ภาคผนวก ข
โปรแกรมประมวลผลภาพ

โปรแกรมประมวลผลภาพ

```
// ไลบารี่พื้นฐาน
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Microsoft.Kinect;      // ไลบารี่สำหรับกล้อง Kinect
using System.Drawing.Imaging;
using System.Runtime.InteropServices;
using System.Threading;
// ไลบารี่สำหรับการประมวลผมภาพ
using Emgu.CV;
using Emgu.CV.CvEnum;
using Emgu.CV.Structure;
using Emgu.Util;
// ไลบารี่สำหรับอินเตอร์เน็ต
using System.Net;
using System.Net.Sockets;
namespace WindowsFormsApplication5
{
    public partial class Form1 : Form
    {
        Point[] position = new Point[3];    // ตัวแปลใช้เก็บตำแหน่งในการเช็ค
        Point[] area = new Point[3];    // ใช้เก็บความกว้างของการเคลื่อนที่
        public Form1()
        {
            InitializeComponent();
```

```
            // การหาตำแหน่งของวัตถุ
            _Tool.H = new Point(20, 35); _Tool.S = new Point(140, 255); _Tool.V = new
Point(130, 255); _Tool.invert = false;
            _objectA.H = new Point(40, 53); _objectA.S = new Point(140, 230); _objectA.V
= new Point(160, 235); _objectA.invert = false;
            _objectB.H = new Point(0, 155); _objectB.S = new Point(180, 255); _objectB.V
= new Point(120, 200); _objectB.invert = true;
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // หา IP ของเครื่องคอมพิวเตอร์
            textBoxIP.Text = GetIP();
            //หาไดร์เวอร์ของกล้อง Kinect Sensor
            foreach (var potentialSensor in KinectSensor.KinectSensors)
            {
                if (potentialSensor.Status == KinectStatus.Connected)
                {
                    this.sensor = potentialSensor;
                    break;
                }
            }
            if (null != this.sensor)    // ถ้าเจอแล้วทำการตั้งค่า
            {
                // ตั้งค่าสีและความละเอียดของภาพ

this.sensor.ColorStream.Enable(ColorImageFormat.YuvResolution640x480Fps15);
                // จัดพื้นที่สำหรับเก็บพิกเซลภาพ
                this.colorPixels = new
byte[this.sensor.ColorStream.FramePixelDataLength];
                // สร้างเหตุการณ์ในการรับภาพใหม่
```

```
                this.sensor.ColorFrameReady += new
EventHandler<ColorImageFrameReadyEventArgs>(sensor_ColorFrameReady);
                // เริ่มจับภาพ
                try
                {
                    imageBox1.Image = ImageFrame;
                    this.sensor.Start();
                }
                catch (System.IO.IOException)
                {
                    this.sensor = null;
                }
            }
            if (null == this.sensor)    //ถ้าไม่สามารถรับภาพได้
            {
                if (MessageBox.Show("Can't Connect Kinect XBOX360", "ERROR!",
MessageBoxButtons.OK, MessageBoxIcon.Stop) ==
System.Windows.Forms.DialogResult.OK)
                {
                    this.Close();
                }
            }
        }
        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (null != this.sensor && this.sensor.IsRunning)   // ก่อนออกจากโปรแกรมให้ปิด
การทำงานของกล้อง Kinect Sensor
            {
                this.sensor.Stop();
            }
        }
```

```
        #region Image Processing
        // ตัวแปลที่ใช้ในการประมวลผลภาพ
        private KinectSensor sensor;
        private byte[] colorPixels;
        Image<Bgr, Byte> ImageFrame = new Image<Bgr, Byte>(640, 480);
        private DepthImagePixel[] depthPixels;
        struct hsvmaxmin    // ชุดข้อมูลของภาพที่รับมา
        {
            public Point H;
            public Point S;
            public Point V;
            public bool invert;
        }
        hsvmaxmin _Tool = new hsvmaxmin();
        hsvmaxmin _objectA = new hsvmaxmin();
        hsvmaxmin _objectB = new hsvmaxmin();
        byte[] colorData = null;
        Bitmap kinectVideoBitmap = null;
        IntPtr colorPtr;
        // อีเว้นจากการรับภาพใหม่เข้ามา
        void sensor_ColorFrameReady(object sender, ColorImageFrameReadyEventArgs
e)
        {
            // ทำการเปิดชุดข้อมูลที่รับมา
            using (ColorImageFrame colorFrame = e.OpenColorImageFrame())
            {
                if (colorFrame == null) return;     // ถ้าไม่มีข้อมูลให้ทำการออกจากลูป
                if (colorData == null)  //กำหนดขนาดของตัวแปลที่จะใช้เก็บภาพ
                    colorData = new byte[colorFrame.PixelDataLength];
                colorFrame.CopyPixelDataTo(colorData);  //ทำการก๊อปปี้พิกเซล
```

```
                Marshal.FreeHGlobal(colorPtr);
                colorPtr = Marshal.AllocHGlobal(colorData.Length);
                Marshal.Copy(colorData, 0, colorPtr, colorData.Length);
                kinectVideoBitmap = new Bitmap(
                    colorFrame.Width,
                    colorFrame.Height,
                    colorFrame.Width * colorFrame.BytesPerPixel,
                    PixelFormat.Format32bppRgb,
                    colorPtr);
                //kinectVideoBitmap.RotateFlip(RotateFlipType.RotateNoneFlipX);
                Image<Bgr, byte> Frame = new Image<Bgr,
byte>(kinectVideoBitmap).PyrDown();     //แปลงภาพจาก bitmap ไปเป็นไฟล์ภาพของ
emgu cv
                ImageFrame = Frame.Flip(FLIP.HORIZONTAL);       //กลับภาพในแกนตั้ง
                imageBox1.Image = ImageFrame;       //แสดงภาพ
                ImageProcessing();      //ฟังก์ชั่นในการประมวลผล
            }
            }
        private void ImageProcessing()
        {
            if (!checkBox_All.Checked)
            {//การประมวลผมลภาพครั้งเดียวค่าเดียว
                Image<Gray, Byte> ImageFrameDetection = cvAndHsvImage(
                ImageFrame,
               Convert.ToInt32(numeric_HL.Value), Convert.ToInt32(numeric_HH.Value),
               Convert.ToInt32(numeric_SL.Value), Convert.ToInt32(numeric_SH.Value),
               Convert.ToInt32(numeric_VL.Value), Convert.ToInt32(numeric_VH.Value),
               true, true, true, checkBox_IV.Checked);      //ฟังก์ชั่นในการประมวลผลภาพ
                imageBox2.Image = ImageFrameDetection;      //แสดงภาพที่ประมวลผล
```

```
            if (checkBox_VAr.Checked) RecDetection(ImageFrameDetection,
ImageFrame, trackBar_VAr.Value, new MCvScalar(0, 0, 255), 0);       //แสดงกรอบสี่เหลี่ยม
        }
        else
        {//การประมวลผลที่ละสี
            Image<Gray, Byte> ImageFrameDetection = cvAndHsvImage(
            ImageFrame,
            Convert.ToInt32(_Tool.H.X), Convert.ToInt32(_Tool.H.Y),
            Convert.ToInt32(_Tool.S.X), Convert.ToInt32(_Tool.S.Y),
            Convert.ToInt32(_Tool.V.X), Convert.ToInt32(_Tool.V.Y),
            true, true, true, _Tool.invert);
            if (checkBox_VAr.Checked) RecDetection(ImageFrameDetection,
ImageFrame, trackBar_VAr.Value, new MCvScalar(0, 255, 255), 0);
            Image<Gray, Byte> ImageFrameDetection1 = cvAndHsvImage(
                ImageFrame,
                Convert.ToInt32(_objectA.H.X), Convert.ToInt32(_objectA.H.Y),
                Convert.ToInt32(_objectA.S.X), Convert.ToInt32(_objectA.S.Y),
                Convert.ToInt32(_objectA.V.X), Convert.ToInt32(_objectA.V.Y),
                true, true, true, _objectA.invert);
            if (checkBox_VAr.Checked) RecDetection(ImageFrameDetection1,
ImageFrame, trackBar_VAr.Value, new MCvScalar(255, 0, 0), 1);
            Image<Gray, Byte> ImageFrameDetection2 = cvAndHsvImage(
                ImageFrame,
                Convert.ToInt32(_objectB.H.X), Convert.ToInt32(_objectB.H.Y),
                Convert.ToInt32(_objectB.S.X), Convert.ToInt32(_objectB.S.Y),
                Convert.ToInt32(_objectB.V.X), Convert.ToInt32(_objectB.V.Y),
                true, true, true, _objectB.invert);
            if (checkBox_VAr.Checked) RecDetection(ImageFrameDetection2,
ImageFrame, trackBar_VAr.Value, new MCvScalar(0, 0, 255), 2);
```

```
                imageBox2.Image =
(ImageFrameDetection.Or(ImageFrameDetection1).Or(ImageFrameDetection2));      //
นำภาพทั้ง 3 ภาพมา or กันเพื่อให้ได้เป็นภาพเดี๋ยวกัน
            }
        }
        private Image<Gray, Byte> cvAndHsvImage(Image<Bgr, Byte> imgFame, int L1,
int H1, int L2, int H2, int L3, int H3, bool H, bool S, bool V, bool I)   //
        {
            //ประกาศตัวแปลที่ใช้ในการแยกสีของภาพ
            Image<Hsv, Byte> hsvImage = imgFame.Convert<Hsv, Byte>();
            Image<Gray, Byte> ResultImage = new Image<Gray, Byte>(hsvImage.Width,
hsvImage.Height);
            Image<Gray, Byte> ResultImageH = new Image<Gray, Byte>(hsvImage.Width,
hsvImage.Height);
            Image<Gray, Byte> ResultImageS = new Image<Gray, Byte>(hsvImage.Width,
hsvImage.Height);
            Image<Gray, Byte> ResultImageV = new Image<Gray, Byte>(hsvImage.Width,
hsvImage.Height);
            Image<Gray, Byte> img1 = inRangeImage(hsvImage, L1, H1, 0);
            Image<Gray, Byte> img2 = inRangeImage(hsvImage, L2, H2, 1);
            Image<Gray, Byte> img3 = inRangeImage(hsvImage, L3, H3, 2);
            Image<Gray, Byte> img4 = inRangeImage(hsvImage, 0, L1, 0);
            Image<Gray, Byte> img5 = inRangeImage(hsvImage, H1, 180, 0);
            #region checkBox Color Mode
            if (H)      //ประมวลผลเฉพาะสี H
            {
                if (I)
                {
                    CvInvoke.cvOr(img4, img5, img4, System.IntPtr.Zero);
                    ResultImageH = img4;
                }
```

```
            else { ResultImageH = img1; }
        }
        if (S) ResultImageS = img2;     //ประมวลผลเฉพาะสี S
        if (V) ResultImageV = img3;     //ประมวลผลเฉพาะสี V
        if (H && !S && !V) ResultImage = ResultImageH;      //ประมวลผลเฉพาะสี H
        if (!H && S && !V) ResultImage = ResultImageS;      //ประมวลผลเฉพาะสี S
        if (!H && !S && V) ResultImage = ResultImageV;      //ประมวลผลเฉพาะสี V
        if (H && S && !V)       //ประมวลผลเฉพาะสี HS
        {
            CvInvoke.cvAnd(ResultImageH, ResultImageS, ResultImageH,
System.IntPtr.Zero);
            ResultImage = ResultImageH;
        }
        if (H && !S && V)       //ประมวลผลเฉพาะสี HV
        {
            CvInvoke.cvAnd(ResultImageH, ResultImageV, ResultImageH,
System.IntPtr.Zero);
            ResultImage = ResultImageH;
        }
        if (!H && S && V)       //ประมวลผลเฉพาะสี SV
        {
            CvInvoke.cvAnd(ResultImageS, ResultImageV, ResultImageS,
System.IntPtr.Zero);
            ResultImage = ResultImageS;
        }
        if (H && S && V)        //ประมวลผลเฉพาะสี HSV
        {
            CvInvoke.cvAnd(ResultImageH, ResultImageS, ResultImageH,
System.IntPtr.Zero);
            CvInvoke.cvAnd(ResultImageH, ResultImageV, ResultImageH,
System.IntPtr.Zero);
```

```
            ResultImage = ResultImageH;
        }
        #endregion
        CvInvoke.cvErode(ResultImage, ResultImage, (IntPtr)null, 1);
        return ResultImage;     //ส่งค่ากลับ
    }
    private Image<Gray, Byte> inRangeImage(Image<Hsv, Byte> hsvImage, int Lo, int
Hi, int con)
    {
        Image<Gray, Byte> ResultImage = new Image<Gray, Byte>(hsvImage.Width,
hsvImage.Height);     //สร้างตัวแปลเก็บภาพ
        Image<Gray, Byte> IlowCh = new Image<Gray, Byte>(hsvImage.Width,
hsvImage.Height, new Gray(Lo));    //แปลงภาพที่ค่าต่ำ
        Image<Gray, Byte> IHiCh = new Image<Gray, Byte>(hsvImage.Width,
hsvImage.Height, new Gray(Hi));     //แปลภาพที่ค่าสูง
        CvInvoke.cvInRange(hsvImage[con], IlowCh, IHiCh, ResultImage);      // นำภาพ
ให้อยู่ในช่วงที่กำหนด
        return ResultImage; //ส่งค่ากลับ
    }
    private void RecDetection(Image<Gray, Byte> img, Image<Bgr, Byte>
showRecOnImg, int areaV, MCvScalar drawcolors,int pos)
    {
        Image<Gray, Byte> imgForContour = new Image<Gray, byte>(img.Width,
img.Height);
        CvInvoke.cvCopy(img, imgForContour, System.IntPtr.Zero);
        IntPtr storage = CvInvoke.cvCreateMemStorage(0);
        IntPtr contour = new IntPtr();
        CvInvoke.cvFindContours(
            imgForContour,
            storage,
            ref contour,
```

```
            System.Runtime.InteropServices.Marshal.SizeOf(typeof(MCvContour)),
            Emgu.CV.CvEnum.RETR_TYPE.CV_RETR_EXTERNAL,
            Emgu.CV.CvEnum.CHAIN_APPROX_METHOD.CV_CHAIN_APPROX_NONE,
            new Point(0, 0));
        Seq<Point> seq = new Seq<Point>(contour, null);
        for (; seq != null && seq.Ptr.ToInt64() != 0; seq = seq.HNext)
        {
            MCvFont fontt = new MCvFont(FONT.CV_FONT_HERSHEY_SIMPLEX, 0.5,
0.5);
            Rectangle bndRec = CvInvoke.cvBoundingRect(seq, 2);
            double areaC = CvInvoke.cvContourArea(seq, MCvSlice.WholeSeq, 1) * -1;
            if (areaC > areaV)
            {
                CvInvoke.cvRectangle(showRecOnImg, new Point(bndRec.X, bndRec.Y),
                    new Point(bndRec.X + bndRec.Width, bndRec.Y + bndRec.Height),
                    drawcolors, 1, LINE_TYPE.CV_AA, 0);
                CvInvoke.cvPutText(showRecOnImg, (bndRec.X + bndRec.Width /
2).ToString() + "," + (bndRec.Y + bndRec.Height / 2).ToString(),
                     new Point(bndRec.X + 5, bndRec.Y - 20), ref fontt, drawcolors);
                position[pos].X = bndRec.X ;
                position[pos].Y = bndRec.Y ;
                area[pos].X = bndRec.Width;
                area[pos].Y = bndRec.Height;
            }
        }
    }
    private void numeric_Lo_ValueChanged(object sender, EventArgs e)
    {
        trackBar_HL.Value = Convert.ToInt32(numeric_HL.Value);
        trackBar_SL.Value = Convert.ToInt32(numeric_SL.Value);
        trackBar_VL.Value = Convert.ToInt32(numeric_VL.Value);
```

```
}
private void numeric_Hi_ValueChanged(object sender, EventArgs e)
{
    trackBar_HH.Value = Convert.ToInt32(numeric_HH.Value);
    trackBar_SH.Value = Convert.ToInt32(numeric_SH.Value);
    trackBar_VH.Value = Convert.ToInt32(numeric_VH.Value);
}
private void numeric_VAr_ValueChanged(object sender, EventArgs e)
{
    trackBar_VAr.Value = Convert.ToInt32(numeric_VAr.Value);
}
private void trackBar_Lo_ValueChanged(object sender, EventArgs e)
{
    if (trackBar_HL.Value >= trackBar_HH.Value)
        trackBar_HH.Value = trackBar_HL.Value;
    numeric_HL.Value = trackBar_HL.Value;
    numeric_SL.Value = trackBar_SL.Value;
    numeric_VL.Value = trackBar_VL.Value;
}
private void trackBar_Hi_ValueChanged(object sender, EventArgs e)
{
    if (trackBar_HH.Value <= trackBar_HL.Value)
        trackBar_HL.Value = trackBar_HH.Value;
    numeric_HH.Value = trackBar_HH.Value;
    numeric_SH.Value = trackBar_SH.Value;
    numeric_VH.Value = trackBar_VH.Value;
}
private void trackBar_VAr_ValueChanged(object sender, EventArgs e)
{
    numeric_VAr.Value = trackBar_VAr.Value;
}
```

```
// ตั้งค่าไลบาร์เป็นค่าเริ่มต้น
private void btn_Reset_Click(object sender, EventArgs e)
{
    trackBar_HH.Value = 0;
    trackBar_HL.Value = 0;
    trackBar_SH.Value = 255;
    trackBar_SL.Value = 0;
    trackBar_VH.Value = 255;
    trackBar_VL.Value = 0;
    checkBox_IV.Checked = false;
}
// เก็บค่าของสีของแต่ละค่าของ ปลายแขนโรบออท ABB
private void btn_setTool_Click(object sender, EventArgs e)
{
    Cursor.Current = Cursors.WaitCursor;
    _Tool.H.X = Convert.ToInt32(numeric_HL.Value);
    _Tool.H.Y = Convert.ToInt32(numeric_HH.Value);
    _Tool.S.X = Convert.ToInt32(numeric_SL.Value);
    _Tool.S.Y = Convert.ToInt32(numeric_SH.Value);
    _Tool.V.X = Convert.ToInt32(numeric_VL.Value);
    _Tool.V.Y = Convert.ToInt32(numeric_VH.Value);
    _Tool.invert = checkBox_IV.Checked;
    Thread.Sleep(500);
    Cursor.Current = Cursors.Default;
}
     // เก็บค่าของสีวัตถุ A
private void btn_objectA_Click(object sender, EventArgs e)
{
    Cursor.Current = Cursors.WaitCursor;
    _objectA.H.X = Convert.ToInt32(numeric_HL.Value);
    _objectA.H.Y = Convert.ToInt32(numeric_HH.Value);
```

```
    _objectA.S.X = Convert.ToInt32(numeric_SL.Value);
    _objectA.S.Y = Convert.ToInt32(numeric_SH.Value);
    _objectA.V.X = Convert.ToInt32(numeric_VL.Value);
    _objectA.V.Y = Convert.ToInt32(numeric_VH.Value);
    _objectA.invert = checkBox_IV.Checked;
    Thread.Sleep(500);
    Cursor.Current = Cursors.Default;
}
// เก็บค่าของสีวัตถุ B
private void btn_objectB_Click(object sender, EventArgs e)
{
    Cursor.Current = Cursors.WaitCursor;
    _objectB.H.X = Convert.ToInt32(numeric_HL.Value);
    _objectB.H.Y = Convert.ToInt32(numeric_HH.Value);
    _objectB.S.X = Convert.ToInt32(numeric_SL.Value);
    _objectB.S.Y = Convert.ToInt32(numeric_SH.Value);
    _objectB.V.X = Convert.ToInt32(numeric_VL.Value);
    _objectB.V.Y = Convert.ToInt32(numeric_VH.Value);
    _objectB.invert = checkBox_IV.Checked;
    Thread.Sleep(500);
    Cursor.Current = Cursors.Default;
}
#endregion
#region TCP/IP
byte[] m_dataBuffer = new byte[10];
IAsyncResult m_result;
public AsyncCallback m_pfnCallBack;
public Socket m_clientSocket;
String GetIP()
{
    String strHostName = Dns.GetHostName();
```

```
            // Find host by name
            IPHostEntry iphostentry = Dns.GetHostByName(strHostName);
            // Grab the first IP addresses
            String IPStr = "";
            foreach (IPAddress ipaddress in iphostentry.AddressList)
            {
                IPStr = ipaddress.ToString();
                return IPStr;
            }
            return IPStr;
        }
            private void connectToServerToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            // See if we have text on the IP and Port text fields
            if (textBoxIP.Text == "" || textBoxPort.Text == "")
            {
                MessageBox.Show("IP Address and Port Number are required to connect
to the Server\n");
                return;
            }
            try
            {
                UpdateControls(false);
                // Create the socket instance
                m_clientSocket = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
                // Cet the remote IP address
                IPAddress ip = IPAddress.Parse(textBoxIP.Text);
                int iPortNo = System.Convert.ToInt16(textBoxPort.Text);
                // Create the end point
```

```
            IPEndPoint ipEnd = new IPEndPoint(ip, iPortNo);
            // Connect to the remote host
            m_clientSocket.Connect(ipEnd);
            if (m_clientSocket.Connected)
            {

                UpdateControls(true);
                //Wait for data asynchronously
                WaitForData();
            }
        }
        catch (SocketException se)
        {
            string str;
            str = "\nConnection failed, is the server running?\n" + se.Message;
            MessageBox.Show(str);
            UpdateControls(false);
        }
    }
    private void UpdateControls(bool connected)
    {
        connectToServerToolStripMenuItem.Enabled = !connected;
        disconnetFromServerToolStripMenuItem.Enabled = connected;
        string connectStatus = connected ? "Connected" : "Not Connected";
        textBoxConnectStatus.Text = connectStatus;
                }

    public void WaitForData()
    {
        try
        {
```

```
            if (m_pfnCallBack == null)
            {
                m_pfnCallBack = new AsyncCallback(OnDataReceived);
            }
            SocketPacket theSocPkt = new SocketPacket();
            theSocPkt.thisSocket = m_clientSocket;
            // Start listening to the data asynchronously
            m_result = m_clientSocket.BeginReceive(theSocPkt.dataBuffer,
                                                   0, theSocPkt.dataBuffer.Length,
                                                   SocketFlags.None,
                                                   m_pfnCallBack,
                                                   theSocPkt);
        }
        catch (SocketException se)
        {
            MessageBox.Show(se.Message);
        }
    }

    public class SocketPacket
    {
        public System.Net.Sockets.Socket thisSocket;
        public byte[] dataBuffer = new byte[1024];
    }
    public void OnDataReceived(IAsyncResult asyn)
    {
        try
        {
            SocketPacket theSockId = (SocketPacket)asyn.AsyncState;
            int iRx = theSockId.thisSocket.EndReceive(asyn);
            char[] chars = new char[iRx + 1];
```

```
System.Text.Decoder d = System.Text.Encoding.UTF8.GetDecoder();
int charLen = d.GetChars(theSockId.dataBuffer, 0, iRx, chars, 0);
System.String szData = new System.String(chars);
if (!checkBox_All.Checked)
{
    Invoke(new EventHandler(delegate
    {
        SendMessage((position[0].X + "," + area[0].X / 2).ToString());
    }));
}
else
{
    int buff = position[0].X + (area[0].X / 2);
    if (position[1].X < buff && (position[1].X + area[1].X) > buff )
    {
        Invoke(new EventHandler(delegate
        {
            SendMessage("A");
        }));
    }
    else if (position[2].X < buff && (position[2].X+area[2].X) > buff )
    {
        Invoke(new EventHandler(delegate
        {
            SendMessage("B");
        }));
    }
    else
    {
        Invoke(new EventHandler(delegate
        {
```

```
                        SendMessage("O");
                    }));
                }
            }
            WaitForData();
        }
        catch (ObjectDisposedException)
        {
            System.Diagnostics.Debugger.Log(0, "1", "\nOnDataReceived: Socket has
been closed\n");
        }
        catch (SocketException se)
        {
            MessageBox.Show(se.Message);
        }
    }

    private void SendMessage(string msg)
    {
        try
        {
            // New code to send strings
            NetworkStream networkStream = new NetworkStream(m_clientSocket);
            System.IO.StreamWriter streamWriter = new
System.IO.StreamWriter(networkStream);
            streamWriter.WriteLine(msg);
            streamWriter.Flush();

            /* Use the following code to send bytes
            byte[] byData = System.Text.Encoding.ASCII.GetBytes(objData.ToString ());
            if(m_clientSocket != null){
```

```
                m_clientSocket.Send (byData);
            }
            */
        }
        catch (SocketException se)
        {
            MessageBox.Show(se.Message);
        }
    }
    private void disconnetFromServerToolStripMenuItem_Click(object sender,
EventArgs e)
    {
        if (m_clientSocket != null)
        {
            m_clientSocket.Close();
            m_clientSocket = null;
            UpdateControls(false);
        }
    }
    #endregion
    // ปุ่มการตั้งค่าความสูงของกล่อง
    private void button1_Click(object sender, EventArgs e)
    {
        textBox1.Text = "Object A :" + (position[1].X + area[1].X / 2).ToString() + "," +
(position[1].Y + area[1].Y / 2).ToString();
        textBox2.Text = "Object B :" + (position[2].X + area[2].X / 2).ToString() + "," +
(position[2].Y + area[2].Y / 2).ToString();
    }
  }
}
```