Towards Self-Adaptive User Interfaces by Holding Posture Recognition for Smartphones

Rene Hörschinger, Marc Kurz, Erik Sonnleitner
Department for Smart and Interconnected Living (SAIL)
University of Applied Sciences Upper Austria
4232 Hagenberg, Austria
email:{firstname.lastname}@fh-hagenberg.at

***Abstract*—People interact with their smartphones in many different ways depending on the phone size, the application itself, or simply whether the user is left or right-handed. From the developer side of view, it can be beneficial to know how the user operates the phone in order to adjust and adapt the user interface accordingly. Therefore, this paper proposes a model that can predict the holding posture of a smartphone at runtime. The model gets the smartphone IMU data on each interaction with the display as well as the location where the user pressed onto the display as input and outputs the most likely holding posture. By using each interaction individually, the model is able to predict which hand is holding the phone at runtime with an accuracy of 89.3%.**

***Index Terms*—posture recognition, self-adaptiveness, smartphone sensing**

I. Introduction

People interact with their mobile devices in many different ways. This makes designing user interfaces challenging as the interface will only suit one type of group while others may struggle because of the way they hold their smartphone. If the smartphone could be aware of how the user currently interacts with the device, user interfaces and the general handling of the mobile device could autonomously adapt to this specific situation enabling self-adaptiveness of the smartphone on a real-time basis.

Steven Hoober conducted a survey in 2013 by observing 1,333 people in public places (streets, airports, bus stops, cafes, trains, busses) and analyzing how they interact with their smartphones [1]. People nowadays use their mobile devices in many different scenarios for example while walking, riding a bus, or standing still, and therefore they adapt their grasp on the smartphone accordingly. After his observation Hoober found out that 49% of the people hold their phone in one hand, 36% hold it cradled (one hand holding the phone, one hand interacting with the touchscreen either with the thumb or the index finger), and 15% use it two-handed. These are the numbers of people actively interacting with the mobile device, while we see in Figure 1 the distribution of all interactions including 22% accounting for voice calls and 18.9% listening to music or watching videos.

Looking at the one-handed interactions, interestingly only 67% of the people use their right hand while 33% use their left hand which does not correlate with the number of left-handed people in the general population of about 10% [2].

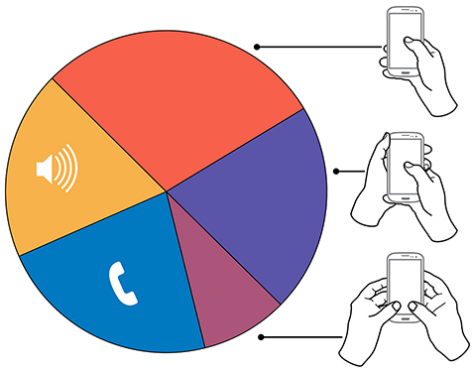

Fig. 1. General distribution of active and passive smartphone interactions [1].

This observation shows that right-handed people are likely to operate their mobile devices from time to time to free up their dominant hand for other tasks. Taking this into consideration, UI designs should not only be focused on right-handed people as nearly 1/3 of the one-handed interactions are done by the left hand.

After studying the one-handed interactions, it can also be seen that there are two ways of holding a phone with one hand:

- Thumb on the display, all 4 other fingers on the side
- Thumb on the display, pinkie on the bottom, the other 3 fingers on the side

Those differences result in different screen coverage, as with the pinkie on the bottom it is harder to reach the top left corner, while the other holding posture makes it tough to reach the bottom right corner which we can see in Figure 2.

Looking at the two-handed interactions with both thumbs it can be seen that people only use this method for gaming, video consumption, and keyboard typing.

Smartphones nowadays tend to get bigger and bigger:

- iPhone 5s, 2013, 4 inch display [3]
- iPhone X, 2017, 5.8 inch display [4]
- iPhone 13 Pro Max, 2022, 6.7 inch display [5]

That makes one-handed interactions with the mobile device harder. Apple for example offers a one-handed mode with a swipe down from the bottom of the display to shift the whole UI downwards. Otherwise, the classic return button on the top

 ISBN: 978-1-68558-047-6

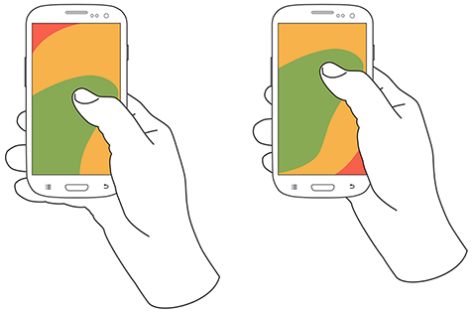

Fig. 2. Thumb coverage of one-handed interaction with the right thumb but different holding postures [1].

left of the screen would be nearly impossible to reach for a user with an average size hand on an iPhone 13 Pro Max. But this practice needs an active interaction of the user which is sub-optimal.

In this paper, we propose a way to detect how the user interacts with the smartphone at runtime in the background so the UI designer can take advantage of that information when designing the interface. A practical example would be the zoom operation in a maps application: To zoom in or out the standard way is to swipe two fingers apart or together which tends to be quite complicated with a big smartphone operated with only one hand. The zoom in that case could be adjusted to work with a long press on the touchscreen and then swiping up- and downwards while still pressing on the screen to zoom in and out. That interaction can easily be done with one hand. Additionally, knowing how the user currently interacts with his/her smartphone, would open possibilities in terms of self-adapting the UI mode of the device in an automated way.

To achieve this recognition, a machine learning (ML) model is trained with the sensor data from the inertial measurement unit (IMU) of a smartphone. The dataset used for the training was recorded with different participants of different sex and age while interacting with the smartphone.

The rest of the paper is structured as follows: Section II gives an overview of related work. Section III describes the methodology applied in our work. Section IV provides the evaluation containing (i) feature engineering, and (ii) model selection. Section V summarizes the achieved results. The final Section VI concludes the paper with a discussion and an outlook on future work.

II. Related work

Detecting the holding position of a mobile device is an active field of study with many different approaches. Wimmer et al. [6] utilize capacitive sensors in their prototype ”HandSense” to determine which hand is holding the device with an accuracy of 80% and 6 different holding positions. Another approach from Hinckley et al. [7] takes advantage of a self-capacitance touchscreen to detect the grip on the smartphone (one-handed and two-handed) as well as to recognize multiple fingers hovering over the touchscreen.

Goel et al. [8] present the prototype ”GripSense” for detecting holding patterns by combining built-in inertial sensors as well as the vibration motor of the mobile device. After a sequence of touchscreen interactions (tapping, swiping) the model reaches 84.4% accuracy at detecting left-, right- and two-handed interactions. Goel et al. utilize ”GripSense” for their next model ”ContextType” [9] and add additional posture-specific touch pattern information to detect the hand posture after each tap with an accuracy of 89.7%.

Park et al. [10] not only use gyroscope and accelerometer data for their Support Vector Machine (SVM) but also touchscreen data like the coordinates of interaction and the size. After training and testing on 6 participants, they reach an accuracy of 87.7% and 92.4% for 5 and 4 hand postures respectively. Löchtefeld et al. [11] have a similar approach to Park et al. but they also include the device orientation and focus on the detection of the holding posture during device unlocking. With a k-nearest neighbor model and Dynamic Time Warping (DTW), they achieve 98.51% accuracy. A comparable model to Löchtefeld et al. comes from Avery et al. [12]. They detect the holding position of the smartphone prior to the first touchscreen interaction with the use of the built-in orientation sensors and DTW with an accuracy of 83.6%.

Summing up these approaches, some of them need external sensors like Wimmer et al., some of them only focus on grabbing the phone and the unlocking process like Löchtefeld et al. and Park et al., and others detect the holding posture during actual user interaction like ”GripSense” from Park et al. but require a certain amount of interactions in order to work properly.

In contrast to the approaches mentioned above, the model presented in this paper does not need any external sensors, does not focus only on the unlocking process of a smartphone, and furthermore detects holding posture changes during the usage of an application at runtime.

III. Approach

Nowadays a lot of mobile applications do not support landscape mode, especially social media apps, for example, Instagram, Twitter, Snapchat, and Facebook. With that in mind and with the information retrieved from the survey of Steven Hoober in Section I we decided to differentiate between 4 different holding postures, seen in Figure 3: Right single-handed, left single-handed, cradle with the left index finger, cradle with the right index finger.

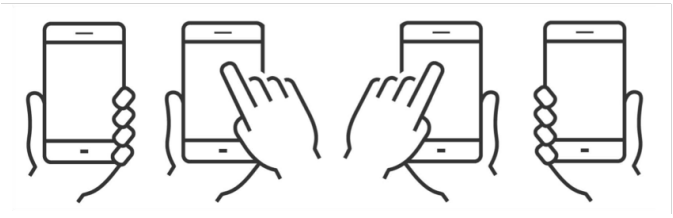

Fig. 3. Different holding postures of a smartphone [12].

 ISBN: 978-1-68558-047-6

The idea is to gather IMU data before and after user interaction with the touchscreen and identify the holding posture only by that event. IMU data is available on every new smartphone which means the model is platform-independent and can be used on all iOS and Android devices.

A. Data recording

The setup for the data recording was as follows: 9 participants (7 male, 2 female) between 23 and 58 were asked to perform multiple tap actions with different holding postures within an iOS app on predefined locations which were: left, middle and right each at the top, middle, and bottom of the touchscreen, as seen in Figure 4.

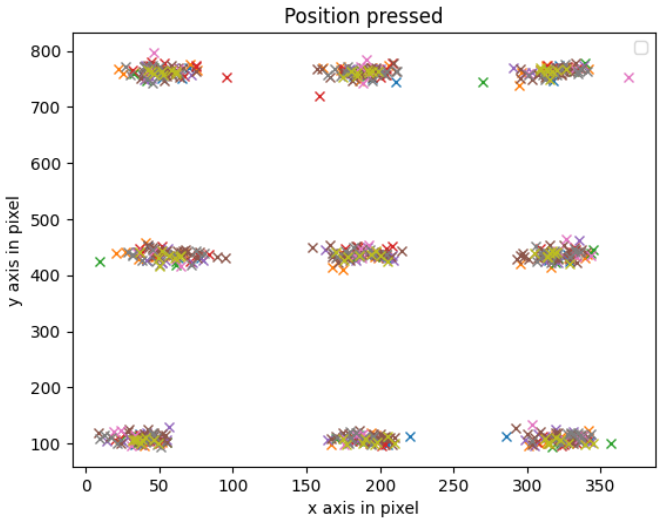

Fig. 4. Position in x and y pixels where the participants tapped onto the touchscreen during data recording.

The iOS app fetched IMU data at 100 Hz in the background and saved a time series of 0.5 seconds before and after each tap action. Furthermore, the orientation (yaw, pitch, roll) of the device got also tracked at 100 Hz, as well as the location of the tap interaction. In total, the following data got recording after each interaction:

- 3 axis accelerometer @ 100 Hz: 300 data points
- 3 axis gyroscope @ 100 Hz: 300 data points
- 3 axis magnetometer @ 100 Hz: 300 data points
- roll, pitch, yaw @ 100 Hz: 300 data points
- x-axis and y-axis (in pixels): 2 data points

The participants pressed multiple times on each defined location which resulted in a dataset of 723 samples with each having 1,202 features. The number of samples per holding posture was nearly equally distributed as seen in Figure 5.

B. Data preprocessing

After the dataset was complete, the approach was followed as shown in the flowchart in Figure 6. On the training set, multiple feature extraction methods were performed:

- **Statistical features**: The time series of each axis were used to extract the following features: mean, median, maximum, minimum, standard deviation, skewness, kurtosis, interquartile range, mad, integrate along the given axis using the composite trapezoidal rule, range, root mean square, variance and the number of peaks. Furthermore, each sensor was used to extract the mean standard deviation and the mean of variance.
- **Frequency features with Fast Fourier Transform (FFT)**: FFT calculation from all axis was done to calculate the following features: mean of FFT magnitude, the standard deviation of FFT magnitude, mean of FFT angle, the standard deviation of FFT angle.
- **Magnitude features**: For each sample the magnitude for each axis was calculated to derive some more statistical features: mean of magnitude, maximum magnitude, minimum magnitude, variance of magnitude, standard

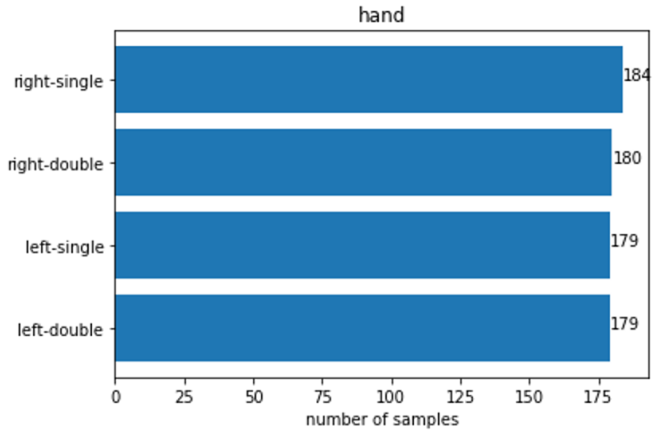

Fig. 5. Samples per posture in the dataset.

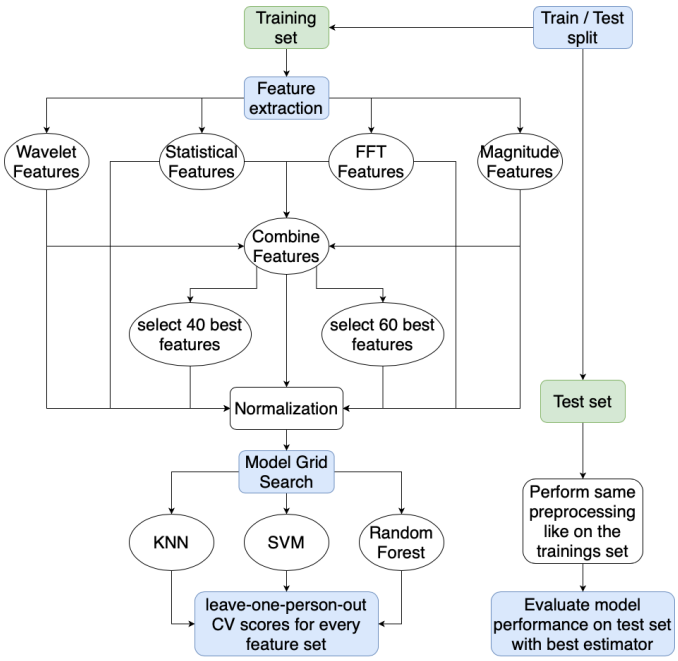

Fig. 6. Flowchart from data to ML model.

deviation of magnitude, and number of peaks.
- **Wavelet features**: To combine time and frequency domain the db3 wavelet was used to get the number of wavelet levels and then the approximated coefficients instead of the detailed ones were used.

IV. Evaluation

In order to find the best model suited for this task, several feature extraction methods were used in combination with various machine learning models.

A. Feature Engineering

First of all, the raw time series data was used as a feature set to find the baseline of the model and see how different models would perform without any preprocessing and feature engineering except for scaling and filtering. The extracted features mentioned in Section III-B were used individually for training and also combined together to a big feature set that accounted for 370 different features. Furthermore, not only each sensor but also each sensor axis was used individually for a feature set, as some axis might perform better and have more relevance in solving the problem than others. For example, the wavelet feature set actually consisted of several different feature sets: only wavelets of accelerometer/gyroscope/magnetometer/motion for only x/y/z axis, all wavelets of each sensor individually, all wavelets of the IMU combined, all wavelets of gyroscope and accelerometer combined, all wavelets of all sensors combined and 3 of these feature sets where the correlation between features above 0.8 and under -0.8 has been removed. That way the wavelet feature set actually consisted of 22 different sets. With a scoring function (ANOVA F-value), the best 40, 80, 120, and 160 features were selected to find out if these new feature sets can increase the accuracy further. In total, this setup accounted for 92 different feature sets:

- raw time series (scaled)
- raw time series (scaled + filtered)
- only statistical features
- only FFT features
- only magnitude features
- only wavelet features
- all features combined
- best 40/80/120/16 features with the scoring function
- all features combined but correlation above 0.8 and under -0.8 removed (114 features)

B. Model selection with gridsearch

The models k-nearest neighbor (KNN), random forest, and support vector machine (SVM) were used for a hyperparameter grid search in Python using the models of scikit-learn. KNN was trained with 96 different hyperparameter variations, random forest with 576, and SVM with 56. Each model was trained with each feature set and with 5-fold cross-validation as well as leave-one-group-out cross-validation. Because the number of samples per participant was not equally distributed, the leave-one-group-out cross might introduce some bias, and since the accuracy numbers were only slightly different at the end, all following numbers from the training refer to 5-fold cross-validation.

The hyperparameters of each model were chosen as follows.

1) KNN:
- **neighbors**: in the range of 3 to 30 excluding multiples of 4 (because of 4 different holding postures)
- **weights**: uniform and distance
- **metric**: euclidean and manhattan

2) Random Forest:
- **bootstrap**: true
- **max depth**: 10, 20, 30, 40
- **max features**: 2, 3, 4, 5
- **min samples leaf**: 2, 4, 5, 8
- **min samples split**: 4, 8, 12
- **n estimators**: 100, 200, 300

3) SVM:
- **kernel**: rbf and linear
- **gamma**: only for rbf kernel: 3^{-8}, 3^{-7}, 3^{-6}, 3^{-5}, 3^{-4}, 3^{-3}, 3^{-2}
- **C**: 3^{0}, 3^{1}, 3^{2}, 3^{3}, 3^{4}, 3^{5}, 3^{6}, 3^{7}, 3^{8}

V. Results

Figure 7 shows different evaluation metrics for different feature sets and models. The first two columns are the best models with the raw time series feature set (scaled, scaled, and filtered) to find the baseline of the model. The third column is the best overall model where all features were tested, and the last column is the best model which only used feature sets with removed correlation. Overall, wavelet features performed extremely well on their own while the magnitude features achieved a low accuracy.

Model Feature set Sensor	Random Forest scaled timeseries IMU	Random Forest filtered timeseries IMU	SVM combined features all	Random Forest wavelets corr < 0.9 all
left-single precision, recall, f1	0.908, 0.937, 0.922	0.865, 0.938, 0.9	0.906, 0.96, 0.932	0.885, 0.885, 0.885
right-single precision, recall, f1	0.865, 0.882, 0.874	0.889, 0.842, 0.865	0.915, 0.796, 0.851	0.85, 0.911, 0.879
left-double precision, recall, f1	0.904, 0.922, 0.913	0.831, 0.86, 0.845	0.839, 0.839, 0.839	0.889, 0.873, 0.881
right-double precision, recall, f1	1.0, 0.923, 0.96	0.962, 0.909, 0.935	0.764, 0.824, 0.792	0.863, 0.815, 0.838
weighted average precision, recall, f1	0.919, 0.917, 0.918	0.887, 0.885, 0.885	0.855, 0.855, 0.855	0.871, 0.871, 0.871
cv score	0.846	0.842	0.883	0.893
accuracy	0.917	0.885	0.852	0.871

Fig. 7. Different evaluation metrics for different ML models on the test set with 5-fold cross-validation.

While the cross-validation score of the raw time series dataset was already at 84.6%, utilizing all features SVM achieved 88.3% and a random forest got an even better result with 89.3% when correlated features were removed.

The last model without the correlated features is ultimately preferable, not only because of the highest cross-validation score but also because of the lowest complexity as it requires fewer features than the other models. Also, looking at the

 ISBN: 978-1-68558-047-6

confusion matrix in Figure 8, the ROC curve and AUC value in Figure 9 there are no abnormalities between the different holding postures, as all perform very similarly.

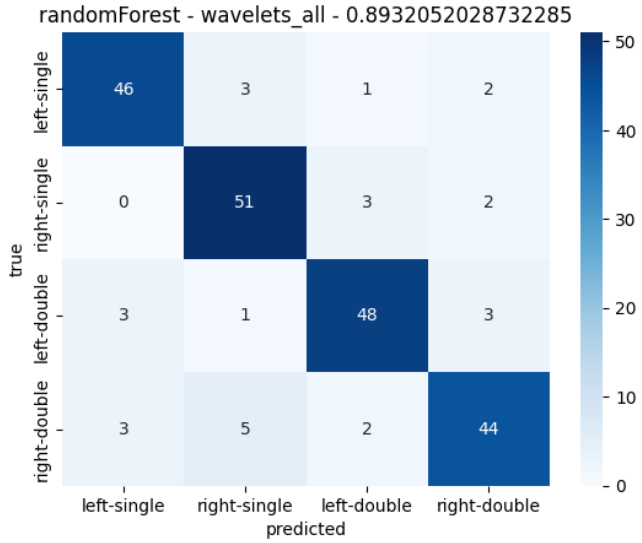

Fig. 8. Confusion matrix of the final model (Random Forest).

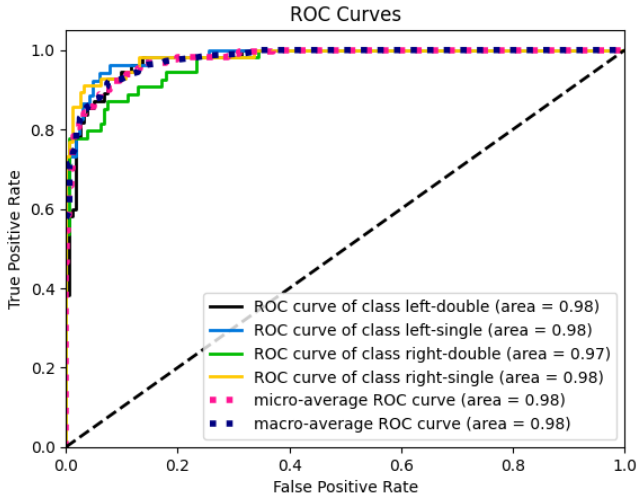

Fig. 9. ROC Curves and AUC value for the final model (Random Forest).

VI. Conclusion and Outlook

By analyzing the sensor data of the built-in IMU of smartphones we were able to detect the holding posture during the usage of an app with an accuracy of 89.3% without the need of external sensors. Therefore, the model is hardware and platform-independent and can be shipped on Android and iOS applications. Interestingly, the raw time series data performed way better than initially expected, feature engineering did increase the overall accuracy and reduced the complexity of the model though.

With the knowledge of the holding posture of the user at runtime, the UI designer can take this information into consideration while designing an intuitive interface. Also, self-adaptive capabilities in terms of the UI of a smartphone could be imaginable with real-time recognition of holding postures.

While this model now only works for tap gestures, we are currently working on adding swipe gestures with a Dynamic Time Warping approach to find out if swipe gestures can further improve the accuracy of the model. Besides, the dataset will be enlarged with new data recordings to compare the machine learning models with neural networks.

References

[1] S. Hoober, "How do users really hold mobile devices?" 2013, accessed: 2023-05-08. [Online]. Available: https://www.uxmatters.com/mt/archives/2013/02/how-do-users-really-hold-mobile-devices.php

[2] C. Hardyck and L. F. Petrinovich, "Left-handedness." *Psychological bulletin*, vol. 84, no. 3, p. 385, 1977, accessed: 2023-05-08. [Online]. Available: https://psycnet.apa.org/record/1978-00208-001

[3] "iPhone 5s - Technical Specifications," https://support.apple.com/kb/sp685?locale=en_GB, accessed: 2023-05-08.

[4] "iPhone X - Technical Specifications," https://support.apple.com/kb/sp770?locale=en_GB, accessed: 2023-05-08.

[5] "iPhone 13 Pro Max - Technical Specifications," https://support.apple.com/kb/SP848?locale=en_GB, accessed: 2023-05-08.

[6] R. Wimmer and S. Boring, "Handsense: discriminating different ways of grasping and holding a tangible user interface," in *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction*. New York, NY, USA: Association for Computing Machinery, 2009, pp. 359–362. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/1517664.1517736?

[7] K. Hinckley, S. Heo, M. Pahud, C. Holz, H. Benko, A. Sellen, R. Banks, K. O'Hara, G. Smyth, and W. Buxton, "Pre-touch sensing for mobile interaction," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2016, pp. 2869–2881. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/2858036.2858095

[8] M. Goel, J. Wobbrock, and S. Patel, "Gripsense: using built-in sensors to detect hand posture and pressure on commodity mobile phones," in *Proceedings of the 25th annual ACM symposium on User interface software and technology*, ser. UIST '12. New York, NY, USA: Association for Computing Machinery, 10 2012, p. 545–554. [Online]. Available: https://doi.org/10.1145/2380116.2380184

[9] M. Goel, A. Jansen, T. Mandel, S. N. Patel, and J. O. Wobbrock, "Contexttype: using hand posture information to improve mobile touch screen text entry," in *Proceedings of the SIGCHI conference on human factors in computing systems*. New York, NY, USA: Association for Computing Machinery, 2013, pp. 2795–2798. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/2470654.2481386

[10] C. Park and T. Ogawa, "A study on grasp recognition independent of users' situations using built-in sensors of smartphones," in *Adjunct Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. New York, NY, USA: Association for Computing Machinery, 2015, pp. 69–70. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/2815585.2815722

[11] M. Löchtefeld, P. Schardt, A. Krüger, and S. Boring, "Detecting users handedness for ergonomic adaptation of mobile user interfaces," in *Proceedings of the 14th international conference on mobile and ubiquitous multimedia*. New York, NY, USA: Association for Computing Machinery, 2015, pp. 245–249. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/2836041.2836066

[12] J. Avery, D. Vogel, E. Lank, D. Masson, and H. Rateau, "Holding patterns: detecting handedness with a moving smartphone at pickup," in *Proceedings of the 31st Conference on l'Interaction Homme-Machine*, ser. IHM '19. New York, NY, USA: Association for Computing Machinery, 12 2019, p. 1–7. [Online]. Available: https://doi.org/10.1145/3366550.3372253

 ISBN: 978-1-68558-047-6