Flow Chart for all the steps (RED – output and GREEN –input)

Step 1

Left image

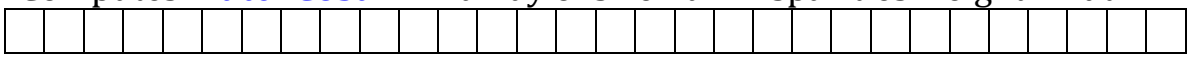
Right image (shifted towards left)

Perform SAD/SSD/NCC

SSD(QImage image1, QImage image2, int minDisparity, int maxDisparity, int offset, double *matchCost)
SAD(QImage image1, QImage image2, int minDisparity, int maxDisparity, int offset, double *matchCost)
NCC(QImage image1, QImage image2, int minDisparity, int maxDisparity, int offset, double *matchCost)

Window size = 2*offset +1 (For example, offset =2, then Window size is 5x5)

Computes **matchCost** : 1-D array of size numDisparities*height*width



Find the disparity with minimum cost for each pixel

FindBestDisparity(double *matchCost, double *disparities, int w, int h, int minDisparity, int numDisparities)

Step 2

Perform smoothing of the match cost values before finding the best disparity.

SSD/SAD/NCC → Gaussian/Bilateral smoothing → FindBestDisparity

Step 3

Segment (computes K-means to segment the image in *color* and *position* space)

GridSegmentation (Compute an initial segmentation) is already implemented

GridSegmentation(segment, numSegments, gridSize, w, h);

1. ComputeSegmentMeans

Compute the mean color and position for each segment

ComputeSegmentMeans(QImage image, int *segment, int numSegments, double (*meanSpatial)[2], double (*meanColor)[3])

2. AssignPixelsToSegments

Assign each pixel to the closest segment using position and color.

AssignPixelsToSegments(QImage image, int *segment, int numSegments, double (*meanSpatial)[2], double (*meanColor)[3], double spatialSigma, double colorSigma)

SSD (computes the matchCost)

3.SegmentAverageMatchCost

Average the match cost for each pixel in a segment.

SegmentAverageMatchCost(int *segment, int numSegments, int w, int h, int numDisparities, double *matchCost)

FindBestDisparity for updated matchCost array