Mateusz Jarosz
Piotr Nawrocki
Leszek Placzkiewicz
Bartlomiej Sniezynski
Marcin Zielinski
Bipin Indurkhya

DETECTING GAZE DIRECTION USING ROBOT-MOUNTED AND MOBILE-DEVICE CAMERAS

Abstract *Two common channels through which humans communicate are speech and gaze. Eye gaze is an important mode of communication: it allows people to better understand each others' intentions, desires, interests, and so on. The goal of this research is to develop a framework for gaze-triggered events that can be executed on a robot and mobile devices and allows experiments to be performed. We experimentally evaluate the framework and techniques for extracting gaze direction based on a robot-mounted camera or a mobile-device camera that are implemented in the framework. We investigate the impact of light on the accuracy of gaze estimation and also how the overall accuracy depends on user eye and head movements. Our research shows that light intensity is important and the placement of a light source is crucial. All of the robot-mounted gaze-detection modules we tested were found to be similar with regard to their accuracy. The framework we developed was tested in a human-robot interaction experiment involving a job-interview scenario. The flexible structure of this scenario allowed us to test different components of the framework in varied real-world scenarios, which was very useful for progressing towards our long-term research goal of designing intuitive gaze-based interfaces for human-robot communication.*

Keywords gaze-direction detection, eye-tracking, face detection, robot, mobile device

Citation Computer Science 20(4) 2019: 453–474

1. Introduction

Social humanoid robots are becoming more and more commonplace. Their capabilities are increasing rapidly: they are equipped with a variety of mechanisms that enable them to perform human-like actions naturally and a variety of sensors to gain information about the surrounding environment (including people). Some of these robots are fully autonomous and are effectively replacing humans in many roles, such as museum tour guides, personal companions, sales assistants, and so on. For such robots, it is important to design human-robot interfaces that are intuitive for a human user. Two of the common channels through which humans communicate are speech and gaze. In this research, our long-term goal is to design intuitive gaze-based interfaces for human-robot communication.

Eye gaze is an important cue that allows people to better understand each others' intentions, desires, point of interest, and so on. Humanoid robots are often equipped with one or more cameras that allow them to capture images of human faces around them. These images can be analyzed with the help of computer vision libraries and image-processing algorithms to estimate their gaze directions. Computational resources on mobile humanoid robots are often constrained to minimize power consumption, so the available cameras can only capture low-resolution images. Special techniques and algorithms have been created to extract gaze direction from high-resolution images, a survey of which can be found in [2]. Some of these techniques can be adopted for low-resolution images.

The goal of this research is to experimentally evaluate the existing techniques for extracting gaze direction based on a robot-mounted camera or a mobile-device camera. We also propose a universal framework for robots and mobile devices for gaze-triggered events.

In the next section, we provide a survey of the existing methods for estimating gaze direction. In Section 3, we provide the details of the eye-gaze estimation techniques tested in our experiments. Section 4 describes two evaluation experiments, and our conclusions are presented in Section 5.

2. Brief survey of gaze-tracking techniques

Gaze tracking is used in many different fields such as cognitive science, psychology, advertising, and medical research, and many techniques have been developed for it. The most widely used current designs are video-based eye trackers [10]. Complex techniques such as face-detection methods, iris-detection methods, and eye-tracking methods are used in this and other related technologies. In this section, we first discuss four different approaches to eye-gaze estimation; namely, eye tracking, face detection, eye center localization, and face landmark detection. Then, we summarize methods for gaze estimation and briefly mention gaze-tracking apps available on robots and mobile devices.

2.1. Eye-tracking approaches

Eye tracking is the process of measuring the motion of an eye relative to the head. Usually, the integration of eye and head positions is used to compute gaze location in a visual scene. Simple eye trackers only report gaze direction relative to the head or for a fixed eyeball position.

Eye-tracking systems are divided into two categories: invasive and remote systems. Invasive systems require physical contact with the user: they are attached to the body. Remote systems that measure the visual activity of a person without any physical contact are called non-invasive systems. The techniques for eye tracking range from simple direct observation through invasive mechanical methods to examining the difference of the electrical potentials between the two sides of an eyeball. Below, we list four main techniques: Electro-oculography, Scleral search coils, Infrared oculography, and Video-oculography [2]. In our research, we found the last method most useful. This method, which is based on video eye tracking, is widely used in commercial eye trackers [2, 8]. Until a few years ago, eye-gaze tracking had been a very complex and expensive process and was limited to only laboratory research. However, rapid technological advancements (increased processor speed and advanced digital video processing) have lowered the cost and increased the accuracy of eye-gaze trackers.

Video oculography may be invasive or non-invasive, and each category can be further split into two other categories depending on whether visible or infrared light is used. Non-invasive (remote) systems are especially useful for HCI and HRI tasks. Most video-based eye trackers take advantage of infrared light, which produces a glint on the cornea (corneal reflection). When the eye or the head moves, the pupil-glint distance remains constant. The location of the glint not only changes when the head moves but also when the gaze direction changes. One example of a system using a single camera and a single source of infrared light is the LC Technologies tracker[1]. Video-oculography makes use of single or multiple cameras to determine eye movement using information obtained from the captured images. Due to the limited field of view of one camera, the measurement may be lost when the angle between the camera and the eye is too great. In such situations, multiple cameras are needed.

2.2. Eye center localization

A comparison of eight state-of-the-art algorithms for pupil detection, tested on low-resolution eye images recorded in remote tracking scenarios can be found [4]. The evaluation was performed on three data sets containing images with varying illumination along with different camera distances, occlusions, head movements, and off-axial camera positions. The compared algorithms included those for head-mounted as well as remote eye-trackers. Algorithms for head-mounted eye-trackers usually work with good quality images of the eye region, as the camera zooms directly on the user's

[1]LC Technologies website https://www.eyegaze.com (Accessed: 13.06.2019)

eyes. On the other hand, algorithms for remote eye-trackers are designed to work on low-quality images, as the cameras are situated at some distance from the user and the eye region needs to be extracted. Additionally, algorithms for remote eye-trackers need to deal with off-axis camera positions or partly visible eyes. From the perspective of our research, three algorithms for remote eye-tracking are of special interest: one proposed by George and Routray, another proposed by Droege and Paulus, and a third by Timm and Barth [3, 7, 13]. Among these three algorithms, the approach by Timm and Barth achieved the best results, achieving stable detection rates on all three data sets [4].

Surprisingly, the ElSe algorithm (which was designed for head-mounted systems) outperformed all of the others on the pupil-detection task on a remote eye-tracker data set [5]. ElSe provides two algorithms to estimate the pupil center. The first algorithm uses a Canny filtered image and morphological operations to detect the pupil-related edges. It selects the best edge by various heuristics, like the shape and intensity of the surrounding region. This edge is used to carry out ellipse fitting, which yields the pupil center and contour. In case the first algorithm is not successful, an advanced blob-detection algorithm is applied to find the pupil center.

Timm and Barth's algorithm takes advantage of the direction of the pixel gradients as a feature to compute the pupil center [13]. This approach is based on the idea that the direction of vector from the pupil center to any pupil or iris contour point is equal to the direction of gradient at the pupil or iris contour point only if center point is actually center of the iris (see Fig. 1). The center of the pupil is computed as follows: for each pixel in the input image, the algorithm accumulates the square product of the normalized displacement vector of the pupil-center candidate to the gradient pixel and normalized pixel gradient vector, running over the entire gradient image. The accumulated sum is multiplied by the inverted intensity of the pupil-center candidate. The pupil-center candidate having the highest accumulated sum is selected as the pupil center. We use this algorithm in our research to detect pupil centers.

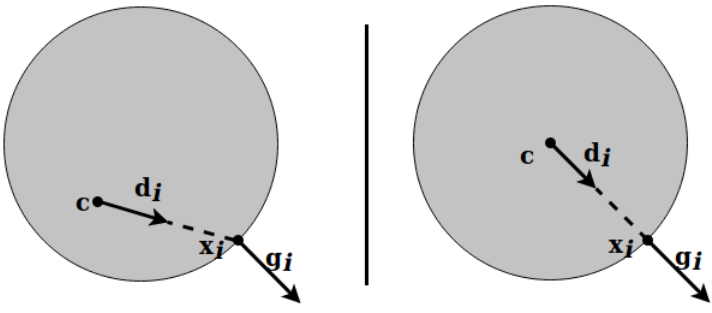

Figure 1. Grey circle represents iris, and light background – sclera. Left: orientation of displacement vector d_i and gradient vector g_i differs; right: they are equal

Roberto Valenti and Theo Gevers proposed another approach using Isophotes, which are contours visible in the image [14]. This method requires a contrast between objects. The limited part of the image containing the circle at which we are looking is extracted. Next, new coordinates are set such that each pixel points towards the highest intensity drop (like the gradient). Finally, we come to the following equation (Equation (1)).

$$D(x,y) = \frac{\{L_x, L_y\}(L_x^2 + L_y^2)}{L_y^2 L_{xx} - 2L_x L_{xy} Ly + L_x^2 L_{yy}} \tag{1}$$

Here, L_x, L_y are the first derivatives of the brightness functions, and $D(x,y)$ is the vector pointing towards the estimated center. Many such vectors are calculated; their intersection is the pupil center.

2.3. Face landmark detection

Kazemi, Vahid, and Sullivan Josephine [9] proposed a method to quickly locate 68 face landmarks (see Fig. 2).

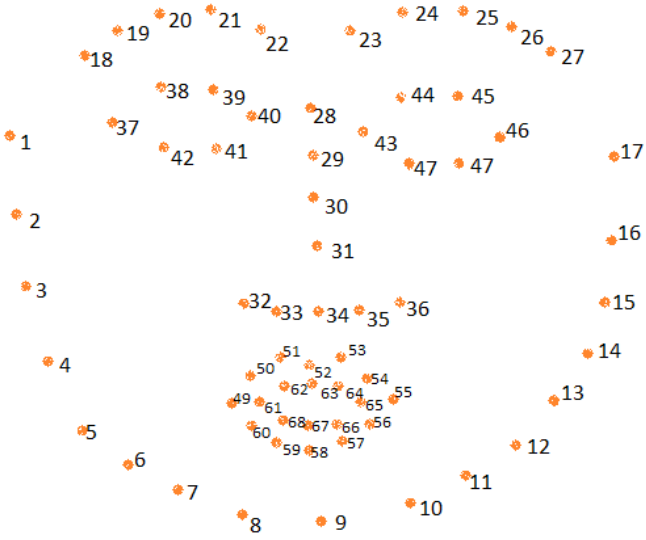

Figure 2. 68 face landmarks [11]

This method is based on a cascade of the regression function. Pixel indexing is also used in terms of their brightness in relation to the currently predicted shape. Individual regressors are trained by the gradient-boosting method and then combined into larger cascades.

Another method that realizes the same functionality has been proposed by T. Baltrusaitis, P. Robinson, and L.-P. Morency [1]. It uses Constrained Local

Neural Field (CLNF), which is an evolution of Constrained Local Models (CLM). The method consists of preparing a separate model for each point sought; then, each pixel in a given area of interest is evaluated to see if it is the point sought.

In the last phase, the pixel with the best value is chosen. This approach can be implemented in a classical way using the random forest method. The authors of the article, however, use Non-Uniform Regularised Landmark Mean-Shift (NU-RLMS), which yields a more reliable result.

2.4. Techniques for gaze estimation

Gaze-tracking approaches can be split into two categories [2]:

Feature-based methods use extracted local features, such as contours (limbus and pupil contour), eye corners, and cornea reflections from an eye image. The goal of these methods is to find informative local features of the eye that are less sensitive to variations in viewpoint and illumination. When accurate iris and pupil features are not available, the performance of these methods degrades.

Their performance is also affected when working outdoors or under strong ambient light.

Feature-based approaches can be classified into two categories: model-based (geometric) and interpolation-based (regression-based) approaches [2]:

Model-based approaches take advantage of an explicit geometric model of the eye to compute a 3D gaze direction vector. Because these 3D model-based techniques rely on metric information, they require camera calibration and a global geometric model of the light sources as well as the camera and monitor positions and orientations. The vast majority of model-based methods consist of the following steps:

- First, the optical axis of the eye is reconstructed in 3D. This is achieved by estimating the cornea and pupil centers.
- Second, the visual axis is reconstructed.
- Finally, the visual axis is intersected with the scene geometry to obtain the point of gaze.

For 3D model-based methods, gaze direction is estimated as a vector from the eyeball center to the iris center.

Interpolation-based approaches assume that the image features can be mapped to the gaze coordinates using a parametric (e.g., polynomial) or non-parametric form (e.g., neural networks). These techniques represent point of gaze as a generic function of the image features instead of explicitly modeling the geometry of the human eye. To calculate the unknown coefficients of the mapping function, calibration data is used for numerical fitting with multiple linear regression or some other similar method. Neural network-based approaches (which are an alternatives to parametric expressions) assume a non-parametric form to compute a mapping from the image features to the gaze coordinates. In these approaches, the coordinates of certain facial points are extracted and sent through a trained neural network. The coordinates of the point of gaze are obtained as the output.

Appearance-based methods use the image content to estimate gaze direction by mapping the data to screen coordinates. These methods do not usually require the calibration of cameras or geometry data because the mapping is made directly on the image contents. For example, Wood *et al.* used this method to estimate gaze direction [15]. They compared their solution to the geometric solution proposed by Wood and Bulling and found the two to be comparable [16]. In this method, a ray is carried from the center of the eye (through the pupil) to the camera. This method is also used to determine facial landmarks in both 2D and 3D. The models have been trained on a synthetic data set that allows one to control the eye position, head position, and lighting; they were able to recognize 28 characteristic points in the eye area.

2.5. Gaze-tracking apps on robot and mobile devices

Eye Gaze is an Android app available from the Google Play Store; this allows a user to stabilize camera shakes (for small displacements) and track gaze direction after detecting the user's facial features. The authors do not say much about the algorithms used in this work – they only mention that they are using OpenCV. OpenCV can also be used to detect faces and facial features (such as the eyes) in the newer version for detecting the landmarks on a face (as discussed in Section 2.3).

In order for this system to work, the user must face the device. Furthermore, eyeglasses or the presence of any extraneous element degrades the app's performance. The best results are obtained under medium illumination: too much or too little illumination affects the system's performance. Eye extremity is the most prominent feature affecting the accuracy of gaze detection. There have been many attempts to obtain the best results and document them with a screen-shot, but the best picture is presented in Figure 3.

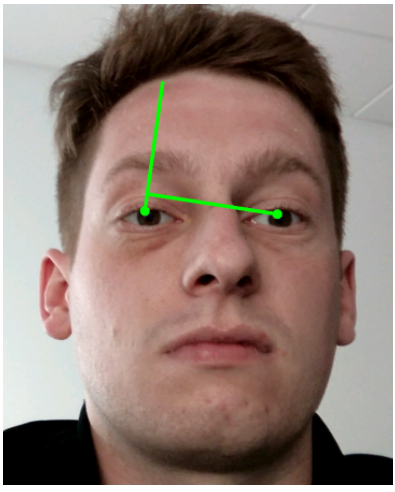

Figure 3. Eye Gaze App – action example

As can be seen in this screenshot, the gaze indicators are quite random. The main advantage of this system is that the indicator beginnings are always exactly in the middle of the eye. A major disadvantage is that, if there is too much or too little lighting (or if the user moves away from the camera), the indicators begin to move in increasingly random directions.

There are also other applications like EyeTab, which performs gaze tracking on mobile devices; however, they are usually fine-tuned for specific devices and do not work properly with others [16]. EyeTab is based on Timm and Barth's algorithm as well as the isophotoes algorithm for accurately finding the pupil (as described in Section 2.2). An ellipse is then fitted into the pupil, and the normal vector is calculated; at this point, the vector can be transformed into gaze angle.

There are many articles describing the approaches to gaze tracking for a robot, but most of them do not provide any working samples that can be evaluated [7, 8, 17, 18]. Some commercial solutions are available, but they are based on commercial eye trackers like Tobii or Pupil Labs. One available solution is for the Pepper robot[2], which is used as a reference point in this study.

3. Framework for gaze-triggered events

We would like to use gaze tracking in human-robot interactions and perform experiments on mobile and robot platforms. Therefore, we have designed and implemented a universal framework for gaze-triggered events. Figure 4 shows its high-level architecture; it can be executed on a Pepper robot and mobile devices using the Android system. The architecture includes both modules available on the Pepper robot as well as those that have been developed by others. The modules are described below.

Pepper video source: uses robot's ALVideoDevice module to retrieve images from robot's camera.

Robot Gaze: uses robot's ALGazeAnalysis module to fetch gaze-related data.

Webcam video source: uses OpenCV functionality to retrieve images from laptop webcam.

Android video source: uses OpenCV functionality to retrieve images from Android device camera.

Custom Gaze: has the following functions:

- uses VideoSource to retrieve images from either robot's or laptop's camera;
- uses self-developed gaze direction-estimating solution to obtain gaze-related data;
- takes advantage of OpenCV and dlib library functionality.

[2]Pepper robot – https://www.softbankrobotics.com/emea/en/pepper.

OpenFace solution: performs the following functions:

- uses VideoSource to retrieve images from either robot's or laptop's camera;
- uses OpenFace gaze direction-estimating solution to obtain gaze-related data;
- takes advantage of OpenFace and OpenCV library functionality.

Android solution: carries out the following functions:

- uses Android video source to retrieve images from Android device's camera;
- uses self-developed gaze direction-estimating solution to obtain gaze-related data;
- takes advantage of OpenCV and dlib library functionality.

Analyzer: makes use of Gaze component functionality to fetch gaze-related data.

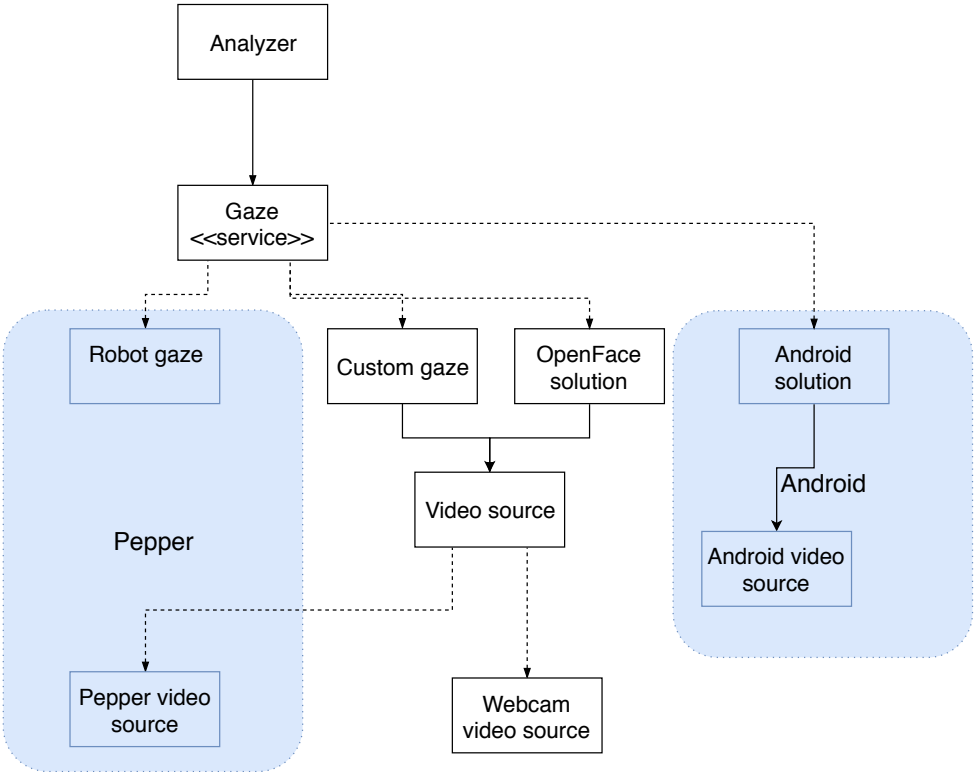

Figure 4. Architecture diagram of system

To determine gaze direction, there are many approaches that can be used on a mobile platform (such as the Pepper robot or on an Android device). However, most of these approaches are based on the same general Algorithm 1 shown below.

Algorithm 1 Gaze direction-estimation algorithm

procedure GETGAZEDIRECTION(*image*)
  $grayImg \leftarrow convertToGrayscale(image)$
  $faces \leftarrow detectFaces(grayImg)$
  $faceLandmarks \leftarrow detectFaceLandmarks(faces)$
  $eyeRegions \leftarrow detectEyeRegions(faceLandmarks)$
  $headPose \leftarrow computeHeadPose(faceLandmarks)$
  for each $eyeRegion \in eyeRegions$ **do**
    $pupilCenter \leftarrow findPupilCenter(eyeRegion)$
    $gaze \leftarrow calculateGaze(pupilCenter, faceLandmarks)$
    $eyeGazeDirection \leftarrow transformToAngles(gaze + headPose)$ ▷ gaze vector needs to be combined with headPose to get vector in camera coordinates
  $gazeDirection \leftarrow combine(eyeGazeDirection)$ ▷ e.g., Averaged
  return *gazeDirection*

3.1. Gaze detection on Pepper robot

Working with the Pepper robot imposes some restrictions on using commonly available libraries as well as on the image resolution and frame rate; for example, when working remotely, the resolution is 1280x960 pixels with one frame per second (FPS) or 640*480 pixels with up to 30 FPS. Because of this restriction, we chose to work with VGA resolution. A better resolution can be obtained when working directly on the robot; however, installation of any library is problematic and can disable the Pepper software. For this reason, we chose to use remote operation.

We explored a few solutions to gaze tracking, but many of them did not pass the early test phase. For example, the solution using ellipse fitting in the sclera was rejected due to the unstable results: in about half of the trials with facial images, it failed to find the eyes. Finally, we selected the two solutions described below.

OpenFace-based solution: This uses the general algorithm (see Algorithm 1) with some adjustments. The OpenFace library provides almost all of the needed functions, such as head-pose estimation, landmark detection, gaze estimation, and so on [1, 15]. OpenFace is written in C++. As a part of this work, Python bindings were created. Also, many possible methods for transforming the gaze vectors to gaze angles were investigated during the tests; the most suitable method was chosen (Equations (2) and (3)).

$$x\_angle = atan2(gazeVectos[x], -gazeVector[y]) \tag{2}$$

$$y\_angle = asin(cos(-gazeVectos[y]/\sqrt{gazeVectos[x]^2 + gazeVector[z]^2}) \tag{3}$$

A key difference with the general algorithm is that the pupil detection is integrated in the landmark detection. OpenFace returns landmarks in 2D and 3D, which makes it much easier to calculate the head pose and gaze direction.

CustomGaze: It also follows the general algorithm (see Algorithm 1) with some adjustments. The algorithm for gaze-direction estimation is based on the Eye-gaze

application[3]. The pupil center is detected with the Timm algorithm [13] described in Section 2.2. The eye-gaze direction vector is computed separately for each eye by taking the difference between the pupil center (as detected by Timm's algorithm) and the eye center. The eye center is computed as the middle point between the outermost and innermost eye-related facial landmarks. The head pose-estimation algorithm employed in this work uses the facial model, which takes advantage of the locations of the eyes, the nose, and the mouth features [6]. These features are not much influenced by facial expressions and vary little across users in comparison to other features (like hairlines or facial contours) [12]. The face model is composed of three distances: L_f , L_n, and L_m, which represent the eye/mouth, nose/mouth, and nose base/nose tip distances, respectively. In this model, the face is assumed to be a plane passing through the locations of the eyes and the mouth features. The facial normal is found by joining the nose base to the nose tip. The symmetry axis for the face is found by joining the midpoint between the eyes to the mouth center. In the camera-centered coordinates, facial normal vector n (which determines the head position) is computed as follows:

$$n = [sin(\sigma)cos(\tau), -sin(\sigma)sin(\tau), -cos(\sigma)] \tag{4}$$

The σ angle is called *slant,* which is the angle that the facial normal makes with vector d normal to the image plane. The θ and τ angles can be extracted from a 2D image. The comprehensive derivation of Formula 4 and supporting theory can be found in [6]. The locations of the eyes, the nose, and the mouth features as well as the distances between each pairing are easily computed from the face landmarks found beforehand. The eye-gaze directions for each eye and the head-pose vector are combined to compute the gaze direction using the following formula:

$$gaze\_vec\_left = 13.101 * \frac{left\_eye\_width}{14.0} * head\_pose\_vec * left\_eye\_gaze\_vec \tag{5}$$

$$gaze\_vec\_right = 13.101 * \frac{right\_eye\_width}{14.0} * head\_pose\_vec * 3 * right\_eye\_gaze\_vec \tag{6}$$

As a reference solution, Pepper-provided libraries were chosen. Unfortunately, the Pepper library's internal work is undocumented; through the tests, however, we found that it uses an infra-red depth sensor along with visible light camera to calculate the head pose.

3.2. Gaze detection on an Android

Gaze detection on an Android platform is performed using two modules in parallel. One module is responsible for facial landmark detection, and the other one performs the gaze estimation. The gaze angles are estimated based on the eye-corner positions obtained by the first module and the iris position calculated by the gaze-estimation

[3]Eye-gaze application repository – https://github.com/iitmcvg/eye-gaze.

module. Head-pose estimation was irrelevant; it is assumed that the user is facing the camera or the phone screen directly. Gaze angles are calculated accordingly using the following equations (Equations (7) and (8)):

$$\begin{aligned} ESF &= irisToLeft/eyeWidth \\ horizontalAngle &= -(MSA * EEF) + (WA * EEF * ESF) \end{aligned} \tag{7}$$

$$\begin{aligned} EHF &= irisToBottom/eyeHeight \\ verticalAngle &= -(MVA * EEF) + (WA * EEF * EHF) \end{aligned} \tag{8}$$

where:

$irisToLeft$**:** distance from left-eye corner to iris,

$eyeWidth$**:** distance between eye corners,

MSA**:** max side angle; equal to 90,

EEF**:** eye edge factor; constant chosen experimentally equal to 0.77,

WA**:** whole angle; equal to 180,

MVA**:** max vertical angle; equal to 90,

$irisToBottom$**:** distance from bottom of eye to iris,

$eyeHeight$**:** distance between bottom and top of eye.

These calculations are made for each eye and then combined to yield the final gaze angle.

4. Evaluation experiments

In this section, we describe two experiments to evaluate three systems: RobotGaze, OpenFace, and CustomGaze. The first experiment was designed to evaluate the influence of room lighting on the correctness of the gaze estimates. The second experiment was aimed at evaluating the accuracy of the real-world solutions. We first present the experimental setting, followed by the experimental procedure; we then present the results and discussion.

4.1. Experimental setting

The test environment consisted of the following components (Fig. 5):

- Pepper robot was placed in front of interlocutor,
- interlocutor interacting with Pepper robot sat in front of it,
- notebook computer on which system was running to control robot,
- direction markers were placed behind robot,
- experiments were performed in room with good lighting conditions,
- robot interacted with one person only,
- distance between interlocutor and robot was between 100 cm and 120 cm,
- interlocutor's head was positioned on same level as robot's head (where camera was).

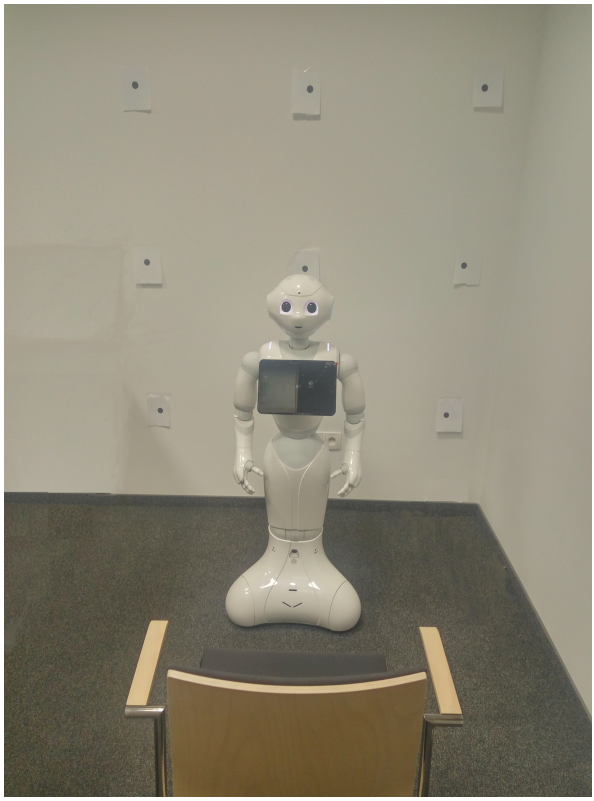

Figure 5. Test environment – robot and direction markers

The central direction marker was located at the level of the robot's head. The distance between the direction markers was 65 cm horizontally and vertically. In this configuration (when the interlocutor looks at the direction markers on the sides, on the top, or on the bottom), the yaw and pitch angles are approximately 20 degrees each. The gaze regions were indicated by the direction markers.

4.2. Experiment 1: Studying effect of luminosity

The system was started in a debug-like mode; the following lighting configurations were tested:

- Full L.: Full lighting (about 1010 lux) available in room.
- Half L.: Half lighting (about 350 lux) available in room.
- No L.: (about 1 lux) without lighting.
- Extra L. F.: Additional lighting (about 230 lux) located in front of interlocutor (15 cm below camera).
- Extra L. LS.: Additional lighting (about 230 lux) located on left side of interlocutor (at his head height).

During the test, the interlocutor was moving both the head and the eyes to focus on the direction markers. The experiment was repeated ten times; all of the shown values are the averages of their results.

Table 1 shows the percentage of correctly fitted areas of interest during each stage of the experiment. RobotGaze is the reference solution using the software available on the robot. The second item in the list is a solution based on the OpenFace library.

Table 1
Results and discussion

	Full L. [%]	Half L. [%]	No L. [%]	Extra L. F. [%]	Extra L. LS. [%]
RobotGaze	66	66	60	55	75
OpenFace solution	68	25	0	60	60
CustomGaze	55	55	27	66	66

The best results for RobotGaze were obtained for Extra L. LS. This is likely due to the better illumination of the eyes as compared to being lit from above (when there is a shadow on the eye cast by the eyebrow). Confirming this hypothesis requires further research. The lack of a similar result in Extra L. F. is most likely caused by the user getting dazzled with the light and not being able to look in a given direction (or squinting involuntarily). Another possible reason may be due to pupil contraction when exposed to light. In Full L., there was a problem with matching the lower three vertical areas, though the horizontal alignment was correct. In conditions Half L. and No L. performance of the system is still good. However, reducing the amount of light significantly hinders finding the face in the image which is necessary to start the experiment.

The best results for CustomGaze were obtained in Extra L. F. and Extra L. LS. This is likely due to the better illumination of the eye. It can be observed that, without lighting, CustomGaze was only occasionally able to classify gaze direction correctly, and only when the user focused on the middle-row markers.

The best results for the OpenFace solution were obtained in Full L. In Half L. and No L., the system performed poorly due to the deteriorating lighting conditions. In No L., the zero accuracy was caused by a lack of face detection during the experiment (which gave an incorrect fit). Extra L. F. and Extra L. LS. gave slightly worse results as compared to Full L. Extra L. LS. gave a similar result to Extra L. F., but the face was not detected in any case.

The results for Full L. were similar in all versions of the system; however, Half L. and No L. gave significantly worse results when using the OpenFace and CustomGaze solutions. The reason for this considerably poor performance when compared to RobotGaze is likely because RobotGaze uses depth maps in addition to the camera image.

Under the external lighting condition, there were different reasons as to why OpenFace performed worse than CustomGaze: the experiments were performed on different days, one of which was rainy and cloudy, and the other was sunny. We were able to block most of the light from the outside (but not all of it). In Extra L. F. and Extra L. LS., we did not observe any significant improvement in the result for the system using OpenFace when compared to what happened in the other tested solutions (CustomGaze and RobotGaze).

A probable cause is that the lighting makes it difficult to detect the face when one side is illuminated but the other is in a shadow. This confirms the hypothesis that the system using the functions offered by the robot software does not only use the camera image. The CustomGaze solution in Extra L. F. and Extra L. LS. performed better than in Full L.

Illumination measurements for the Android solution were carried out under four conditions: normal, light, dark, and very dark. As in the next experiment, the time measurement was performed 15 times for each condition. The measurement was taken from the start of the video frame to when eye contact was made. All of the individual measurements are included in the chart in Figure 6.

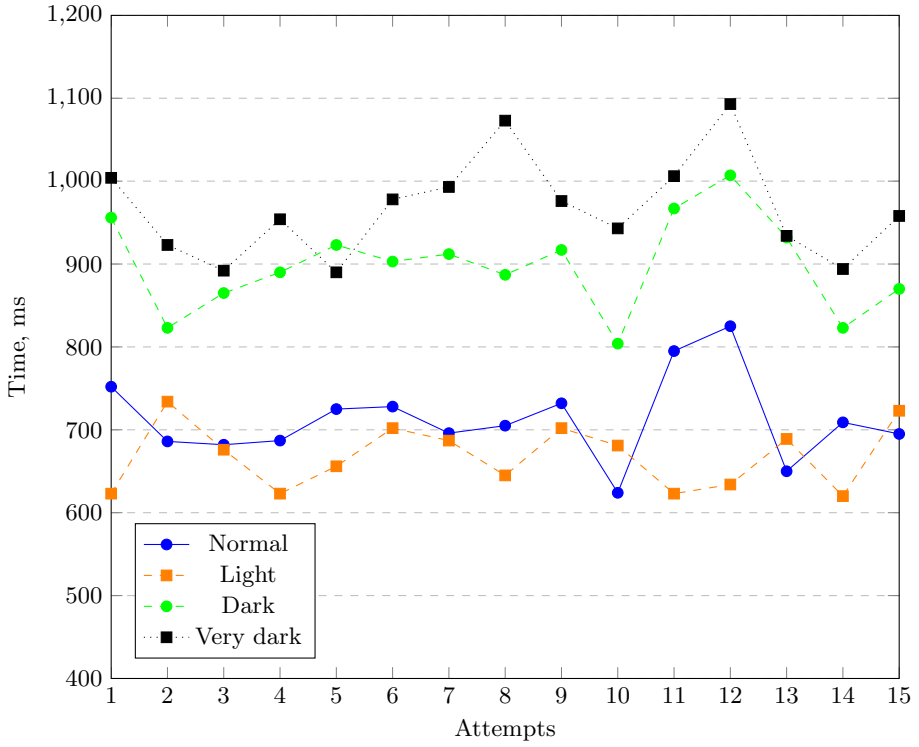

Figure 6. Influence of lighting comparison

The combined results from measuring the influence of light on the speed and accuracy of detection are presented in Table 2.

Table 2
Android influence of lighting – times

	Normal [ms]	Light [ms]	Dark [ms]	Very dark [ms]
Average	712	669	899	960
Standard deviation	49	35	54	62

As expected, the best results were achieved in a well-lit place and the worst in a very dark room. The lowest standard deviation was recorded for a very bright case. In other cases, the deviation was comparable. Significant differences in the accuracy of detection were not observed outside a very dark room, where the accuracy was sometimes very poor.

4.3. Experiment 2: Accuracy of tested solutions

In this experiment, the system was started in the debug mode, where the robot measures only the interlocutor's gaze direction. Every three seconds, the robot records the direction (one out of nine regions) in which the interlocutor has been gazing for the last three seconds. The interlocutor's gaze was fixed within the same analysis window but changed between subsequent windows. The interlocutor was asked to focus his or her gaze on the direction markers placed behind the robot. Each test included focusing the gaze on each direction marker. The system was configured to estimate the gaze direction every 500 milliseconds; so, within a three-second window, there were six measurements. The system used yaw and pitch angles to classify the gaze into one of nine regions. The final gaze direction region is the most frequently detected region within the analysis window.

The system classified gaze as horizontal middle if the yaw angle fell within a range of ± 15 degrees (as shown in Table 3). If the yaw angle exceeded ± 15 degrees, the gaze was classified as right or left, respectively. Similarly, the gaze was categorized as vertical middle if the pitch angle fell within a range of ± 15 degrees. If the pitch angle was outside the ± 15-degree range, the gaze was classified as up or down, correspondingly.

Table 3
Area of interest division

horizontal $90^\circ - 15^\circ$ vertical $90^\circ - 15^\circ$	horizontal $15^\circ - (-15^\circ)$ vertical $90^\circ - 15^\circ$	horizontal $-15^\circ - (-90^\circ)$ vertical $90^\circ - 15^\circ$
horizontal $90^\circ - 15^\circ$ vertical $15^\circ - (-15^\circ)$	horizontal $15^\circ - (-15^\circ)$ vertical $15^\circ - (-15^\circ)$	horizontal $-15^\circ - (-90^\circ)$ vertical $15^\circ - (-15^\circ)$
horizontal $90^\circ - 15^\circ$ vertical $-15^\circ - (-90^\circ)$	horizontal $15^\circ - (-15^\circ)$ vertical $-15^\circ - (-90^\circ)$	horizontal $-15^\circ - (-90^\circ)$ vertical $-15^\circ - (-90^\circ)$

The goal of the test was to evaluate the accuracy of the algorithms for gaze estimation while the interlocutor made head and eye movements. The interlocutor could fix his or her gaze on a direction marker using any of the following three ways:

- moving both head and eyeballs (head + eyes);
- moving only eyeballs but not head (eyes);
- moving head only with eyeballs fixed on robot camera (on its head)[4].

The results of this experiment (averaged over all of the participants) are presented in Table 4.

Table 4

Accuracy of classifying gaze into one of nine gaze regions depending on head and eye movements

Solution name	Head + eyes [%]	Eyes [%]	Head [%]
RobotGaze	66	22	55
OpenFace	68	22	60
CustomGaze	55	33	49

As seen in Table 4, the best results were obtained when using both the eyes and the head to look at a marker. Each tested solution achieved a higher accuracy when the gaze was focused on a direction marker by moving both the head and the eyes rather moving the eyes alone. In all of these cases, the classifications were mostly accurate when the interlocutor was focusing his or her gaze on a direction marker from the upper or middle rows. Moving only the eyes often resulted in the gaze being classified as if the interlocutor were looking at the robot. The only exceptions were some sporadic correct classifications when the interlocutor focused on a direction marker from the middle or upper rows.

We also found that accurate gaze estimation results mostly depend on the estimation of head orientation; only about one third of the results depend on eye-gaze estimation 4. An analysis of the yaw and pitch angles shows that, despite an incorrect classification of gaze into the gaze regions, the algorithms were able to compute the correct yaw and pitch angles (which differed in the different gaze regions). This suggests that, by manipulating the thresholds for determining the classification into particular regions, the system would be able to correctly classify the gaze direction.

The Android solution was also tested to determine its gaze-detection accuracy. The values returned by the application were compared with the ground truth. The tests were carried out first for the horizontal axis and then for the vertical axis. The tests started at -60 degrees, moving in increments of 15 degrees until reaching 60 degrees, thereby giving a set of 9 measurements. The gaze measurement results are presented in Table 5.

[4]In this case, if the eye is fixed on the robot camera, merely moving the head should not change the detected gaze direction.

Table 5
Android calculated angles corresponded to real angles

Horizontal angles			Vertical angles		
Expected	Calculated	Difference	Expected	Calculated	Difference
−60	−71	11	−60	−88	28
−45	−42	3	−45	−33	12
−30	−22	8	−30	−23	7
−15	−9	6	−15	12	27
0	1	1	0	−3	3
15	−4	19	15	6	9
30	25	5	30	28	2
45	58	13	45	28	17
60	83	23	60	41	19
	Average	10		Average	14
	StDev	7		StDev	9

The accuracy of the horizontal measurements is slightly better: both the average error and deviation are smaller. Figure 7 presents the results in a graph; from this, it can be seen that, the more a person looks sideways horizontally or up-and-down vertically, the greater is the error.

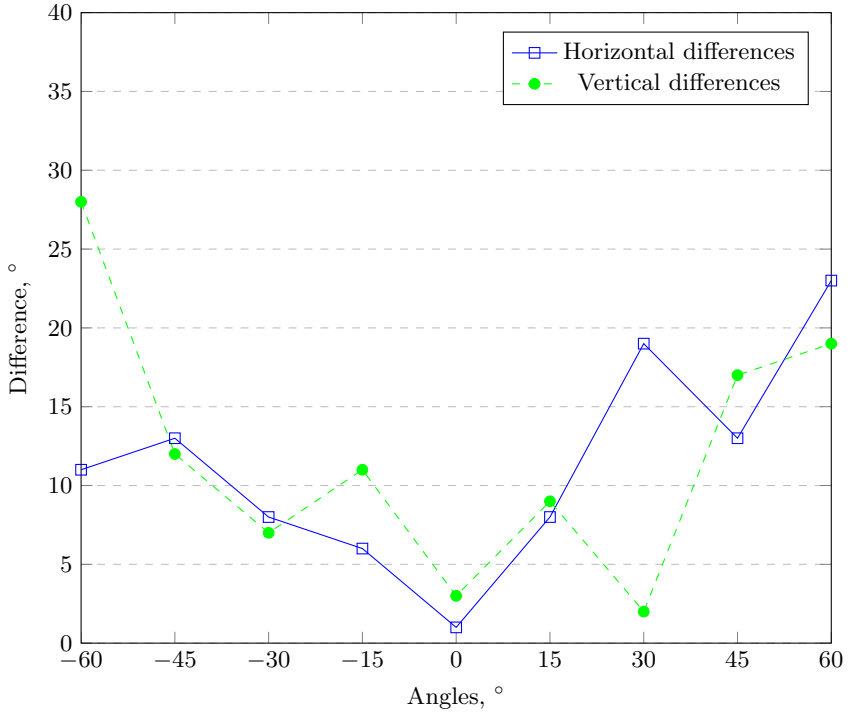

Figure 7. Angle test results

The least error occurs when a person looks straight ahead. After 30 attempts, very accurate results were recorded in about 40% of the cases, satisfactory results in about 30% of the cases, and incorrect values in 30% of the cases.

4.4. Experiment 3: Human-robot interaction in job interview scenario

The goal of this experiment was to field test the developed framework for gaze-triggered events in a real-world human-robot interaction. We used a job interview scenario to evaluate the performance of the system behavior as a proof of concept of our platform. During the tests, the system was estimating gaze direction, taking advantage of the RobotGaze, CustomGaze, and OpenFace solution modules.

The system executed a job interview scenario provided in the JSON format. All of the steps of the scenario were correctly performed by the robot, and the performance of all tested solutions were in agreement with the results obtained in the previous experiment. The robot asked a question in each step of the scenario and waited for a specified amount of time for the interlocutor's response. The actions performed by the robot were dependent on the interlocutor's gaze (which was analyzed during the response time). In this scenario, the robot was only speaking in the response (there was no gestures). In the future, we plan to enhance the quality of the human-robot interaction by incorporating other types of actions such as gesturing and posturing by moving parts of the robot's body.

5. Conclusions

In this research, we analyzed various approaches for estimating gaze direction to determine which algorithms are suitable for implementation. Next, gaze-based systems that facilitate human-robot interaction were designed and developed for the Pepper robot and an Android device. These systems incorporate the OpenFace solution and the ALGazeAnalysis module (which is available on the Pepper robot) together with algorithms developed by us.

A series of experiments were performed to measure the system's performance. The gaze-estimating solutions were evaluated under different lighting conditions and with different combinations of head and eye movements. The results of the experiments show that the tested systems achieve a comparable accuracy. It was observed that head movements provide a more informative cue to deduce gaze direction than eye movements do. We also found that providing additional light sources at the sides of the interlocutor can significantly improve the vertical gaze estimation.

Due to the high cost of commercial gaze-tracking solutions, a few open-source software-based gaze trackers have been developed that do not need any special hardware and make use of off-the-shelf equipment. However, the accuracy of these software-based gaze trackers is not as good as the commercial gaze trackers. More research needs to be done on these open-source gaze trackers to improve their accuracy and stability.

Acknowledgements

The research presented in this paper was supported by the National Center for Research and Development (NCBR) under Grant No. POLTUR2/5/2018.

References

[1] Baltrušaitis T., Robinson P., Morency L.P.: Constrained Local Neural Fields for Robust Facial Landmark Detection in the Wild. In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 354–361. 2013.

[2] Chennamma H., Yuan X.: A survey on eye-gaze tracking techniques, *Indian Journal of Computer Science and Engineering*, vol. 4(5), pp. 388–393, 2013.

[3] Droege D., Paulus D.: Pupil center detection in low resolution images. In: *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications*, pp. 169–172, ACM, 2010.

[4] Fuhl W., Geisler D., Santini T., Rosenstiel W., Kasneci E.: Evaluation of state-of-the-art pupil detection algorithms on remote eye images. In: *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pp. 1716–1725, ACM, 2016.

[5] Fuhl W., Santini T.C., Kübler T., Kasneci E.: ElSe: Ellipse Selection for Robust Pupil Detection in Real-World Environments. In: *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications*, pp. 123–130. ACM, 2016.

[6] Gee A., Cipolla R.: Determining the gaze of faces in images, *Image and Vision Computing*, vol. 12(10), pp. 639–647, 1994.

[7] George A., Routray A.: Fast and Accurate Algorithm for Eye Localisation for Gaze Tracking in Low-Resolution Images, *IET Computer Vision*, vol. 10(7), pp. 660–669, 2016.

[8] Kar A., Corcoran P.: A Review and Analysis of Eye-Gaze Estimation Systems, Algorithms and Performance Evaluation Methods in Consumer Platforms, *IEEE Access*, vol. 5, pp. 16495–16519, 2017.

[9] Kazemi V., Sullivan J.: One Millisecond Face Alignment with an Ensemble of Regression Trees. In: *27th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, United States, 23–28 June 2014*, pp. 1867–1874, IEEE Computer Society, 2014.

[10] Mohamed A.O., Silva da M.P., Courboulay V.: *A history of eye gaze tracking*, Rapport Interne, 2007. https://hal.archives-ouvertes.fr/hal-00215967.

[11] Sagonas C., Antonakos E., Tzimiropoulos G., Zafeiriou S., Pantic M.: 300 faces In-the-Wild Challenge: database and results, *Image and Vision Computing*, vol. 47, pp. 3–18, 2016.

[12] Sapienza M., Camilleri K.P.: *Fasthpe: a recipe for quick head pose estimation*, Systems and Control Engineering, Department of Systems and Control Engineering, University of Malta, Msida, Malta, 2011.

[13] Timm F., Barth E.: Accurate Eye Centre Localisation by Means of Gradients, *Visapp*, vol. 11, pp. 125–130, 2011.

[14] Valenti R., Gevers T.: Accurate Eye Center Location and Tracking Using Isophote Curvature. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2008*, pp. 1–8, IEEE, 2008.

[15] Wood E., Baltrušaitis T., Zhang X., Sugano Y., Robinson P., Bulling A.: Rendering of Eyes for Eye-Shape Registration and Gaze Estimation. In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3756–3764, 2015.

[16] Wood E., Bulling A.: EyeTab: Model-based gaze estimation on unmodified tablet computers. In: *Proceedings of the Symposium on Eye Tracking Research and Applications*, pp. 207–210, ACM, 2014.

[17] Yamazoe H., Utsumi A., Yonezawa T., Abe S.: Remote Gaze Estimation with a Single Camera Based on Facial-Feature Tracking without Special Calibration Actions. In: *Proceedings of the 2008 Symposium on Eye Tracking Research & Applications*, pp. 245–250, ACM, 2008.

[18] Zhang X., Sugano Y., Fritz M., Bulling A.: Appearance-Based Gaze Estimation in the Wild. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4511–4520. 2015.

Affiliations

Mateusz Jarosz
AGH University of Science and Technology, Faculty of Computer Science, Electronics, and Telecommunications, Department of Computer Science, al. A. Mickiewicza 30, 30-059 Krakow, Poland, majarosz.j50@gmail.com,
ORCID ID: https://orcid.org/0000-0002-9829-0127

Piotr Nawrocki
AGH University of Science and Technology, Faculty of Computer Science, Electronics, and Telecommunications, Department of Computer Science, al. A. Mickiewicza 30, 30-059 Krakow, Poland, piotr.nawrocki@agh.edu.pl,
ORCID ID: https://orcid.org/0000-0003-4512-9337

Leszek Placzkiewicz
AGH University of Science and Technology, Faculty of Computer Science, Electronics, and Telecommunications, Department of Computer Science, al. A. Mickiewicza 30, 30-059 Krakow, Poland, leszekplaczkiewicz@gmail.com

Bartlomiej Sniezynski
AGH University of Science and Technology, Faculty of Computer Science, Electronics, and Telecommunications, Department of Computer Science, al. A. Mickiewicza 30, 30-059 Krakow, Poland, bartlomiej.sniezynski@agh.edu.pl,
ORCID ID: https://orcid.org/0000-0002-4206-9052

Marcin Zielinski
AGH University of Science and Technology, Faculty of Computer Science, Electronics, and Telecommunications, Department of Computer Science, al. A. Mickiewicza 30, 30-059 Krakow, Poland, kmarcinzielinski@gmail.com

Bipin Indurkhya
AGH University of Science and Technology, Faculty of Computer Science, Electronics, and Telecommunications, Department of Computer Science, al. A. Mickiewicza 30, 30-059 Krakow, Poland, bipin@agh.edu.pl,
ORCID ID: https://orcid.org/0000-0002-3798-9209

Received: 21.09.2019
Revised: 11.10.2019
Accepted: 11.10.2019