COMS W4735y: Visual Interfaces to Computers
Spring, 2015
Assignment 3: Due April 9, 2015

Description of Visual Relations: Design Specifications

The goal of this assignment is to explore how well a visual image can be described in human language terms. You are to write a program that takes as input a two-dimensional image of the Columbia campus plus a "source" and a "target" location, and then produces as output non-numeric descriptions of these locations (the "what") and non-numeric directions to follow ("the where") from the source to the target. The assignment also asks you to evaluate the effectiveness of these natural language outputs.

More exactly, you are to write a visual user interface that displays a binary image of the main campus as seen from above, and which then responds to your (x,y) designation of a source with an English description of that location--which is not necessarily within a building, and which need not give the building's name. For example, it could print out: "This is south of and east of and near to a building (which is central and squarish)". Note that this is a description of a place southeast of Low, but "Low" doesn't ever have to be mentioned, since a user, especially a novice one (i.e., a visitor) probably doesn't know place names.

This first location also serves as the source (S) of a journey. A second (x,y) designation of a target (T) could also be a building, or a general place. The program would then respond by giving a series of directions from S to G. For example: "Go to the building that is east and near (which is cross-shaped). Then go to the building that is north (which is oriented east-to-west). Then go to the building that is north and east (which is medium-sized and oriented north-to-south)". The descriptions of the buildings (the ones in the parens) may or may not be part of the directions. Then, you give the map, a mouse, and the set of directions to the user, and ask the user to click on each of the places. You report the accuracy of their path-following.

So, your first main job is to write programs that use computer vision and a primitive form of inference: first, encode the buildings' shapes, from a map given online (the "what"); then, determine their spatial relationships to each other (the "where"); lastly, filter out any relationships that are unnecessary because they can be easily inferred. This filtering is not difficult, but critical to the next step. This allows you to convert any (x,y) absolute numbers into natural language relative descriptions.

Your second main job is to use these descriptions, which may be inexact, to choose whatever path from S to G that is as reliable as possible. You give this full set of directions to two friends, and then ask the friends to follow the directions one step at a time, by recording the mouse clicks that the friend makes as he or she follows the directions. Your program then reports how closely your friend was able to follow your description of the visual input.

In more detail, your friend gets eight different tasks, each with a different path, and each with a different path description. Each task is to follow a set of directions from S1 to G1, four without the information in parentheses so that it is just based on "where", and four with the information in parentheses so that it is also based on "what". You do this with two friends, so that you can test both conditions on all paths in a random way. For example, Friend A gets S1G1 through S4G4 no parens, S5G5 through S8G8 with parens; Friend B gets S5G5 through S8G8 no parens, S1G1 through S4G4 with parens. You then report

on the accuracy (did they get to the target at all?) and any other behavior that is notable.

The required images and other aids to the construction of the program are found on the class web pages, in standard portable formats (.pgm, .txt). You can create your own point-and-click interface or modify any existing one, but it is your obligation to make your approach understandable to the grading staff.

To help structure the assignment, it is broken down into four steps with equal credit, with the full assignment worth the 20 points toward your final grade. As a general rule, whatever you do in code or in writeup, style counts. It is your obligation to write it all up so that the instructor and/or the TA can understand it on the very first try.

1 Step 1 (5 points): Basic infrastructure and building features and descriptions: the "what".

This step does not require any user interface; it does the "front end" vision processing to get "what".

It finds a way of describing each building in terms of a vocabulary of shapes. These can include geometric figures, sizes, orientations, and other identifying adjectives such as extremums. For example, you might have "rectangular", "L-shaped", "narrow", "lumpy" as geometric descriptions; you might have "tiny", "small", average", "large" as size descriptions; you might have "oriented east-to-west", "oriented north-to-south", "symmetric" as orientation descriptions; you might have "most-central", "south-eastern-most" as extrema. For example, Low Library can be encoded as "squarish large symmetric most-central".

But anything you produce as a description must be automatically computed from the given data. For example, sizes must be computed based on a measurement, not from a hard-coded table. This means that, except for the extrema, more than one building can have same adjective: there should be a number of "L-shaped" buildings, a number of "small" ones, a number of "east-to-west", etc. You should do this by having a method or function for each shape description that evaluates a shape, and returns a boolean value for the description.

These methods can have magic numbers within them (for example, for "small", you can compare areas or diagonal lengths against a constant), as long as the magic number is justified with a comment: how or why did you set it at the value you did. But it is not fair to hard code your descriptions in a manually-generated table or code. That is, you can't say "return (buildingNumber == 5)" or its equivalents; you have to compute the answer based on visual properties.

You can consider the following descriptions to start with, but you can use a subset, or change them, or add new descriptions:

1. small(est) / medium / large(st)
2. long(est) / thin(est)
3. square(st) / rectangular / (most)nonRectangular
4. simpleBoundary / jaggedBoundary / hasBumps / hasDents
5. singleBuilding / multipleBuilding
6. I-shaped / L-shaped / C-shaped / partlyCurved
7. cornersSharp / cornersChewedOff
8. symmetricEastWest / symmetricNorthSouth / irregularlyShaped

9. orientedEastWest / orientedNorthSouth
10. centrallyLocated / onBorder
11. northernmost / southernmost / eastermost / westernmost

The following data files were created for the assignment: "ass3-campus.pgm", "ass3-labeled.pgm", "ass3-table.txt". The first file is a binary image of the main campus as seen from above, where a large number represents the buildings and zeros represent the space between them. The second file is an integer-valued image based on the first, in which each building is given an encoded integer, and all the pixels belonging to the same building are encoded with the same integer; zero still means empty space. The third file is a text file which translates the encoded integer into a string, so that some of your assignment's answers (but not the visual interface itself) can come out in English. There are 27 buildings in all, since some of them (like the Chemistry buildings, or the southeastern dorms) have been combined.

The file "ass3-labeled.pgm" has been created explicitly to make these tasks easy; to get the shape features for Building N, one only has to scan the image for the occurrences of the encoded integer N. The file "ass3-table.txt" makes the translation of the encoded integer to a string easy, too.

For this step, have your system print out, for each building, the usual geometric features (such as you used in Assignment 1) and also these additional new descriptions. You should then, for each of the 27 shapes, print out: the building name, the (x,y) of the center of mass, the area, and the upper left and lower right coordinates of its minimum bounding rectangle (MBR), and then the description of its shape. You must clearly indicate what vocabulary of descriptions you used, and how you do the computer vision to natural language translation in a general way.

2 Step 2 (5 points): Describing compact spatial relations: the "where".

Now, design and code the boolean-valued functions for building pairs: North(S, T), South(S, T), East(S, T), West(S, T), and Near(S, T). These are read, "North of S is T" and "Near to S is T". Each function takes the descriptions of the source building S and the target building T and returns true if they are in the given relationship; i.e., North(S, T) means that from source to target you go north. Note that there may be some necessary approximations: some buildings like Low do not completely fill their MBR, and Computer Science actually has a hole in it. Note also that the four directional relationships will use nearly the same code, but you should decide how best to define those four without reference to any specific building, just with reference to their generic shape features. It is probably more humanly accurate to have North be more complicated than simply comparing y coordinates alone. If you wish, North can be affected by size and shape, for example. Lastly, note that Near *must* be affected by the size of the building: it should be harder to be near to Alma Mater than to be near to Low.

Now, create *all* the binary spatial "where" relationships that apply to *every* building pair, according to your definition of North (and its three compass equivalents) and Near. This part should generate on the order of O(B*B*R) binary relationships, where B = 27 buildings, and R = 5 relationships, as every building pair either has or does not have a given relationship. You can print these out if you wish to check them, but there are far too many to be useful. Note that the relationships need not be symmetric: Near(S, T) may not automatically imply Near(T, S), depending on how you define the relationships.

Then, filter these "where" relationships leaving only the ones that cannot be inferred by the usual transitivity rules. This can be a bit tricky, as it is the opposite of transitive closure (technically, it is called

transitive *reduction*). For example, CEPSR satisfies both North(Low, CEPSR) and North(Butler, CEPSR), but you can drop the second one. This is because we know from the map North(Butler, Low), so we can infer North (Butler, CEPSR) anyway. There are fast ways to filter the compass directions via transitivity, although some positions can legitimately be East (or West, etc.) of more than one building even after filtering. For example, North(St. Paul's, Avery) and North(St. Paul's, Fayerweather), but neither Avery nor Fayerweather are North of each other.

But Near is tricky, as it interacts with compass directions in a more complex way, and it depends upon your definition. For example, it is likely that Near (Uris, Computer Center). If it is the case that according to your definitions that anything that is Near Uris would be always North of Low, then you might not have to keep North(Low, Computer Center). This is true even though there is no other building between Computer Center and Low that would allow you to drop North(Low, Computer Center) on the basis of using that building in the transitivity of North. Note also that there may be a number of buildings or places which aren't Near anything, and a number of building or places that no other buildings or places are Near; these are not the same, as Near may not be symmetric.

For this step, show your code, and for each building, whichever relationships have survived these filtering steps (even if you proceed to further Steps). For each building, this should be much fewer than the 27*5 possible relationships each building can enter into, although the exact number will vary by building. For this output, you should use building names rather than numbers.

3 Step 3 (5 points): Source and Target Description and User Interface

You now create the infrastructure you need to solve the problem in general, but without testing it yet.

Your system provides an S and a T by giving an (x,y) for both. You print out the description of S that is possible, whether or not it is in a building. Essentially, S is a tiny new virtual one-pixel building, so it has no "what". You must dynamically use the code you wrote above to describe its "where", its position relative to existing buildings. If this pixel is one that returns a positive encoded integer rather than a zero, you can also use "in". You should describe this virutal building just like you did in Step 2. That is, you can say something like "near Kent and south of St. Paul's" or, "in Kent and south of St. Paul's".

(Note that theoretically, every possible (x,y) pixel description can be precomputed and stored, but this would be very costly: it would a problem with complexity O(W*H*B*R), where W is the width and H is the height of the image, and here W=275 and H=495. We won't do this.)

However, note that whatever description you give for the S will be shared by other pixels as well. You need now to display the uncertainty that this click causes, as a sort of pixel cloud. For example, if you click on a point near Carman, and then describe it as "near to Carman, south of Lerner, west of Butler", there will be many other pixels near S that are described in exactly the same way. For each (x,y), then, you must not only give the description, but display on the image *all* the pixels in this equivalence class. (For example, you can color them all green.) The second (x,y) gives the target T. You must do the same: describe its location and show its equivalence class of pixels that are described the same way. (For example, you can color them all red.) Please note that there is a way to do this without having to generate full descriptions for every pixel.

To make this more practical, and to set the system up for the evaluation in the last Step, you should make the (x,y) interactive. That is, display the map, and allow yourself to click on it to determine the (x,y)

by mouse input. It should work like this: your first click is taken to be the (x,y) of S, and your system shows a green cloud and prints out a text description of S. Then your second click is taken to be the (x,y) of T, and your system shows a red cloud and prints out a text description of T.

For this step, show your code, and give three well-chosen S and T pairs, their clouds, and their descriptions as places using the "where" description with building names. Also: report the (x,y) that has the *smallest* cloud, and the one with the *biggest* cloud: you can do this either by experiment through clicking, or algorithmically through a brute-force search through all possible (x,y).

4 Step 4 (5 points): Creativity: Path Generation

Finally, you must describe a path from S to T, and save it, in order to give to a friend. The most general real-world problem would be to describe a number of places along the way that are outside of buildings, and give directions to these places based purely on their "where" description. But describing places outside of buildings is too hard, and users might not know building names. Instead, what you want to do is first give a direction from S to a Building 1 center, then a direction from Building 1 center to a Building 2 center, and so on until you can finally give a direction from Building N center to T. (This last direction is sometimes called "terminal guidance".) This will allow you to use both "where" and "what", without building names.

Clearly you want some sort of shortest path algorithm on some graph, from S, through intermediate buildings, to T. For this, you need to use a graph that links Building I to Building J only if Building J can be described in relation to Building I. For example, it is very unlikely, no matter what you definitions that use, that you will be able to describe Butler using CEPSR or the reverse: there are simply too many buildings between them to expect that North(Butler, CEPSR) has survived pruning. Or, viewed another way, if you are at Butler, saying "Go north" won't get you to CEPSR at all; you will end up at Alma Mater (or possibly some other buildings that are equally possible as well). In general, this J-can-be-described-from-I graph is sparse, and binary. For his assignment, you can ignore physical distances; solve the problem using just binary relationships.

The best way to create this sequence of directions is to do a search. At each step of the search, you pick a "good" sequence of directions, and extend it by one more direction, and then see if you are at T. Whether you use breadth-first, depth-first, best-first, A*, greedy, etc., is up to you, but the problem is simple enough to pick a simple method. You can consider the goodness of the final sequence of directions to be measured by how *likely* you are to end up at G once you follow it; try to pick unambiguous directions.

Note that because directions are approximate, what you get from any search may not be very useful. For example, to get from S = St. Paul's to T = Avery, you may find that the only direction is "Go north and near (to the vertically-oriented medium-sized building)" But that may also describe Fayerweather. So your likelihood of success is only .5. But, surprisingly, if you were to go from S = St. Paul's to GT= Schermerhorn, which is a longer trip, and you said "Go to the building that is north and near, then go to the building that is north and near", you would probably end up in exactly in the right place, whether you went via Avery correctly or via Fayerweather incorrectly. That is, sometimes the semantics are self-correcting.

So, for this step, you need to take your directions generated for eight paths, and find two friends to

follow them. Use the interface you developed in Step 3. Randomize them as talked about in the introduction to this assignment. Your friend is given the directions, then clicks on each location that they describe. At each click, your interface shows the clicked location and records the click. When your friend finishes, you measure how close the final location is to the intended T. Since sometimes your friend may find themselves in a position where the next direction makes no sense, your friend indicates this with an "I'm lost" click. For a given friend, and for each of the eight paths, record the path the friend took, and its closeness to T when it was completed (or abandoned).

For this step, show your code, the 16 experiments (two friends with eight paths each), and discuss what your user study means. That is, you will have to describe how well your system for taking an image in and producing natural language out worked, and why.

5 Checklist of deliverables

1. Your assignment consists of a writeup with examples, then a listing of all the code used. Any code that you did not write yourself has to be documented with a statement about its source and an explanation of why you have permission to use it.
2. For all steps, your writeup explains: what design choices you made and why, what algorithms you used, what you observed in the output, and how well you believe your design worked.
3. For step 1, you must show your computer vision processing of buildings into quantitative location and shape numbers. You must also show your choice of descriptive categories, and the algorithms that translate the numbers into descriptions.
4. For step 2, you must describe each of the 27 buildings with their pruned spatial relationships.
5. For step 3, you must show the S and T "clouds" for three interesting paths, and the set of directions, including the parentheses, that describe them. Show also the most confused place (biggest cloud) and least confused place (smallest cloud) that your system generates.
6. For step 4, you must record 16 experiments, display the accuracy of their results, and comment on how the different kinds of descriptions were or were not helpful.