University of Umeå
Department of Computing Science
SE-901 87 Umeå, Sweden

UMINF-12.18
October 2012

Toward an Active Database Platform for Guiding Urban Pedestrians [1]

by

Michael Minock, Johan Mollevik and Mattias Åsander

[1]The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 270019 (SpaceBook project www.spacebook-project.eu) as well as a grant through the Kempe foundation (www.kempe.com).

ABSTRACT

We present an Android-based platform for incrementally presenting spoken route directions to guide pedestrians to destinations. Our approach makes heavy use of stored procedures and triggers in an underlying PostGIS spatial database. In fact most of the 'intelligence' of our prototype resides in database stored procedures and tables. As such it represents an example of a challenging real world case study for the use of persistent stored modules (PSM) in a complex mobility application. It also provides a platform to study performance tradeoffs for complex event processing over spatial data streams.

1 Introduction

The automated generation of route directions has been the subject of many recent academic studies [2, 11, 9, 8, 13, 3, 12, 7] and commercial projects (e.g. products by Garmin, TomTom, Google, Apple, etc.). While most focus has been dedicated to automobile drivers, there has also been an effort to provide route directions to pedestrians (e.g. Google and SIRI). The pedestrian case is particularly challenging because the location of the pedestrian is not just restricted to the road network and the pedestrian is able to quickly face different directions. In addition the scale of the pedestrian's world is much finer, thus requiring more detailed data representation. Finally the task is complicated by the fact that the pedestrian, for safety, should endeavor to keep their eyes and hands free – there is no room for a fixed dashboard screen to assist in presenting route directions. We take this last constraint at full force – in our prototype there is no map display; the only mode of presentation is text-to-speech instruction heard incrementally through the pedestrian's earpiece.

We focus here on the problem of providing incremental spoken route directions to guide a pedestrian from their current position to a given destination. Such a problem yields two related metrics of evaluation: *(1) what is the system's effectiveness in actually guiding pedestrians from a given initial position to a given destination position?*; *(2) how many simultaneous users can a system scale to?*. These two metrics most certainly trade off against one another. While our initial focus has mostly been on improving metric 1 measures, metric 2 is increasingly a consideration.

1.1 Organization of this report

This report describes a test-bed prototype that we implemented to explore the pure navigation case. Section 2 of this report introduces the terminology and concepts we use in our work. The terminology is based largely on that of Richer and Klippel [11], although we limit ourselves to only a subset of their terms and adapt the terminology slightly. Section 3 presents the overall architecture of the prototype. Because of the centrality of the SpatialDB in our prototype, section 4 describes the table definitions and dynamic state within the spatial database. These tables correspond to the concepts presented in section 2. Section 4 shows how we implement a 'policy' that maps from complex spatial/temporal state to commands to generate route directions. Section 5 informally reports on how our system initially performs for several tests carried out in Umeå in October 2012, Sweden. Section 6 concludes.

2 Terminology

The *path network* upon which a pedestrian may be directed to travel is made up of *branching points* and *path segments*, as illustrated in figure 1. For example the points labeled '5401' and '5522' are branching points and the line from point 5401 to point 5522 represents a path segment. Path segments are directed and are often not straight line edges, but rather

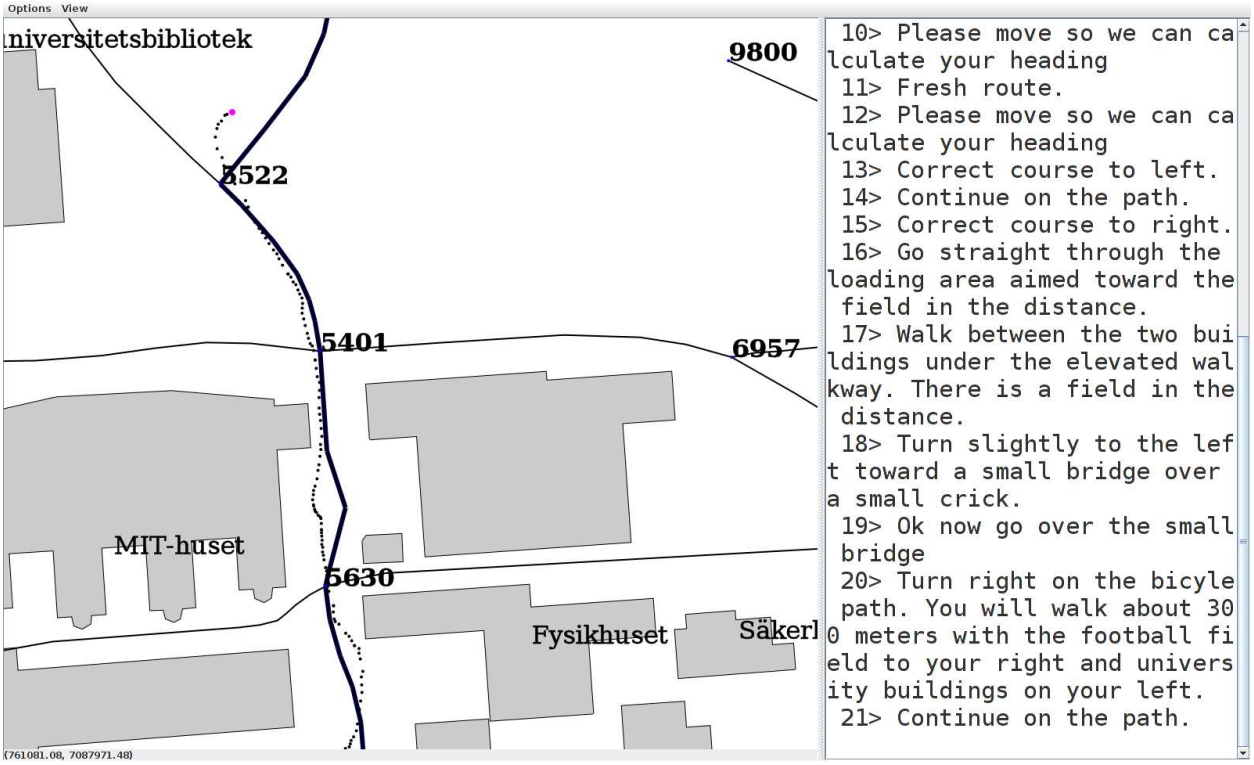

Figure 1: Guiding a pedestrian on a route on Umeå University's campus

are defined as a sequence of directed *elementary segments* that, chained together, represent curves or the meandering of a path segment. The path segment from branching point 5401 to branching point 5522 consists of 3 elementary segments. A *path* is a connected sequence of path segments that would take a pedestrian from some origin branching point to a destination branching point. In figure 1 we see a marked path with an origin and a destination that is off the map. A *route* demarcates a path, consisting of *route segments* and *decision points* which in turn demarcate associated path segments and branching points of the path.

Landmarks are entities in space that have associated point, linear or polygonal geometries. Landmarks also have associated *types* (restaurant, bar, museum, street, park, university_building, etc.) and names (e.g. 'MIT-Huset', 'Fysikhuset', etc.). There is a general linking relation that allows arbitrarily named *properties* and *values* to be associated with landmarks (color, architectural style, etc.).

3 System Overview

In figure 2 we see the main components of our architecture. The key components are the PhoneApp, the SpatialDB, the RoutePlanner and finally the Controller.

The PhoneApp runs on the user's Android phone and logs GPS measurements as longitude, latitude, antenna error triples every second to the SpatialDB. The PhoneApp likewise

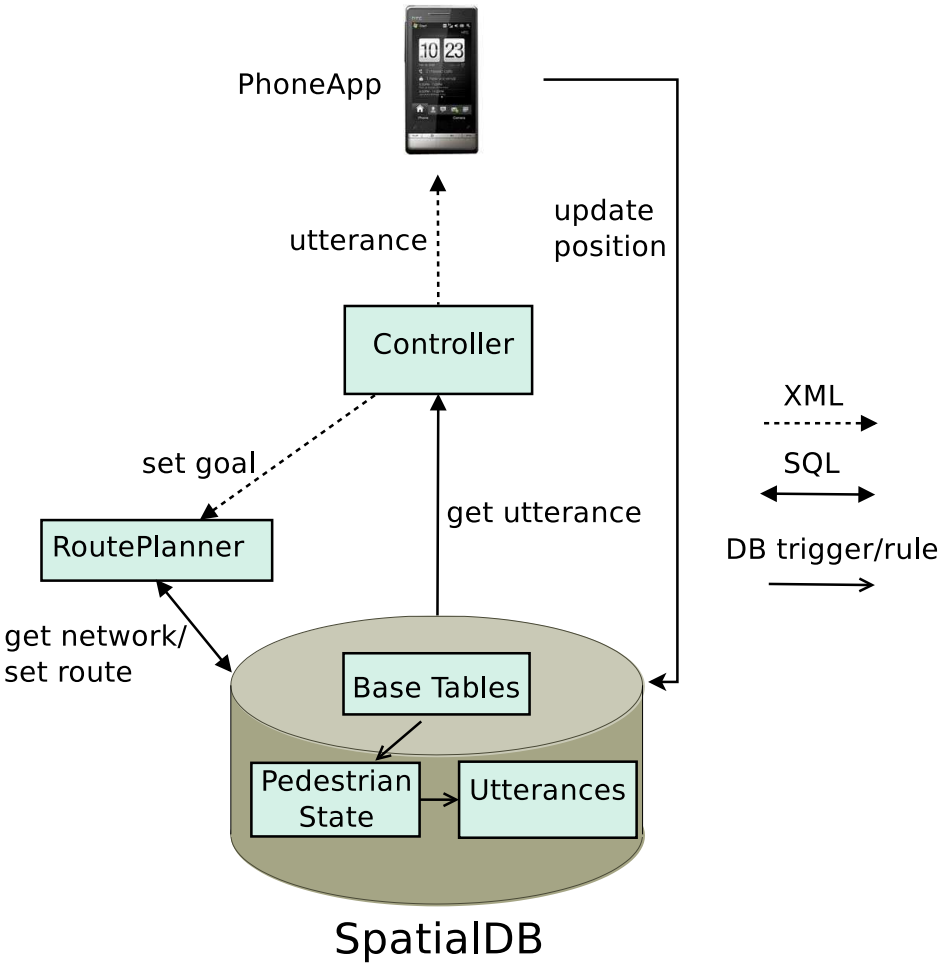

Figure 2: The basic architecture

accepts text messages from the CONTROLLER that are voiced using Google's text-to-speech engine running on Android.

The SpatialDB is the single repository for all *state* in the system. This means that the SpatialDB represents the path network, the landmarks, the GPS measures, routes and a log of previously issued utterances. In addition there is a state table that is dynamically derived via *stored procedures*. These state tuples capture the complex spatial and communication state of the pedestrian through time. Finally the SpatialDB contains a set of communication rules that select the utterance, if any, that should be voiced to the pedestrian. The SpatialDB is implemented within PostGIS/PostgreSQL, using both PostgreSQL triggers and rules with stored procedures implemented in PL/PGSQL.

The ROUTEPLANNER is a simple component that plans a route from the branching point closest to the pedestrian's current position to a destination branching point within the path network. The method used is simply A* search using a *straight line distance* heuristic run over the path network with cost based on path segment length[10]. To set up the search, the

RoutePlanner issues an SQL query to the SpatialDB to bring in the relevant part of the path network. After performing the search, the RoutePlanner inserts the result into the `Route` table in the SpatialDB.

The Controller runs a very simple (less than 20 lines of Java code) control loop that polls the database for what utterance message to send next to the PhoneApp and when it is necessary to invoke the RoutePlanner for a new goal.

There are two infrastructure components that do not appear in figure 2, but should be mentioned for the sake of completeness: the PhoneServer and the IceBroker. The PhoneServer represents the PhoneApp in the back end and shunts GPS position reports to the SpatialDB as well as shunting text message issued by the Controller onward to the PhoneApp for voicing. The IceBroker allows components to publish and subscribe to data streams (e.g. GPS data) or to issue remote procedure calls on other components (e.g. executing SQL queries, etc). It represents a slightly higher level of abstraction and functionality than a pure socket-based client server protocol would support [4].

4 The SpatialDB

Because of the centrality that the SpatialDB plays in our prototype, we describe in some detail the tables in the SpatialDB, how they are initially populated or dynamically generated.

4.1 The base tables

Figure 3 shows the base tables of the database from figure 2 grouped into tables representing the path network, landmarks, routes and the pedestrian name and time series of reported GPS positions[1]. These tables mirror exactly the terminology of section 2. Attributes named `point` or `line` and `geom` are PostGIS geometry types.

The base tables are populated by external processes that add data either at database build time or at run-time. Specifically the landmark and path network tables of figure 3 are populated at database build time by converting OpenStreetMaps XML data[1][2] to tuples in our schema. The pedestrian tables are populated by a very simple pedestrian registration process as well as the run-time GPS logging at one update per second from PhoneApp. The route tables are populated at run-time by the RoutePlanner once a call for a new goal is made by the Controller.

4.2 Pedestrian state table

Pedestrian 'state' is represented as a tuple in a single table with many attributes (`PedestrianState`). The attributes have varied types (boolean, integer, real, PostGIS geometry types, time stamps, etc.) and are described below:

[1]Note that the table and attribute names used here are slightly different in the actual implementation. We document these differences in the README file accompanying our (future) open-source software distribution.
[2]Obtained at `http://openstreetmaps.org`.

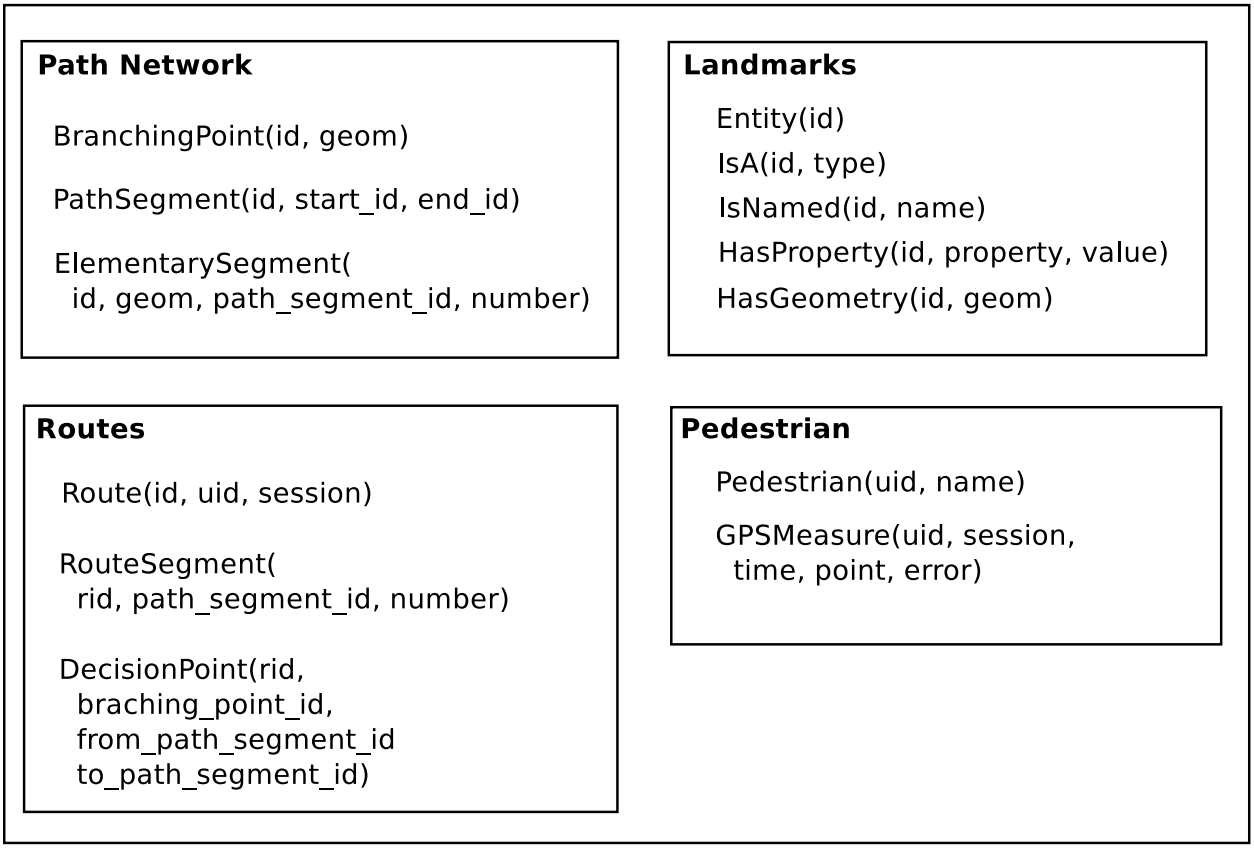

Figure 3: The base tables

`uid`: This is the pedestrian's id.

`session`: This is the session of the pedestrian.

`phone_time`: This is the time-stamp recorded on the phone by PhoneApp.

`insert_time`: This is the actual time at which the state tuple is inserted into the database. The difference between `phone_time` and `insert_time` represents the *position report latency* plus the *clock difference* between the PhoneApp and the SpatialDB.

`position`: This is the smoothed and filtered position that is the best guess as to the actual position of the pedestrian. Currently this smoothing and filtering process is very simplistic.

`GPS_error`: This is the GPS error measure reported by the PhoneApp.

`speed`: This is a smoothed value that estimates the pedestrian's speed in meters per second.

`heading`: This is a smoothed angle value (in degrees, with North at 0°, East at 90°, etc) that represents the direction in which the user is facing under the assumption that pedestrians always face in the direction that they are traveling. Quite often this field has the `NULL` value to represent that we do not have a reliable heading value for example

at the start of a session, when the user is at a stand still, when we are not getting consistent GPS measures, etc.

`on_path`: This is a boolean value that is true if the pedestrian's position is within a distance threshold (currently set at 10 meters) to an elementary segment of a path segment of the current route.

`at_branching_point`: This is a boolean value that is true if the pedestrian's position is within a distance threshold (currently set at 5 meters) to a branching point on the path demarcated by the current route.

`in_path_segment`: This is a boolean value that is true if `on_path` is true and `at_branching_point` is false. In other words `in_path_segment` is true if the pedestrian is positioned on a path segment between two branching points.

`at_goal`: This is true if the pedestrian is within a distance threshold (currently set at 10 meters) of the currently pursued goal.

`standing_still`: This is true if the last three seconds show an average speed of less than some constant, currently .5 meters per second.

`receiving_TTS`: This is true when an utterance is being voiced on the PhoneApp.

`heading_correction`: This is a computed angle that gives the clockwise rotation necessary to align the pedestrians heading with the heading of the elementary segment that they are currently on. In the case that the pedestrian is not on an elementary segment of a path segment, this angle is the 'overland' best correction to their current heading.

`current_goal`: This is the id of the current goal.

`current_tour`: This is the id of the current tour.

`euclidean_distance_to_goal`: This is the current distance 'as the crow flies', from the pedestrians position to the goal.

`path_distance_to_goal`: This is the current summed distance of all elementary path segments between the pedestrian and the goal (plus any additional distance required to get on a path segment in the case that the user is not already there).

`heading_toward_goal`: This is the angle heading that the goal is in from the user as the crow flies.

A PostgreSQL trigger on inserts into the `GPSMeasure` table executes a stored procedure that builds and inserts a state tuple into `PedestrianState`. Although this requires a fair bit of calculation, given that states only need to be calculated once per second, currently the representation is being calculated well under budget in single user trials. Since inserts into the `GPSMeasure` table are once per second, so too are inserts into into the `PedestrianState` table. Thus we record a pedestrian state for every second of their session, when testing the system with a single pedestrian.

4.3 Communication rules

As we monitor pedestrian state, we need to decide when and which utterances to voice to the pedestrian to guide them to their goal. In this initial prototype we model this as a simple *reactive system* implemented in a set of *event-condition-action* (ECA) rules [6] on the `PedestrianState` table. We term these *communication rules* where the *events* are inserts into `PedestrianState` table, *conditions* are queries on the inserted tuple possibly joined with additional tables and *actions* are inserts into an `Utterance` table (see figure 5). The `Utterance` table records exactly which utterances will be, or have been, voiced to the pedestrian.

```
1  CREATE RULE TurnThroughDecisionPoint AS
2  ON insert TO PedestrianState
3  WHERE  NEW.onPath AND NEW.atBranchingPoint
4         AND NOT NEW.receivingTTS
5  DO ALSO (
6    INSERT INTO Utterance(uid, session, time, utterance)
7      VALUES(
8        NEW.uid,
9        NEW.session,
10       NEW.time,
11       currentTurnUtterance(NEW.uid));
12    );
```

Figure 4: An example communication rule

An example communication rule appears in figure 4. Following the syntax of PostgreSQL, rules are named (`turnThroughDecisionPoint` in line 1) with specification of events (e.g. line 2), conditions (e.g. lines 3-4) and actions (e.g. lines 6-11). The purpose of this rule is to direct the pedestrian on to the next route segment as they arrive at a decision point.

Additional rules are:

`continueRouteSegment`: The action is to encourage the pedestrian to continue following the route segment they are in. The conditions for this are that the pedestrian is making good progress on a route segment. An analogous rule is defined over elementary segments.

`correctHeadingToLeft`: The action is to tell the pedestrian to correct their course to the left. The condition is that the pedestrian is veering off the current elementary segment to the right. Analogous rules are `correctHeadingToRight` and `correctHeadingTurnAround`.

`offPathHeadingCorrection` Given to direct the pedestrian toward the most appropriate path segment. The condition is that the pedestrian is not on any path segment of the current route.

`distanceReport` given to report how much further to the goal.

`orientToGoal` given to report that the pedestrian is facing the direction of their goal.

`encourageMovement` given to inform the pedestrian that they need to walk so that a heading can be calculated.

When multiple communication rules simultaneously have true conditions, only one is allowed to generate an utterance. This is guaranteed by implementing a separate PostgreSQL rule on inserts into the `Utterance` table. Lexicographic order on rule names determines an a precedence relation among communication rules.

4.4 Generation and realization of utterances

When a communication rule inserts an utterance into the `Utterance` table, it must be voiced (i.e. *realized*) to the pedestrian. This is achieved by the Controller which polls the the `Utterance` table, shunting new utterances to the PhoneApp. The Controller currently does this once per second.

```
Utterance(uid, session, start_time, end_time, text)
```

```
DecisionPointInstruction(id, from_path_segment_id, to_path_segment_id, text)
RouteSegmentInstruction(id, path_segment_id, text)
ElementarySegmentInstruction(id, elementary_segment_id, text)
```

Figure 5: The utterance and pre-generated instructions tables

Currently utterances are pre-generated via an off-line natural language generation package that systematically computes route instructions (possibly including references to landmarks) over all possible decision points and route segments. Authoring tools enable us to override default generated utterances with human written content. For example utterances 16-20 in figure 1 were authored into the system.

No matter their source, pre-generated utterances are stored in the tables `DecisionPointInstruction`, `RouteSegmentInstruction` and `ElementarySegmentInstruction` shown in figure 5. Stored procedures (e.g. `currentTurnUtterance` in figure 4) retrieve these utterances and insert them into the `Utterance` table for immediate realization.

5 Initial Observations

We have implemented the prototype described in this report and conducted a series of pilot tests. Most of our tests were dedicated to validating capabilities and confirming bug fixes.

While we we have not yet run any formal experiments, as of October 2012 we have developed the system to a state that will soon be sufficiently robust and effective for navigation experiments with random human subjects.

5.1 Run time performance and stability

The run time performance of the system is adequate. For a typical test under the conditions depicted in figure 1, the average time to calculate the pedestrian states was well under the 1 second budget. The average lag time of GPS reports are approximately 60 ms.

The stability of the system has improved substantially from our first running prototype (in June 2012) to the prototype at the end of our latest development phase (in October 2012). These stability issues were addressed mostly by redesigning and recoding initial components. In addition we have systematically tracked known problems and feature requests using the REDMINE bug tracker. Our prototype must be very reliable before we commit substantial resources to evaluation.

5.2 Effectiveness of navigation

To be blunt, our initial implementation, before any experiences were gathered and parameters tweaked, would not have been able to reliably guide a user to a goal. For example problems like the quantity and timing of utterances (too much or too little speech, utterances issued too late or too early) and oscillations in the calculation of facing direction led to a frustrating user experience. Thus much effort was directed toward fixing parameters in the underlying system, coding alternative phrasings, adding further communication rules and state variables, etc. In addition we determined that scheduling of utterances in synchronization with user position is a critical capability that is not easily finessed in our purely reactive approach. This orients our future efforts toward the challenging problem of predicting user position and scheduling utterances accordingly.

Testing with the VirtualPedestrian

While figure 1 shows the display of our VIRTUALPEDESTRIAN tool in tracker mode, figure 6 shows our VIRTUALPEDESTRIAN tool in a 'virtual mode'. In this mode the user controls the heading and speed of the pedestrian on a map. The map portion of the tracker shows the plot of the GPS positions along with their inferred position. The portion to the right gives the log of utterances that the system generates. The VIRTUALPEDESTRIAN in 'virtual' mode engages in exactly the same protocol that the PHONEAPP engages in with the system. Moreover the tool can simulate GPS error and the route can be hidden from the human operator and text can be voiced as text-to-speech. In this mode we can run tests and perhaps even evaluation without having to go out in the streets.

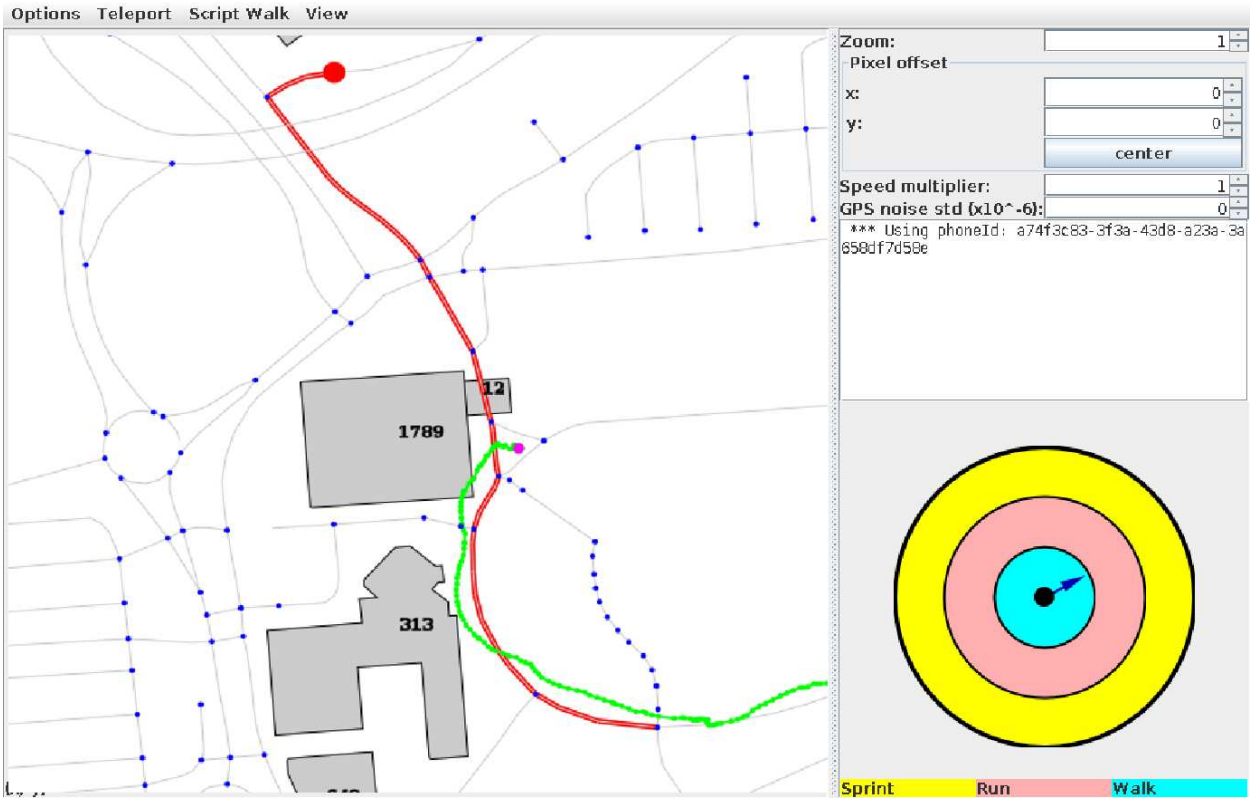

Figure 6: A human subject using VirtualPedestrian to follow a virtual route.

Actual field tests

No matter how many virtual pedestrian tests we run, what counts is how the system actually performs with real users in the field who do not know the system, but can only follow the instructions that the system voices. Our plan here is to start field testing by generating random tours unknown to a single human tester. Since the field tester will be unaware of the destination that they are currently being guided to, the system will need to be effective in guiding the tester through route following instructions. Even if the tester is one of the authors of this report, this will give us insight into the effectiveness of various communication strategies. No doubt other unforeseen issues will also come up. Only after performing this cheaper form of *auto-evaluation* (or testing) shall we consider a larger evaluation with random human subjects.

6 Conclusions

This report has provided a snap-shot description (as of October 2012) of our work on building a pedestrian navigation system based on active database technology. Although we have only performed cursory testing so far, we believe that the system holds out promise as a

scalable platform to effectively guide pedestrians to goals in the city. The ability to author content directly into the system gives a practical approach to override machine generated descriptions. Moreover this authoring approach may underlie a future method by which textual descriptions (or perhaps even audio content) may be crowd sourced.

The work here also brought to the surface some interesting query processing issues. Because of the noisy nature of position reports, we have found statistical time-series queries of particular use. For example time series queries such as "Is the standard deviation of heading less than 10 degrees over the last 10 seconds?" are the basis of determining facts like whether we have a stable heading, which in turn is a condition for rules like `orientToGoal` of section 4.3. One can imagine even more complex time-series queries that could determine, for example, if the pedestrian is likely to be waiting for a traffic light ("Has the user walked straight up to a road and waited longer than 3 seconds in stand still?"), is disoriented ("Has the user returned to this spot after walking in a 'circle' lasting 3-4 minutes?"), is making progress ("Over the last minute has the user moved at least 30 meters nearer to the goal?"), etc.

Thus far we have not yet been forced off a traditional relational approach in favor of a stream-based approach. Since our focus has been on boosting scores on metric 1 (see section 1), we have in fact been limited to running 30 minute tests with single users – in essence isolating our attention to windows of no more than 2000 tuples in the `PedestrianState` table. As we transition to larger scale studies and in particular explore methods to crowd source audio route instructions, we anticipate transitioning to a stream based approach [5]. With our attention firmly focused on our two evaluation metrics, it will be interesting to see how this project progresses.

7 Acknowledgments

Our schema design for the path network and landmarks was developed through earlier discussions with William Mackaness and Phil Bartie of SpaceBook partner Edinburgh University. The basic notion of basing communication actions on dynamic pedestrian state is inspired by the work of Oliver Lemon's group at SpaceBook partner Heriot Watt University. Initially we used a PhoneApp (and server) implemented by SpaceBook partner Liquid Media. Finally we would like to acknowledge the programming work of Markus Karlsson and Linus Närva at Umeå University.

References

[1] S. Coast. How OpenStreetMap is changing the world. In *proc. of W2GIS*, page 4, 2011.

[2] R. Dale, S. Geldof, and J.-P. Prost. Using natural language generation in automatic route description. *Journal of Research and Practice in Information Technology*, 37(1), 2005.

[3] M. Duckham, S. Winter, and M. Robinson. Including landmarks in routing instructions. *J. Locat. Based Serv.*, 4(1):28–52, Mar. 2010.

[4] M. Henning. A new approach to object-oriented middleware. *IEEE Internet Computing*, 8(1):66–75, Jan. 2004.

[5] S. J. Kazemitabar, U. Demiryurek, M. H. Ali, A. Akdogan, and C. Shahabi. Geospatial stream query processing using Microsoft SQL Server StreamInsight. *proc. of VLDB*, 3(2):1537–1540, 2010.

[6] W. Kim, editor. *Modern Database Systems: The Object Model, Interoperability, and Beyond*. ACM Press and Addison-Wesley, 1995.

[7] W. Mackaness, J. Boye, S. Clark, M. Fredriksson, H. Geffner, O. Lemon, M. Minock, and B. Webber. The spacebook project: Pedestrian exploration of the city using dialogue based interaction over smartphones. In *Proceedings of the 8th Symposium on Location-Based Services (LBS)*, Vienna, Austria, 2011.

[8] Y. Miyazaki and T. Kamiya. Pedestrian navigation system for mobile phones using panoramic landscape images. In *SAINT*, pages 102–108, 2006.

[9] C. Nothegger, S. Winter, and M. Raubal. Computation of the salience of features. *Spatial Cognition and Computation*, 4:113–136, 2004.

[10] J. Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[11] K.-F. Richter and A. Klippel. A model for context-specific route directions. In *Spatial Cognition*, pages 58–78, 2004.

[12] M. Roth and A. Frank. Computing em-based alignments of routes and route directions as a basis for natural language generation. In *COLING*, pages 958–966, 2010.

[13] M. Theune, D. Hofs, and M. V. Kessel. The virtual guide: A direction giving embodied conversational agent. In *Proc. of Interspeech 2007*, pages 27–31, 2007.