Dokumentation
Leap Motion Controller & Durovis Dive

Arno Fuhrmann

31. März 2016

Datum: 10. März 2016
Betreuer: Prof. Dr. Thomas Gabel

Zusammenfassung

Virtual Reality ist eine Technologie, die derzeit als großer Trend in der Technikszene aufkommt. Nach den letzten großen Technikmessen wie der Global Gaming Expo und der International CES dominieren VR-Anwendungen und Spiele die Berichterstattungen der Presse. Eine weitere aufkommende Technologie, die man in diesem Zusammenhang ins Auge fassen sollte, ist die Einbindung von Infrarotsteuerung um die Hände zum Navigieren in Applikationen abzubilden. Nachfolgend auf das Projekt "Virtual Reality on Smartphonesünd der erstellten Dokumentation Ëinführung in Unity3D und Durovis Diveünter Verwendung einer VR-Brille des Anbieters Durovis, folgt nun mit dieser Dokumentation eine Einführung in die Verwendung der Infrarot-Gestensteuerung mit dem Controller des Anbieters Leap Motion, der eine SDK für Unity anbietet und für den Durovis eine Halterung für die Dive VR-Brille zur Verfügung stellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Quelltexte

1 Einführung

Dieses Dokument gibt eine Einführung in die Kombination von Leap Motion Controller (Infrarot Gestensteuerung) und die Durovis Dive VR-Brille.

1.1 Einrichten des Leap Motion Controllers

Wir beginnen damit, den Leap Motion Controller mit dem Android Gerät zu verbinden. Im Folgenden sind Hard- und Softwareanforderungen beschrieben, darauf folgt eine kurze Installationsanweisung.

Hardware

- Android Smartphone (hier: LG G3)
- Leap-Motion-Controller
- Micro-USB3.0-USB3.0-Kabel
- USB-Buchse-Micro-USB-Stecker-Adapterkabel

Software

- Leap Daemon Skeletal APK (aus dem Leap Motion Android SDK)
- Man findet die App im Ordner „LeapDeveloperKit“

Als erstes muss die Leap Daemon Skeletal APK auf das Smartphone verschoben werden. Dazu muss das Smartphone mit dem Computer verbunden werden, alle nötigen Treiber installiert und die APK Datei auf den internen Speicher oder die SD-Karte des Smartphones kopiert werden.
Der nächste Schritt ist die APK auf dem Smartphone zu installieren. Bei Problemen hierbei bitte den Hinweis am Ende des Kapitels beachten. Ist die Leap Daemon Skeletal Applikation erfolgreich installiert, kann der Leap Motion Controller mithilfe des Micro-USB-Adapters mit dem Smartphone verbunden werden. Um die Verbindung zu kontrollieren, muss nun die Leap Daemon Skeletal App gestartet werden.

Hinweis: Um Applikationen auf dem Android Gerät zu installieren, die vom Computer kopiert wurden, muss die Einstellung „Installation von unbekannten Quellen“ im Menü Einstellungen → Sicherheit aktiviert werden.

1.2 Durovis Dive und Dive SDK in Unity

Zur Einführung in die Verwendung von Durovis Dive mit Unity3D, siehe Dokumentation „Einführung in Unity3D und Durovis Dive“

1.3 Leap Motion SDK in Unity3D

Zum Einbinden der Leap Motion SDK muss nur das von Leap Motion zur Verfügung gestellte unitypackage importiert werden. Möchte man in verschiedenen Projekten Leap Motion verwenden, ist es sinnvoll das unitypackage in den Standard Ordner „...\Unity\Editor\Standard Assets“ zu verschieben, dann kann sie nach Neustart von Unity einfach über das Kontextmenü „Import Package“ im Projekt-Ordner eingebunden werden.

2 Interaktion und Gesten

Im folgenden Kapitel wird der Umgang mit dem eingebundenen SDK erläutert.

2.1 Erste Schritte mit dem HandController

Nach dem Einbinden des Leap Motion SDK kann man nun eine kleine Testumgebung einrichten um sich mit dem wichtigsten Element, dem HandController vertraut zu machen. Hierfür legen wir in unserem leeren Projekt einen Hand Controller an, indem wir das HandController-Prefab aus dem Ordner Assets → Prefabs in unser Scenen-Fenster verschieben.

Nach dem Hinzufügen des HandControllers macht es Sinn, zur besseren Betrachtung, die Größe des HandControllers anzupassen. Ein Wert von 25 bei allen Scale-Werten lässt die Hände wie in der nachfolgenden Abbildung groß genug erscheinen, um damit erste Tests zu machen. Die richtige Position des HandControllers, durch die auch bei Laufzeit die Position der Arme/Hände festgelegt wird, kann einfach durch Ausprobieren ermittelt werden.

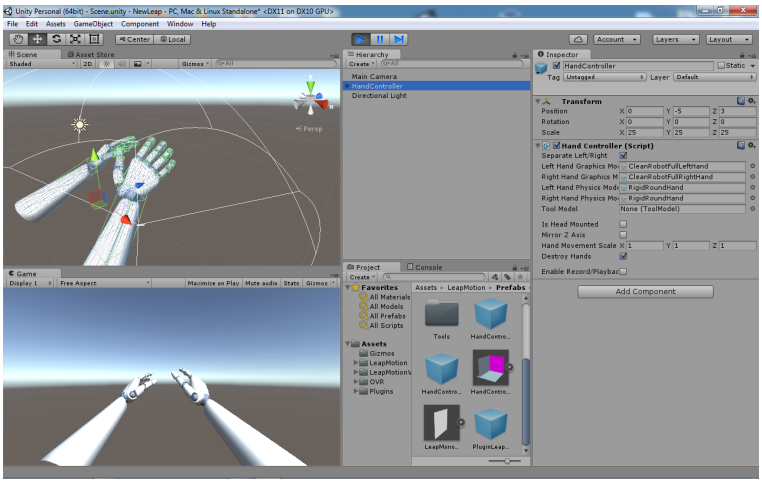

Abb. 1: Einbinden des Leap Motion SDK in Unity3D

Möchte man den Armen/Händen ein anderes Aussehen geben, kann man dies über das Auswählen anderer Models in den Public-Variablen des HandController-Scripts, wie auf der Abbildung oben auf der rechten Seite zu sehen. Hier kann man einfach für das linke und rechte „... Hand Graphics Model“ ein anderes Modell aus „LeapMotion → Prefabs“ und dem entsprechenden „HandModel...“-Unterordner wählen. Durch die beiden „... Hand Physics Model“ ist bereits ein RigidBody hinzugefügt, der für Kollisionen mit Objekten sorgt, denen ebenfalls ein RigidBody hinzugefügt wurde. Im nächsten Kapitel werden wir dies Nutzen, um den ersten Kontakt zwischen den Händen und Objekten zu testen.

2.2 Mit Objekten „hantieren“

Nun, da wir die Arme/Hände in unsere Szene projizieren können, möchten wir mit den Händen auch Objekte anfassen und damit agieren können. Dazu fügen wir zunächst eine „Plane“ hinzu, plazieren einen Würfel („Cube“) darüber und geben dem Würfel einen RigidBody.

Starten wir nun die Szene, können wir den Würfel auf der Ebene verschieben, herumrollen, hochheben, wegwerfen oder von der Ebene herunterschubsen. Somit haben wir unseren ersten Kontakt zu Objekten hergestellt und der Phantasie kann in der Entwicklung freier Lauf gegeben werden.

In den folgenden Kapiteln wird weiter erläutert, wie man mit Objekten umgehen kann. Vor allem wird auf das Greifen von Objekten und das Navigieren durch z.B. Drücken von Schaltern etc. eingegangen.

2.3 Greifen von Objekten

Um mit Elemtenten in der virtuellen Umgebung interagieren zu können, ist es unabdingbar, Objekte greifen und bewegen zu können. Im vorherigen Kapitel wurde bereits gezeigt, wie Objekte mit einfachen Mitteln, die durch Unity zur Verfügung gestellt werden, bewegt werden können. Versucht man nun mit diesen Einstellungen z.B. den erstellten Würfel zu greifen und zu halten, stellt man fest, dass es schwierig bis unmöglich ist, dies zuverlässig zu bewerkstelligen. Hier stellt uns die Leap Motion SDK jedoch ein Werkzeug zur Verfügung, das in diesem Kapitel näher beschrieben wird.

Wurde das Leap Motion SDK in Unity importiert, findet sich das Script „MagneticPinch.cs“ im Unterordner LeapMotion → Scripts → Utils. Dieses Script kann man nun dem linken und rechten Prefab für das „Hand Physics Model“ hinzufügen, indem man auf dem im vorletzten Kapitel erstellten „HandController“ klickt und im Inspector mit einem Doppelklick auf das entsprechende Prefab klickt. Nun öffnet sich die Inspector-Ansicht für das entsprechende Prefab, wo man durch Drag&Drop das Script hinzufügen kann.
Starten wir das Beispiel jetzt, kann man den Würfel greifen, halten und bewegen, ohne dass er sich von den Fingern löst, solange man einen „Pinch“ ausführt,

also die Finger zusammenhält. Man stellt allerdings fest, dass der im Beispiel verwendete Würfel an einer durchsichtigen Feder zu hängen scheint und sich an dieser auch herumwirbeln lässt. Es ist also ratsam, das Script „MagneticPinch“ auf die eigenen Bedürfnisse anzupassen.

Kopieren wir also das Script in unseren eigenen Ordner für Scripte und benennen es um, z.B. in „Pinch.cs“. Wenn wir das Script öffnen und ganz nach unten scrollen, finden wir die Abfrage „$if(grabbed_! = null)$“. Dieser Code wird also ausgeführt, wenn wir etwas in der Hand halten. Die komplette Abfrage sieht aus wie folgt:

```
1 // Accelerate what we are grabbing toward the pinch.
2   if (grabbed_ != null) {
3     Vector3 distance = pinch_position - grabbed_.
      transform.position;
4     grabbed_.GetComponent<Rigidbody>().AddForce(
      forceSpringConstant * distance);
5   }
```

Quelltext 1: Greifzustand abfragen

Wie man feststellen kann, wird dem Objekt, das festgehalten wird, eine Kraft in Richtung der festhaltenden Fingern, abhängig von seiner Entfernung, hinzugefügt. Dies bewirkt den vorhin festgestellten Effekt eines flexiben Bands zwischen Fingern und Würfel.

2.4 Gesten erkennen

Ein weiteres interessantes Thema in Leap Motion ist die Gestenerkennung. Für das Erkennen von Tippen mit dem Finger („Key Tap“ und „Screen Tap“), Wischgesten („Swipe“) und Kreisbewegung („Circle“) stellt das Leap Motion SDK Funktionen zur Verfügung.

Die folgenden Abbildungen zeigen, wie die Gesten ausgeführt werden um vom Leap Motion SDK erkannt zu werden:

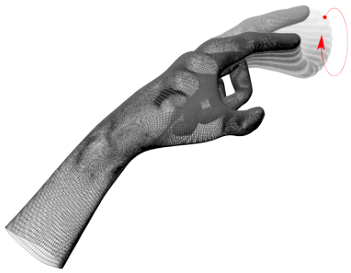

Abb. 2: Kreisgeste „Circle“

Abb. 3: Wischgeste „Swipe“

Abb. 4: Tippgeste „Key Tap“

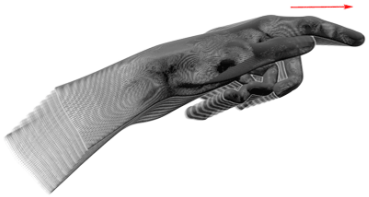

Abb. 5: Tippgeste „Screen Tap“

Diese Gesten müssen in einem Script zuerst aktiviert werden. Dazu verwendet man folgende Befehle:

```
1 controller.EnableGesture (Gesture.GestureType.
    TYPE_CIRCLE);
2 controller.EnableGesture (Gesture.GestureType.
    TYPE_KEY_TAP);
3 controller.EnableGesture (Gesture.GestureType.
    TYPE_SCREEN_TAP);
4 controller.EnableGesture (Gesture.GestureType.
    TYPE_SWIPE);
```

Quelltext 2: Benötigte Gesten aktivieren

Diese Aktivierung kann in der Funktion Start() erfolgen und um Zugriff auf die Klasse „Controller“ zu haben, muss man Leap einbinden („using Leap;“). Ein neuer Controller wird einfach als globale Variable „Controller controller = new Controller();“ definiert.

Nach der Aktivierung kann man in der Funktion Update() alle Gesten des aktuellen Frames abfragen, durch diese Liste iterieren und den jeweiligen Gestentyp abfragen. Hier bietet sich ein Switch-Case an. Der Quelltext dazu sieht aus wie folgt:

```
1  Frame frame = controller.Frame ();
2  gestures = frame.Gestures ();
3
4  foreach (Gesture g in gestures) {
5
6    if (!g.IsValid)
7      continue;
8
9    switch (g.Type) {
10   case Gesture.GestureType.TYPESWIPE:
11     Debug.Log ("SWYPE");
12     break;
13   case Gesture.GestureType.TYPESCREENTAP:
14     Debug.Log ("SCREENTAP");
15     break;
16   case Gesture.GestureType.TYPECIRCLE:
17     Debug.Log ("CIRCLE");
18     break;
19   case Gesture.GestureType.TYPEKEYTAP:
20     Debug.Log ("KEYTAP");
21     break;
22   }
23 }
```

Quelltext 3: Switch-Case Gestentyp

Zum Testen muss das Script nun dem Handcontroller hinzugefügt werden und nach dem Starten der Scene kann man in der Console sehen, wie die Gesten erkannt werden.

3 Anwendungsbeispiel

Da nun die Grundlagen erarbeitet wurden, folgt nun ein Anwendungsbeispiel mit Gesten. In diesem Beispiel soll man durch wischen zwischen verschiedenen

Filmen, die mit Bild, Titel, Beschreibung und Kino-Startdatum angezeigt werden sollen, hin und her navigieren und durch antippen ein Link im Browser zur Detailbeschreibung und dem Trailer geöffnet werden.

3.1 Scene aufbauen

Zuerst werden in einer neuen leeren Scene die Objekte angelegt, mit denen interagiert werden soll. Wie in Kapitel 2 beschrieben, sollte neben der standardmäßig enthaltenen MainCamera und dem DirectionalLight ein Handcontroller angelegt werden. Zusätzlich legen wir mehrere Canvas an, platzieren das „Start“-Canvas in der Mitte vor der Camera und alle weiteren um die gleiche Entfernung seitlich (x-Werte) versetzt. Im Beispiel wird der Wert 1000 als Versatz gewählt und insgesamt 3 Canvas angelegt, die ein Bild und mehrere Texte enthalten. Ist die Scene fertig, sieht sie aus wie auf der folgenden Abbildung gezeigt:

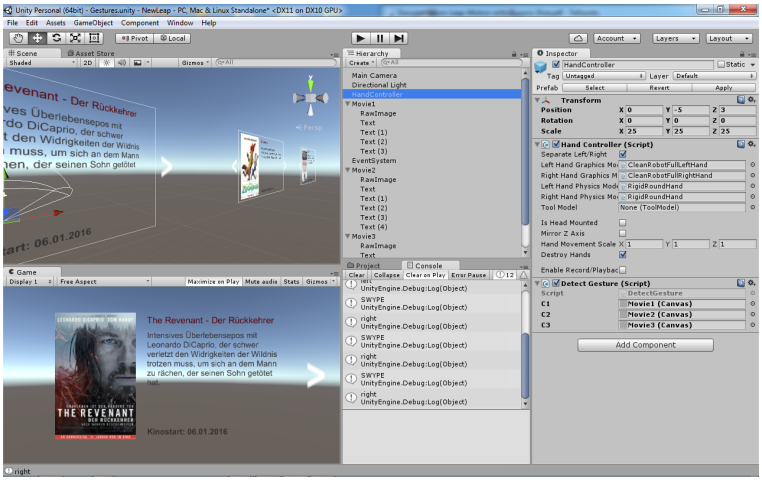

Abb. 6: Scenenaufbau des Anwendungsbeispiels

Mit diesen Objekten ist die Scene fertig gestellt und wir können zur Programmierung übergehen.

3.2 Gestensteuerung implementieren

Wie im Kapitel 2 erläutert, müssen die gewünschten Gesten aktiviert werden. Die Funktion Start() enthält also die Aktivierung von „Key Tap“ und „Swipe“. Außerdem müssen wir die Startposition unserer Canvas-Objekte initialisieren. Die Variablen für die Startposition sind global (über der Start()-Funktion) definiert.

```
1 void Start () {
2   controller.EnableGesture (Gesture.GestureType.
       TYPE_KEY_TAP);
3   controller.EnableGesture (Gesture.GestureType.
       TYPE_SWIPE);
4
5   c1start = c1.transform.position;
6   c2start = c2.transform.position;
7   c3start = c3.transform.position;
8 }
```

Quelltext 4: Start()-Funktion des Anwendungsbeispiels

Die Variable Controller sollte ebenfalls global, wie in Kapitel 2 beschrieben, definiert werden. In der Funktion Update() können dann alle im aktuellen Frame enthaltenen Gesten abgefragt und in einer GestureList abgespeichert werden, über die dann mittels foreach()-Schleife iteriert werden kann. In jeder dieser Iteration sollte dann der Gestentyp des aktuellen Gestenobjekts abgefragt werden um auf unterschiedliche Gesten reagieren zu können. Sehr hilfreich ist hier die Switch-Case-Anweisung wie in Kapitel 2 beschrieben.
Nun kann man bei einer Wischgeste abfragen ob sie horizontal erfolgt ist, indem man eine neue SwipeGesture() mit der aktuellen Geste aus der foreach()-Schleife instanziiert und von dieser abfragt, ob SwipeGesture.Direction.x größer als SwipeGesture.Direction.y ist. Nach dieser Abfrage, kann man nun die links-rechts-Richtung überprüfen. Ist die Direction.x < 0, handelt es sich um eine Linksbewegung. Bei x > 0 ist es eine Wischbewegung nach rechts.

Mit diesen gesammelten Informationen kann man nun jeweils die Position der drei Canvas' (im Beispiel um 1000) in die entsprechende Richtung verschieben. Um einen schöneren Effekt zu erhalten, kann man statt des Versetzens auch mit für eine harmonische Bewegung mit Lerp arbeiten.

Wichtig ist außerdem, die verwendete Geste für eine gewisse Zeit (z.B. eine halbe Sekunde) zu sperren, damit man kein unvorhergesehenes Verhalten riskiert. Die dafür vorgesehene folgende Routine kann z.B. über den IEnumerator
StartCoroutine (DelayNextGesture (Gesture.GestureType.TYPESWIPE, 0.5f));
aufgerufen werden:

```
1 private IEnumerator DelayNextGesture(Gesture.
    GestureType type, float delay)
2   {
3     controller.EnableGesture(type, false);
4     yield return new WaitForSeconds(delay);
5     controller.EnableGesture(type);
6   }
7 }
```

Quelltext 5: Zeitweises blocken von Gesten

Der gesamte Quelltext ist in dieser Dokumentation als Anhang enthalten.

4 Weiterführende Informationen

Für eine Einführung und die ersten Schritte mit Leap Motion ist es sinnvoll, die C# SDK Documentation von Leap Motion zu Rate zu ziehen:

`https://developer.leapmotion.com/documentation/csharp/index.html`

Für den tieferen Einstieg, findet man alle Elemente in der C# API Reference:

`https://developer.leapmotion.com/documentation/unity/api/Leap_Classes.html`

5 Ausblick

Für Leap Motion gibt es sehr viele mögliche Anwendungsbereiche. Setzt sich die Technik durch, können Dateimanager, Anwendungssoftware und auch Betriebssysteme mit der Möglichkeit ausgestattet werden, sie intuitiv mit Gesten und durch Greifen und Verschieben von Elementen zu bedienen.
Einen Desktop dreidimensional anzeigen, mit plastischen Dateien und Ordnern, die man verschieben oder durch Wegwerfen löschen kann und scrollen und zoomen per Gesten sind nur einige Ideen die in diesem Bereich umgesetzt werden können.

Auch der Bereich der Spiele bietet sehr viele Möglichkeiten und es finden bereits einige Entwicklungen für Virtual Reality und Leap Motion statt. Ein denkbarer Ansatz im Bereich der Spiele ist, neben dem Umsetzen von Brettspielen wie Schach, Go, Mensch ärgere dich nicht etc., das Lösen von Rätseln zum freischalten von Leveln. Man könnte beispielsweise zum Öffnen von Türen das Lösen eines Codes durch Zusammensetzen von dreidimensionalen Puzzle-Teilen oder das Durchtrennen von Kabeln am Türöffnungsmechanismus vorgeben und Sprengkörper entschärfen lassen.

Durch Gestensteuerung ist es möglich, Spiele ähnlich wie mit Wii-Controllern zu kontrollieren. So kann man zum Beispiel Fahr- oder Flugzeuge steuern, indem man durch das Hin- und Herbewegen der Hände nach links und rechts steuert und durch Kippen der Hände nach vorn oder hinten beschleunigt bzw. bremst.

Anhang

```
1  using UnityEngine;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System;
5  using Leap;
6
7  public class DetectGesture : MonoBehaviour {
8
9    public Controller controller = new Controller();
10   public GestureList gestures = new GestureList();
11   public Canvas c1;
12   public Canvas c2;
13   public Canvas c3;
14
15   Vector3 c1start;
16   Vector3 c2start;
17   Vector3 c3start;
18
19   Frame frame;
20
21   private Vector3 swipeStart = Vector3.zero;
22   private Vector3 swipeEnd = Vector3.zero;
23
24   private int currentPos = 1;
25
26   void Start () {
27     controller.EnableGesture (Gesture.GestureType.
         TYPE_KEY_TAP);
28     controller.EnableGesture (Gesture.GestureType.
         TYPE_SWIPE);
29
30     c1start = c1.transform.position;
31     c2start = c2.transform.position;
32     c3start = c3.transform.position;
33   }
34
35   void Update () {
36
37     frame = controller.Frame ();
38     gestures = frame.Gestures ();
39
40     foreach (Gesture g in gestures) {
41       if (!g.IsValid)
```

```
42          continue;
43
44        if (g.Type == Gesture.GestureType.TYPESWIPE) {
45          Debug.Log ("SWYPE");
46
47          SwipeGesture sg = new SwipeGesture (g);
48
49          bool isHorizontal = Math.Abs (sg.Direction.x)
               > Math.Abs (sg.Direction.y);
50
51          if (isHorizontal) {
52            if (sg.Direction.x < 0f) {
53              Debug.Log ("left");
54              switch (currentPos) {
55              case 1:
56                currentPos = 2;
57                StartCoroutine ("swipeleft");
58                break;
59              case 2:
60                currentPos = 3;
61                StartCoroutine ("swipeleft");
62                break;
63              case 3:
64                currentPos = 3;
65                break;
66              }
67            } else if (sg.Direction.x > 0f) {
68              Debug.Log ("right");
69              switch (currentPos) {
70              case 1:
71                currentPos = 1;
72                break;
73              case 2:
74                currentPos = 1;
75                StartCoroutine ("swiperight");
76                break;
77              case 3:
78                currentPos = 2;
79                StartCoroutine ("swiperight");
80                break;
81              } // end switch
82            } // end else if "left"
83          } // end if Math
84
85          StartCoroutine (DelayNextGesture (Gesture.
               GestureType.TYPESWIPE, 0.3f));
```

```
86        }
87
88        if (g.Type == Gesture.GestureType.TYPEKEYTAP) {
89          Debug.Log ("SCREEN TAP");
90          switch (currentPos) {
91          case 1:
92            Application.OpenURL ("http://www.kino.de/
                film/the-revenant-der-rueckkehrer-2015/")
                ;
93            break;
94          case 2:
95            Application.OpenURL ("http://www.kino.de/
                film/zoomania-2016/");
96            break;
97          case 3:
98            Application.OpenURL ("http://www.kino.de/
                film/spotlight/");
99            break;
100         }
101
102         StartCoroutine (DelayNextGesture (Gesture.
              GestureType.TYPEKEYTAP, 1.0f));
103       }
104     } // end foreach gesture
105   } // end update
106
107   void swiperight () {
108     c1start.x += 1000;
109     c2start.x += 1000;
110     c3start.x += 1000;
111
112     c1.transform.position = c1start;
113     c2.transform.position = c2start;
114     c3.transform.position = c3start;
115   }
116
117   void swipeleft () {
118     c1start.x -= 1000;
119     c2start.x -= 1000;
120     c3start.x -= 1000;
121
122     c1.transform.position = c1start;
123     c2.transform.position = c2start;
124     c3.transform.position = c3start;
125   }
126
```

```
127   private IEnumerator DelayNextGesture(Gesture.
          GestureType type, float delay)
128   {
129     controller.EnableGesture(type, false);
130     yield return new WaitForSeconds(delay);
131     controller.EnableGesture(type);
132   }
133 }
```

Quelltext 6: Gesamtquelltext des Anwendungsbeispiels