Do You See What I See: Towards A Gaze-Based Surroundings Query Processing System

Shinjae Kang, Byungjo Kim, Sangrok Han, and Hyogon Kim
Korea University
Seoul, Korea
{sjkang03, mirene, sangrok, hyogon}@korea.ac.kr

ABSTRACT
A smart car can be defined in various dimensions. In this paper, we consider a gaze-based driver query processing system where the vehicle recognizes where the driver is looking at when she asks about roadside landmarks. By linking the tracked gaze of the driver with annotated maps and geo-location information, the system can let the vehicle provide immediate answers to the driver. We discuss the design and implementation of the system, along with the test results during real driving.

Author Keywords
Gaze tracking; driving; query processing; map; safety.

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

INTRODUCTION

A 'smart car' is a broad notion, but one intelligence that smart cars will likely have in future is answering the driver's questions about surrounding landmarks and points of interest (POIs) based on driver's gaze. For example, queries like "What is that building?" or "Is there a restaurant on this side?" can be useful while driving because the driver does not have to look away from the road to consult information devices inside the vehicle such as sat-nav map. It would distract driving, and raise the chances of accidents.

In this paper, we build a prototype system that combines driver's gaze, geo-location information, and annotated map to answer such queries. Although there have been studies on approximating the gaze with head rotation [8, 9], fast eye movements that do not accompany head rotation frequently take place during driving for safety concerns [2]. Even in case the head rotation takes place, eyes can make movements in addition to them. Therefore, eye movements tracking in addition to head rotation measurements should be performed to track the driver's gaze more precisely and answer queries with lower error rates. Although there is a recent work that uses both the head turn and the eye gaze to track driver's view and attention [13], it requires a head-mounted camera (Google Glass) that may not be always available to a casual driver, in addition to separate driver-looking cameras. Our system uses a single, relatively cheap dashboard-mounted camera to track both and to find the object that the driver is looking at.

There has also been research on similar query systems that use other input modalities such as finger pointing [4, 11]. But using fingers with hands on the steering wheel should be more constrained than eye gaze and can interfere with driving. On the other hand, finger pointing with a hand off the steering wheel would be less safe.

Below, we discuss the design and implementation of the gaze-based surroundings query processing system. We also present test results during real driving. Finally, we discuss issues that affect the performance of the system and future work that will address them.

SYSTEM DESIGN AND IMPLEMENTATION

Figure 1 shows the overall architecture of the system. It is mainly composed of five components: 1) head rotation tracking, 2) eye gaze tracking, 3) geo-location, 4) map access, and 5) speech output. The most central functionality that is developed in this paper is the eye gaze tracking. Below, we describe the five components in detail, with the emphasis on the eye tracking.

Head rotation tracking using Kinect

Instead of developing a native head rotation tracking module, we employ Kinect™ V1 because it provides relatively accurate head rotation measurements at a low cost. The field of view of the Kinect detection algorithm is measured to be approximately 46° wide [1], so it is sufficient to capture the entirety of the driver's face from the dash board. Amon and Fuhrmann show through measurements that with approximately 1 meter between the driver's face and Kinect, the error in the horizontal rotation angle reported by the device is 2° on average in most turnable rotation positions [1]. For face

AutomotiveUI15, September 1 3, 2015, Nottingham, UK

http://dx.doi.org/10.1145/2799250.2799285

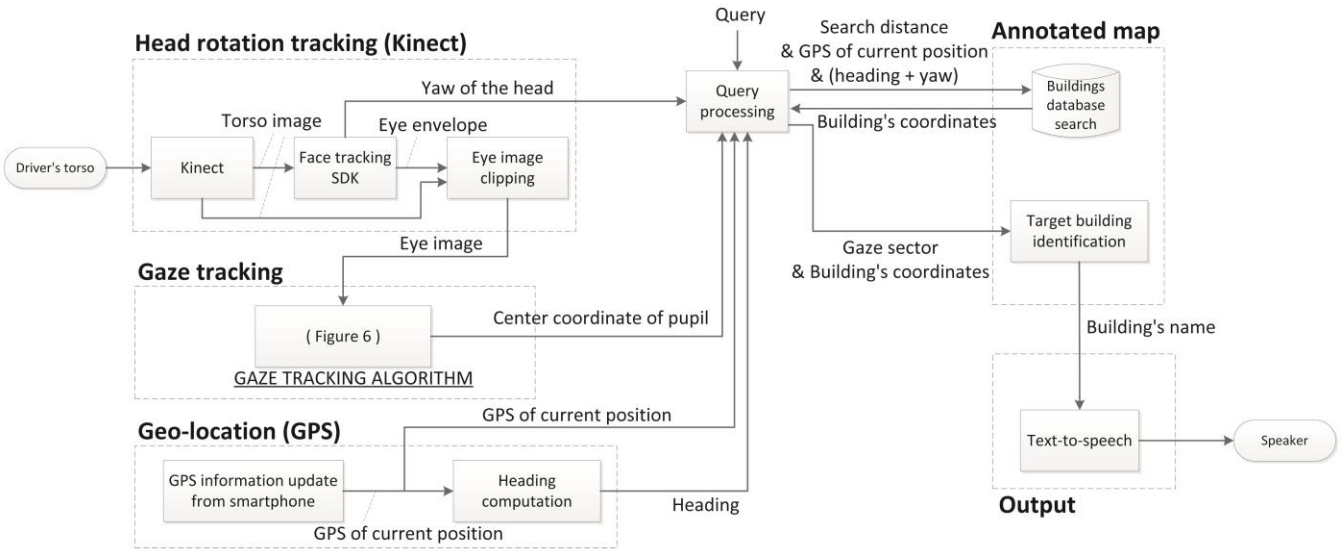


Figure 1. System architecture

tracking with Kinect SDK v.1.8, we select the near mode and the seat mode. The near mode lets Kinect recognize the driver within 1 meter, and the seat mode lets it work with only the torso without the entire body image.

Once the face is detected (Figure 2(a)), Kinect can track the head rotation with a model in which it can detect movements around three axes (Figure 2(b)). Among these, we only use 'Yaw' in our system. This is because in the current implementation we only consider the horizontal gaze movements. Vertical movements are not considered in this paper, as it would be only necessary when landmarks overlap in horizontal coordinates but having different heights. Also, we assume that 'Roll' does not occur frequently, so it is not considered in this paper, either. Coping with 'Pitch' and 'Roll' is left for future work. Kinect face tracking specification says the system tracks user's head yaw within 45°, but works best when less than 30° [6]. In Figure 3, we show a yaw trace example that we obtained during a real driving experiment. Kinect generates six to eight image frames each second, and the trace spans over two minutes. As we notice, the driver rarely turns the head over 30°, so Kinect serves our purpose well.

The Kinect face model also provides the approximate positions of the eyes in the face (Figure 2(c)), and returns its coordinates among which we employ four: topmost, bottommost, leftmost, and rightmost. Using this information, we clip the eyes area and give it to the gaze tracking component to obtain the pupil positions.

GAZE TRACKING

Note that since the clipped image is a small fraction of the original Kinect camera image, the gaze tracking algorithm has to work with a low resolution image of the eyes to detect the pupil positions. Approximately, the eye image size is 240×100 pixels, clipped from the original 1280×960 Kinect image. Due to this small number of pixels in each dimension, it lowers the precision of the pupil movement detection. In the future work, we will consider a separate camera image that has high resolution for the eyes.

Effect of head turn on pupil disposition

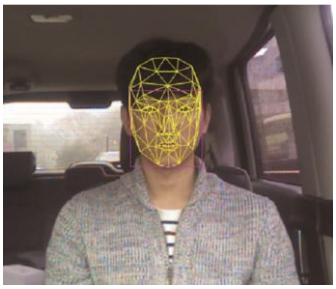
When the yaw (head rotation) y is not zero, the pupil disposition changes from the perspective of the Kinect camera. In Figure 4, take the driver's right eye for example. On the left, the pupil is at the center of the eye. But as the subject turns the head, the pupil looks to move to the right from the perspective of the camera. Suppose the pixel dispositions for the maximum head turn and zero head turn are x_2 and x_1, respectively, both with straight eye gaze. (These two values are obtained when we calibrate the system at the beginning of the system operation.) In this paper, we limit the maximum value of head turn that can be processed by the system to 30°. Then the additional (quasi-) pixel disposition of the straight gaze given a non-zero yaw $0° \le y \le 30°$ is given as

$$(x_2 - x_1) \cdot \frac{y}{30°} \qquad (1)$$

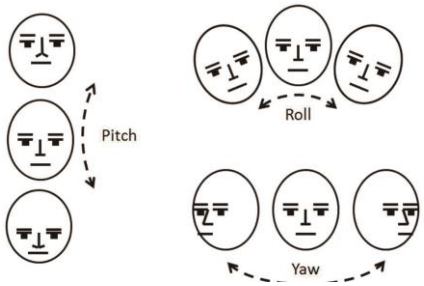
This entity will be used later to compute the center of the view field for the eye gaze under the head rotation of y. Note that it is a linear approximation, and we will use a better model to reduce the gaze tracking error stemming from it.

We observe that we do not need to compute the pupil disposition for both eyes for two reasons. First, since the driver is looking far away, the pupil dispositions are equal on both eyes. Namely, vergence is not a factor for consistently looking at far objects as in driving. Second, when head turn takes place, the eye in the direction of the turn (from the camera perspective) is compressed in the eye image, and it becomes difficult to translate the pupil movements in pixels into the angular values. Namely, when the driver turns the head to left we use her right eye (which is closer to Kinect), and *vice versa* (Figure 4). Therefore, we use only one eye in the gaze tracking algorithm. Finally, the horizontal size distortion for

the eye closer to camera is not significant when the maximum head turn is limited to 30°. But in future work, we will reflect it on the eye gaze estimation algorithm for a higher precision.



(a) Face



(b) Movements model

(c) Positions of eyes

Figure 2. Kinect models

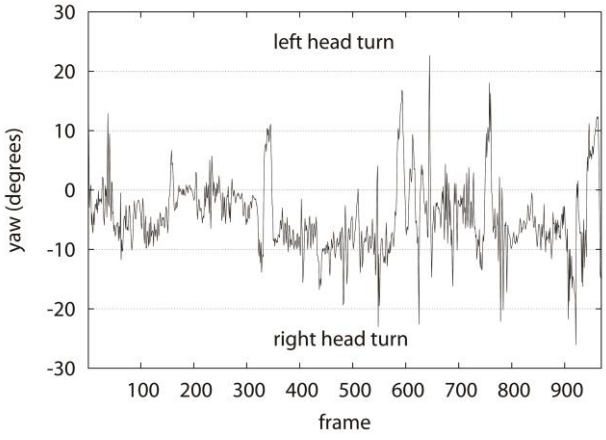


Figure 3. Observed driver head rotations during a driving experiment

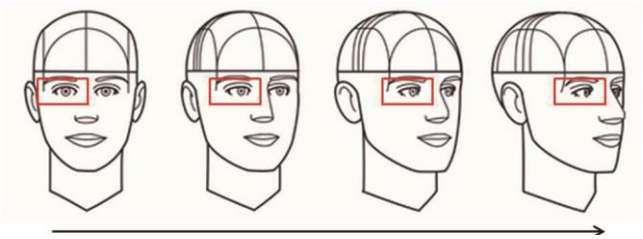


Figure 4. Pupil disposition change as a result of head rotation

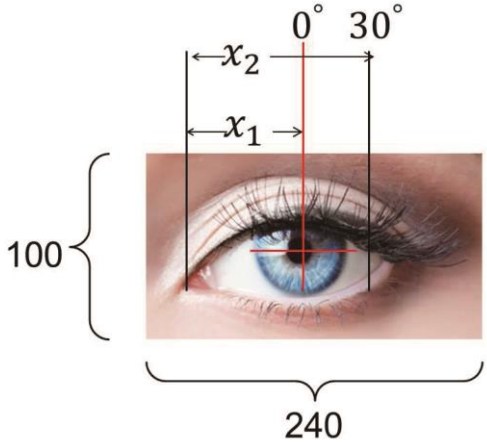


Figure 5. Getting pupil dispositions from the Kinect eye image

Gaze tracking algorithm

The gaze tracking algorithm is composed of several modules grouped into three phases as depicted in Figure 6. The first step is to binarize the color image from the eye image. The original RGB image is first turned into a grayscale image, and then pixels are classified into either white or black. The classification uses a threshold value that divides the two colors, which is adjusted by the overall brightness of the eye image. The varying threshold is useful when the lighting condition changes as we face during driving, *e.g.* from sunlight, headlight, or streetlight.

The radius of the pupil affects the threshold to be used in the binarization to find the size of the pupil. Through erosion and dilation, we eliminate as much as possible the area in the image that is not the pupil (*i.e.*, noise). Then, we obtain the center of the pupil and its coordinate. This output from the gaze tracking algorithm cannot be used immediately. Instead, we need to filter it for any wild fluctuations. The fluctuations can stem from two sources. The first is vibration caused by the vehicle engine and the uneven road surface. The second is from the eye itself, which is *saccade*. Any spikes from either should be eliminated. Below, we briefly discuss the saccade and how we eliminate them.

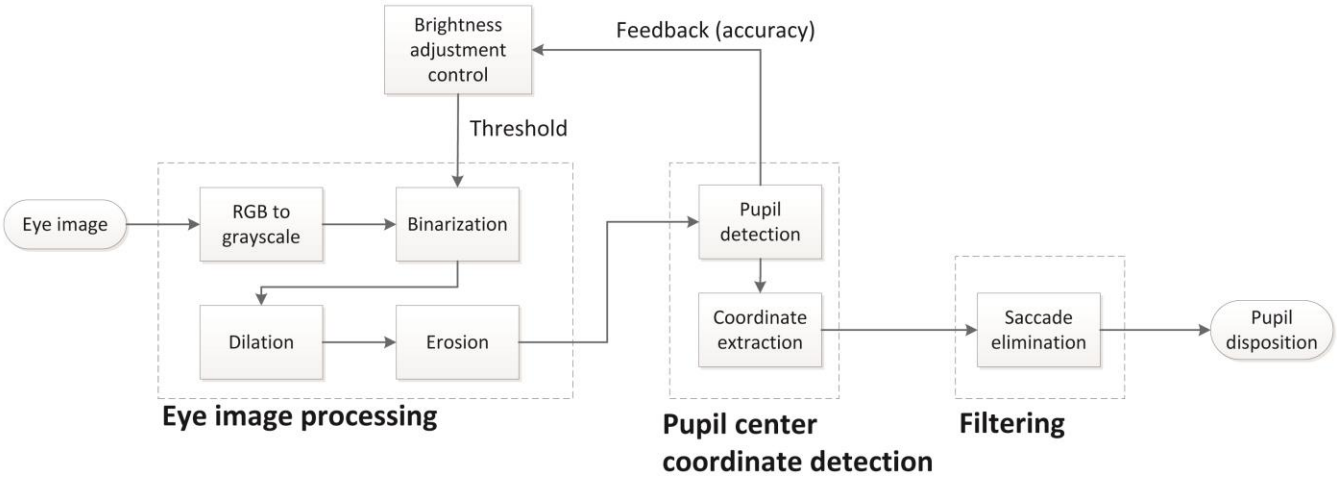


Figure 6. Gaze tracking algorithm

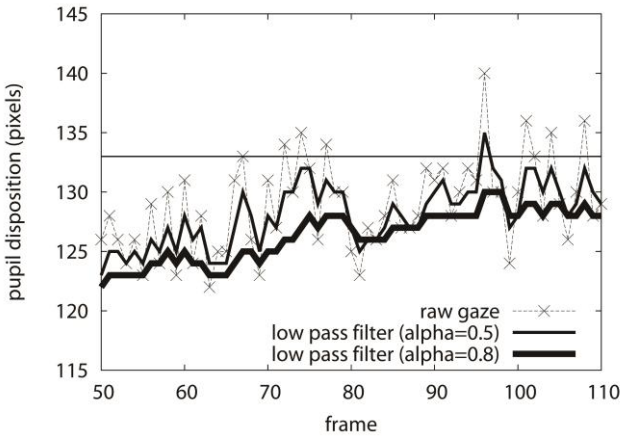


Figure 7. Saccade elimination

Filtering out driver's saccades

Humans and many animals do not look at a scene in fixed steadiness; instead, human eyes make jerky saccadic movements and stop several times, moving very quickly between each stop. One reason for the saccadic movement is that the high-resolution portion of vision is very small in humans, only about 1-2° of vision. By moving the eye, small parts of a scene can be sensed with greater resolution. This uncontrollable and fast saccadic movements happen even when the driver intently looks at an object, and it works as noise to the gaze tracking algorithm. Figure 7 shows an example of the pupil movements in pixels against eye image frames when the test subject is gazing at a point on a test panel 1 meter ahead. The black curve is the raw gaze trace, and the horizontal line in the middle of the figure is the boundary between adjacent sectors in the driver's view field (see 'Sectorizing view field' below). In this example, the driver is looking close to the boundary but not crossing it. But we notice that the gaze tracker is reporting values across the boundary at least seven times during trace.

In order to smooth out the saccadic noise, we use a low-pass filter. The smoothed pupil angular movement $\bar{\phi}$ at t is given by

$$\bar{\phi}_t = (1-\alpha)\cdot\phi_t + \alpha\cdot\bar{\phi}_{t-1}$$

where $\phi(t)$ is the raw pupil movement at t and α is the weighting coefficient. Figure 7 shows the corresponding pupil dispositions in pixels where larger α value leads to more stable gaze trace. At α = 0.5, the boundary crossing is reduced to one, and at α = 0.8, none. In order to obtain fast response that matches fast gaze shift between landmarks, however, we set α to 0.5 in the current prototype. The saccade elimination also counteracts the pupil movements due to vehicle- and road- induced image vibrations.

GEO-LOCATION

The next pieces of information we need are the current position of the vehicle and its heading. They are obtained from the GPS readings. In the current implementation, we use the GPS values provided by an Android smart phone. The GPS readings are generated by the smart phone at 1Hz on Android.

Let two GPS coordinates be $P_t = (a_1, o_1)$ and $P_{t+\Delta t} = (a_2, o_2)$ where a is the latitude and o is the longitude of a coordinate. From these coordinates, we calculate the vehicle heading as

$$h = \tan^{-1}\frac{\sin\Delta o\cos a_2}{\cos a_1\sin a_2 - \sin a_1\cos a_2\cos\Delta o} \qquad (2)$$

where $\Delta o = o_2 - o_1$. Note that the equation is for the case when the denominator has positive sign, but when it is negative we need to add or subtract π depending on the sign of the numerator.

One caveat in using Equation (2) is when the two consecutive GPS readings are the same. For instance, it can happen when the vehicle stands still due to traffic. If we keep such GPS readings, we can get incorrect "guesses" such as 0°, 90°, 180°, *etc*. By using immediately earlier GPS reading that is not

equal, however, we can eliminate the fluctuation, which is shown as the solid curve.

Combining the heading h, the head rotation y, and the gaze angle ϕ we finally obtain the azimuth of the gaze as

$$Z = h + y + \bar{\phi} = \theta + \bar{\phi}$$

which we use to get a landmark or a building in the projection (Figure 8). As $h + y$ defines the center of the driver's view field, we will call it θ below for convenience.

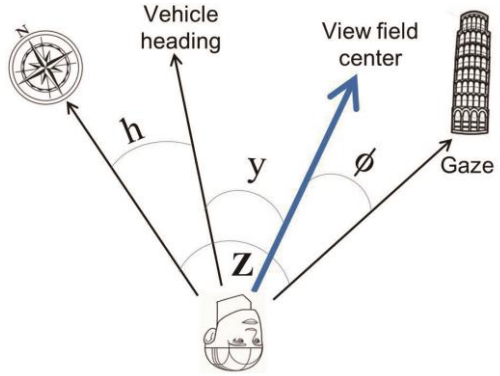


Figure 8. Azimuth Z as an approximation of true gaze direction

MAPPING ON ANNOTATED MAP

Once the azimuth Z and the current vehicle position P are obtained, we are ready to overlap the gaze on the annotated map so as to see which landmark or building it first crosses on the map. Due to the imprecision in the computation of $\bar{\phi}$, however, Z is only an approximation to the true gaze direction. Therefore, we can only broadly specify the gaze direction using Z. Below, we first discuss how we define the broadly specified areas in the driver's field of view.

Sectorizing view field

During driving, gazing at an object to the corners of the eye tends to be dangerous. Also, for large $h + \bar{\phi}$, the detection algorithm will have difficulty to track the driver's pupil. For these safety and precision reasons, therefore, we assume that the driver's view field from eye gaze movement is limited to 60°. So in total, we require $|y + \phi| \leq 60^{\circ}$ on either side of the vehicle heading.

As mentioned above, due to the fact that Z is an imprecise approximation to the true gaze, we need approximate map matching operation. For this purpose, we divide the view field of 60° into smaller sectors. Then we search the nearest building or landmark in the sector where the gaze $\bar{\phi}$ falls. To determine a proper size of each sector, we performed an experiment. In the laboratory setting, we place a test panel 1 meter in front of the test subject. With the head fixed ($y = 0$), the test subject moves the eye gaze within 60° range, and see if the gaze tracking algorithm correctly matches the sector with the true gaze direction. We start from 3 sectors, and increase the number to 6, 9, 12, and 15 with increasingly narrow sectors. Table 1 shows the result.

Sector size	*Detection rate*
20°	100% (10/10)
10°	80% (16/20)
6.7°	70% (21/30)
5°	45% (18/40)
4°	38% (19/50)

Table 1. Sector size and correct detection ratio

The experiment shows that in order to obtain upwards of 80% precision, we need to have each sector wider than 10°. So in our implementation, we set it at 15°. The precision at this sector size is 95%. We perform another experiment where the head turn takes place simultaneously. In order to check for the worst case, we tested with the maximum yaw of ± 30°. Note that in these cases the gaze tracking algorithm has to deal with 60° worth of pupil movements. Table 2 show that still the gaze tracking produces higher than 80% precision.

Yaw (head)	*Gaze tracking precision*
0°	95% (19/20)
+30°	85% (17/20)
-30°	80% (16/20)

Table 2. Gaze tracking precision with maximum allowed head turns

The view field of 240×100 pixel eye image is divided into with 4 sectors with the boundaries shown in Figure 9. For instance, the first sector includes pixels 75, 100, and all in between, the second includes 101, 120, and all in between, *etc*. The reason that the sector sizes are not even is that the four eye boundary coordinates (top, bottom, left, right) tend to fluctuate. Therefore, when we calibrate the gaze tracker, gazing at the center position in each sector (big dot in Figure 9) results in a set of scattered samples of gazed position. In order to establish the boundary between adjacent sectors s_i and s_{i+1}, we find the rightmost sample from s_i and the leftmost sample from s_{i+1} and set the boundary at the middle. Note that pixels left of 75 and right of 167 are outside the 60° view field used in the prototype.

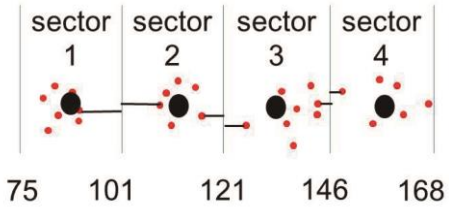


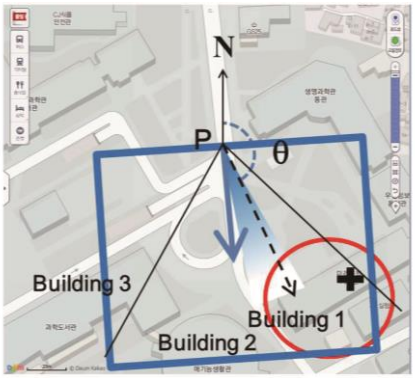
Figure 9. Sectors in pixels

The sectors boundaries calibrated as above are added with the pixel disposition from the seeming pupil movement (Equation 1) for non-zero yaws. Now, the center of the view field is aligned with θ (essentially the center of head), to which the actual pupil disposition ϕ is added to find the

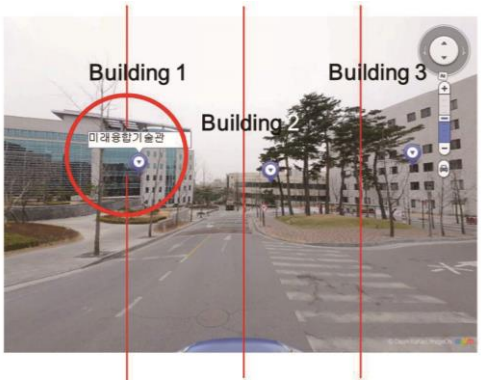
matching sector according to the sector partitions in Figure 9. Finally, the length of the lateral edge of the sector can be configured to d meters, *e.g.* 100 meters.

Matching against an annotated map

It would be a major undertaking to build an annotated map of an entire city or a geographical area, so it goes well beyond our capability. In this paper, we pick multiple buildings from a test area and obtain their GPS coordinates and names. We build a database of the records for the buildings where each record is composed of the GPS coordinate and the attributes including the building name. This is our annotated virtual map. In future work, we will develop methods to automatically generate the database from publicly available maps.



(a) Annotated map



(b) Sectors and annotations superimposed on driver's view

Figure 10. Sectors used in matching

Given the current position P of the vehicle and $\theta = h + y$, we consider a rectangle that is of size $d \times d$ aligned with θ (Figure 10(a)). This rectangle is the search scope at (P, θ). At this point, the driver can select one of the sectors using head turn and gaze. We check which building coordinates intersect with the gazed sector, and sort them in the order of distance. Then we pick the closest building and returns the attributes. However, there is one more factor that we need to consider. In Figure 10(b), suppose the driver is gazing at the edge of the Sector 2 where there is 'Building 1'. If we use only a single coordinate for the building, *e.g.* its center coordinate, the gazed sector may not cross the coordinate. In the figure, the cross marks the center coordinate, and it does not intersect with the gazed sector, even if the building itself does. Therefore, in the annotated map database, the coordinate of a building or a landmark should be extended to cover all or most of the visible structure for a better matching performance. For instance, we can use a set of coordinates that forms a convex hull of the given landmark. Below, we will compare the performance under the two implementation choices.

Query input and query response output

As the last component of the prototype system, there is the query response output. Since the primary reason behind choosing the gaze as the input modality is to avoid moving driver's gaze from the road ahead to an in-car display, we use the text-to-speech (TTS) output modality. Once the system finds the annotation of the building, the text is given to the TTS module so that it can be translated and played. We implement the TTS module using Microsoft Speech API (SAPI) v5.4 [7].

In the same vein, the query input in the current system does not use finger or speech. The speech input is a difficult problem in itself amid various driving noises, and it has been also shown to have a driver distraction problem [12]. So we assume that a steering wheel mounted button can be used in combination with the gaze for the driver to issue the query at the push. In the current implementation, therefore, the test subject pushes a key to issue the query. However, in future, the input modality will have to include the speech input for more complex types of queries than just asking the identity and whatever information that the annotation has for the queried landmark.

DRIVING EXPERIMENTS

Test setup

We test the developed prototype in real road environments while the vehicle is on the move. We drove a car at the maximum of 30Km/h along straight roads, in the Paju City in Korea (Figure 11(a)). There are nine test buildings along the streets in the driving path (with check marks) along which we make multiple rounds. As discussed above, these are the ones whose records are in the map database. For driving safety, we test the system on the passenger side, where the instruments are also placed. When nearing test buildings, the test subject hits the enter key while gazing at the building.

The test hardware is composed of three parts: a smartphone providing GPS sensor readings, Kinect, and a PC that runs the software system. The software processes the gaze, identify a matching landmark in the annotated map database, and produces TTS output for the query.

Performance evaluation

In the first experiment, we use a single coordinate for each building. We test two cases for head rotation. First, the head is fixed to look straight ahead, *i.e.*, the yaw is 0°. Second, the head moves freely horizontally. Table 3 shows the results for these two cases. For the fixed head case, out of 135 queries,

(a) Test area map

(b) Driver's view (boxed area in (a))

Figure 11. Driving test in Paju City streets

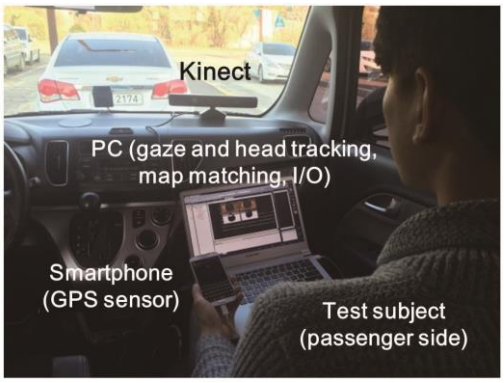


Figure 12. In-vehicle test setup

the system reported building names for 101, among which 79 were correct. In the free head experiment, the test subject queried about 117 buildings, and 81 of them were reported by the system among which 56 were correct.

	Head fixed	Head free
Hit rate	74.8% (101/135)	69.2% (81/117)
Accuracy	78.2% (79/101)	69.1% (56/81)
Answered & correct	58.5% (79/135)	47.9% (56/117)

Table 3. Identification performance under single point coordinate

Next, we test with five coordinates for a building: the center and four corners. Table 4 shows the performance. We can observe that the precision visibly increases by using multiple coordinates for landmarks. In both cases, it exceeds 65%. It is significantly higher than the free finger pointing experiments [11].

	Head fixed	Head free
Hit rate	87.4% (118/135)	85.5% (100/117)
Accuracy	82.2% (97/101)	76.0% (76/100)
Answered & correct	71.9% (97/135)	65.0% (76/117)

Table 4. Identification performance under five points coordinate

DISCUSSION

Here, we briefly discuss the issues in the current implementation that need to be solved to improve the precision of the system in the future.

GPS-induced problems

Currently we use the GPS output from a smart phone, as tapping the GPS output from the vehicle itself is not available. The GPS sensing frequency on Android is 1Hz, and it causes a practical problem. The vehicle can move a significant distance between two GPS readings, the vehicle position P from which the driver's view is projected can be incorrectly located. This contributes to incorrect detection of the landmarks. Also, the GPS reading itself can have a significant error. It leads to the same problem. Finally, the vehicle heading is computed based on the extrapolation of the last two GPS readings, and the errors from the individual readings lead to incorrect projection of the vehicle heading θ. In future work, we will use a better GPS module that provides higher number of readings per unit time, and with greater precision.

Kinect issues

There is an issue with the performance of Kinect depth camera that uses infrared (IR) imaging for face recognition. When the ambient sunlight is strong, we find that the depth camera experiences problem. The performance of the system over time varies greatly depending on the ambient lighting condition. A face recognition algorithm more robust to sunlight is desired.

As we mentioned above, Kinect fails to produce eye area image when the road is not even thus camera image shifts violently. It leads to the failure of eye gaze estimation. This problem is not specific to Kinect, and we will have to solve it for any camera system.

Calibration issue

Since the pixel distance is used to measure the gaze movement, we need calibration before the system is used. For instance, if the driver's head moves back and forth, the number of pixels can be different for the same amount of pupil movement. Therefore, we need to improve the implementation so that the relative magnitude of the pupil center movement is

measured for the given eye size, not the absolute number of pixels.

Gaze tracking issue

After sunset, the streetlight and the headlights of the passing vehicles can abruptly change the lighting on the eyes. Our eye gaze tracking algorithm is designed to dynamically change the threshold, but it works less accurately when the change is too abrupt. We need to work on a more robust algorithm against abrupt lighting condition changes on driver's eyes.

RELATED WORK

There are existing works that utilize gaze tracking in automotive environment. Oh and Kwak [9] maps the driver's gaze to control the vehicle head lamp. However, this work does not extract the gaze directly, but approximates it with face direction. Misu *et al*. also uses the face direction to replace eye gaze, and constructs a query system on it [8]. Poitschke *et al*. implements a multi-modal interaction system where the gaze is incorporated as one of the inputs [10]. But the focus is on the driver operation inside vehicle, where the gaze is used to relieve the driver of cognitive burden that could arise by using other input modalities to manipulate driver assistance and infotainment systems. Kern *et al*. also explore gaze as a new modality to control the infotainment system, in order to replace the interaction on the touch screen [5]. Fletcher and Zelinsky [3] determines if the driver is inattentive to road events using gaze tracking to warn the driver if it is missed by the driver. As for query processing, Fujimura *et al*. uses the wheel-constrained finger pointing as the input modus for querying and displaying the results to the head-up display [4]. Rümelin *et al*. uses free-hand pointing to identify and interact with distant objects [11].

CONCLUSION

In this paper, we design and implement a gaze-tracked surroundings query processing system for smart cars. We test the prototype system on real driving situations, and obtain upwards of 65% accurate answers for all queried buildings alongside the tested streets. Through the implementation and real-road testing, we find technical issues to improve the performance further. Finally, the precise gaze tracking technology can be used for other applications, such as driver distraction monitoring and dynamic headlight adjustments.

ACKNOWLEDGMENTS

This work was supported by the Ministry of Education, Science and Technology of Korea through the Mid-career Researcher Program under NRF Grant 2011-0028892.

REFERENCES

1. C. Amon and F. Fuhrmann. 2014. Evaluation of the Spatial Resolution Accuracy of the Face Tracking System for Kinect for Windows V1 and V2. In *Proceedings of the 6th Congress of Alps-Adria Acoustics Association.*
2. T. Bär, J. F. Reuter, M. Yguel, and J. M. Zöllner. 2012. Driver Head Pose and Gaze Estimation based on Multi-Template ICP 3D Point Cloud Alignment. In *Proceedings of the IEEE Conference on Intelligent Transportation Systems (ITSC).*
3. L. Fletcher and A. Zelinsky. 2009. Driver Inattention Detection based on Eye Gaze-Road Event Correlation. *Int. J. Rob. Res.* 28, 6 (June 2009), 774–801.
4. K. Fujimura, L. Xu, C. Tran, R. Bhandari, and V. Ng-Thow-Hing. 2013. Driver queries using wheel-constrained finger pointing and 3-D head-up display visual feedback. In *Proceedings of the 5th International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI'13).*
5. D. Kern, A. Mahr, S. Castronovo, A. Schmidt, and C. Müller. 2010. Making use of drivers' glances onto the screen for explicit gaze-based interaction. In *Proceedings of the 2nd International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI'10).*
6. MSDN Library. Face Tracking. Retrieved April 10, 2015 from https://msdn.microsoft.com/en-s/library/jj130970.aspx.
7. Microsoft. Speech API (SAPI) 5.4. Available at https://msdn.microsoft.com/en-us/library/ee125663(v=vs.85).aspx.
8. T. Misu, A. Raux, I. Lane, J. Devassy, and R. Gupta. 2013. Situated Multi-modal Dialog System in Vehicles. In *Proceedings of the 6th Workshop on Eye Gaze in Intelligent Human Machine Interaction (GazeIn).*
9. J. Oh and N. Kwak. 2012. Recognition of a Driver's Gaze for Vehicle Headlamp Control. *IEEE Transactions on Vehicular Technology 61, 5 (June 2012), 2008–2017.*
10. T. Poitschke, F. Laquai, S. Stamboliev, and G. Rigoll. 2011. Gaze-based interaction on multiple displays in an automotive environment. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC).*
11. S. Rümelin, C. Marouane, and A. Butz. 2013. Free-hand pointing for identification and interaction with distant objects. In *Proceedings of the 5th International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI'13).*
12. D. L. Strayer, J. Turrill, J. R. Coleman, E. V. Ortiz, and J. M. Cooper. 2014. Measuring Cognitive Distraction in the Automobile II: Assessing In-Vehicle Voice-Based Interactive Technologies. AAA Foundation for Traffic Safety report.
13. A. Tawari, A. Mogelmose, S. Martin, T. B. Møeslund, and M. M. Trivedi. 2014. Attention Estimation By Simultaneous Analysis of Viewer and View. In *Proceedings of the IEEE Conference on Intelligent Transportation Systems (ITSC).*