**Assignment 3:**

**Description of Visual Relations**

**Jason Mann**

**4/9/13**

**COMS 4735: Visual Interfaces for Computers**

Table of Contents

**Application OverView 4**

**Assignment Detail 5**

**Part 1: Building Features and Descriptions 5**

**Part 2: Describing Compact Spatial Relations 6**

**Part 3: Source and Goal Description 7**

**Part 4: Path Generation 8**

**OutPut 9**

**Appendix A: Part one 9**

**Appendix B: Part Two 10**

**Appendix C: Part Three 20**

**Code 22**

**MapReader.pro 22**

**main.cpp 22**

**mapreader.h 23**

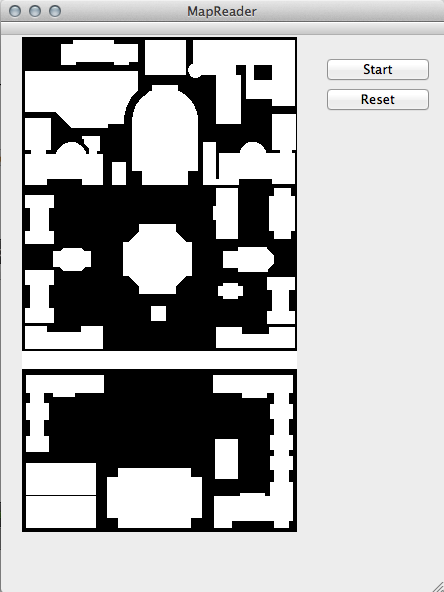
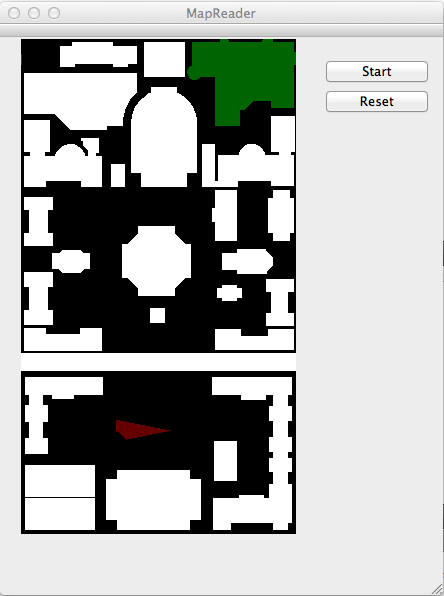
**mapreader.cpp 25**

**mapreader.ui 38**

**clickablelabel.h 39**

# Application Overview

**This assignment was generally to take in a binary image representing a map (in this case Columbia’s Morningside campus) and to provide descriptions for the buildings shown, describe their spatial relations, use those spatial relations to map clusters of similarly described areas, and finally to generate paths through the map and record their accuracy through experimentation.**

**I programmed this assignment in C++ (gcc/stdc++11) with the assistance of OpenCV 2.4.3 library for vision related tasks and used the Qt library for the simple GUI. The form seen in the picture below is the application, and displays the campus image, and has two buttons, start and reset. The start button is used after two places on the map are clicked (signifying source and goal), and then the program provides you with step by step instructions as you follow along with clicks. The entire process is recorded to measure the accuracy of the clicks.**

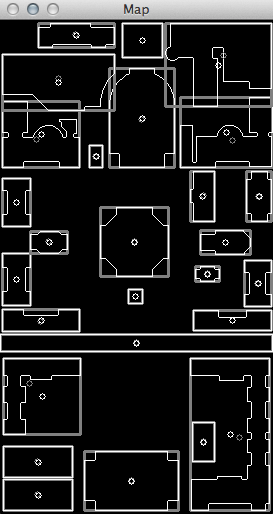
# Assignment Detail

Step One: Building Features and Descriptions

**To describe the buildings in the given image, I used built in functions from OpenCV to help me process the data. I used the findContours() method to give me contours of all of the ‘blobs’ (here buildings) in the image. Then I processed those contours, finding their moments(including area), center of mass, and minimum bounding rectangles. With this data I then assigned natural language descriptions to the buildings in three descriptive categories: shape, size, and orientation.**

**Distinguishing between sizes was done fairly simply, by keeping track of the max, min, and average area found on the map, and calculating the desired cutoffs for separation into ‘small,’ ‘medium,’ and ‘large’. For my purposes, a large building was anything with area greater than the average and range summed then divided by 6. Small used the same values but differed them and divided by 4. Medium was left as everything in between.**

**The shape descriptors were the most complex. I first split off the odd shaped buildings by checking which ones had very different locations for the center of mass and the center of the bounding rectangle. These were then split again by specific traits, giving “L-shaped”, “U-shaped”, and “lumpy.” This was done by checking whether the center of mass was inside a building contour, and if not then checking if it had 3 sides (U shaped) or just 2 (L shaped). The remainder were attributed as lumpy. Narrow was given the next highest precedence, and was labeled such if the width or the height was greater than three times the other. I-shaped buildings were found by checking the midpoints of each pair of sides of a contour for not continuing to the bounding rectangle. ‘Cross shaped’ was next, by checking if the corners of the enclosing rectangle were filled. ‘Squarish’ and ‘rectanglular‘ were last, but were easily found by comparing the contour area to the bounding rectangles area.**

**Orientation was allowed to be north-south or east-west, but was only applied if the building was ‘half narrow’ or had a width/height that was 1.5 times greater than the other. The specific element that was greater determined the directionality.**

SEE APPENDIX A FOR DATA

Step Two: Describing Compact Spatial Relations

**The compact spatial relations in this assignment were specified as functions north(s,g), south, east, west, and near, where s is the source and g is the goal. This relationship translates roughly to North of s is g. To implement the cardinal directions, I used an algorithm that created a ‘fan’ of acceptance, by using slopes and checking multiple points on the bounds of the building. This was an improvement on an earlier algorithm I implemented that merely checked the center of masses against the y or x value (for n-s/e-w). It also helps in establishing transitive relations, and is rationally reasoned, by imagining the compass directions (north is from northwest to northeast). The near relation was found by iterating over the points on the contour of each building, and testing if their minimum distance is less than the default set distance modified by a size ratio. The default distance for nearness was set at the dimensions of the image summed divided by 40, and the size of the source building was allowed to adjust this default distance the ratio of the area of the contour divided by the average area, times 10. This allows for building size to affect the likelihood of being ‘near to’ as can be seen by the large number of buildings near to (called Uris).**

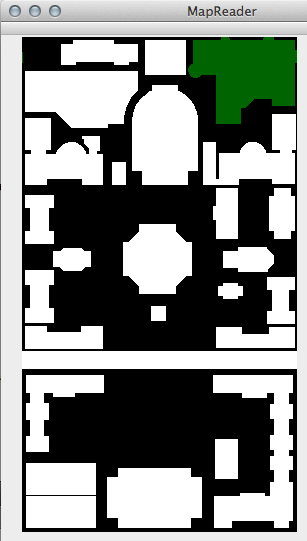
**The next step was to create a table of all the possible connections in the image and reduce them to only the necessary descriptors. I kept a 2D array for each spatial relation and used a transitive closure/reduction algorithm to eliminate unnecessary specifications. I used a simple reduction algorithm on the cardinal directions, but decided against allowing nearness to affect the reduction. I came to this conclusion through experimentation with different methods and results, but have included code to show the process of further reduction based on nearness.**

**After the tables were minimized I stored the spatial relations of a building in the object representing the building itself for ease of access. I have provided the printed out minimal binary pairs of spatial relations for each building, but I have also included a text graphic showing the 2D charts before and after the reductions.**

**SEE APPENDIX B FOR DATA AND GRAPHICS**

Step Three: Source and Goal Description

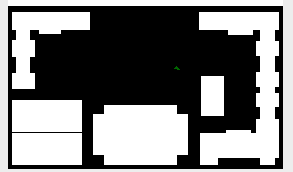
**In step three we were to dynamically use the code for the above spatial relations to create clouds of equivalence classes based on a single (x,y) input. This input was taken from mouse clicks on the form, and in preparation for the next step, the clouds are shown up to two at a time. The first cloud is displayed as a green blob (source), the second as a red blob (goal). I found these blobs by using a variation of breadth first search outwards from the pixel selected, creating a terminating case when the spatial relation of a pixel was no longer the same as the original source pixel. However, I did not generate all possible relations, but instead used a transitive reduction method similar to the one described above. On top of the transitive reduction, I only allowed for a maximum of one relation per direction (chosen by closeness to the pixel specified), to allow the equivalence classes to be larger. None of the nearness relations were pruned however, while multiple ‘near’s added crucial implied information, such as inbetween. The last step of reduction was to see if any of the nearness relations to the original pixel (and the rest of the equivalence class) already contained the closest directional relations and eliminate those directional relations. A special case was provided for those pixels contained within the contour of a building, because stating ‘in’ a building was not allowed, I instead opted to have those clouds be only ‘near’ to that building, and have no other relations. This allowed for the buildings themselves to be clouds, and added accuracy.**

** Because of this last equivalence class option, the cloud contained by and surrounding (called Mudd) was the largest:**

( 239 , 41 ) is NEAR ( the lumpy, large structure )(called Mudd, Engineering Terrace, Fairchild & Computer Science )

**There were however a few clouds less than 10 pixels because of the complex long distance directional relations and the areas immediately surrounding their intersecting vectors, which were the smallest such as (the picture is zoomed for easier visualization):**

( 169 , 395 ) is NORTH of ( the cross-shaped, large, east to west oriented structure )(called Butler Library ), and EAST of ( the L-shaped, medium sized structure )(called Journalism & Furnald ), and NEAR ( the cross-shaped, large, east to west oriented structure )(called Butler Library )

****

**Most of the clouds created (outside of the buildings) varied between 60 and 600 pixels with most of the regions showing around 400 pixels.**

**SEE APPENDIX C FOR PAIRS and DATA**Step 4: Path Generation

**The final component of this assignment was to first find paths between the previous sets of sources and goals, and then display the three paths randomly to three users, varying the level of detail. The three levels of detail are: no information beside spatial relations, the inclusion of building descriptions, and finally the inclusion of building names.**

**I generated my paths by a limited depth first search, in which I started at the source, and explored each of its spatial relations to buildings. I afterwards only explored relations between other buildings until I reached a building that was included in the goal’s spatial relations. At this point I provided terminal guidance from the reverse of the goal’s spatial relation to the current building. I returned the complete path as a vector up the search tree, keeping only the tree with minimum distance. The paths found were not always optimal, I believe because of my transitive reduction stages, but combined as well with the fact that my spatial relations for the cardinal directions sometimes retained long distance relationships when there were no interfering buildings in the sight line (even if that sight line was only a few pixels wide). Because we were not concerned with speed, the generation of my paths take upwards of a minute due to the heavy memory requirement of copying the path represented by a vector at every branch.**

**I was not able to complete 9 experiments with 3 separate people, because the output of natural language describing the path to take was not fully complete in time, and would have been hard for a layperson to understand. I found a failure in my algorithm used for announcing the path that uses the relations of the currently searched node incorrectly, by following the relation in the completely opposite direction, and was only able to fix it at the last minute. I have however included an example of an intended path description (note the first and last step are source and goal, and have no traditional building description). Although the natural language announcing of directions is not complete, the paths now seem to be somewhat optimal, though there is some confusion in the north east corner, with two buildings being described as lumpy, large structures.**

**Pair 2 actual output:**

**distance: 657**

**step 1 :**

**step 2 : the lumpy, large structure**

**step 3 : the large, north to south oriented structure**

**step 4 : the cross-shaped, large structure**

**step 5 : the squarish, small structure**

**step 6 : the narrow, large, east to west oriented structure**

**step 7 : the L-shaped, medium sized structure**

**step 8 : the rectangular, medium sized, east to west oriented structure**

**step 9 :**

**Pair 2 intended output:**

**distance: 657**

**step 1 : ( 255 , 21 ) is near ( the lumpy, large structure )(called Mudd, Engineering Terrace, Fairchild & Computer Science )**

**step 2 : Now go near (the lumpy, large structure) (called Mudd, Engineering Terrace, Fairchild & Computer Science)**

**step 3 : Now go west (the large, north to south oriented structure) (called Gymnasium & Uris )**

**step 4 : Now go south (the cross-shaped, large structure )(** **called Low Library )**

**step 5 : Now go south and near (the squarish, small structure )** **(called Alma Mater )**

**step 6 : Now go south and near (the narrow, large, east to west oriented structure ) (called College Walk)**

**step 7 : Now go south and near (the L-shaped, medium sized structure ) (called Journalism & Furnald)**

**step 8 : Now go south and near (the rectangular, medium sized, east to west oriented structure) (called Lerner)**

**step 9 : Now go east and near ( 127 , 447 ) is NEAR ( the cross-shaped, large, east to west oriented structure )(called Butler Library )**

**As you can see, the ordering is correct, and semi-optimal, though the directions are not included because they are not correct, and muddle up the directions. Also, the recording of the clicks is currently working (full descriptions from debugging):**

224 , 35 ) is ( the narrow, medium sized, east to west oriented structure )(called Dodge ), ( the lumpy, large structure )(called Schermerhorn ), are SOUTH of ( the large, north to south oriented structure )(called Gymnasium & Uris ), ( the squarish, medium sized structure )(called Schapiro CEPSR ), and are WEST of ( the lumpy, large structure )(called Schermerhorn ), ( the large, north to south oriented structure )(called Gymnasium & Uris ), ( the squarish, medium sized structure )(called Schapiro CEPSR ), and are NEAR

**30 from intended target Now go**

( 142 , 99 ) is ( the squarish, medium sized structure )(called Schapiro CEPSR ), ( the narrow, medium sized, east to west oriented structure )(called Pupin ), are NORTH of ( the narrow, medium sized, east to west oriented structure )(called Dodge ), ( the cross-shaped, large structure )(called Low Library ), ( the rectangular, medium sized, north to south oriented structure )(called Avery ), and are SOUTH of ( the lumpy, large structure )(called Mudd, Engineering Terrace, Fairchild & Computer Science ), and are EAST of ( the lumpy, large structure )(called Chandler & Havemeyer ), ( the large, east to west oriented structure )(called Physical Fitness Center ), and are WEST of ( the rectangular, medium sized, north to south oriented structure )(called Avery ), ( the rectangular, small, north to south oriented structure )(called Computer Center ), ( the lumpy, large structure )(called Chandler & Havemeyer ), ( the lumpy, large structure )(called Schermerhorn ), ( the large, east to west oriented structure )(called Physical Fitness Center ), ( the lumpy, large structure )(called Mudd, Engineering Terrace, Fairchild & Computer Science ), ( the squarish, medium sized structure )(called Schapiro CEPSR ), ( the narrow, medium sized, east to west oriented structure )(called Pupin ), and are NEAR

# Output

# **Appendix A: Step One**

**Abbreviations: CoM – Center of Mass, Bound UL – Bounding Rectangle Upper Left Corner, LR – Lower Right Corner**

# **1 (Pupin): CoM: [76, 14] Area: 1538.5 Bound UL: [39, 3] LR: [116, 28]**

# **Description: narrow, medium sized, east to west oriented structure**

# **2 (Schapiro CEPSR): CoM: [143, 20] Area: 1360 Bound UL: [123, 3] LR: [164, 38]**

# **Description: squarish, medium sized structure**

# **3 (Mudd, Engineering Terrace, Fairchild & Computer Science): CoM: [224, 35] Area: 5870 Bound UL: [166, 3] LR: [273, 87]**

# **Description: lumpy, large structure**

# **4 (Physical Fitness Center): CoM: [59, 58] Area: 5214.5 Bound UL: [3, 34] LR: [116, 91]**

# **Description: large, east to west oriented structure**

# **5 (Gymnasium & Uris): CoM: [142, 99] Area: 5606 Bound UL: [110, 48] LR: [176, 148]**

# **Description: large, north to south oriented structure**

# **6 (Schermerhorn): CoM: [233, 120] Area: 3696 Bound UL: [181, 77] LR: [274, 148]**

# **Description: lumpy, large structure**

# **7 (Chandler & Havemeyer): CoM: [37, 119] Area: 3429 Bound UL: [3, 81] LR: [81, 148]**

# **Description: lumpy, large structure**

# **8 (Computer Center): CoM: [96, 136] Area: 286 Bound UL: [90, 125] LR: [104, 148]**

# **Description: rectangular, small, north to south oriented structure**

# **9 (Avery): CoM: [204, 175] Area: 1090 Bound UL: [191, 151] LR: [216, 202]**

# **Description: rectangular, medium sized, north to south oriented structure**

# **10 (Fayerweather): CoM: [259, 176] Area: 1108 Bound UL: [247, 151] LR: [273, 202]**

# **Description: cross-shaped, medium sized, north to south oriented structure**

# **11 (Mathematics): CoM: [17, 182] Area: 1106 Bound UL: [3, 158] LR: [32, 207]**

# **Description: I-shaped, medium sized, north to south oriented structure**

# **12 (Low Library): CoM: [135, 221] Area: 3782 Bound UL: [101, 187] LR: [170, 257]**

# **Description: cross-shaped, large structure**

# **13 (St. Paul's Chapel): CoM: [226, 222] Area: 1019.5 Bound UL: [201, 210] LR: [252, 235]**

# **Description: cross-shaped, medium sized, east to west oriented structure**

# **14 (Earl Hall): CoM: [49, 221] Area: 706 Bound UL: [31, 211] LR: [69, 234]**

# **Description: cross-shaped, small, east to west oriented structure**

# **15 (Lewisohn): CoM: [17, 259] Area: 1218 Bound UL: [3, 233] LR: [32, 286]**

# **Description: I-shaped, medium sized, north to south oriented structure**

# **16 (Philosophy): CoM: [258, 263] Area: 1002 Bound UL: [245, 240] LR: [273, 287]**

# **Description: I-shaped, medium sized, north to south oriented structure**

# **17 (Buell & Maison Francaise): CoM: [208, 253] Area: 302 Bound UL: [196, 246] LR: [221, 262]**

# **Description: cross-shaped, small, east to west oriented structure**

# **18 (Alma Mater): CoM: [136, 276] Area: 196 Bound UL: [129, 269] LR: [144, 284]**

# **Description: squarish, small structure**

# **19 (Dodge): CoM: [41, 301] Area: 1485 Bound UL: [3, 289] LR: [81, 312]**

# **Description: narrow, medium sized, east to west oriented structure**

# **20 (Kent): CoM: [233, 300] Area: 1365 Bound UL: [194, 290] LR: [273, 311]**

# **Description: narrow, medium sized, east to west oriented structure**

# **21 (College Walk): CoM: [137, 322] Area: 4624 Bound UL: [1, 314] LR: [274, 332]**

# **Description: narrow, large, east to west oriented structure**

# **22 (Journalism & Furnald): CoM: [30, 363] Area: 2444 Bound UL: [4, 338] LR: [82, 415]**

# **Description: L-shaped, medium sized structure**

# **23 (Hamilton, Hartley, Wallach & John Jay): CoM: [240, 417] Area: 5532 Bound UL: [191, 338] LR: [271, 491]**

# **Description: U-shaped, large, north to south oriented structure**

# **24 (Lion's Court): CoM: [204, 421] Area: 858 Bound UL: [193, 402] LR: [216, 442]**

# **Description: rectangular, small, north to south oriented structure**

# **25 (Lerner Hall): CoM: [38, 441] Area: 2139 Bound UL: [4, 426] LR: [74, 458]**

# **Description: rectangular, medium sized, east to west oriented structure**

# **26 (Butler Library): CoM: [132, 460] Area: 5130 Bound UL: [85, 431] LR: [180, 491]**

# **Description: cross-shaped, large, east to west oriented structure**

# **27 (Carman): CoM: [38, 474] Area: 2139 Bound UL: [4, 459] LR: [74, 491]**

# **Description: rectangular, medium sized, east to west oriented structure**

# **Appendix B: Step Two**

COMPLETE RELATIONS

north

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

1 Pupin

2 Schapir

3 Mudd, E

4 Physica 1

5 Gymnasi 1 1

6 Scherme 1 1

7 Chandle 1 1

8 Compute 1 1 1

9 Avery 1 1 1

10 Fayerwe 1 1

11 Mathema 1 1 1

12 Low Lib 1 1 1 1 1 1

13 St. Pau 1 1 1 1 1 1

14 Earl Ha 1 1 1 1 1 1

15 Lewisoh 1 1 1 1 1 1

16 Philoso 1 1 1 1 1 1

17 Buell & 1 1 1 1 1 1 1

18 Alma Ma 1 1 1 1 1 1 1

19 Dodge 1 1 1 1 1 1 1 1 1

20 Kent 1 1 1 1 1 1 1 1 1 1

21 College 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

22 Journal 1 1 1 1 1 1 1 1 1 1 1 1 1

23 Hamilto 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

24 Lion's 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

25 Lerner 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

26 Butler 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

27 Carman 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

south

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

1 Pupin 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

2 Schapir 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

3 Mudd, E 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

4 Physica 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

5 Gymnasi 1 1 1 1 1 1 1 1 1 1 1 1 1 1

6 Scherme 1 1 1 1 1 1 1 1 1 1 1 1 1 1

7 Chandle 1 1 1 1 1 1 1 1 1 1 1

8 Compute 1 1 1 1 1 1 1 1 1 1 1

9 Avery 1 1 1 1 1 1 1 1

10 Fayerwe 1 1 1 1 1 1 1

11 Mathema 1 1 1 1 1 1 1 1

12 Low Lib 1 1 1 1 1 1 1

13 St. Pau 1 1 1 1 1 1 1

14 Earl Ha 1 1 1 1 1 1 1

15 Lewisoh 1 1 1 1 1 1

16 Philoso 1 1 1 1

17 Buell & 1 1 1 1 1

18 Alma Ma 1 1 1 1

19 Dodge 1 1 1 1 1

20 Kent 1 1 1 1

21 College 1 1 1 1 1 1

22 Journal 1 1

23 Hamilto

24 Lion's

25 Lerner 1

26 Butler

27 Carman

east

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

1 Pupin 1 1

2 Schapir 1

3 Mudd, E

4 Physica 1 1 1 1

5 Gymnasi 1

6 Scherme

7 Chandle 1 1 1 1

8 Compute 1 1

9 Avery 1

10 Fayerwe

11 Mathema 1 1 1 1 1

12 Low Lib 1 1 1 1 1

13 St. Pau

14 Earl Ha 1 1 1 1 1

15 Lewisoh 1 1 1 1 1 1

16 Philoso

17 Buell & 1

18 Alma Ma 1 1

19 Dodge 1 1 1

20 Kent

21 College

22 Journal 1 1 1

23 Hamilto

24 Lion's

25 Lerner 1 1 1

26 Butler 1 1

27 Carman 1

west

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

1 Pupin

2 Schapir 1

3 Mudd, E 1 1 1 1 1

4 Physica

5 Gymnasi 1 1

6 Scherme 1 1 1 1

7 Chandle

8 Compute 1

9 Avery 1 1 1 1

10 Fayerwe 1 1 1 1 1 1

11 Mathema

12 Low Lib 1 1 1

13 St. Pau 1 1 1 1

14 Earl Ha

15 Lewisoh

16 Philoso 1 1 1 1 1

17 Buell & 1 1 1

18 Alma Ma 1 1

19 Dodge

20 Kent 1 1 1

21 College

22 Journal

23 Hamilto 1 1 1 1

24 Lion's 1 1

25 Lerner

26 Butler 1 1

27 Carman

near

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

1 Pupin 1 1

2 Schapir 1 1 1 1

3 Mudd, E 1 1 1

4 Physica 1 1 1 1

5 Gymnasi 1 1 1 1 1 1 1 1

6 Scherme 1 1 1 1

7 Chandle 1 1 1

8 Compute 1 1

9 Avery 1 1

10 Fayerwe 1 1

11 Mathema 1 1

12 Low Lib 1 1

13 St. Pau 1 1 1 1

14 Earl Ha 1 1

15 Lewisoh 1 1

16 Philoso 1 1

17 Buell & 1

18 Alma Ma

19 Dodge 1 1

20 Kent 1 1

21 College 1 1 1 1

22 Journal 1 1 1

23 Hamilto 1 1 1 1

24 Lion's 1 1

25 Lerner 1 1 1

26 Butler 1 1 1 1

27 Carman 1 1

MINIMIZED

NORTH

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

1 Pupin

2 Schapir

3 Mudd, E

4 Physica 1

5 Gymnasi 1 1

6 Scherme 1 1

7 Chandle 1

8 Compute 1 1

9 Avery 1

10 Fayerwe 1

11 Mathema 1

12 Low Lib 1 1 1

13 St. Pau 1 1 1

14 Earl Ha 1 1

15 Lewisoh 1

16 Philoso 1

17 Buell & 1

18 Alma Ma 1

19 Dodge 1 1

20 Kent 1 1

21 College 1 1 1

22 Journal 1

23 Hamilto 1

24 Lion's 1

25 Lerner 1

26 Butler 1

27 Carman 1

SOUTH

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

1 Pupin 1 1

2 Schapir 1 1

3 Mudd, E 1 1

4 Physica 1 1 1

5 Gymnasi 1 1 1

6 Scherme 1 1 1

7 Chandle 1 1

8 Compute 1 1

9 Avery 1

10 Fayerwe 1

11 Mathema 1

12 Low Lib 1

13 St. Pau 1 1

14 Earl Ha 1

15 Lewisoh 1

16 Philoso 1

17 Buell & 1

18 Alma Ma 1

19 Dodge 1

20 Kent 1

21 College 1 1 1 1

22 Journal 1

23 Hamilto

24 Lion's

25 Lerner 1

26 Butler

27 Carman

EAST

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

1 Pupin 1

2 Schapir 1

3 Mudd, E

4 Physica 1 1 1

5 Gymnasi 1

6 Scherme

7 Chandle 1 1 1

8 Compute 1 1

9 Avery 1

10 Fayerwe

11 Mathema 1 1

12 Low Lib 1 1 1

13 St. Pau

14 Earl Ha 1

15 Lewisoh 1 1

16 Philoso

17 Buell & 1

18 Alma Ma 1 1

19 Dodge 1

20 Kent

21 College

22 Journal 1 1 1

23 Hamilto

24 Lion's

25 Lerner 1

26 Butler 1 1

27 Carman 1

WEST

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

1 Pupin

2 Schapir 1

3 Mudd, E 1 1

4 Physica

5 Gymnasi 1 1

6 Scherme 1 1 1

7 Chandle

8 Compute 1

9 Avery 1 1 1

10 Fayerwe 1 1

11 Mathema

12 Low Lib 1 1 1

13 St. Pau 1

14 Earl Ha

15 Lewisoh

16 Philoso 1 1

17 Buell & 1 1

18 Alma Ma 1 1

19 Dodge

20 Kent 1

21 College

22 Journal

23 Hamilto 1 1 1 1

24 Lion's 1 1

25 Lerner

26 Butler 1 1

27 Carman

NEAR

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

1 Pupin 1 1

2 Schapir 1 1 1 1

3 Mudd, E 1 1 1

4 Physica 1 1 1 1

5 Gymnasi 1 1 1 1 1 1 1 1

6 Scherme 1 1 1 1

7 Chandle 1 1 1

8 Compute 1 1

9 Avery 1 1

10 Fayerwe 1 1

11 Mathema 1 1

12 Low Lib 1 1

13 St. Pau 1 1 1 1

14 Earl Ha 1 1

15 Lewisoh 1 1

16 Philoso 1 1

17 Buell & 1

18 Alma Ma

19 Dodge 1 1

20 Kent 1 1

21 College 1 1 1 1

22 Journal 1 1 1

23 Hamilto 1 1 1 1

24 Lion's 1 1

25 Lerner 1 1 1

26 Butler 1 1 1 1

27 Carman 1 1

1:

South of Pupin is:

Physical Fitness Center 1

Gymnasium & Uris 2

East of Pupin is:

Schapiro CEPSR 3

Near to Pupin is:

Schapiro CEPSR 4

Physical Fitness Center 5

2:

South of Schapiro CEPSR is:

Gymnasium & Uris 6

Computer Center 7

East of Schapiro CEPSR is:

Mudd, Engineering Terrace, Fairchild & Computer Science 8

West of Schapiro CEPSR is:

Pupin 9

Near to Schapiro CEPSR is:

Pupin 10

Mudd, Engineering Terrace, Fairchild & Computer Science 11

Physical Fitness Center 12

Gymnasium & Uris 13

3:

South of Mudd, Engineering Terrace, Fairchild & Computer Science is:

Schermerhorn 14

Dodge 15

West of Mudd, Engineering Terrace, Fairchild & Computer Science is:

Schapiro CEPSR 16

Gymnasium & Uris 17

Near to Mudd, Engineering Terrace, Fairchild & Computer Science is:

Schapiro CEPSR 18

Gymnasium & Uris 19

Schermerhorn 20

4:

North of Physical Fitness Center is:

Pupin 21

South of Physical Fitness Center is:

Chandler & Havemeyer 22

Computer Center 23

Buell & Maison Francaise 24

East of Physical Fitness Center is:

Schapiro CEPSR 25

Gymnasium & Uris 26

Schermerhorn 27

Near to Physical Fitness Center is:

Pupin 28

Schapiro CEPSR 29

Gymnasium & Uris 30

Chandler & Havemeyer 31

5:

North of Gymnasium & Uris is:

Pupin 32

Schapiro CEPSR 33

South of Gymnasium & Uris is:

Avery 34

Low Library 35

Dodge 36

East of Gymnasium & Uris is:

Mudd, Engineering Terrace, Fairchild & Computer Science 37

West of Gymnasium & Uris is:

Physical Fitness Center 38

Chandler & Havemeyer 39

Near to Gymnasium & Uris is:

Pupin 40

Schapiro CEPSR 41

Mudd, Engineering Terrace, Fairchild & Computer Science 42

Physical Fitness Center 43

Schermerhorn 44

Chandler & Havemeyer 45

Computer Center 46

Avery 47

6:

North of Schermerhorn is:

Schapiro CEPSR 48

Mudd, Engineering Terrace, Fairchild & Computer Science 49

South of Schermerhorn is:

Avery 50

Fayerweather 51

Low Library 52

West of Schermerhorn is:

Physical Fitness Center 53

Computer Center 54

Mathematics 55

Near to Schermerhorn is:

Mudd, Engineering Terrace, Fairchild & Computer Science 56

Gymnasium & Uris 57

Avery 58

Fayerweather 59

7:

North of Chandler & Havemeyer is:

Physical Fitness Center 60

South of Chandler & Havemeyer is:

Mathematics 61

Alma Mater 62

East of Chandler & Havemeyer is:

Gymnasium & Uris 63

Schermerhorn 64

Avery 65

Near to Chandler & Havemeyer is:

Physical Fitness Center 66

Computer Center 67

Mathematics 68

8:

North of Computer Center is:

Schapiro CEPSR 69

Physical Fitness Center 70

South of Computer Center is:

Low Library 71

Earl Hall 72

East of Computer Center is:

Schermerhorn 73

Fayerweather 74

West of Computer Center is:

Chandler & Havemeyer 75

Near to Computer Center is:

Gymnasium & Uris 76

Chandler & Havemeyer 77

9:

North of Avery is:

Schermerhorn 78

South of Avery is:

St. Paul's Chapel 79

East of Avery is:

Fayerweather 80

West of Avery is:

Computer Center 81

Mathematics 82

Earl Hall 83

Near to Avery is:

Schermerhorn 84

St. Paul's Chapel 85

10:

North of Fayerweather is:

Schermerhorn 86

South of Fayerweather is:

St. Paul's Chapel 87

West of Fayerweather is:

Avery 88

Low Library 89

Near to Fayerweather is:

Schermerhorn 90

St. Paul's Chapel 91

11:

North of Mathematics is:

Chandler & Havemeyer 92

South of Mathematics is:

Earl Hall 93

East of Mathematics is:

Schermerhorn 94

Low Library 95

Near to Mathematics is:

Chandler & Havemeyer 96

Earl Hall 97

12:

North of Low Library is:

Mudd, Engineering Terrace, Fairchild & Computer Science 98

Gymnasium & Uris 99

Computer Center 100

South of Low Library is:

Alma Mater 101

East of Low Library is:

Avery 102

St. Paul's Chapel 103

Buell & Maison Francaise 104

West of Low Library is:

Mathematics 105

Earl Hall 106

Lewisohn 107

Near to Low Library is:

Avery 108

Alma Mater 109

13:

North of St. Paul's Chapel is:

Gymnasium & Uris 110

Avery 111

Fayerweather 112

South of St. Paul's Chapel is:

Philosophy 113

Buell & Maison Francaise 114

West of St. Paul's Chapel is:

Low Library 115

Near to St. Paul's Chapel is:

Avery 116

Fayerweather 117

Philosophy 118

Buell & Maison Francaise 119

14:

North of Earl Hall is:

Computer Center 120

Mathematics 121

South of Earl Hall is:

Lewisohn 122

East of Earl Hall is:

Low Library 123

Near to Earl Hall is:

Mathematics 124

Lewisohn 125

15:

North of Lewisohn is:

Earl Hall 126

South of Lewisohn is:

Dodge 127

East of Lewisohn is:

Low Library 128

Alma Mater 129

Near to Lewisohn is:

Earl Hall 130

Dodge 131

16:

North of Philosophy is:

St. Paul's Chapel 132

South of Philosophy is:

Kent 133

West of Philosophy is:

Low Library 134

Alma Mater 135

Near to Philosophy is:

St. Paul's Chapel 136

Kent 137

17:

North of Buell & Maison Francaise is:

St. Paul's Chapel 138

South of Buell & Maison Francaise is:

Kent 139

East of Buell & Maison Francaise is:

Philosophy 140

West of Buell & Maison Francaise is:

Earl Hall 141

Alma Mater 142

Near to Buell & Maison Francaise is:

St. Paul's Chapel 143

18:

North of Alma Mater is:

Low Library 144

South of Alma Mater is:

College Walk 145

East of Alma Mater is:

Philosophy 146

Kent 147

West of Alma Mater is:

Lewisohn 148

Dodge 149

19:

North of Dodge is:

Gymnasium & Uris 150

Lewisohn 151

South of Dodge is:

College Walk 152

East of Dodge is:

Alma Mater 153

Near to Dodge is:

Lewisohn 154

College Walk 155

20:

North of Kent is:

Philosophy 156

Buell & Maison Francaise 157

South of Kent is:

College Walk 158

West of Kent is:

Alma Mater 159

Near to Kent is:

Philosophy 160

College Walk 161

21:

North of College Walk is:

Alma Mater 162

Dodge 163

Kent 164

South of College Walk is:

Journalism & Furnald 165

Hamilton, Hartley, Wallach & John Jay 166

Lion's Court 167

Butler Library 168

Near to College Walk is:

Dodge 169

Kent 170

Journalism & Furnald 171

Hamilton, Hartley, Wallach & John Jay 172

22:

North of Journalism & Furnald is:

College Walk 173

South of Journalism & Furnald is:

Lerner Hall 174

East of Journalism & Furnald is:

Kent 175

Hamilton, Hartley, Wallach & John Jay 176

Lion's Court 177

Near to Journalism & Furnald is:

Dodge 178

College Walk 179

Lerner Hall 180

23:

North of Hamilton, Hartley, Wallach & John Jay is:

College Walk 181

West of Hamilton, Hartley, Wallach & John Jay is:

Dodge 182

Journalism & Furnald 183

Lerner Hall 184

Carman 185

Near to Hamilton, Hartley, Wallach & John Jay is:

Kent 186

College Walk 187

Lion's Court 188

Butler Library 189

24:

North of Lion's Court is:

College Walk 190

West of Lion's Court is:

Lerner Hall 191

Carman 192

Near to Lion's Court is:

Hamilton, Hartley, Wallach & John Jay 193

Butler Library 194

25:

North of Lerner Hall is:

Journalism & Furnald 195

South of Lerner Hall is:

Carman 196

East of Lerner Hall is:

Butler Library 197

Near to Lerner Hall is:

Journalism & Furnald 198

Butler Library 199

Carman 200

26:

North of Butler Library is:

College Walk 201

East of Butler Library is:

Hamilton, Hartley, Wallach & John Jay 202

Lion's Court 203

West of Butler Library is:

Lerner Hall 204

Carman 205

Near to Butler Library is:

Hamilton, Hartley, Wallach & John Jay 206

Lion's Court 207

Lerner Hall 208

Carman 209

27:

North of Carman is:

Lerner Hall 210

East of Carman is:

Butler Library 211

Near to Carman is:

Lerner Hall 212

Butler Library 213

# **Appendix C: Step Three**

# ****Three Source/Goal pair clouds and descriptions:****

# OSX Lion:Users:on2valhalla:Desktop:Screen Shot 2013-04-09 at 8.18.50 AM.pngOSX Lion:Users:on2valhalla:Desktop:Screen Shot 2013-04-09 at 8.18.58 AM.png

**Pair 1:**

**Start:**

**( 192 , 382 ) is NORTH of ( the rectangular, small, north to south oriented structure )(called Lion's Court ), and NEAR ( the rectangular, small, north to south oriented structure )(called Lion's Court ), ( the U-shaped, large, north to south oriented structure )(called Hamilton, Hartley, Wallach & John Jay ),**

**Goal:**

**( 23 , 16 ) is NORTH of ( the large, east to west oriented structure )(called Physical Fitness Center ), and WEST of ( the narrow, medium sized, east to west oriented structure )(called Pupin ), and NEAR ( the large, east to west oriented structure )(called Physical Fitness Center ),**

**Pair 2:**

# 

# Start:

# ( 212 , 43 ) is NEAR ( the lumpy, large structure )(called Mudd, Engineering Terrace, Fairchild & Computer Science ),

# Goal:

# ( 127 , 447 ) is NEAR ( the cross-shaped, large, east to west oriented structure )(called Butler Library ),

# OSX Lion:Users:on2valhalla:Desktop:Screen Shot 2013-04-09 at 8.18.41 AM.png

**Pair 3:**

# Start:

# ( 13 , 223 ) is NORTH of ( the I-shaped, medium sized, north to south oriented structure )(called Lewisohn ), and WEST of ( the cross-shaped, small, east to west oriented structure )(called Earl Hall ), and NEAR ( the I-shaped, medium sized, north to south oriented structure )(called Lewisohn ),

# Goal:

# ( 239 , 150 ) is WEST of ( the cross-shaped, medium sized, north to south oriented structure )(called Fayerweather ), and NEAR ( the cross-shaped, medium sized, north to south oriented structure )(called Fayerweather ), ( the lumpy, large structure )(called Schermerhorn ),

# Code

MapReader.pro

#-------------------------------------------------

#

# Project created by QtCreator 2013-04-06T10:03:39

#

#-------------------------------------------------

QT += core gui

#QMAKE\_CXX = /opt/local/bin/g++

QMAKE\_CXXFLAGS = -fpermissive -std=c++11

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

TARGET = MapReader

TEMPLATE = app

SOURCES += main.cpp\

mapreader.cpp

HEADERS += mapreader.h \

clicklabel.h

FORMS += mapreader.ui

INCLUDEPATH += /usr/local/include/

LIBS += -L/usr/local/lib/ \

-lopencv\_core \

-lopencv\_highgui \

-lopencv\_imgproc \

-lopencv\_features2d \

-lopencv\_objdetect \

-lopencv\_calib3d

OTHER\_FILES += \

ass3-table.txt \

ass3-labeled.pgm \

ass3-campus.pgm

main.cpp

#include "mapreader.h"

#include "clicklabel.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MapReader w;

w.show();

return a.exec();

}

mapreader.h

#ifndef MAPREADER\_H

#define MAPREADER\_H

#include <QMainWindow>

#include <QMouseEvent>

#include <QDebug>

#include <string>

#include <iostream>

#include <vector>

#include <queue>

#include <fstream>

#include <algorithm>

#include <unordered\_set>

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc/imgproc.hpp>

using namespace std;

using namespace cv;

namespace Ui {

class MapReader;

}

struct Building {

int number;

string name;

Moments mom;

Point centerOfMass;

vector<Point> contour;

unordered\_set< int > north;

unordered\_set< int > south;

unordered\_set< int > east;

unordered\_set< int > west;

unordered\_set< int > near;

Rect MBR;

string description;

Building()

{

}

Building(Point pt, int area)

{

this->number = -1;

this->name = "";

this->contour.push\_back(pt);

this->mom = moments(Mat(contour));

this->mom.m00 = area;

this->centerOfMass = pt;

this->MBR = Rect(pt, Size(1,1));

this->description = "";

}

bool operator==(const Building &rhs)

{

return (centerOfMass == rhs.centerOfMass)

&& (number == rhs.number);

}

friend ostream& operator<< (ostream &o, const Building &b)

{

o << b.number << " (" << b.name << "): "

<< "CoM: " << b.centerOfMass << " Area: " << b.mom.m00

<< " Bound UL: [" << b.MBR.x << ", " << b.MBR.y

<< "] LR: [" << b.MBR.x + b.MBR.width << ", " << b.MBR.y +b.MBR.height

<< "]\n Description: " << b.description;

return o;

}

friend QDebug operator<< (QDebug q, const Building &b)

{

q << b.number << " (" << b.name.c\_str() << "): "

<< "CoM: (" << b.centerOfMass.y << ", "

<< b.centerOfMass.x << ") Area: " << b.mom.m00

<< " Bound UL: [" << b.MBR.x << ", " << b.MBR.y

<< "] LR: [" << b.MBR.x + b.MBR.width << ", " << b.MBR.y +b.MBR.height

<< "]\n Description: " << b.description.c\_str();

return q;

}

};

namespace std

{

// Hash combination emulates from Boost library

template<>

class hash<Point> {

public:

size\_t operator()(const Point &pt) const

{

hash< int > iHash;

size\_t hash = iHash(pt.x);

hash ^= iHash(pt.y)

+ 0x9e3779b9 + (hash << 6) + (hash >> 2);

return hash;

}

};

}

class MapReader : public QMainWindow

{

Q\_OBJECT

public:

explicit MapReader(QWidget \*parent = 0);

~*MapReader*();

void displayMat(const cv::Mat& image);

void getNames(string fileName);

void processFeatures(const cv::Mat& image, cv::Mat& wRects);

void describeBuildings();

void findRelations();

void printRelations(const vector< vector< bool > > &rel);

void printRelations();

void printBinaryPairs();

// void printFeatures(const Point &pt, const vector< unordered\_set< int > > &features);

// vector<Point> findCloud(const Point &pt, const vector<unordered\_set<int> > &features);

void retrMinFeatures(Building &buildPt);

void printFeatures(const Building &buildPt);

vector<Point> findCloud(const Building &buildPt);

void drawCloud(vector<Point> cloud);

void clicked(QMouseEvent \*e);

bool north(const Building &s, const Building &g);

bool south(const Building &s, const Building &g);

bool east(const Building &s, const Building &g);

bool west(const Building &s, const Building &g);

bool near(const Building &s, const Building &g);

vector<Building> search();

vector<Building> dfs(const vector<Building> &moves);

private slots:

void on\_btnStart\_clicked();

private:

Ui::MapReader \*ui;

Mat campusImage;

Mat campusLabeled;

Mat cloudImage;

vector<Building> buildings;

vector< vector< bool > > northR;

vector< vector< bool > > southR;

vector< vector< bool > > eastR;

vector< vector< bool > > westR;

vector< vector< bool > > nearR;

vector<Building> currentPath;

vector<int> distances;

int curIdx;

Building source, goal;

bool ready, takeinput;

int minArea, maxArea, avgArea, areaRange;

};

#endif // MAPREADER\_H

mapreader.cpp

#include "mapreader.h"

#include "ui\_mapreader.h"

MapReader::MapReader(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MapReader)

{

ui->setupUi(this);

ui->lblImage->setParent(this);

campusImage = imread("../../../../../ass3-campus.pgm", CV\_LOAD\_IMAGE\_GRAYSCALE);

campusLabeled = imread("../../../../../ass3-labeled.pgm", CV\_LOAD\_IMAGE\_GRAYSCALE);

getNames("../../../../../ass3-table.txt");

// cout << campusLabeled.rows << " " << campusLabeled.cols << " "

// << (int)campusLabeled.at<uchar>(40,7) << endl;

Mat wRects(campusImage.size(), CV\_8UC1, Scalar(0));

processFeatures(campusLabeled, wRects);

namedWindow("Map");

imshow("Map", wRects);

describeBuildings();

findRelations();

// for(uchar i = 0; i < buildings.size() ; i++)

// for( uchar j = 0; j < buildings.size(); j++)

// cout << buildings[i] << endl;

// if(near( buildings[j], buildings[i]) )

// cout << (int) buildings[i].number << ": " << buildings[i].name

// << " is near to " << buildings[j].name << endl;

printBinaryPairs();

cvtColor(campusImage, campusImage, CV\_GRAY2BGR);

displayMat(campusImage);

ready = true;

takeinput = false;

}

MapReader::~*MapReader*()

{

delete ui;

}

void MapReader::getNames(string fileName)

{

ifstream iFile;

iFile.open(fileName.c\_str());

string line;

while(getline(iFile, line))

{

size\_t eq = line.find\_first\_of("=");

size\_t f = line.find\_first\_of("\"", eq+1) + 1;

size\_t e = line.find\_first\_of("\"", f+1);

Building b;

b.number = atoi(line.substr(0,eq).c\_str());

b.name = line.substr(f, e-f);

buildings.push\_back(b);

// cout << (int) b.number << " " << b.name << endl;

}

}

void MapReader::processFeatures(const Mat& labeled, Mat& wRects)

{

// find contours in the filtered image

vector< vector<Point> > contours;

vector< Vec4i> hierarchy;

cv::findContours(labeled.clone(),

contours, // a vector of contours

hierarchy, // a hierarchy of contours if there are parent

//child relations in the image

CV\_RETR\_EXTERNAL, // retrieve the external contours

CV\_CHAIN\_APPROX\_SIMPLE); // an approximation algorithm

// iterate through all the top-level contours

for(char idx = 0 ; idx >= 0; idx = hierarchy[idx][0] )

{

char number = labeled.at<uchar>(contours[idx][0]);

Building (\*b) = &buildings[number - 1];

(\*b).contour = contours[idx];

(\*b).MBR = boundingRect(contours[idx]);

(\*b).mom = cv::moments(cv::Mat(contours[idx]));

(\*b).centerOfMass = cv::Point((\*b).mom.m10/(\*b).mom.m00,

(\*b).mom.m01/(\*b).mom.m00);

// cout << (int) (\*b).number << endl;

rectangle(wRects, (\*b).MBR, cv::Scalar(128), 2);

drawContours(wRects, contours, idx, Scalar(255));

circle( wRects, (\*b).centerOfMass, 3, Scalar(128), 1 );

circle( wRects, Point( (\*b).MBR.x + (\*b).MBR.width/2, (\*b).MBR.y + (\*b).MBR.height/2 ),

3, Scalar(255), 1 );

// wRects += labeled;

}

}

void MapReader::describeBuildings()

{

// Compute ranges/averages for size

minArea = campusImage.rows \* campusImage.cols;

maxArea =0;

int totalArea =0;

for( uchar i = 0; i < buildings.size(); i++)

{

Building b = buildings[i];

totalArea += b.mom.m00;

if(b.mom.m00 < minArea)

minArea = b.mom.m00;

if(b.mom.m00 > maxArea)

maxArea = b.mom.m00;

}

avgArea = totalArea / buildings.size();

areaRange = maxArea - minArea;

for( uchar i = 0; i < buildings.size(); i++)

{

Building &b = buildings[i];

b.description = "the ";

// b.description = "";

int centerx = b.MBR.x + b.MBR.width/2, centery = b.MBR.y + b.MBR.height/2;

//FIND SHAPE DESCRIPTORS

if( abs(b.centerOfMass.x - (b.MBR.x + b.MBR.width/2)) + abs(b.centerOfMass.y - (b.MBR.y + b.MBR.height/2))

>= campusImage.cols \* .025 )

{

//Center of mass is different from center of bounding rectangle

if( campusImage.at<uchar>( b.MBR.y + b.MBR.height/2, b.MBR.x + b.MBR.width/2 ) != 0

|| campusImage.at<uchar>(b.centerOfMass) != 0)

b.description += "lumpy, ";

else

{

//differ between u-shaped and l-shaped

bool north = false, south = false, east = false, west = false ;

for(int i = 1; i < b.MBR.height/2 - 1; i++)

{

if( (north && south) || (east && west) )

break;

if(!north)

if (campusImage.at<uchar>(centery - i, centerx ) != 0 )

north = true;

if(!south)

if (campusImage.at<uchar>(centery + i, centerx ) != 0 )

south = true;

if(!east)

if (campusImage.at<uchar>(centery, centerx + i ) != 0 )

east = true;

if(!west)

if (campusImage.at<uchar>(centery, centerx - i ) != 0 )

west = true;

}

if ((north && south) || (east && west))

b.description += "U-shaped, ";

else

b.description += "L-shaped, ";

}

}

else if(b.MBR.width > b.MBR.height \* 3 || b.MBR.height > b.MBR.width \* 3 )

b.description += "narrow, ";

else if( (campusImage.at<uchar>(centery, b.MBR.x + 1) == 0

&& campusImage.at<uchar>(centery, b.MBR.x + b.MBR.width - 1) == 0)

||(campusImage.at<uchar>(b.MBR.y + 1, centerx) == 0

&& campusImage.at<uchar>(b.MBR.y + b.MBR.height - 1, centerx) == 0) )

b.description += "I-shaped, ";

else if( campusImage.at<uchar>(b.MBR.y + 2, b.MBR.x + 2) == 0

&& campusImage.at<uchar>(b.MBR.y + 2, b.MBR.x + b.MBR.width - 2) == 0

&& campusImage.at<uchar>(b.MBR.y + b.MBR.height - 2, b.MBR.x + 2) == 0

&& campusImage.at<uchar>(b.MBR.y + b.MBR.height - 2, b.MBR.x + b.MBR.width - 2) == 0 )

b.description += "cross-shaped, ";

else if( b.mom.m00 >= b.MBR.width \* b.MBR.height \* .85)

{

if( abs(b.MBR.width - b.MBR.height) < .3 \* b.MBR.width )

b.description += "squarish, ";

else

b.description += "rectangular, ";

}

// else

// cout << (int) b.number << " "<< b.name << " " << (int)campusImage.at<uchar>(centery, b.MBR.x + 1) << " "

// << (int)campusImage.at<uchar>(centery, b.MBR.x + b.MBR.width - 1) << " "

// << (int)campusImage.at<uchar>(b.MBR.y + 1, centerx) << " "

// << (int)campusImage.at<uchar>(b.MBR.y + b.MBR.height - 1, centerx) << endl;

// cout << (int) b.number << " "<< b.name << " " << b.description << endl;

// FIND SIZE DESCRIPTORS

if( b.mom.m00 > avgArea + areaRange / 6)

b.description += "large";

else if( b.mom.m00 < avgArea - areaRange / 4)

b.description += "small";

else

b.description += "medium sized";

//ORIENTATION descriptors

if (b.MBR.width > b.MBR.height \* 1.5)

b.description += ", east to west oriented structure";

else if ( b.MBR.height > b.MBR.width \* 1.5)

b.description += ", north to south oriented structure";

else

b.description += " structure";

// // LOCATION descriptors

// string ending = "erly ";

// if (b.centerOfMass.y < campusImage.rows / 3)

// {

// b.description += "north";

// if(b.centerOfMass.x < campusImage.cols / 3)

// b.description += "-west";

// if(b.centerOfMass.x > campusImage.cols - campusImage.cols / 3)

// b.description += "-east";

// }

// else if (b.centerOfMass.y > campusImage.rows - campusImage.rows / 3)

// {

// b.description += "south";

// if(b.centerOfMass.x < campusImage.cols / 3)

// b.description += "-west";

// if(b.centerOfMass.x > campusImage.cols - campusImage.cols / 3)

// b.description += "-east";

// }

// else if(b.centerOfMass.x < campusImage.cols / 3)

// b.description += "west";

// else if(b.centerOfMass.x > campusImage.cols - campusImage.cols / 3)

// b.description += "east";

// else

// {

// ending = " ";

// b.description += "centered";

// }

// b.description += ending + "structure";

cout << b << endl;

}

}

//Convert cv::Mat to QImage and display

void MapReader::displayMat(const cv::Mat& image)

{

//BGR openCV Mat to QImage

QImage img\_qt = QImage((const unsigned char\*)image.data,image.cols, image.rows, image.step, QImage::Format\_RGB888);

//For Binary Images

if (img\_qt.isNull()){

//ColorTable for Binary Images

QVector<QRgb> colorTable;

for (int i = 0; i < 256; i++)

colorTable.push\_back(qRgb(i, i, i));

img\_qt = QImage((const unsigned char\*)image.data,image.cols, image.rows, image.step, QImage::Format\_Indexed8);

img\_qt.setColorTable(colorTable);

}

//Display the QImage in the Label

QPixmap img\_pix = QPixmap::fromImage(img\_qt.rgbSwapped()); //BGR to RGB

this->ui->lblImage->setPixmap(img\_pix.scaled(ui->lblImage->size(), Qt::KeepAspectRatio));

}

bool MapReader::north(const Building &s, const Building &g)

{

if((g.MBR.width > campusImage.cols \*.9 || s.MBR.width > campusImage.cols \* .9)

&& g.MBR.y + g.MBR.height < s.MBR.y)

return true;

if(g.MBR.y + g.MBR.height \* .8 < s.MBR.y

&& (abs(g.centerOfMass.x - s.centerOfMass.x) \* 2 < abs(g.centerOfMass.y - s.centerOfMass.y)

|| abs(g.centerOfMass.x - s.MBR.x) \* 2 < abs(g.centerOfMass.y - s.centerOfMass.y)

|| abs(g.centerOfMass.x - (s.MBR.x + s.MBR.width)) \* 2 < abs(g.centerOfMass.y - s.centerOfMass.y)))

return true;

else

return false;

}

bool MapReader::south(const Building &s, const Building &g)

{

if((g.MBR.width > campusImage.cols \*.9 || s.MBR.width > campusImage.cols \* .9)

&& g.MBR.y > s.MBR.y + s.MBR.height)

return true;

if(g.MBR.y > s.MBR.y + s.MBR.height \* .8

&& (abs(g.centerOfMass.x - s.centerOfMass.x) \* 2 < abs(g.centerOfMass.y - s.centerOfMass.y)

|| abs(g.centerOfMass.x - s.MBR.x) \* 2 < abs(g.centerOfMass.y - s.centerOfMass.y)

|| abs(g.centerOfMass.x - (s.MBR.x + s.MBR.width)) \* 2 < abs(g.centerOfMass.y - s.centerOfMass.y)))

return true;

else

return false;

}

bool MapReader::east(const Building &s, const Building &g)

{

if(g.MBR.width > campusImage.cols \*.9 || s.MBR.width > campusImage.cols \* .9)

return false;

if(g.MBR.x > s.MBR.x + s.MBR.width \* .8

&& (abs(g.centerOfMass.y - s.centerOfMass.y) \* 5 < abs(g.centerOfMass.x - s.centerOfMass.x)

|| abs(g.centerOfMass.y - s.MBR.y) \* 5 < abs(g.centerOfMass.x - s.centerOfMass.x)

|| abs(g.centerOfMass.y - (s.MBR.y + s.MBR.height)) \* 5 < abs(g.centerOfMass.x - s.centerOfMass.x)))

return true;

else

return false;

}

bool MapReader::west(const Building &s, const Building &g)

{

if(g.MBR.width > campusImage.cols \*.9 || s.MBR.width > campusImage.cols \* .9)

return false;

if(g.MBR.x + g.MBR.width \* .8 < s.MBR.x

&& (abs(g.centerOfMass.y - s.centerOfMass.y) \* 5 < abs(g.centerOfMass.x - s.centerOfMass.x)

|| abs(g.centerOfMass.y - s.MBR.y) \* 5 < abs(g.centerOfMass.x - s.centerOfMass.x)

|| abs(g.centerOfMass.y - (s.MBR.y + s.MBR.height)) \* 5 < abs(g.centerOfMass.x - s.centerOfMass.x)))

return true;

else

return false;

}

bool MapReader::near(const Building &s, const Building &g)

{

if(s.MBR == g.MBR)

return false;

// adjust for size of structure

int defaultDistance = (campusImage.cols + campusImage.rows) / 40;

int adjustment = (s.mom.m00/avgArea) \* 10;

int nearDistance = adjustment + defaultDistance;

// cout << s.name << " " << nearDistance << endl;

// accept if the MBRs overlap

if((s.MBR & g.MBR).area() > 0)

return true;

if( (g.MBR.width > campusImage.cols \*.9 || s.MBR.width > campusImage.cols \* .9)

&& (abs(g.centerOfMass.y - s.centerOfMass.y) < nearDistance

|| abs(g.MBR.y - s.centerOfMass.y) < nearDistance

|| abs(g.centerOfMass.y - s.MBR.y) < nearDistance

|| abs(g.MBR.y - s.MBR.y) < nearDistance ) )

return true;

if(abs(g.centerOfMass.y - s.centerOfMass.y)

+ abs(g.centerOfMass.x - s.centerOfMass.x) > 8 \* defaultDistance)

return false;

for(uchar i = 0; i < s.contour.size(); i++)

{

Point sP = s.contour[i];

for(uchar j = 0; j < g.contour.size(); j++)

{

Point gP = g.contour[j];

if(abs(sP.x - gP.x) + abs(sP.y - gP.y) < nearDistance )

return true;

}

}

return false;

}

void MapReader::findRelations()

{

northR = vector< vector< bool > >();

southR = vector< vector< bool > >();

eastR = vector< vector< bool > >();

westR = vector< vector< bool > >();

nearR = vector< vector< bool > >();

//FIND ALL RELATIONS

for(uchar i = 0; i < buildings.size(); i++)

{

northR.push\_back( vector< bool >() );

southR.push\_back( vector< bool >() );

eastR.push\_back( vector< bool >() );

westR.push\_back( vector< bool >() );

nearR.push\_back( vector< bool >() );

for(uchar j = 0; j < buildings.size(); j++)

{

northR[i].push\_back( north( buildings[i], buildings[j] ));

southR[i].push\_back( south( buildings[i], buildings[j] ));

eastR[i].push\_back( east( buildings[i], buildings[j] ));

westR[i].push\_back( west( buildings[i], buildings[j] ));

nearR[i].push\_back( near( buildings[i], buildings[j] ));

}

}

for(uchar i = 0; i < buildings.size(); i++)

for(uchar j = 0; j < buildings.size(); j++)

{

if( northR[i][j] )

buildings[i].north.insert(j);

if( southR[i][j] )

buildings[i].south.insert(j);

if( eastR[i][j] )

buildings[i].east.insert(j);

if( westR[i][j] )

buildings[i].west.insert(j);

if( nearR[i][j] )

buildings[i].near.insert(j);

}

cout << "\n\n\nCOMPLETE\n";

cout << "north\n";

printRelations(northR);

cout << "south\n";

printRelations(southR);

cout << "east\n";

printRelations(eastR);

cout << "west\n";

printRelations(westR);

cout << "near\n";

printRelations(nearR);

// eliminate duplicate compass relations via transitivity

for(uchar i = 0; i < buildings.size(); i++)

for(uchar j = 0; j < buildings.size(); j++)

{

if(i == j)

continue;

if( northR[i][j] )

for( uchar k = 0; k < buildings.size(); k++ )

if( northR[j][k] )

buildings[i].north.erase(k);

if( southR[i][j] )

for( uchar k = 0; k < buildings.size(); k++ )

if( southR[j][k] )

buildings[i].south.erase(k);

if( eastR[i][j] )

for( uchar k = 0; k < buildings.size(); k++ )

if( eastR[j][k] )

buildings[i].east.erase(k);

if( westR[i][j] )

for( uchar k = 0; k < buildings.size(); k++ )

if( westR[j][k] )

buildings[i].west.erase(k);

}

// // eliminate further duplicates from near relations

// for(uchar i = 0; i < buildings.size(); i++)

// for(uchar j = 0; j < buildings.size(); j++)

// if( nearR[i][j] )

// for( uchar k = 0; k < buildings.size(); k++ )

// {

// if( buildings[i].north.count(k) > 0

// && buildings[j].north.count(k) > 0 )

// {

// if( buildings[i].mom.m00 < buildings[j].mom.m00 )

// buildings[i].north.erase(k);

// else

// buildings[j].north.erase(k);

// }

// if( buildings[i].south.count(k) > 0

// && buildings[j].south.count(k) > 0 )

// {

// if( buildings[i].mom.m00 < buildings[j].mom.m00 )

// buildings[i].south.erase(k);

// else

// buildings[j].south.erase(k);

// }

// if( buildings[i].east.count(k) > 0

// && buildings[j].east.count(k) > 0 )

// {

// if( buildings[i].mom.m00 < buildings[j].mom.m00 )

// buildings[i].east.erase(k);

// else

// buildings[j].east.erase(k);

// }

// if( buildings[i].west.count(k) > 0

// && buildings[j].west.count(k) > 0 )

// {

// if( buildings[i].mom.m00 < buildings[j].mom.m00 )

// buildings[i].west.erase(k);

// else

// buildings[j].west.erase(k);

// }

// }

cout << "\n\n\nMINIMIZED\n";

printRelations();

}

void MapReader::printRelations(const vector< vector< bool > > &rel)

{

cout << " ";

for(uchar i = 0; i < buildings.size() ; i++)

{

if(i < 9)

cout << " " << i+1 << " ";

else

cout << " " << i+1;

}

cout << endl;

for(uchar i = 0; i < buildings.size() ; i++)

{

if(i < 9)

cout << " " << i+1 << " " ;

else

cout << i+1 << " " ;

cout << buildings[i].name.substr(0,7);

int buffer = 7 - buildings[i].name.size();

for (int i = 0; i < buffer; ++i)

cout << " ";

for( uchar j = 0; j < buildings.size(); j++)

{

if (rel[i][j])

cout << " " << rel[i][j] << " ";

else

cout << " ";

}

cout << endl;

}

cout << endl;

}

void MapReader::printRelations()

{

cout << "NORTH\n ";

for(uchar i = 0; i < buildings.size() ; i++)

{

if(i < 9)

cout << " " << i+1 << " ";

else

cout << " " << i+1;

}

cout << endl;

for(uchar i = 0; i < buildings.size() ; i++)

{

if(i < 9)

cout << " " << i+1 << " " ;

else

cout << i+1 << " " ;

cout << buildings[i].name.substr(0,7);

int buffer = 7 - buildings[i].name.size();

for (int i = 0; i < buffer; ++i)

cout << " ";

for( uchar j = 0; j < buildings.size(); j++)

{

if ( buildings[i].north.count(j) > 0 )

cout << " " << 1 << " ";

else

cout << " ";

}

cout << endl;

}

cout << endl;

cout << "SOUTH\n ";

for(uchar i = 0; i < buildings.size() ; i++)

{

if(i < 9)

cout << " " << i+1 << " ";

else

cout << " " << i+1;

}

cout << endl;

for(uchar i = 0; i < buildings.size() ; i++)

{

if(i < 9)

cout << " " << i+1 << " " ;

else

cout << i+1 << " " ;

cout << buildings[i].name.substr(0,7);

int buffer = 7 - buildings[i].name.size();

for (int i = 0; i < buffer; ++i)

cout << " ";

for( uchar j = 0; j < buildings.size(); j++)

{

if ( buildings[i].south.count(j) > 0 )

cout << " " << 1 << " ";

else

cout << " ";

}

cout << endl;

}

cout << endl;

cout << "EAST\n ";

for(uchar i = 0; i < buildings.size() ; i++)

{

if(i < 9)

cout << " " << i+1 << " ";

else

cout << " " << i+1;

}

cout << endl;

for(uchar i = 0; i < buildings.size() ; i++)

{

if(i < 9)

cout << " " << i+1 << " " ;

else

cout << i+1 << " " ;

cout << buildings[i].name.substr(0,7);

int buffer = 7 - buildings[i].name.size();

for (int i = 0; i < buffer; ++i)

cout << " ";

for( uchar j = 0; j < buildings.size(); j++)

{

if ( buildings[i].east.count(j) > 0 )

cout << " " << 1 << " ";

else

cout << " ";

}

cout << endl;

}

cout << endl;

cout << "WEST\n ";

for(uchar i = 0; i < buildings.size() ; i++)

{

if(i < 9)

cout << " " << i+1 << " ";

else

cout << " " << i+1;

}

cout << endl;

for(uchar i = 0; i < buildings.size() ; i++)

{

if(i < 9)

cout << " " << i+1 << " " ;

else

cout << i+1 << " " ;

cout << buildings[i].name.substr(0,7);

int buffer = 7 - buildings[i].name.size();

for (int i = 0; i < buffer; ++i)

cout << " ";

for( uchar j = 0; j < buildings.size(); j++)

{

if ( buildings[i].west.count(j) > 0 )

cout << " " << 1 << " ";

else

cout << " ";

}

cout << endl;

}

cout << endl;

cout << "NEAR\n ";

for(uchar i = 0; i < buildings.size() ; i++)

{

if(i < 9)

cout << " " << i+1 << " ";

else

cout << " " << i+1;

}

cout << endl;

for(uchar i = 0; i < buildings.size() ; i++)

{

if(i < 9)

cout << " " << i+1 << " " ;

else

cout << i+1 << " " ;

cout << buildings[i].name.substr(0,7);

int buffer = 7 - buildings[i].name.size();

for (int i = 0; i < buffer; ++i)

cout << " ";

for( uchar j = 0; j < buildings.size(); j++)

{

if ( buildings[i].near.count(j) > 0 )

cout << " " << 1 << " ";

else

cout << " ";

}

cout << endl;

}

cout << endl;

}

void MapReader::printBinaryPairs()

{

int relCount = 1;

for(uchar i = 0; i < buildings.size(); i++)

{

cout << (int)buildings[i].number << ":\n";

if(buildings[i].north.size() > 0 )

{

cout << "North of " << buildings[i].name << " is:\n";

for(uchar j = 0; j <buildings.size(); j++)

if( buildings[i].north.count(j) > 0 )

cout << "\t" << buildings[j].name << "\t" << relCount++ << endl;

}

if(buildings[i].south.size() > 0 )

{

cout << "South of " << buildings[i].name << " is:\n";

for(uchar j = 0; j <buildings.size(); j++)

if( buildings[i].south.count(j) > 0 )

cout << "\t" << buildings[j].name << "\t" << relCount++ << endl;

}

if(buildings[i].east.size() > 0 )

{

cout << "East of " << buildings[i].name << " is:\n";

for(uchar j = 0; j <buildings.size(); j++)

if( buildings[i].east.count(j) > 0 )

cout << "\t" << buildings[j].name << "\t" << relCount++ << endl;

}

if(buildings[i].west.size() > 0 )

{

cout << "West of " << buildings[i].name << " is:\n";

for(uchar j = 0; j <buildings.size(); j++)

if( buildings[i].west.count(j) > 0 )

cout << "\t" << buildings[j].name << "\t" << relCount++ << endl;

}

if(buildings[i].near.size() > 0 )

{

cout << "Near to " << buildings[i].name << " is:\n";

for(uchar j = 0; j <buildings.size(); j++)

if( buildings[i].near.count(j) > 0 )

cout << "\t" << buildings[j].name << "\t" << relCount++ << endl;

}

cout << endl << endl;

}

}

void MapReader::printFeatures(const Building &buildPt)

{

QDebug debugMessage(QtDebugMsg);

bool last = true;

string join = "";

debugMessage << "(" << buildPt.centerOfMass.x

<< ", " << buildPt.centerOfMass.y << ") is ";

for(auto i = buildPt.north.begin(); i != buildPt.north.end(); i++)

debugMessage << "(" << buildings[\*i].description.c\_str()

<< ")(called " << buildings[\*i].name.c\_str() << "), ";

if(buildPt.north.size() > 0)

debugMessage << join.c\_str() << "are NORTH of ";

else

last = false;

if(last)

join = "and ";

last = true;

for(auto i = buildPt.south.begin(); i != buildPt.south.end(); i++)

debugMessage << "(" << buildings[\*i].description.c\_str()

<< ")(called " << buildings[\*i].name.c\_str() << "), ";

if(buildPt.south.size() > 0)

debugMessage << join.c\_str() << "are SOUTH of ";

else

last = false;

if(last)

join = "and ";

last = true;

for(auto i = buildPt.east.begin(); i != buildPt.east.end(); i++)

debugMessage << "(" << buildings[\*i].description.c\_str()

<< ")(called " << buildings[\*i].name.c\_str() << "), ";

if(buildPt.east.size() > 0)

debugMessage << join.c\_str() << "are EAST of ";

else

last = false;

if(last)

join = "and ";

last = true;

for(auto i = buildPt.west.begin(); i != buildPt.west.end(); i++)

debugMessage << "(" << buildings[\*i].description.c\_str()

<< ")(called " << buildings[\*i].name.c\_str() << "), ";

if(buildPt.west.size() > 0)

debugMessage << join.c\_str() << "are WEST of ";

else

last = false;

if(last)

join = "and ";

last = true;

for(auto i = buildPt.near.begin(); i != buildPt.near.end(); i++)

debugMessage << "(" << buildings[\*i].description.c\_str()

<< ")(called " << buildings[\*i].name.c\_str() << "), ";

if(buildPt.near.size() > 0)

debugMessage << join.c\_str() << "are NEAR ";

else

last = false;

}

vector<Point> MapReader::findCloud(const Building &buildPt)

{

vector<Point> cloud;

queue<Point> children;

unordered\_set<Point, hash<Point> > seen;

Point curPoint;

int count = 1;

children.push(buildPt.centerOfMass);

while(!children.empty())

{

curPoint = children.front();

children.pop();

if(seen.find(curPoint) != seen.end())

continue;

seen.insert(curPoint);

cloud.push\_back(curPoint);

Building curBuild(curPoint, minArea);

retrMinFeatures(curBuild);

if( buildPt.north != curBuild.north

|| buildPt.south != curBuild.south

|| buildPt.east != curBuild.east

|| buildPt.west != curBuild.west

|| buildPt.near != curBuild.near )

{

cloud.pop\_back();

continue;

}

// Get Children

Point n = Point(curPoint.x, curPoint.y-1),

s = Point(curPoint.x, curPoint.y+1),

e = Point(curPoint.x+1, curPoint.y),

w = Point(curPoint.x-1, curPoint.y);

if(seen.count(w) == 0 && curPoint.x != 0)

children.push(w);

if(seen.count(e) == 0 && curPoint.x != campusImage.cols)

children.push(e);

if(seen.count(n) == 0 && curPoint.y != 0)

children.push(n);

if(seen.count(s) == 0 && curPoint.y != campusImage.rows)

children.push(s);

count++;

}

return cloud;

}

void MapReader::retrMinFeatures(Building &buildPt)

{

if( campusImage.at<Vec3b>(buildPt.centerOfMass) == Vec3b(255,255,255) )

{

buildPt.near.insert(campusLabeled.at<char>(buildPt.centerOfMass) - 1);

return;

}

int minDistN, minDistS, minDistE, minDistW, minN, minS, minE, minW;

minDistN=minDistS=minDistE=minDistW = campusImage.rows + campusImage.cols;

for(uchar i = 0; i < buildings.size(); i++)

{

int dist = abs(buildings[i].centerOfMass.x - buildPt.centerOfMass.x)

+ abs(buildings[i].centerOfMass.y - buildPt.centerOfMass.y);

if( near( buildings[i], buildPt ) )

buildPt.near.insert(i);

if( north( buildings[i], buildPt ) && dist < minDistN)

{

minDistN = dist;

minN = i;

}

if( south( buildings[i], buildPt ) && dist < minDistS)

{

minDistS = dist;

minS = i;

}

if( east( buildings[i], buildPt ) && dist < minDistE)

{

minDistE = dist;

minE = i;

}

if( west( buildings[i], buildPt ) && dist < minDistW)

{

minDistW = dist;

minW = i;

}

}

//Keep only the closest direction relation

if( minDistN < campusImage.rows + campusImage.cols )

buildPt.north.insert(minN);

if( minDistS < campusImage.rows + campusImage.cols )

buildPt.south.insert(minS);

if( minDistE < campusImage.rows + campusImage.cols )

buildPt.east.insert(minE);

if( minDistW < campusImage.rows + campusImage.cols )

buildPt.west.insert(minW);

for(auto i = buildPt.near.begin(); i != buildPt.near.end(); i++)

{

if ( !buildPt.north.empty()

&& north( buildings[\*(buildPt.north.begin())], buildings[\*i]) )

buildPt.north.clear();

if ( !buildPt.south.empty()

&& south( buildings[\*(buildPt.south.begin())], buildings[\*i]) )

buildPt.south.clear();

if ( !buildPt.east.empty()

&& east( buildings[\*(buildPt.east.begin())], buildings[\*i]) )

buildPt.east.clear();

if ( !buildPt.west.empty()

&& west( buildings[\*(buildPt.west.begin())], buildings[\*i]) )

buildPt.west.clear();

}

}

void MapReader::clicked(QMouseEvent \*e)

{

QDebug debugMessage(QtDebugMsg);

if(takeinput)

{

Point pt(e->x(), e->y());

int dist = abs(pt.x - currentPath[curIdx].centerOfMass.x)

+ abs(pt.y - currentPath[curIdx].centerOfMass.y);

debugMessage << dist << " from intended target";

distances.push\_back(dist);

if(curIdx == currentPath.size() -1)

{

takeinput == false;

return;

}

int i = currentPath[curIdx].number;

int j = currentPath[curIdx + 1].number;

debugMessage << "Now go ";

string join = "";

printFeatures(currentPath[curIdx]);

if(!buildings[i].north.empty() && buildings[i].north.count(j) > 0)

{

debugMessage << "north ";

join = "and ";

}

if(!buildings[i].south.empty() && buildings[i].south.count(j) > 0)

{

debugMessage << join.c\_str() << "south ";

join = "and ";

}

if(!buildings[i].east.empty() && buildings[i].east.count(j) > 0)

{

debugMessage << join.c\_str() << "east ";

join = "and ";

}

if(!buildings[i].west.empty() && buildings[i].west.count(j) > 0)

{

debugMessage << join.c\_str() << "west ";

join = "and ";

}

if(!buildings[i].near.empty() && buildings[i].near.count(j) > 0)

{

debugMessage << join.c\_str() << "near ";

}

curIdx++;

return;

}

// cout << e->x() << "," << e->y() << endl;

Point pt(e->x(), e->y());

Building buildPt(pt, minArea);

retrMinFeatures(buildPt);

vector<Point> cloud = findCloud(buildPt);

if(!ready)

{

drawCloud(cloud);

goal = buildPt;

ready = true;

qDebug() << "Goal: ";

}

else

{

cloudImage = campusImage.clone();

drawCloud(cloud);

source = buildPt;

ready = false;

qDebug() << "Start: ";

}

printFeatures(buildPt);

}

void MapReader::drawCloud(vector<Point> cloud)

{

for (std::vector<Point>::iterator i = cloud.begin(); i != cloud.end(); ++i)

{

if( !ready )

cloudImage.at<Vec3b>(\*i) = Vec3b(0,0,100);

else

cloudImage.at<Vec3b>(\*i) = Vec3b(0,100,0);

// qDebug() << (\*i).x << ", " << (\*i).y;

}

displayMat(cloudImage);

}

vector<Building> MapReader::search()

{

vector<Building> moves;

//keep total distance in first element

Building dist;

dist.number = 0;

moves.push\_back(dist);

moves.push\_back(source);

moves = dfs(moves);

qDebug() << "distance: " << moves[0].number;

for(uchar i = 1; i < moves.size(); i++)

qDebug() << "step " << i << ": " << moves[i].description.c\_str();

return moves;

}

vector<Building> MapReader::dfs(const vector<Building> &moves)

{

if(moves[0].number > (campusImage.rows + campusImage.cols))

return vector<Building>();

Building curBuild = moves.back();

if(curBuild.north == goal.north && curBuild.south == goal.south

&& curBuild.east == goal.east && curBuild.west == goal.west

&& curBuild.near == goal.near )

{

return moves;

}

int idx = curBuild.number;

vector<Building> minMoves = moves, tmp;

minMoves[0].number = (campusImage.rows + campusImage.cols);

if( goal.north.count(idx) || goal.south.count(idx)

|| goal.east.count(idx) || goal.west.count(idx) || goal.near.count(idx))

{

tmp = moves;

tmp.push\_back(goal);

return tmp;

}

for(auto i = curBuild.north.begin(); i != curBuild.north.end(); i++)

{

Building nextBuild = buildings[\*i];

if(std::find(moves.begin(), moves.end(), nextBuild) != moves.end())

continue;

int dist = abs(nextBuild.centerOfMass.x - curBuild.centerOfMass.x)

+ abs(nextBuild.centerOfMass.y - curBuild.centerOfMass.y);

tmp = moves;

tmp[0].number += dist;

tmp.push\_back(nextBuild);

tmp = dfs(tmp);

if(!tmp.empty() && tmp[0].number < minMoves[0].number)

minMoves = tmp;

}

for(auto i = curBuild.south.begin(); i != curBuild.south.end(); i++)

{

Building nextBuild = buildings[\*i];

if(std::find(moves.begin(), moves.end(), nextBuild) != moves.end())

continue;

int dist = abs(nextBuild.centerOfMass.x - curBuild.centerOfMass.x)

+ abs(nextBuild.centerOfMass.y - curBuild.centerOfMass.y);

tmp = moves;

tmp[0].number += dist;

tmp.push\_back(nextBuild);

tmp = dfs(tmp);

if(!tmp.empty() && tmp[0].number < minMoves[0].number)

minMoves = tmp;

}

for(auto i = curBuild.east.begin(); i != curBuild.east.end(); i++)

{

Building nextBuild = buildings[\*i];

if(std::find(moves.begin(), moves.end(), nextBuild) != moves.end())

continue;

int dist = abs(nextBuild.centerOfMass.x - curBuild.centerOfMass.x)

+ abs(nextBuild.centerOfMass.y - curBuild.centerOfMass.y);

tmp = moves;

tmp[0].number += dist;

tmp.push\_back(nextBuild);

tmp = dfs(tmp);

if(!tmp.empty() && tmp[0].number < minMoves[0].number)

minMoves = tmp;

}

for(auto i = curBuild.west.begin(); i != curBuild.west.end(); i++)

{

Building nextBuild = buildings[\*i];

if(std::find(moves.begin(), moves.end(), nextBuild) != moves.end())

continue;

int dist = abs(nextBuild.centerOfMass.x - curBuild.centerOfMass.x)

+ abs(nextBuild.centerOfMass.y - curBuild.centerOfMass.y);

tmp = moves;

tmp[0].number += dist;

tmp.push\_back(nextBuild);

tmp = dfs(tmp);

if(!tmp.empty() && tmp[0].number < minMoves[0].number)

minMoves = tmp;

}

for(auto i = curBuild.near.begin(); i != curBuild.near.end(); i++)

{

Building nextBuild = buildings[\*i];

if(std::find(moves.begin(), moves.end(), nextBuild) != moves.end())

continue;

int dist = abs(nextBuild.centerOfMass.x - curBuild.centerOfMass.x)

+ abs(nextBuild.centerOfMass.y - curBuild.centerOfMass.y);

tmp = moves;

tmp[0].number += dist;

tmp.push\_back(nextBuild);

tmp = dfs(tmp);

if(!tmp.empty() && tmp[0].number < minMoves[0].number)

minMoves = tmp;

}

return minMoves;

}

void MapReader::on\_btnStart\_clicked()

{

currentPath = search();

displayMat(campusImage);

qDebug() << "Begin at:";

printFeatures(currentPath[1]);

takeinput = true;

curIdx = 1;

}

mapreader.ui

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>MapReader</class>

<widget class="QMainWindow" name="MapReader">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>443</width>

<height>571</height>

</rect>

</property>

<property name="windowTitle">

<string>MapReader</string>

</property>

<widget class="QWidget" name="centralWidget">

<widget class="ClickableLabel" name="lblImage" native="true">

<property name="geometry">

<rect>

<x>21</x>

<y>15</y>

<width>275</width>

<height>495</height>

</rect>

</property>

<property name="sizePolicy">

<sizepolicy hsizetype="Fixed" vsizetype="Fixed">

<horstretch>0</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

<property name="mouseTracking">

<bool>true</bool>

</property>

<property name="text" stdset="0">

<string/>

</property>

</widget>

<widget class="QPushButton" name="btnStart">

<property name="geometry">

<rect>

<x>320</x>

<y>20</y>

<width>114</width>

<height>32</height>

</rect>

</property>

<property name="text">

<string>Start</string>

</property>

</widget>

<widget class="QPushButton" name="btnReset">

<property name="geometry">

<rect>

<x>320</x>

<y>50</y>

<width>114</width>

<height>32</height>

</rect>

</property>

<property name="text">

<string>Reset</string>

</property>

</widget>

</widget>

<widget class="QMenuBar" name="menuBar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>443</width>

<height>22</height>

</rect>

</property>

</widget>

<widget class="QToolBar" name="mainToolBar">

<attribute name="toolBarArea">

<enum>TopToolBarArea</enum>

</attribute>

<attribute name="toolBarBreak">

<bool>false</bool>

</attribute>

</widget>

<widget class="QStatusBar" name="statusBar"/>

</widget>

<layoutdefault spacing="6" margin="11"/>

<customwidgets>

<customwidget>

<class>ClickableLabel</class>

<extends>QWidget</extends>

<header>clickablelabel.h</header>

</customwidget>

</customwidgets>

<resources/>

clickablelabel.h

#ifndef CLICKLABEL\_H

#define CLICKLABEL\_H

#include <QLabel>

#include "mapreader.h"

class ClickableLabel : public QLabel

{

Q\_OBJECT

public:

explicit ClickableLabel( const QString& text ="", QWidget \* parent=0) :

QLabel(parent)

{

this->setText(text);

}

explicit ClickableLabel( QWidget \* parent) :

QLabel(parent)

{

}

~*ClickableLabel*() {}

public slots:

void clicked( QMouseEvent \*event)

{

((MapReader\*)this->parentWidget())->clicked(event);

}

protected:

void *mousePressEvent* ( QMouseEvent \* event )

{

emit clicked(event);

}

};

#endif // CLICKLABEL\_H