Hochschule
Augsburg University of
Applied Sciences

INTERACTION
ENGINEERING

AirTimer

How to operate a countdown timer without touching or speaking
and how to provide user feedback without a display

Jonas Riegel, Felix Kittler

Course Project for Interaction Engineering
Prof. Dr. Michael Kipp
Augsburg University of Applied Sciences
Wintersemester 2016 / 2017

Inhalt

1. Introduction

1.1 Abstract

AirTimer is a research project on exploring alternative approaches for the most common ways of human computer interaction: The device, a clock-like countdown timer, is not operated by any buttons, keyboard or touchscreen, but by in-air gestures only. Its user feedback is not given on a screen or on a LCD, but only by a simple ring of 60 LEDs.

1.2 Motivation

In the paper presentation of a fellow student, we heard about issues when working with touch-input, and we learned about an approach to provide user support or assistance: The authors of the paper[1] *"(...) present an interaction technique which uses multimodal feedback to help users address in-air gesture systems. The feedback tells them how ("do that") and where ("there") to gesture, using light, audio and tactile displays."*

Based on that idea, we wanted to take the concept even one step further by reducing the used features to a minimum:

- Input only by 3 different single-hand in-air gestures
- Output only by a 360° LED ring

Accordingly, the fundamental questions of our work are:

- Gestures: Find a balance between convenience and necessity
- Display: Define clear but definite way of communication

As an input device, we wanted to use Leap Motion. To reduce complexity, the output LED ring was only simulated on screen. As software, we used Processing 3.
A hardware implementation with a single-board computer (like Raspberry Pi or Arduino) is not in scope of this project - a proof of concept was conducted, actual implementation is planned as "future work".

1.3 Related work

- Do That, There: An Interaction Technique for Addressing In-Air Gesture Systems
  Euan Freeman, Stephen Brewster, Vuokko Lantz
  Conference for Human-Computer Interaction 2016, San Jose, CA, USA
- Interactive Light Feedback: Illuminating Above-Device Gesture Interfaces
  Euan Freeman, Stephen Brewster, and Vuokko Lantz
  Human-Computer Interaction – INTERACT 2015
- Ambient Gesture-Recognizing Surfaces with Visual Feedback
  Tobias Grosse-Puppendahl, Sebastian Beck, Daniel Wilbers, Steeven Zeiß, Julian von Wilmsdorff, and Arjan Kuijper
  Distributed, Ambient, and Pervasive Interactions. DAPI 2014

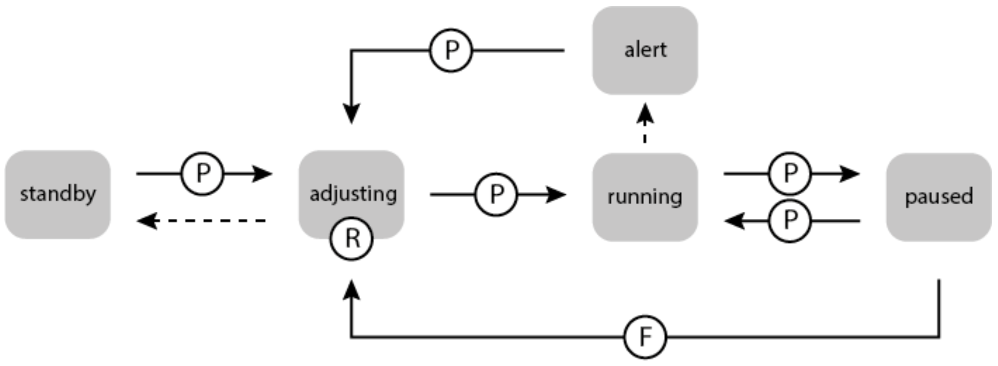
[1] *Do that, there - An Interaction Technique for Addressing In-Air Gesture Systems* - see *1.3 Related Work*

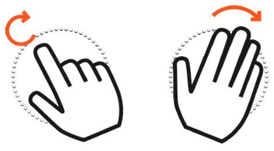
2. Concept

2.1 Gesture-based input

To develop the gesture set for the timer device, we analyzed the different steps usually taken when setting up a countdown. We found that only one action requires specific, precise user input - which is when adjusting the desired countdown duration (minutes and/or seconds). For all other actions, be it confirming, continuing, resuming or resetting, it seemed sufficient to implement rougher, simpler gestures.

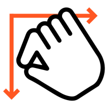
Taking this into account, we tried to reduce the variability of gestures even more: If already simplifying, why not take it to an extreme? We wanted to map many different user actions to one single gesture - and react only based on context, by implementing a simple but powerful “menu structure”:



Punch		For advancing further by confirming the current step, a “punch” (similar to the movement when hitting a buzzer) is used. Only in status “paused” and “alert”, when no logical subsequent action is available, a punch goes back to “adjusting”.
Rotate		For adjusting minutes, a circling gesture with the index finger was foreseen, to remind the user of “winding up” the clock. For seconds, a pitch / tilt movement with the flat hand should trigger a mechanism to increase or decrease the set time (no relative mapping, but like a throttle control).
Flick		For cancelling or resetting, a swipe gesture was implemented.

Note: The actions drawn as dashed arrows are controlled automatically.

However, due to the problem known as “midas problem”, we soon understood that adjusting minutes and seconds by simply tracking any rotation gestures of the hand would only achieve very unsatisfactory results: During adjustment, it was not possible to easily confirm the currently set time: While performing the “punch” gesture to start the countdown, rotation gestures were recognized and falsified the time setting.
To avoid this, the gesture set was revised and fine-tuned. In particular, a pinch-gesture was introduced, to replace the rotation, to easily determine start and end of the adjusting gesture:

Pinch		While maintaining a pinch gesture (closed index and thumb tips), movement of the hand was tracked and mapped to a respective time setting. X-axis (up and down) was used for minutes Y-axis (left and right) was used for seconds

2.2 User feedback by LEDs

Because the “device” would later work like a clock, as a matter of course, we used 60 LEDs, aligned on a ring. As said earlier, to simplify test setup, the ring was simulated on computer screen.

On startup, in an animation of one second, the ring completes a run through all color hues and then fades to black. The application then waits for user input. To distinguish this adjustment mode from the initial standby mode, the LEDs are now given a 20% brightness.

This default 20% brightness however increases, according to the distance of the user’s hand to the leap motion device: To provide feedback regarding hand tracking status and the ideal hand distance, we defined a range from 300 to 700mm from the leap motion as preferred gesture distance. Within this distance, the brightness increases according to the user’s hand position, with the center of the range at 500mm being the ideal distance and resulting in 100% brightness. Below 300 and above 700mm from the Leap Motion device, so outside of the preferred distance range, the LED brightness remains at 20%.

During adjustment, but also later when counting down, minutes and seconds are shown by dedicated colors: The number of remaining minutes is shown by lighting the respective number of LEDs on the ring constantly in red. On the contrary, the seconds are shown in green, and only one single LED shows the amount of remaining seconds.
This concept allows to show two values - the amount of minutes and seconds - simultaneously on one ring. For example, a time of 04:32 would show four red LEDs on position 1-4, and one single LED at position 32 (no matter if in adjustment mode or while running the countdown).

The above described color concept is subject to one alteration: Once there are no more minutes left to display when counting down, all seconds are shown constantly in green, as the minutes were before. This ensures that the user is aware of the time running out.

When reaching zero, the whole ring blinks in red by fading in and out.

3. Implementation

As briefly mentioned in the introduction, we used *Processing 3* and the *Leap Motion for Processing* Library by Darius Morawiec to simulate the LED ring.

3.1 Arduino & real LED ring

Initially, as a test, we connected an Arduino Uno with an LED ring. Input received from the Leap Motion was evaluated within Processing, running on a regular computer, calculated output was then sent to the Arduino via USB. The test delivered promising results, but to reduce complexity, we decided to leave the hardware implementation ouf of the scope of this project.

3.2 Processing

Focus was then primarily on building the navigation concept described in chapter 2.1. In a first concept, we replaced gesture recognition and used the keyboard to trigger certain events, like the push or the swipe gesture.

In this state of the project, we still expected to implement rotary gestures for adjusting minutes and seconds. Thus, we planned two separate steps - first setting up minutes, then performing a push gesture to confirm, and then setting up seconds. This left us with the following states of the app:

`0` - off, when the app is not initialized
`1` - standby, the app is initialized, but currently inactive.
`2` - starting, interim state during startup (animation, 1 second)
`3` - while adjusting minutes
`4` - while adjusting seconds
`5` - timer running / counting down
`6` - timer paused / countdown on hold
`7` - timer reached 0 / alarm

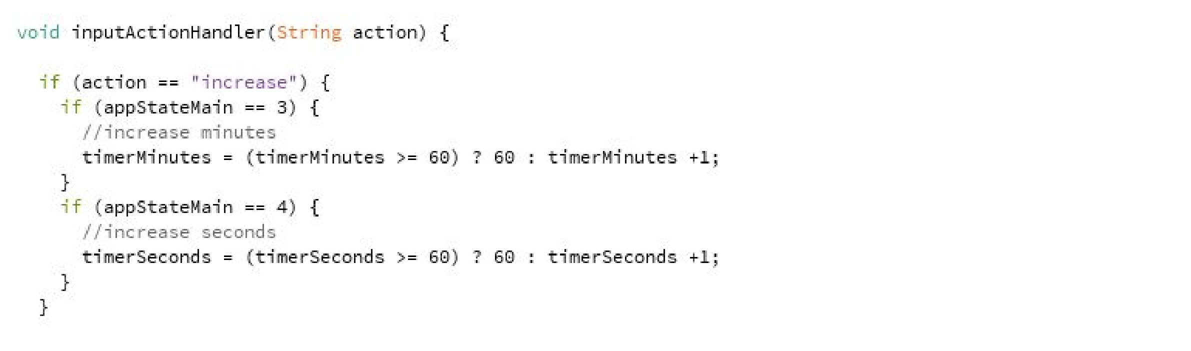
Basically, in each frame's `draw()`, the program would check the current status, which is saved in the `int` variable called `appStateMain`, and call a respective function. As mentioned in 2.1 Gesture Input, status 1 and status 6 finish automatically, based on time. All other status changes are invoked by a gesture, mostly by the “punch / push” gesture.

As stated earlier, in the beginning, keyboard input was used to test the program’s navigation concept. In this stage, the ring simulation on screen was fine-tuned and brought to its final look. We did not experience any problems with implementing the program as described in our concept.

However, the next step was connecting the Leap Motion to Processing, where we encountered several problems.

We started off with built-in gestures, primarily `leapOnScreenTapGesture` , which then called the same functions that were earlier triggered by key input. Because `leapOnScreenTapGesture` triggered multiple times within one punch / push gesture, we implemented helper variables, introducing a short delay after each triggering.

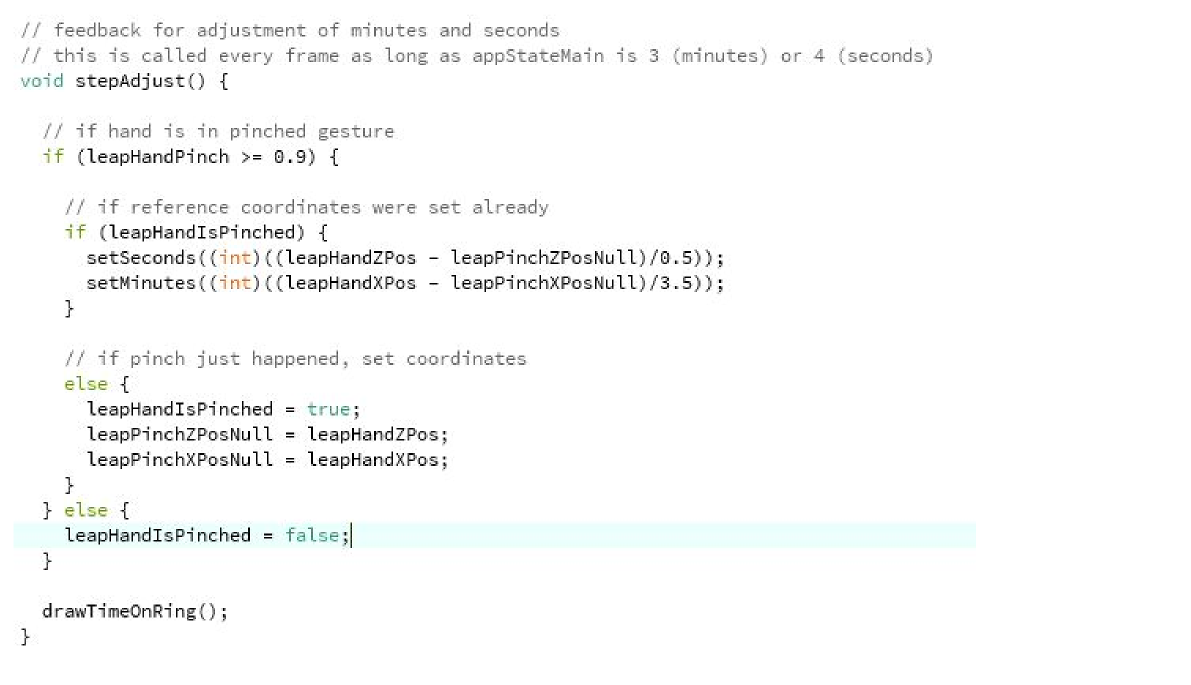
At this time, we also replaced the methods for adjusting the time. Before, two separate keys on the keyboard were used to increase or decrease the currently set time. Now, we called the same functions and decided based on the stabilized position of the hand, if minutes (or seconds, depending on the current menu step) should be increased or decreased. This worked in general, but it was not possible to set a detailed time: When confirming with a punch, the hand movement falsified the currently set time and left the user with an incorrect setting.



```
void inputActionHandler(String action) {

  if (action == "increase") {
    if (appStateMain == 3) {
      //increase minutes
      timerMinutes = (timerMinutes >= 60) ? 60 : timerMinutes +1;
    }
    if (appStateMain == 4) {
      //increase seconds
      timerSeconds = (timerSeconds >= 60) ? 60 : timerSeconds +1;
    }
  }
```

Excerpt of `inputActio nHandler`, which (in an early stage of the project) was called upon key input (event based).



```
// feedback for adjustment of minutes and seconds
// this is called every frame as long as appStateMain is 3 (minutes) or 4 (seconds)
void stepAdjust() {

  // if hand is in pinched gesture
  if (leapHandPinch >= 0.9) {

    // if reference coordinates were set already
    if (leapHandIsPinched) {
      setSeconds((int)((leapHandZPos - leapPinchZPosNull)/0.5));
      setMinutes((int)((leapHandXPos - leapPinchXPosNull)/3.5));
    }

    // if pinch just happened, set coordinates
    else {
      leapHandIsPinched = true;
      leapPinchZPosNull = leapHandZPos;
      leapPinchXPosNull = leapHandXPos;
    }
  } else {
    leapHandIsPinched = false;
  }

  drawTimeOnRing();
}
```

Excerpt of `stepAdjust`, called once a frame when in adjustment mode

To avoid this, we decided to introduce a pinch gesture: Only when tips of digit finger and thumb would be close enough to each other, the tracked movement of the hand is used to manipulate the time setting.

Because it is easy to distinguish between different axes here, we decided to merge the two adjustment steps into one, allowing setup of minutes and seconds simultaneously. Movement upwards and downwards is used to set minutes, left and right for the seconds. This is also in a strong contrast to the punch gesture, which would be executed in the third axis, towards the device.

Furthermore, we decided to write an own function to recognize the punch gesture: The before mentioned `leapOnScreenTapGesture` did not prove reliable.

```
// permanently check leap motion input for punch gesture
// it saves the height of the stabilizedHand to an array
void punchDetector() {

  // this should not be checked every frame
  // but only every 0.1 seconds
  if (leapPunchDetectorDelay < millis() - 100) {

    leapPunchDetectorDelay = millis();

    //leapHandTracker[5] saves the last 5 positions of the hand
    for (int i = 4; i >= 0; i--) {
      //on position 0, store current position
      if (i == 0) {
        leapHandTracker[i] = leapHandHeight;
      }
      // on position 1-4, store positions of last 4 frames
      else {
        leapHandTracker[i] = leapHandTracker[i-1];
      }
    }

    //compare values in the array
    //only if array is fully set
    if (leapHandTracker[4] !=0) {
      for (int i=0; i<5; i++) {

        // current position must be 300 lower than any of the last 4
        if (leapHandTracker[0] - 300  > leapHandTracker[i]) {
          for (int j=0; j<5; j++) {
            leapHandTracker[j] = 0f;
          }
          inputActionHandler("punch");
          break;
        }
      }
    }
  }
}
```

Function `punchDetector`, called every frame throughout the runtime of the app, storing the hand position to an array and comparing the values to detect a punch gesture.

4. Conclusion and future work

As it can be seen in the demo video, gesture input works stable and reliable, with room for improvement. In very simple user tests and also when presenting our work, we found that it might be reasonable to again separate the adjustment steps for minutes / seconds. Even though this would not increase the precision of the gesture recognition, it would still simplify usage for the user, as he/she can concentrate on one setting at a time. To explore this in more depth, a more detailed user study might be helpful.

Regarding the hardware side, long-term goal was to implement the project without the usage of a regular computer, so on a single-board computer like Arduino only. Aside from very basic hardware restrictions (How to connect the Leap Motion to the Arduino?), there might be also other obstacles in the way (installing the Leap Motion SDK for Processing and Processing itself on the Arduino?).

The source code of our project is available in a public github repository at https://github.com/felixkittler/AirTimer and will be kept updated on further project progress.

The demo video can be seen on vimeo at https://vimeo.com/202667641.

February 2017
Jonas Riegel
Felix Kittler