Human-Computer Interaction Group

Leibniz Universität Hannover

Mobile Interaction
Summer 2024

Prof. Dr. Michael Rohs, Shashank Ahire, M.Sc.

Assignment 5

All exercises that are not explicitly declared as group tasks must be done individually and handed in individually. Identical submissions are treated as plagiarism. Plagiarism may lead to loss of exam bonus points.

You can submit the solution to this task in English or German until Wednesday, May 8, at 23:59 via https://assignments.hci.uni-hannover.de. Create a pdf file that contains the text and images of your solution, name it "Assignment-05-<Firstname>-<Lastname>.pdf", and save it together with the exported project (Android Studio: File → Export → Export to Zip File) in a single zip file. Your submission must consist of a single zip file containing all necessary files. The name of the .zip file, as well as the names of the contained files, **must not contain any umlauts.** Therefore, please resolve umlauts in file names.

Android Phones: Please let us know if you need an Android phone for the assignments. We can lend a limited number of simple Android phones for the duration of the semester. These must only be used for the assignments.

Exercise 1: Gestures (5 points)

In this task you should examine and describe aspects of gestures.

a) Name one advantage and one disadvantage of touchscreen gestures compared to graphical user interfaces on small screens. **(2 points)**

b) Do you use gestures on your smartphone or computer? If so: which ones and for what? Which gestures might be useful on a mobile/wearable device? Give reasons. **(3 points)**

Exercise 2: Protractor (13 points)

In this task, you will deepen your understanding of the Protractor algorithm for detecting unistroke gestures. To do so, you will have to complete the implementation of the template available on Stud.IP (MI-Assignment-05-ProtractorKotlinTemplate.zip) and evaluate your own gesture set (see below).

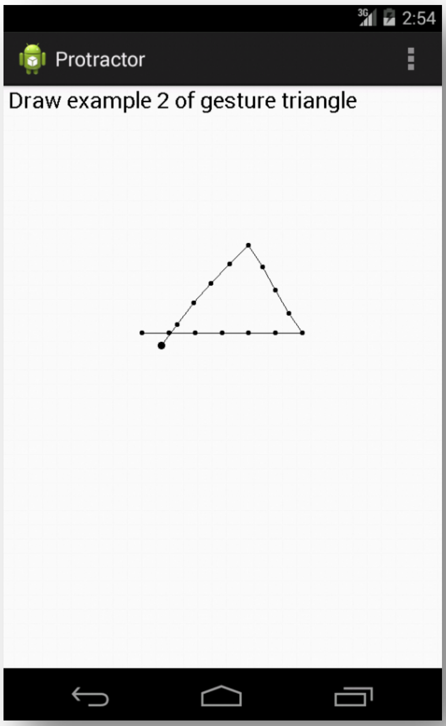
To make it easier to evaluate your own gesture set, an automated test can be started via the menu item "Test gestures - new participant". An array must be filled with the gestures to be tested. The order should be random. Each gesture should be repeated 10 times.

a) Complete the implementation of the following methods in the GestureRecognizer class. The places where you should insert source code are marked with @todo. The methods to be completed are: **pathLength(), centroid(), translate(), rotate()** and **optimalAngle(). (5 points)**

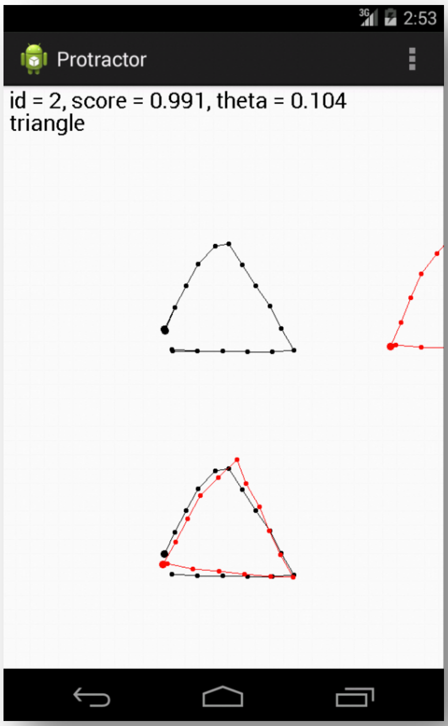
b) Enter a custom gesture set that uses gestures other than those used by the $1-Recognizer. Your gesture set should include 3 unistroke gestures. For each gesture 3 instances should be entered as templates. The input of templates can be called up via the menu item "Train Gestures" (not necessary the first time you call up the application). **(3 points)**

c) Now test the recognition rate for your gesture set with yourself as a test user. Use the automated test method and enter each gesture 10 times and note which gesture was recognized in each case. To do this, create a so-called "Confusion Matrix". The rows of this 3x3 matrix represent the entered gesture, the columns represent the recognized gesture. The entries in the matrix represent frequencies. Use dashes to enter as which gesture your gesture was recognized. The sum of the entries must therefore be 30 (10 attempts x 3 different gestures). **(3 points)**

d) Evaluate the Confusion Matrix. Calculate the recognition rate for each gesture. Do the results meet your expectations? Which of your gestures was most frequently not recognized? Why do you think that was the case? **(2 points)**



Draw example 2 of gesture triangle



id = 2, score = 0.991, theta = 0.104
triangle

Part of the submission is the answering of the questions (text and graphics) as PDF and the exported project.

Exercise 3: Composables with State (19 points)

In this exercise we use the "Tip Time" Android example to learn about state in Compose. Open the CodeLab at https://developer.android.com/codelabs/basic-android-kotlin-compose-using-state#2 and download the starter code. Open it in Android Studio and follow the steps of the CodeLab as instructed below. Also, answer the questions given below.

a) Explain `text = stringResource(R.string.calculate_tip)`. **(1 point)**

b) In Android Studio, open MainActivity.kt and activate the Split view (showing the source code on the left and the preview on the right). What happens to the preview if you set the spacer height to 0.dp? **(1 point)**

c) Add the EditNumberField composable function. Continue to the end of step 4 of the CodeLab. Why is `val amountInput = "0"` not useful for representing the state of the text field? **(1 point)**

d) What is an "initial composition" and what is a "recomposition"? **(2 points)**

e) How does the system track which state has changed? How to create mutable observable state? **(2 points)**

f) Continue to the end of step 5. In your own words, explain the problem with the mutable observable value staying at its initial value. **(2 points)**

g) Continue to step 6. Briefly explain how the Kotlin property delegation works and how it is used here. Briefly explain how "remember" works. **(3 points)**

h) Do steps 7 and 8. What type of keyboard is appropriate for the app? **(1 point)**

i) Continue to the end of step 10. Briefly explain the term "state hoisting". Is `EditNumberField` stateless or stateful at the end of step 10? Explain. **(3 points)**

j) Why are the parameters `value` and `onValueChanged` both necessary for `EditNumberField`? **(1 point)**

Now extend the app with a field to enter the desired tip percentage. Start with step 2 of https://developer.android.com/codelabs/basic-android-kotlin-compose-calculate-tip#2.

Then do steps 3 and 4.

k) Do step 5. How does the keyboard behavior change when the "Tip Percentage" text field uses the ImeAction.Done rather than the ImeAction.Next? Which behavior is preferable from a usability perspective? **(2 points)**