - Y_nx_n - Z_nx_n \\
0000X_nY_nZ_n1 - X_ny_n - Y_ny_n - Z_ny_n
\end{bmatrix}
\begin{bmatrix}
a_{11} \\
a_{12} \\
a_{13} \\
a_{14} \\
a_{21} \\
a_{22} \\
a_{23} \\
a_{24} \\
a_{41} \\
a_{42} \\
a_{43}
\end{bmatrix}
=
\begin{bmatrix}
x_1 \\
y_1 \\
x_2 \\
y_2 \\
x_3 \\
y_3 \\
\vdots \\
x_n \\
y_n
\end{bmatrix}$$

$$DQ = R$$

Bu matris çarpımında D ve R biliniyor. Buradan Q vektörünü bulabiliriz. **Singular Value Decomposition** yöntemi ile Q vektörünü buluyoruz. Artık A matrisimi doldurmak için tüm bilinmeyenleri elde ettik. Projeksiyon matrisini (A) oluşturduğumuzda artık kameramız kalibre edilmiştir.

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 3
YAPILAN İŞ	Projeksiyon Matrisi ile 2 Boyutlu İmar (Reconstruct) Araştırma	TARİH : 25.06.19

1.2 PROJEKSİYON MATRİSİ İLE 2 BOYUTLU İMAR (RECONSTRUCT)

Kameralar 3 boyutlu bir noktayı projeksiyon matrisi ile çarparak 2 boyutlu bir nokta elde eder.

$$\begin{bmatrix} a \\ b \\ w \end{bmatrix} = P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Bir önceki aşamada **SVD** yöntemi ile **Q** vektörünü bulduk. Şimdi ise 2 boyutlu dünya koordinatlarını hesaplayabiliriz. Resim koordinatlarını ve projeksiyon matrisini biliyorsak bu işlemi kolaylıkla yapabiliriz.

$$\begin{aligned} a &= Xp_{11} + Yp_{12} + Zp_{13} + p_{14} \\ b &= Xp_{21} + Yp_{22} + Zp_{23} + p_{24} \\ w &= Xp_{31} + Yp_{32} + Zp_{33} + p_{34} \\ a &= xw \\ b &= yw \end{aligned}$$

a ve **b**, **w** cinsinden yazılarak aşağıdaki denklem sistemi elde edilir.

$$\begin{aligned} xXp_{31} + xYp_{32} + xZp_{33} + xp_{34} &= Xp_{11} + Yp_{12} + Zp_{13} + p_{14} \\ yXp_{31} + yYp_{32} + yZp_{33} + yp_{34} &= Xp_{21} + Yp_{22} + Zp_{23} + p_{24} \end{aligned}$$

Elde ettiğimiz bu denklem sisteminde **X**, **Y**, **Z** 'li terimleri eşitliğin solunda, diğer terimleri ise eşitliğin sağında toplarsak aşağıdaki denklem sistemini elde ederiz.

$$\begin{aligned} X(xp_{31} - p_{11}) + Y(xp_{32} - p_{12}) + Z(xp_{33} - p_{13}) &= p_{14} - xp_{34} \\ X(yp_{31} - p_{21}) + Y(yp_{32} - p_{22}) + Z(yp_{33} - p_{23}) &= p_{24} - yp_{34} \end{aligned}$$

Yukarıdaki denklem sistemini matrise aktaracak olursak:

$$\begin{bmatrix} xp_{31} - p_{11} & xp_{32} - p_{12} & xp_{33} - p_{13} \\ yp_{31} - p_{21} & yp_{32} - p_{22} & yp_{33} - p_{23} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} p_{14} - xp_{34} \\ p_{24} - yp_{34} \end{bmatrix}$$

Yukarıdaki matriste **SVD** yöntemi ile **X** ve **Y** değerleri hesaplanır. Hesaplanan değerlerin sonucu bize noktanın gerçek dünyadaki iki boyuttaki konunu verir.

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 4
YAPILAN İŞ	Boş Form Oluşturma ve Uygulamaya Giriş	TARİH : 26.06.19

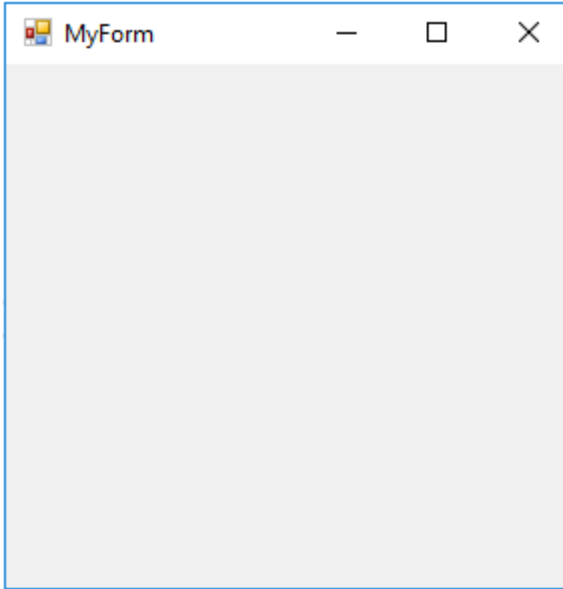
1.3 UYGULAMAYA GİRİŞ

Tek kamera ile 2 boyutlu kalibrasyon uygulaması geliştireceğiz. Uygulamamız Visual C++ uygulaması olacaktır. Uygulamada bir adet fotoğraf yükleyip, fare imleci ile seçeceğimiz kalibrasyon noktalarını belirleyip gerekli işlemlere(Kalibrasyon Teorisi kısmında anlatılan teorik ve matematiksel işlemler) tabi tutup görüntü üzerinde mesafe ve alan ölçümü yapacağız.

Aşağıdaki link üzerinden gerekli adımları takip ederek bir masaüstü uygulaması oluşturabilirsiniz.

<https://social.msdn.microsoft.com/Forums/vstudio/en-US/e6fbde42-d872-4ab3-8000-41ab22a4a584/visual-studio-2017-windows-forms?forum=winformsdesigner>

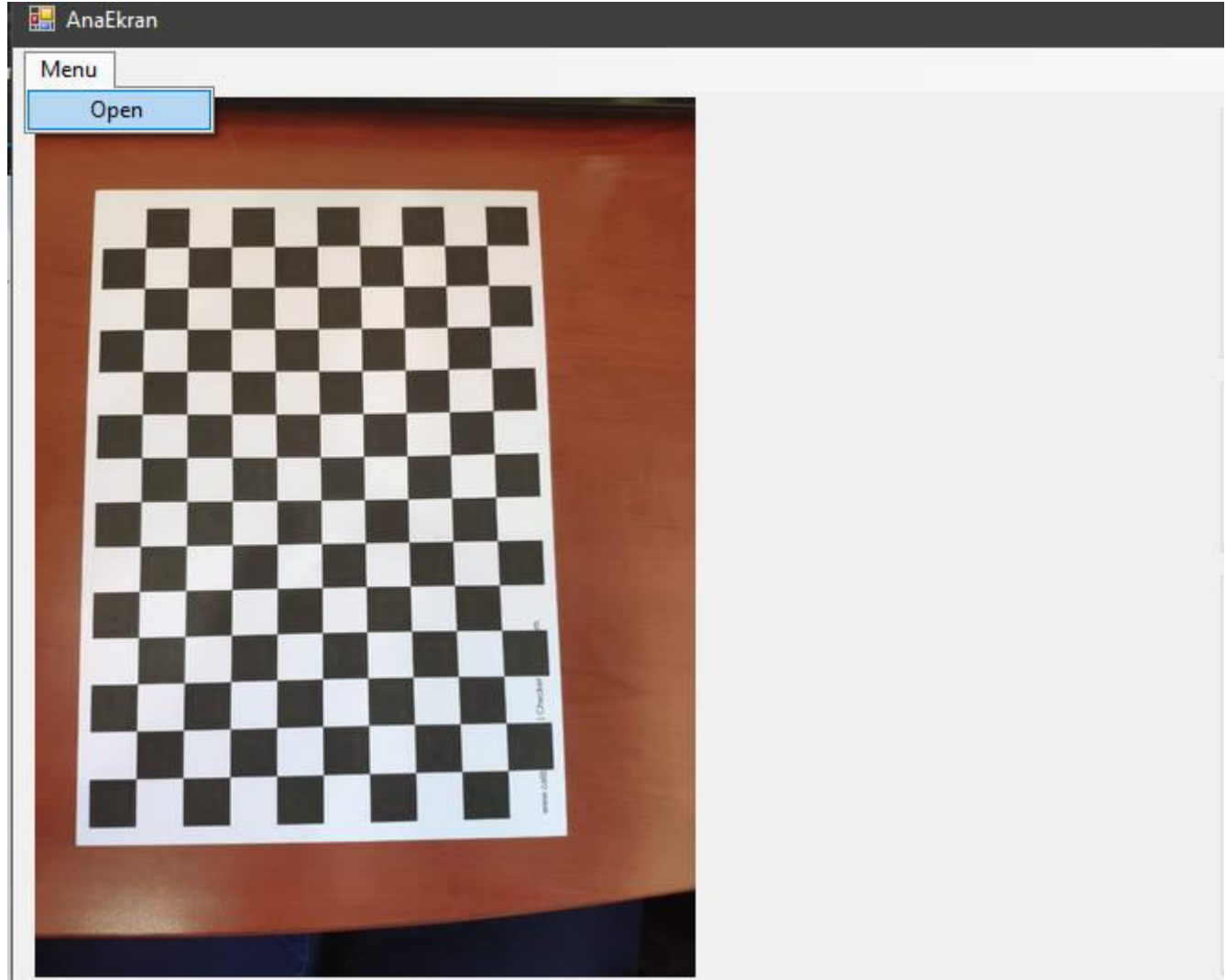
Adımların sonucunda aşağıdaki gibi bir form elde edeceğiz.



İkinci adımda oluşturulan boş form üzerine PictureBox ekleyip yüklenen görüntüyü orada göstereceğiz.

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 5
YAPILAN İŞ	PictureBox'ta İmge Gösterme	TARİH : 27.06.19

1.4 PICTUREBOX'TA İMGE GÖSTERME



Öncelikle formun header dosyasında **design** kısmına geliyoruz (Form dosyasına sağ tıklanıp **ViewDesign** diyebiliriz). **Toolbox**'dan **open file dialog** ve **menu strip** ekleyip yukarıdaki gibi tasarım elde ediyoruz. Temiz ve okunaklı kod yazma açısından fonksiyonlarımızı yazacağımız **.cpp** ve **.h** dosyası oluşturuyoruz. **Solution Explorer** kısmından **Header Files** klasörüne sağ tıklayıp **Add> New Item** diyerek oluşturabilirsiniz.

.bmp uzantılı 24 bitlik görüntüyü yükleyebilmek için aşağıdaki fonksiyonu **.cpp** uzantılı dosyamıza yazıyoruz.

```
BYTE* LoadBMP(int% width, int% height, long% size, LPCTSTR bmpfile)
{
    // declare bitmap structures
    BITMAPFILEHEADER bmpheader;
    BITMAPINFOHEADER bmpinfo;
    // value to be used in ReadFile funcs
    DWORD bytesread;
    // open file to read from
    HANDLE file = CreateFile(bmpfile, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING,
    FILE_FLAG_SEQUENTIAL_SCAN, NULL);
```

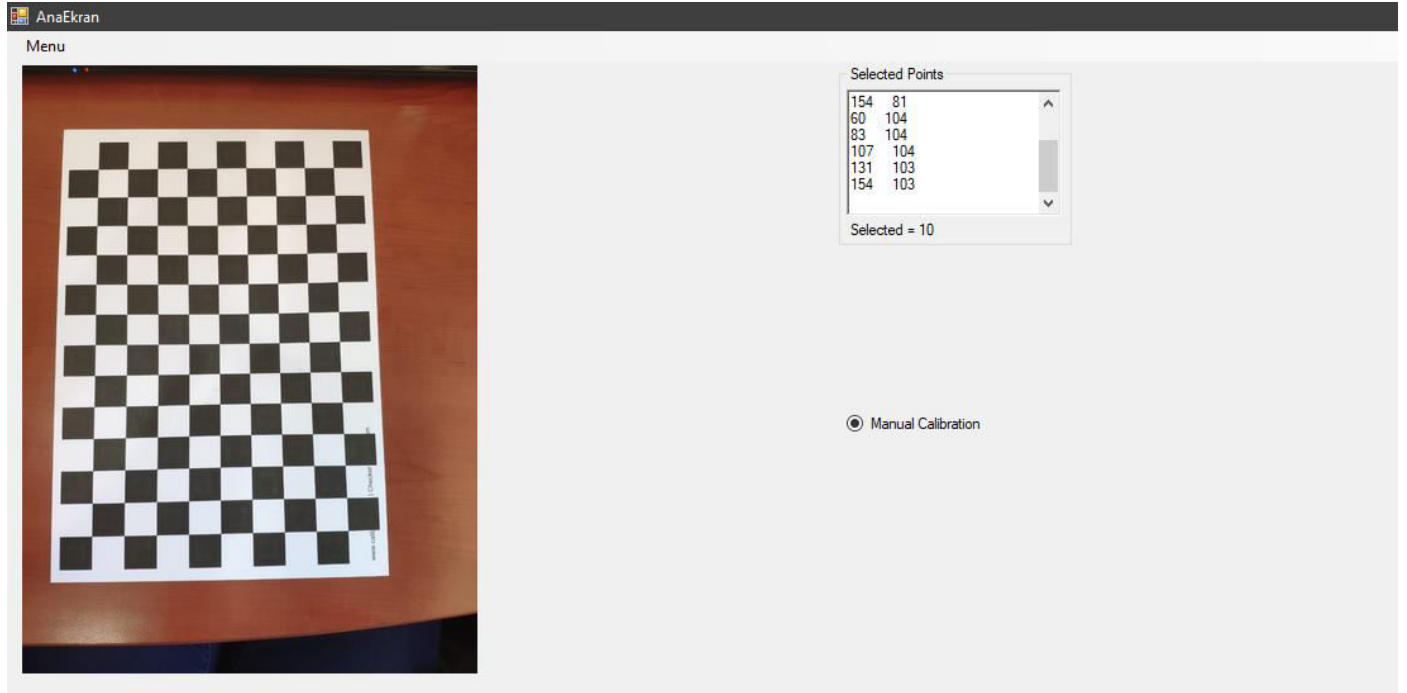
KONTROL SONUCU

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 6
YAPILAN İŞ	PictureBox'ta İmge Gösterme	TARİH : 27.06.19
<pre> if (NULL == file) return NULL; // couldn't open file // read file header if (ReadFile(file, &bmpheader, sizeof(BITMAPFILEHEADER), &bytesread, NULL) == false) { CloseHandle(file); return NULL; } //read bitmap info if (ReadFile(file, &bmpinfo, sizeof(BITMAPINFOHEADER), &bytesread, NULL) == false) { CloseHandle(file); return NULL; } // check if file is actually a bmp if (bmpheader.bfType != 'MB') { CloseHandle(file); return NULL; } // get image measurements width = bmpinfo.biWidth; height = abs(bmpinfo.biHeight); // check if bmp is uncompressed if (bmpinfo.biCompression != BI_RGB) { CloseHandle(file); return NULL; } // check if we have 24 bit bmp if (bmpinfo.biBitCount != 24) { CloseHandle(file); return NULL; } // create buffer to hold the data size = bmpheader.bfSize - bmpheader.bfOffBits; BYTE* Buffer = new BYTE[size]; // move file pointer to start of bitmap data SetFilePointer(file, bmpheader.bfOffBits, NULL, FILE_BEGIN); // read bmp data if (ReadFile(file, Buffer, size, &bytesread, NULL) == false) { delete[] Buffer; CloseHandle(file); return NULL; } // everything successful here: close file and return buffer CloseHandle(file); return Buffer; } //LOADPMB </pre> <p>LoadBMP() fonksiyonunun <i>header</i>'ını da <i>header</i> dosyasına ekliyoruz.</p> <pre> BYTE* LoadBMP(int% width, int% height, long% size, LPCTSTR bmpfile); </pre> <p>Formumuzda görüntü yükleyebilmek için aşağıdaki gibi <i>global</i> değişkenler tanımlıyoruz.</p> <pre> LPCTSTR giris; LPCTSTR cikis; int width; int height; long size; BYTE* buffer; </pre>		
KONTROL SONUCU		

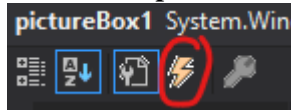
KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 7
YAPILAN İŞ	PictureBox'ta İmge Gösterme	TARİH : 27.06.19
<p>MenuStrip'in ClickEvent'ini oluşturuyoruz (Design kısmında Butonuna 2 kez tıklayarak). Daha sonra Form'da oluşan ClickEvent fonksiyonuna geliyoruz ve aşağıdaki kodları yazıyoruz.</p> <pre>private: System::Void openToolStripMenuItem_Click(System::Object^ sender, System::EventArgs^ e) { if (openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK) { pictureBox1->ImageLocation = openFileDialog1->FileName; CString str; str = openFileDialog1->FileName; giris = (LPCTSTR)str; buffer = LoadBMP(width, height, size, giris); } }</pre> <p>Artık uygulamamızın başlangıç aşaması hazır. Diğer kısımda uygulamaya eklenen görüntü üzerinde Kalibrasyon noktalarını(Calibration points) seçeceğiz.</p>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 8
YAPILAN İŞ	PictureBox'tan Fare İle Kalibrasyon Noktalarını Alma	TARİH : 28.06.19

1.5 PICTUREBOX'TAN FARE İLE KALİBRASYON NOKTALARINI ALMA



Öncelikle formumuza 1 adet **radioButton**, 1 adet **label** ve 1 adet de **richTextBox** ekliyoruz. Daha sonra **pictureBox**'a sağ tıklayıp **Properties** kısmını açıyoruz ve orada **Events** butonuna tıklıyoruz.



Listeden **MouseUp event**'ine iki kez tıklıyoruz. Form'umuzda otomatik olarak bu **event**'in fonksiyonu oluşacaktır.

Form'da global olarak

`int calibCount = 0;` tanımlıyoruz.

Oluşturulan fonksiyona gelip aşağıdaki kodları yazıyoruz.

```
private: System::Void pictureBox1_MouseUp(System::Object^ sender,
System::Windows::Forms::MouseEventArgs^ e) {
    if (radioButton1->Checked) {
        calibCount++;
        label1->Text = "Selected = " + calibCount;
        richTextBox1->AppendText(e->X.ToString() + " " + e->Y.ToString() + " \n");
        richTextBox1->ScrollToCaret();
        StreamWriter^ write = gcnew StreamWriter("image_points.txt");
        write->WriteLine(richTextBox1->Text);
        write->Close();
    }
}
```

Manual Calibration radioButtonu tıklanmış iken **pictureBox**'ta farenin sol tuşu ile tıkladığımız her nokta hem **richTextBox1**'e hem de **image_points.txt** dosyasına yazılacak. Ayrıca kaç kez tıkladığı bilgisini de **label** ile ekranda gösteriyoruz.

KONTROL SONUCU

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 9
YAPILAN İŞ	Kalibrasyon Tahtasının Hazırlanması	TARİH : 01.07.19

1.6 KALİBRASYON TAHTASININ HAZIRLANMASI

Kalibrasyon tahtası için <https://calib.io/pages/camera-calibration-pattern-generator> sitesinden kalibrasyon tahtası hazırlamak için aşağıdaki ayarlamaları yapıyoruz.

Target Type

Checkerboard

Board Width [mm]

297

Board Height [mm]

210

Rows

10

Columns

14

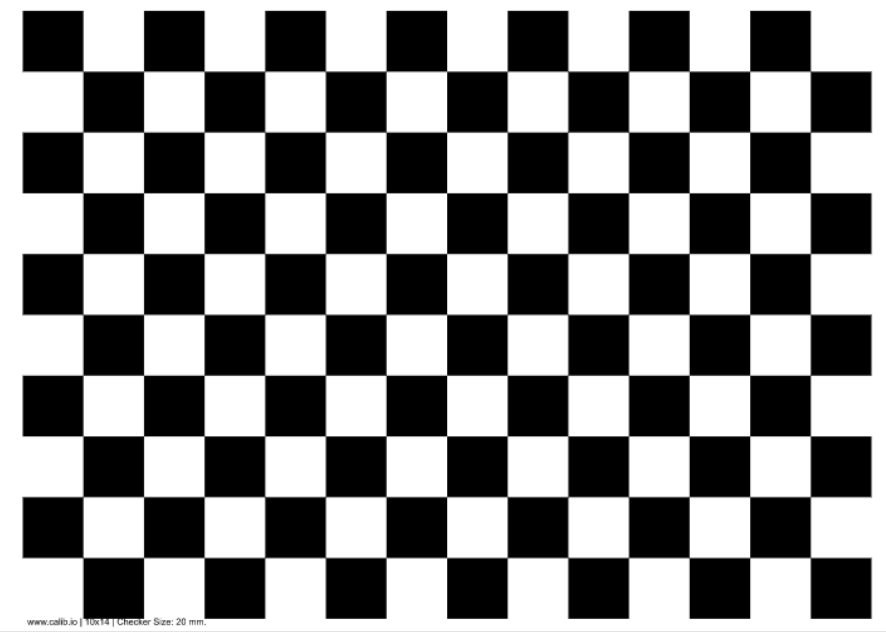
Checker Width / Circle Spacing [mm]

20

Circle Diameter [mm]

5

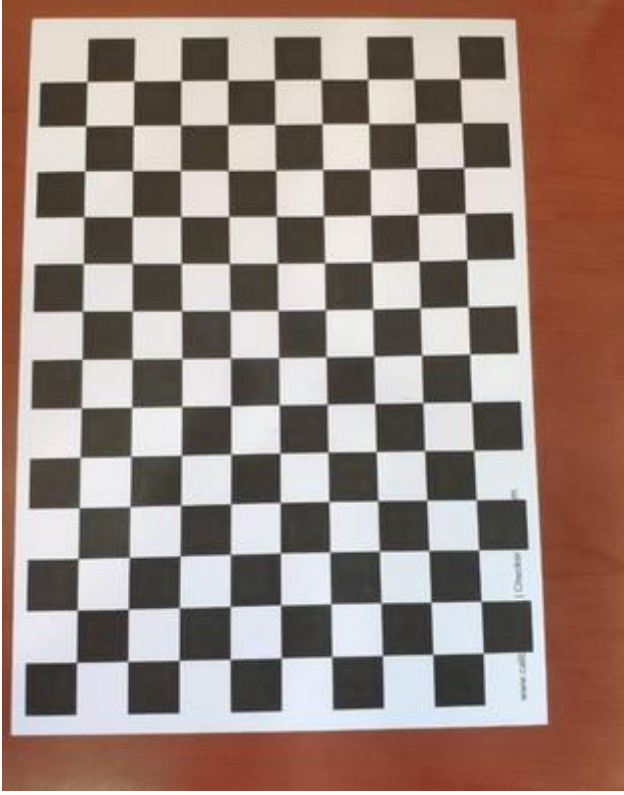
Ayarlamaların sonucunda aşağıdaki gibi bir görüntü elde ediyoruz.



Daha sonra **Save calibration board as PDF** a tıklayıp bir çıktı alıyoruz.

KONTROL SONUCU

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 10
YAPILAN İŞ	Kalibrasyon Tahtasının Hazırlanması	TARİH : 01.07.19



Görüntünün bir fotoğrafını çekip **.jpg** formatında kaydedip daha sonra **OnlineJPG-BMPconverter** üzerinden **.bmp** formatına çeviriyoruz.

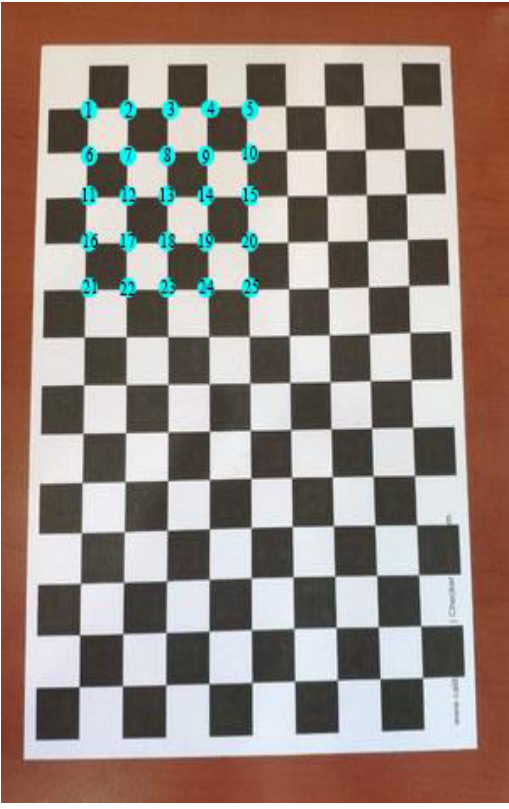
KONTROL SONUCU

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 11
YAPILAN İŞ	Kalibrasyon için Girdilerin Hazırlanması	TARİH : 02.07.19

1.7 KALİBRASYON İÇİN GİRDİLERİN HAZIRLANMASI

Oluşturduğumuz form uygulamasını çalıştırıyoruz. Sonra **.bmp** uzantılı görüntüyü açıyoruz. **Manual Calibration** seçeneği seçili iken 25 tane kalibrasyon noktası belirliyoruz. Noktaları aşağıdaki gibi sıra ile seçeceğiz.

Daha sonra ise noktalar **image_points.txt**'ye kaydedilmiş olacaktır. Metin dosyasının adını **calibration_image_points.txt** olarak değiştirebiliriz.

Gerçek dünya koordinatları	Seçilecek noktalar
0 0 0 20 0 0 40 0 0 60 0 0 80 0 0 0 20 0 20 20 0 40 20 0 60 20 0 80 20 0 0 40 0 20 40 0 40 40 0 60 40 0 80 40 0 0 60 0 20 60 0 40 60 0 60 60 0 80 60 0 0 80 0 20 80 0 40 80 0 60 80 0 80 80 0	

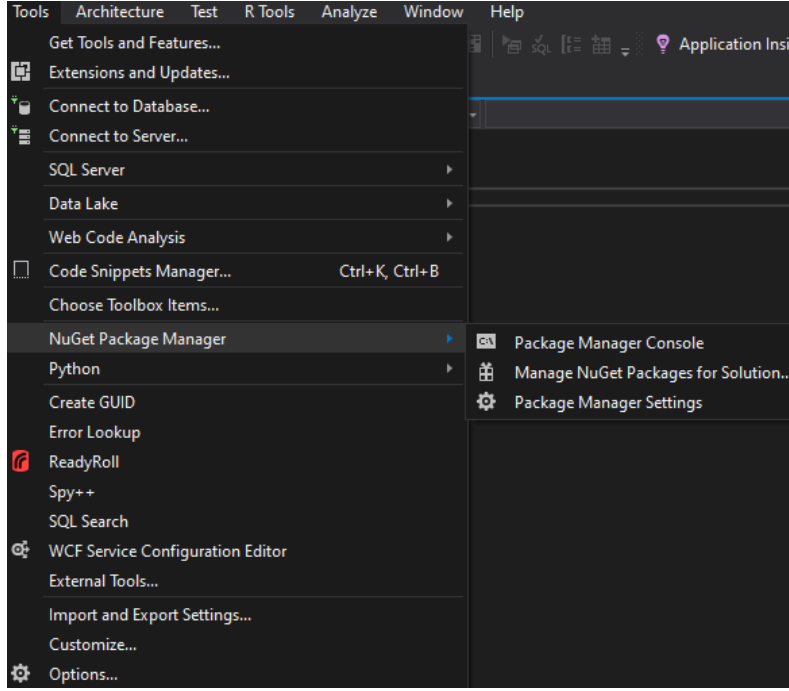
Yukarıda verilen noktalar sırasıyla seçtiğimiz 25 noktanın gerçek dünyada karşılık düştüğü noktalardır. Bu noktaları **calibration_world_points.txt** adında yeni bir metin dosyasına kaydediyoruz.

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 12
YAPILAN İŞ	Eigen Kütüphanesinin eklenmesi.	TARİH : 03.07.19

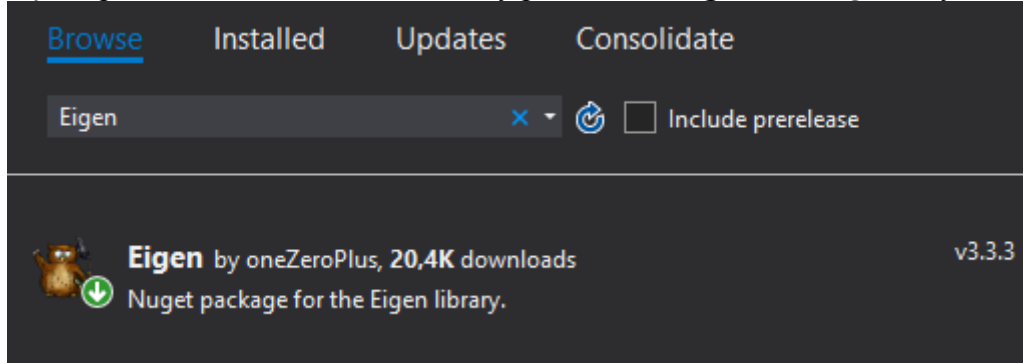
1.8 EIGEN KÜTÜPHANESİNİN EKLENMESİ

Visual Studio için kütüphane ekleme kendi içinde barındırdığı paket yönetim sistemi ile gerçekleştiriliyor. **NuGet Package Manager** diye adlandırılan bu system biz birçok kütüphane ve modülleri sunuyor. Matris hesaplamalarında ve **Kalibrasyon Teorisi** kısmında anlattığım **SVD** yöntemini kullanmada işimizi kolaylaştıracak **Eigen** kütüphanesini ekleyeceğiz.

Öncelikle **araç çubuğu**'ndan Tools'a, sonra NuGet Package Manager'a, sonra da **Manage Nuget Packages for Solution**'a tıklıyoruz.



Açılan pencerede **Browse** kımına tıklayıp **arama çubuğu**'ndan **Eigen** diye aratıyoruz.



Çıkan **Eigen** paketine tıkladıktan sonra sağ tarafta açılan pencereden projemizi seçip **Install** diyoruz. Yeni açılacak olan pencerede **OK** butonuna tıklıyoruz.

Yüklenen kütüphanemiz aracılığı ile fonksiyon ve değişkenleri kullanabilmek için **.cpp** uzantılı dosyamıza aşağıdaki kodları kopyalıyoruz.

```
#include<Eigen/Dense>
#include<Eigen/src/SVD/JacobiSVD.h>
usingnamespace Eigen;
```

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 13
YAPILAN İŞ	Kalibrasyon Fonksiyonu	TARİH : 04.07.19
<p>1.9 KALİBRASYON FONKSİYONU</p> <p>Öncelikle fonksiyonlarımızın yer aldığı .h uzantılı header dosyasına aşağıdaki fonksiyon başlığını yazıyoruz.</p> <pre>void ProjeksiyonMatrisiHesapla(intnoktaSayisi, float * dünyaNoktalari, float * imgeNoktalari, float * A);</pre> <p>Bu fonksiyonun parametreleri sırasıyla kalibrasyon işleminin kaç nokta kullanılarak yapılacağı, bu noktaların gerçek dünya koordinatları dizisi, bu noktaların görüntü üzerindeki koordinatlarının dizisi ve sonuç olarak hesaplanan projeksiyon matrisinin yazılacağı dizi.</p> $\begin{bmatrix} X_1Y_1Z_110000-X_1x_1-Y_1y_1-Z_1z_1 \\ 0000X_1Y_1Z_11-X_1y_1-Y_1x_1-Z_1z_1 \\ X_2Y_2Z_210000-X_2x_2-Y_2y_2-Z_2z_2 \\ 0000X_2Y_2Z_21-X_2y_2-Y_2x_2-Z_2z_2 \\ X_3Y_3Z_310000-X_3x_3-Y_3y_3-Z_3z_3 \\ 0000X_3Y_3Z_31-X_3y_3-Y_3x_3-Z_3z_3 \\ \vdots \\ X_nY_nZ_n10000-X_nx_n-Y_ny_n-Z_nz_n \\ 0000X_nY_nZ_n1-X_ny_n-Y_nx_n-Z_nz_n \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{24} \\ a_{41} \\ a_{42} \\ a_{43} \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \\ \vdots \\ x_n \\ y_n \end{bmatrix}$ <p>DQ = R</p> <p>Kalibrasyon Teorisi kısmında bahsettiğimiz yukarıdaki matrisleri hatırlamakta fayda var. Fonksiyonun gövdesi aşağıdaki gibidir.</p> <pre>void ProjeksiyonMatrisiHesapla(intnoktaSayisi/*kalibrasyonda kullanılacak nokta sayisi*/, float* dünyaNoktalari/*gercek dünya koordinatlari*/, float* imgeNoktalari/*goruntudeki koordinatlar*/, float* A/*projeksiyon matrisi*/) { //D Q = R esitliginde D ve R'yi biliyoruz ve dolduruyoruz MatrixXf D(noktaSayisi * 2, 11); //D matrisi VectorXf R(noktaSayisi * 2); // R vektoru float X, Y, Z, x, y, z; // D matrisinde kullanılacak 2D ve 3D duzlem noktaları for (int i = 0; i < noktaSayisi; i++) { /*Gercek dünya noktalarından (3 boyutlu) X,Y,Z degerlerini belirliyoruz*/ X = dünyaNoktalari[i * 3]; Y = dünyaNoktalari[i * 3 + 1]; Z = dünyaNoktalari[i * 3 + 2]; /*fotograftaki noktalardan (2 boyutlu) x ve y degerlerini belirliyoruz*/ x = imgeNoktalari[i * 2]; y = imgeNoktalari[i * 2 + 1]; /*R matrisini dolduruyoruz*/ R(i * 2) = x; R(i * 2 + 1) = y; } }</pre>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 14
YAPILAN İŞ	Kalibrasyon Fonksiyonu	TARİH : 04.07.19
<pre> /*D matrisinde 1 noktaya karsilik olarak 2 denklem var. Bu matris icin sablon satirlar tanimlaniyor*/ /*X,Y,Z ve x,y degerleri yukarida atanirken, her iterasyonda D matrisi icin denk dusen degerler asagida atanacak*/ float mevcutNoktalarD[22] = { X,Y,Z,1, 0,0,0,0, -X * x, -Y * y, -Z * z, 0,0,0,0, X,Y,Z,1, -X * y, -Y * y, -Z * y }; /*D'yi dolduruyoruz*/ for (int j = 0; j < 11; j++) { D(i * 2, j) = mevcutNoktalarD[j]; D(i * 2 + 1, j) = mevcutNoktalarD[j + 11]; } } /*SVD (Singular Value Decomposition) yontemini kullanarak DQ=R denkleminde bilinen D ve R'ye gore Q matrisini cozuyoruz*/ VectorXf solution(11); //Q matrisi JacobiSVD<MatrixXf> svd(D, ComputeFullU ComputeFullV); solution = svd.solve(R); //Q matrisini(vektorunu) bulduktan sonra, artik A projeksiyon matrisini belirleyebilirsiniz for (int i = 0; i < 11; i++) { A[i] = solution(i); } //A projeksiyon matrisi icin tum degerleri belirliyoruz fakat sondaki a44 degeri teoride 1'e esit oldugu icin ona direkt 1 atiyoruz A[11] = 1; } </pre>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 15
YAPILAN İŞ	İmar Fonksiyonu	TARİH :05.07.19

1.10 İMAR FONKSİYONU

Öncelikle fonksiyonlarımızın yer aldığı .h uzantılı header dosyasına aşağıdaki fonksiyon başlığını yazıyoruz.

```
void İmarEt(intnoktaSayisi, float * imarImgeNoktalari, float * p, float * w);
```

Bu fonksiyonun parametreleri sırasıyla kalibrasyon işleminin kaç nokta kullanılarak yapılacağı, bu noktaların görüntü üzerindeki koordinatlarının dizisi, imar için kullanılacak projeksiyon matrisi ve elde edilen gerçek dünya koordinatlarının üzerine yazacağı bu noktaların sayısı uzunluğundaki diziyi gösteriyor.

$$\begin{bmatrix} xp_{31} - p_{11}xp_{32} - p_{12}xp_{33} - p_{13} \\ yp_{31} - p_{21}yp_{32} - p_{22}yp_{33} - p_{23} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} p_{14} - xp_{34} \\ p_{24} - yp_{34} \end{bmatrix}$$

Projeksiyon Matrisini kullanarak 2 boyutlu İmar kısmında bahsettiğimiz yukarıdaki matrisleri hatırlamakta fayda var.

Fonksiyonun gövdesi aşağıdaki gibidir.

```
void İmarEt(intnoktaSayisi/*imar edilecek noktaların sayısı*/, float* imarImgeNoktalari/*imar edilecek noktaların görüntü koordinatlarını tutar*/,
float* p/*projeksiyon matrisi*/, float* w/*gerçek dünya koordinatları*/) {

    /*
    [a]   [X]
    [b]   = P x [Y]
    [w]           [Z]
                  |
                  | [1]
                  |
    |             |
    |             | ----> 3D (3 boyutlu) dünya koordinatları
    |             |
    |             | ----> projeksiyon matrisi
    |             |
    |             | ----> 2D (2 boyutlu) dünya koordinatları
    */

    MatrixXf A(2, 3);
    Vector2f B(2, 1);

    float x, y;

    for (int i = 0; i < noktaSayisi; i++) {
        x = imarImgeNoktalari[i * 2];
        y = imarImgeNoktalari[i * 2 + 1];
    }
}
```

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 16
YAPILAN İŞ	İmar Fonksiyonu	TARİH :05.07.19
<pre> /* xp31 - p11, xp32 - p12, xp33 - p13 yp31 - p21, yp32 - p22, yp33 - p23 A matrisi */ A << x * p[8] - p[0], x*p[9] - p[1], x*p[10] - p[2], y*p[8] - p[4], y*p[9] - p[5], y*p[10] - p[6]; //Bilinmeye bir matris var o da asagidaki /* [X] [Y] --> W matrisi [Z] bu 3x1 lik matrisi bulmak icin A matrisinin B matrisine gore SVD yontemi ile cozecegiz */ /* p14 - xp34 p24 - yp34 B matrisi */ B <<p[3] - x * p[11], p[7] - y * p[11]; Vector3f solution; JacobiSVD<MatrixXf> svd(A, ComputeFullU ComputeFullV); solution = svd.solve(B); w[i * 3] = solution(0); w[i * 3 + 1] = solution(1); w[i * 3 + 2] = solution(2); /* SVD uygulandiktan sonra XYZ degerlerini yani 3D dunyanin 2D dunyaya projeksiyonunu bulmus oluyoruz */ } </pre>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 17
YAPILAN İŞ	Dosya İşlemleri için Fonksiyon	TARİH :08.07.19
1.11 DOSYA İŞLEMLERİ İÇİN FONKSİYON <p>Kalibrasyon için Girdilerin Hazırlanması kısmında hakkında bahsettiğimiz ve oluşturduğumuz metin dosyalarından okunacak noktalar verisi için yazacağımız fonksiyonun başlık kısmı aşağıdaki gibidir.</p> <pre>void dosyadanDegerleriOku(intnoktaSayisi, constchar * imgeYolu, constchar * dünyaYolu, float * imge, float * dünya);</pre> <p>Fonksiyonun parametreleri sırasıyla, kalibrasyon için noktaların sayısı, görüntü noktalarının koordinatlarının okunacağı yol, gerçek dünya noktalarının koordinatlarının okunacağı yol, görüntü için okunan noktaların atanacağı dizi, gerçek dünya için okunan noktaların atanacağı diziyi gösterir.</p> <p>Fonksiyonun gövdesi aşağıdaki gibidir.</p> <pre>void dosyadanDegerleriOku(intnoktaSayisi, constchar* imgeYolu, constchar* dünyaYolu, float* imge, float* dünya) { FILE* imgeDosya; //calibration_image_points.txt dosyası için File Pointer FILE* dünyaDosya;//calibration_world_points.txt dosyası için File Pointer imgeDosya = fopen(imgeYolu, "r"); //okuma modunda calibration_image_points.txt dosyasını açtık rewind(imgeDosya); dünyaDosya = fopen(dünyaYolu, "r");//okuma modundan calibration_world_points.txt dosyasını açtık rewind(dünyaDosya); for (int i = 0; i <noktaSayisi; i++) { // kaç tane nokta kalibrasyon noktası belirlediysek o kadar dönecek fscanf(imgeDosya, "%f", &imge[i * 2]); //görüntü noktalarının x koordinatlarını okuyup imge dizisine atadık fscanf(imgeDosya, "%f", &imge[i * 2 + 1]);//görüntü noktalarının y koordinatlarını okuyup imge dizisine atadık fscanf(dünyaDosya, "%f", &dünya[i * 3]);//gerçek dünya noktalarının x koordinatlarını okuyup dünya dizisine atadık fscanf(dünyaDosya, "%f", &dünya[i * 3 + 1]);//gerçek dünya noktalarının y koordinatlarını okuyup dünya dizisine atadık fscanf(dünyaDosya, "%f", &dünya[i * 3 + 2]);//gerçek dünya noktalarının z koordinatlarını okuyup dünya dizisine atadık } fclose(imgeDosya); //calibration_image_points.txt dosyası için açtığımız File Pointer'ı kapatıyoruz fclose(dünyaDosya);//calibration_world_points.txt dosyası için açtığımız File Pointer'ı kapatıyoruz }</pre>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 18
YAPILAN İŞ	Formda Kalibrasyon İşlemi	TARİH :09.07.19

1.12 FORMDA KALİBRASYON İŞLEMİ

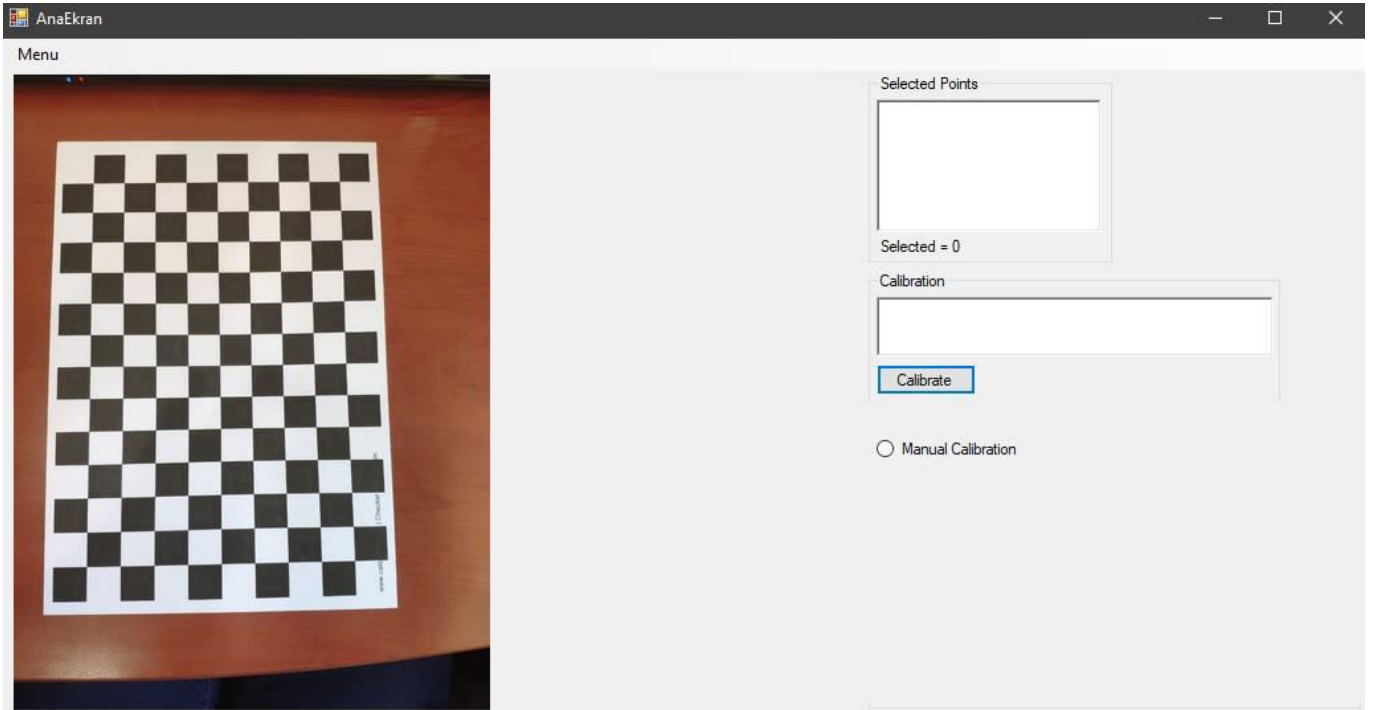
Öncelikle aşağıdaki değişkenleri *global*'de tanımlıyoruz.

```
float* dünya;
float* imge;
float* projeksiyon = newfloat[12];

constchar* imgeYolu = "calibration_image_points.txt";
constchar* dünyaYolu = "calibration_world_points.txt";
```

Daha sonra arayüz kısmında bir adet richTextBox, bir adet Button ekliyoruz. Devamında Butona ClickEvent veriyoruz ve oluşan fonksiyonun içinde kalibrasyon işlemini gerçekleştiriyoruz.

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
    richTextBox2->Clear();
    imge = newfloat[calibCount * 2]; //imge dizisinin boyutu seçilen nokta sayısına göre
    belirleniyor.(x ve y olduğu için 2'ye çarpılıyor)
    dünya = newfloat[calibCount * 3]; //imge dizisinin boyutu seçilen nokta sayısına göre
    belirleniyor.(x,y ve z olduğu için 3'e çarpılıyor)
    dosyadanDegerleriOku(calibCount, imgeYolu, dünyaYolu, imge, dünya);
    ProjeksiyonMatrisiHesapla(calibCount, dünya, imge, projeksiyon);
    for (int i = 0; i < 3; i++) {
        richTextBox2->AppendText(
            projeksiyon[i * 4 + 0].ToString() + ""
            + projeksiyon[i * 4 + 1].ToString() + ""
            + projeksiyon[i * 4 + 2].ToString() + ""
            + projeksiyon[i * 4 + 3].ToString() + " \n");
    }
    StreamWriter^ yaz = gcnewStreamWriter("projection.txt");
    yaz->Writeline(richTextBox2->Text);
    yaz->Close();
}
```



KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 19
YAPILAN İŞ	Formda İmar İşlemi	TARİH :10.07.19

1.13 FORMDA İMAR İŞLEMİ

Öncelikle aşağıdaki değişkenleri *global*'de tanımlıyoruz.

```
int p1x, p1y, p2x, p2y, p3x, p3y;
float w1x, w1y, w2x, w2y, w3x, w3y;
```

Daha sonra arayüz kısmında üç adet radioButton, sekiz adet label ekliyoruz.

Daha sonra MouseUp event'inin içine aşağıdaki kodları yazıyoruz.

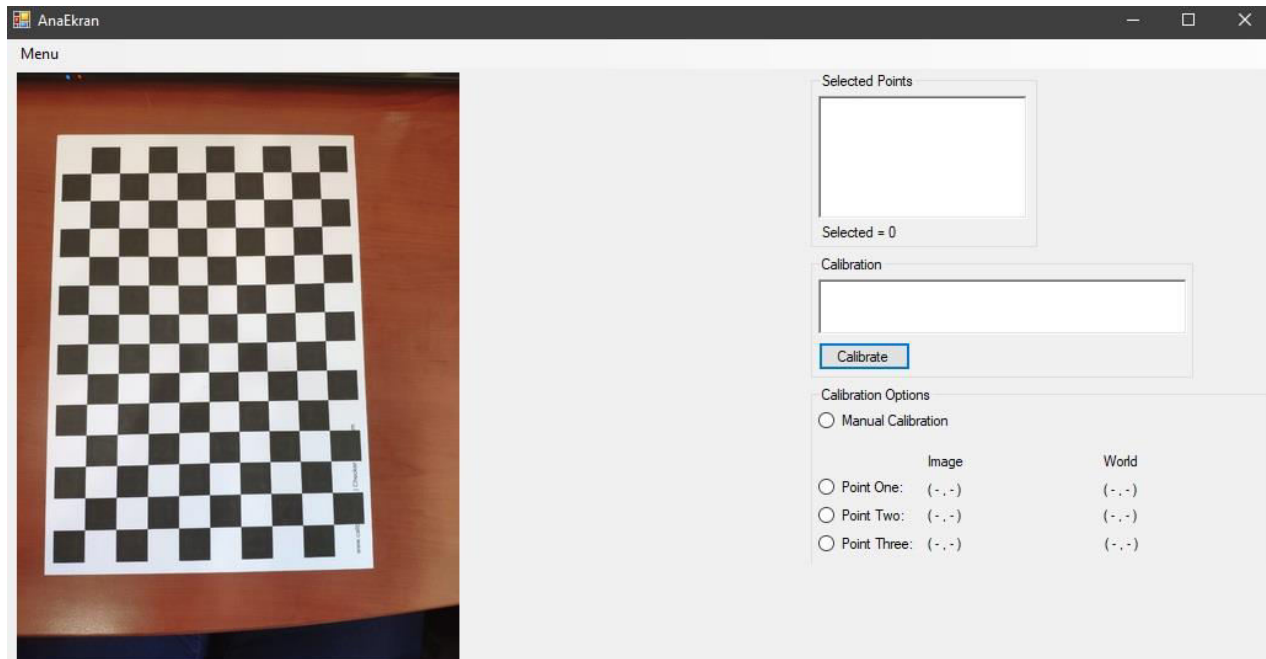
```
elseif(radioButton2->Checked) //Point One seçildi ise
{
    p1x = e->X; //farenin x noktasını p1x'e ata
    p1y = e->Y; //farenin y noktasını p1x'e ata

    label2->Text = "( "+ p1x.ToString() +", "+ p1y.ToString() +)"; // seçilen x
ve y koordinatını Image sütununda Point One satırına yazdırıyoruz.

    float* test1 = newfloat[2]; // seçilmiş x ve y noktalarını atayacağımız geçici
dizi oluşturuluyor(sadece x ve y olduğu için boyutu 2).
    test1[0] = p1x;
    test1[1] = p1y;
    float* gercekDunya = newfloat[3]; //ImarEt fonksiyonundan geri dönecek olan
gerçek dünya dizisi oluşturuluyor.
    ImarEt(1, test1, projeksiyon, gercekDunya); // projeksiyon matrisi sonucunda
hesaplanan gercekDunya değişkeni dolduruluyor

    w1x = gercekDunya[0]; //yeni x ve y değerlerine sahip gercekDunya değişkeni
ekranda gösterilmek üzere atanıyor
    w1y = gercekDunya[1];
    label7->Text = "( "+ w1x.ToString() +", "+ w1y.ToString() +)"; // hesaplanan
gerçek dünya için x ve y koordinatını World sütununda Point One satırına yazdırıyoruz.

    delete[] test1;
    delete[] gercekDunya;
}
```



KONTROL SONUCU

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 20
YAPILAN İŞ	Formda İmar İşlemi	TARİH :10.07.19
<pre> elseif (radioButton3->Checked)//Point Two seçildi ise { p2x = e->X; p2y = e->Y; label3->Text = "("+ p2x.ToString() +" , "+ p2y.ToString() +")"; float* test1 = newfloat[2]; test1[0] = p2x; test1[1] = p2y; float* gercekDunya = newfloat[3]; ImarEt(1, test1, projeksiyon, gercekDunya); w2x = gercekDunya[0]; w2y = gercekDunya[1]; label8->Text = "("+ w2x.ToString() +" , "+ w2y.ToString() +")"; delete[] test1; delete[] gercekDunya; } elseif (radioButton4->Checked)//Point Three seçildi ise { p3x = e->X; p3y = e->Y; label4->Text = "("+ p3x.ToString() +" , "+ p3y.ToString() +")"; float* test1 = newfloat[2]; test1[0] = p3x; test1[1] = p3y; float* gercekDunya = newfloat[3]; ImarEt(1, test1, projeksiyon, gercekDunya); w3x = gercekDunya[0]; w3y = gercekDunya[1]; label9->Text = "("+ w3x.ToString() +" , "+ w3y.ToString() +")"; delete[] test1; delete[] gercekDunya; } </pre>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 21
YAPILAN İŞ	Görüntü Üzerinde Mesafe ve Alan Ölçümü	TARİH :11.07.19

1.14 GÖRÜNTÜ ÜZERİNDE MESAFE ÖLÇÜMÜ

Aşağıdaki değişkenleri *global*'de tanımlıyoruz.

```
double mesafe;
double alan;
```

Öncelikle arayüz kısmında bir adet button ve bir adet de label ekliyoruz.

$P_1(x_1, y_1)$: *Point One*

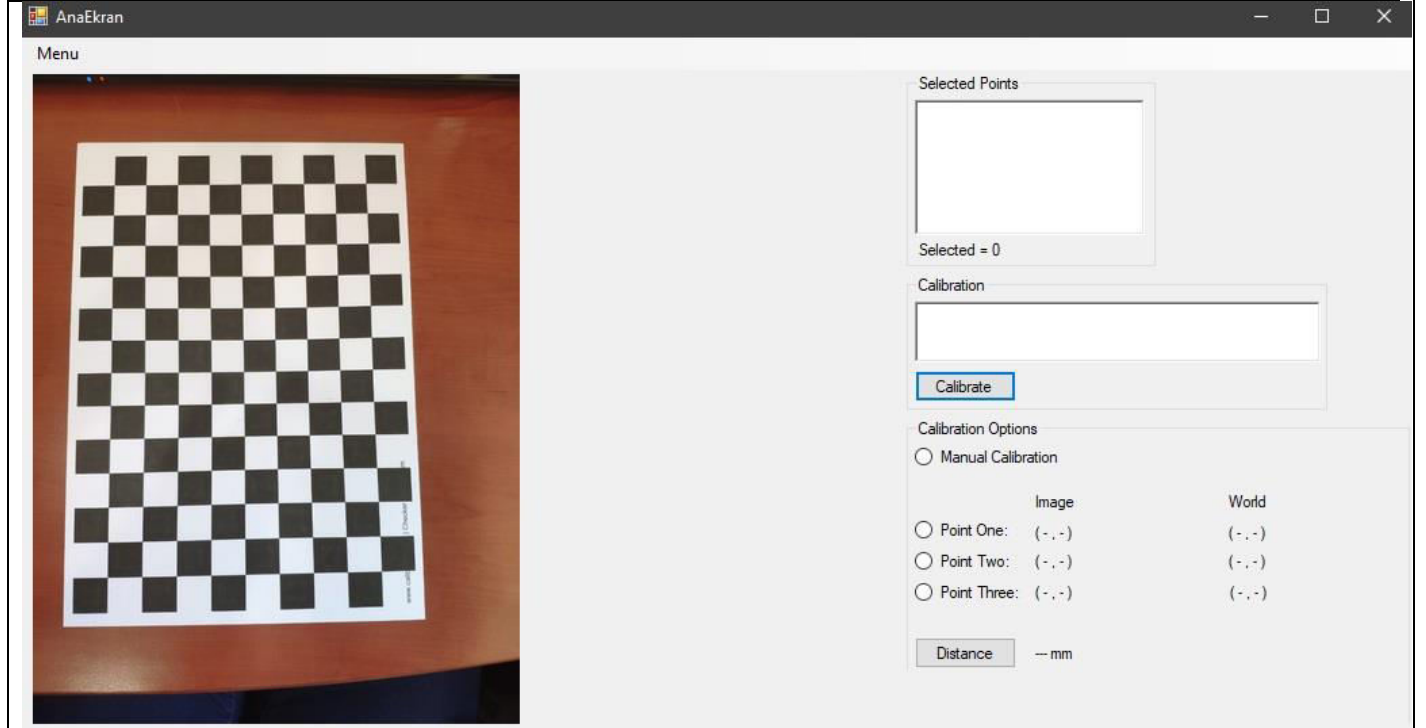
$P_2(x_2, y_2)$: *Point Two*

$$|P_1P_2| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Distance butonu için *clickEvent* oluşturuyoruz. Oluşan fonksiyonun içine aşağıdaki kodları yazıyoruz.

```
mesafe = sqrt(pow(w1x - w2x, 2) + pow(w1y - w2y, 2));
label10->Text = mesafe.ToString() + " mm";
```

Artık iki nokta arasındaki mesafeyi ölçebiliriz.



KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 22
YAPILAN İŞ	Görüntü Üzerinde Mesafe ve Alan Ölçümü	TARİH :11.07.19

1.15 GÖRÜNTÜ ÜZERİNDE ALAN ÖLÇÜMÜ

Arayüz kısmında bir adet button ve bir adet de label ekliyoruz.

$P_1(x_1, y_1)$: *Point One*

$P_2(x_2, y_2)$: *Point Two*

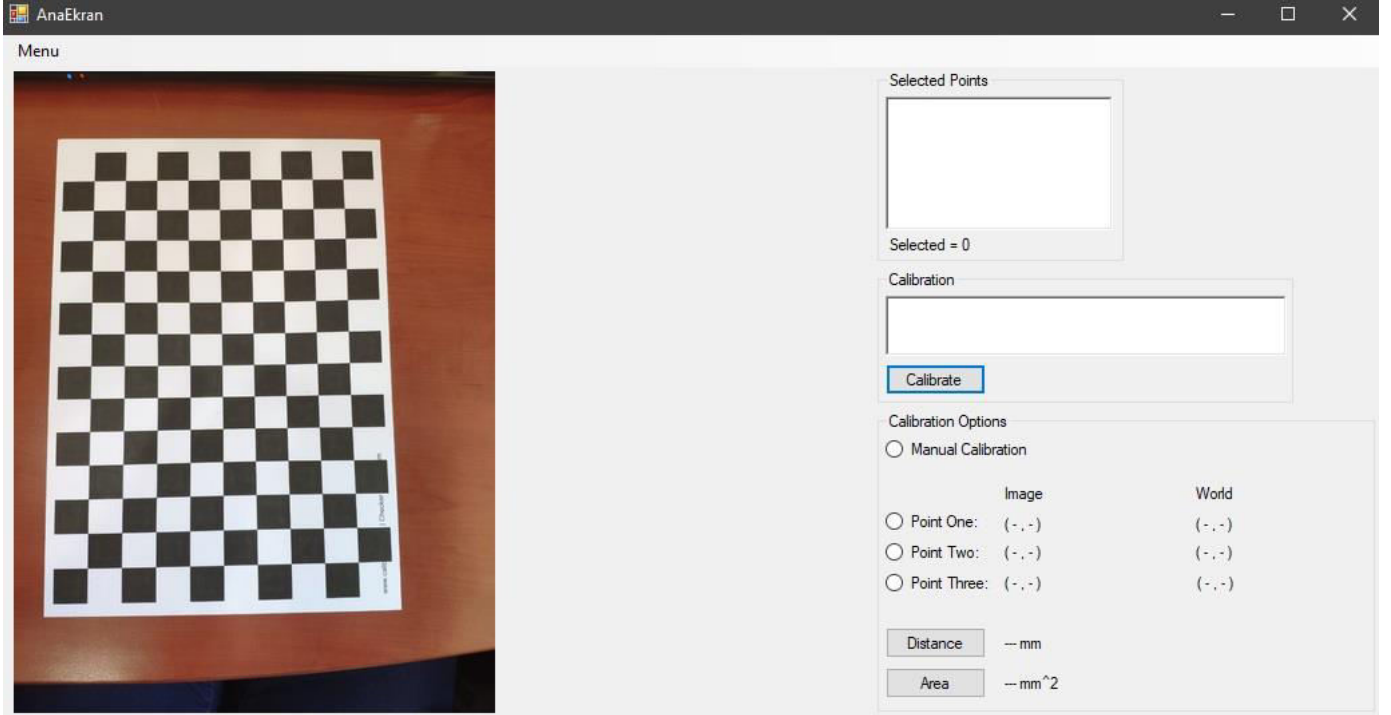
$P_3(x_3, y_3)$: *PointThree*

$$Area = \frac{1}{2} \vee x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2) \vee$$

Area butonu için *clickEvent* oluşturuyoruz. Oluşan fonksiyonun içine aşağıdaki kodları yazıyoruz.

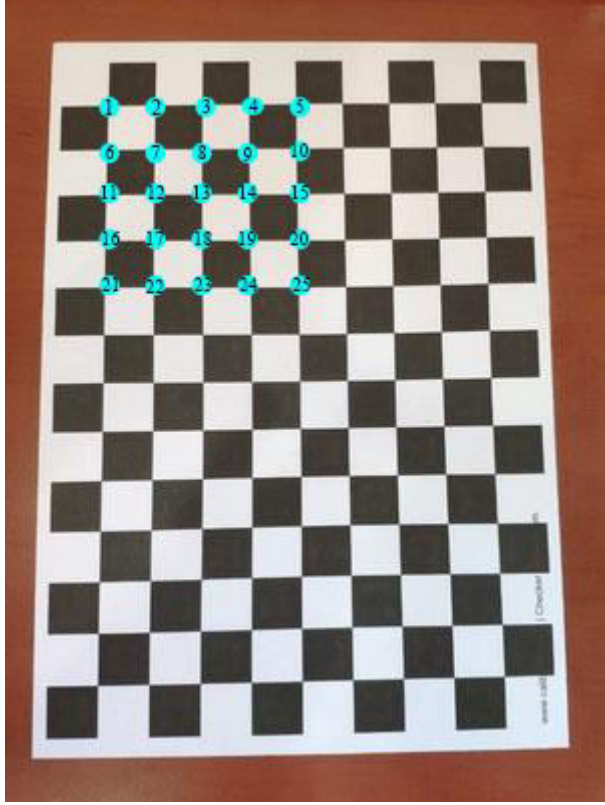
```
alan = abs((w1x*(w1y - w3y) + w2x * (w3y - w1y) + w3x * (w1y - w2y))/2);
label111->Text = alan.ToString() + " mm^2";
```

Artık üç nokta arasındaki alanı bulabiliriz.

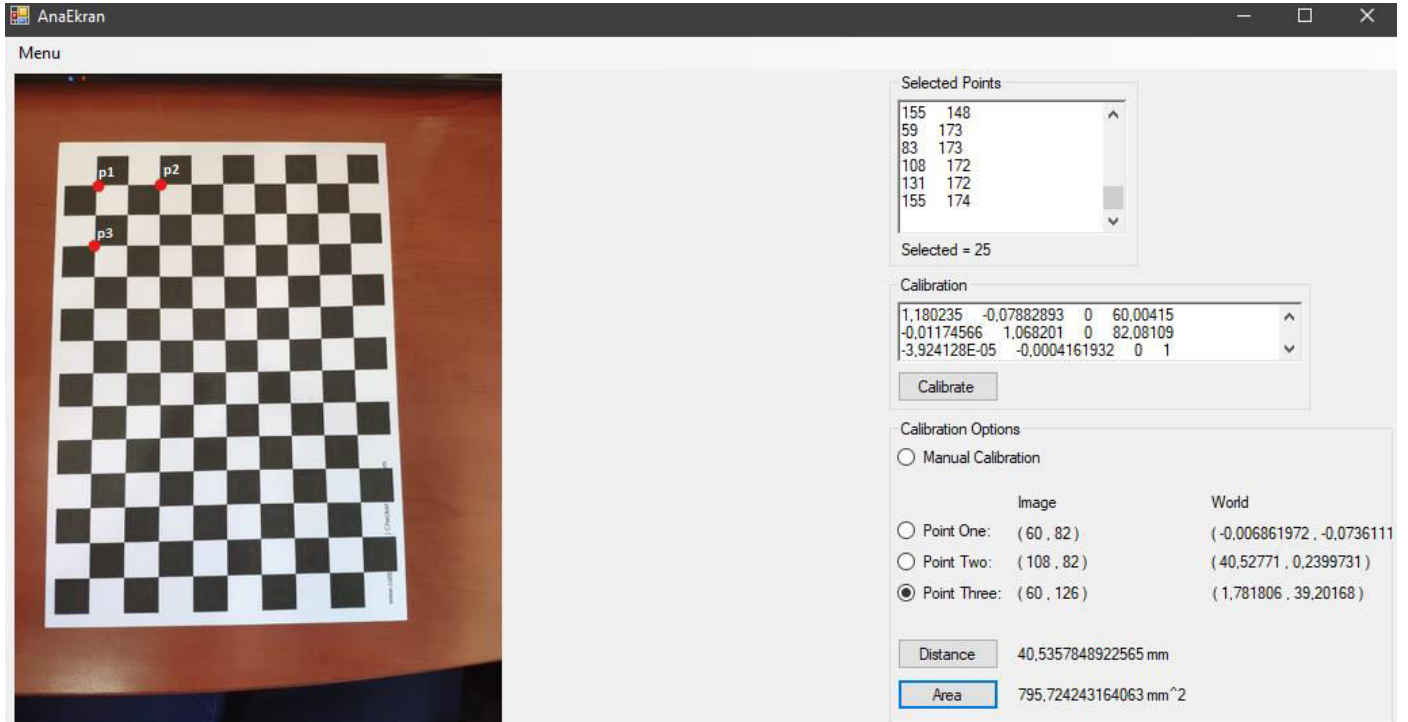


KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 23
YAPILAN İŞ	Uygulamanın Test Edilmesi	TARİH :12.07.19

1.15 UYGULAMANIN TEST EDİLMESİ



Öncelikle **Manual Calibration** seçili iken görüntü üzerinde **25** noktayı seçiyoruz. Daha sonra **Calibrate** Button'una tıklıyoruz. Sonuçlarımızı kontrol ediyoruz. Noktalarımız kalibre edildikten sonra aşağıdaki görüntüdeki gibi ölçüm noktalarını belirliyoruz.



KONTROL SONUCU

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 24
YAPILAN İŞ	Uygulamanın Test Edilmesi	TARİH :12.07.19

Kalibrasyon Tahtasının Hazırlanması kısmında karelerin kenar ölçüsünü **20 mm** olarak belirlemiştik. Bu yüzden P_1 ve P_2 arasındaki mesafe **40 mm** olmalı. Bizim de hesapladığımız mesafe yaklaşık olarak **40 mm** çıktı.

Alanı hesaplarken ise $(40 \times 40)/2$ işleminden **800 mm²** olarak çıkıyor. Bizim sonucumuz da yaklaşık olarak bu değer çıktı.

Hesaplama sonuçlarında çıkan hatalar, yani **800 mm²** yerine **795,72 mm²** çıkması, **40 mm** yerine **40,53mm** sonuçları manuel kalibrasyon ile alakalıdır. Bizim el ile görüntü üzerinde seçtiğimiz kalibrasyon noktası tam olarak karelerin köşesine denk gelmediği için sonuçlar arasında sapma problemi ortaya çıkıyor. Bu hatayı gidermek için manuel nokta seçme yerine **Hough** kullanarak otomatik olarak köşe noktalarını belirleyeceğiz.

KONTROL SONUCU

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 25
YAPILAN İŞ	Projeye OpenCv Kütüphanesinin Eklenmesi.	TARİH :16.07.19

1.16 PROJEYE OPENCV KÜTÜPHANESİNİN EKLENMESİ

Uygulamayı test ederken ortaya çıkan problemlerden birisi de noktaların koordinatlarının belirlenmesiydi. Şöyle ki manuel olarak belirlediğimizden dolayı tam olarak istediğimiz köşe noktasını seçemiyoruz. Bu sorunu ortadan kaldırmak ve kalibrasyon noktalarının belirlenmesini otomatikleştirmek için OpenCV aracılığıyla Hough Transform kullanacağız. İlk aşamada Hough ile kenarları belirleyeceğiz. İkinci aşamada kenarların kesişimini referans alarak kalibrasyon noktalarını belirleyeceğiz. Hough'u kullanabilmek için öncelikle OpenCV kütüphanesini kurmalıyız. Aşağıdaki bağlantılar üzerinden OpenCV'yi kurabilirsiniz.

<https://www.deciphertechnic.com/install-opencv-with-visual-studio/>

https://www.youtube.com/watch?v=M-VHaLHC4XI&t=607s&ab_channel=DecipherTechnic

Kurulumu yaptıktan sonra .cpp uzantılı fonksiyonlarımızın bulunduğu dosyaya aşağıdaki kodu yazıyoruz.

```
#include<opencv2/opencv.hpp>
usingnamespace cv;
```

Artık OpenCV kütüphanelerini kullanabiliriz.

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 26
YAPILAN İŞ	Hough ile Görüntü Üzerinde Doğruları Bulma(OpenCV Kullanarak)	TARİH :17.07.19

1.17 HOUGH İLE GÖRÜNTÜ ÜZERİNDE DOĞRULARI BULMA(OPENCV KULLANARAK)

Şimdiye kadar görüntümüzün üzerinde kalibrasyon noktalarını manuel olarak alıyorduk, fakat şimdi bu noktaların belirlenmesini otomatik yapacağız. Uygulamamızda **Hough** kullanarak ilk önce doğruları bulacağız. Daha sonra bulduğumuz doğruların kesişim noktalarını belirleyip gerekli eliminasyonlar sonrasında kalibrasyon noktalarımıza elde edeceğiz.

NOT: Görüntü üzerinde doğruların bulunması ve **OpenCV** ile uygulamanması konusuna

<https://www.learnopencv.com/hough-transform-with-opencv-c-python/> bağlantısı üzerinden erişebilirsiniz.

Aşağıdaki **DogruBul()** fonksiyonu aracılığı ile görüntümüzdeki doğruları bulup bu doğruların başlangıç ve bitiş noktalarını **noktalar.txt** dosyasına yazdırıyoruz.

```
int DogruBul() {
    fstream dosya;
    dosya.open("noktalar.txt"); //bulacagimiz tum cizgilerin baslangic ve bitis noktalarinin x ve y koordinatlarini bu dosyaya kaydedecegiz
    dosya.clear();
    Mat src = imread("bmp.bmp",0); //goruntuyu aliyoruz ve gri seviyeye donusturuyoruz.
    Mat dst, cdst; // 2 tane matris olusturuyoruz, uygulayacagimiz fonksiyonlari sonucunu yazdirmak icin
    Canny(src, dst, 50, 200, 3); // cizgileri bulabilmek icin once Canny uyguluyoruz.
    cvtColor(dst, cdst, COLOR_GRAY2BGR); //GRAY seviyeden BGR seviyesine donusturuyoruz
    vector<Vec2f> lines; //Hough uygulanmasi sonucu cizgileri yazdiracagimiz bir vektor turunden degisken tanimliyoruz.
    // Vec2f anlama 2 tane float degiskene sahip olmasi ve bu vektor degiskeninin bir struct oldugunu gosteriyor.
    HoughLines(dst, lines, 1, CV_PI / 180, 150, 0, 0); // Hough uygulayarak fotograftaki dogrulari(cizgileri) buluyoruz.
    /*Hough sonucunda buldugumuz dogrulari fotografa cizdirecegiz*/
    for (size_t i = 0; i < lines.size(); i++) // buldugumuz cizgilerin sayisi kadar gidiyoruz.
    {
        float rho = lines[i][0]; // RHO hesaplamalarda kullanilacak piksel olarak cozunurluk
        float theta = lines[i][1]; // THETA hesaplamalarda kullanilacak radyan turunden aci
        Point pt1, pt2; // PT1 ve PT2 her bir cizginin baslangic ve bitis konumu: Point turunden olduklari icin herbirinin x ve y degerleri var.
        /*her bir cizgiyi(dogruyu) cizdirebilmek icin hesaplamalar asagidaki gibidir.*/
        double a = cos(theta);
        double b = sin(theta);
        double x0 = a * rho;
        double y0 = b * rho;
        pt1.x = cvRound(x0 + 1000 * (-b));
        pt1.y = cvRound(y0 + 1000 * (a));
        pt2.x = cvRound(x0 - 1000 * (-b));
        pt2.y = cvRound(y0 - 1000 * (a));
        /*Hesaplamalarin sonu*/
        dosya << pt1.x << "\t" << pt1.y << "\t" << pt2.x << "\t" << pt2.y << endl;
    }
    // her bir dogrunun baslangic ve bitis noktalarinin
    // x,y koordinatlarini dosyaya yazdiriyoruz
    line(cdst, pt1, pt2, Scalar(0, 0, 255), 1); // line() fonksiyonu ile goruntunun uzerine hesaplamalar sonucu buldugumuz noktalarin arasinda olan dogruyu cizdiriyoruz
}
imshow("bulunan dogrular", cdst); // goruntu uzerine tum dogrulari cizdikten sonra imshow() fonksiyonu ile gosteriyoruz
waitKey();
return 0;
}
```

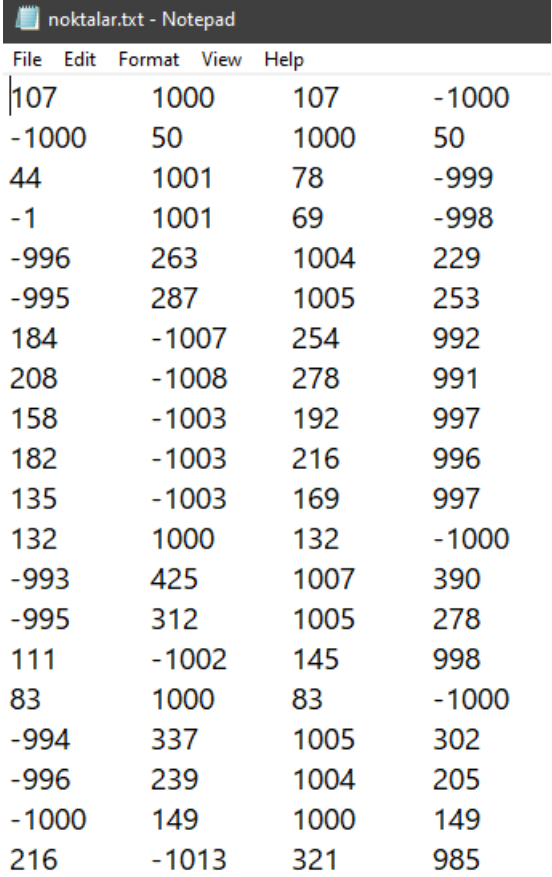
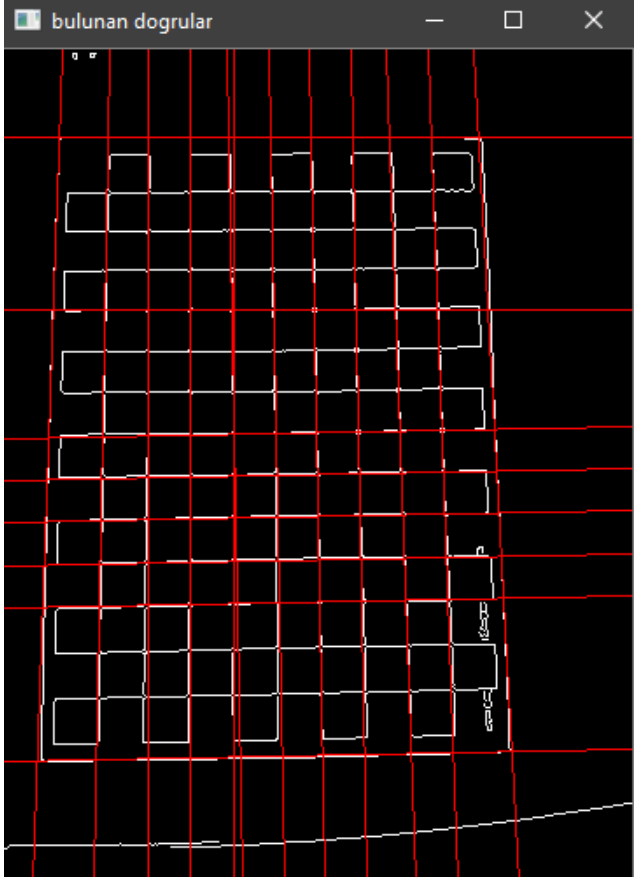
Yukarıdaki fonksiyonu **.cpp** dosyamıza yazıyoruz.

NOT: Kodun işlevi yorum satırlarında yazmaktadır. Ayrıca DogruBul() fonksiyonunu Open buttonunun event fonksiyonunun içerisinde çağırabiliriz.

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 27
YAPILAN İŞ	Hough ile Görüntü Üzerinde Doğruları Bulma(OpenCV Kullanarak)	TARİH :17.07.19

Sonuç olarak göreceğimiz çıktı ve **noktalar.txt** dosyasının içeriği aşağıdaki gibidir.

NOT: Dosya içeriğinde sütunlar sırasıyla **doğrununbaşlangıcının x'i, y'si, doğrunun bitişinin x'i, y'si**.

Dosya içeriği	Bulunmuş doğruların çıktısı
 <pre> 107 1000 107 -1000 -1000 50 1000 50 44 1001 78 -999 -1 1001 69 -998 -996 263 1004 229 -995 287 1005 253 184 -1007 254 992 208 -1008 278 991 158 -1003 192 997 182 -1003 216 996 135 -1003 169 997 132 1000 132 -1000 -993 425 1007 390 -995 312 1005 278 111 -1002 145 998 83 1000 83 -1000 -994 337 1005 302 -996 239 1004 205 -1000 149 1000 149 216 -1013 321 985 </pre>	

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 28
YAPILAN İŞ	Doğruların Kesişim Noktasını Bulma(Teori)	TARİH :18.07.19

1.18 DOĞRULARIN KESİŞİM NOKTASINI BULMA(TEORİ)

İki doğrunun kesişim noktasını bulmak için çeşitli yöntemler vardır. Biz de geometri kullanarak öncelikle doğruları matematiksel ifade edeceğiz. Daha sonra bu matematiksel denklem sistemini çözeceğiz.

Bulacağımız sonuç bizim kesişim noktamız olacaktır.

Öncelikle iki noktamız olsun, bunlar (x_1, y_1) ve (x_2, y_2) diye başlangıç ve bitiş noktalarıdır. Bu noktaların oluşturduğu doğrunun deklemini doğru dekleminden bulabiliriz.

Aşağıdaki gibi iki doğrumuz varsa bu doğruların kesişimini bulmak için ise

$$1. \quad a_1x + b_1y = c_1$$

$$2. \quad a_2x + b_2y = c_2$$

yukarıdaki iki denklemi çözmeliyiz. Çözmek için 1. Denklemini b_2 ile, 2. denklemi ise b_1 ile çarpmalı daha sonra 1. denklemden 2. denklemi çıkarmalıyız. Bu işlemler sonucunda aşağıdaki sonucu elde ederiz.

$$\bullet \quad (a_1b_2 - a_2b_1)x = c_1b_2 - c_2b_1$$

Yukarıdaki denklemde x 'i yalnız bırakıp kesişim noktasının x konumunu bulabiliriz. Aynı işlemleri y 'i bulmak için de yapabiliriz.

Yukarıda yapılan işlemlerin *pseudo* kodu aşağıdaki gibidir.

```

determinant = a1b2 - a2b1
if (determinant == 0)
{
// Doğrular kesişmiyor
}
else
{
x = (c1b2 - c2b1)/determinant
y = (a1c2 - a2c1)/determinant
}

```

Yukarıdaki *pseudo* kodda hesaplama sonucunda bulunan x ve y değerleri iki doğrunun kesişim noktasının x ve y koordinatlarıdır.

Kodlamada da bu adımları gerçekleştireceğiz. Fakat doğruların matematiksel tanımında bir ön adım yapmamız gerek. Bu adımlar aşağıdaki tabloda solda gösterilmiştir. Sağdaki şekil ise soldaki adımların nereden geldiğini gösteriyor.

Kod üzerinde doğrunun matematiksel tanımı	Soldaki denklemlerin geometrik gösterimi
<pre> a1 = B.y - A.y; b1 = A.x - B.x; c1 = a1 * (A.x) + b1 * (A.y); </pre>	

KONTROL SONUCU

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 29
YAPILAN İŞ	Doğruların Kesişim Noktasını Bulma(Uygulama)	TARİH :19.07.19

1.19 DOĞRULARIN KESİŞİM NOKTASINI BULMA(UYGULAMA)

Teori kısmında anlattığımız konunun kodlanmasını ve uygulamaya dönüştürülmesini bu kısımda anlatacağız. Aşağıdaki fonksiyonu **.cpp** uzantılı dosyamıza yazıyoruz. Bu fonksiyonu DogruBul fonksiyonunun içerisinde **imshow()** fonksiyonunun koştugu satırdan hemen önce çağırabiliriz.

```
void kesisimBul() {
    ifstream dosya;
    dosya.open("noktalar.txt"); // dogruBul() fonksiyonundaki noktaları yazdığımız dosyadan bu
    sefer okuma yapacağız
    ofstream cikti;
    cikti.open("out.txt"); // kesisim noktalarını yazacağımız cikti dosyası
    int dizi[80];
    Point A, B, C, D; // 2 doğru var ve bu 2 doğrunun 2 noktası var(baslangic ve bitis). A ve B
    ilk 1. doğru için noktalar olsun, C ve D 2. doğru için noktalar
    int x, y; // kesisim noktalarının x ve y noktalarını dosyaya yazdırmak için x ve y
    degiskenlerini tanımlıyoruz.
    Mat src = imread("bmp.bmp"); // üzerinde kesisim noktalarını göstereceğimiz görüntü
    int width = src.size().width; //görüntü width'i
    int height = src.size().height; //görüntünün height'i
    /* noktalar dosyasından koordinatları okuyoruz ve diziye yazıyoruz*/
    for (int i = 0; i < 20; i++) {
        dosya >> dizi[i * 4];
        dosya >> dizi[i * 4 + 1];
        dosya >> dizi[i * 4 + 2];
        dosya >> dizi[i * 4 + 3];
    }
    /*kesisim testini uyguluyoruz*/
    for (int i = 0; i < 20; i++) {
        /*ilk doğru için baslangic ve bitis konumlarını A ve B noktalarına tanımlıyoruz*/
        A.x = dizi[i * 4];
        A.y = dizi[i * 4 + 1];
        B.x = dizi[i * 4+2];
        B.y = dizi[i * 4+3];
        for (int j = i+1; j < 20; j++)
        {
            /*geriye çizgiler için baslangic ve bitis konumlarını C ve D noktalarına tanımlıyoruz */
            C.x = dizi[j * 4];
            C.y = dizi[j * 4+1];
            D.x = dizi[j * 4+2];
            D.y = dizi[j * 4+3];
            /*determinan bulabilmek için a1,b1,c1 ve a2,b2,c2 hesaplanıyor*/
            double a1 = B.y - A.y;
            double b1 = A.x - B.x;
            double c1 = a1 * (A.x) + b1 * (A.y);
            double a2 = D.y - C.y;
            double b2 = C.x - D.x;
            double c2 = a2 * (C.x) + b2 * (C.y);
            //determinant hesaplanıyor
            double determinant = a1 * b2 - a2 * b1;
            if (determinant == 0)
            {
                //hicbirsey yapma, cunku paraleller
            }
        }
    }
}
```

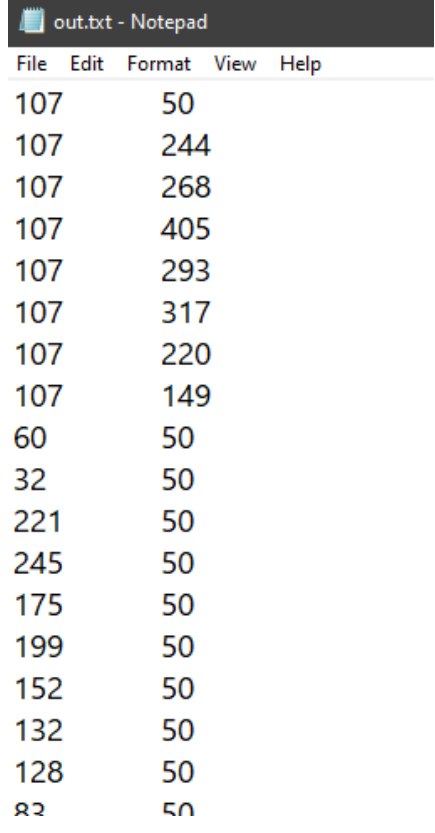
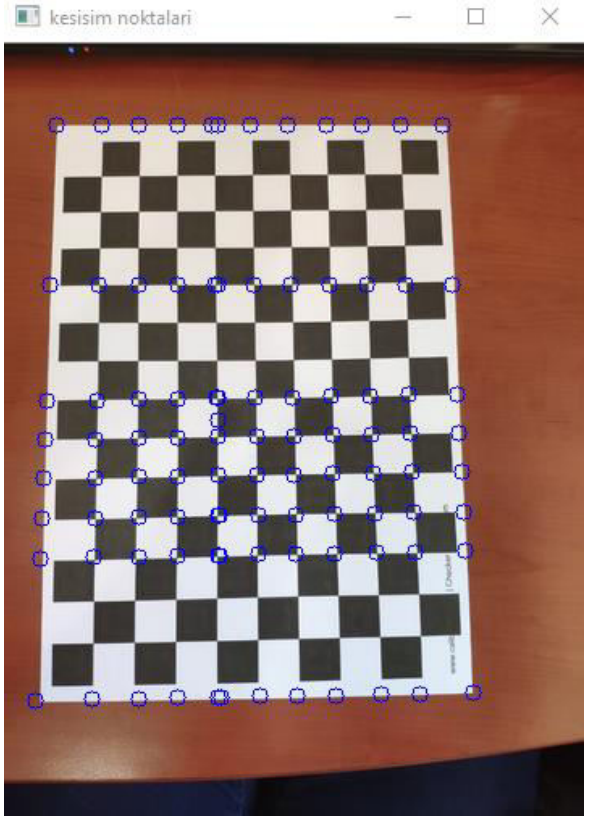
KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 30
YAPILAN İŞ	Doğruların Kesişim Noktasını Bulma(Uygulama)	TARİH :19.07.19

```

else
{
    x = (b2*c1 - b1 * c2) / determinant;
    y = (a1*c2 - a2 * c1) / determinant;
    if (x<0 || x > width || y < 0 || y > height) { // eger kesisim noktasi
        //hicbirsey yapma, cunku araliga dahil degil
    }
    else {
        cikti << x <<"\t"<< y << endl; // kesisim noktaları araliga
        dahilse kesisim noktalarını dosyaya yazdır
        circle(src, Point(x, y), 5, Scalar(255, 0, 0));
    }
}
}
}
cikti.close();
dosya.close();
imshow("kesisim noktaları", src);
}

```

Öncelikle doğruların başlangıç ve bitiş koordinatlarını yazdığımız dosyadan okuma yapıp o doğruları bir diziye yazıyoruz. Daha sonra dizideki doğruları teker teker kesişim testine sokuyoruz. En sonda bulduğumuz sonuçları çıktı dosyasına yazdırıyoruz. Ayrıca kesişim noktaları dosyaya yazarken **OpenCV**'nin **circle()** fonksiyonunu kullanarak görüntü üzerinde işaretliyoruz. Sonuç olarak dosya içeriği ve çıktı aşağıdaki gibidir.

Dosya içeriği	Bulunmuş kesişim noktalarının çıktısı
	

NOT: Kodun işlevi yorum satırlarında yazmaktadır.

KONTROL SONUCU

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 31
YAPILAN İŞ	Fonksiyonları Dinamik Yapma(1/3)	TARİH :22.07.19

1.20 FONKSİYONLARI DİNAMİK YAPMA(1/3)

OpenCV kurulumu sonrasında yaptığımız fonksiyonlardan aldığımız sonuçları dosyaya yazdırıyorduk. Ayrıca elimizde sadece bir örnek görüntü olduğundan döngülerin de döngü sayısını, dizilerin boyutunu ve saire statik olarak belirledik. Tabii bunlar deneme amaçlıydı. Fonksiyonlarımızın bu hali istenen bir durum değil. Verileri dosyaya yazmak ve oradan okumak da sürekli bir iş yükü getiriyor. Bu kısımda bu iş yükünden kurtulmayı anlatacağım

İşlemlerimizde en son kesişim noktalarını bulduk. İyi veya kötü elimizde bir veri kümesi oldu. Fakat istemediğimiz sonuçlarla da karşılaştık. Örneğin kesişim noktasının hatalı çıkması, köşe olmayan bir noktada hatalı kesişim noktası bulma ve b. Bu sorunları gidermek ve gerçek kesişim noktası belirlemek için ilerideki kısımlarda kesişim noktalarını elimine etme ile alakalı algoritma geliştirip bunu uygulayacağız. Şimdi ise gelelim asıl konumuza. Öncelikle dosyalarla olan ilişkimizi kesmemiz ve bir class dizisi oluşturup verileri oraya yazıp oradan okumamız gerekiyor. Fakat Doğruları bulmak için yazdığımız fonksiyonda veriler sürekli artıyor ve maksimum ne kadar boyut gerekli bunu tahmin edemiyoruz. Bu problem gidermek için Single Linked List Veri Yapısı kullanacağız. Kullandığım Veri Yapısı ile alakalı olan teorik bilgilere SARTAJ SAHNI'nin Data Structures, Algorithms And Applications In C++ kitabında 172. sayfasından erişebilirsiniz.

Teorik bilgileri edindikten sonra kodlama kısmına geçebiliriz.

Önce aşağıdaki gibi bir sınıf oluşturuyoruz.

```
class Noktalar {
public:
    int veri;
    Noktalar* sonraki;
};
```

veri değişkeni bulunan noktaların sırasıyla ekleneceği değişken, sonraki değişken ise bir pointer olup bir dizide bir sonraki değeri göstermekle yükümlü.

Fonksiyonun bulunduğu nokta konumunu eklemek için ise noktaEkle() fonksiyonumuz aşağıdaki gibi:

```
void noktaEkle(Noktalar** baslikReferans, int veri)
{
    Noktalar* yeniNokta = new Noktalar();// yeni dugum olustur
    yeniNokta->veri = veri;// veriyi koy
    yeniNokta->sonraki = (*baslikReferans);//eski listeyi dugumden ayir
    (*baslikReferans) = yeniNokta;//yeni dugume isaret etmesi icin basligi tasi
}
```

Eklediğimiz herhangi bir konumdaki noktaya erişmek için ise noktaGetir() fonksiyonumuz aşağıdaki gibi:

```
int noktaGetir(Noktalar* baslik, int indis) { //Bağlantılı listenin başlık göstergesini alır ve
argüman olarak girilen indisdeki veriyi döndürür
    Noktalar* simdiki = baslik;

    int sayac = 0;
    /*pointer bizim girdiğimiz indisi gösterene kadar sayac arttırılır.*/
    while (simdiki != NULL) {
        if (sayac == indis) {
            return(simdiki->veri);
        }
        sayac++;
        simdiki = simdiki->sonraki;
    }
}
```

KONTROL SONUCU

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 32
YAPILAN İŞ	Fonksiyonları Dinamik Yapma (1/3)	TARİH :22.07.19
<p>Ayrıca listemizde kaç tane veri olduğu bilgisine de erişmemiz gerek. Bunun için ise aşağıdaki fonksiyonu yazıyoruz:</p> <pre>int boyutBulNoktalar(Noktalar* baslik) { //Bağlantılı listenin başlık göstergesini alır ve başlıkta kaç tane düğüm olduğunu döndürür Noktalar* simdiki = baslik; int sayac = 0; while (simdiki != NULL) { //liste sonuna gelene kadar şimdiki düğümü gösteren pointer bir sonrakine işaret eder. sayac++; simdiki = simdiki->sonraki; } return sayac; }</pre> <p>Bu fonksiyonların hemen altında globalde Noktalar* noktalarBaslik = NULL; listesini tanımlayabiliriz.</p>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 33
YAPILAN İŞ	Fonksiyonları Dinamik Yapma (2/3)	TARİH :23.07.19
1.21 FONKSİYONLARI DİNAMİK YAPMA (2/3) <p>Bir önceki kısımda yaptığımız liste Hough Transformdan dönen nokta konumları verisini eklemek içindi. Şimdi ise oluşturacağımız liste kesişim testinden dönen verileri eklemek için olacak.</p> <p>Öncelikle aşağıdaki gibi bir sınıf oluşturuyoruz:</p> <pre> class Kesisimler { public: int veri; Kesisimler* sonraki; }; </pre> <p>Kesişim noktalarını ekleyebilmek için aşağıdaki fonksiyonu yazıyoruz:</p> <pre> void kesisimEkle(Kesisimler** baslikReferans, int veri) { Kesisimler* yeniKesisim = new Kesisimler();// yeni dugum olustur yeniKesisim->veri = veri;// veriyi koy yeniKesisim->sonraki = (*baslikReferans);//eski listeyi dugumden ayir (*baslikReferans) = yeniKesisim;//yeni dugume isaret etmesi icin basligi tasi } </pre> <p>Kesişim noktalarını eklediğimiz listede herhangi indisteki veriye erişmek için aşağıdaki fonksiyonu yazıyoruz:</p> <pre> int kesisimGetir(Kesisimler* baslik, int indis) { //Bağlantılı listenin başlık göstergesini ve aranan indisi alır ve argüman olarak girilen indisdeki veriyi döndürür Kesisimler* simdiki = baslik; int sayac = 0; /*pointer bizim girdiğimiz indisi gösterene kadar sayac arttırılır.*/ while (simdiki != NULL) { if (sayac == indis) { return(simdiki->veri); } sayac++; simdiki = simdiki->sonraki; } } </pre> <p>Kesişim noktalarının eklendiği listede toplamda kaç adet veri olduğunu öğrenmek için aşağıdaki fonksiyonu yazıyoruz:</p> <pre> int boyutBulKesisimler(Kesisimler* baslik) { //Bağlantılı listenin başlık göstergesini alır ve başlıkta kaç tane düğüm olduğunu döndürür Kesisimler* simdiki = baslik; int sayac = 0; while (simdiki != NULL) { //liste sonuna gelene kadar simdiki düğümü gösteren pointer bir sonrakine isaret eder. sayac++; simdiki = simdiki->sonraki; } return sayac; } </pre> <p>Bu fonksiyonların hemen altında globalde <code>Kesisimler* kesisimlerBaslik = NULL;</code> listesini tanımlayabiliriz.</p>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 34
YAPILAN İŞ	Fonksiyonları Dinamik Yapma (3/3)	TARİH :24.07.19
1.22 FONKSİYONLARI DİNAMİK YAPMA (3/3) <p>DogruBul() fonksiyonunun dosyaya veri yazmasını kaldırıp yerine Linked List'e veri eklemesini sağlıyoruz. Fonksiyonda line() fonksiyonunun çağrıldığı yerin hemen üstüne aşağıdaki kodu yazıyoruz:</p> <pre>//pt1.x -> pt1.y -> pt2.x -> pt2.y -> NULL noktaEkle(&noktalarBaslik, pt2.y); noktaEkle(&noktalarBaslik, pt2.x); noktaEkle(&noktalarBaslik, pt1.y); noktaEkle(&noktalarBaslik, pt1.x);</pre> <p>Yorum satırında eklemeler sonrasında listede nasıl sıralama oluşacağı gösterilmiştir. kesimBul() fonksiyonu için de bazı değişiklikler yapmamız gerek. Öncelikle nokta sayısını belirlemek için çağırığımız fonksiyonu değiştiriyoruz:</p> <pre>int noktaSayisi = boyutBulNoktalar(noktalarBaslik);</pre> <p>Dosyadan noktaları okuyup kesişim testini uyguladığımız döngüler içerisinde yapacağımızı değişiklikler aşağıdaki gibi:</p> <pre>/*ilk dogru icin baslangic ve bitis konumlarini A ve B noktalarina tanimliyoruz*/ A.x = noktaGetir(noktalarBaslik, (i * 4)); A.y = noktaGetir(noktalarBaslik, (i * 4 + 1)); B.x = noktaGetir(noktalarBaslik, (i * 4 + 2)); B.y = noktaGetir(noktalarBaslik, (i * 4 + 3)); /*geriye cizgiler icin baslangic ve bitis konumlarini C ve D noktalarina tanimliyoruz */ C.x = noktaGetir(noktalarBaslik, (j * 4)); C.y = noktaGetir(noktalarBaslik, (j * 4 + 1)); D.x = noktaGetir(noktalarBaslik, (j * 4 + 2)); D.y = noktaGetir(noktalarBaslik, (j * 4 + 3));</pre> <p>Devamında circle() fonksiyonunun koştugu satırın hemen yukarısında dosyaya yazdırmak yerine aşağıdaki gibi listeye ekliyoruz:</p> <pre>//x -> y -> NULL kesimEkle(&kesimBaslik, y); kesimEkle(&kesimBaslik, x);</pre> <p>Ve sonrasında tüm dosya bağımlılıklarını ve hata veren satırları siliyoruz.</p>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 35
YAPILAN İŞ	Fonksiyonları Dinamik Yapma (3/3)	TARİH :24.07.19
Sonuç olarak DogruBul() ve kesisimBul() fonksiyonlarımız aşağıdaki gibi olacak:		
<pre> int DogruBul() { Mat gray; cvtColor(src, gray, COLOR_BGR2GRAY); Mat dst, cdst; // 2 tane matris olusturuyoruz, uygulayacagimiz fonksiyonlari sonucunu yazdirmak icin Canny(gray, dst, 50, 200, 3); // cizgileri bulabilmek icin once Canny uyguluyoruz. cvtColor(dst, cdst, COLOR_GRAY2BGR); //GRAY seviyeden BGR seviyesine donusturuyoruz vector<Vec2f> lines;//Hough uygulanmasi sonucu cizgileri yazdiracagimiz bir vektor turunden degisken tanimliyoruz. // Vec2f anlami 2 tane float degiskene sahip olmasi ve bu vektor degiskeninin bir struct oldugunu gosteriyor. HoughLines(dst, lines, 1, CV_PI / 180, 150, 0, 0); // Hough uygulayarak fotograftaki dogrulari(cizgileri) buluyoruz. /*Hough sonucunda buldugumuz dogrulari fotografa cizdirecegiz*/ for (size_t i = 0; i < lines.size(); i++) // buldugumuz cizgilerin sayisi kadar gidiyoruz. { float rho = lines[i][0]; // RHO hesaplamalarda kullanilacak piksel olarak cozunurluk float theta = lines[i][1]; // THETA hesaplamalarda kullanilacak radyan turunden aci Point pt1, pt2; // PT1 ve PT2 her bir cizginin baslangic ve bitis konumu: Point turunden olduklari icin herbirinin x ve y degerleri var. /*her bir cizgiyi(dogruyu) cizdirebilmek icin hesaplamalar asagidaki gibidir.*/ double a = cos(theta); double b = sin(theta); double x0 = a * rho; double y0 = b * rho; pt1.x = cvRound(x0 + 1000 * (-b)); pt1.y = cvRound(y0 + 1000 * (a)); pt2.x = cvRound(x0 - 1000 * (-b)); pt2.y = cvRound(y0 - 1000 * (a)); /*Hesaplamalarin sonu*/ //pt1.x -> pt1.y -> pt2.x -> pt2.y -> NULL noktaEkle(&noktalarBaslik, pt2.y); noktaEkle(&noktalarBaslik, pt2.x); noktaEkle(&noktalarBaslik, pt1.y); noktaEkle(&noktalarBaslik, pt1.x); line(cdst, pt1, pt2, Scalar(0, 0, 255), 1); // line() fonksiyonu ile goruntunun uzerine hesaplamalar sonucu buldugumuz noktalarin arasinda olan dogruyu cizdiriyoruz } kesisimBul(); //dogrular arasinda kalan kesisimi bulabilmek icin kesisimBul() fonksiyonunu cagiriyoruz. imshow("bulunan dogrular", cdst); // goruntu uzerine tum dogrulari cizdikten sonra imshow() fonksiyonu ile gosteriyoruz waitKey(); return 0; } </pre>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 36
YAPILAN İŞ	Fonksiyonları Dinamik Yapma (3/3)	TARİH :24.07.19
<pre> void kesisimBul() { int noktaSayisi = boyutBulNoktalar(noktalarBaslik); Point A, B, C, D; // 2 dogru var ve bu 2 dogrunun 2 noktasi var(baslangic ve bitis). A ve B ilk 1. dogru icin noktalar olsun, C ve D 2. dogru icin noktalar int x, y; // kesisim noktalarinin x ve y noktalarini dosyaya yazdirmek icin x ve y degiskenlerini tanimliyoruz. int width = src.size().width; //goruntunu width'i int height = src.size().height; //goruntunun height'i /*kesisim testini uyguluyoruz*/ for (int i = 0; i < noktaSayisi/4; i++) { /*ilk dogru icin baslangic ve bitis konumlarini A ve B noktalarina tanimliyoruz*/ A.x = noktaGetir(noktalarBaslik, (i * 4)); A.y = noktaGetir(noktalarBaslik, (i * 4 + 1)); B.x = noktaGetir(noktalarBaslik, (i * 4 + 2)); B.y = noktaGetir(noktalarBaslik, (i * 4 + 3)); for (int j = i+1; j < noktaSayisi/4; j++) { /*geriye cizgiler icin baslangic ve bitis konumlarini C ve D noktalarina tanimliyoruz */ C.x = noktaGetir(noktalarBaslik, (j * 4)); C.y = noktaGetir(noktalarBaslik, (j * 4 + 1)); D.x = noktaGetir(noktalarBaslik, (j * 4 + 2)); D.y = noktaGetir(noktalarBaslik, (j * 4 + 3)); /*determinan bulabilmek icin a1,b1,c1 ve a2,b2,c2 hesaplaniyor*/ double a1 = B.y - A.y; double b1 = A.x - B.x; double c1 = a1 * (A.x) + b1 * (A.y); double a2 = D.y - C.y; double b2 = C.x - D.x; double c2 = a2 * (C.x) + b2 * (C.y); //determinant hesaplaniyor double determinant = a1 * b2 - a2 * b1; if (determinant == 0) { //hicbirsey yapma, cunku paraleller } else { x = (b2*c1 - b1 * c2) / determinant; y = (a1*c2 - a2 * c1) / determinant; if (x<0 x > width y < 0 y > height) { // eger kesisim noktasi goruntuden tasmissa //hicbirsey yapma, cunku araliga dahil degil } else { //x -> y -> NULL kesisimEkle(&kesisimlerBaslik, y); kesisimEkle(&kesisimlerBaslik, x); circle(src, Point(x, y), 5, Scalar(255, 0, 0)); } } } } imshow("kesisim noktalari", src); mesafeBul(); } </pre>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 37
YAPILAN İŞ	Kesişim Noktası Eliminasyonu	TARİH :25.07.19
1.23 KESİŞİM NOKTASI ELİMİNASYONU <p>Kesişim noktalarını bulduğumuzda ve görüntü üzerinde gösterdiğimizde hatalı noktalar olduğunu gördük. Bu noktaların eliminasyon işlemi için geliştireceğimiz algoritma aşağıdaki gibi ilerliyor:</p> <p>Önce kesişim noktalarını alıp aralarındaki mesafeleri ölçeceğiz. Devamında noktalar arasındaki mesafeler görüntünün genişliğini geçmezse (yani resmimizin içerisindeyse) bu mesafeyi yeni bir diziye ekleyeceğiz. Sonra mesafeler arasında yoğunlukta olan mesafeyi bulup eşik değeri olarak belirliyoruz. Eğer bir noktanın yoğunlukta olan mesafe kadar uzağında iki ve daha fazla komşusu varsa o nokta elimine edilmeyecek. Bu koşulu sağlamayanlar ise elimine edilecek.</p> <pre> void mesafeBul() { int kesisimNoktasiSayisi = boyutBulKesisimler(kesisimlerBaslik); //linked listden nokta saısını bulduk int treshold; //eşik değeri belirlemek için değişken oluşturuyoruz. // her konumun x ve y diye 2 verisi bulunmakta, bu yüzden nokta sayısını ikiye bölüyoruz. int* trueOrFalse = new int[kesisimNoktasiSayisi/2]; // noktanın elimine edilip edilmeyeceğine karar verdikten konumunu atayacağımız dizi int** distance = new int*[kesisimNoktasiSayisi/2]; //noktalar arasındaki mesafeleri atayacağımız 2 boyutlu liste oluşturuyoruz for (int i = 0; i < kesisimNoktasiSayisi/2; i++) { // dinamik diziyi 2 boyutlu yapıyoruz. distance[i] = new int[kesisimNoktasiSayisi/2]; } int width = src.size().width; //goruntunu width'i for (int i = 0; i < kesisimNoktasiSayisi/2; i++) { for (int j = 0; j < kesisimNoktasiSayisi/2; j++) { if (i != j) { //mesafe fonksiyonunu uyguluyoruz ve bulduğumuz sonucu dizide gerekli konuma setliyoruz. distance[i][j] = sqrt(pow(kesisimGetir(kesisimlerBaslik, i * 2) - kesisimGetir(kesisimlerBaslik, j * 2), 2) + pow(kesisimGetir(kesisimlerBaslik, i * 2 + 1) - kesisimGetir(kesisimlerBaslik, j * 2 + 1), 2)); } else { // i j'ye eşitse noktanın kendisidir demektir distance[i][j] = -1; } } } int * d = new int[kesisimNoktasiSayisi/2]; //resimden taşan noktaları elemek için oluşturduğumuz dizi for (int i = 0; i < kesisimNoktasiSayisi/2; i++) { d[i] = width; for (int j = 0; j < kesisimNoktasiSayisi/2; j++) { if (i != j) { if (d[i] > distance[i][j]) { // eğer noktalar arasındaki mesafe resmin genişliğinden küçükse d[i] = distance[i][j]; } } } } } </pre>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 38
YAPILAN İŞ	Kesişim Noktası Eliminasyonu	TARİH :25.07.19
<pre> int maksimumDeger = enCokOlusanSayi(d, (kesisimNoktasiSayisi/2)); //noktalar arası mesafelerde en çok oluşan mesafeyi belirliyoruz. for (int i = 0; i < kesisimNoktasiSayisi/2; i++) { treshold = 0; for (int j = 0; j < kesisimNoktasiSayisi/2; j++) { if (i != j) { if ((distance[i][j] <= (maksimumDeger+1)) && (distance[i][j] >= (maksimumDeger-1))) { treshold++; } if (treshold >= 2) { //yoğunlukta olan mesafe kadar uzaklıkta 2 ve daha fazla komşusu varsa bu nokta kalsın trueOrFalse[i] = true; } else { //if koşulunu sağlamadığı için elimine et trueOrFalse[i] = false; } } } } //elimine edilmiş noktaları kırmızı daire içerisine alıyoruz //elimine edilmemiş noktaları yeşil daire içerisine alıyoruz for (int i = 0; i < kesisimNoktasiSayisi/2; i++) { if (trueOrFalse[i] == 1) { circle(src, Point(kesisimGetir(kesisimlerBaslik, i * 2), kesisimGetir(kesisimlerBaslik, i * 2+1)), 5, Scalar(0, 255, 0)); // elimine sonrasi ideal kesisim noktasi Yeşil renk } else { circle(src, Point(kesisimGetir(kesisimlerBaslik, i * 2), kesisimGetir(kesisimlerBaslik, i * 2+1)), 5, Scalar(0, 0, 255)); // elimine sonrasi ideal olmayan Kirmizi } } delete[] d; delete[] distance; delete[] trueOrFalse; imshow("elimine edilmis", src); } </pre>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 39
YAPILAN İŞ	Kesişim Noktası Eliminasyonu	TARİH :25.07.19

Sonuç olarak elde edeceğimiz çıktı aşağıdaki gibidir:



Yukarıda da görüldüğü gibi hatalı kesişim noktalarının çoğu elimine edildi. Elbette biraz hata payı vardır.

KONTROL SONUCU

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 40
YAPILAN İŞ	Otomatik Kalibrasyon İçin Yüklenen Görüntüyü Otomatik Okuma	TARİH :26.07.19

1.24 OTOMATİK KALİBRASYON İÇİN YÜKLENEN GÖRÜNTÜYÜ OTOMATİK OKUMA

Yapılacak olan işlemler sırasıyla şöyle:

İlk olarak Open dediğimizde açılan görüntüyü kaydetmek için projenin StartUpPath'ini alıyoruz. Bunun için Open butonunun event fonksiyonuna aşağıdaki kodu yazıyoruz:

```
CString cikti = Application::StartupPath + "/goruntu.bmp";
LPCTSTR output = (LPCTSTR)cikti;
```

Devamında SaveBMP() fonksiyonunu oluşturuyoruz. Fonksiyonlar dosyasının header'ına aşağıdaki başlığı yazıyoruz:

```
bool SaveBMP(BYTE* Buffer, int width, int height, long paddedsize, LPCTSTR bmpfile);
```

.cpp dosyasına ise aşağıdaki kodu yazıyoruz:

```
bool SaveBMP(BYTE* Buffer, int width, int height, long paddedsize, LPCTSTR bmpfile)
{
    // declare bmp structures
    BITMAPFILEHEADER bmfh;
    BITMAPINFOHEADER info;

    // and initialize them to zero
    memset(&bmfh, 0, sizeof(BITMAPFILEHEADER));
    memset(&info, 0, sizeof(BITMAPINFOHEADER));

    // fill the fileheader with data
    bmfh.bfType = 0x4d42;          // 0x4d42 = 'BM'
    bmfh.bfReserved1 = 0;
    bmfh.bfReserved2 = 0;
    bmfh.bfSize = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER) + paddedsize;
    bmfh.bfOffBits = 0x36;        // number of bytes to start of bitmap bits

    // fill the infoheader

    info.biSize = sizeof(BITMAPINFOHEADER);
    info.biWidth = width;
    info.biHeight = height;
    info.biPlanes = 1;            // we only have one bitplane
    info.biBitCount = 24;         // RGB mode is 24 bits
    info.biCompression = BI_RGB;
    info.biSizeImage = 0;         // can be 0 for 24 bit images
    info.biXPelsPerMeter = 0x0ec4; // paint and PSP use this values
    info.biYPelsPerMeter = 0x0ec4;
    info.biClrUsed = 0;           // we are in RGB mode and have no palette
    info.biClrImportant = 0;     // all colors are important

    // now we open the file to write to
    HANDLE file = CreateFile(bmpfile, GENERIC_WRITE, FILE_SHARE_READ,
        NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    if (file == NULL) {
        CloseHandle(file);
        return false;
    }
    // write file header
    unsigned long bwritten;
    if (WriteFile(file, &bmfh, sizeof(BITMAPFILEHEADER), &bwritten, NULL) == false) {
        CloseHandle(file);
        return false;
    }
}
```


KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 41
YAPILAN İŞ	Otomatik Kalibrasyon İçin Yüklenen Görüntüyü Otomatik Okuma	TARİH :26.07.19

```
// write infoheader
if (WriteFile(file, &info, sizeof(BITMAPINFOHEADER), &bwritten, NULL) == false) {
    CloseHandle(file);
    return false;
}
// write image data
if (WriteFile(file, Buffer, paddedsize, &bwritten, NULL) == false) {
    CloseHandle(file);
    return false;
}

// and close file
CloseHandle(file);

return true;
} // SaveBMP
```

Sonra fonksiyonu Open butonunun event fonksiyonunda LPCTSTR output = (LPCTSTR)cikti; satırından hemen sonra aşağıdaki gibi çağırıyoruz:

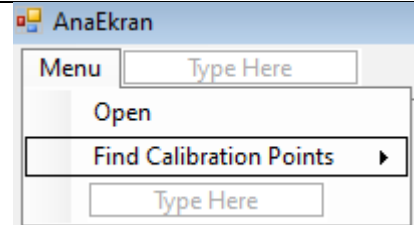
```
SaveBMP(buffer, width, height, size1, output);
```

Kaydettiği konum .sln dosyasının olduğu klasörde \x64 klasörünün altında /Debug kalörüdür.

Imread() fonksiyonunun otomatik olarak okuması için globalde sadece src matrisini tanımlıyoruz ve geri kalan tüm tanımlamaları siliyoruz.

```
Mat src = imread("../x64/Debug/goruntu.bmp"); // üzerinde kesisim noktalarini gösterecegimiz goruntu
```

Ayrıca DogruBul() fonksiyonunu Open butonunun event fonksiyonunda çağırıyorduk. Şimdi ise Design kısmında Open butonun altında aşağıdaki gibi buton oluşturuyoruz ve DogruBul() fonksiyonunu oradan çağırıyoruz.

Design kısmı:	Formun header'ında oluşan event
	<pre>private: System::Void findCalibrationPointsToolStripMenuItem_Click(System::Object^ sender, System::EventArgs^ e) { DogruBul(); }</pre>

Artık projemize eklenen görüntüler de dinamik bir şekilde alınmakta.

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 42
YAPILAN İŞ	Gerçek Zamanlı Mono Kalibrasyon(Teori ve Uygulanması için Detaylar)	TARİH :29.07.19

1.25 GERÇEK ZAMANLI MONO KALİBRASYON (TEORİ VE UYGULANMASI İÇİN DETAYLAR)

Şimdiye kadar yaptığımız aşamaların hepsi yüklenen fotoğraf üzerinden ilerliyordu. Fakat şimdi gerçek zamanlı olarak kalibrasyon işlevini gerçekleştireceğiz. Yazdığımız ve şimdiye kadar uyguladığımız fonksiyonların hepsi aynı kalacak. Sadece küçük değişimler sonrasında gerçek zamanlı olarak kalibrasyon işlevini yapacağız.

Teorik olarak bu yapının tek farkı her saniye bize yeni bir fotoğraf gelmesi ve bu işlevlerin hepsini her saniye tekrarlamak. Bunu yapmak çok da basit değil. Optimizasyon ve bellek sorunları oluşacaktır. Ayrıca kalibre edeceğimiz görüntüyü yakalayıp kalibrasyon noktalarımızı da elde etmeliyiz. Bu kalibrasyon noktaları kesişim testi sonucunda ortaya çıkan noktalar kümesidir. Anlayacağınız üzere en son oluşan noktaların verisini de yakalamalıyız.

İşlem aşamalarımız sırasıyla optimize bir yapı oluşturmak, yüklenen fotoğrafa uygulanan adımların aynısını uygulamak(sorunsuz ve hızlı bir şekilde), dinamik ve kullanıcıya yönelik bir ürün için kullanıcı gereksinimlerini ve olası sorunları tespit edip gerekli geliştirmeler sonucu arayüzde ve arkaplanda değişimler yapmak, ölçüm işlemlerinin dinamik olarak geliştirilmesi, yeni ölçüm işlemleri uygulanması ve son olarak uygulamanın testi ve kullanımı.

KONTROL SONUCU

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 43
YAPILAN İŞ	Gerçek Zamanlı Mono Kalibrasyonda Kalibrasyon Noktalarının Otomatikleştirilmesi	TARİH :30.07.19

1.26 GERÇEK ZAMANLI MONO KALİBRASYONDA KALİBRASYON NOKTALARININ OTOMATİKLEŞTİRİLMESİ

Daha önce yazdığımız fonksiyonlarda otomatikleştirme için doğrunun başlangıç ve bitiş konumlarının verisini tutan, doğruların kesişim noktalarının konum verisini tutan linked listler yazmış ve kullanmıştık. Manuel Kalibrasyondan da hatırlayacağınız gibi görüntü üzerinde seçilen kalibrasyon noktalarını ve bunlara uygun gelen gerçek dünya konumlarını dosyaya yazıyorduk. Fakat şimdi gerçek zamanlı mono kalibrasyonda sürekli olarak dosyaya yazmak işimizi çok karmaşıktırıyor. Zira her frame’de dosyaya yazıp sonra dosyayı temizleyip diğer frame için kalibrasyon noktalarını dosyaya yazmak beraberinde çok fazla iş yükü getirir. Bu sorunu çözmek adına linked list yapısını kullanacağız. Ama sadece en son yakaladığımız frame için dosyaya yazma işlevini gerçekleştireceğiz. Kalibrasyon noktaları için linked list kodları aşağıdaki gibidir.

İmge için kalibrasyon noktalarının tutulduğu linked list kodları.

```

class IKalibrasyon {
public:
    int veri;
    IKalibrasyon* sonraki;
};

void iKalibrasyonEkle(IKalibrasyon** baslikReferans, int veri)
{
    IKalibrasyon* yeniKesisim = new IKalibrasyon();// yeni dugum olustur
    yeniKesisim->veri = veri;// veriyi koy
    yeniKesisim->sonraki = (*baslikReferans);//eski listeyi dugumden ayir
    (*baslikReferans) = yeniKesisim;//yeni dugume isaret etmesi icin basligi tasi
}

int iKalibrasyonGetir(IKalibrasyon* baslik, int indis) { //Bağlantılı listenin başlık göstergesini
ve aranan indisi alır ve argüman olarak girilen indisdeki veriyi döndürür
    IKalibrasyon* simdiki = baslik;

    int sayac = 0;
    /*pointer bizim girdiğimiz indisi gösterene kadar sayac arttırılır.*/
    while (simdiki != NULL) {
        if (sayac == indis) {
            return(simdiki->veri);
        }
        sayac++;
        simdiki = simdiki->sonraki;
    }
}

int boyutBulIKalibrasyonNoktaları(IKalibrasyon* baslik) { //Bağlantılı listenin başlık
göstergesini alır ve başlıkta kaç tane düğüm olduğunu döndürür
    IKalibrasyon* simdiki = baslik;

    int sayac = 0;
    while (simdiki != NULL) { //liste sonuna gelene kadar şimdiki düğümü gösteren pointer bir
sonrakine işaret eder.
        sayac++;
        simdiki = simdiki->sonraki;
    }

    return sayac;
}

```

KONTROL SONUCU

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 44
YAPILAN İŞ	Gerçek Zamanlı Mono Kalibrasyonda Kalibrasyon Noktalarının Otomatikleştirilmesi	TARİH :30.07.19

Gerçek Dünya için kalibrasyon noktalarının tutulduğu linked list kodları.

```

class DKalibrasyon {
public:
    int veri;
    DKalibrasyon * sonraki;
};

void dKalibrasyonEkle(DKalibrasyon ** baslikReferans, int veri)
{
    DKalibrasyon * yeniKesisim = new DKalibrasyon();// yeni dugum olustur
    yeniKesisim->veri = veri;// veriyi koy
    yeniKesisim->sonraki = (*baslikReferans);//eski listeyi dugumden ayir
    (*baslikReferans) = yeniKesisim;//yeni dugume isaret etmesi icin basligi tasi
}

int dKalibrasyonGetir(DKalibrasyon * baslik, int indis) { //Bağlantılı listenin başlık göstergesini
ve aranan indisi alır ve argüman olarak girilen indisdeki veriyi döndürür
    DKalibrasyon * simdiki = baslik;

    int sayac = 0;
    /*pointer bizim girdiğimiz indisi gösterene kadar sayac arttırılır.*/
    while (simdiki != NULL) {
        if (sayac == indis) {
            return(simdiki->veri);
        }
        sayac++;
        simdiki = simdiki->sonraki;
    }
}

int boyutBulDKalibrasyonNoktaları(DKalibrasyon * baslik) { //Bağlantılı listenin başlık
göstergesini alır ve başlıkta kaç tane düğüm olduğunu döndürür
    DKalibrasyon* simdiki = baslik;

    int sayac = 0;
    while (simdiki != NULL) { //liste sonuna gelene kadar şimdiki düğümü gösteren pointer bir
sonrakine işaret eder.
        sayac++;
        simdiki = simdiki->sonraki;
    }

    return sayac;
}

```

Son olarak aşağıdaki gibi başlık işaretçilerini de tanımlıyoruz.

```

IKalibrasyon* IKalibrasyonBaslik = NULL;
DKalibrasyon* DKalibrasyonBaslik = NULL;

```

KONTROL SONUCU

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 45
YAPILAN İŞ	Gerçek Zamanlı Mono Kalibrasyon Uygulanması(1/2)	TARİH :31.07.19
<p>1.27 GERÇEK ZAMANLI MONO KALİBRASYON UYGULANMASI (1/2)</p> <p>Şimdi ise gerçek zamanlı olarak görüntü elde edebiliriz. Bunun için OpenCV'nin VideoCapture fonksiyonunun dan yararlanacağız. .cpp dosyamızın da globalde</p> <p>VideoCapture vCap(0);</p> <p>tanımlamasını yapıyoruz. 0(sıfır) diye belirttiğimiz şey görüntü alacağımız bilgisayarın kamerasının işaretçisi. Eğer bilgisayara bağlı ek bir kamera varsa bunun işaretçisi 1(bir) olacak. Daha sonra fonksiyonların içerisinde tanımladığımız değişkenlerin çoğunu silip globalde tanımlıyoruz.</p> <p>Globalde tanımlanacak değişkenler</p> <pre> const char* imgeYolu1 = "calibration_image_points.txt"; const char* dünyaYolu1 = "calibration_world_points.txt"; int iKalibrasyonNoktaSayisi = 0; Mat src; Mat dst, cdst; // 2 tane matris olusturuyoruz, uygulayacagimiz fonksiyonlari sonucunu yazdirmek icin Mat gray; vector<Vec2f> lines;//Hough uygulanmasi sonucu cizgileri yazdiracagimiz bir vektor turunden degisken tanimliyoruz. Point pt1, pt2; // PT1 ve PT2 her bir cizginin baslangic ve bitis konumu: Point turunden olduklari icin herbirinin x ve y degerleri var. Point A, B, C, D;// 2 dogru var ve bu 2 dogrunun 2 noktasi var(baslangic ve bitis). A ve B ilk 1. dogru icin noktalar olsun, C ve D 2. dogru icin noktalar float rho; float theta; double a, b, x0; int noktaSayisi; int x, y; // kesisim noktalarinin x ve y noktalarini dosyaya yazdirmek icin x ve y degiskenlerini tanimliyoruz. double a1; double b1; double c1; double a2; double b2; double c2; int treshold; //eşik değeri belirlemek için değişken oluşturuyoruz. int kesisimNoktasiSayisi; int kalibreX, kalibreY, kalibreZ = 0; fstream imgeDosya; fstream dünyaDosya; int* trueOrFalse; </pre>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 46
YAPILAN İŞ	Gerçek Zamanlı Mono Kalibrasyon Uygulanması(2/2)	TARİH : 01.08.19
1.28 GERÇEK ZAMANLI MONO KAİBRASYON UYGULANMASI (2/2)		
<p><code>int Kamera(int tahtaBoyutu , float* maksimumDeger);</code> diye yeni bir fonksiyon oluşturuyoruz. Bu fonksiyon daha önce yazdığımız fonksiyonların tamamını içerecek. Yapı tam olarak aşağıdaki gibi olacak.</p>		
<p>Kamera() fonksiyonu:</p> <pre> int Kamera(int tahtaBoyutu , float* maksimumDeger) { imgeDosya.open(imgeYolu1); dunyaDosya.open(dunyaYolu1); for (;;) { vCap >> src; register int width = src.size().width; //goruntunu width'i register int height = src.size().height; //goruntunun height' cvtColor(src, gray, COLOR_BGR2GRAY); Canny(gray, dst, 50, 200, 3); // cizgileri bulabilmek için önce Canny uyguluyoruz. cvtColor(dst, cdst, COLOR_GRAY2BGR); //GRAY seviyeden BGR seviyesine donusturuyoruz //Vec2f anlami 2 tane float degiskene sahip olmasi ve bu vektor degiskeninin bir struct oldugunu gosteriyor. HoughLines(dst, lines, 1, CV_PI / 180, 150, 0, 0); // Hough uygulayarak fotograftaki dogrulari(cizgileri) buluyoruz. /*Hough sonucunda buldugumuz dogrulari fotografa cizdirecegiz*/ for (size_t i = 0; i < lines.size(); i++) // buldugumuz cizgilerin sayisi kadar gidiyoruz. { rho = lines[i][0]; // RHO hesaplamalarda kullanilacak piksel olarak cozunurluk theta = lines[i][1]; // THETA hesaplamalarda kullanilacak radyan turunden aci /*her bir cizgiyi(dogruyu) cizdirebilmek için hesaplamalar asagidaki gibidir.*/ a = cos(theta); b = sin(theta); x0 = a * rho; register double y0 = b * rho; pt1.x = cvRound(x0 + 1000 * (-b)); pt1.y = cvRound(y0 + 1000 * (a)); pt2.x = cvRound(x0 - 1000 * (-b)); pt2.y = cvRound(y0 - 1000 * (a)); /*Hesaplamalarin sonu*/ //pt1.x -> pt1.y -> pt2.x -> pt2.y -> NULL noktaEkle(&noktalarBaslik, pt2.y); noktaEkle(&noktalarBaslik, pt2.x); noktaEkle(&noktalarBaslik, pt1.y); noktaEkle(&noktalarBaslik, pt1.x); line(cdst, pt1, pt2, Scalar(0, 0, 255), 1); // line() fonksiyonu ile goruntunun uzerine hesaplamalar sonucu buldugumuz noktalarin arasinda olan dogruyu cizdiriyoruz } noktaSayisi = boyutBulNoktalar(noktalarBaslik); </pre>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 47
YAPILAN İŞ	Gerçek Zamanlı Mono Kalibrasyon Uygulanması(2/2)	TARİH : 01.08.19
<pre> /*kesisim testini uyguluyoruz*/ for (register int i = 0; i < noktaSayisi / 4; i++) { /*ilk dogru icin baslangic ve bitis konumlarini A ve B noktalarina tanimliyoruz*/ A.x = noktaGetir(noktalarBaslik, (i * 4)); A.y = noktaGetir(noktalarBaslik, (i * 4 + 1)); B.x = noktaGetir(noktalarBaslik, (i * 4 + 2)); B.y = noktaGetir(noktalarBaslik, (i * 4 + 3)); for (register int j = i + 1; j < noktaSayisi / 4; j++) { /*geriye cizgiler icin baslangic ve bitis konumlarini C ve D noktalarina tanimliyoruz */ C.x = noktaGetir(noktalarBaslik, (j * 4)); C.y = noktaGetir(noktalarBaslik, (j * 4 + 1)); D.x = noktaGetir(noktalarBaslik, (j * 4 + 2)); D.y = noktaGetir(noktalarBaslik, (j * 4 + 3)); /*determinan bulabilmek icin a1,b1,c1 ve a2,b2,c2 hesaplaniyor*/ a1 = B.y - A.y; b1 = A.x - B.x; c1 = a1 * (A.x) + b1 * (A.y); a2 = D.y - C.y; b2 = C.x - D.x; c2 = a2 * (C.x) + b2 * (C.y); //determinant hesaplaniyor register double determinant = a1 * b2 - a2 * b1; if (determinant == 0) { //hicbirsey yapma, cunku paraleller } else { x = (b2*c1 - b1 * c2) / determinant; y = (a1*c2 - a2 * c1) / determinant; if (x<0 x > width y < 0 y > height) { // eger kesisim noktasi görüntüden tasmissa //hicbirsey yapma, cunku araliga dahil degil } else { //x -> y -> NULL kesisimEkle(&kesisimlerBaslik, y); kesisimEkle(&kesisimlerBaslik, x); } } } } } </pre>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 48
YAPILAN İŞ	Gerçek Zamanlı Mono Kalibrasyon Uygulanması(2/2)	TARİH : 01.08.19
<pre> kesisimNoktasiSayisi = boyutBulKesisimler(kesisimlerBaslik); //linked listden nokta saısını bulduk // her konumun x ve y diye 2 verisi bulunmakta, bu yüzden nokta sayısını ikiye bölüyoruz. trueOrFalse = new int[kesisimNoktasiSayisi / 2]; // noktanın elimine edilip edilmeyeceğine karar verdikten konumunu atayacağımız dizi register int** distance = new int*[kesisimNoktasiSayisi / 2]; //noktalar arasındaki mesafeleri atayacağımız 2 boyutlu liste oluşturuyoruz for (register int i = 0; i < kesisimNoktasiSayisi / 2; i++) { // dinamik diziyi 2 boyutlu yapıyoruz. distance[i] = new int[kesisimNoktasiSayisi / 2]; } // int width = src.size().width; //goruntunu width'i for (register int i = 0; i < kesisimNoktasiSayisi / 2; i++) { for (register int j = 0; j < kesisimNoktasiSayisi / 2; j++) { if (i != j) { //mesafe fonksiyonunu uyguluyoruz ve bulduğumuz sonucu dizide gerekli konuma setliyoruz. distance[i][j] = sqrt(pow(kesisimGetir(kesisimlerBaslik, i * 2) - kesisimGetir(kesisimlerBaslik, j * 2), 2) + pow(kesisimGetir(kesisimlerBaslik, i * 2 + 1) - kesisimGetir(kesisimlerBaslik, j * 2 + 1), 2)); } else { // i j'ye eşitse noktanın kendisidir demektir distance[i][j] = -1; } } } register int * d = new int[kesisimNoktasiSayisi / 2]; //resimden taşan noktaları elemek için oluşturduğumuz dizi for (register int i = 0; i < kesisimNoktasiSayisi / 2; i++) { d[i] = width; for (register int j = 0; j < kesisimNoktasiSayisi / 2; j++) { if (i != j) { if (d[i] > distance[i][j]) { // eğer noktalar arasındaki mesafe resmin genişliğinden küçükse d[i] = distance[i][j]; } } } } maksimumDeger[0] = enCokOlusanSayi(d, (kesisimNoktasiSayisi / 2)); //noktalar arası mesafelerde en çok oluşan mesafeyi belirliyoruz. for (register int i = 0; i < kesisimNoktasiSayisi / 2; i++) { treshold = 0; for (register int j = 0; j < kesisimNoktasiSayisi / 2; j++) { if (i != j) { if ((distance[i][j] <= (maksimumDeger[0] + 1)) && (distance[i][j] >= (maksimumDeger[0] - 1))) { treshold++; } if (treshold >= 2) { //yoğunlukta olan mesafe kadar uzaklıkta 2 ve daha fazla komşusu varsa bu nokta kalsın trueOrFalse[i] = true; } else { //if koşulunu sağlamadığı için elimine et trueOrFalse[i] = false; } } } } </pre>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 49
YAPILAN İŞ	Gerçek Zamanlı Mono Kalibrasyon Uygulanması(2/2)	TARİH : 01.08.19
<pre> //elimine edilmiş noktaları kırmızı daire içerisine alıyoruz //elimine edilmemiş noktaları yeşil daire içerisine alıyoruz for (register int i = 0; i < kesisimNoktasiSayisi / 2; i++) { if (trueOrFalse[i] == 1) { circle(src, Point(kesisimGetir(kesisimlerBaslik, i * 2), kesisimGetir(kesisimlerBaslik, i * 2 + 1)), 5, Scalar(0, 255, 0)); // eliminate sonrasi ideal kesisim noktasi Yeşil renk iKalibrasyonEkle(&IKalibrasyonBaslik, kesisimGetir(kesisimlerBaslik, i * 2 + 1)); iKalibrasyonEkle(&IKalibrasyonBaslik, kesisimGetir(kesisimlerBaslik, i * 2)); kalibreX = ((kesisimGetir(kesisimlerBaslik, i * 2) / (maksimumDeger[0] + 0.001))) * tahtaBoyutu; kalibreY = ((kesisimGetir(kesisimlerBaslik, i * 2 + 1) / (maksimumDeger[0]+0.001))) * tahtaBoyutu; dKalibrasyonEkle(&DKalibrasyonBaslik, kalibreZ); dKalibrasyonEkle(&DKalibrasyonBaslik, kalibreY); dKalibrasyonEkle(&DKalibrasyonBaslik, kalibreX); } } iKalibrasyonNoktaSayisi = boyutBulIKalibrasyonNoktaları(IKalibrasyonBaslik); iKalibrasyonNoktaSayisi /= 2; imshow("elimine edilmiş", src); if (waitKey(30) > 0) { //kesisim noktalarını burada kaydet bir yere; for (int i = 0; i < iKalibrasyonNoktaSayisi; i++) { imgeDosya.clear(); dünyaDosya.clear(); imgeDosya << iKalibrasyonGetir(IKalibrasyonBaslik, i * 2) << "\t" << iKalibrasyonGetir(IKalibrasyonBaslik, i * 2 + 1) << endl; dünyaDosya << dKalibrasyonGetir(DKalibrasyonBaslik, i * 3) << "\t" << dKalibrasyonGetir(DKalibrasyonBaslik, i * 3 + 1) << "\t" << dKalibrasyonGetir(DKalibrasyonBaslik, i * 3 + 2) << endl; } destroyAllWindows(); break; } noktalarBaslik = NULL; kesisimlerBaslik = NULL; IKalibrasyonBaslik = NULL; DKalibrasyonBaslik = NULL; delete[] d; delete[] distance; delete[] trueOrFalse; } imgeDosya.close(); dünyaDosya.close(); imwrite("sonuc.bmp", src); return iKalibrasyonNoktaSayisi; } </pre>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 50
YAPILAN İŞ	Form Üzerinde Değişiklikler	TARİH : 02.08.19

1.29 FORM ÜZERİNDE DEĞİŞİKLİKLER

Uygulamanın son hali aşağıdaki gibidir

Menu strip buttonları:

Kameradan alınan veya yükleme sonucu elde edilen görüntüdeki bulunmuş noktaların sayısı ilk label’ımızda görülecektir. Kalibrasyon nokta sayısı eşik değeri dediğimiz numericUpDown nesnesi ile Kameradan alınan görüntüde kalibrasyon noktalarının eşik değerini belirliyoruz. Belirlenen değer aşıldığında o frame’i yakalayıp formda pictureBox’a yerleştirilecek. Kalibrasyon tahtası boyutu dediğimiz numericUpDown nesnesi ile ise kalibrasyon tahtamızın aralıklarının kaç millimetre olduğunu belirliyoruz. Onun hemen altındaki Label ile ise adım adım neler yapmamız gerekiyor onu görüyoruz. M/K Noktaları dediğimiz richTextBox’ta ise manuel olarak seçtiğimiz noktaların konumları gözüküyor. Bu seçimi hemen onun yanındaki Manuel Kalibrasyon radioButton’u seçili iken yapıyoruz. Kalibre Et button’u ise bulunmuş kalibrasyon noktalarını kalibre ediyor. Otomatik veya manuel olarak seçilmiş olması önemli değil her ikisini de içeriyor. Kalibrasyon işleminden sonar ise tıklanan noktaların görüntüdeki ve gerçek dünyadaki konumları Görüntü noktaları ve Gerçek dünya noktaları richTextBox’larında tutuluyor. Geriye kalan radioButton’lar ise matematiksel işlemlerde kullanılacak.

KONTROL SONUCU

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 51
YAPILAN İŞ	Matematiksel işlemler(1/4)	TARİH : 05.08.19

1.29 MATEMATİKSEL İŞLEMLER(1/4)

Matematiksel işlemler için radioButtonlar

İki nokta arasındaki mesafeyi bulmak için pictureBox'ta tıklanan iki noktanın x ve y koordinatlarını bir vektörde tutuyoruz ve daha sonra bu noktalar arasındaki mesafeyi daha önceki formülümüzle buluyoruz. Tıklama esnasında dikkat etmemiz gereken şey ilk noktanın farenin sol butonuyla son noktanın ise sağ butonuyla tıklanması gerektiği. İlk olarak pictureBox'tıklanma event'inin kodlarına bakalım.

```
private: System::Void pictureBox1_MouseClick
else if (radioButton5->Checked || radioButton6->Checked || radioButton7->Checked)
{
    Pen^ pen = gcnew Pen(Color::Blue, 2);
    pictureBox1->CreateGraphics()->DrawLine(pen, e->X - 4, e->Y - 4, e->X + 4, e->Y + 4);
    pictureBox1->CreateGraphics()->DrawLine(pen, e->X - 4, e->Y + 4, e->X + 4, e->Y - 4);
}
```

Tıklanan noktaları pictureBox üzerinde X ile işaretliyoruz. MouseUp eventinin içerisinde ise daha önce yazdığımız fonksiyonu aşağıdaki şekile güncelliyoruz.

```
private: System::Void pictureBox1_MouseUp
if (radioButton5->Checked) { // distance b/w 2 coord
    vecTiklanan.push_back(e->X);
    vecTiklanan.push_back(e->Y);
    richTextBox3->AppendText(e->X.ToString() + "-" + e->Y.ToString() + "\n");
    richTextBox3->ScrollToCaret();
    float* test1 = new float[2]; // seçilmiş x ve y noktalarını atayacağımız
    geçici dizi oluşturuluyor(sadece x ve y olduğu için boyutu 2).
    test1[0] = e->X;
    test1[1] = e->Y;
    float* gercekDunya = new float[3]; //ImarEt fonksiyonundan geri dönecek olan
    gerçek dünya dizisi oluşturuluyor.
    ImarEt(1, test1, projeksiyon, gercekDunya); // projeksiyon matrisi sonucunda
    hesaplanan gercekDunya değişkeni dolduruluyor
    richTextBox4->AppendText(gercekDunya[0] + " - " + gercekDunya[1] + "\n");
    richTextBox4->ScrollToCaret();
    vecXY.push_back(gercekDunya[0]);
    vecXY.push_back(gercekDunya[1]);

    delete[] test1;
    delete[] gercekDunya;

    switch (e->Button) // sağ buton tıklandı mı?
    {
        case System::Windows::Forms::MouseButtons::Right:
            pictureBox1->CreateGraphics()->DrawLine(pen, vecTiklanan[0],
            vecTiklanan[1], vecTiklanan[2], vecTiklanan[3]);
            vecTiklanan.clear();
            break;
    }
}
```

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 52
YAPILAN İŞ	Matematiksel işlemler(1/4)	TARİH : 05.08.19
Hesapla Button'una tıkladığımızda ise hesaplanacak fonksiyonun kodu aşağıdaki gibi güncellendi		
<pre>private: System::Void button2_Click_1 { if (radioButton5->Checked) { mesafe = sqrt(pow((vecXY[0] - vecXY[2]), 2) + pow((vecXY[1] - vecXY[3]), 2)); label10->Text = "Ölçüm sonucu : " + mesafe.ToString() + " mm"; vecXY.clear(); } }</pre>		
KONTROL SONUCU		

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 53
YAPILAN İŞ	Matematiksel işlemler(2/4)	TARİH : 06.08.19

1.30 MATEMATİKSEL İŞLEMLER(2/4)

Eğrinin uzunluğunu bulmak için MouseClick eventimiz diğerleriyle ortak olduğu için ona deyinmiyorum. Fakat MouseUp eventindeki güncellenen kısım aşağıdaki gibidir

```
private: System::Void pictureBox1_MouseUp
{
    else if (radioButton6->Checked) {

        richTextBox3->AppendText(e->X.ToString() + "-" + e->Y.ToString() + "\n");
        richTextBox3->ScrollToCaret();
        vecTiklanan.push_back(e->X);
        vecTiklanan.push_back(e->Y);
        float* test1 = new float[2]; // seçilmiş x ve y noktalarını atayacağımız
        geçici dizi oluşturuluyor(sadece x ve y olduğu için boyutu 2).
        test1[0] = e->X;
        test1[1] = e->Y;
        float* gercekDunya = new float[3]; //ImarEt fonksiyonundan geri dönecek olan
        gercek dünya dizisi oluşturuluyor.
        ImarEt(1, test1, projeksiyon, gercekDunya); // projeksiyon matrisi sonucunda
        hesaplanan gercekDunya değişkeni dolduruluyor

        richTextBox4->AppendText(gercekDunya[0] + " - " + gercekDunya[1] + "\n");
        richTextBox4->ScrollToCaret();
        vecXY.push_back(gercekDunya[0]);
        vecXY.push_back(gercekDunya[1]);

        delete[] test1;
        delete[] gercekDunya;

        switch (e->Button)
        {
            case System::Windows::Forms::MouseButtons::Right:
                for (int i = 0; i < (vecTiklanan.size() / 2) - 1; i++) {
                    pictureBox1->CreateGraphics()->DrawLine(pen, vecTiklanan[2 * i], vecTiklanan[2 * i + 1], vecTiklanan[2 * i + 2], vecTiklanan[2 * i + 3]);
                }
                vecTiklanan.clear();
                break;
        }
    }
}
```

Hesapla Button'una tıkladığımızda ise hesaplanacak fonksiyonun kodu aşağıdaki gibi güncellendi

```
private: System::Void button2_Click_1
{
    else if (radioButton6 ->Checked)
    {
        mesafe = 0;
        for (int i = 0; i < (vecXY.size()/2) - 1; i++) {
            mesafe += sqrt(pow(vecXY[2 * i + 2] - vecXY[2 * i], 2) + pow(vecXY[2 * i + 3] - vecXY[2 * i + 1], 2));
        }
        label10->Text = "Ölçüm sonucu : " + mesafe.ToString() + " mm";
        vecXY.clear();
    }
}
```

KONTROL SONUCU

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 54
YAPILAN İŞ	Matematiksel işlemler(3/4)	TARİH : 07.08.19

1.31 MATEMATİKSEL İŞLEMLER(3/4)

Çevrelenmiş nesnenin alanını bulmak için MouseClick eventimiz diğerleriyle ortak olduğu için ona deyinmiyorum.

Fakat MouseUp eventindeki güncellenen kısım aşağıdaki gibidir

```
private: System::Void pictureBox1_MouseUp
else if (radioButton7->Checked) {

    richTextBox3->AppendText(e->X.ToString() + "-" + e->Y.ToString() + "\n");
    richTextBox3->ScrollToCaret();
    vecTiklanan.push_back(e->X);
    vecTiklanan.push_back(e->Y);
    float* test1 = new float[2]; // seçilmiş x ve y noktalarını atayacağımız
    geçici dizi oluşturuluyor(sadece x ve y olduğu için boyutu 2).
    test1[0] = e->X;
    test1[1] = e->Y;
    float* gercekDunya = new float[3]; //ImarEt fonksiyonundan geri dönecek olan
    gerçek dünya dizisi oluşturuluyor.
    ImarEt(1, test1, projeksiyon, gercekDunya); // projeksiyon matrisi sonucunda
    hesaplanan gercekDunya değişkeni dolduruluyor

    richTextBox4->AppendText(gercekDunya[0] + " - " + gercekDunya[1] + "\n");
    richTextBox4->ScrollToCaret();
    vecXY.push_back(gercekDunya[0]);
    vecXY.push_back(gercekDunya[1]);

    delete[] test1;
    delete[] gercekDunya;

    switch (e->Button)
    {
    case System::Windows::Forms::MouseButtons::Right:
        for (int i = 0; i < (vecTiklanan.size() / 2) - 1; i++) {
            pictureBox1->CreateGraphics()->DrawLine(pen, vecTiklanan[2 *
i], vecTiklanan[2 * i + 1], vecTiklanan[2 * i + 2], vecTiklanan[2 * i + 3]);
        }
        int vecSize = vecTiklanan.size();
        pictureBox1->CreateGraphics()->DrawLine(pen, vecTiklanan[0],
vecTiklanan[1], vecTiklanan[vecTiklanan.size() - 2], vecTiklanan[vecTiklanan.size() - 1]);
        vecTiklanan.clear();
        break;
    }
}
```

Hesapla button'una tıkladığımızda bulunacak olan alan fonksiyonu ise aşağıdaki gibidir.

```
private: System::Void button2_Click_1
else if (radioButton7->Checked)
{
    alan = alanBul();
    label10->Text = "Ölçüm sonucu : " + alan.ToString() + " mm^2";
    vecXY.clear();
}
```

Burada yazılacak olan alanBul() fonksiyonu bi sonraki kısımdadır.

KONTROL SONUCU

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 55
YAPILAN İŞ	Matematiksel işlemler(4/4)	TARİH : 08.08.19

1.32 MATEMATİKSEL İŞLEMLER(4/4)

alanBul() fonksiyonu:

```
int alanBul() {
    int enBuyukX = 0;
    int enBuyukY = 0;
    int enKucukX = 100000;
    int enKucukY = 100000;
    int sayici = 0;
    for (int i = 0; i < vektorImge.size() / 2; i++) {
        if (enBuyukX < vektorImge[2 * i]) {
            enBuyukX = vektorImge[2 * i];
        }
        if (enBuyukY < vektorImge[2 * i + 1]) {
            enBuyukY = vektorImge[2 * i + 1];
        }
        if (enKucukX > vektorImge[2 * i]) {
            enKucukX = vektorImge[2 * i];
        }
        if (enKucukY > vektorImge[2 * i + 1]) {
            enKucukY = vektorImge[2 * i + 1];
        }
    }

    for (int i = enKucukX; i < enBuyukY; i++) {
        for (int j = enKucukY; j < enBuyukY; j++) {
            sayici++;
        }
    }

    return sayici;
}
```

Bu fonksiyon aracılığıyla seçilmiş olan alan içerisinde kalan pikselleri sayarak alanı buluyoruz.

KONTROL SONUCU

KISIM	Mono Kamera Kalibrasyon	YAPRAK NO : 56
YAPILAN İŞ	Uygulamanın test aşaması ve kullanımı	TARİH : 09.08.19

1.33 UYGULAMANIN TEST AŞAMASI VE KULLANIMI

Çevrimiçi kullanım:

Öncelikle Kalibrasyon nokta sayısı eşik değerini ve tahta boyutunu belirliyoruz. Sonra kamerayı aç butonuna tıklıyoruz ve görüntüyü elde ediyoruz. Eğer istiyorsak manuel olarak nokta ekleyebiliriz. Sonrasında kalibre et diyoruz ve kalibrasyon işlemlerini tamamlıyoruz. Şimdi matematiksel işlemleri gerçekleştirebiliriz.

Çevrimdışı kullanım:

Öncelikle Kalibrasyon tahta boyutunu belirliyoruz. Sonra görüntü yükle diyoruz. Devamında kalibrasyon noktalarını bul butonuna tıklayıp noktaları buluyoruz. İstersek manuel nokta da ekleyebiliriz. Kalibre et diyoruz ve kalibrasyon işlemlerini tamamlıyoruz. Şimdi matematiksel işlemleri gerçekleştirebiliriz.