Protractor3D: A Closed-Form Solution to Rotation-Invariant 3D Gestures

Sven Kratz
Deutsche Telekom Laboratories, TU Berlin
Ernst-Reuter-Pl. 7, 10587 Berlin, Germany
sven.kratz@tu-berlin.de

Michael Rohs
Deutsche Telekom Laboratories, TU Berlin
Ernst-Reuter-Pl. 7, 10587 Berlin, Germany
michael.rohs@telekom.de

ABSTRACT

Protractor 3D is a gesture recognizer that extends the 2D touch screen gesture recognizer Protractor [8] to 3D gestures. It inherits many of Protractor's desirable properties, such as high recognition rate, low computational and low memory requirements, ease of implementation, ease of customization, and low number of required training samples. Protractor 3D is based on a closed-form solution to finding the optimal rotation angle between two gesture traces involving quaternions. It uses a nearest neighbor approach to classify input gestures. It is thus well-suited for application in resource-constrained mobile devices. We present the design of the algorithm and a study that evaluated its performance.

Author Keywords

Gesture recognition, gesture-based interaction, template matching, rotation invariance, nearest neighbor approach

ACM Classification Keywords

H.5.2 Information Interfaces and Presentation: User interfaces; I5.2 Pattern recognition: Design methodology—*Classifier design and evaluation*

INTRODUCTION AND RELATED WORK

Acceleration sensors are currently in wide use in mobile devices. Such sensors allow applications to use information about the device's movement. This information can not only be used to set the correct screen orientation but, more importantly, it allows the implementation of novel user interfaces for gaming, AR reality browsing (e.g. Layar), device-to-device interaction [1, 3, 12] or even authentication (by moving two devices in the same way [11] or by using motion-based gestures, which are gestures that users enter by moving the mobile device with their arm [4]). Motion gesture recognition can be implemented using advanced machine learning techniques such as Support Vector Machines (SVM) [10] or Parzen Windows Classifiers (PWC) [2]. However, as a growing body of recent work shows, ad-hoc or simpler classification techniques that are mostly data-driven are growing in popularity. On of the reasons for this is that adapting state-of-the art machine learning techniques can be cumbersome. Most of them require special libraries or entail a high implementation effort, due to the high mathematical and algorithmic complexities of the approaches used. For instance, while the theoretical concepts of SVMs should be graspable to most developers, training SVMs requires a Quadratic Programming (QP) [10] solver, which is non-trivial to implement, and is in most cases supplied by specialized third-party toolkits. However, optimizing such advanced classifiers so that they achieve their best classification performance is likely to be beyond the grasp of a standard mobile application developer, as this requires detailed knowledge of the workings of a particular classifier and also a great deal of experience in machine learning. A number of recent publications have focused on implementation simplicity, for instance [7], an extension of Wobbrock's $1 Gesture Recognizer [14] for use with 3D acceleration data, or work by which extensively use a data-driven approach based on Dynamic Time Warping (DTW) [9], a simple but effective pattern matching technique. Both of these simple gesture recognition techniques use a template matching strategy. This means that gesture recognition is usually performed on a per-user basis, by matching the user's input data with templates that were previously entered by her. Other research has focused on slightly more complex, but still basic, machine learning techniques, e.g. [4]. This previous body of work shows that the simple techniques usually perform relatively well, with average correct recognition rates above 90%.

In this work, we present a contribution that aims to significantly improve the computational requirements and the correct recognition rate of data-driven motion gesture recognizers. A major problem of existing recognition techniques for motion gestures is that the gestures cannot be recognized in a rotation-invariant way. For instance, a symbolic gesture, such as drawing a circular shape in the air may be performed either in the vertical or horizontal plane or simply using a non-standard grip on the mobile device. Such rotated input data cannot be matched accurately with training templates that were entered using a different device posture. In the following, we refer to this problem as the *template–gesture rotation problem*. Protractor [8], an extension to Wobbrock's $1 gesture recognizer, addressed this problem in 2D by extending the original algorithm with a closed-form solution to finding the optimal rotation between template and entered gesture.

IUI'11, February 13–16, 2011, Palo Alto, California, USA.

In the following, we present *Protractor3D*, a gesture recognition algorithm based on the closed-form solution to the absolute orientation problem for measurements in two 3D coordinate systems [5]. This technique solves the template–gesture rotation problem for motion gesture recognition. An evaluation we conducted shows that, using actual input generated by test subjects, Protractor3D significantly increases gesture recognition accuracy in comparison to an implementation that does not apply rotation correction when matching entered gesture data to gesture templates.

SOLUTION TO THE OPTIMAL GESTURE – TEMPLATE ROTATION PROBLEM

The solution to the absolute orientation problem for points measured in two different Cartesian coordinate systems [5] can be applied directly to solving the gesture–template rotation problem. In the following we will only discuss Horn's technique in sufficient detail to implement a rotation-invariant recognition algorithm for motion gestures. For the relevant mathematical derivations and correctness proof we refer the reader back to Horn's original work.

The Template Rotation Problem Let C be a gesture class represented by a set of template gestures. Each template is a sequence of 3D accelerometer values. We consider a gesture $\mathbf{g}$ entered by the user, again a sequence of 3D accelerometer values. Ideally, this gesture is entered in a way similar to a template gesture $\mathbf{t}$ of gesture class C. However, it is very likely that the posture with which the user enters $\mathbf{g}$ is different from the one with which the user entered the template $\mathbf{t}$. This can be due to variations of the user's grip on the mobile device or also differences in the user's own body posture, such as being seated or standing, while entering $\mathbf{g}$. For gestural interfaces, it is in many cases undesirable to constrain the device posture for gesture entry, as for many applications the device's movement ought to determine the type of gesture entered, not the device's posture. To solve the gesture–template rotation problem, we must find a rotation R that minimizes the sum of squares error between $\mathbf{g}$ and $\mathbf{t}$.

$$\sum_{i=1}^{n} \|\mathbf{t}_i - R(\mathbf{g}_i)\|^2$$

between the points of $\mathbf{t}$ and the rotated points of $\mathbf{g}$. Minimizing this sum of squares is equivalent to maximizing the scalar (or dot) product of $\mathbf{t}$ and $R(\mathbf{g})$ [5].

$$\sum_{i=1}^{n} R(\mathbf{g_i})^T \cdot \mathbf{t_i}$$

FINDING THE OPTIMAL ROTATION USING QUATERNIONS

Horn uses a technique based on Quaternions to determine the optimal rotation R. Using the compound product $\mathring{q}\mathbf{g}\mathring{q}^*$, that represents $R(\mathbf{g})$ utilizing Quaternions, the maximization of the above equation can be reformulated as follows:

$$\sum_{i=1}^{n} (\mathring{q}\mathbf{g_i}\mathring{q}^*)^T \cdot \mathbf{t_i}$$

We assume here that $\mathbf{g}$ and $\mathbf{t}$ have been centered around the origin of the coordinate system, i.e. $\mathbf{g}$ and $\mathbf{t}$ are differences of the original point sequences and their respective centroids $\overline{\mathbf{g}}$ and $\overline{\mathbf{t}}$. This means that the centroid of both $\mathbf{g}$ and $\mathbf{t}$ is the null vector. Horn shows that this is equivalent to finding the unit Quaternion that maximizes

$$\mathring{q}^T N \mathring{q}$$

N is a matrix that is derived from the matrix sum of the products of $\mathbf{t}$ and $\mathbf{g}$:

$$M = \sum_{i=1}^{n} \mathbf{t_i} \cdot \mathbf{g_i}^T$$

M, defined by its elements can be written as

$$M = \begin{pmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{pmatrix}$$

where $S_{xx} = \sum_{i=1}^{n} \mathbf{g}_{x,i}\mathbf{t}_{x,i}$, $S_{xy} = \sum_{i=1}^{n} \mathbf{g}_{x,i}\mathbf{t}_{y,i}$ and so forth.

The Matrix N can then be constructed from the elements of M, so that:

$$N = \begin{bmatrix} S_{xx}+S_{yy}+S_{zz} & S_{yz}-S_{zy} & S_{zx}-S_{xz} & S_{xy}-S_{yx} \\ S_{yz}-S_{zy} & S_{xx}-S_{yy}-S_{zz} & S_{xy}-S_{yx} & S_{zx}+S_{xz} \\ S_{zx}-S_{xz} & S_{xy}-+S_{yx} & -S_{xx}+S_{yy}-S_{zz} & S_{yz}+S_{zy} \\ S_{xy}-S_{yx} & S_{zx}+S_{xz} & S_{yz}+S_{zy} & -S_{xx}-S_{yy}+S_{zz} \end{bmatrix}$$

Matrix Product is Maximized through Eigenvector

$\mathring{q}^T N \mathring{q}$ is maximized by finding the eigenvector $\mathring{e}_m$ corresponding to the largest positive eigenvalue of N. By normalizing $\mathring{e}_m$, we obtain the quaternion $\mathring{q} = w + iq_x + jq_y + kq_z$, which encodes the optimal rotation angle $\theta = 2\cos^{-1}(w)$ and the unit representation of the corresponding rotation axis $(q_x, q_y, q_z)^T$. The optimal rotation matrix R is thus obtained from θ and the imaginary parts of $\mathring{q}$ as follows.

Let $\mathbf{s} = \sin(-\theta), \mathbf{c} = \cos(-\theta),$ then

$$R = \begin{bmatrix} q_x^2(1-q_x^2)\mathbf{c} & q_x q_y(1-\mathbf{c})-q_z\mathbf{s} & q_x q_z(1-\mathbf{c})+q_y\mathbf{s} \\ q_x q_y(1-\mathbf{c})+q_z\mathbf{s} & q_y^2(1-q_y^2)\mathbf{c} & q_y q_z(1-\mathbf{c})-q_x\mathbf{c} \\ q_x q_z(1-\mathbf{c}) & q_y q_z(1-\mathbf{c})+q_x\mathbf{s} & q_z^2+(1-q_z^2)\mathbf{c} \end{bmatrix}$$

PROTRACTOR3D GESTURE CLASSIFIER

Using the optimal solution to the gesture – template rotation problem, we can now formulate the Protractor3D Gesture Classification algorithm.

Input and Output We define $C = \bigcup C_i,\ i \in \mathbb{N}$ as the set of all gesture classes (or gesture library), i.e. the different gesture movements that Protractor3D is trained to detect. Each C_i contains a number of training gestures $t_{C_i,k},\ k \in \mathbb{N}$. For a given input gesture g, Protractor3D will find the the template $t_{C_x,y} \in C$ with the lowest Euclidean distance, corrected for rotation, with respect to g.

Subsampling, Scaling, Centering In order for gestures and templates to be comparable, we subsample every input gesture to consist of a fixed number of points. We chose $n = 32$

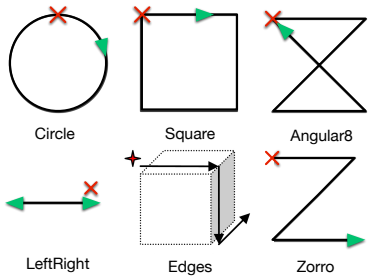

Figure 1. This set of iconic gestures is a subset of the gestures we used in our study.

as the number of subsampled points per gesture, as a higher n did not have a noticeable effect on the gesture recognition rate. We followed the same subsampling strategy as Wobbrock et al. [14], adapted to work with 3D data. After subsampling, the input gesture is scaled to fit into a cube of a fixed side length, in our case $s = 100$. Finally, to be able to perform optimal alignment of the gesture data, we calculate the centroid of the input gesture and subtract it from all gesture points, thus centering the gesture. When creating gesture templates for the gesture library, user inputs are subsampled, scaled and centered in the same way before they are added to the gesture library.

Gesture Recognition In the main part of the algorithm the gesture input, transformed as described in the previous section, is compared to every template in the gesture library, using rotational correction, to minimize the effects of device posture. The algorithm uses the Euclidean distance as the metric to compare the distance between the template and the input gesture. An advantage of Protractor3D is that it provides the angle of absolute rotation θ between an input and the template during comparison steps. Using θ helps Protractor3D decide if the template should be rejected as a possible recognition candidate. This does not need to be applied if complete rotation-invariance is desired. In our implementation, we have set the cutoff angle θ_{cut} to $\pm 45°$. If all gestures in the gesture library have been rejected due to θ being greater than θ_{cut}, Protractor3D recognizes the input gesture as "unrecognized." Otherwise, Protractor3D reports the gesture class of the template with the lowest Euclidean distance with respect to the input gesture as the recognized gesture class.

EVALUATION

To evaluate the recognition performance of Protractor3D, we recorded gestures from ten paid test subjects. This approximate number of subjects has been also been used in previous studies on motion gestures [9, 13]. The gesture set consisted of 11 gestures, six of which consisted of iconic gestures, partially used in previous work by [13]. The other half of the gestures was derived from the requirements of a related study on mobile gestures conducted in our group [6], which includes a user-defined gesture (*own*), everyday movements (*shakehand, shakearm*) and also some usual movements performed with mobile phones ("take out of *pocket*", "take out of *handbag*"). Each test subject recorded at least 40 repetitions of every gesture in the gesture set. A SHAKE SK6 sensor package was used to capture the data at a rate of 100Hz. The accuracy and sampling rate of the SHAKE SK6 acceleration sensor is comparable to those integrated into modern smartphones. To delimit individual gesture entries, the users were required to press and hold the navigation button of the SHAKE SK6 during gesture entry, and to release the button upon termination of the gesture entry. To determine the benefit of rotational correction that is delivered by Protractor3D, we measured the *Correct Recognition Rate* (CRR) of Protractor3D with rotational correction, as well as Protractor3D without rotational correction, which in the following we refer to as MSE (mean square error).

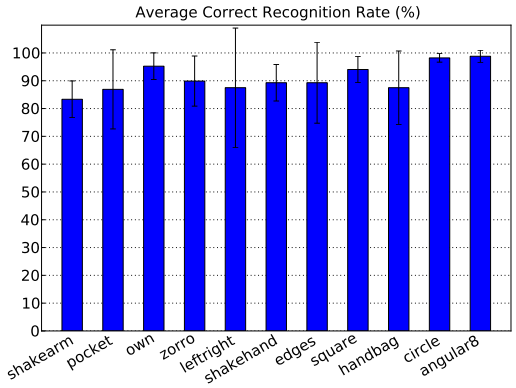

Figure 2. The average correct recognition (CRR) rates by gesture for Protractor3D, with a training set size of 5. The error bars show the 95% confidence intervals of the gesture CRR means.

Choice of Validation and Test Set In order to evaluate the performance of Protractor3D, we defined *training* and *validation* sets for each of the gesture types on a per-user basis. We chose a 40:60 split between training and validation sets. When constructing the training and validation sets, we took only the first 40 gesture entries per gesture class, and discarded the rest (if present). Thus, for each gesture class the training set consisted of the first 16 gestures, and the remaining 24 were used at the validation set. As a consequence, we define CRR as the number of correctly recognized gestures divided by the size of the validation set. To "train" Protractor3D, we used the last five gestures of the training set for each gesture, for each user. We chose this low number of training samples to demonstrate the ability of Protractor3D to produce relatively robust gesture recognition results with only a few training samples. This property is advantageous for users seeking to rapidly add new gestures to their mobile device or developers who wish to rapidly prototype new gestures in a mobile user interface they are creating.

Recognition Results Figure 2 shows the recognition results using Protractor3D with a training set size of 5 per gesture class. "Angular 8" was the gesture with the highest correct recognition rate of 98.9%, whereas "shakearm" has the lowest correct recognition rate of 83.3%. The reason for these differences in gesture recognition rate is unclear, but it could

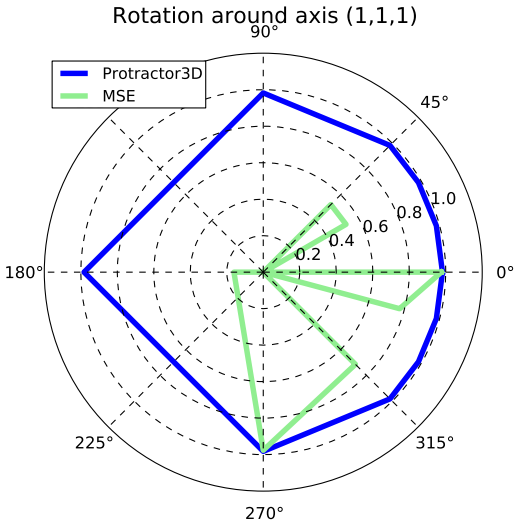

Figure 3. Polar plot of influence of rotation on the correct recognition rate. Protractor3D has a constant CRR under all rotations and thus is rotation-invariant, whereas MSE-only is highly susceptible to changes in the rotation of the input gestures, and in some cases falls back to zero recognized gestures.

be that the movements corresponding to the gestures with a lower correct recognition rate may have been more ambiguous to the test subjects than the gestures which achieved a higher gesture recognition rate. This ambiguity is also reflected, for instance, in the relatively large standard error in the average recognition of *leftright*, where we did not specify the exact way this movement was to be performed.

Effect of Rotational Correction To verify if Protractor3D has a noticeable correction effect on rotated input data, we repeated the evaluation of the algorithm with pre-rotated input gestures and compared the CRR of Protractor3D and (unrotated) mean square error (MSE) matching. We chose $(1, 1, 1)^T$ as the rotation axis and pre-rotated the input gestures by 0, 15, 30, 45, 90 and 180 Degrees (positive and negative). As can be seen in Figure 3, MSE is highly unstable under rotation, yielding very poor recognition results. In contrast, Protractor3D yields the same CRR for each rotation angle, thus showing that Protractor3D is rotation-invariant and truly finds the optimal rotation compensation for any given rotation of the input points.

CONCLUSION AND FUTURE WORK

We presented a lightweight recognizer for motion-based 3D gestures that solves the problem of finding the optimal rotation between an input gesture and a template gesture by employing a closed-form solution. This solution to the template–gesture rotation problem is a major step over exhaustive search, since the latter requires finding the optimum in a three-dimensional search space. A brute force approach of checking all possible rotations for all templates in the gesture library is thus out of the question, in particular for resource-constrained mobile devices. Our contribution is the application of a closed-form solution to finding the optimal rotation based on a quaternion representation originally proposed by Horn [5]. This approach leads to an efficient and accurate gesture recognizer that can be implemented easily. Protractor 3D requires only a low number of training gestures and thus imposes a low overhead in design and prototyping phases of gesture-based interfaces, in which designers (or later also users) wish to come up with their own custom gestures. The design and personalization of gesture-based interfaces thus becomes more approachable by designers as well as users. Remaining challenges include gesture segmentation and detecting the initiation of a gesture in the stream of sensor data. However, this is a general problem that needs to be solved by all gesture recognition systems that do not rely on a separate clutching mechanism.

REFERENCES

1. R. Dachselt and R. Buchholz. Natural throw and tilt interaction between mobile phones and distant displays. In *CHI EA '09: Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*, pages 3253–3258, New York, NY, USA, 2009. ACM.
2. R. DUDA, P. HART, and D. STORK. *Pattern Classification*. Wiley, 2000.
3. K. Hinckley. Synchronous gestures for multiple persons and computers. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 149–158, New York, NY, USA, 2003. ACM.
4. M. Hoffman, P. Varcholik, and J. LaViola Jr. Breaking the Status Quo: Improving 3D Gesture Recognition with Spatially Convenient Input Devices. In *Proc. of IEEE VR 2010*, 2010.
5. B. K. Horn, H. M. Hilden, and S. Negahdaripour. Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America*, 5(7):1127–1135, 1988.
6. N. Kischnick, S. Kratz, and S. Moeller. An improved approach to gesture-based authentication for mobile devices (poster). In *Symposium On Usable Privacy and Security (SOUPS) 2010*, 2010.
7. S. Kratz and M. Rohs. A $3 gesture recognizer: simple gesture recognition for devices equipped with 3d acceleration sensors. In *IUI '10: Proceeding of the 14th international conference on Intelligent user interfaces*, pages 341–344, New York, NY, USA, 2010. ACM.
8. Y. Li. Protractor: a fast and accurate gesture recognizer. In *CHI '10: Proceedings of the 28th international conference on Human factors in computing systems*, pages 2169–2172, New York, NY, USA, 2010. ACM.
9. J. Liu, L. Zhong, J. Wickramasuriya, and V. Vasudevan. uWave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*, 5(6):657–675, 2009.
10. S. Marsland. *Machine Learning: An Algorithmic Approach*. Chapman & Hall / CRC, 2009.
11. R. Mayrhofer and H. Gellersen. Shake well before use: Intuitive and secure pairing of mobile devices. *IEEE Transactions on Mobile Computing*, 8(6):792–806, 2009.
12. T. Pering, Y. Anokwa, and R. Want. Gesture connect: facilitating tangible interaction with a flick of the wrist. In *TEI '07: Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 259–262, New York, NY, USA, 2007. ACM.
13. T. Schloemer, B. Poppinga, N. Henze, and S. Boll. Gesture recognition with a wii controller. In *Proc. TEI '08*, pages 11–14. ACM, 2008.
14. J. O. Wobbrock, A. D. Wilson, and Y. Li. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proc. UIST '07*, pages 159–168, New York, NY, USA, 2007. ACM.