Heidelberg Instruments Nano: Coding Assignment Report

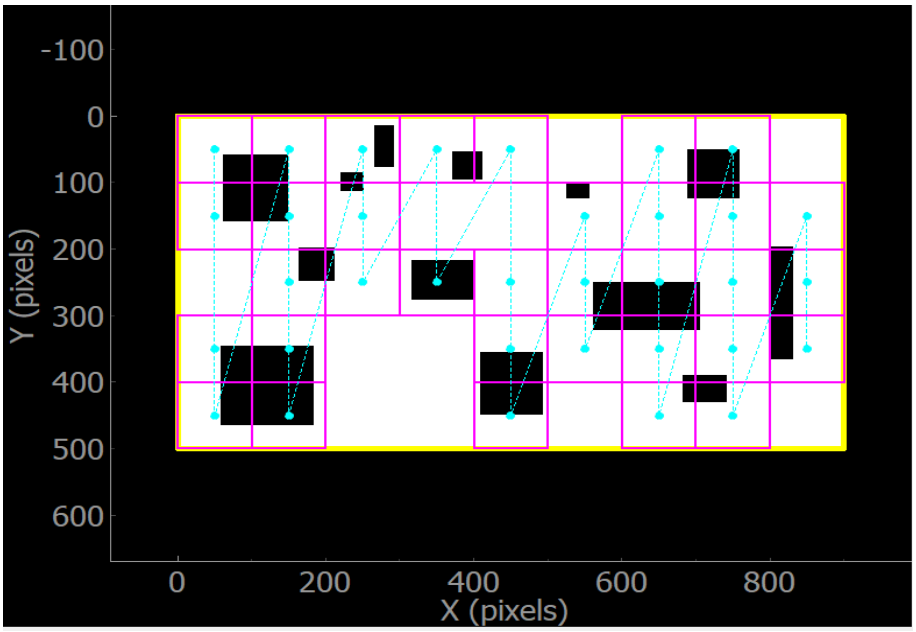
1st Task

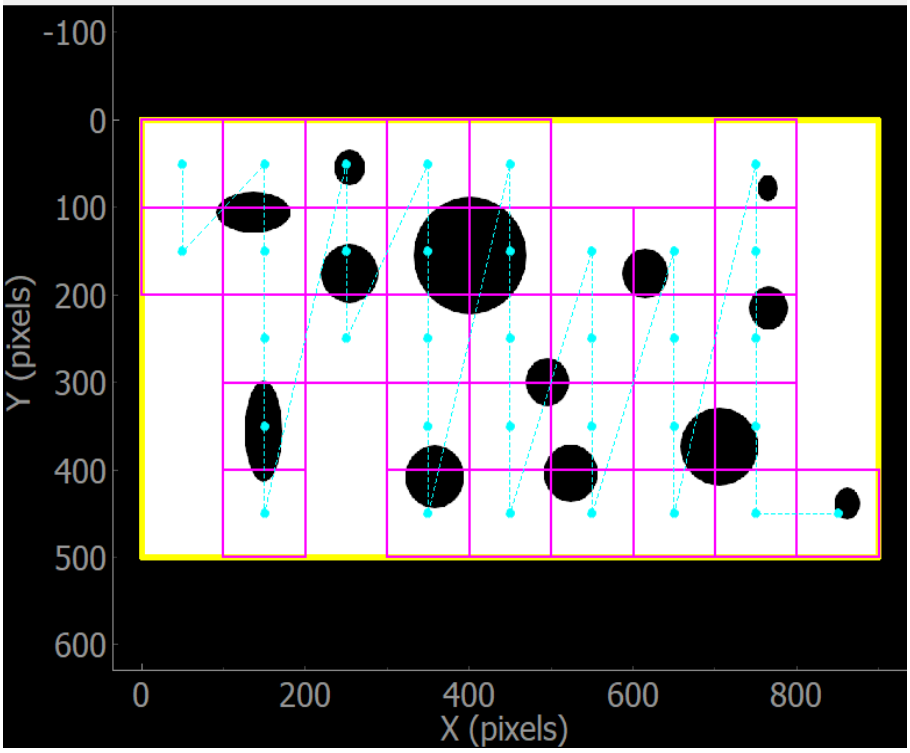
Skip completely empty cells

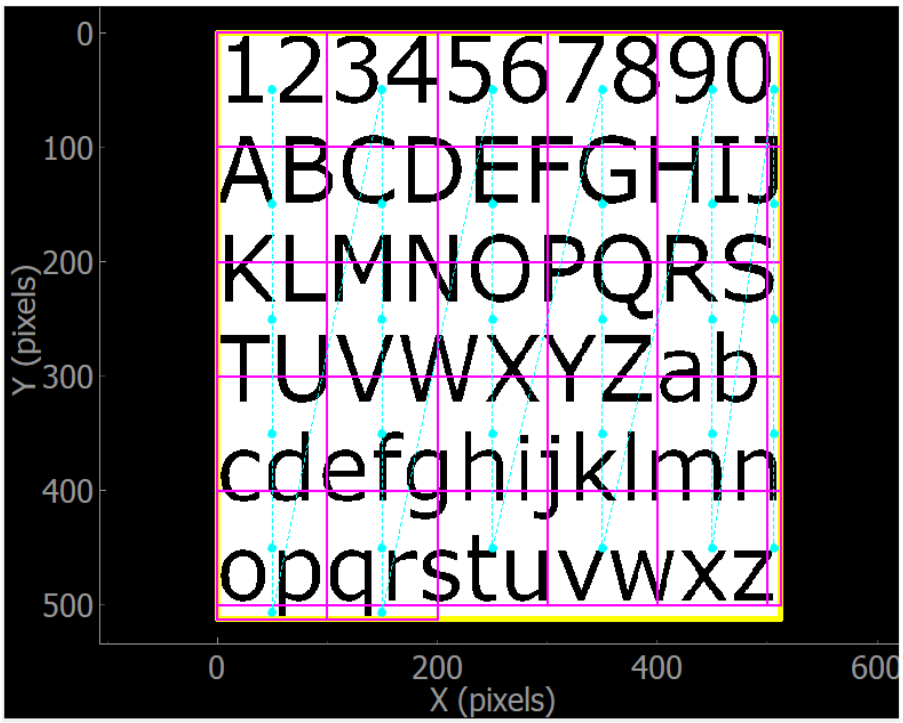
```
std::vector<Rect> Splitter::smartSplit(cv::Mat image,
    unsigned int splitDimXMax,
    unsigned int splitDimYMax)


{
     std::vector<Rect> result = {};
    unsigned int imageDimX = image.cols;
    unsigned int imageDimY = image.rows;

    unsigned int gridDimX = static_cast<unsigned int>(ceil(float(imageDimX) / splitDimXMax));
    unsigned int gridDimY = static_cast<unsigned int>(ceil(float(imageDimY) / splitDimYMax));
    for (unsigned int i = 0; i < gridDimX; ++i)
    {
        for (unsigned int j = 0; j < gridDimY; ++j)
        {
            Rect cell = {};
            cell.topLeft.x = i * splitDimXMax;
            cell.topLeft.y = j * splitDimYMax;
            cell.bottomRight.x = std::min((i + 1) * splitDimXMax, imageDimX);
            cell.bottomRight.y = std::min((j + 1) * splitDimYMax, imageDimY);
            cv::Rect myROI(cell.topLeft.x, cell.topLeft.y, cell.bottomRight.x - cell.topLeft.x, cell.bottomRight.y - cell.topLeft.y);//Rect(x,y,width,height)
            int TotalNumberOfPixels = (cell.bottomRight.x - cell.topLeft.x) * (cell.bottomRight.y - cell.topLeft.y);
            int ZeroPixels = TotalNumberOfPixels - cv::countNonZero(image(myROI)); //Dark pixels
             if (ZeroPixels > 0)
            {
                result.push_back(cell);
            }
        }
    }
        return result;
}
```

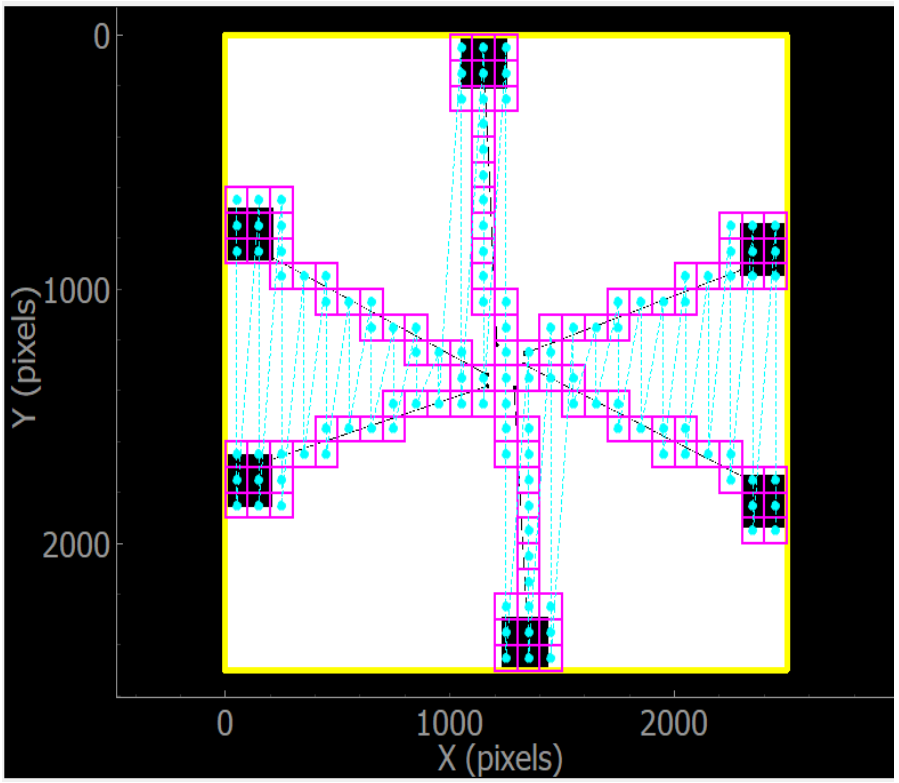
For the first task, we would like to include the area that has only black pixels,

I have created a Rectangle Region of Interest (ROI) cv:: Rect(x, y, width, height) and then I calculate all the pixels of this area, I subtract the nonzero(no black) pixels; thus if the Zero Pixels is greater than zero, I have at least one black pixel and I wish to include this cell. We could see below that the results are as we desired.


-100
0
100
200
300
400
500
600
Y (pixels)
0
200
400
600
800
X (pixels)


-100
0
100
200
300
400
500
600
Y (pixels)
0
200
400
600
800
X (pixels)


1234567890
ABCDEFGHIJ
KLMNOPQRS
TUVWXYZab
cdefghijklmn
opqrstuvwxz
0
100
200
300
400
500
Y (pixels)
0
200
400
600
X (pixels)

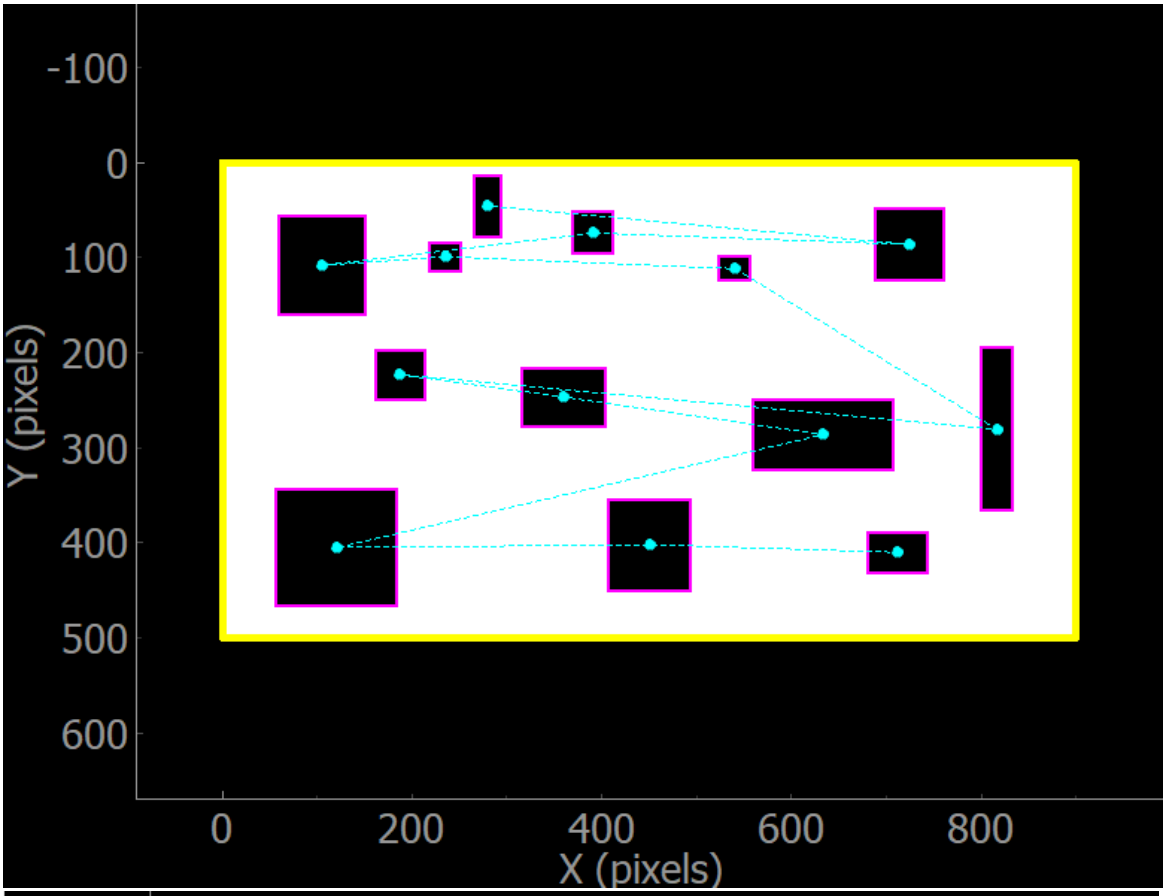

0
1000
2000
Y (pixels)
0
1000
2000
X (pixels)

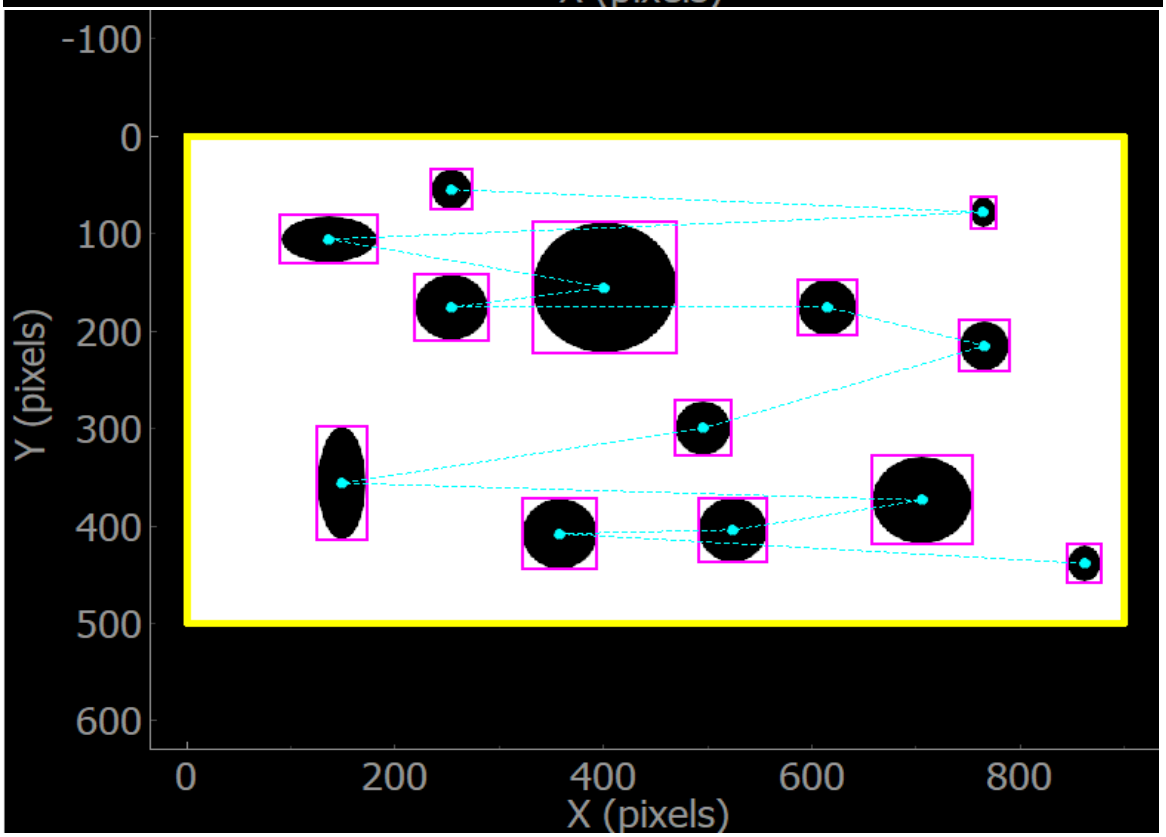
2nd Task

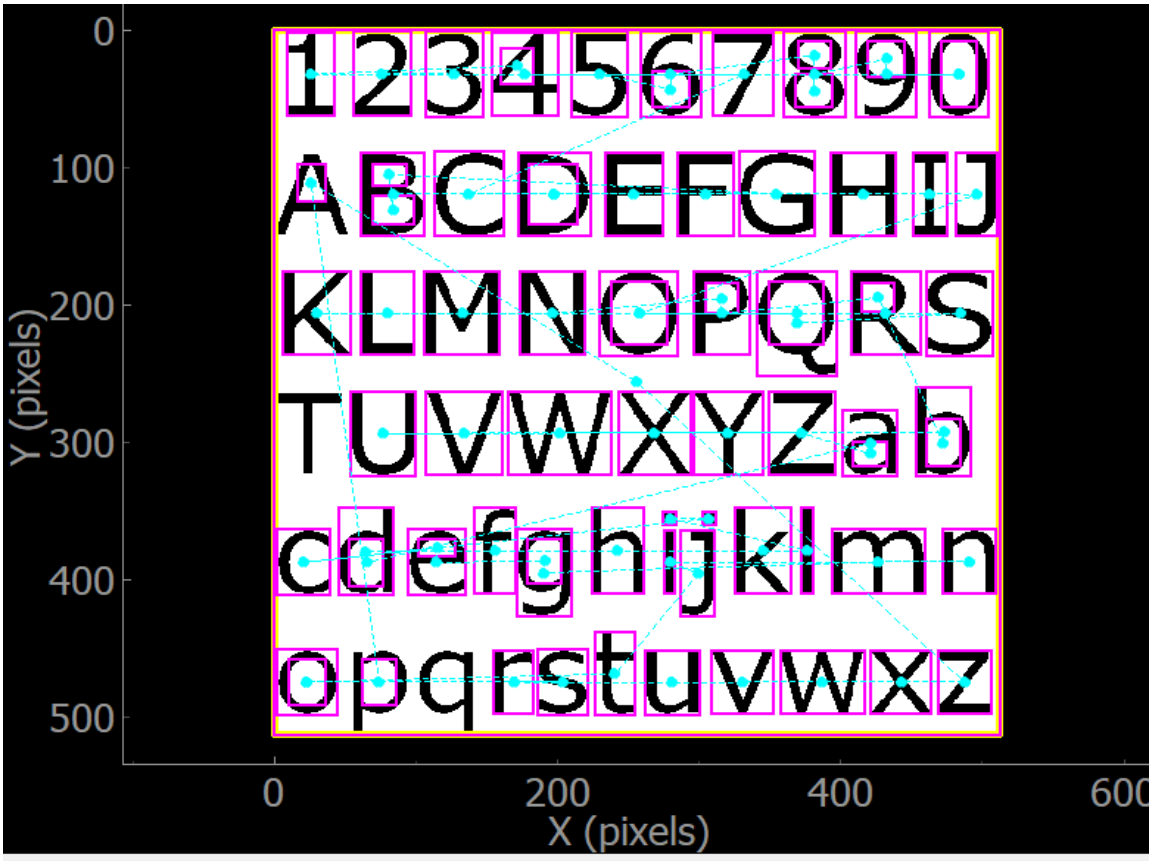
Auto Crop cells to the smallest bounding box

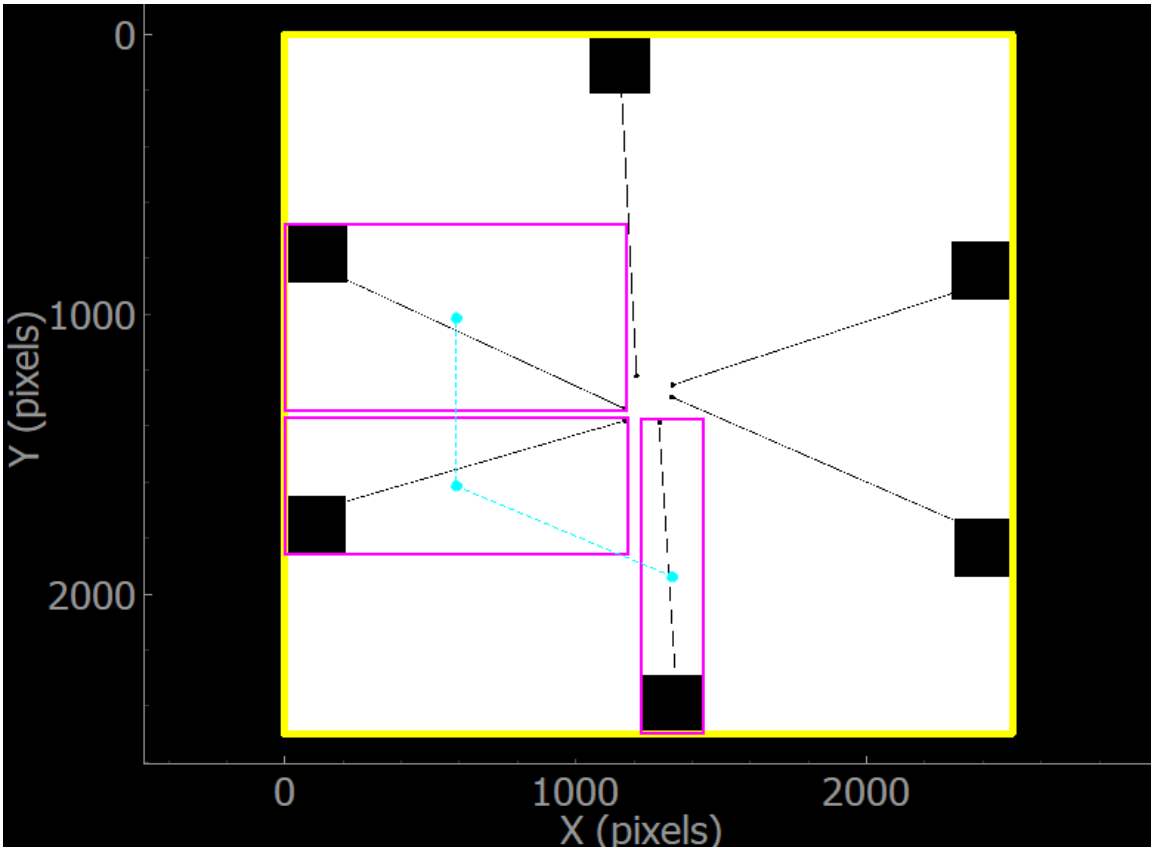
```
std::vector<Rect> Splitter::smartSplit(cv::Mat image,
    unsigned int splitDimXMax,
    unsigned int splitDimYMax)

{
    std::vector<Rect> result = {};
    Rect cell = {};
    std ::vector<std ::vector<cv::Point>> contours;

    //To store hierarchy(nestedness)
    std ::vector<cv::Vec4i> hierarchy;
    std::vector <double> distance;
    std::vector <int> indeces;

    //Find contours
    findContours(image, contours, hierarchy, cv::RETR_TREE,cv:: CHAIN_APPROX_SIMPLE);
    for (int i = 1; i < contours.size(); i++)
    {
        auto rect = cv::boundingRect(contours[i]);
        cell.topLeft.x = rect.tl().x;
        cell.topLeft.y = rect.tl().y;
        cell.bottomRight.x = rect.br().x;
        cell.bottomRight.y = rect.br().y;
        result.push_back(cell);
    }

    return result;
}
```

findContours(image, contours, hierarchy, cv::RETR_TREE,cv:: CHAIN_APPROX_SIMPLE); cv::RETR_TREE calculates the full hierarchy of the contour, cv::CHAIN_APPROX_SIMPLE returns only the endpoints that are necessary for drawing the contour line. Hence I can receive the coordinates for my cell by utilizing the tl (top left) and br (bottom right) point of the contour. I loop from i=1, since I noticed that for i=0 the whole image is consider a contour. The result are pretty satisfying for the first two images, some letters are not perfectly contoured and in contact_leads_2d I have a mediocre result.


-100
0
100
200
300
400
500
600
Y (pixels)
0
200
400
600
800
X (pixels)


-100
0
100
200
300
400
500
600
Y (pixels)
0
200
400
600
800
X (pixels)


1234567890
ABCDEFGHIJ
KLMNOPQRS
TUVWXYZab
cdefghijklmn
opqrstuvwxz
0
100
200
300
400
500
Y (pixels)
0
200
400
600
X (pixels)

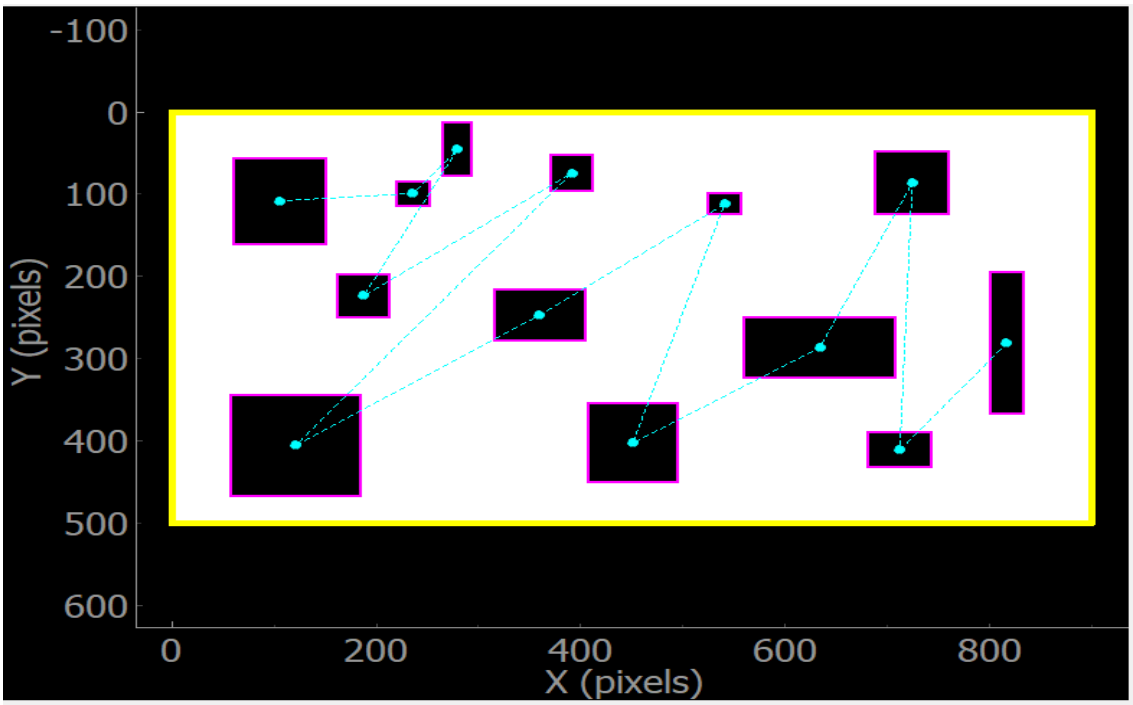

0
1000
2000
Y (pixels)
0
1000
2000
X (pixels)

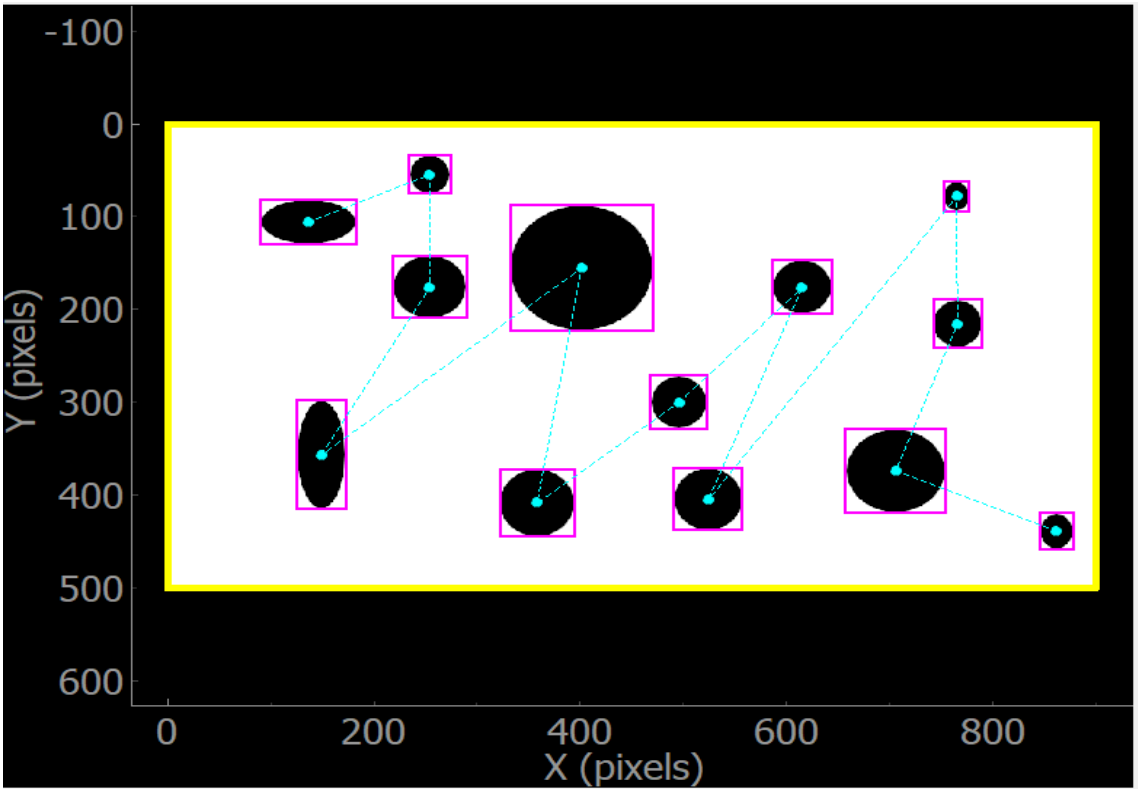
3rd Task

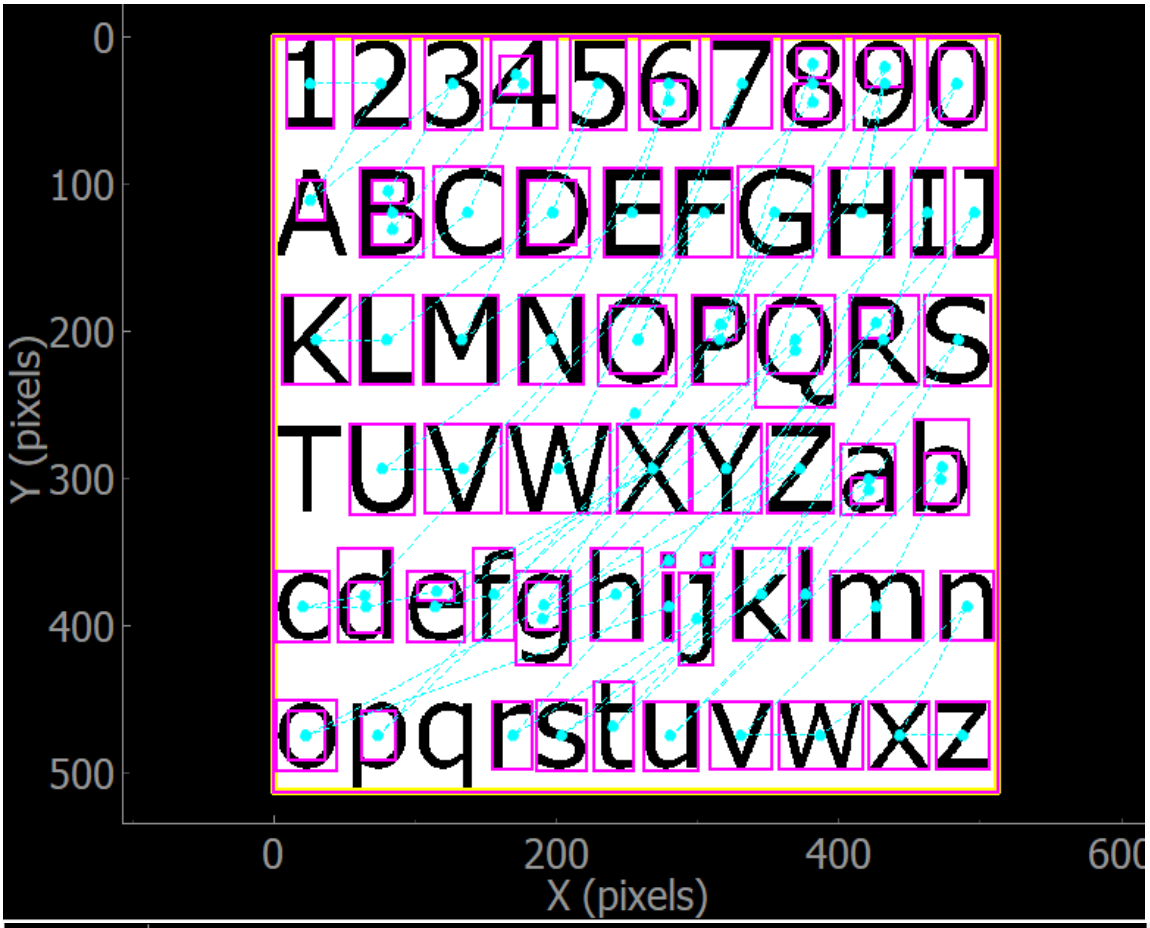
Optimize the traversal order of cells so that nearby cells are patterned together

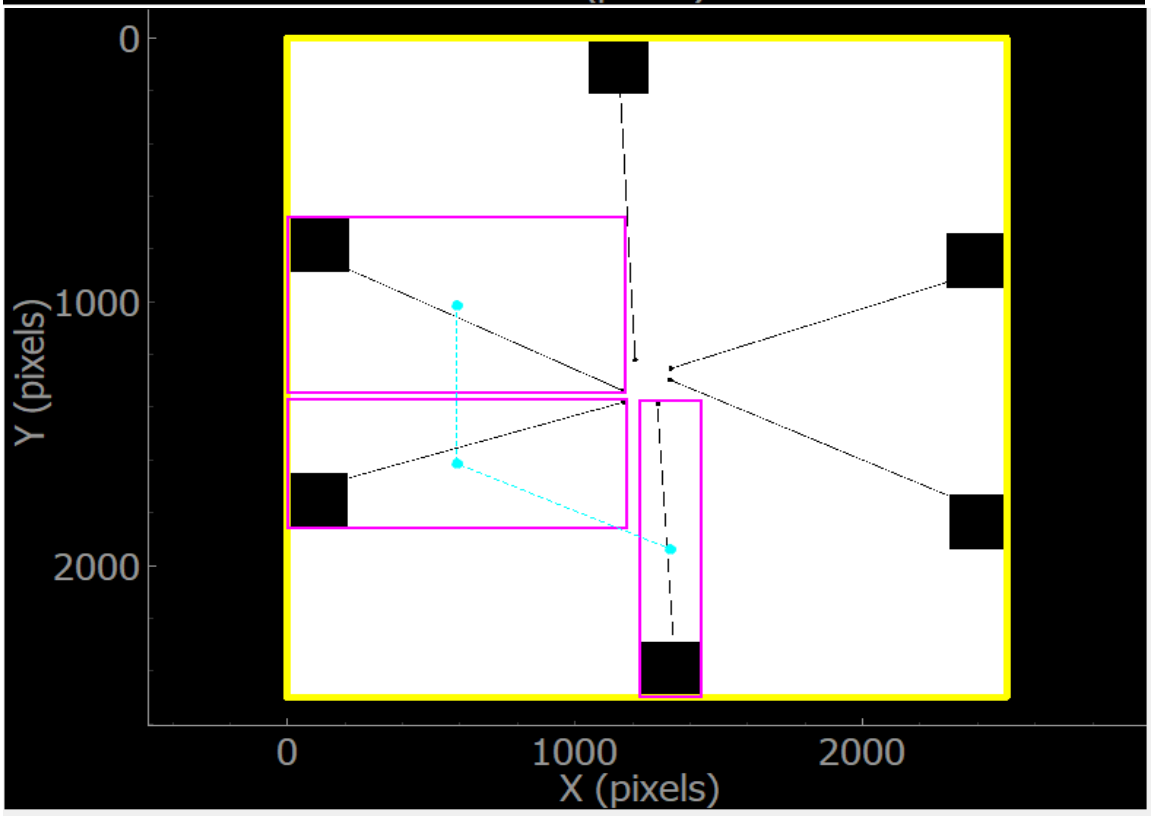
```
std::vector<Rect> Splitter::smartSplit(cv::Mat image,
    unsigned int splitDimXMax,
    unsigned int splitDimYMax)
{
    std::vector<Rect> result = {};
    Rect cell = {};
    std ::vector<std ::vector<cv::Point>> contours;
    //To store hierarchy(nestedness)
    std ::vector<cv::Vec4i> hierarchy;
    //Find contours


    findContours(image, contours, hierarchy, cv::RETR_TREE,cv:: CHAIN_APPROX_SIMPLE);
    std::vector <double> distance(contours.size());
    std::vector <int> indeces(contours.size());
    for (int i = 1; i < contours.size(); i++)
    {
        auto rect = cv::boundingRect(contours[i]);
        distance[i] = sqrt(pow((rect.tl().x + rect.br().x)/2, 2) + pow((rect.tl().y + rect.br().y)/2, 2));
        indeces[i] = i;
    }

    sort(indeces.begin(), indeces.end(), [&](int A, int B) -> bool {
        if (distance[A] != distance[B])
            return distance[A] < distance[B];
        else
            return distance[A];
        });

    for (int i = 1; i < indeces.size(); i++)
    {
        std::cout << distance[indeces[i]] << "dist" << indeces[i] << "ind"<<"\n";
        auto rect = cv::boundingRect(contours[indeces[i]]);
        cell.topLeft.x = rect.tl().x;
        cell.topLeft.y = rect.tl().y;
        cell.bottomRight.x = rect.br().x;
        cell.bottomRight.y = rect.br().y;
        result.push_back(cell);
    }

    return result;
}
```

I have added a vector of (double) distances that calculate the distance of the center of the rectangular contours from (0,0) and a vector of indeces. In the next step I sort the vector of indeces based on their distance, consequently we will minimize the traversal order of the cells. Same as before we loop from i=1 to avoid the contour of the whole image. The results have the similar contours as before but improved regarding their traversal order.


-100
0
100
200
300
400
500
600
Y (pixels)
0
200
400
600
800
X (pixels)


-100
0
100
200
300
400
500
600
Y (pixels)
0
200
400
600
800
X (pixels)


1234567890
ABCDEFGHIJ
KLMNOPQRS
TUVWXYZab
cdefghijklmn
opqrstuvwxz
0
100
200
300
400
500
Y (pixels)
0
200
400
600
X (pixels)


0
1000
2000
Y (pixels)
0
1000
2000
X (pixels)

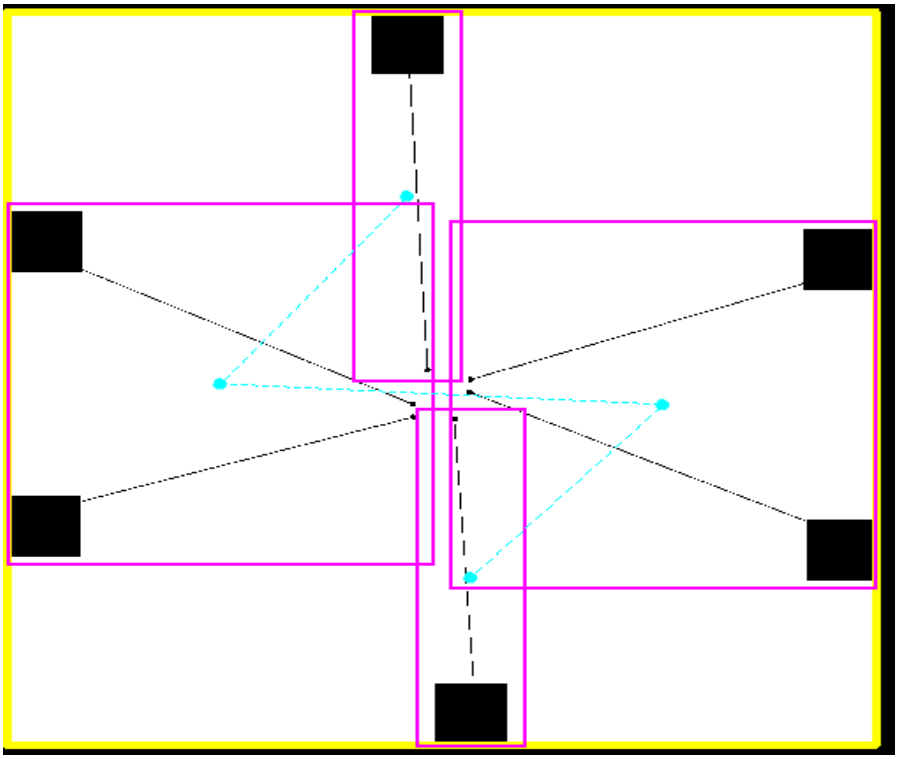
(Optional) Minimize the splitting of structures

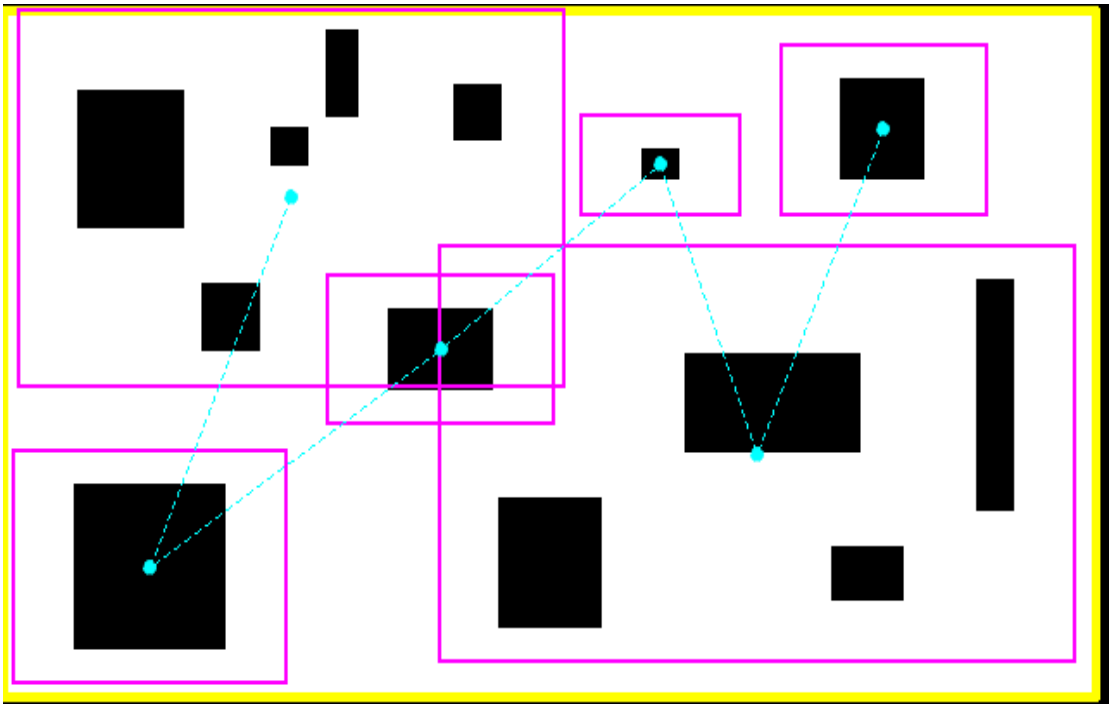
```
std::vector<Rect> Splitter::smartSplit(cv::Mat image,
    unsigned int splitDimXMax,
    unsigned int splitDimYMax)
{
    std::vector<Rect> result = {};
    Rect cell = {};
    std::vector<std::vector<cv::Point>> contours;
     std::vector<cv::Vec4i> hierarchy;
     cv::Mat dilate_img, thres_img;
    //--- performing Otsu threshold ---
    cv::threshold(image, thres_img, 0,255, cv::THRESH_OTSU | cv::THRESH_BINARY_INV);
    //cv::imshow("thresh1", thres_img);
    cv::Mat elementKernel = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(100,40), cv::Point(-1, -1));
    cv::dilate(thres_img, dilate_img, elementKernel,cv::Point(-1,-1),1);

     findContours(dilate_img, contours, hierarchy, cv::RETR_TREE, cv::CHAIN_APPROX_SIMPLE);
    std::vector <double> distance(contours.size());
    std::vector <int> indeces(contours.size());
    for (int i = 0; i < contours.size(); i++)
    {
        auto rect = cv::boundingRect(contours[i]);
        distance[i] = sqrt(pow((rect.tl().x + rect.br().x) / 2, 2) + pow((rect.tl().y + rect.br().y) / 2, 2));
        indeces[i] = i;
    }

    sort(indeces.begin(), indeces.end(), [&](int A, int B) -> bool {
        if (distance[A] != distance[B])
            return distance[A] < distance[B];});

    for (int i = 0; i < indeces.size(); i++)
    {
        std::cout << distance[indeces[i]] << "dist" << indeces[i] << "ind" << "\n";
        auto rect = cv::boundingRect(contours[indeces[i]]);
        cell.topLeft.x = rect.tl().x;
        cell.topLeft.y = rect.tl().y;
        cell.bottomRight.x = rect.br().x;
        cell.bottomRight.y = rect.br().y;
        result.push_back(cell);
    }

    return result;
}
```

Image thresholding is used to binarize the image based on pixel intensities, in our case it is not necessary to use it, since the image is already binary. We need to dilate the image in order to add pixels to the boundaries of objects in our images and group more black pixels together (to be more specific in the photo of letter_numbers to connect them as a block of text). We need to select a suitable morphological operation for the above examples that will form a single region along the horizontal direction. We choose a kernel that is larger in width than the height. Thus, we start with an element kernel of size (50,25):

cv::Mat elementKernel = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(50,25), cv::Point(-1, -1));
We also switch back the contour starting from i=0, since the way of selecting contours changed.

We are pleased with our result and with trial and error, attempt to find a suitable kernel for our next image. For Size(100,50):
We are finally able to get all the black pixels with this approach for the contact _leads image
and another great result for the rectangles.





Finally for the ellipse example we use:

```
cv::Mat elementKernel = cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(100,50), cv::Point(-1, -1));
```