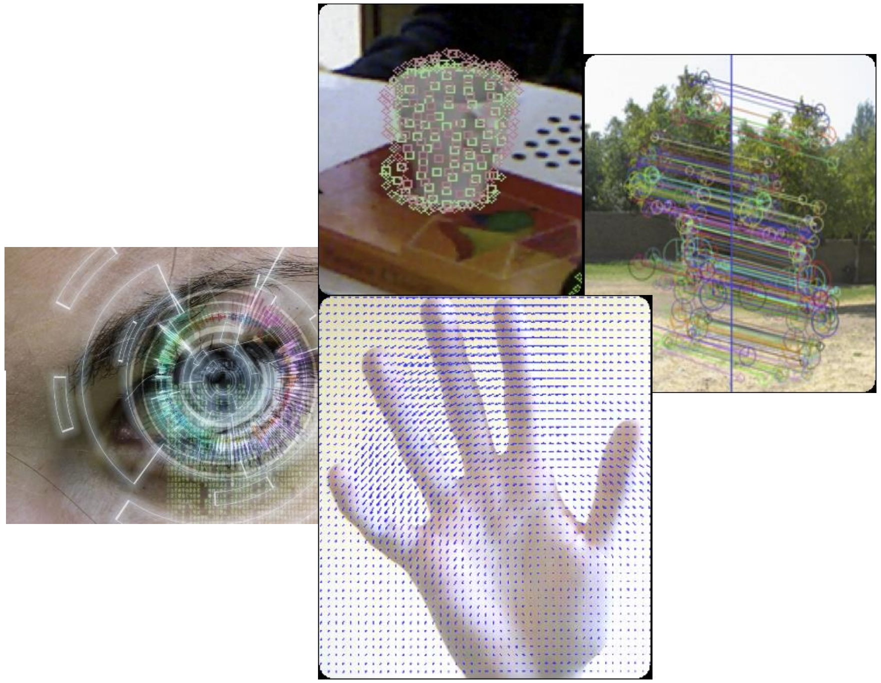
Intelligent Vision Processing Lab

COMPUTER VISION 비젼 프로그래밍

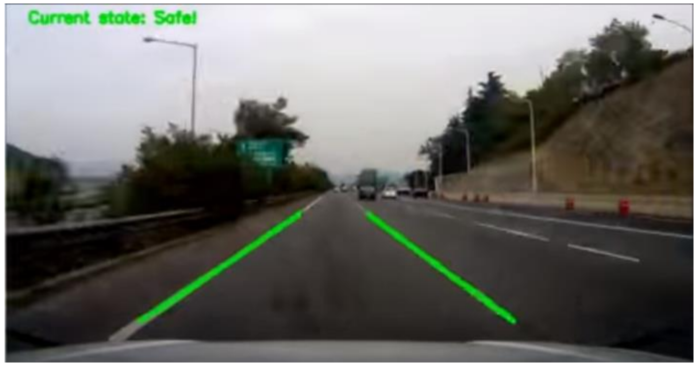


8장. point, line, contour extraction (Practice) (점, 선, 외곽선 추출)(실습)

Sub-Project #2

- Automatic Lane Detection For Smart Car
  - 목표: 주행 중 도로 영상에서 실시간 차선 검출
  - 주요 사용 기능: Hough Transform 기법

- 결과 발표
  - 11월 11일 월요일 수업 시간
- 주요 평가항목
  - 효과적인 차선 검출 결과(기본)
  - 차선 이탈 시 경고는 어떻게??(추가점)



Canny Edge Extractor

- Canny 연산자 API
  - void Canny(InputArray **image**, OutputArray **edges**, double **threshold1**, double **threshold2**, int **apertureSize**=3, bool **L2gradient**=false)

Parameters:	•**image** – single-channel 8-bit input image. •**edges** – output edge map; it has the same size and type as image . •**threshold1** – first threshold for the hysteresis procedure. •**threshold2** – second threshold for the hysteresis procedure. •**apertureSize** – aperture size for the Sobel() operator. •**L2gradient** – a flag, indicating whether a more accurate norm shoul d be used to calculate the image gradient magnitude (L2gradient=tr ue), or whether the default norm is enough (L2gradient=false).

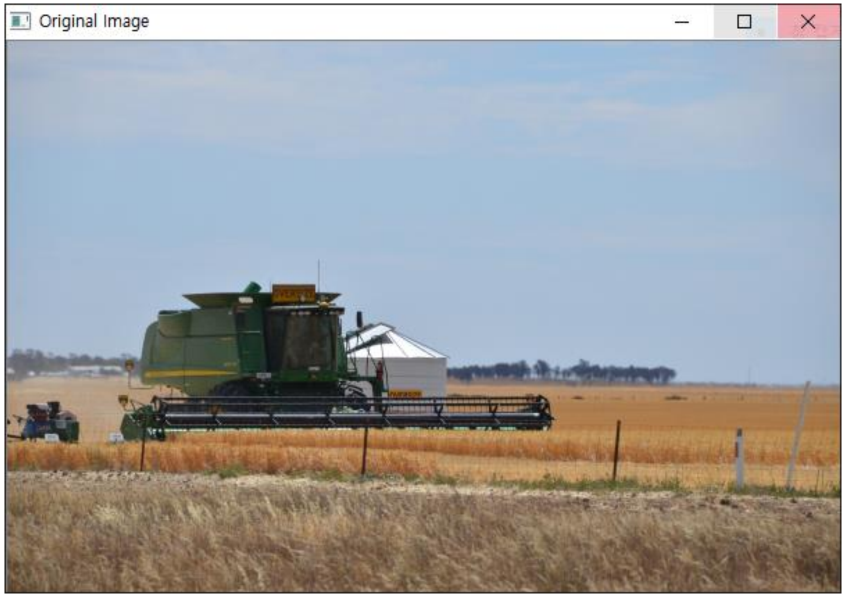
Canny Edge Extractor

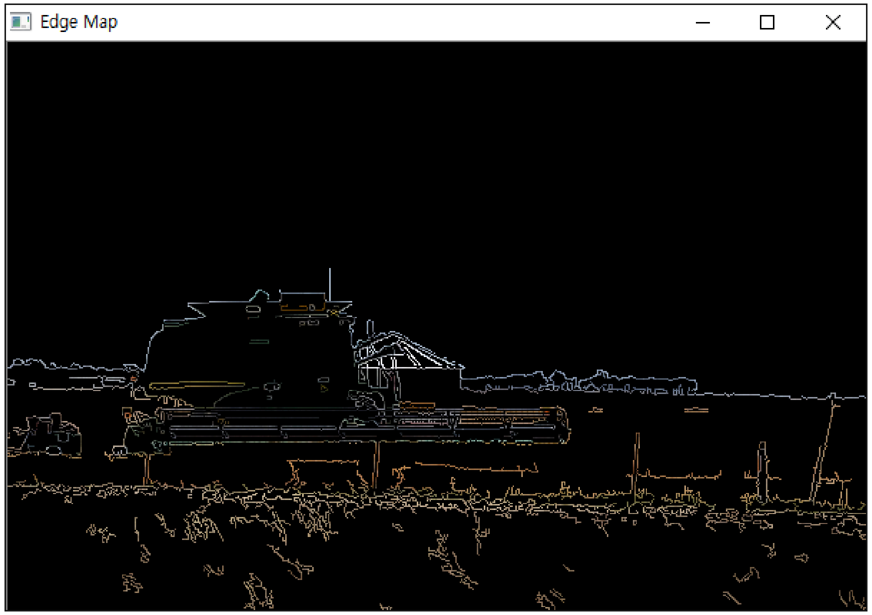
- Canny 연산자 기본 예제

```
int main( int argc, char** argv )
{
            Mat image, result, dst, dst1;
            int ratio = 3, lowThreshold=100;
            int kernel_size = 3;
            char* window_name = "Edge Map";

            /// Load image
            image = imread( "test.jpg", 1);    // Read the file

             if (image.empty()){                                    // Check for invalid input
                       cout << "Could not open or find the image" << std::endl;
                       return -1;
             }

             // Display the image
           cv::namedWindow("Original Image");
           cv::imshow("Original Image",image);

            /// Canny detector
            Canny( image, dst, lowThreshold, lowThreshold*ratio, kernel_size );

            /// Using Canny's output as a mask, we display our result
            dst1 = Scalar::all(0);

            image.copyTo( dst1, dst);
            imshow( window_name, dst1 );

            waitKey(0);

            return 0;
  }
```

Canny Edge Extractor

- 수행 결과:





Canny Edge Extractor

- Canny 연산자 기본 예제 with Trackbar

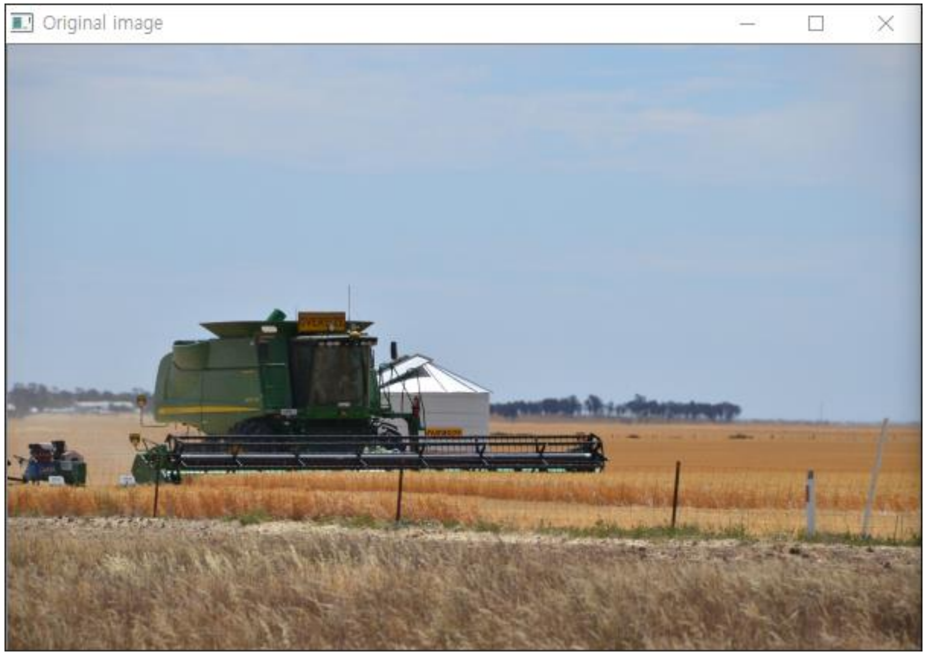
```
/// Global variables
Mat src, src_gray;
Mat dst, detected_edges;

int edgeThresh = 1;
int lowThreshold;
int const max_lowThreshold = 100;
int ratio = 3;
int kernel_size = 3;
char window_name[] = "Edge Map";

/**
 * @function CannyThreshold
 * @brief Trackbar callback - Canny thresholds input with a ratio 1:3
 */
void CannyThreshold(int, void*)
{
        /// Reduce noise with a kernel 3x3
        blur( src_gray, detected_edges, Size(3,3) );

        /// Canny detector
        Canny( detected_edges, detected_edges, lowThreshold, lowThreshold*ratio, kernel_size );

        /// Using Canny's output as a mask, we display our result
        dst = Scalar::all(0);

        src.copyTo( dst, detected_edges);
        imshow( window_name, dst );
 }
```

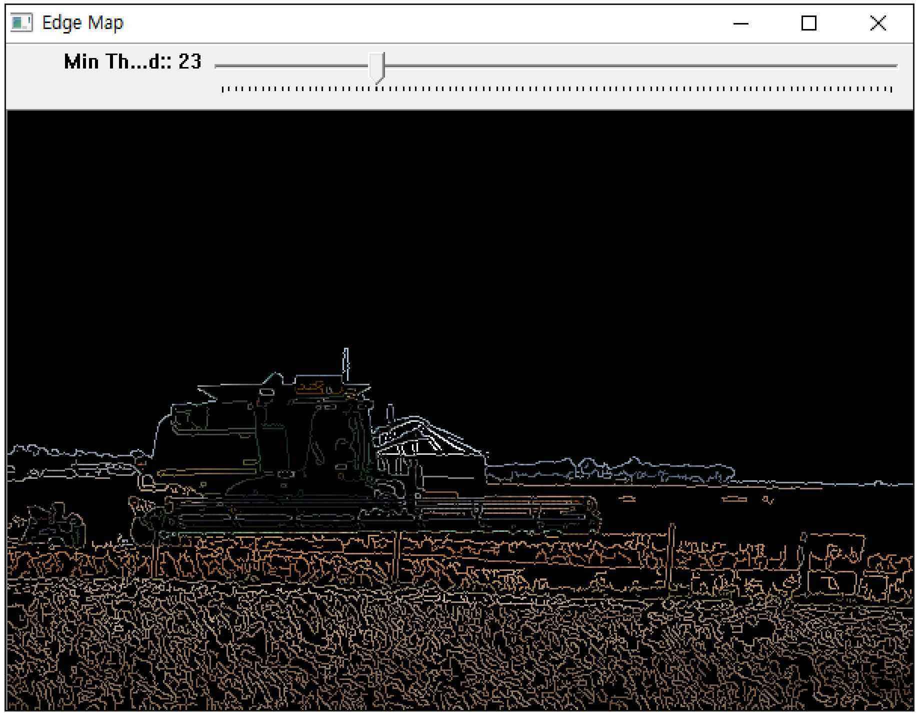
Canny Edge Extractor

```
int main( int argc, char** argv )
{
        /// Load an image
        src = imread("test.jpg",1 );

        if( !src.data )  { return -1; }

        ///Display the original image
        namedWindow("Original image");
        imshow("Original image",src);

        /// Create a matrix of the same type and size as src (for dst)
        dst.create( src.size(), src.type() );

        /// Convert the image to grayscale
        cvtColor( src, src_gray, COLR_BGR2GRAY );

        /// Create a window
        namedWindow( window_name, WINDOW_AUTOSIZE );

        /// Create a Trackbar for user to enter threshold
        createTrackbar( "Min Threshold:", window_name, &lowThreshold, max_lowThreshold, CannyThreshold );

        /// Show the image
        CannyThreshold(0, 0);

        /// Wait until user exit program by pressing a key
        waitKey(0);
         return 0;
  }
```

Canny Edge Extractor

- 수행 결과: 스크롤에 따른 결과 변화 관찰.....!!!!





Min Th...d:: 23

Hough Transform(허프 변환)

HoughCircles

- Finds circles in **a grayscale image** using the Hough transform.
- void HoughCircles(InputArray **image**, OutputArray **circles**, int **method**, double **dp**, double **minDist**, double **param1**=100, double **param2**=100, int **minRadius**=0, int **maxRadius**=0)

Parameters:	•**image** – 8-bit, single-channel, grayscale input image. •**circles** – Output vector of found circles. Each vector is encoded as a 3-element floati ng-point vector . •**circle_storage** – In C function this is a memory storage that will contain the output s equence of found circles. •**method** – Detection method to use. Currently, the only implemented method is CV_ HOUGH_GRADIENT , which is basically *21HT* , described in [Yuen90]. •**dp** – Inverse ratio of the accumulator resolution to the image resolution. For exampl e, if dp=1 , the accumulator has the same resolution as the input image. If dp=2 , the accumulator has half as big width and height. •**minDist** – Minimum distance between the centers of the detected circles. If the para meter is too small, multiple neighbor circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed. •**param1** – First method-specific parameter. In case of CV_HOUGH_GRADIENT , it is th e higher threshold of the two passed to the Canny() edge detector (the lower one is t wice smaller). •**param2** – Second method-specific parameter. In case of CV_HOUGH_GRADIENT , it is the accumulator threshold for the circle centers at the detection stage. The smaller it is, the more false circles may be detected. Circles, corresponding to the larger accum ulator values, will be returned first. •**minRadius** – Minimum circle radius. •**maxRadius** – Maximum circle radius.

Hough Transform(허프 변환)

HoughLines

- Finds lines in **a binary image** using the standard Hough transform.
- void HoughLines(InputArray **image**, OutputArray **lines**, double **rho**, double **theta**, int **threshold**, double **srn**=0, double **stn**=0)

Parameters:	•**image** – 8-bit, single-channel binary source image. The image may be modified by the function. •**lines** – Output vector of lines. Each line is represented by a two-element vector . [0] is the dista nce from the coordinate origin (top-left corner of the image). [1] is the line rotation angle in radi ans ( ). •**rho** – Distance resolution of the accumulator in pixels. •**theta** – Angle resolution of the accumulator in radians. •**threshold** – Accumulator threshold parameter. Only those lines are returned that get enough vo tes ( ). •**srn** – For the multi-scale Hough transform, it is a divisor for the distance resolution rho . The co arse accumulator distance resolution is rho and the accurate accumulator resolution is rho/srn . If both srn=0 and stn=0 , the classical Hough transform is used. Otherwise, both these parameters should be positive. •**stn** – For the multi-scale Hough transform, it is a divisor for the distance resolution theta. •**method** – One of the following Hough transform variants: • **CV_HOUGH_STANDARD** classical or standard Hough transform. Every line is represen ted by two floating-point numbers , where is a distance between (0,0) point and the li ne, and is the angle between x-axis and the normal to the line. Thus, the matrix must be (the created sequence will be) of CV_32FC2 type • **CV_HOUGH_PROBABILISTIC** probabilistic Hough transform (more efficient in case if t he picture contains a few long linear segments). It returns line segments rather than t he whole line. Each segment is represented by starting and ending points, and the ma trix must be (the created sequence will be) of the CV_32SC4 type. • **CV_HOUGH_MULTI_SCALE** multi-scale variant of the classical Hough transform. The li nes are encoded the same way as CV_HOUGH_STANDARD.

Hough Transform(허프 변환)

HoughLinesP

- Finds line segments in **a binary image** using **the probabilistic** Hough transform.
- void HoughLinesP(InputArray **image**, OutputArray **lines**, double **rho**, double **theta**, int **threshold**, double **minLineLength**=0, double **maxLineGap**=0)

Parameters:	•**image** – 8-bit, single-channel binary source image. The image may be modified by the function. •**lines** – Output vector of lines. Each line is represented by a 4-element vector , where and are the ending points of each detected line segment. •**rho** – Distance resolution of the accumulator in pixels. •**theta** – Angle resolution of the accumulator in radians. •**threshold** – Accumulator threshold parameter. Only those lines are returned that get enough votes ( ). •**minLineLength** – Minimum line length. Line segments shorter than that are rejected. •**maxLineGap** – Maximum allowed gap between points on the same line to link them.

Hough Transform(허프 변환) – HoughLines 활용 실습

- HoughLines API 이용 실습

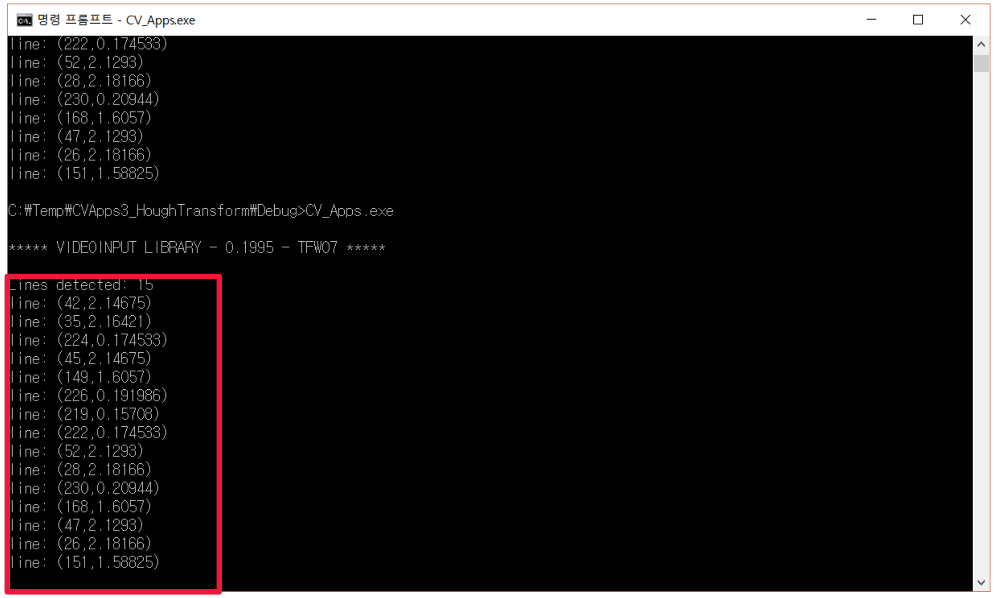
```
/// Function header
//constexpr auto PI = 3.14;
float const PI= 3.14;

int main( int argc, char** argv )
{
        cv::Mat image= cv::imread("road.jpg", 0);
        cv::namedWindow("Original Image");
        cv::imshow("Original Image",image);

         // 캐니 알고리즘 적용
        cv::Mat contours;
        cv::Canny(image, contours, 125, 350);

         // 선 감지 위한 허프 변환
        std::vector<cv::Vec2f> lines;
        cv::HoughLines(contours, lines,
        1, PI/180, // 단계별 크기
        80);  // 투표(vote) 최대 개수 ← 수에 따른 변화 관찰 필요 60, 40 등

         // 선 그리기
        cv::Mat result(contours.rows, contours.cols, CV_8U, cv::Scalar(255));
        std::cout << "Lines detected: " << lines.size() << std::endl;

        (계속)
```

Hough Transform(허프 변환) – HoughLines 활용 실습

```
        // 선 벡터를 반복해 선 그리기
        std::vector<cv::Vec2f>::const_iterator it= lines.begin();
        while (it!=lines.end()) {
                float rho = (*it)[0];   // 첫 번째 요소는 rho 거리
                float theta = (*it)[1]; // 두 번째 요소는 델타 각도
                if (theta < PI/4. || theta > 3.*PI/4.) { // 수직 행
                        cv::Point pt1(rho/cos(theta), 0); // 첫 행에서 해당 선의 교차점
                        cv::Point pt2((rho-result.rows*sin(theta))/cos(theta), result.rows);
                     // 마지막 행에서 해당 선의 교차점
                        cv::line(image, pt1, pt2, cv::Scalar(255), 1); // 하얀 선으로 그리기

                 } else { // 수평 행
                        cv::Point pt1(0,rho/sin(theta)); // 첫 번째 열에서 해당 선의 교차점
                        cv::Point pt2(result.cols,(rho-result.cols*cos(theta))/sin(theta));
                     // 마지막 열에서 해당 선의 교차점
                        cv::line(image, pt1, pt2, cv::Scalar(255), 1); // 하얀 선으로 그리기
                }
                std::cout << "line: (" << rho << "," << theta << ")₩n";
                ++it;
        }

         cv::namedWindow("Detected Lines with Hough");
        cv::imshow("Detected Lines with Hough",image);

         cv::waitKey(0);
         return 0;
    }
```

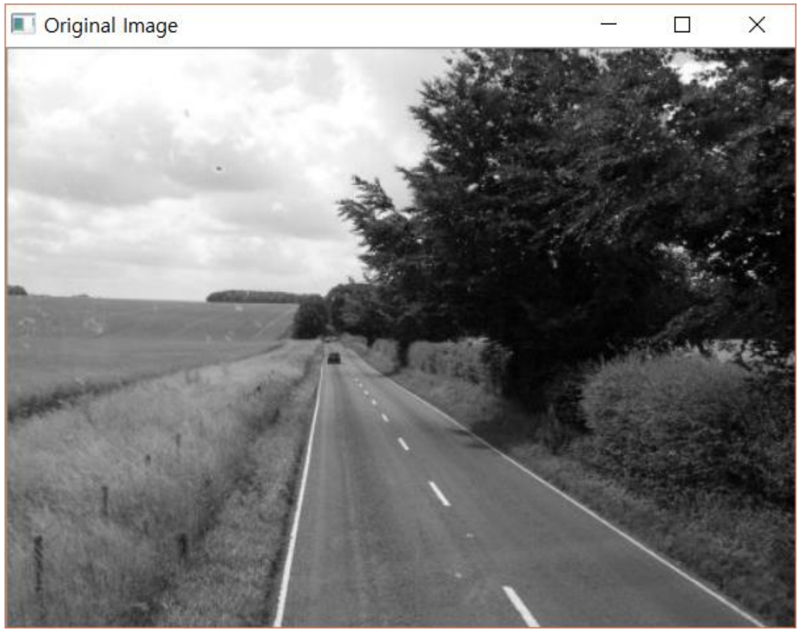
Hough Transform(허프 변환) - 실습

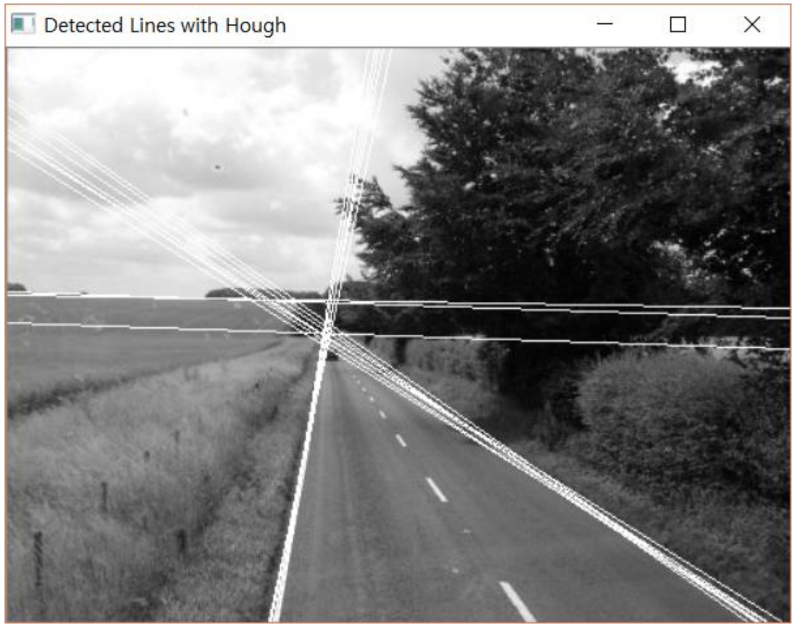
- 수행 결과:



```
line: (222,0.174533)
line: (52,2.1293)
line: (28,2.18166)
line: (230,0.20944)
line: (168,1.6057)
line: (47,2.1293)
line: (26,2.18166)
line: (151,1.58825)

C:\Temp\CVApps3_HoughTransform\Debug>CV_Apps.exe

***** VIDEOINPUT LIBRARY - 0.1995 - TFW07 *****

Lines detected: 15
line: (42,2.14675)
line: (35,2.16421)
line: (224,0.174533)
line: (45,2.14675)
line: (149,1.6057)
line: (226,0.191986)
line: (219,0.15708)
line: (222,0.174533)
line: (52,2.1293)
line: (28,2.18166)
line: (230,0.20944)
line: (168,1.6057)
line: (47,2.1293)
line: (26,2.18166)
line: (151,1.58825)
```

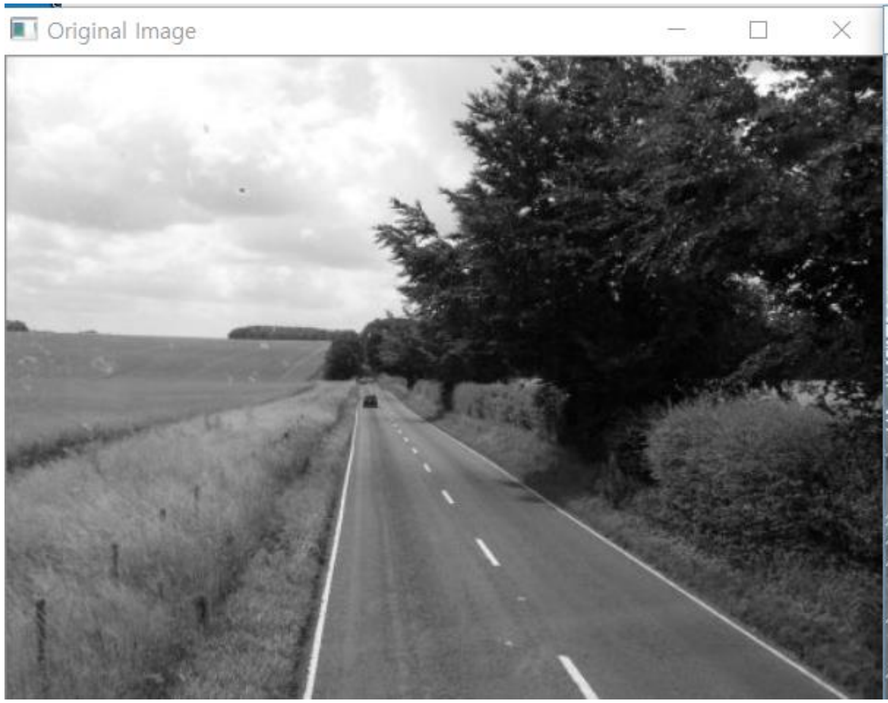
투표(vote) 최대 개수=80 설정

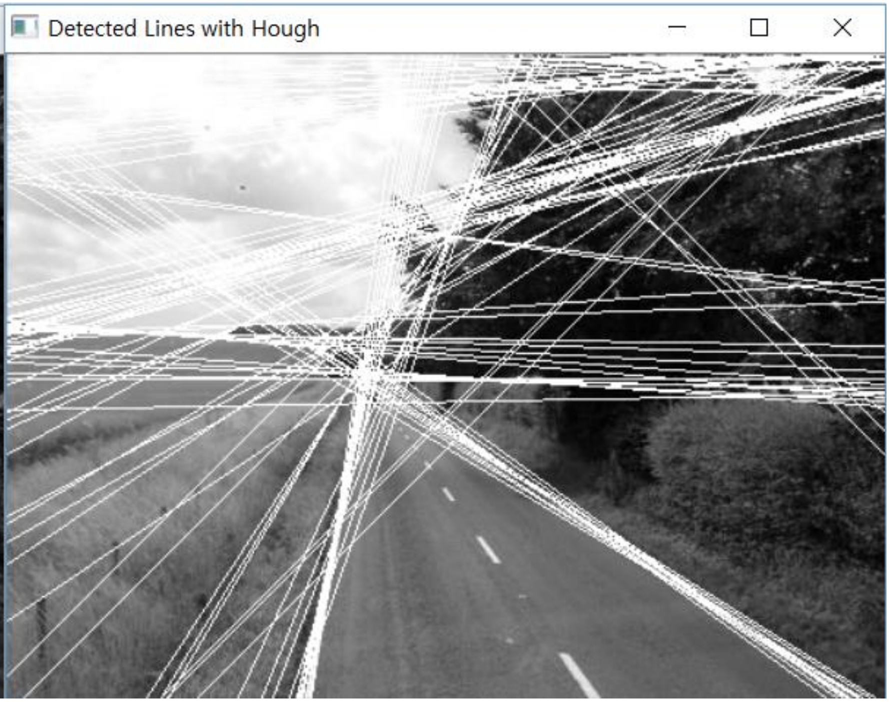




Hough Transform(허프 변환) - 실습

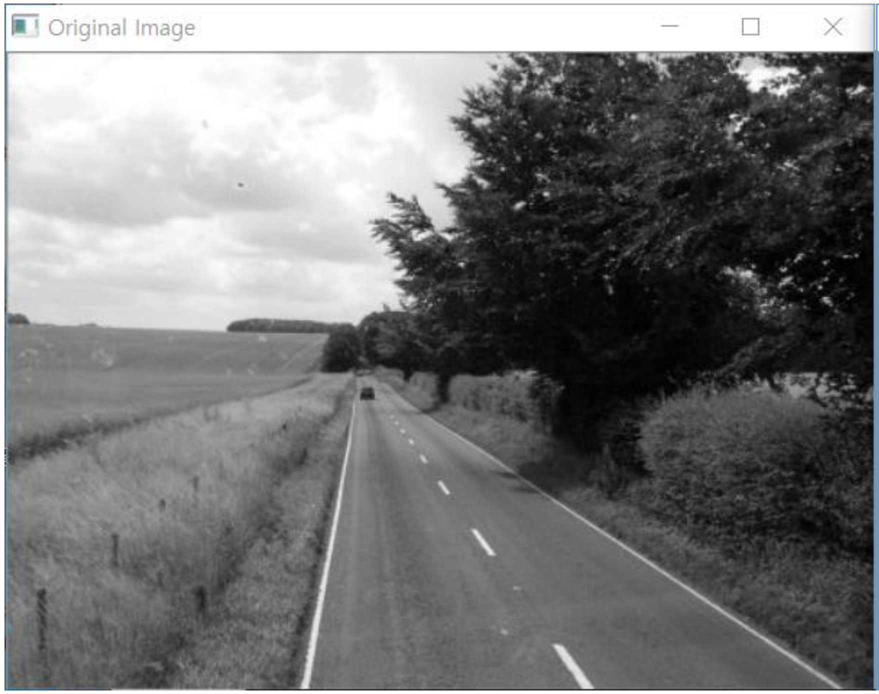
- 수행 결과:

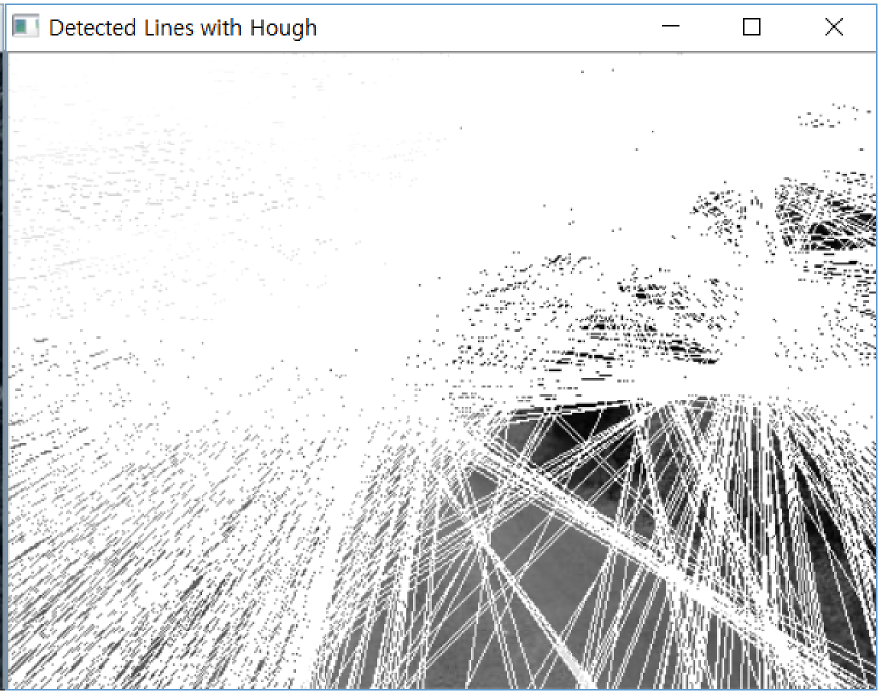




투표(vote) 최대 개수=60 설정

Hough Transform(허프 변환) - 실습





투표(vote) 최대 개수=40 설정

Hough Transform(허프 변환) – HoughCircles 활용 실습

- HoughCircles 활용 예제 코드

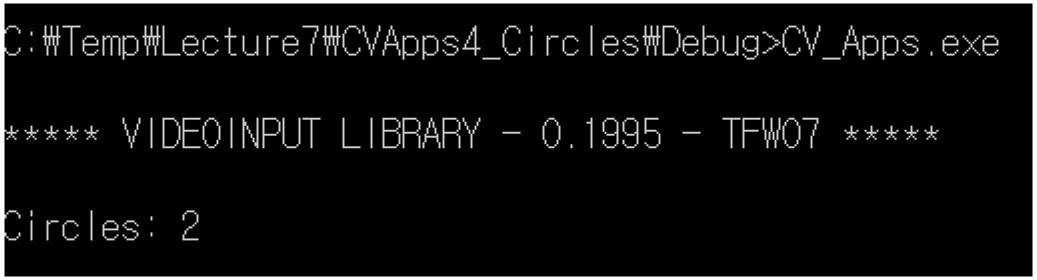
```
int main( int argc, char** argv )
{
        Mat image;
        // Detect circles
       image= cv::imread(argv[1],0);

       cv::GaussianBlur(image,image,cv::Size(5,5),1.5);

       std::vector<cv::Vec3f> circles;
       cv::HoughCircles(image, circles, HOUGH_GRADIENT,
       2,   // accumulator resolution (size of the image / 2)
       50,  // minimum distance between two circles
       200, // Canny high threshold
       100, // minimum number of votes
       25, 100); // min and max radius

       std::cout << "Circles: " << circles.size() << std::endl;

         (계속)
```

Hough Transform(허프 변환) – HoughCircles 활용 실습

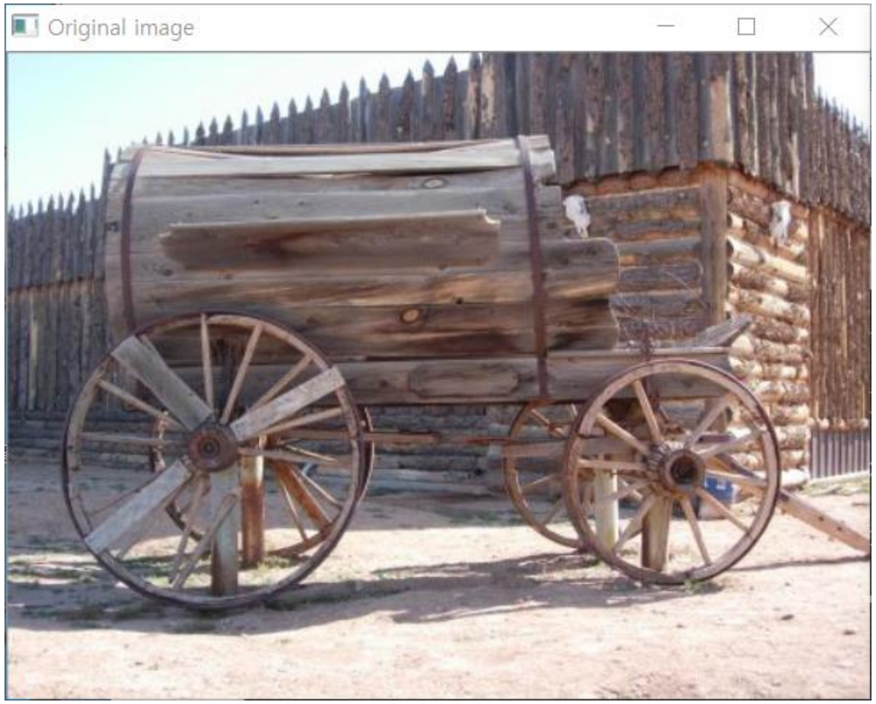
```
    // Draw the circles
    Mat image1= cv::imread(argv[1], 1);

    std::vector<cv::Vec3f>::const_iterator itc= circles.begin();

    while (itc!=circles.end()) {

      cv::circle(image1,
      cv::Point((*itc)[0], (*itc)[1]),          // circle centre
      (*itc)[2],                                //circle radius
      cv::Scalar(255),                          // color
      2);                                       // thickness

      ++itc;
    }

    cv::namedWindow("Detected Circles");
    cv::imshow("Detected Circles",image1);

    cv::waitKey();
    return 0;
}
```

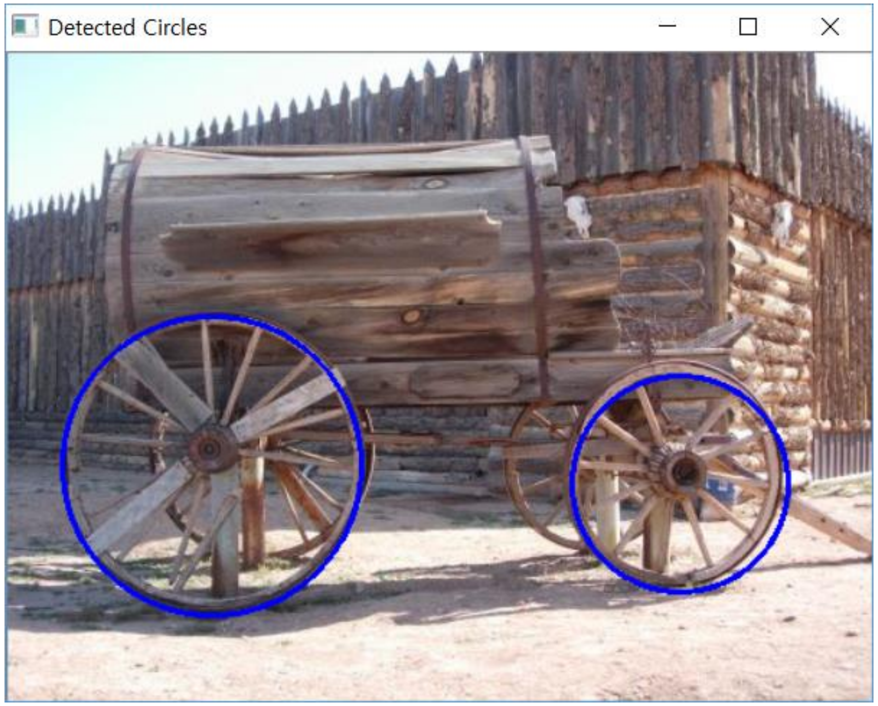
Hough Transform(허프 변환) – HoughCircles 활용 실습

- 수행 결과: 원 2개를 검출하였음.



```
C:\Temp\Lecture7\CVApps4_Circles\Debug>CV_Apps.exe

***** VIDEOINPUT LIBRARY - 0.1995 - TFW07 *****

Circles: 2
```





Hough Transform(허프 변환) – Contour 추출 실습

- Contour 검출 및 그리기
  - **findContours(~)**
    - void findContours(InputOutputArray **image**, OutputArrayOfArrays **contours**, OutputArray **hierarchy**, int **mode**, int **method**, Point **offset**=Point())

Parameters:

•**image** – Source, an 8-bit single-channel image. Non-zero pixels are treated as 1's. Zero pixels remain 0's, so the image is treated as binary . You can use compare() , inRange() , threshold() , adaptiveThreshold() , Canny() , and others to create a binary image out of a grayscale or color one. The function modifies the image while extracting the contours. If mode equals to CV_RETR_CCOMP or CV_RETR_FLOODFILL, the input can also be a 32-bit integer image of labels (CV_32SC1).
•**contours** – Detected contours. Each contour is stored as a vector of points.
•**hierarchy** – Optional output vector, containing information about the image topology. It has as many elements as the number of contours. For each i-th contour contours[i] , the elements hierarchy[i][0] , hiearchy[i][1] , hiearchy[i][2] , and hiearchy[i][3] are set to 0-based indices in contours of the next and previous contours at the same hierarchical level, the first child contour and the parent contour, respectively. If for the contour i there are no next, previous, parent, or nested contours, the corresponding elements of hierarchy[i] will be negative.
•**mode** – Contour retrieval mode (if you use Python see also a note below).

- **RETR_EXTERNAL** retrieves only the extreme outer contours. It sets hierarchy[i][2]=hierarchy[i][3]=-1 for all the contours.
- **RETR_LIST** retrieves all of the contours without establishing any hierarchical relationships.
- **RETR_CCOMP** retrieves all of the contours and organizes them into a two-level hierarchy. At the top level, there are external boundaries of the components. At the second level, there are boundaries of the holes. If there is another contour inside a hole of a connected component, it is still put at the top level.
- **RETR_TREE** retrieves all of the contours and reconstructs a full hierarchy of nested contours. This full hierarchy is built and shown in the OpenCV contours.c demo.

•**method** – Contour approximation method (if you use Python see also a note below).

- **CHAIN_APPROX_NONE** stores absolutely all the contour points. That is, any 2 subsequent points (x1,y1) and (x2,y2) of the contour will be either horizontal, vertical or diagonal neighbors, that is, max(abs(x1-x2),abs(y2-y1))==1.
- **CHAIN_APPROX_SIMPLE** compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, an up-right rectangular contour is encoded with 4 points.
- **CHAIN_APPROX_TC89_L1,CV_CHAIN_APPROX_TC89_KCOS** applies one of the flavors of the The-Chin chain approximation algorithm. See [TehChin89] for details.

•**offset** – Optional offset by which every contour point is shifted. This is useful if the contours are extracted from the image ROI and then they should be analyzed in the whole image context.

Hough Transform(허프 변환) – Contour 추출 실습

- drawContours(~)
  - void drawContours(InputOutputArray **image**, InputArrayOfArrays **contours**, int **contourIdx**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, InputArray **hierarchy**=noArray(), int **maxLevel**=INT_MAX, Point **offset**=Point())

Parameters:	•**image** – Destination image. •**contours** – All the input contours. Each contour is stored as a point vector. •**contourIdx** – Parameter indicating a contour to draw. If it is negative, all the contour s are drawn. •**color** – Color of the contours. •**thickness** – Thickness of lines the contours are drawn with. If it is negative (for exam ple, thickness=CV_FILLED), the contour interiors are drawn. •**lineType** – Line connectivity. See line() for details. •**hierarchy** – Optional information about hierarchy. It is only needed if you want to dr aw only some of the contours (see maxLevel). •**maxLevel** – Maximal level for drawn contours. If it is 0, only the specified contour is drawn. If it is 1, the function draws the contour(s) and all the nested contours. If it is 2, the function draws the contours, all the nested contours, all the nested-to-nested c ontours, and so on. This parameter is only taken into account when there is hierarchy available. •**offset** – Optional contour shift parameter. Shift all the drawn contours by the specifi ed . •**contour** – Pointer to the first contour. •**externalColor** – Color of external contours. •**holeColor** – Color of internal contours (holes).

Hough Transform(허프 변환) – Contour 추출 실습

- findContour 활용 예제 코드

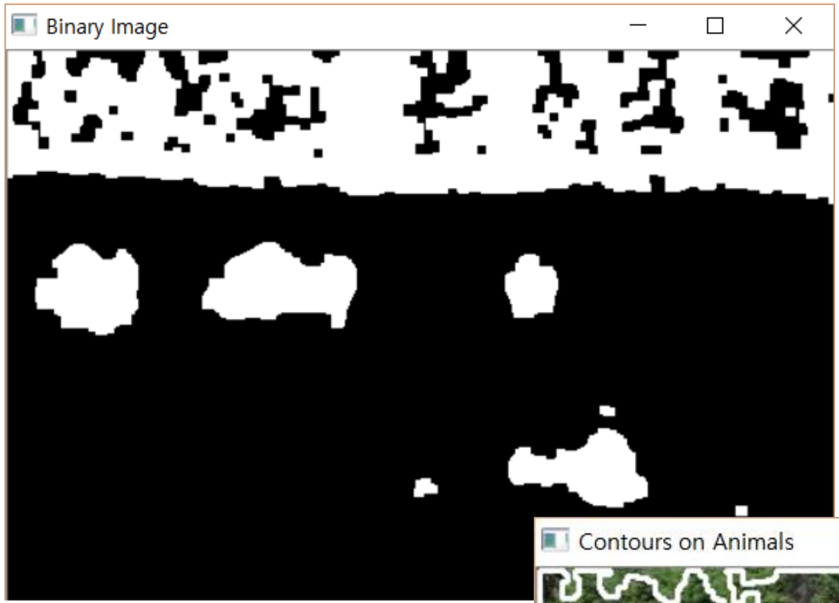
```
int main()
{
        // Read input binary image
        cv::Mat image= cv::imread("binaryGroup.bmp",0);
        if (!image.data)
        return 0;

        cv::namedWindow("Binary Image");
        cv::imshow("Binary Image",image);

        // Get the contours of the connected components
        std::vector<std::vector<cv::Point>> contours;
        cv::findContours(image,
                contours, // a vector of contours
                CV_RETR_EXTERNAL, // retrieve the external contours
                CV_CHAIN_APPROX_NONE); // retrieve all pixels of each contours

  (계속)
```

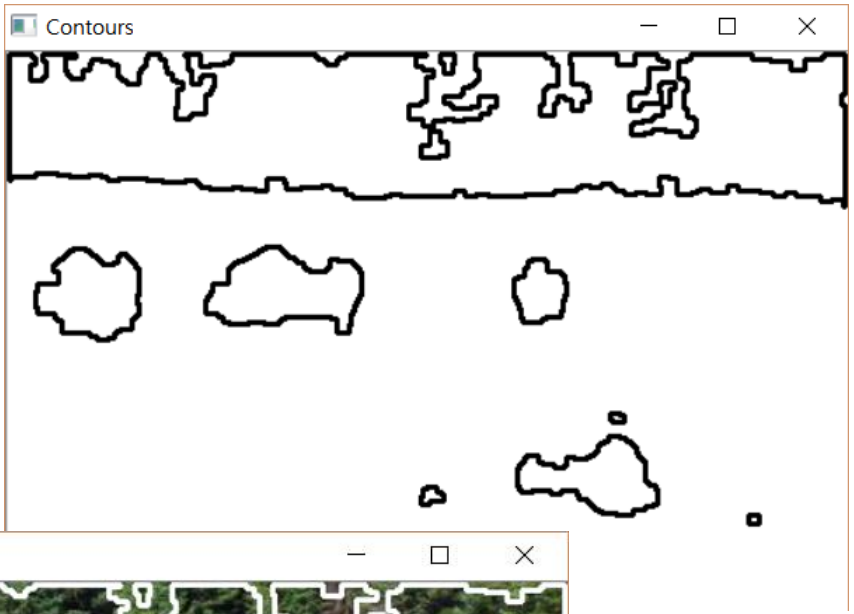
Hough Transform(허프 변환) – Contour 추출 실습

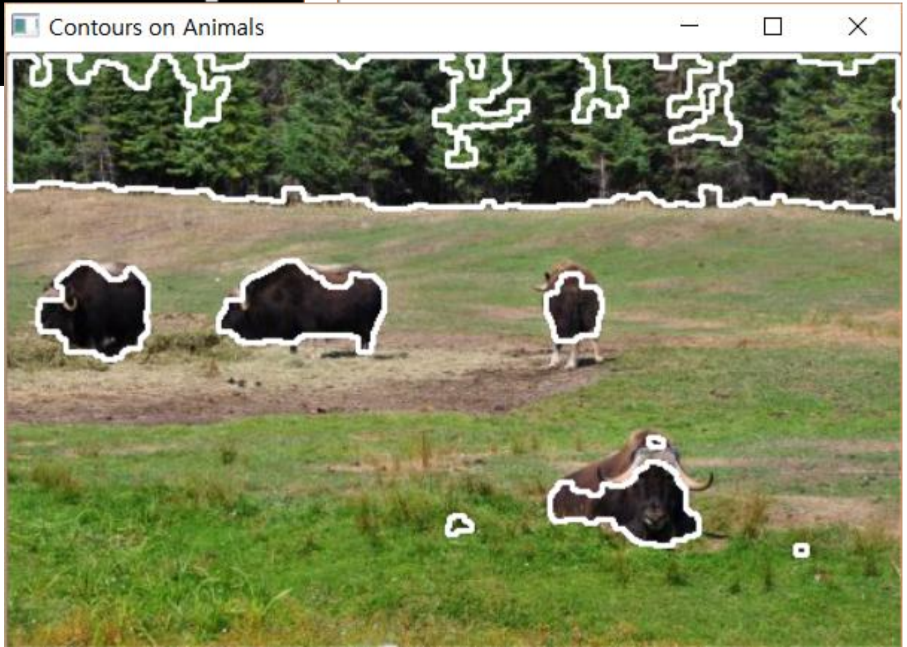
```
        // draw black contours on white image
        cv::Mat result(image.size(),CV_8U,cv::Scalar(255));
        cv::drawContours(result,contours,
        -1, // draw all contours
        cv::Scalar(0), // in black
        2); // with a thickness of 2

        cv::namedWindow("Contours");
        cv::imshow("Contours",result);
        // draw contours on the original image
        cv::Mat original= cv::imread("group.jpg",1);
        cv::drawContours(original,contours, -1, // draw all contours
                cv::Scalar(255,255,255), // in white
                2); // with a thickness of 2

        cv::namedWindow("Contours on Animals");
        cv::imshow("Contours on Animals",original);

        cv::waitKey(0);
        return 0;
}
```

Hough Transform(허프 변환) – Contour 추출 실습

- 수행 결과: 원 2개를 검출하였음.





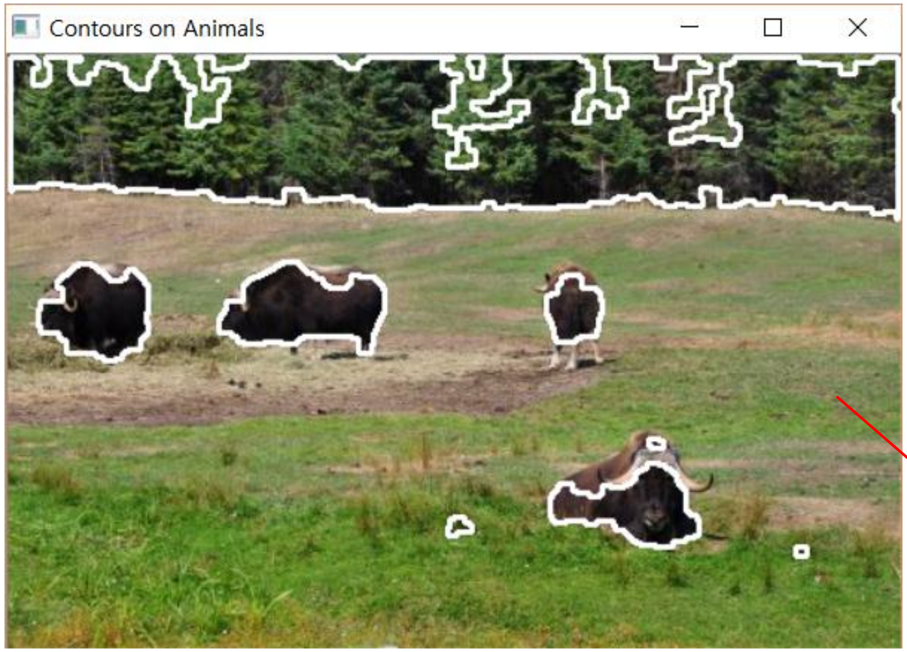


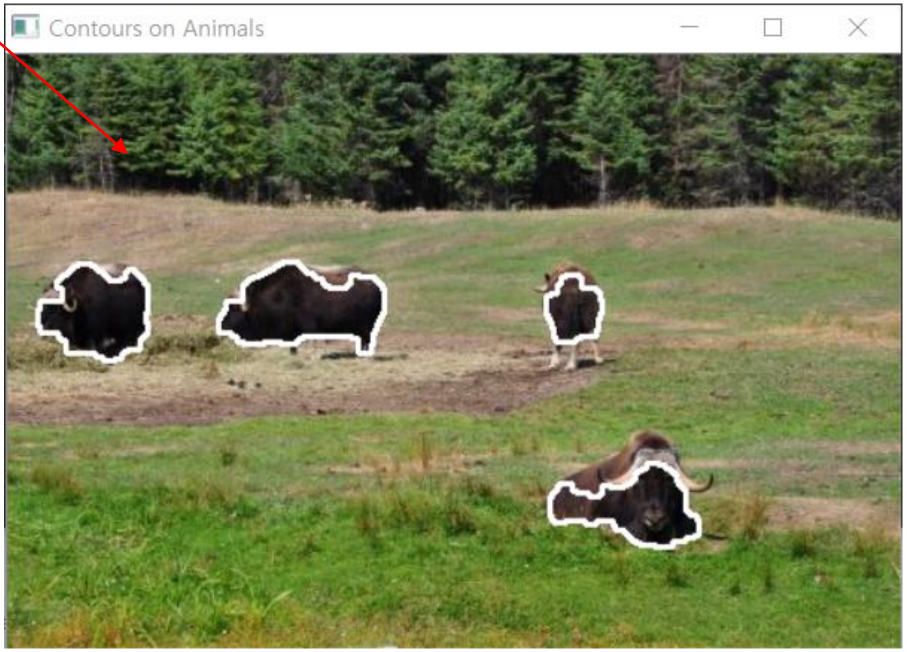
Hough Transform(허프 변환) – Contour 필터링 후 추출 실습

```
// draw black contours on white image
cv::Mat result(image.size(),CV_8U,cv::Scalar(255));
cv::drawContours(result,contours,
-1, // draw all contours
cv::Scalar(0), // in black
2); // with a thickness of 2

cv::namedWindow("Contours");
cv::imshow("Contours",result);

// Eliminate too short or too long contours
int cmin= 100;  // minimum contour length
int cmax= 1000; // maximum contour length
std::vector<std::vector<cv::Point>>::const_iterator itc= contours.begin();
while (itc!=contours.end()-1) {

        if (itc->size()-1 < cmin || itc->size()-1 > cmax)
                itc= contours.erase(itc);
        else
                ++itc;
}
```

Hough Transform(허프 변환) – Contour 필터링 후 추출 실습

- Contour 크기에 따른 필터링을 통한 잡음 제거 후 결과





Contour Descriptor 추출

- Contour Descriptor 추출 예제 코드

파일 입력(imread()함수) 시 여러분들
영상 파일 path를 확인해야 합니다.

```
int main()
{
        // Read input binary image
        cv::Mat image= cv::imread("binaryGroup.bmp",0);
        if (!image.data)
              return 0;

        cv::namedWindow("Binary Image");
        cv::imshow("Binary Image",image);

        // Get the contours of the connected components
        std::vector<std::vector<cv::Point>> contours;
        cv::findContours(image,
                contours, // a vector of contours
                CV_RETR_EXTERNAL, // retrieve the external contours
                CV_CHAIN_APPROX_NONE); // retrieve all pixels of each contours

   (계속)
```

Contour Descriptor 추출

```
// draw black contours on white image
cv::Mat result(image.size(),CV_8U,cv::Scalar(255));
cv::drawContours(result,contours,
-1, // draw all contours
cv::Scalar(0), // in black
2); // with a thickness of 2

cv::namedWindow("Contours");
cv::imshow("Contours",result);

// Eliminate too short or too long contours
int cmin= 100;  // minimum contour length
int cmax= 1000; // maximum contour length
std::vector<std::vector<cv::Point>>::const_iterator itc= contours.begin();
while (itc!=contours.end()-1) {

if (itc->size()-1 < cmin || itc->size()-1 > cmax)
itc= contours.erase(itc);
else
++itc;
}
```

Contour Descriptor 추출

(계속)
```
    // draw contours on the original image
    cv::Mat original= cv::imread("group.jpg",1);
    cv::drawContours(original,contours,
            -1, // draw all contours
            cv::Scalar(255,255,255), // in white
            2); // with a thickness of 2

    cv::namedWindow("Contours on Animals");
    cv::imshow("Contours on Animals",original);
```

<<새로운 descriptor 삽입>> (다음 ppt)

```
    cv::waitKey(0);
    return 0;
}
```

Contour Descriptor 추출: 사용 APIs

- 활용 APIs
  - **approxPolyDP**
  - **boundingRect**
  - **line**
  - **convexHull**
  - **moments**

Contour Descriptor 추출: 사용 APIs

- **approxPolyDP:** Approximates a polygonal curve(s) with the specified precision.
  - void approxPolyDP(InputArray **curve**, OutputArray **approxCurve**, double **epsilon**, bool **closed**)

Parameters:	•**curve** – Input vector of a 2D point stored in: • std::vector or Mat (C++ interface) • CvSeq or `` CvMat (C interface) •**approxCurve** – Result of the approximation. The type should match the type o f the input curve. In case of C interface the approximated curve is stored in the memory storage and pointer to it is returned. •**epsilon** – Parameter specifying the approximation accuracy. This is the maximu m distance between the original curve and its approximation. •**closed** – If true, the approximated curve is closed (its first and last vertices are connected). Otherwise, it is not closed. •**header_size** – Header size of the approximated curve. Normally, sizeof(CvCont our) is used. •**storage** – Memory storage where the approximated curve is stored. •**method** – Contour approximation algorithm. Only CV_POLY_APPROX_DP is sup ported. •**recursive** – Recursion flag. If it is non-zero and curve is CvSeq*, the function c vApproxPoly approximates all the contours accessible from curve by h_next and v_next links.

Contour Descriptor 추출: 사용 APIs

- **moments:** Calculates all of the moments up to the third order of a polygon or rasterized shape.
  - Moments **moments**(**array**, bool **binaryImage**=false)

Parameters:	•**array** – Raster image (single-channel, 8-bit or floating-point 2D array) or an array (or) of 2D points (Point or Point2f). •**binaryImage** – If it is true, all non-zero image pixels are treated as 1's. The parameter is used for images only. •**moments** – Output moments.

- Line:
  - void line(Mat& **img**, Point **pt1**, Point **pt2**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)

Parameters:	•**img** – Image. •**pt1** – First point of the line segment. •**pt2** – Second point of the line segment. •**color** – Line color. •**thickness** – Line thickness. •**lineType** – Type of the line: • **8** (or omitted) - 8-connected line. • **4** - 4-connected line. • **CV_AA** - antialiased line. •**shift** – Number of fractional bits in the point coordinates.

Contour Descriptor 추출: 사용 APIs

- **boundingRect:** Calculates the up-right bounding rectangle of a point set.
  - Rect **boundingRect**(InputArray **points**)

Parameters	•**curve** – Input vector of 2D points, stored in std::vector or Mat. •**closed** – Flag indicating whether the curve is closed or not.

- **convexHull:** Finds the convex hull of a point set.
  - void **convexHull**(InputArray **points**, OutputArray **hull**, bool **clockwise**=false, bool **returnPoints**=true)

Parameters:	•**points** – Input 2D point set, stored in std::vector or Mat. •**hull** – Output convex hull. It is either an integer vector of indices or vector of points. In the first case, the hull elements are 0-based indices of the convex hull points in the original array (since t he set of convex hull points is a subset of the original point set). In the second case, hull elemen ts are the convex hull points themselves. •**hull_storage** – Output memory storage in the old API (cvConvexHull2 returns a sequence contai ning the convex hull points or their indices). •**clockwise** – Orientation flag. If it is true, the output convex hull is oriented clockwise. Otherwise , it is oriented counter-clockwise. The assumed coordinate system has its X axis pointing to the ri ght, and its Y axis pointing upwards. •**orientation** – Convex hull orientation parameter in the old API, CV_CLOCKWISE or CV_COUNTER CLOCKWISE. •**returnPoints** – Operation flag. In case of a matrix, when the flag is true, the function returns co nvex hull points. Otherwise, it returns indices of the convex hull points. When the output array is std::vector, the flag is ignored, and the output depends on the type of the vector: std::vector<int > implies returnPoints=true, std::vector<Point> implies returnPoints=false.

Contour Descriptor 추출: 삽입코드 부분(1)

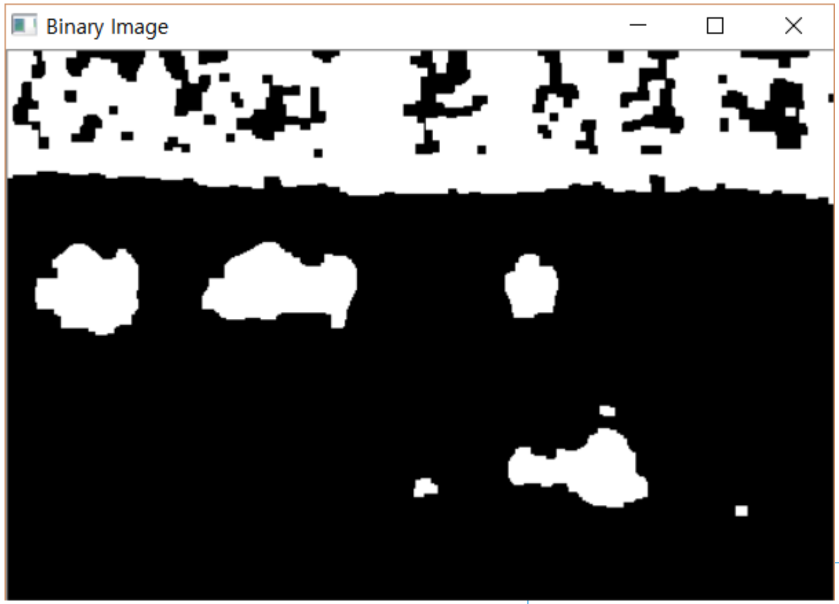
```
  (계속)
std::cout << " Step-2) Press any key to detect shape descriptor...!! " << std::endl;
cv::waitKey(0);

image= cv::imread("F:/Temp/Images/binaryGroup.bmp",0);

// testing the bounding box
cv::Rect r0= cv::boundingRect(cv::Mat(contours[0]));
cv::rectangle(result,r0,cv::Scalar(0),2);

// testing the enclosing circle
float radius;
cv::Point2f center;
cv::minEnclosingCircle(cv::Mat(contours[1]),center,radius);
cv::circle(result,cv::Point(center),static_cast<int>(radius),cv::Scalar(0),2);

// testing the approximate polygon
std::vector<cv::Point> poly;
cv::approxPolyDP(cv::Mat(contours[2]),poly,5,true);

std::cout << "Polygon size: " << poly.size() << std::endl;

// Iterate over each segment and draw it
std::vector<cv::Point>::const_iterator itp= poly.begin();
while (itp!=(poly.end()-1)) {
        cv::line(result,*itp,*(itp+1),cv::Scalar(0),2);
        ++itp;
}
// last point linked to first point
cv::line(result,*(poly.begin()),*(poly.end()-1),cv::Scalar(20),2);
 (계속)
```

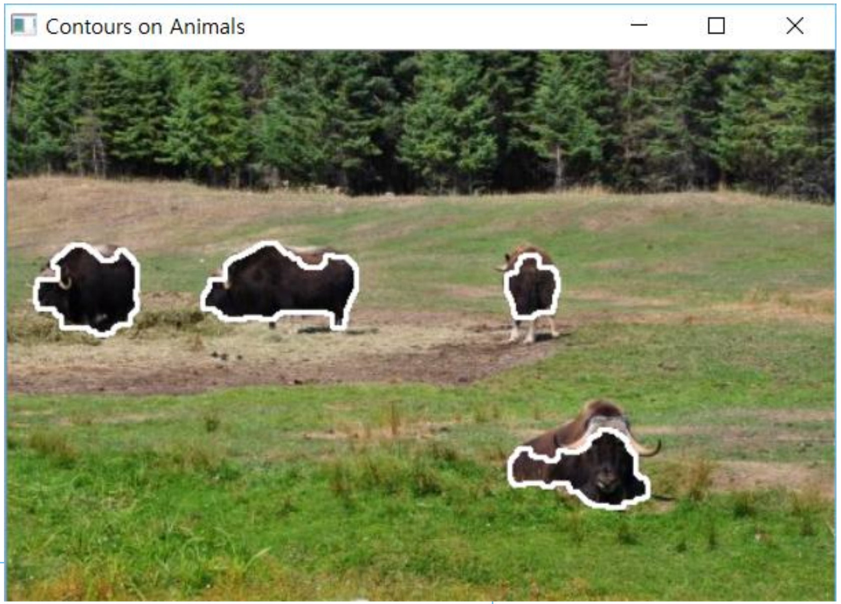
Contour Descriptor 추출: 삽입코드 부분(2)

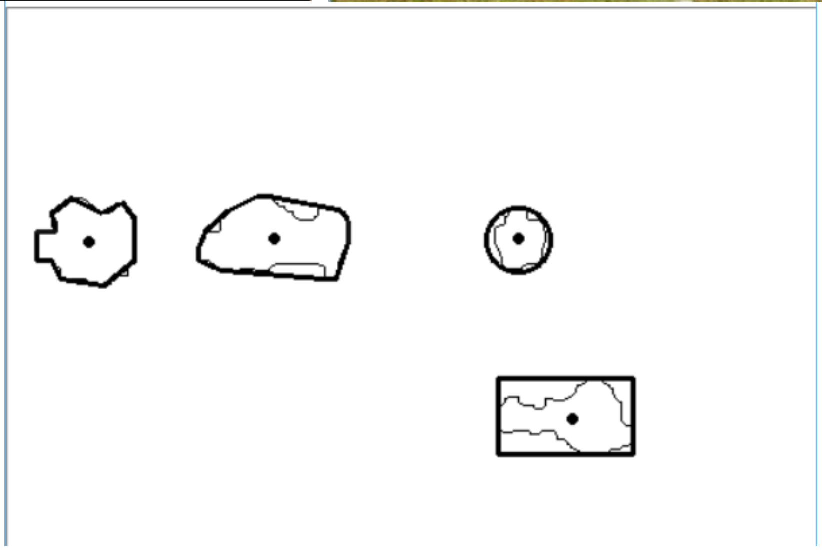
```
      (계속)
     // testing the convex hull
     std::vector<cv::Point> hull;
     cv::convexHull(cv::Mat(contours[3]),hull);

     // Iterate over each segment and draw it
     std::vector<cv::Point>::const_iterator it= hull.begin();
     while (it!=(hull.end()-1)) {
          cv::line(result,*it,*(it+1),cv::Scalar(0),2);
          ++it;
     }
     // last point linked to first point
     cv::line(result,*(hull.begin()),*(hull.end()-1),cv::Scalar(20),2);

     // testing the moments
     // iterate over all contours
     itc= contours.begin();
     while (itc!=contours.end()) {
          // compute all moments
          cv::Moments mom= cv::moments(cv::Mat(*itc++));

          // draw mass center
          cv::circle(result,  // position of mass center converted to integer
          cv::Point(mom.m10/mom.m00,mom.m01/mom.m00),
          2,cv::Scalar(0),2); // draw black dot
     }

     cv::namedWindow("Some Shape descriptors");
     cv::imshow("Some Shape descriptors",result);
     cv::waitKey();
     return 0;
}
```

Contour Descriptor 추출

- 수행 결과: 개별 객체에 대한 shape 형태가 다르게 추출될 수 있음을 알 수 있음

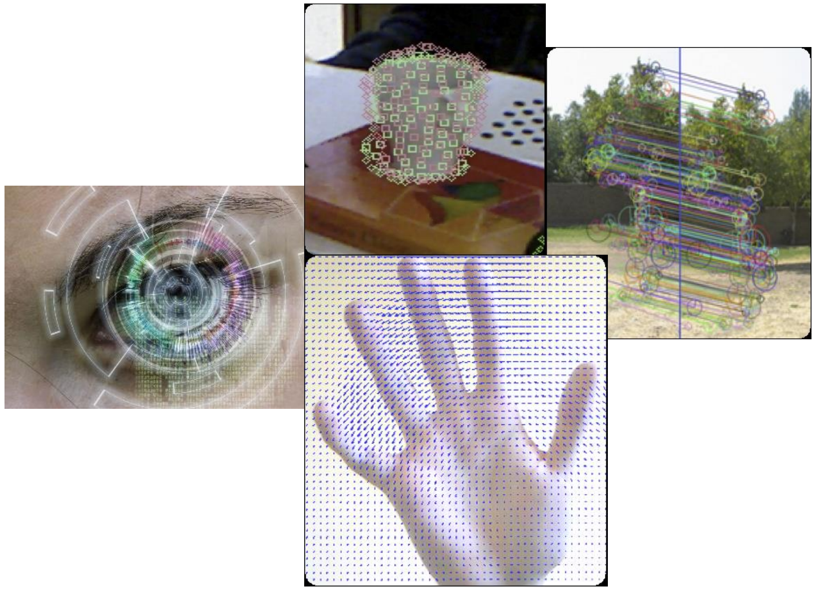






COMPUTER VISION 비전 프로그래밍

Thank you and Question?



IVPL
Intelligent Vision Processing Lab