**Video Processing**

n,m,l = img.shape

#processing

for i in range(n):

for j in range(m):

if(sum(img[i,j]) <= 254\*3):

img[i][j] = (0,0,0)

else:

img[i][j] = (255,255,255)

####

Here n and m denote the number of rows and columns of pixels in the image.

Looping over the pixels: if the sum of values of all the three fields (B, G, R) is less than or equal to (254\*3) {or if the average of the colors in a pixel is less than or equal to 254, then it has been set to [0, 0, 0] (black). Otherwise, the pixel value is made [255,255,255] (white).

**Dijsktra**

closed\_list = []

queue = []

min\_distance = np.full((n,m),np.inf)

visited = np.full((n,m), False)

parent = np.full((n,m), None)

These are a few declarations used.

**closed\_list** stores the traversed nodes.

**min\_distance** is a 2D array to store the minimum distance between a point and the starting point.

**visited** is a 2D array to denote whether a node has been visited or not.

**parent** is a 2D array storing the parent of each node

dijsktra(img\_djiksktra,start,stop)

Calling the dijsktra function

def dijsktra(img, start, stop):

global parent

queue.append(start)

min\_distance[start[0],start[1]] = 0

visited[start[0],start[1]] = True

parent[start[0],start[1]] = None

node = None

The start node is being appended to the queue, min\_distance of start is updated to 0 and that node is marked visited.

while(len(queue) != 0):

cv2.imshow("Path Planning",img)

cv2.waitKey(1)

node = queue.pop(0)

if(node == stop):

break

(i,j) = node

for i1 in range(i-1,i+2):

for j1 in range(j-1,j+2):

if(np.array\_equal(img[i1,j1], [255,255,255]) == True or (i1,j1) == stop) and (i1,j1) not in closed\_list:

dist = math.sqrt((i1-i)\*\*2 + (j1-j)\*\*2)

if(visited[i1,j1] == True):

if(dist + min\_distance[i,j] < min\_distance[i1,j1]):

min\_distance[i1,j1] = dist + min\_distance[i,j]

parent[i1,j1] = node

else:

queue.append((i1,j1))

min\_distance[i1,j1] = min\_distance[i,j] + dist

parent[i1,j1] = node

visited[i1,j1] = True

if(np.array\_equal(img[i1,j1], [255,255,255])):

img[i1,j1] = yellow

Repetitively doing steps while the length of queue doesn’t become 0.

The first element of the queue is popped and stored in **node**.

Immediately break out of the loop if the popped node is the end node.

Inspecting all the neighbors of the current node,

* If the node is white in color(i.e not visited) or if it is the stop node and it is not traversed, then the distance between the node and neighbor is stored in **dist** .
  + If that node is not visited, the min\_distance and its parent is updated. It is marked visited and colored yellow.
  + Otherwise, a comparison is made between the previously stored min\_distance. And if and only if the dist < min\_distance , the min\_distance is updated to dist and the parent is updated to **node**.
* Otherwise, continue to the next iteration.

**Path Tracing**

stack = []

node = stop

while(node != None):

stack.append(node)

node = parent[node[0]][node[1]]

while(len(stack) != 0):

x = stack.pop()

for i in range(-1,2):

for j in range(-1,2):

img\_djiksktra\_path[i + x[0], j + x[1]] = (255,0,127)

cv2.imshow("Path",img\_djiksktra\_path)

cv2.waitKey(2)

while the node is not equal to **None** [the parent of the start node is None]

The node is appended into the stack.

Then, until the stack is empty, elements are popped out from the rear repeatedly, and these nodes are marked [colored violet].

for i in range(-1,2):

for j in range(-1,2):

img\_djiksktra\_path[i + x[0], j + x[1]] = (255,0,127)

This is done to broaden the line tracing shortest path.

**Astar**