Ric

Arguments： D:\gitprojects\Ric\data\KITTI\000008\_10.png D:\gitprojects\Ric\data\KITTI\000008\_11.png D:\gitprojects\Ric\data\KITTI\000008\_10\_match.txt D:\gitprojects\Ric\data\model.yml.gz

Data in the file “000008\_10\_match.txt”

In a line, 371 5 383 9: the matching pixel index in row and column, (x1, y1, x2, y2). The data is loaded into the variable “outMat” by the function void ReadMatches(const char \*filename, FImage& outMat).

1. int main(int argc, char\*\* argv)

GetCostMap(argv[1], argv[4], costMap);

ReadMatches(argv[3], matches);

RIC ric;

ric.SetSuperpixelSize(100);

ric.Interpolate(img1, img2, costMap, matches, u, v);

OpticFlowIO::WriteFlowFile(u.pData, v.pData, w, h, outName);

OpticFlowIO::SaveFlowAsImage(outName, u.pData, v.pData, w, h);

1. void GetCostMap(char\* imgName, char\* modelName, FImage& outCostMap)

cv::Ptr<cv::ximgproc::StructuredEdgeDetection> sEdge = cv::ximgproc::createStructuredEdgeDetection(modelName); // ("./model.yml.gz"); the model is trained with the “Structured edge detection toolbox”

sEdge->detectEdges(fImg1, edges);

outCostMap[i\*w + j] = edges.at<float>(i + borderSize, j + borderSize); // costmap is the intensity of the edge image

1. void RIC::Interpolate(FImage& img1, FImage& img2, FImage& costMap, FImage& inputMatches, FImage& outU, FImage& outV)

// Distance transform for each matching pixel,

for (int i = 0; i < matchingCnt; i++){ // each matching pixel

float\* p = inputMatches.rowPtr(i);

float x = p[0]; // row of matching pixel

float y = p[1]; // col of matching pixel

int idx = y\*w + x; // convert (row, col) to 1 dimension index

matLabels[idx] = i; // the pixel index with the matching list index

matDistanceMap[idx] = costMap[idx]; // distance is from the costMap

}

GeodesicDistanceTransform(costMap.pData, matDistanceMap.pData, matLabels.pData, w, h); // some components will have the same label