Combining eye-tracking with semantic scene labelling in a car

Working Paper · January 2018

CITATIONS
0

READS
901

8 authors, including:

Thomas de Boer
Delft University of Technology
7 PUBLICATIONS **11** CITATIONS
SEE PROFILE

Jim Hoogmoed
Delft University of Technology
3 PUBLICATIONS **20** CITATIONS
SEE PROFILE

Nathan Michiel Looye
Delft University of Technology
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Jim van der Toorn
Delft University of Technology
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Combining eye-tracking with semantic scene labelling in a car

Working Paper · January 2018

CITATIONS
0

READS
110

8 authors, including:

Pavlo Bazilinskyy
Delft University of Technology
38 PUBLICATIONS **54** CITATIONS

SEE PROFILE

Joost de Winter
Delft University of Technology
217 PUBLICATIONS **2,338** CITATIONS

SEE PROFILE

Combining eye-tracking with semantic scene labelling in a car

T. A. B. de Boer, J. Hoogmoed, N. M. Looye, J. R. P. van der Toorn, R. P. de Vos, J. Stapel, P. Bazilinskyy, J. C. F. de Winter

Working paper, January 2018

Abstract. We present an integration of a mobile eye tracker with semantic scene labeling in a car, using the Robot Operating System (ROS). Eye-gaze data from a head-mounted eye tracker is mapped onto the image of a forward facing camera in a car, by using marker detection, feature matching, and homography. Semantic scene labeling is applied to the image of the car, enabling the system to determine at what type of object the driver is looking.

1. Introduction

Semantic gaze mapping is the mapping of eye movement information to objects in the world. This concept would entail several new opportunities in car driving. For example, a driver assistance system could ignore a dangerous situation if it has ascertained that the driver has already noticed the situation, thereby minimizing nuisance warnings to the driver. Similarly, Tawari et al. [1] pointed out that "by knowing which pedestrians the driver has and has not seen, measures against collisions can be taken more accurately." Also for non-dangerous situations gaze mapping can be useful. For example, by displaying speed or distance to objects looked at the driver on a head-up display, the driver could be assisted in assessing traffic situations. Another application is gaze mapping in combination with vehicle-to-vehicle communication, so that a car knows if other road users have seen it.

In this paper, the design of a system is proposed, with the primary goal to combine eye tracker data with the camera feed of an instrumented car. The studied case is giving feedback when the driver is looking at individual objects, such as a car or a person. For this design, a Pupil Labs eye tracker and a Toyota Prius Hybrid Passenger vehicle are used. The Prius has a forward-facing camera with object recognition in the form of semantic scene labeling. It uses ROS as a framework, which is a viable choice for education and research purposes because of its open nature and online support [2][3].

2. System architecture

The setup consists of a head-mounted eye tracker with a forward-facing camera, denoted as C_{head}. The

car camera, denoted as C_{car}, is mounted behind the rear-view mirror. Both cameras face in a forward direction. C_{head} rotates and translates as the driver rotates and translates his/her head, whereas C_{car} is fixed to the car.

The camera images are continuously published to ROS topics and accessible by the developed ROS package. Furthermore, the eye tracker software publishes the driver's gaze position $\vec{p}$ as a set of x- and y-coordinates to a ROS topic. $\vec{p}$ is relative to the eye tracker C_{head}, denoted as ${}^{H}\vec{p}$.

The system is divided into three phases: two transformation phases (see Section 3.4) and one feedback phase (Fig. 1).

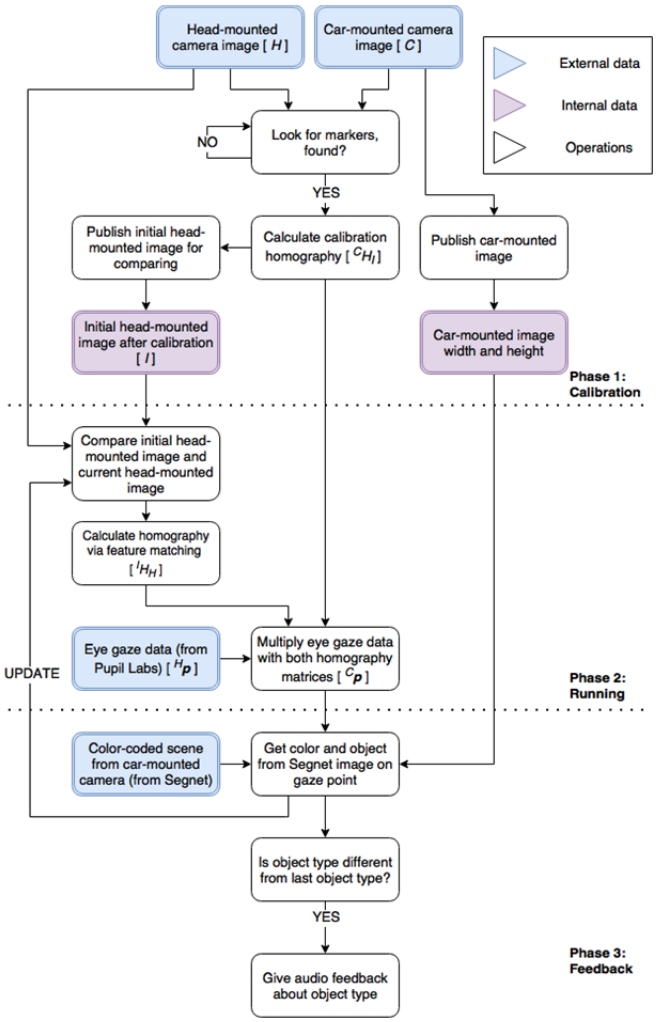


Figure 1. The system architecture with the three phases. The Calibration phase is executed once, followed by the Running phase which is executed continuously. The Feedback phase is initiated from the Running phase.

The first phase is a calibration phase. Here the C_{head} and the C_{car} images are compared. Marker detection (Section 3.3) is used to find corresponding points in the two cameras. When these are found, the initial homography matrix (Section 3.1) is computed: ${}^{C}H_{I}$, from the initial C_{head} image I to the C_{car} image C. The selected I is saved for further calculations in phase two.

The second phase is the running phase of the software. Corresponding points inside the car in both the current C_{head} image H and the initial image I are found and matched via feature matching (Section 3.2). These points are used to create a second homography matrix: ${}^{I}H_{H}$, mapping the C_{head} image H to the initial image I. The gaze data from the eye tracker is subsequently multiplied with both homography matrices to project the gaze data on the car image. Finally, the mapped gaze data is combined with the semantic scene labeling of the car to determine the type of object. (Section 3.5) This phase loops continuously allowing constant movement of C_{head}.

The third phase is initiated if the recognized object type matches one on which feedback is desired. If so, feedback is given (Section 3.5).

3. Subsystem architecture

To transform the gaze ${}^{H}\vec{p}$ to ${}^{C}\vec{p}$, homography, feature matching, and marker detection, are used. These methods, along with object recognition, are further described in this section.

3.1. Homography

Homography is a method for finding a transformation from one image to the other using a minimum of four points in each image. It maps a point on one projection of a plane to another projection of the plane, taking rotation, translation, skewness, and zoom, into account [4]. Homography assumes that all points are on the same plane, which simplifies the estimation but possibly introduces errors when the scene is not planar. Therefore, it is essential to choose points in the same plane, which is done in this design. First, the corresponding points, which are used for homography in OpenCV, are filtered based on distance and symmetry, decreasing the number of

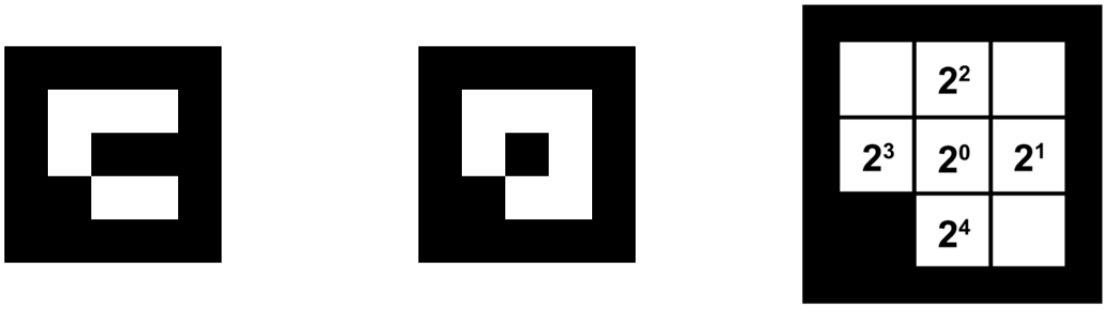
false matches. Then, during homography, the corresponding points are again filtered using RANSAC (RANdom SAmple Consensus), which tries to find the maximum number of matches that correspond with a randomly calculated transformation. A threshold is set that describes the maximum allowed error in pixels for a matched pair to pass. In our design, the threshold is set to 10 to keep the processing speed high.

3.2. Feature matching

To create any homography matrix, corresponding points in two images are needed. Feature matching is a method in computer vision for finding point correspondences between two different images, without prior knowledge about relative camera positions and rotations or the scene. It is often used in camera stabilization, stitching images, or object tracking. Feature matching is a two-step process: detection and matching. Regarding detection, OpenCV has included several feature detectors. Herein, the ORB algorithm [5] is used for the real time application in this design. Once the features in the two images are detected, a matcher is used to find the corresponding pairs of the same features. This is done with either Brute Force (BF) matching.

3.3. Marker detection

The second method to determine corresponding points is marker detection. To detect four pairs of points in C_{car} and C_{head} that match, four different markers (Figure 2a,b) are used. The markers consist of a 5 x 5 matrix, and are detectable in both images by a script using OpenCV. Valid markers have a black square in one of the corners, which is also used to rotate the marker upright. To find the markers in the images, a custom Python script is used as opposed to using traditional libraries. These libraries often require markers with a 7 x 7 matrix, which are harder to read than 5 x 5 markers of equal size. After detection, the marker ID is determined using Figure 2c, and the Python script returns the coordinates of the center of each marker.



a b c

Figure 2. (a), (b) Examples of detectable markers consisting of a 5x5 matrix bar code with black borders. (c) Marker ID calculation. Every box except for the four corners has a unique power of two as value. The ID is calculated by adding the values of the black boxes, e.g., the ID of marker (a) is $2^0 + 2^1 = 3$.

3.4. Double homography

To map the coordinates of the driver's gaze ${}^{H}\vec{p}$ on C, creating ${}^{C}\vec{p}$, two homography matrices ${}^{C}H_{I}$ and ${}^{I}H_{H}$ are used: ${}^{C}\vec{p} = {}^{C}H_{I} \cdot {}^{I}H_{H} \cdot {}^{H}\vec{p}$. The reason for two homography matrices instead of one that directly transforms H to C is because that images are too dissimilar for feature matching to work reliably (see Fig. 3). H includes the interior of the car, and only a small part of the image is the outside environment as seen in C. The initial head image I and the current head image H are relatively similar, therefore feature matching is possible on I and H. To optimize this feature matching, I is cropped to remove unwanted scenery. Marker detection for finding corresponding points is more accurate than feature matching when images are dissimilar, but requires four markers (Fig. 2a,b) to be visible in both the cameras C_{head} and C_{car}. C_{head} has a low shutter speed resulting in motion blur and making detecting markers a challenge, especially with camera movement. Therefore marker detection is used in the first phase, followed by feature matching in the second phase.

3.5. Object and focus point recognition

When the coordinates from the driver's gaze are mapped to C_{car}, the system determines at what object type the driver is looking. Object recognition is handled by semantic scene labeling. Every pixel of the forward facing camera of the car has a color, depending on what object the pixel belongs to (e.g., car, person, bicycle), as seen in Figure 3c. Semantic scene labeling in the test car is equipped with SegNet, which is a Deep Convolutional Neural Network for pixel-wise segmentation [6]. Using the semantic scene labeling and the gaze data of the

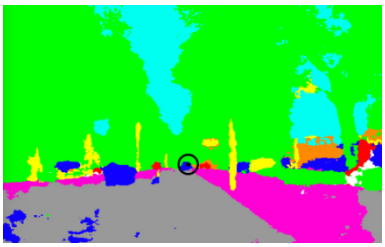
driver, the color of the corresponding pixel is extracted. Since the semantic scene labeling can be inaccurate, not only the color of the pixel itself, but the dominant color within a circle with radius 25 pixels around the gaze coordinate is returned. The detection of the dominant color is to ensure that feedback is only given if the person is truly looking at an object. If this dominant color matches a color on which feedback must be given (e.g., the color representing cars), audio feedback in the form of a voice is played, telling the driver at what kind of object he or she is looking.

4. Road testing

The system has been tested on the Toyota Prius while driving on public roads. During these tests, the driver looked at a distant car (Fig. 3a). This gaze data is plotted by the system onto C_{car} (Fig. 3b). At the same time, the semantic scene labeling interprets the whole C_{car} image, finds the car in the distance (Fig. 3b) and colors it blue. The dominating color in a radius of 25 pixels around the user's gaze (Fig. 3c) is blue, so the system knows the user is looking at a car. Phase 3 then gives audio feedback to the user by saying "Car" using synthesized voice.

a

b



c

Figure 3. A print-screen of the video feeds at the same point in time, with the gaze position plotted onto the video. Due to the latency of ROS, the images do not match exactly, but this does not affect plotting the user's gaze. (a) C_{head} image of the eye tracker, H. The markers visible in this image are used solely for feature matching. (b) C_{car} image facing forward, C. (c) The SegNet image of C. Green = trees, light-blue = sky, dark-blue = cars, pink = sidewalk, gray = road, red = person, yellow = poles, orange = buildings, and white = bicycles.

5. Comment

For research purposes, this design already provides a valuable platform. The fact that this system is designed in ROS makes it possible for other researchers to include a better eye tracker or a different car system, and integrate it into existing applications already built on the platform. If researchers can determine in real-time what object a driver is looking at, this would yield a wealth of new possible research paradigms in areas such as hazard perception research [7].

The overall system had a limited accuracy (of about 2 degrees in stationary conditions), ran at 10 Hz, and was often unreliable. A different eye tracking system could yield a reduction of misses and false positives. Furthermore, clear features have to be present inside the car to improve feature matching, and the camera brightness has to be manually adjusted to detect the markers.

Because the homography only works for points in the same plane, offsets will emerge when looking at points at different distances. For a future version, this can be countered by implementing the stereo vision cameras of the Prius, and using this data to counteract the offset. In the test setup feature matching was the limiting factor on the range of head motion. For example, the planar features on the dashboard fell outside the scope of C_{head} when looking in the side mirrors. More features spread out over the dashboard will allow a better range of operation. However, those features will likely not be planar, making homography unusable. In that case, epipolar geometry can be used instead of homography, or in combination with homography by rectification [8].

6. References

[1] A. Tawari, A. Møgelmose, S. Martin, T. B. Moeslund, M. Trivedi. Attention estimation by simultaneous analysis of viewer and view. 17th International Conference on Intelligent Transportation Systems, Qingdao, Chin, 2014.

[2] K. Conley, M. Quigley, B. Gerkey. ROS: an open-source robot operating system. Proc. Open-Source Software Workshop Int. Conf. Robotics and Automation, 2009.

[3] O. Sahin Tas, A. Hellmund, S. Wirges. Robot operating system: A modular software framework for automated driving. IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), 2016.

[4] D. Selviah, Z. Li. Comparison of image alignment algorithms. London Communications Symposium, 2011.

[5] K. Konolige, E. Rublee, V. Rabaud. ORB: an efficient alternative to SIFT or SURF. Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV), 2011.

[6] R. Cipolla, V. Badrinarayanan, A. Kendall. SEGNET: A deep convolutional encoder-decoder architecture for image segmentation. CoRR, 2015.

[7] P. Bazilinskyy, N. Heisterkamp, P. Luik, S. Klevering, A. Haddou, M. Zult, G. Dialynas, D. Dodou, J. C. F. de Winter. Eye movements while cycling in GTA V. 2018. Retrieved from https://www.researchgate.net/publication/319987337_Eye_movements_while_cycling_in_GTA_V

[8] D. Fu, L. Li, H. Zhang. Image matching based on epipolar and local homography constraints. Proceedings of SPIE - The International Society for Optical Engineering, 2007.

7. Supplementary materials

The code is open-source and available on GitHub: https://github.com/Mrdirtysocks51/pupil_ros

A demonstration movie is available here: https://www.dropbox.com/s/52igdhl9cd3fnnv/cameras-exp.mov?dl=0

Figure S1. Toyota Prius test vehicle.

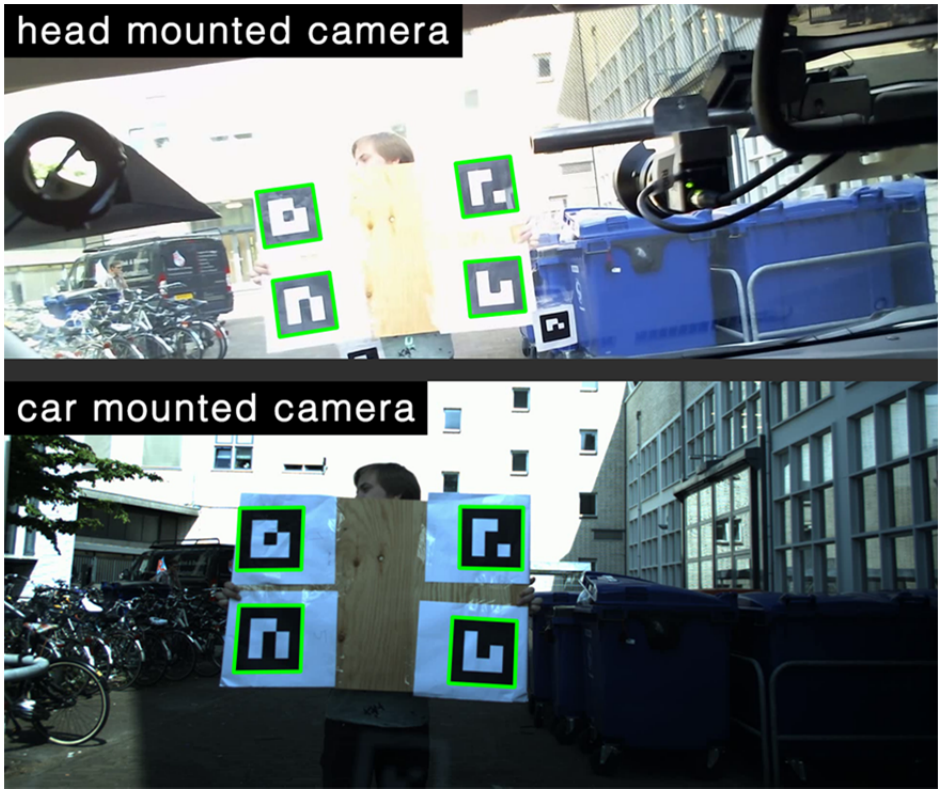


Figure S2. Marker detection in the calibration process

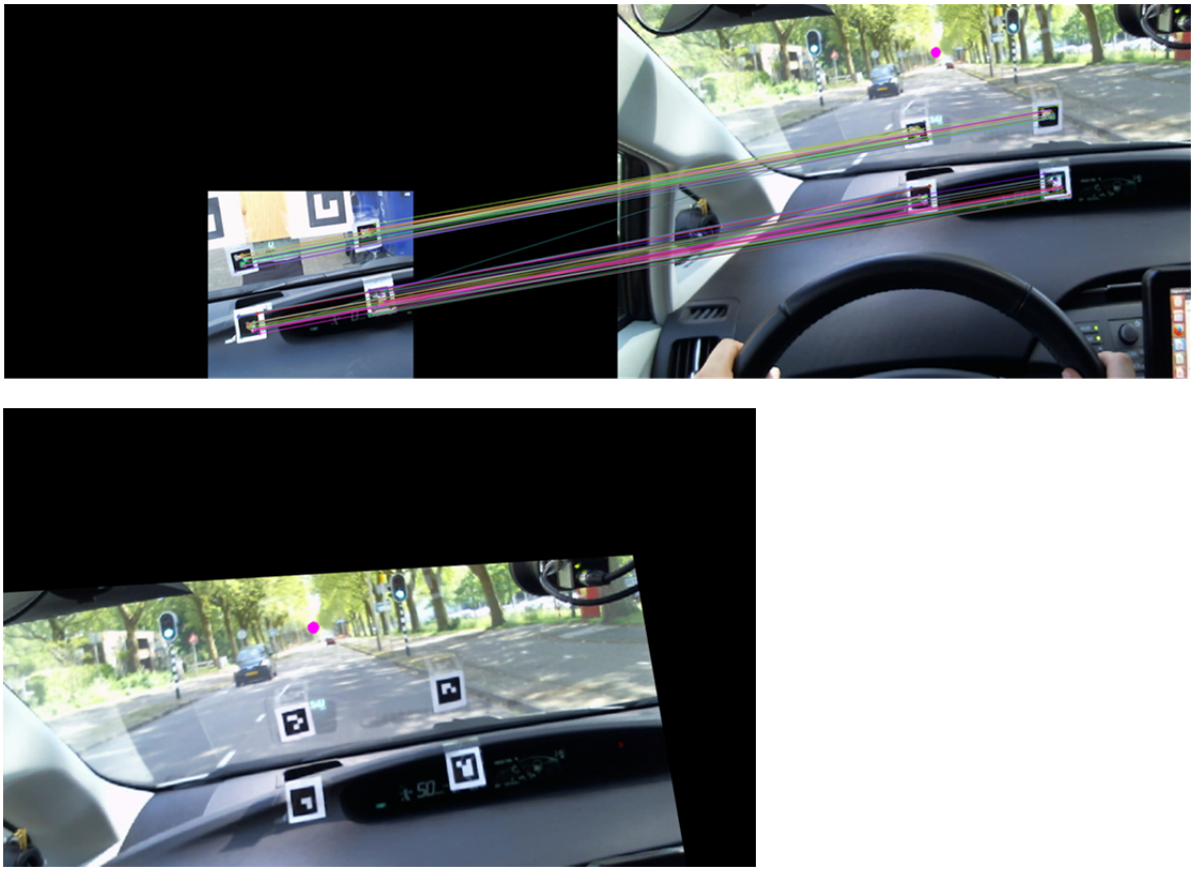


Figure S3. Feature matching. Left top = Initial head mounted image, I , Right top = Head-mounted camera image, H . Matches are shown by colored lines. Bottom = warped image calculated using the homography matrix ${}^{I}H_{H}$.