A simple approach to the route following problem based on the identification and localization of landmarks

Erik Odenman
Supervisor: Gabriel Skantze

April 12, 2013

Abstract

The route following problem involves writing a program that on a map can follow a route described by a person in natural language. A simple approach to the problem, where words describing landmarks are identified and localized, is implemented and tested. The approach does not recognize spatial words like cardinal directions, so the tests give a hint of how important these words are when describing and following routes. It is concluded that landmarks are successfully identified, but important details of the routes are lost.

Referat

Ett enkelt tillvägagångssätt för att följa vägbeskrivningar baserat på identifiering och lokalisering av landmärken

Problemet som ges är att skapa ett program för att följa vägbeskrivningar givna i naturligt talspråk. Ett enkelt tillvägagångssätt, som bygger på att ord som beskriver landmärken identifieras och lokaliseras, implementeras och testas. Metoden känner inte igen spatiala ord såsom väderstreck, så testen ger en antydan till hur viktiga dessa ord är för vägbeskrivningar. Det konstateras att identifiering av landmärken lyckas, men att viktiga detaljer av vägarna som beskrivs går förlorade.

Contents

1 Introduction

1.1 Background

The problem of making a computer follow a route description given in human language is one that has been studied in multiple papers[1][2].[4] While the problem can be approached using linguistic notation it becomes difficult to cover all the spatial words that one could use in the description of a path, not taking into account the translation of words into multiple languages. A more convenient solution is therefore highly appreciated.

A spatial semantic was developed by Levit and Roy,[2] who used it to create a program that could follow complex routes through a map. However, they left the translation of natural language into this semantic as an open problem. In their representation an instruction consists of multiple components describing for instance reference objects, path descriptors as well as quentitative aspects. Vogel and Jurafsky[1] simplified it a bit, as their representation only involved a reference object and a frame of reference. They did however approach the problem with parsing instructions in natural language, and created a program that through a reinforced learning technique learned to navigate between landmarks and pass them on the right sides.

For this project the frame of reference will be ignored as well, leaving only the reference object as representation for an instruction.

1.2 Purpose

The purpose of this paper is to create and analyze a simpler approach to the route following problem, one that does not involve categorizing words into different spatial categories but instead only revolves around identifying and locating landmarks. By analyzing how well this approach works one can get an idea of how important words like the ones for cardinal directions are for the ability of describing and following a route. It should also be mentioned that while the projects named earlier incorporated prior knowledge of the map (in form of names and locations of landmarks) into their algorithms, this will not be the case in this paper.

1.3 Definitions

1.3.1 Tf-idf

Tf-idf stands for *term frequency - inverse document frequency* and is a commonly used technique for measuring a document's relevance to a search word[5][6].[7] It is the product of two functions, *tf* and *idf*, where *tf* is a measure of how frequent the word is in the document and *idf* is a measure of how unique the word is to the document.

1.3.2 Instruction giver

The entity describing a route on a map for the instruction follower to try and follow. In the sample data used for this project the instruction giver is a person.

1.3.3 Instruction follower

The entity whose task is to follow the route described by the instruction giver as closely as possible. The purpose of this paper is to create and analyze an instruction follower.

1.3.4 Reference path

The path that the instruction giver follows while describing the route to the instruction follower. See section 2.1

1.3.5 Landmark

A landmark is an (immobile) object on a map, which the instruction giver can refer to when describing a route. For example, 'the tunnel' and 'the bus stop' are landmarks while 'north' and 'go' are not. There is nothing that prevents multiple landmarks to have the same name, and different instruction givers might refer to the same landmark using different words.

2 Problem statement

2.1 Sample data

The data used for processing as well as testing has been provided from a different project.[3] It contains test sessions involving 10 people and 5 different maps. For each map, the test person was asked to describe the depicted route and at the same time move the mouse cursor along it. The descriptions were then recorded, transcripted and divided into sentences, and the movement of the mousecursor was saved as a sequence of coordinates. No restrictions were made on the language as the persons were allowed to speek freely.

2.2 Assumptions

The following assumptions were made to make the problem easier to approach:

- *Every sentence uttered by the instruction giver only contains one instruction.* This is true for the most part but the instruction giver might also combine several instructions into one sentence. One example of this is sentences like 'go north until you pass the crossing and then walk toward the house on your left'. These do not occur too often though, and should not make too much of an impact when they do.
- *Every landmark can be represented by a single word.* Although some landmarks are usually described by more than one word, like in 'bus stop', one word often suffices as a distinguishable representation.

3 Method

3.1 Data processing phase

The purpose of the data processing phase is to transform the sample data to a format suitable for the purpose of following new route descriptions given in text. As this will require knowledge of the positions on the map where a particular word has been used, such a mapping will be created.

For each instruction given by the instruction givers the point where the instruction ended can be seen as the goal point. This point will therefore be mapped to the words used in the instruction, and after repeating this procedure for multiple data sets every used word will map to a set of points on the map where the word was uttered (see Figure 1. In other words, a many to many relationship between words and points is created. From now on, when referring to a word's set of points on a map, this is what is referred to.

3.2 Finding words relevant to landmarks

For each instruction it is crucial to find the word that represents the instruction the best. This involves eliminating words that haven't been found in the data collection phase as well as common words like 'go' and 'and', that don't contribute any useful information about the next landmark that is about to be visit. For this task a version of the tf-idf technique is used to create a function that takes a word and a position on the map and returns a number describing the relevance of the word. This number is later used to compare words and decide which is the most useful for continuing the path. The function will be the product of two functions, the tf-function and the idf-function, as follows.

3.2.1 Tf-function

The tf-function should take as parameters the map that is currently being searched, the position we are currently at as well as the word to test. The purpose of the function is to eliminate words that are uncommon for instructions in the particular area or haven't been come across in the data collection phase at all. It is also important that very common words are not too favoured. The definition that was used in this project is the following (where m is the current map, p the current position and w is the word to test):

$$tf(m,p,w) = \begin{cases} 0, & \text{if } f(m,p,w) = 0 \\ 1 + log(f(m,p,w)), & \text{otherwise} \end{cases}$$

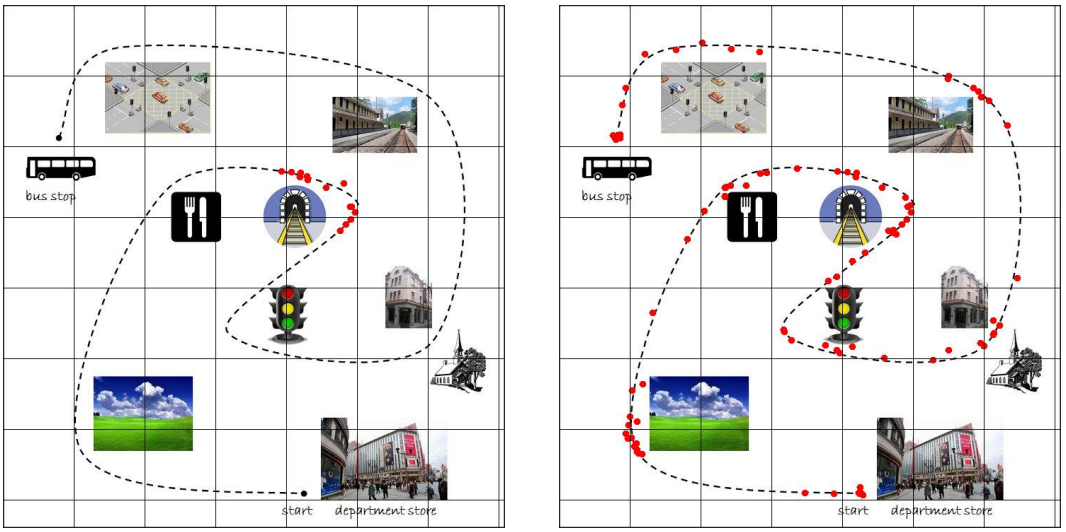
Here $f(m,p,w)$ is the raw frequency of the word w within some surrounding area of the position p. Different sizes of this area were tested and bigger sizes tended to yield better results, as the risk of the algorithms getting lost along the way decreased.

3.2.2 Idf-function

As the tf-function favours words that occur a lot in an area, the purpose of the idf-function is to eliminate words that are common all over the map. To achieve this the map is divided into a grid of square cells, as depicted below. A word common all over the map will occur in a larger portion of the cells than a word only used in a particular area of the map. If we define C as the set of all cells and C_w the set of cells containing the word w, the idf-function can be defined as follows:

$$idf(m, w) = log(\frac{|C|}{|C_w|})$$

It should be noted that the base of the logarithm is not relevant, as changing it only contributes a constant factor to the answer.[5] For both the tf and idf functions e was used as the base for the logarithm.



(a) Occurences of the word 'tunnel' (b) Occurences of the word 'you'

Figure 1: Displaying the difference between a word describing a landmark (left picture), and a very common word (right picture). The points in the left picture occur in a smaller subset of the square cells than those in the right picture, and consequently produce a higher idf-value

3.3 Connecting the points

When the most relevant word from each instruction are selected the next step in the algorithm is responsible for choosing a point mapped to each word and connect these to a path. One important definition that has to be made involves how to measure the goodness of a particular path. Without this measure there won't be a way to compare different paths and choose the best one.

One can, for instance, imagine that a path crossing itself and returning to the same point again is a bad one, and that a penalty for this would be fitting. On the other hand there are often repetitions in the sample data, as the instruction giver sometimes makes mistakes and the instruction follower sometimes asks for clarifications. Another property that a good path usually has is smoothness, so a measure that favours paths with as few sharp turns as possible would be an option as well. The question of whether a path is good or bad turns out to be hard to answer, without looking at the reference path, and no answer that makes the question justice will be given in this paper. For the sake of simplicity the measure that was chosen is the one that minimizes the total length of the path, while still visiting all the points in the right order. This can be somewhat justified by the assumption that when a landmark is used in an instruction, it is often pretty close to the current location (jumps all across the map in one instruction are not very usual).

To produce the path that visites all landmarks in the right order, and does so with the minimum total length, two algorithms were implemented and compared. Both take a map, a list of instructions and a starting point, and they both produce a list of points describing a path.

3.3.1 Greedy algorithm

The greedy approach involves always choosing the next point so as to optimize the solution one step ahead from the current point. In this way a path is created incrementally and when a point is chosen it is guaranteed to be in the final result. This means that a local optimum is always found in each step but there is no guarantee that the length of the final path will be optimal.

```
Pseudo code:
greedy_path(map, instructions, start) {
    result <- []
    append start to result
    position <- start
    for instruction in instructions
        w <- the word in instruction with the highest relevance
        position <- choose a point in map where w has been used
                    and where the distance to position is minimal
        append position to result
    return result
}
```

3.3.2 Dynamic Programming algorithm

Unlike the greedy algorithm, the dynamic programming algorithm does not make any decisions about which points to use until the very last step. It is very possible that a better solution can be achieved by choosing a point that at the time seems suboptimal. To avoid missing these solutions an optimal path is saved for each possible end point in

each step of the algorithm. In the last step of the algorithm the best end point is chosen and the optimal path ending there is recovered.

```
Pseudo code:
dp_path(map, instructions, start) {
    let best_path[p] be the best path ending in point p
    set best_path[start] to the empty set and mark all other
            points as not reachable
    for instruction in instructions
        initialize new_best_path and mark every point as not reachable
        for each reachable point p in best_path
            w <- the word in the instruction with highest relevance to p
            for each point wp where w has been used
                update new_best_path[wp] if a better path can be
                        created by moving from p to wp
        best_path <- new_best_path
    best <- the best end point in best_path
    recover and return best_path[best]
}
```

3.4 Testing the algorithms

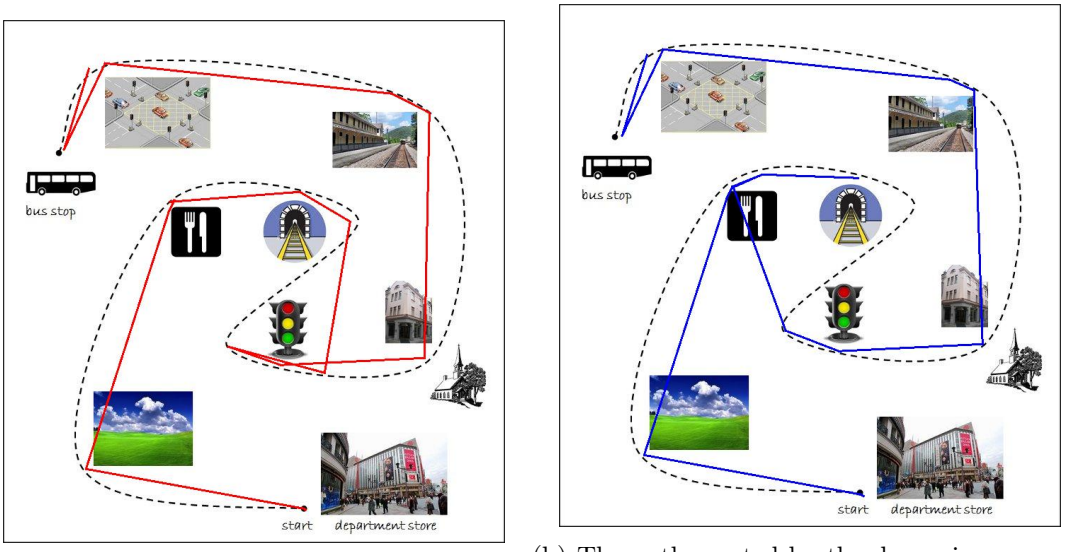
Testing was done on a subset of the provided data, but no session was used for both data processing and testing. A big part of the evaluation consisted of manual inspection of the routes produced, which proved to be the best way to get an understanding of why the algorithms behaved the way the did. To get a more precise measure to compare the results with the reference path was used. By comparing the paths generated by the algorithms above to the reference path, a measure of its error can be computed as the summary of the difference in each point. To get a value more useful in comparing results between different maps this measure was also divided by the total length of the reference path and expressed as a percentage. Results on a subset of the tests are described in Table 1.

Table 1: Measuring the error (in pixels) for the two path building algorithms

Algorithm	Test 1		Test 2		Test 3		Test 4	
	Error	%	Error	%	Error	%	Error	%
Greedy	5088	257	**4185**	**169**	**1667**	**66**	**3266**	**124**
Dynamic Programming	**3576**	**180**	5482	221	1717	68	3955	150

4 Results

Any clear distinction between the greedy and the dynamic programming algorithm is hard to find. They both perform poorly on a majority of the test cases while both find



(a) The path created by the greedy algorithm

(b) The path created by the dynamic programming algorithm

Figure 2: An illustration of Test 3

decent paths in some cases. One example is Test 3, as seen in Figure 2. In this test case the route description was simplified to the following list of representative words: *[yes, department, west, field, restaurant, east, restaurant, tunnel, eh, yeah, traffic, east, church, hotel, station, west, junction, bus, junction, bus]*. One can see that all the landmarks are identified and visited, but as some instructions lack information about any landmark some less informative words are used as well. As of now this is not detected by the algorithms.

Small details in the routes are never found, as is expected by an algorithm that does not recognize spatial words, but for the most part the landmarks are visited in the right order.

5 Conclusions

5.1 Advantages and disadvantages

An algorithm that does not consider the meaning of spatial words will often miss small details in the routes described and will have trouble passing landmarks on the right side. On the other hand, the results show that it is possible to create a rough estimation of the route without any knowledge of the meaning of words or prior knowledge of landmarks. Although this kind of approach will not be able to produce results of the same quality as other more advanced methods, one big advantage of it is the language independence. Being able to follow routes in any language (as long as sample data for that language has been provided) is a huge convenience.

5.2 Further improvements

There are a lot of improvements that one can imagine if it is to be studied further. The most obvious one is a better way of evaluating the goodness of a path without knowledge of the reference path. As mentioned earlier possible approaches to this could involve maximizing smoothness of the path, by taking into account angles between consecutive points.

Another part of the algorithm that needs refinement is the way that the possible locations of landmarks is decided. A single occurence of a word in the wrong place should not make a big impact on the outcome, so it is crucial to find a way to filter those points out. This becomes especially important would the amount of data processed increase. Although I have not investigated possible methods to accomplish this, I am convinced many such methods exist.

5.3 Predictions

Despite the advantage of language independence that this approach to the route following problem offers, it seems unlikely that an algorithm ignoring words as meaningful as cardinal directions will be used in any bigger application. As convenient as it may be to base a route on only a small subset of the words real world applications will probably need a more sofisticated approach.

References

[1] Learning to Follow Navigational Directions. Adam Vogel and Dan Jurafsky, Department of Computer Science, Stanford University. http://nlp.stanford.edu/pubs/spatial-acl2010.pdf

[2] Interpretation of Spatial Language in a Map Navigation Task. Levit M, Roy D, BBN Technol. Corp., Cambridge, MA

[3] A Testbed for Examining the Timing of Feedback using a Map Task. Gabriel Skantze, Department of Speech Music and Hearing, KTH, Stockholm, Sweden. http://www.speech.kth.se/prod/publications/files/3761.pdf

[4] Spatial Cognition: The Role of Landmark, Route, and Survey Knowledge in Human and Robot Navigation. Steffen Werner, Bernd Krieg-Bruckner, Hanspeter A. Mallot, Karin Schweizer Christian Freksa. http://act-r.psy.cmu.edu/~douglass/Douglass/Agents/TopicPapers/QSR/Freksa/werner97spatial.pdf

[5] Tf–idf. Wikipedia - The Free Encyclopedia. http://en.wikipedia.org/wiki/Tf-idf

[6] Understanding Inverse Document Frequency: On theoretical arguments for IDF. Stephen Robertson http://www.soi.city.ac.uk/~ser/idfpapers/Robertson_idf_JDoc.pdf

[7] Machine Learning :: Text feature extraction (tf-idf). Christian S. Perone. http://pyevolve.sourceforge.net/wordpress/?p=1589