COLLEGE OF ENGINEERING

Electrical Engineering & Computer Science

Video Radar

Calculating the speed of an object in real time using a video camera.

Considered approaches

To build our system and application, we started off knowing we were going to need a camera, computer vision library, and:

- OpenCV
- ZED Stereo Camera: Two cameras side-by-side that parse depth by differences between the two images.

The Kinect

The Kinect is a camera device that was made by Microsoft for use with video games for their Xbox 360 games console. The Kinect has 2 cameras and a infrared laser projector. The first camera is a color camera for normal color view. The second camera is an infrared camera. The infrared projector projects a pattern of dots in front of it that the infrared camera then picks up. The Kinect then determines the distance depending on the pattern and spread of the dots.

Possible Applications

Our project could have several application in real life, tracking various subject, here are af few:

- Cars
- Athletes
- Sporting objects
- Analyzing crowds
- Animals

The Problem

The main method for people to track the speed of an object as they go by is to use a radar gun or laser gun, but unfortunately these methods have issues, they can only be used to check the speed of one object, the user can not tell exactly which object is being measured if there are multiple close together, and they must have an active user.

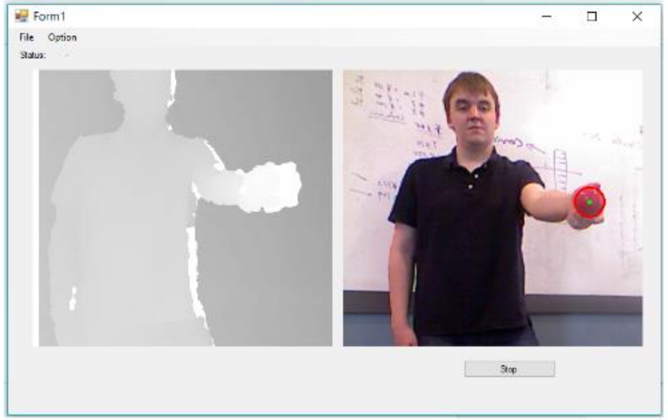
Our Process

First the Kinect camera calculates the depth map based on the infrared projections. Then it passes both the depth map image and the color image to our program. We then run an Emgu CV detection algorithm which identifies the red ball and its center. We then look up the distance to the center of the ball on the depth map. Next, we calculate the real world coordinates of the ball based on its distance from the camera. Finally, we calculate velocity based on change in position vs change in time.

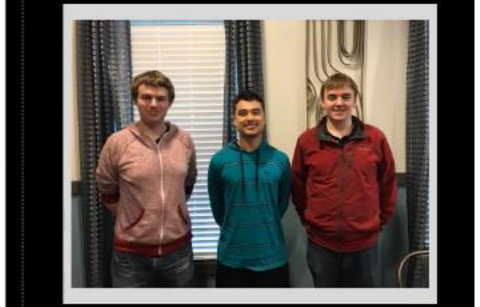
The Solution

Our solution to this problem is to use a camera in order to overcome these issues. For our proof of concept, we are using a Kinect camera. The use of the kinect allows us to measure the distance from the camera to an object so that we can track the velocity of objects. Since the video is being displayed the user can also see which object is going what velocity.

Our solution consists of a Graphical User Interface (GUI) application, which is made very simple for ease of use, that has a box that displays the live camera feed, where the objects that are being tracked are surrounded by a colored box. The velocity of the object is displayed in the text box to the side of the GUI. There is only one button that starts the camera and object tracking and stops it when clicked again. Our solution will also generate a comma separated text file so the user can review the data at a later date.



Ball identification with depth map



Team members:

Alexander Bailey (baileyal@oregonstate.edu)
Benjamin Wick (wickbe@oregonstate.edu)
Dylan Washburne (washburd@oregonstate.edu)

Project Advisors:

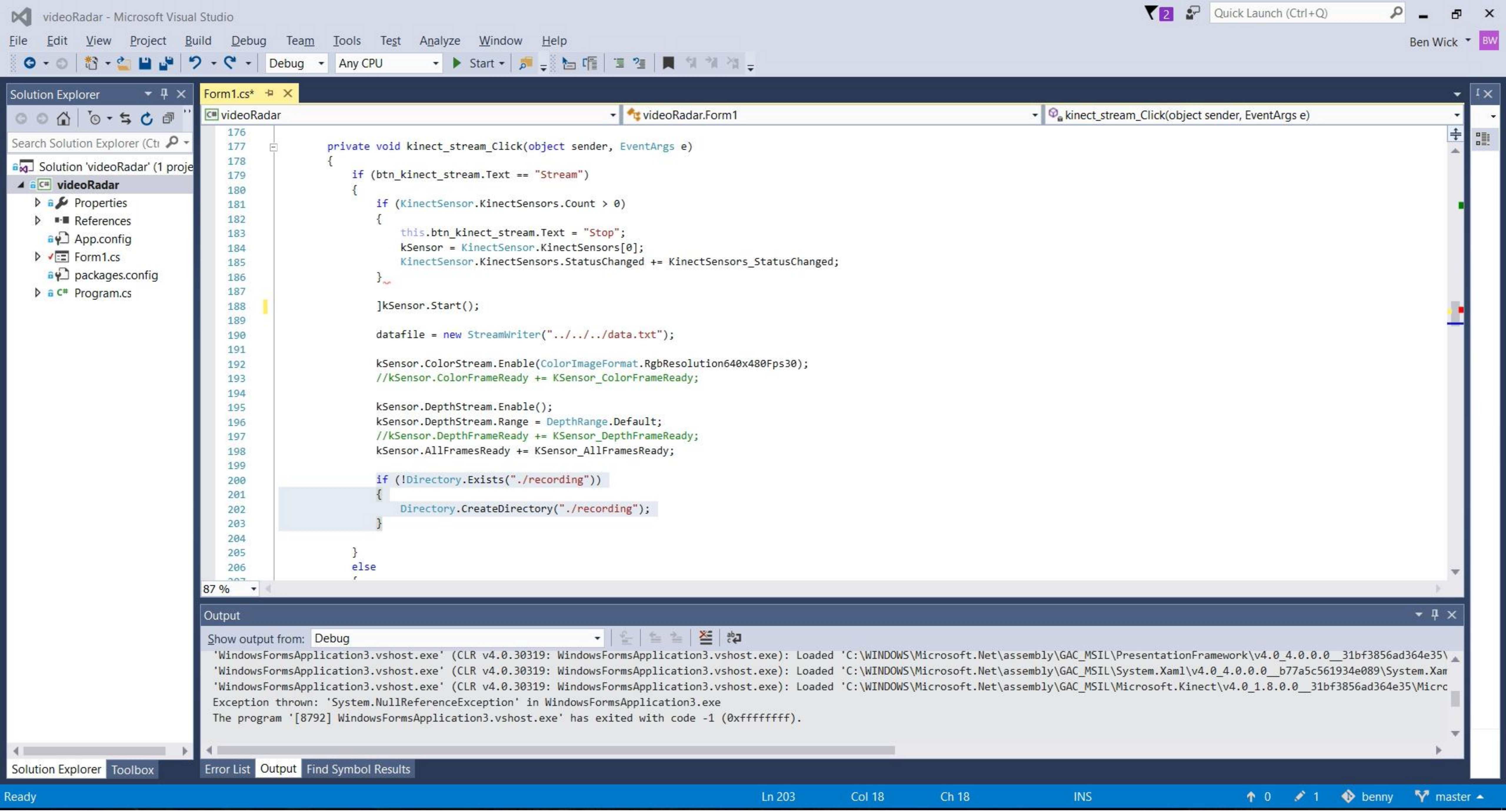
Professor Kevin McGrath
Professor Kirsten Winters
Jon Dodge, Teaching Assistant

Sponser:

Alex Neighbors

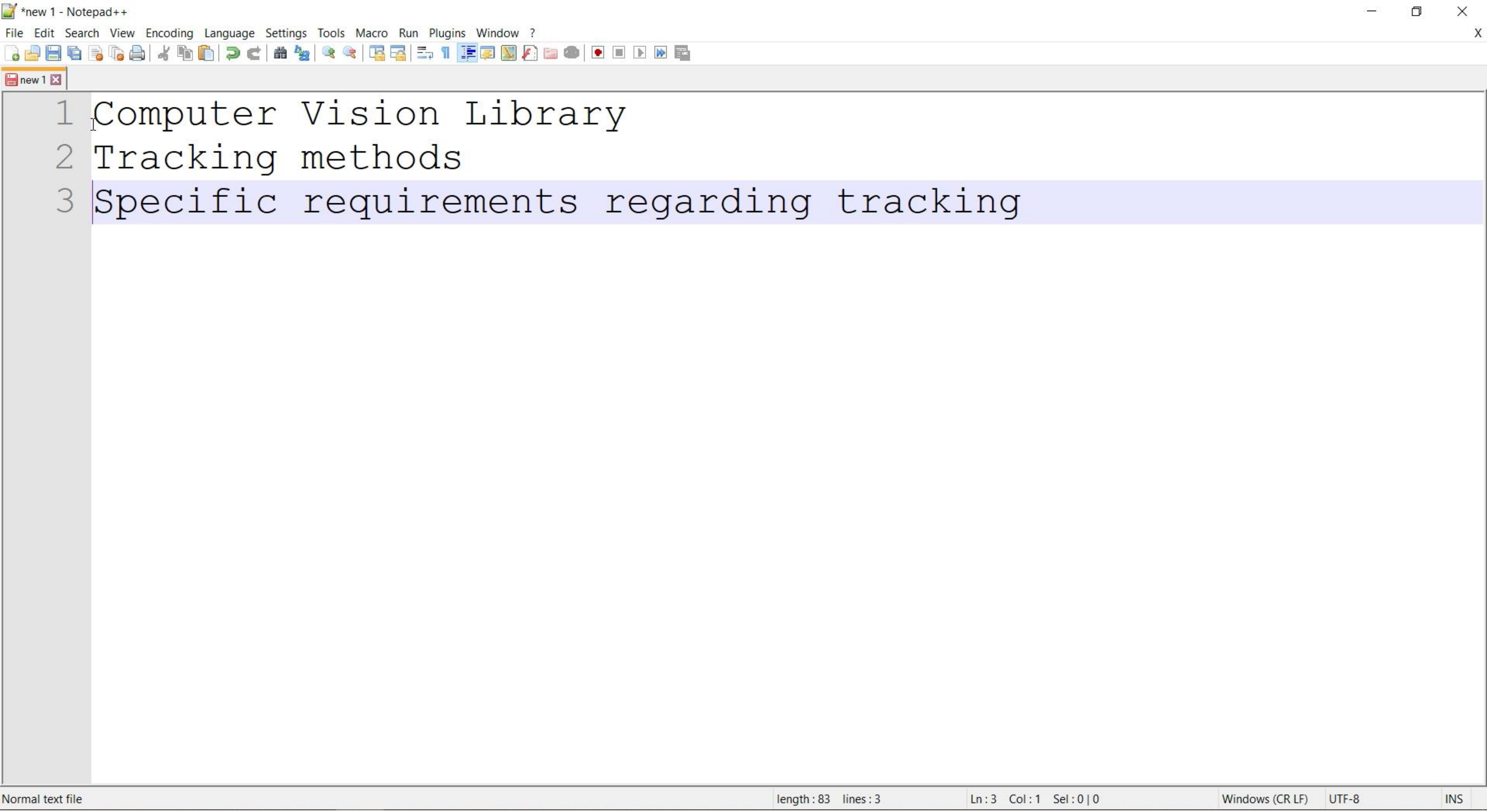
Conclusion

- There are many different applications for this kind of system
- Efficient tracking methods are difficult to create
- To track different types of objects within one application there would have to be a different algorithm for each object which would be difficult
- Computer vision has endless possibilities



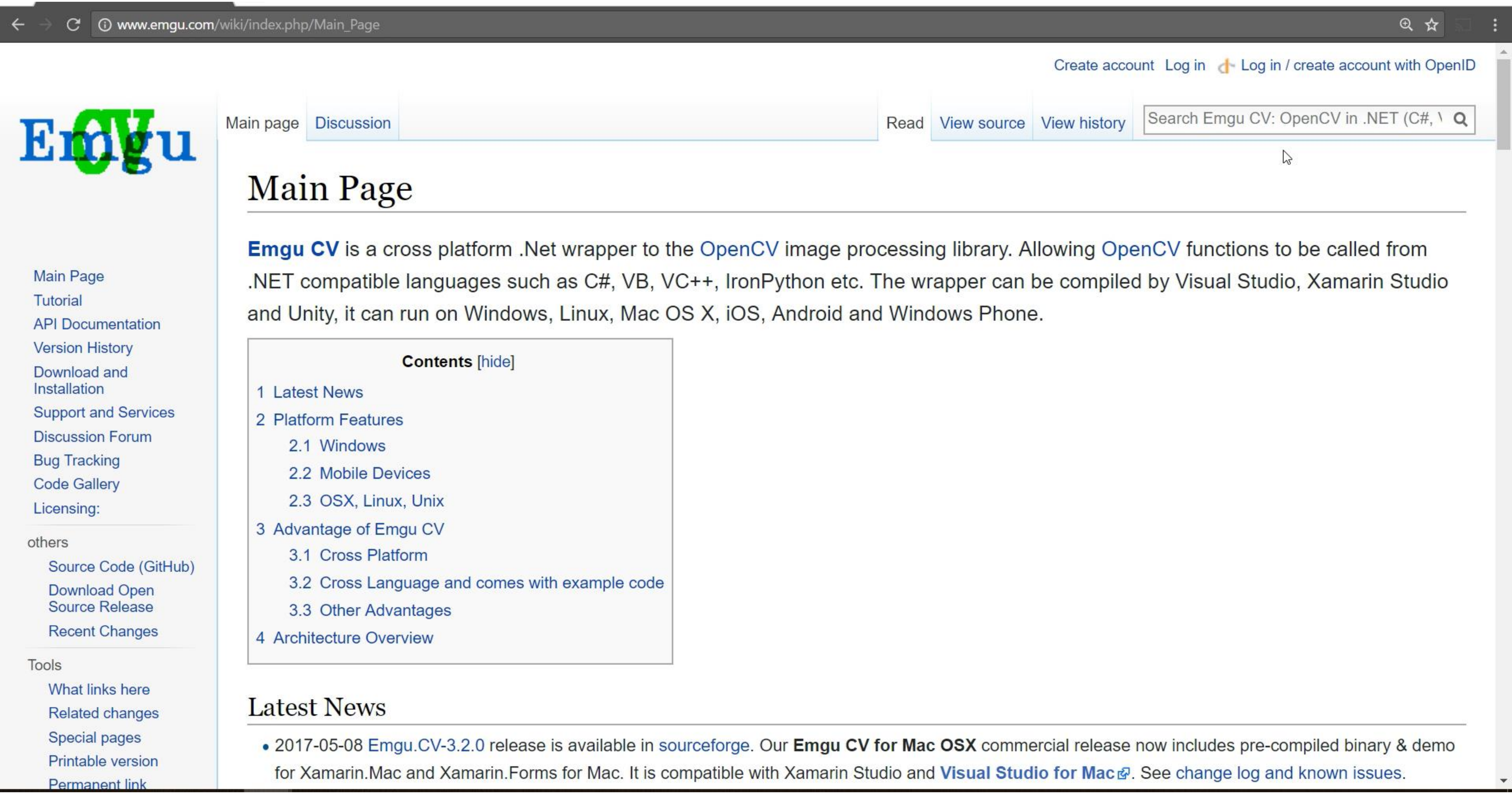
```
        private void kinect_stream_Click(object sender, EventArgs e)
        {
            if (btn_kinect_stream.Text == "Stream")
            {
                if (KinectSensor.KinectSensors.Count > 0)
                {
                    this.btn_kinect_stream.Text = "Stop";
                    kSensor = KinectSensor.KinectSensors[0];
                    KinectSensor.KinectSensors.StatusChanged += KinectSensors_StatusChanged;
                }

                ]kSensor.Start();

                datafile = new StreamWriter("../../../data.txt");

                kSensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
                //kSensor.ColorFrameReady += KSensor_ColorFrameReady;

                kSensor.DepthStream.Enable();
                kSensor.DepthStream.Range = DepthRange.Default;
                //kSensor.DepthFrameReady += KSensor_DepthFrameReady;
                kSensor.AllFramesReady += KSensor_AllFramesReady;

                if (!Directory.Exists("./recording"))
                {
                    Directory.CreateDirectory("./recording");
                }

            }
            else
            {
```

```
'WindowsFormsApplication3.vshost.exe' (CLR v4.0.30319: WindowsFormsApplication3.vshost.exe): Loaded 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\PresentationFramework\v4.0_4.0.0.0__31bf3856ad364e35\
'WindowsFormsApplication3.vshost.exe' (CLR v4.0.30319: WindowsFormsApplication3.vshost.exe): Loaded 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Xaml\v4.0_4.0.0.0__b77a5c561934e089\System.Xam
'WindowsFormsApplication3.vshost.exe' (CLR v4.0.30319: WindowsFormsApplication3.vshost.exe): Loaded 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\Microsoft.Kinect\v4.0_1.8.0.0__31bf3856ad364e35\Micrc
Exception thrown: 'System.NullReferenceException' in WindowsFormsApplication3.exe
The program '[8792] WindowsFormsApplication3.vshost.exe' has exited with code -1 (0xffffffff).
```



*new 1 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
new 1
Normal text file
length : 83 lines : 3
Ln : 3 Col : 1 Sel : 0 | 0
Windows (CR LF)
UTF-8
INS

```
Computer Vision Library
Tracking methods
Specific requirements regarding tracking
```



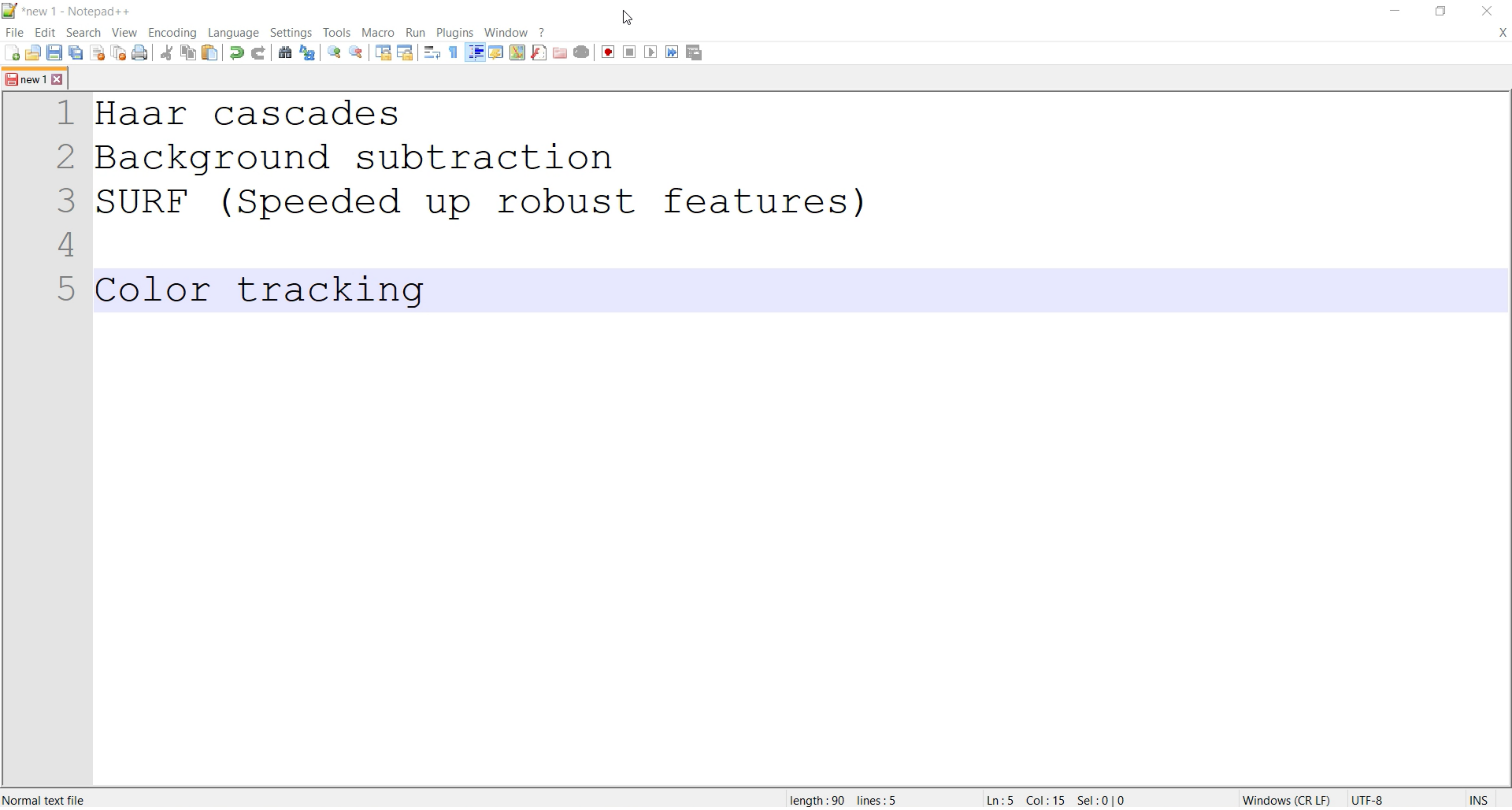
Main Page

Emgu CV is a cross platform .Net wrapper to the OpenCV image processing library. Allowing OpenCV functions to be called from .NET compatible languages such as C#, VB, VC++, IronPython etc. The wrapper can be compiled by Visual Studio, Xamarin Studio and Unity, it can run on Windows, Linux, Mac OS X, iOS, Android and Windows Phone.

Contents [hide]

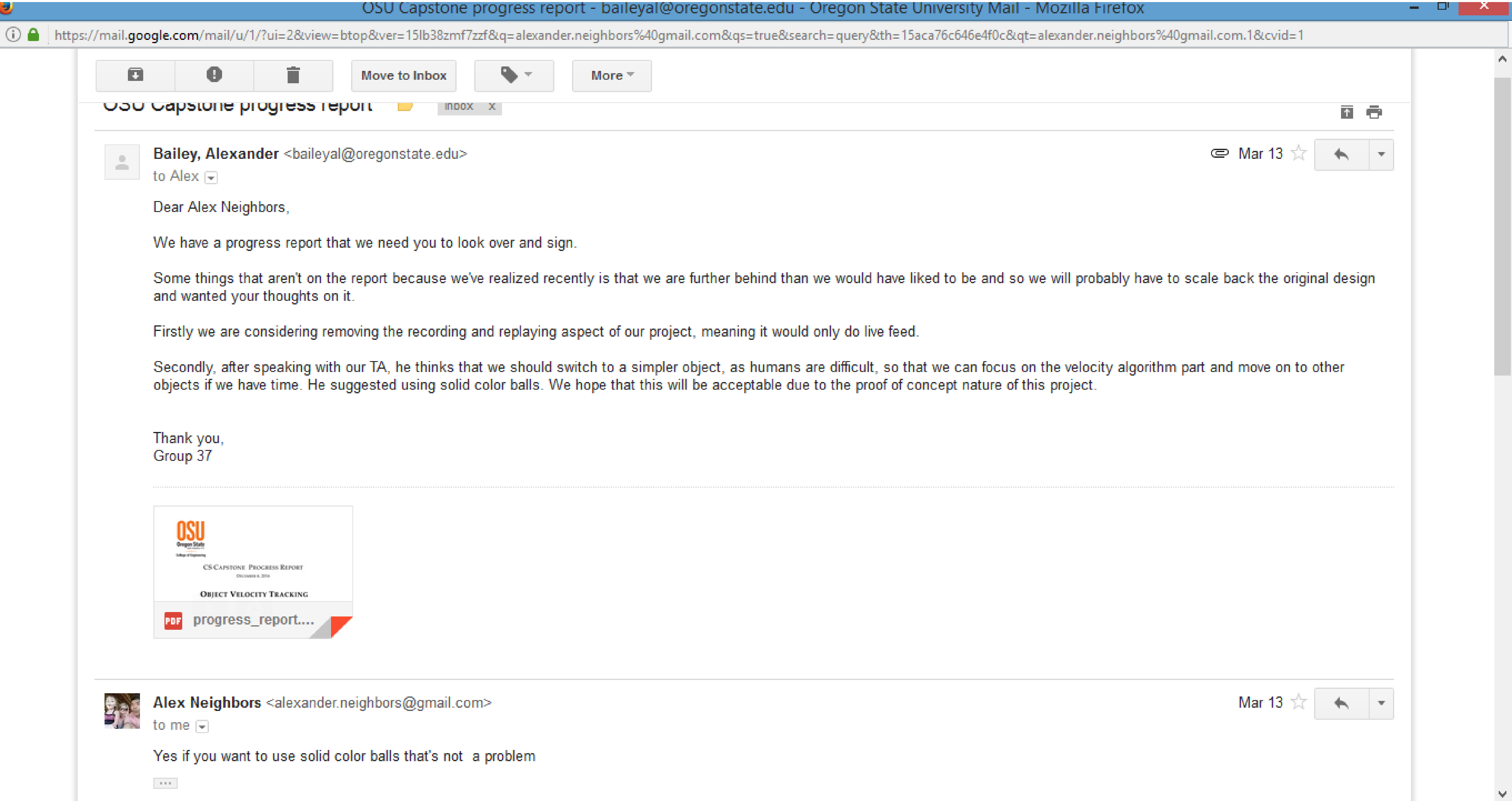
Latest News

- 2017-05-08 Emgu.CV-3.2.0 release is available in sourceforge. Our **Emgu CV for Mac OSX** commercial release now includes pre-compiled binary & demo for Xamarin.Mac and Xamarin.Forms for Mac. It is compatible with Xamarin Studio and **Visual Studio for Mac**. See change log and known issues.



*new 1 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
new 1
Normal text file
length : 90 lines : 5
Ln : 5 Col : 15 Sel : 0 | 0
Windows (CR LF)
UTF-8
INS

```
Haar cascades
Background subtraction
SURF (Speeded up robust features)

Color tracking
```


OSU Capstone progress report - baileyal@oregonstate.edu - Oregon State University Mail - Mozilla Firefox
https://mail.google.com/mail/u/1/?ui=2&view=btop&ver=15lb38zmf7zzf&q=alexander.neighbors%40gmail.com&qs=true&search=query&th=15aca76c646e4f0c&qt=alexander.neighbors%40gmail.com.1&cvid=1
Move to Inbox
More
Inbox x

OSU Capstone progress report

Bailey, Alexander <baileyal@oregonstate.edu>
to Alex

Mar 13

Dear Alex Neighbors,

We have a progress report that we need you to look over and sign.

Some things that aren't on the report because we've realized recently is that we are further behind than we would have liked to be and so we will probably have to scale back the original design and wanted your thoughts on it.

Firstly we are considering removing the recording and replaying aspect of our project, meaning it would only do live feed.

Secondly, after speaking with our TA, he thinks that we should switch to a simpler object, as humans are difficult, so that we can focus on the velocity algorithm part and move on to other objects if we have time. He suggested using solid color balls. We hope that this will be acceptable due to the proof of concept nature of this project.

Thank you,
Group 37

OSU
Oregon State
College of Engineering
CS Capstone Progress Report
December 4, 2016
Object Velocity Tracking

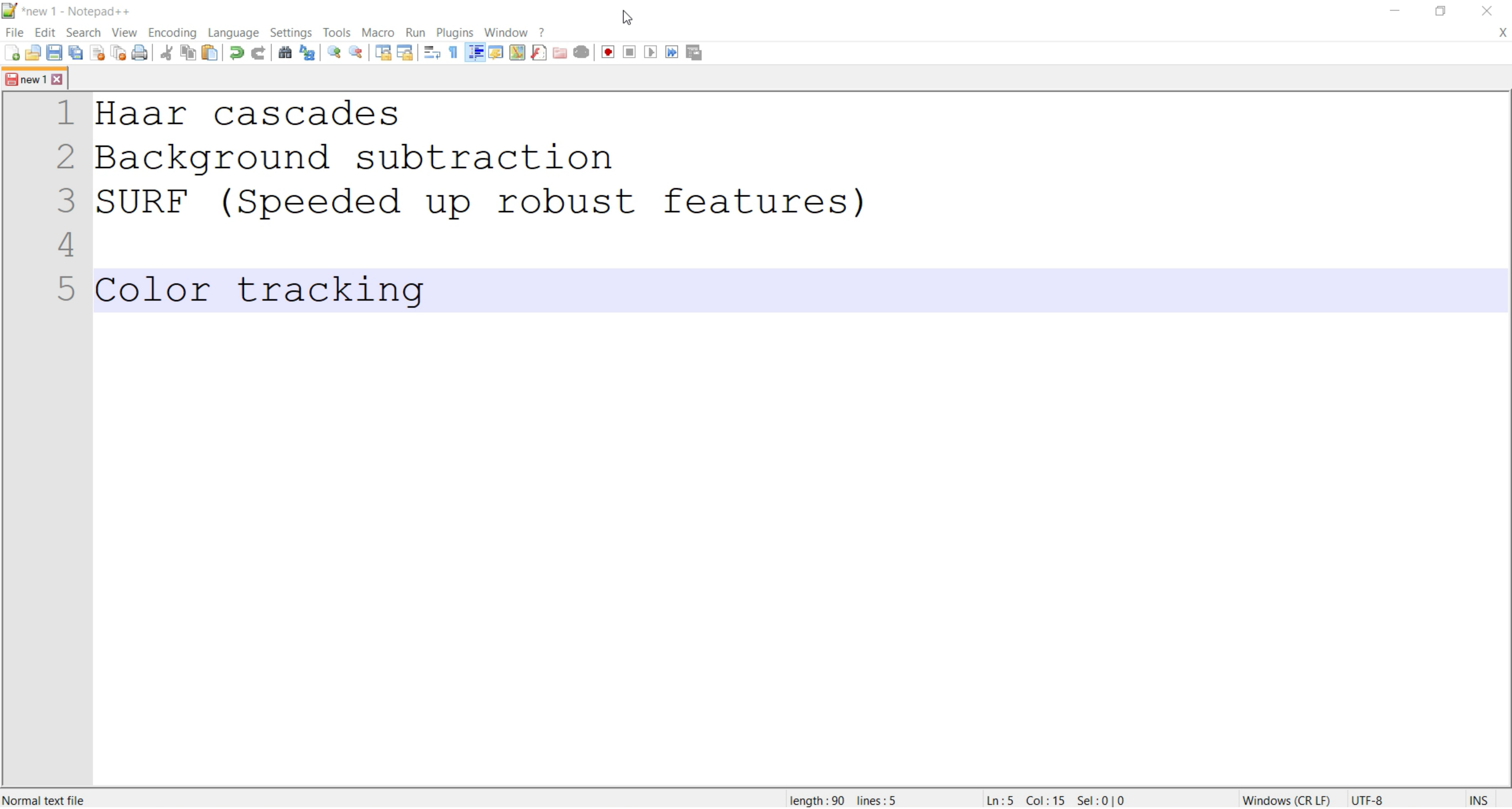
PDF progress_report....

Alex Neighbors <alexander.neighbors@gmail.com>
to me

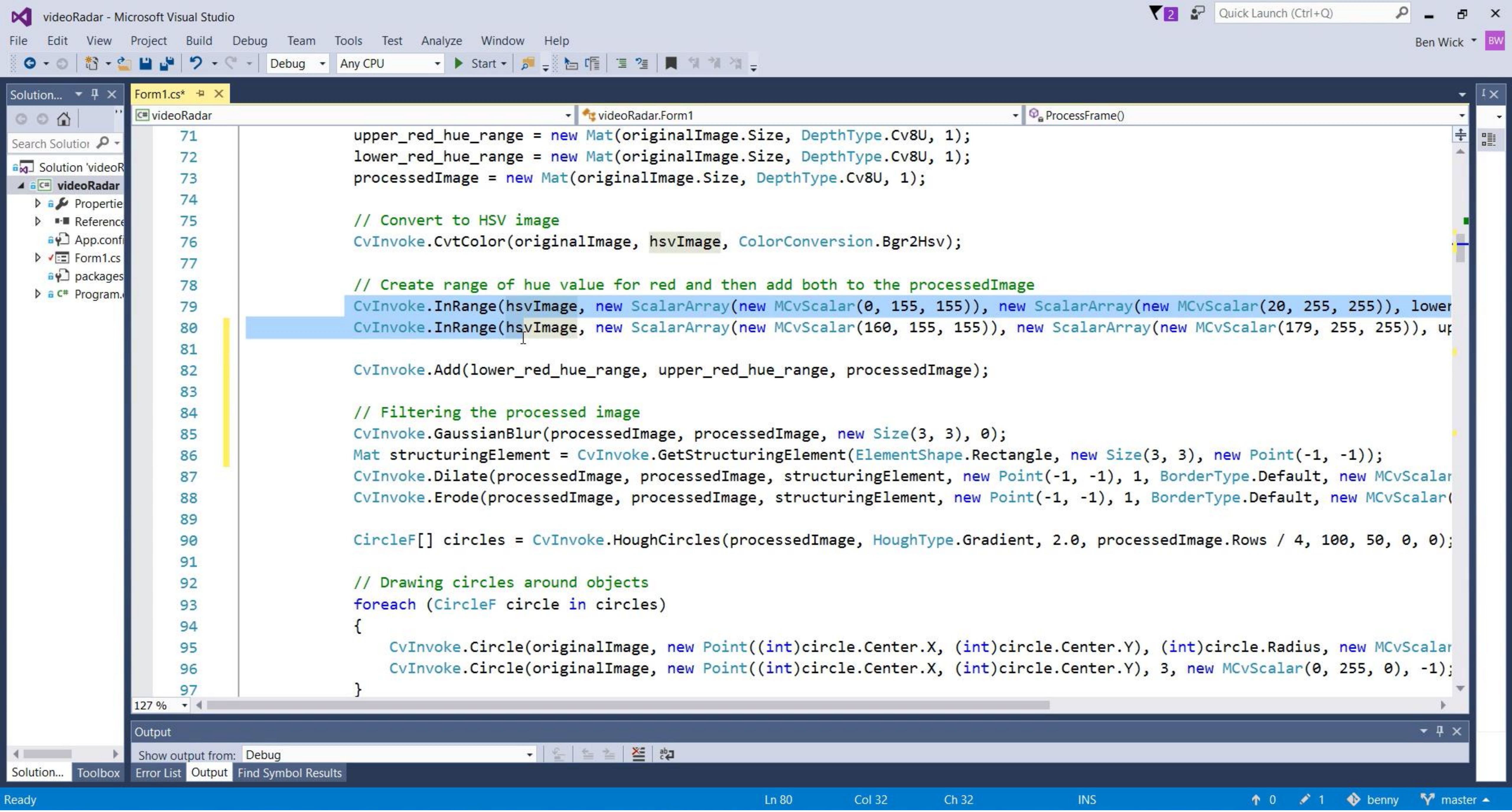
Mar 13

Yes if you want to use solid color balls that's not a problem

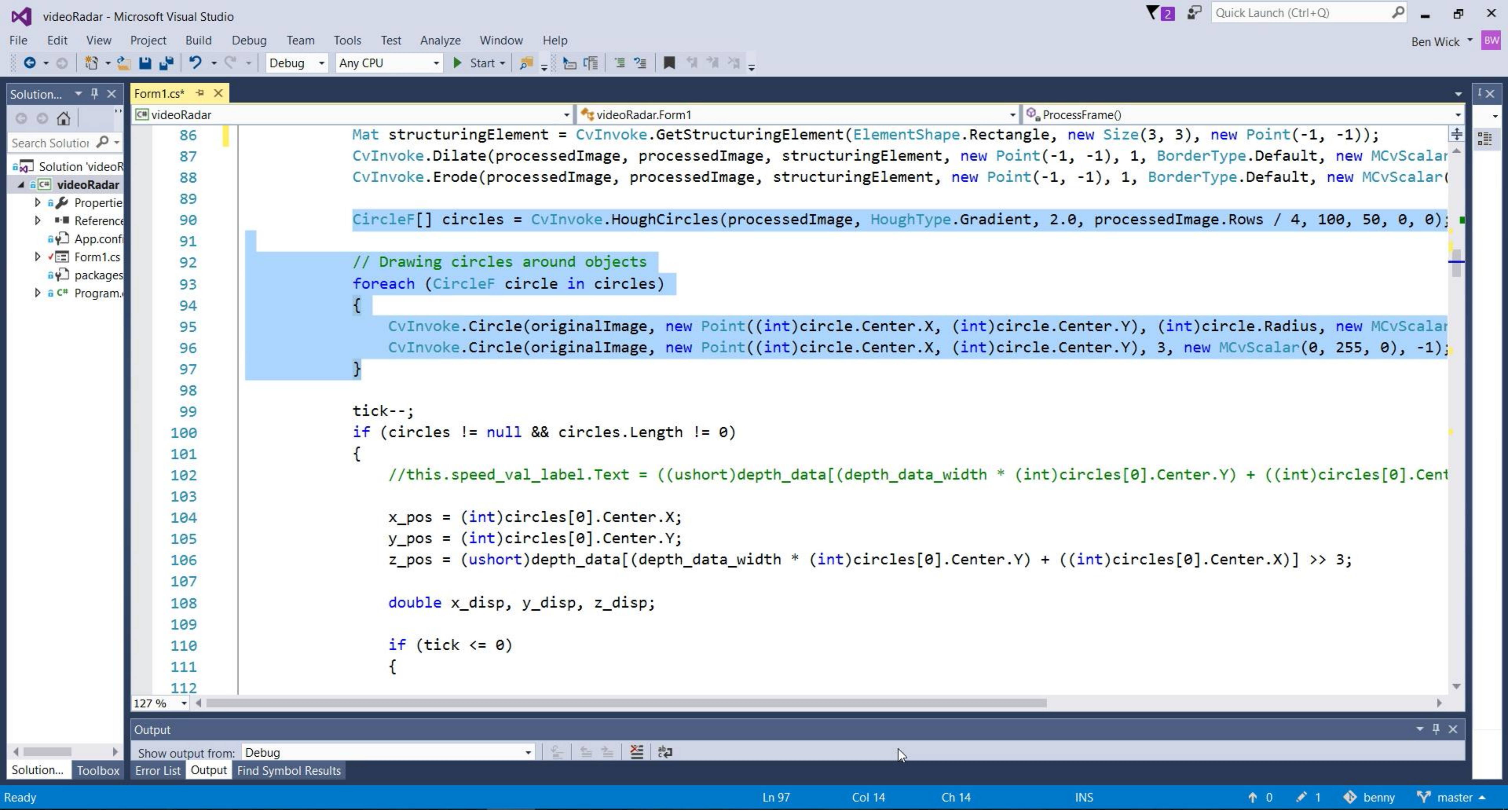
...


*new 1 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
new 1
Normal text file
length : 90 lines : 5
Ln : 5 Col : 15 Sel : 0 | 0
Windows (CR LF)
UTF-8
INS

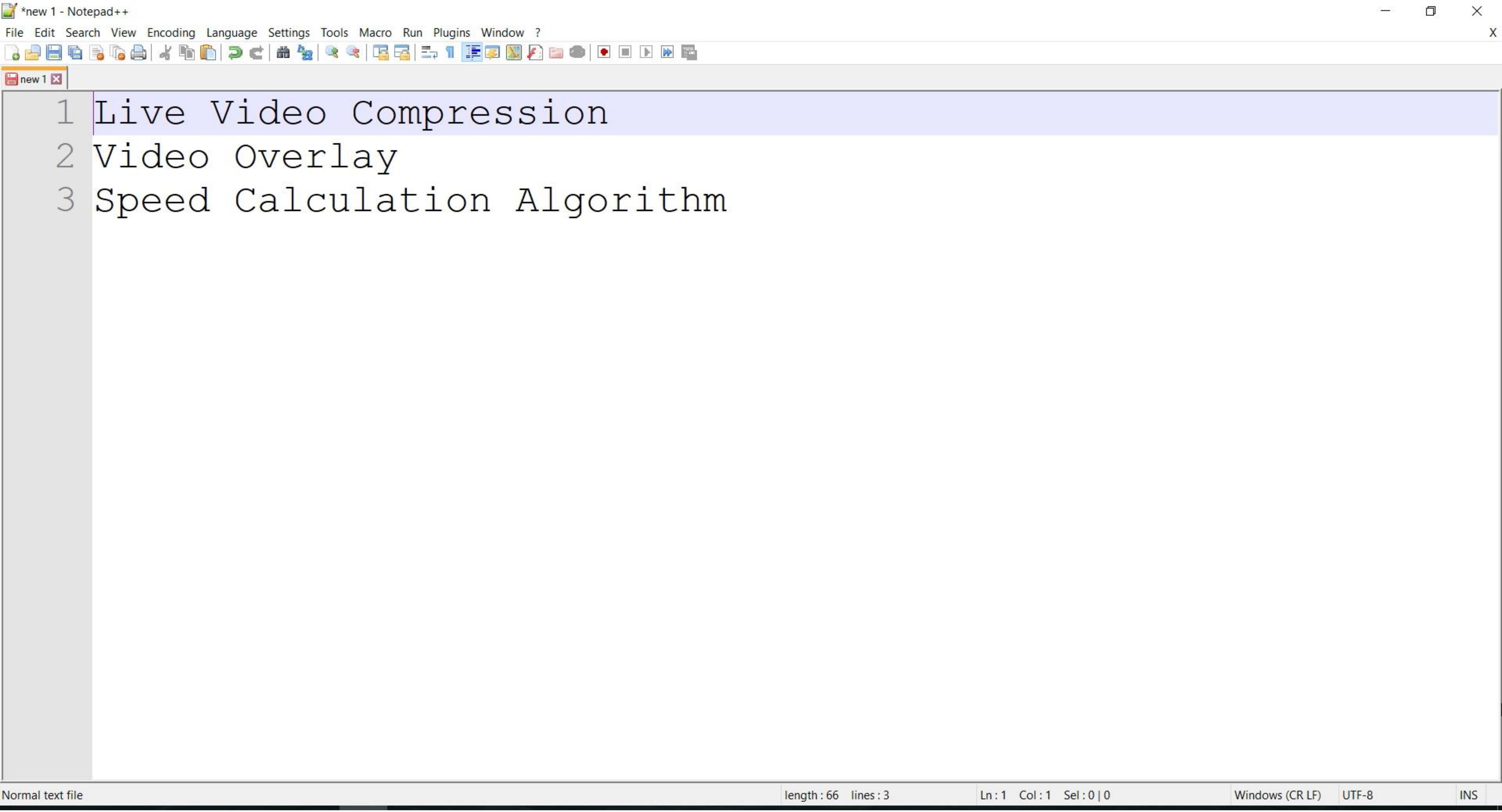
```
Haar cascades
Background subtraction
SURF (Speeded up robust features)

Color tracking
```



```
upper_red_hue_range = new Mat(originalImage.Size, DepthType.Cv8U, 1);
lower_red_hue_range = new Mat(originalImage.Size, DepthType.Cv8U, 1);
processedImage = new Mat(originalImage.Size, DepthType.Cv8U, 1);

// Convert to HSV image
CvInvoke.CvtColor(originalImage, hsvImage, ColorConversion.Bgr2Hsv);

// Create range of hue value for red and then add both to the processedImage
CvInvoke.InRange(hsvImage, new ScalarArray(new MCvScalar(0, 155, 155)), new ScalarArray(new MCvScalar(20, 255, 255)), lower
CvInvoke.InRange(hsvImage, new ScalarArray(new MCvScalar(160, 155, 155)), new ScalarArray(new MCvScalar(179, 255, 255)), up

CvInvoke.Add(lower_red_hue_range, upper_red_hue_range, processedImage);

// Filtering the processed image
CvInvoke.GaussianBlur(processedImage, processedImage, new Size(3, 3), 0);
Mat structuringElement = CvInvoke.GetStructuringElement(ElementShape.Rectangle, new Size(3, 3), new Point(-1, -1));
CvInvoke.Dilate(processedImage, processedImage, structuringElement, new Point(-1, -1), 1, BorderType.Default, new MCvScalar
CvInvoke.Erode(processedImage, processedImage, structuringElement, new Point(-1, -1), 1, BorderType.Default, new MCvScalar(

CircleF[] circles = CvInvoke.HoughCircles(processedImage, HoughType.Gradient, 2.0, processedImage.Rows / 4, 100, 50, 0, 0);

// Drawing circles around objects
foreach (CircleF circle in circles)
{
    CvInvoke.Circle(originalImage, new Point((int)circle.Center.X, (int)circle.Center.Y), (int)circle.Radius, new MCvScalar
    CvInvoke.Circle(originalImage, new Point((int)circle.Center.X, (int)circle.Center.Y), 3, new MCvScalar(0, 255, 0), -1);
}
```

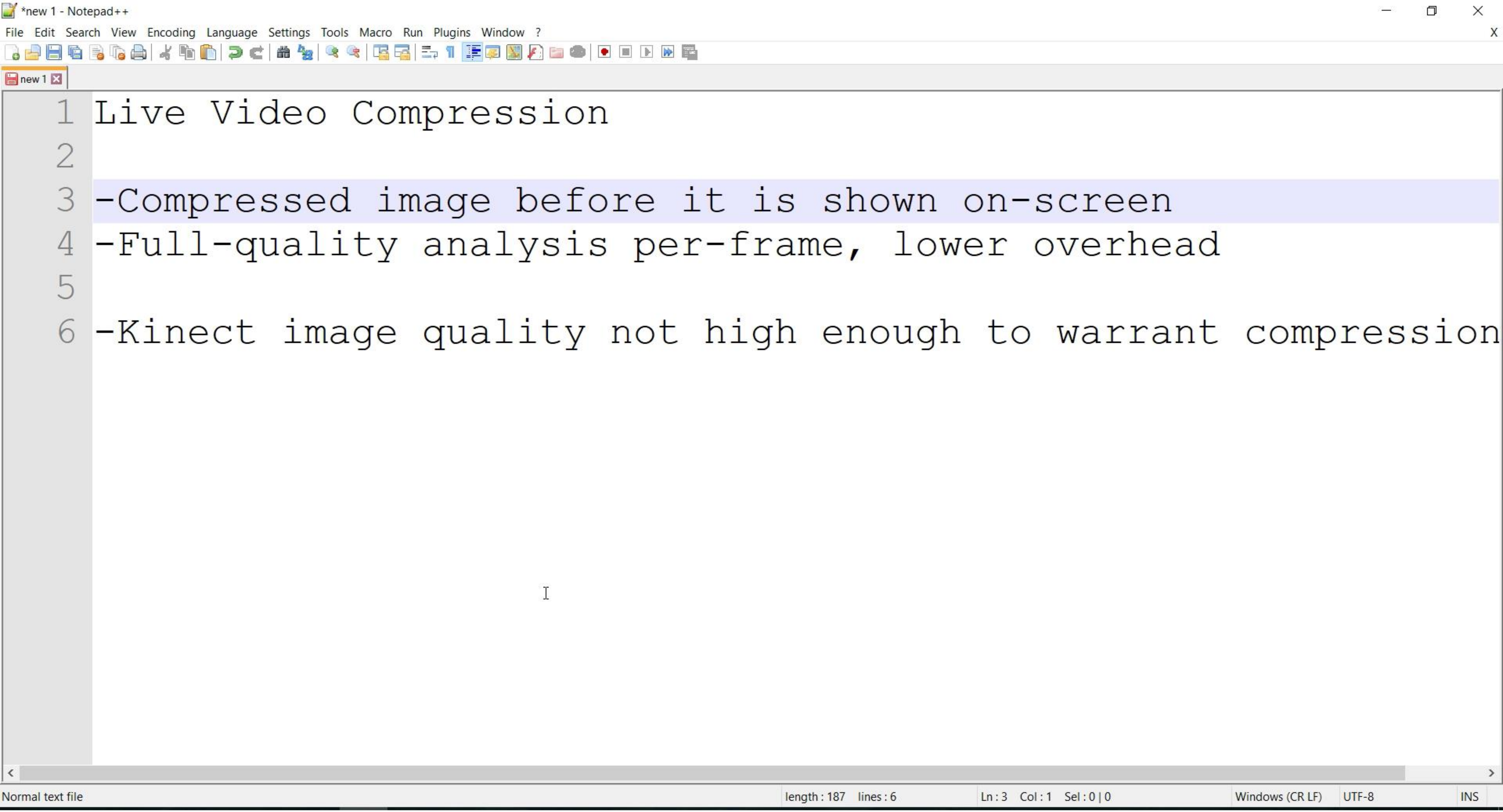


```
Mat structuringElement = CvInvoke.GetStructuringElement(ElementShape.Rectangle, new Size(3, 3), new Point(-1, -1));
CvInvoke.Dilate(processedImage, processedImage, structuringElement, new Point(-1, -1), 1, BorderType.Default, new MCvScalar
CvInvoke.Erode(processedImage, processedImage, structuringElement, new Point(-1, -1), 1, BorderType.Default, new MCvScalar(

CircleF[] circles = CvInvoke.HoughCircles(processedImage, HoughType.Gradient, 2.0, processedImage.Rows / 4, 100, 50, 0, 0);

// Drawing circles around objects
foreach (CircleF circle in circles)
{
    CvInvoke.Circle(originalImage, new Point((int)circle.Center.X, (int)circle.Center.Y), (int)circle.Radius, new MCvScalar
    CvInvoke.Circle(originalImage, new Point((int)circle.Center.X, (int)circle.Center.Y), 3, new MCvScalar(0, 255, 0), -1);
}

tick--;
if (circles != null && circles.Length != 0)
{
    //this.speed_val_label.Text = ((ushort)depth_data[(depth_data_width * (int)circles[0].Center.Y) + ((int)circles[0].Cent

    x_pos = (int)circles[0].Center.X;
    y_pos = (int)circles[0].Center.Y;
    z_pos = (ushort)depth_data[(depth_data_width * (int)circles[0].Center.Y) + ((int)circles[0].Center.X)] >> 3;

    double x_disp, y_disp, z_disp;

    if (tick <= 0)
    {

```


*new 1 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
new 1

```
Live Video Compression
Video Overlay
Speed Calculation Algorithm
```

Normal text file
length : 66 lines : 3
Ln : 1 Col : 1 Sel : 0 | 0
Windows (CR LF)
UTF-8
INS

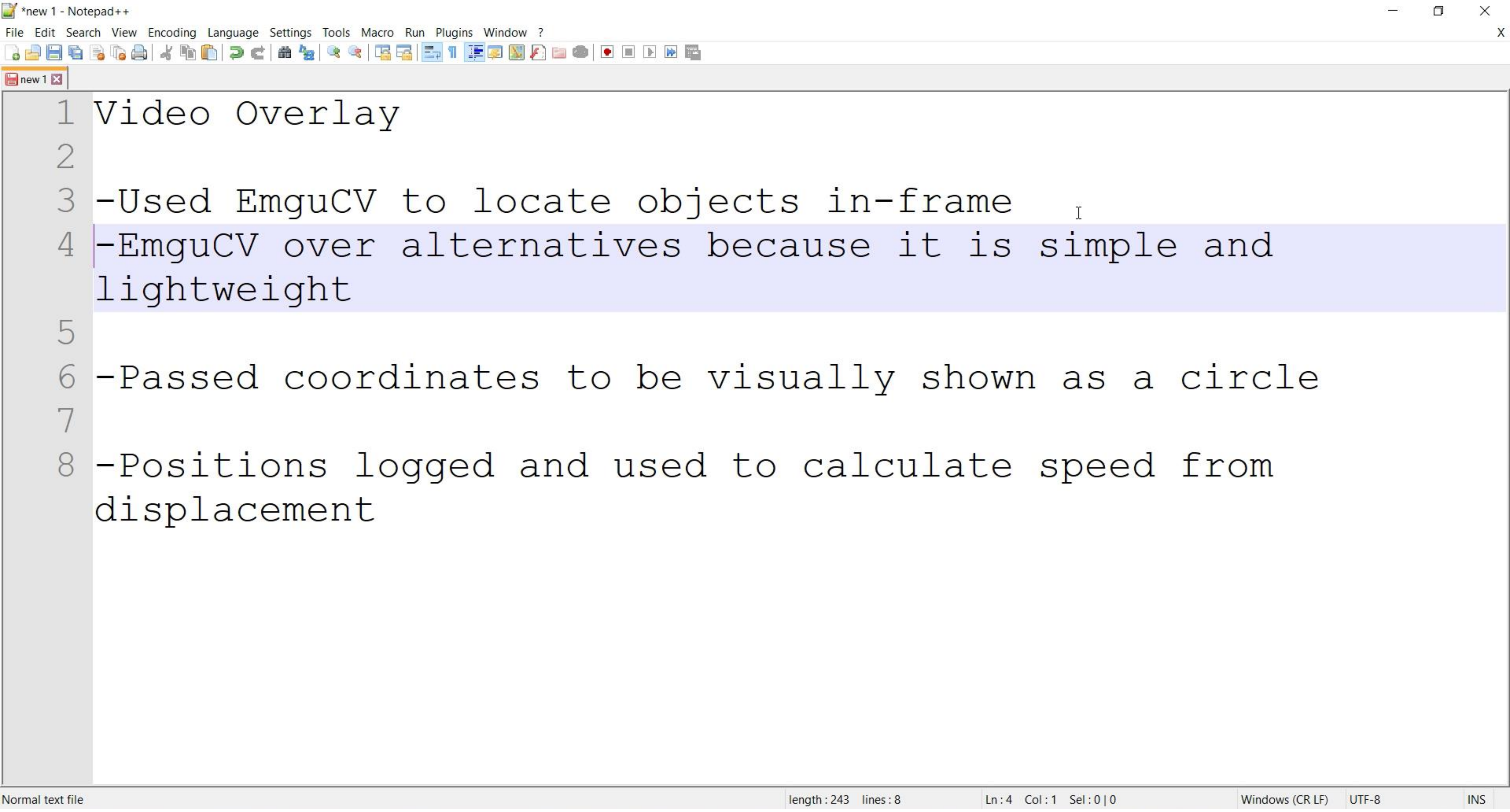


Live Video Compression

-Compressed image before it is shown on-screen

-Full-quality analysis per-frame, lower overhead

-Kinect image quality not high enough to warrant compression



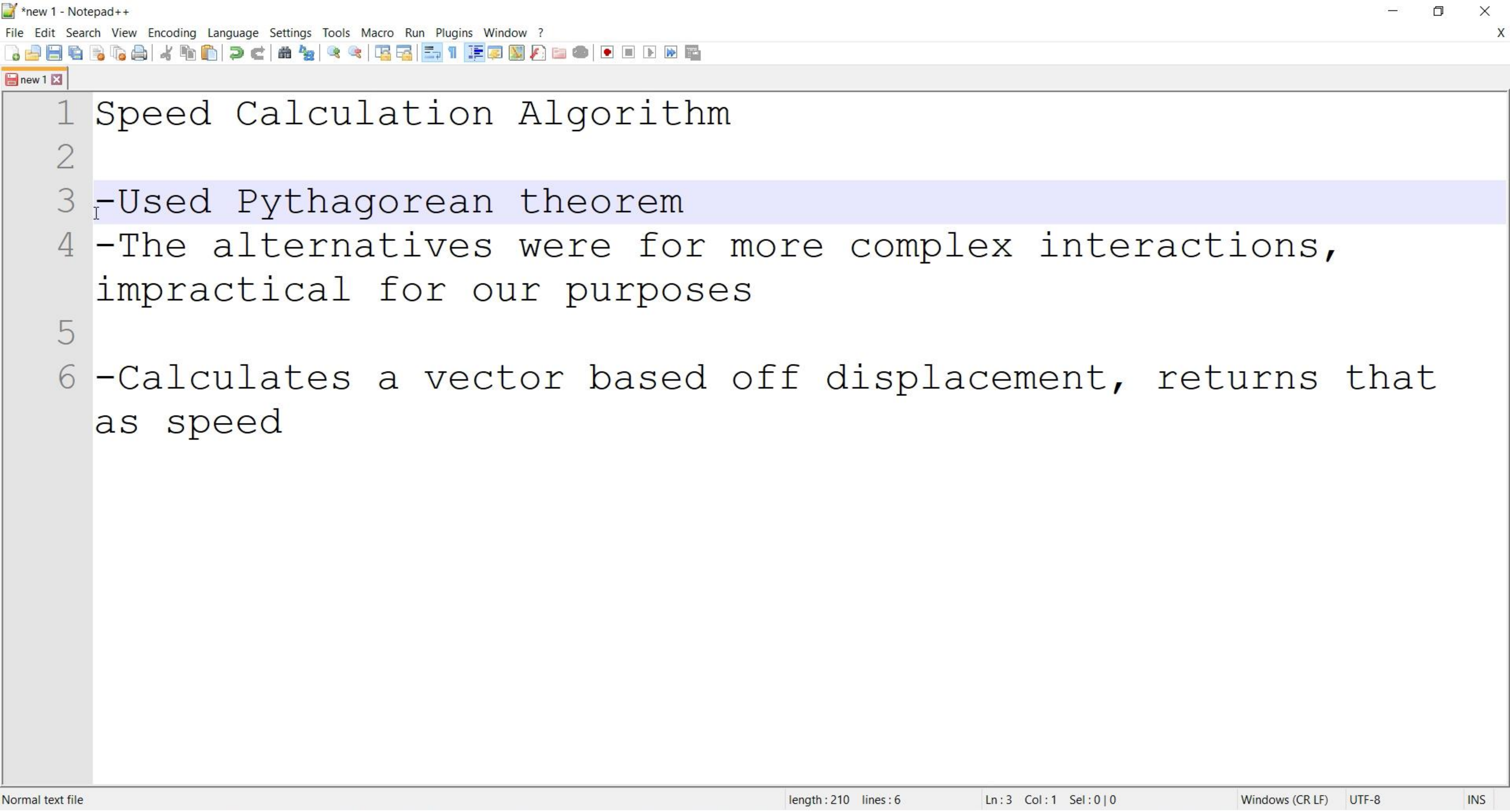
Video Overlay

-Used EmguCV to locate objects in-frame

-EmguCV over alternatives because it is simple and lightweight

-Passed coordinates to be visually shown as a circle

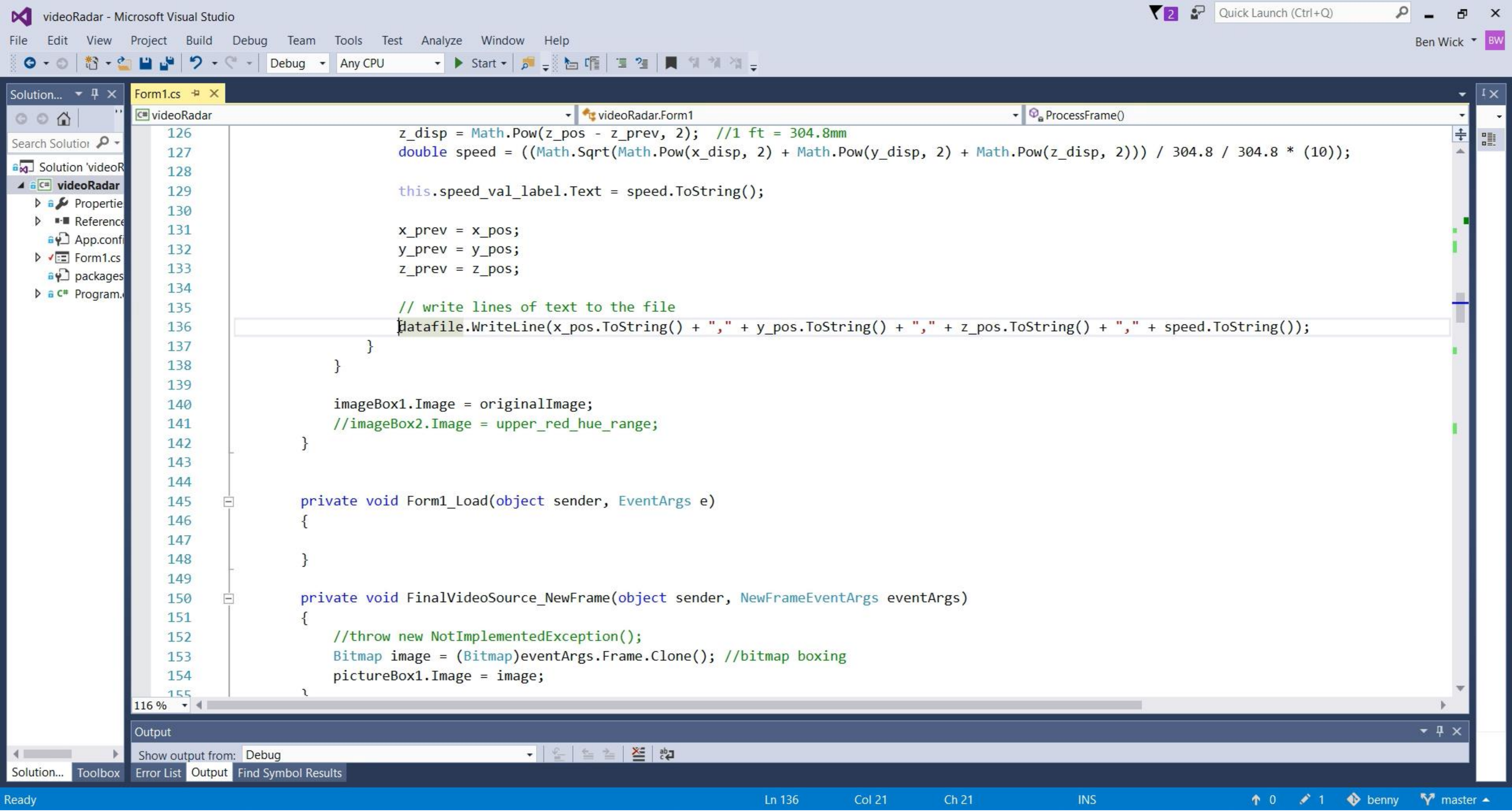
-Positions logged and used to calculate speed from displacement



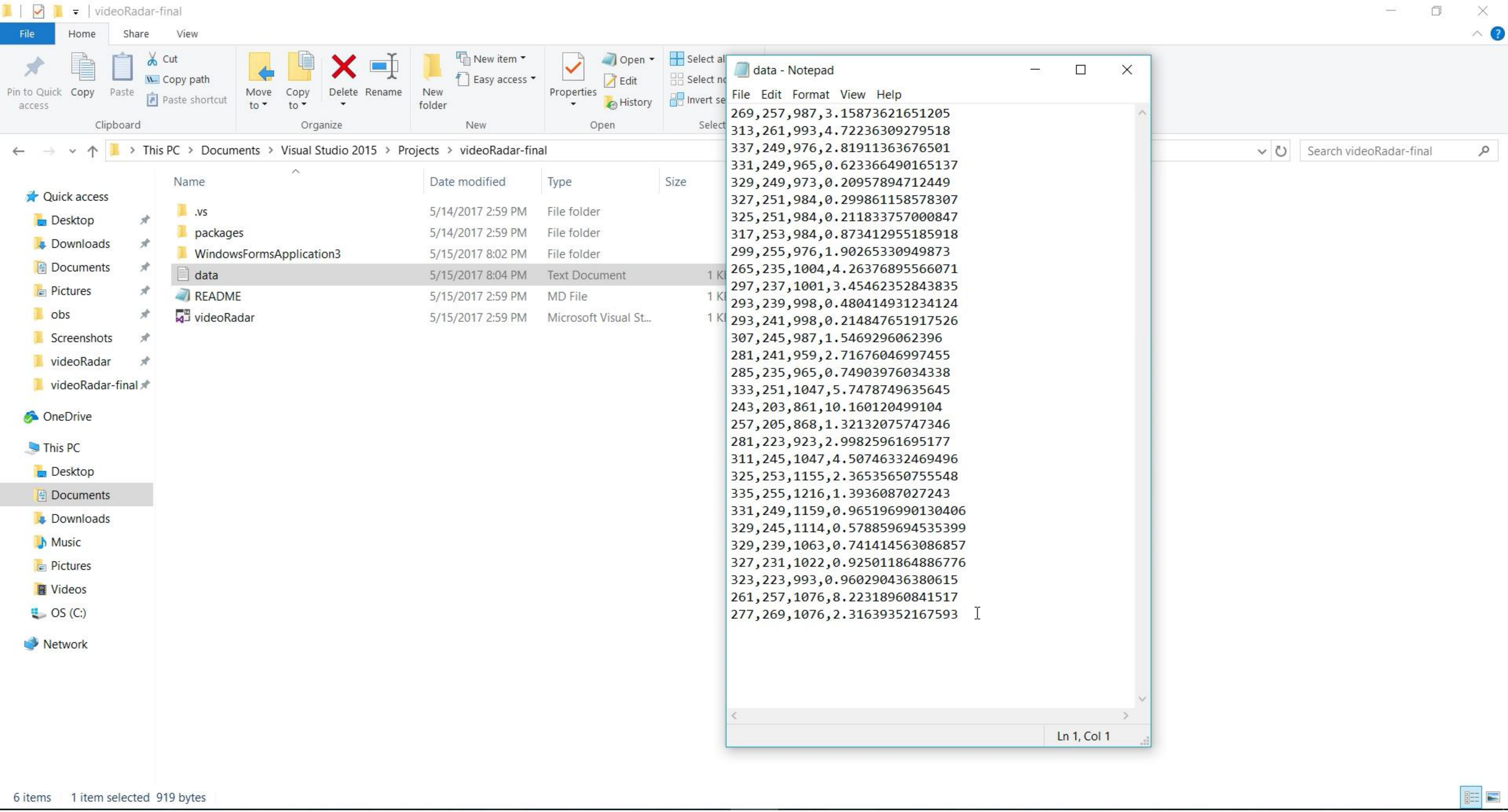
Speed Calculation Algorithm

-Used Pythagorean theorem
-The alternatives were for more complex interactions, impractical for our purposes

-Calculates a vector based off displacement, returns that as speed

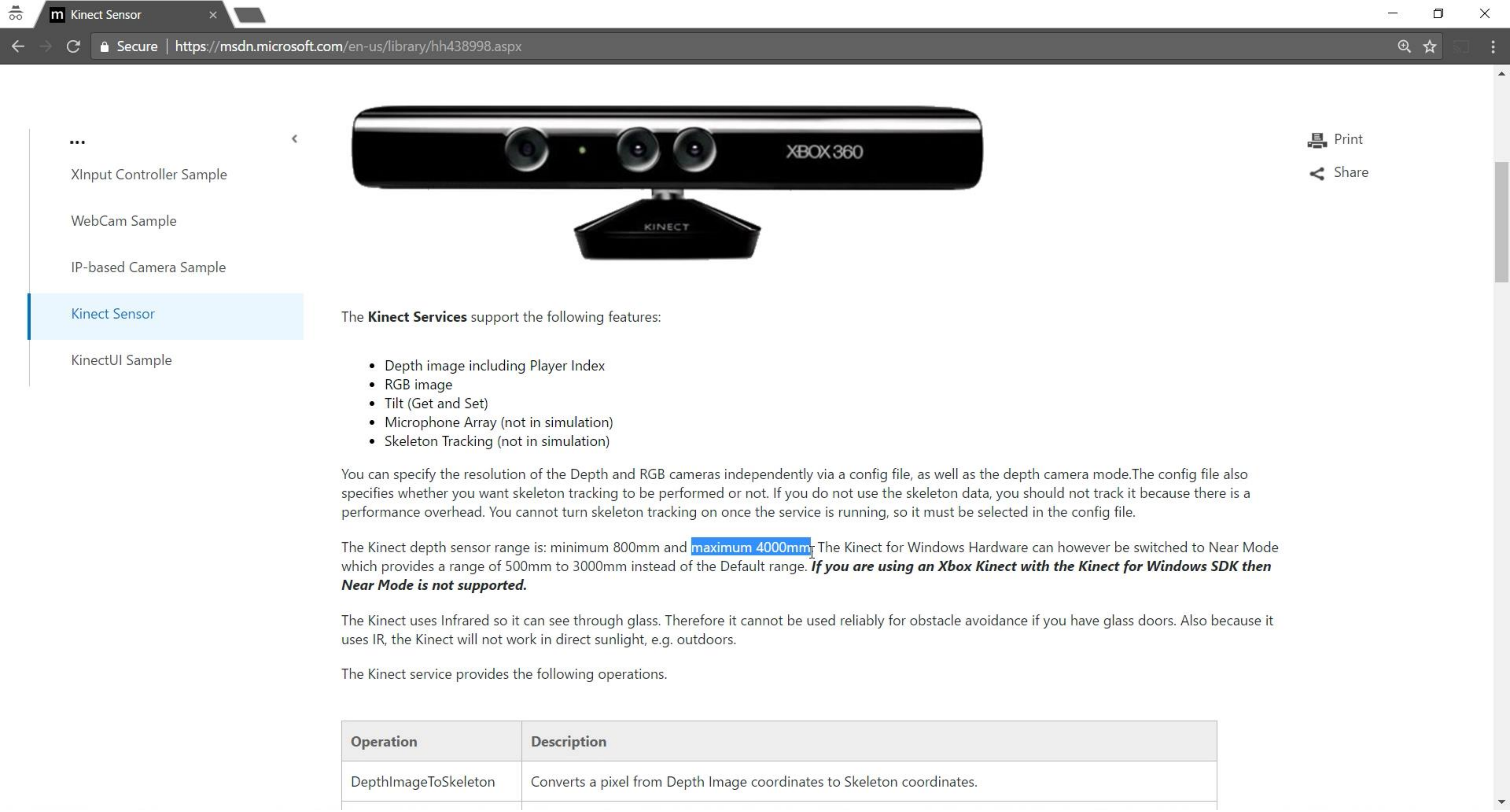


```
                    z_disp = Math.Pow(z_pos - z_prev, 2);  //1 ft = 304.8mm
                    double speed = ((Math.Sqrt(Math.Pow(x_disp, 2) + Math.Pow(y_disp, 2) + Math.Pow(z_disp, 2))) / 304.8 / 304.8 * (10));

                    this.speed_val_label.Text = speed.ToString();

                    x_prev = x_pos;
                    y_prev = y_pos;
                    z_prev = z_pos;

                    // write lines of text to the file
                    datafile.WriteLine(x_pos.ToString() + "," + y_pos.ToString() + "," + z_pos.ToString() + "," + speed.ToString());
                }
            }

            imageBox1.Image = originalImage;
            //imageBox2.Image = upper_red_hue_range;
        }


        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void FinalVideoSource_NewFrame(object sender, NewFrameEventArgs eventArgs)
        {
            //throw new NotImplementedException();
            Bitmap image = (Bitmap)eventArgs.Frame.Clone(); //bitmap boxing
            pictureBox1.Image = image;
        }
```



Name	Date modified	Type	Size
.vs	5/14/2017 2:59 PM	File folder	
packages	5/14/2017 2:59 PM	File folder	
WindowsFormsApplication3	5/15/2017 8:02 PM	File folder	
data	5/15/2017 8:04 PM	Text Document	1 KB
README	5/15/2017 2:59 PM	MD File	1 KB
videoRadar	5/15/2017 2:59 PM	Microsoft Visual St...	1 KB

```
269,257,987,3.15873621651205
313,261,993,4.72236309279518
337,249,976,2.81911363676501
331,249,965,0.623366490165137
329,249,973,0.20957894712449
327,251,984,0.299861158578307
325,251,984,0.211833757000847
317,253,984,0.873412955185918
299,255,976,1.90265330949873
265,235,1004,4.26376895566071
297,237,1001,3.45462352843835
293,239,998,0.480414931234124
293,241,998,0.214847651917526
307,245,987,1.5469296062396
281,241,959,2.71676046997455
285,235,965,0.74903976034338
333,251,1047,5.7478749635645
243,203,861,10.160120499104
257,205,868,1.32132075747346
281,223,923,2.99825961695177
311,245,1047,4.50746332469496
325,253,1155,2.36535650755548
335,255,1216,1.3936087027243
331,249,1159,0.965196990130406
329,245,1114,0.578859694535399
329,239,1063,0.741414563086857
327,231,1022,0.925011864886776
323,223,993,0.960290436380615
261,257,1076,8.22318960841517
277,269,1076,2.31639352167593
```



The **Kinect Services** support the following features:

- Depth image including Player Index
- RGB image
- Tilt (Get and Set)
- Microphone Array (not in simulation)
- Skeleton Tracking (not in simulation)

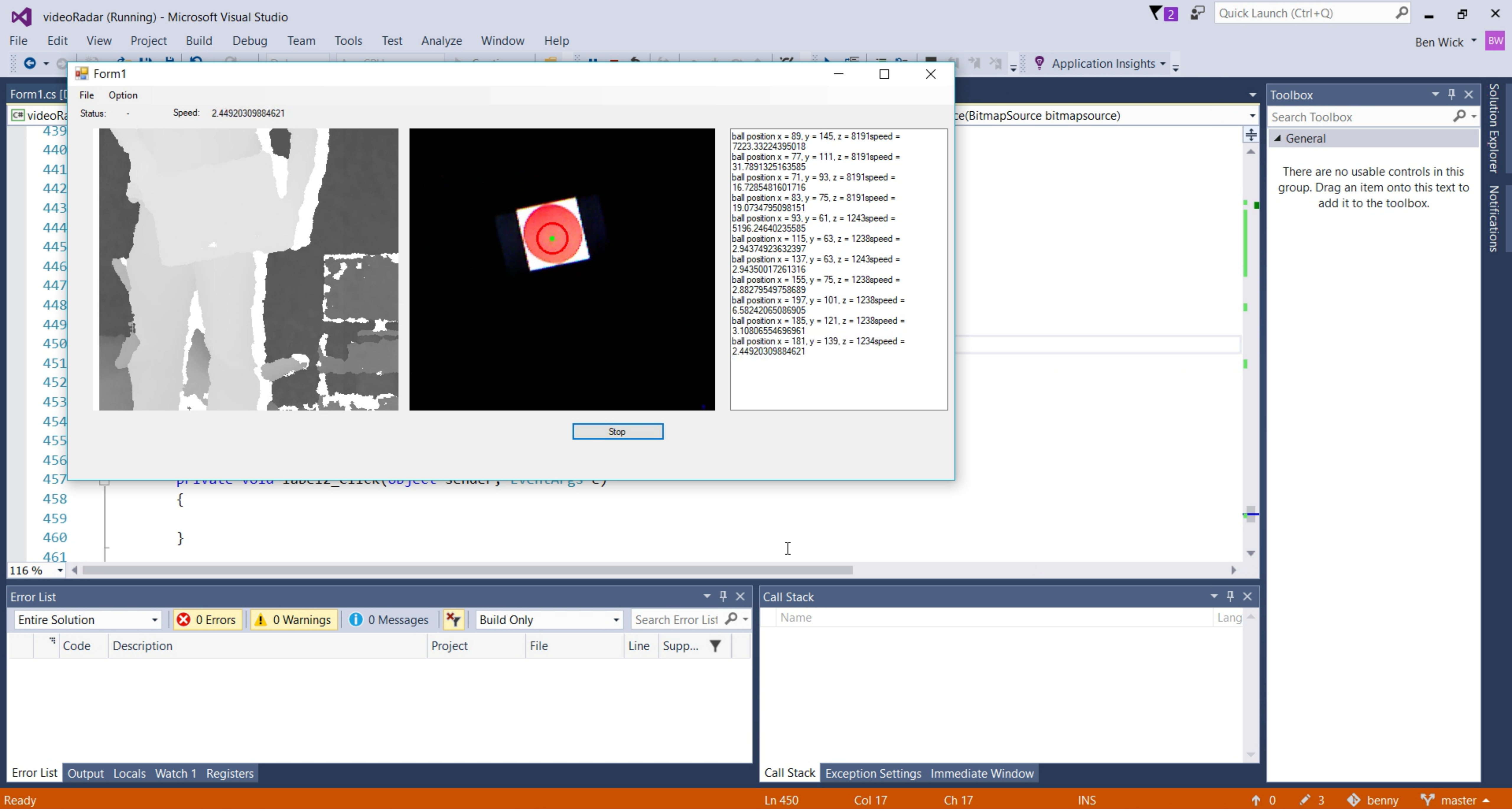
You can specify the resolution of the Depth and RGB cameras independently via a config file, as well as the depth camera mode.The config file also specifies whether you want skeleton tracking to be performed or not. If you do not use the skeleton data, you should not track it because there is a performance overhead. You cannot turn skeleton tracking on once the service is running, so it must be selected in the config file.

The Kinect depth sensor range is: minimum 800mm and maximum 4000mm. The Kinect for Windows Hardware can however be switched to Near Mode which provides a range of 500mm to 3000mm instead of the Default range. ***If you are using an Xbox Kinect with the Kinect for Windows SDK then Near Mode is not supported.***

The Kinect uses Infrared so it can see through glass. Therefore it cannot be used reliably for obstacle avoidance if you have glass doors. Also because it uses IR, the Kinect will not work in direct sunlight, e.g. outdoors.

The Kinect service provides the following operations.

Operation	Description
DepthImageToSkeleton	Converts a pixel from Depth Image coordinates to Skeleton coordinates.



Status: -

Speed: 2.44920309884621

```
ball position x = 89, y = 145, z = 8191speed =
7223.33224395018
ball position x = 77, y = 111, z = 8191speed =
31.7891325163585
ball position x = 71, y = 93, z = 8191speed =
16.7285481601716
ball position x = 83, y = 75, z = 8191speed =
19.0734795098151
ball position x = 93, y = 61, z = 1243speed =
5196.24640235585
ball position x = 115, y = 63, z = 1238speed =
2.94374923632397
ball position x = 137, y = 63, z = 1243speed =
2.94350017261316
ball position x = 155, y = 75, z = 1238speed =
2.88279549758689
ball position x = 197, y = 101, z = 1238speed =
6.58242065086905
ball position x = 185, y = 121, z = 1238speed =
3.10806554696961
ball position x = 181, y = 139, z = 1234speed =
2.44920309884621
```

```
        private void label2_Click(object sender, EventArgs e)
        {

        }
```

COLLEGE OF ENGINEERING

Electrical Engineering & Computer Science

Video Radar

Calculating the speed of an object in real time using a video camera.

Considered approaches

To build our system and application, we started off knowing we were going to need a camera, computer vision library, and:

- OpenCV
- ZED Stereo Camera: Two cameras side-by-side that parse depth by differences between the two images.

The Kinect

The Kinect is a camera device that was made by Microsoft for use with video games for their Xbox 360 games console. The Kinect has 2 cameras and a infrared laser projector. The first camera is a color camera for normal color view. The second camera is an infrared camera. The infrared projector projects a pattern of dots in front of it that the infrared camera then picks up. The Kinect then determines the distance depending on the pattern and spread of the dots.

Possible Applications

Our project could have several application in real life, tracking various subject, here are af few:

- Cars
- Athletes
- Sporting objects
- Analyzing crowds
- Animals

The Problem

The main method for people to track the speed of an object as they go by is to use a radar gun or laser gun, but unfortunately these methods have issues, they can only be used to check the speed of one object, the user can not tell exactly which object is being measured if there are multiple close together, and they must have an active user.

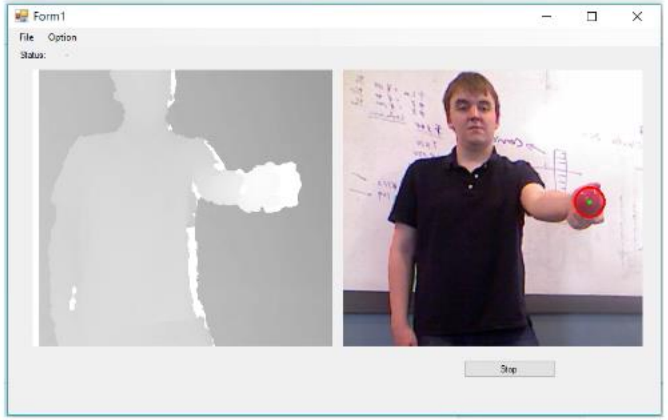
Our Process

First the Kinect camera calculates the depth map based on the infrared projections. Then it passes both the depth map image and the color image to our program. We then run an Emgu CV detection algorithm which identifies the red ball and its center. We then look up the distance to the center of the ball on the depth map. Next, we calculate the real world coordinates of the ball based on its distance from the camera. Finally, we calculate velocity based on change in position vs change in time.

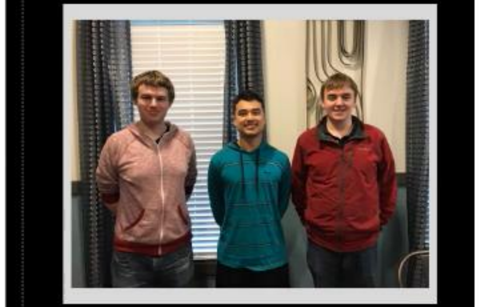
The Solution

Our solution to this problem is to use a camera in order to overcome these issues. For our proof of concept, we are using a Kinect camera. The use of the kinect allows us to measure the distance from the camera to an object so that we can track the velocity of objects. Since the video is being displayed the user can also see which object is going what velocity.

Our solution consists of a Graphical User Interface (GUI) application, which is made very simple for ease of use, that has a box that displays the live camera feed, where the objects that are being tracked are surrounded by a colored box. The velocity of the object is displayed in the text box to the side of the GUI. There is only one button that starts the camera and object tracking and stops it when clicked again. Our solution will also generate a comma separated text file so the user can review the data at a later date.



Ball identification with depth map



Team members:
Alexander Bailey (baileyal@oregonstate.edu)
Benjamin Wick (wickbe@oregonstate.edu)
Dylan Washburne (washburd@oregonstate.edu)

Project Advisors:
Professor Kevin McGrath
Professor Kirsten Winters
Jon Dodge, Teaching Assistant

Sponser:
Alex Neighbors

Conclusion

- There are many different applications for this kind of system
- Efficient tracking methods are difficult to create
- To track different types of objects within one application there would have to be a different algorithm for each object which would be difficult
- Computer vision has endless possibilities

Oregon State UNIVERSITY