Embodied interaction

Exam Mini-project

Luca Mangano
20125514
&
Emil Rosenlund Høeg
20125315

May 29, 2016

1 Paper Annotations

This section contains two short annotations of electable papers from the *International Workshop on Movement and Computing 2014/15* (MOCO). Both papers relates well to the overall purpose of the mini-project i.e. to offer an interactive platform for visualising the movement of painting, and offer variations depending on the expression. A paper was chosen by each group-member, and annotations were done individually.

Luca: *Collection and characterization of emotional body behaviors*

Fourati and Pelachaud [1] run a study in order to find solutions to the lack of techniques in order to categorize bodily expression of emotions. The most used and most affirmed coding systems is the Laban Movement Analysis (LMA), which has be) analyses. The study wanted to find a general and reliable system that based on different levels movement descriptions could analyze and deduce the emotional expression. As reported in figure 1, they divided the description levels in three: anatomical, directional, and posture/movement. these include different aspects of movement, posture and position. Anatomical refers to the physical state of different body parts such as upper/lower body, directional refers to the displacement in which the movement is taking place, such as front/back, right/left in all three dimensions, lastly posture/movement considers the postural status and changes that take place.

They used actors in order to record daily actions in different emotional states. They had actors enact in joy, anger, panic, fear, anxiety, sadness, shame, pride and neutral. They finally used motion capture recordings in order to have participants classify the different enactments and thus evaluate the movement-description system.

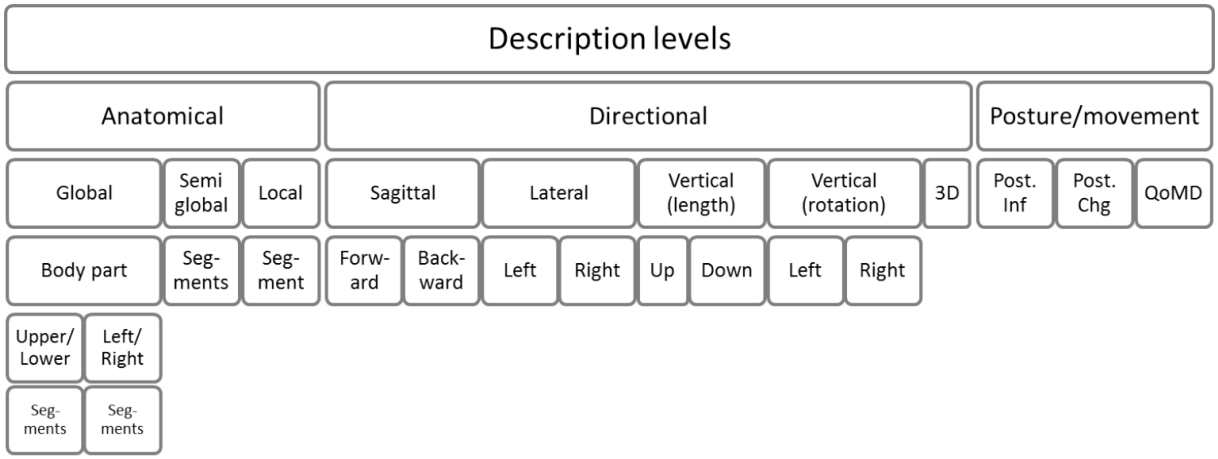

Figure 1: Quality of movement level description

Emil: *Interactive Movement Analytics Platform*

The paper *Mova: Interactive Movement Analytics Platform* presents a prototype (dubbed Mova) for an interactive movement analytics platform with the aim of providing a computational framework that facilitates movement information representation and analysis[2]. The system is initially purposely designed to convert motion-capture (mocap) recordings and map them onto a virtual skeleton whereas motion of individual joints can be visualised using segmentation and feature extraction. The visualisation facilitates simple kinematic characteristics (Speed, acceleration, jerk), as well as Laban effort parameters (i.e. time, space, weight and flow), gestures (e.g. swipe, drawing, shapes) and even emotions extrapolated from detection of posture and quality of movement. They supply pre-recordings of feature extraction and analysis of punch, dab, flick, etc (i.e. Laban's eight efforts).

The Mova system accepts a small selection of mocap-files *inter alia* BVH (Biovision hierarchical data), Acclaim and Collada. The program converts the file to an internal format with as high latency as 160 FPS (frames-per-second), but to facilitate a tractable feature-extraction a subset of frames is selected through either linear sampling k-th frame selected by the user (i.e. Skip Frames and padding) or by keyframe extraction.

The Graphical user interface (GUI) constructed by the team ultimately serves as an interactive platform for analysis and parallel visualisation of movement. The team suggests a few future improvements including automatic feature extraction, support for combining multi-model movement data and adding annotations to key-movement from the recordings.

The tool developed by Alemi et al [2] can be accessed online[1] and files can be uploaded and viewed directly. Unfortunately the tool only accepts .BVH files (biovision hierarchical data), and as of now, Leap data cannot be converted into BVH, even though the advantages of this is obviously apparent e.g. can it be used for quick animations and even real-time inverse kinematics.

[1]Mova-tool link:`http://www.sfu.ca/~oalemi/mova/`

2 Miniproject

The miniproject we developed was aimed to enable users draw in 3D using leap motion. On top of this, the movement quality would affect the stroke and color users were using.

The approved project description reads:

> *Create a 3D painting program using leap motion in unity, that can detect the quality of the movements and change strokes and paint thickness accordingly.*

The quality of movement mentioned in the description is defined according to the Laban Movement Analysis-model. More specifically the Effort-category subdivided into space, weight, time and flow) factors with opposing elements i.e. *Fighting polarity* and *Indulging polarity*. An example of this is free (fluid, uncontrollable, open-hearted vs. bound (rigid, contained, controlled) movement. See the effort graph below:

Laban Effort Graph

Direction/space direct or indirect

Weight Heavy or light

Speed Quick or sustained

Flow Bound or free

Although the LMA is a theoretical and experimental system for observing the body movement holistically, it has previously been used in user experience design heuristics[2], and considering the basic efforts defined by Ludolf Laban, often requires full body movement, the key-components are hand-gestures (e.g. punch, slash, flick). In a previous study automatic LMA was achieved using a Wearable Acceleration Unit attached to the user's wrist to assess the recorded expressive qualities of movement[3].

A standard example Unity scene was used as a starting point; this scene[3] would allow users to draw lines when the thumb was at a certain distance from the hand and subsequently music could be played. The scene was stripped down of the superfluous excess (i.e. music and lighting interaction) and the basic functionalities of "pinch-and-draw" were kept. Based on the papers reviewed and the movement workshop we had on April 14-15, simple movement qualities were created. An initial distinction was drawn between calm and aggressive movements; relating aggressiveness with the hand posture of a fist, and calm to a single finger, it was decided to increase the size of the brush when the users would draw more aggressively. This was not proven to have a meaningful enough impact, thus the shape of the brush was modified as well. A smoother (rounder) brush was used to represent calm, while a more edgy shape (square) was used to represent aggressiveness. Movement meaning was then taken into consideration, and the speed was used in order to affect the size of the brush so to have a larger footprint when not moving. This was motivated by the fact that when drawing faster, the movement can be considered more evasive and dynamic. Color modification was also implemented in the system in order to reflect the emotions intended. Blue was

[2] `http://www.uxmatters.com/mt/archives/2010/02/laban-movement-analysis-for-user-experience-design.php`

[3] Available at: `https://github.com/leapmotion/MusicalFingerPaint`

used as base color semiotic, since in western general culture it is used to indicate stability and peace, it was associated to low speed. On the other hand, red (usually associated with passion and intensity) was used at high speed as the opposite extreme of blue. In between these two colors, shades of blue, purple, and red were adopted so to ease the passage between the two extremes and still convey the emotion reflected in the drawing. Color drawing can be disabled, but the only color change that will take place will be between black for standard drawing and red for "fist" drawing to still reflect the aggressiveness of the movement. The source-code is available in the appendix, and the project can be accessed at Github[4]

2.1 Technical specifications

The painting program utilizes Unity3D v5.3.3f1, with code written in C#, and a Leap Motion 010 with v2 SDK. the new Orion SDK does not yet support development on iOS, and the painting program has to be accessible from both operating systems.

2.2 Future Prospects

The Leap Motion painter offers a unique 'quality of movement' response. Compared to the Mova-tool it is, however, of a more abstract character. Where the Mova offers a wide selection of analysis-ready tools, the Leap-painter is generating a visual response to user-input such as speed, velocity and gesture. As of now, Unfortunately the tool only accepts .BVH files (biovision hierarchical data),and as of now, Leap data cannot be converted into BVH, even though the advantages of this is obviously apparent e.g. can it be used for quick animations and even real-time inverse kinematics. The Mova does not currently support real-time analysis or feedback, but ultimately we would like to implement an interactive Laban model that can offer a direct real-time response i.e. in accordance with the Laban qualities of movement. An interactive model could ultimately also be used for full body capture of e.g. dance and could be used as evaluation-tool in educational scenarios.

References

[1] N. Fourati and C. Pelachaud, "Collection and characterization of emotional body behaviors," in *Proceedings of the 2014 International Workshop on Movement and Computing*, p. 49, ACM, 2014.

[2] O. Alemi, P. Pasquier, and C. Shaw, "Mova: Interactive movement analytics platform," in *Proceedings of the 2014 International Workshop on Movement and Computing*, p. 37, ACM, 2014.

[3] D. Silang Maranan, S. Fdili Alaoui, T. Schiphorst, P. Pasquier, P. Subyen, and L. Bartram, "Designing for movement: evaluating computational models using lma effort qualities," in *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, pp. 991–1000, ACM, 2014.

[4]Leap Painter source code: `https://github.com/LucaMangano/embodied`

CanvasAutomation.cs

```
using UnityEngine;
using System.Collections.Generic;
using Leap;
using System;

public class CanvasAutomation : MonoBehaviour {

  public Material material_;
  public Material mat0;
  public Material mat1;
  public Material mat2;
  public Material mat3;
  public Material mat4;
  public Material mat5;
  public Material mat6;

  private bool colorEnable;

  private const int INTERPOLATION_AMT = 3;
  private const int MAX_POINTS = 500; //cannot go higher, memory problems
  private const int POINT_MEMORY = 50;
  private const int TUBE_FACES = 50;
  private const int VERTEX_NUM = TUBE_FACES * 2;
  private const int TRIANGLE_NUM = TUBE_FACES * 6;
  private const float MULTI_HAND_PHASE = 0.1f;

  //size of the brush and the multiplier for a more direct size change
  int size = 50;
  float multiplier_size = 1f;

  private bool FIST = false;

  private Mesh mesh_;

  private int point_index_ = 0;

  private float line_width_ = 0.6f;
  private float current_line_width_ = 0.0f;
  private float multi_hand_phase_ = 0.0f;

  private Vector3 last_velocity_;
  private Vector3 last_drawn_point_;
  private Vector3[] drawn_points_;
  private Vector3[] last_points_;
  private Vector3[] last_shape_;

  private bool drawing_;
  private List<Vector3> next_points_;

  void Start () {
    colorEnable = false;
    mesh_ = new Mesh();
      drawing_ = false;

      next_points_ = new List<Vector3>();
      drawn_points_ = new Vector3[MAX_POINTS];
      last_points_ = new Vector3[POINT_MEMORY];
      last_shape_ = null;
```

```
59  }
60
61  //called from the drawer.cs file, used to draw when pinching
62  public void AddNextPoint(Vector3 next, bool fist) {   //get the input
     for fist as well
63    if (!drawing_) {
64      for (int i = 0; i < POINT_MEMORY; ++i)
65        last_points_[i] = next;
66      last_drawn_point_ = next;
67      last_velocity_ = Vector3.zero;
68    }
69
70    if (fist)
71      multiplier_size = 2f;
72    else
73      multiplier_size = 0.5f;
74
75    FIST = fist;
76
77    next_points_.Add(next);
78    drawing_ = true;
79  }
80
81  //lateupdate used instead of update
82  void LateUpdate() {
83    drawing_ = next_points_.Count != 0;
84
85    UpdateNextPoint();
86    Graphics.DrawMesh(mesh_, transform.localToWorldMatrix, material_, 0);
87  }
88
89  //external source
90  private void UpdateNextPoint() {
91    if (drawing_) {
92      for (int i = POINT_MEMORY - 1; i > 0; i--)
93        last_points_[i] = last_points_[i - 1];
94
95      Vector3 point1 = next_points_[0];
96      Vector3 point2 = next_points_[1 % next_points_.Count];
97      float t = Mathf.Sin(multi_hand_phase_) / 2.0f + 0.5f;
98      last_points_[0] = point1 + t * (point2 - point1);
99      multi_hand_phase_ += MULTI_HAND_PHASE;
100     next_points_.Clear();
101   }
102
103   for (int i = 1; i <= INTERPOLATION_AMT; ++i) {
104     IncrementIndices ();
105
106     Vector3 draw_point = InterpolatedPoint((1.0f * i) /
     INTERPOLATION_AMT);
107     last_velocity_ = last_velocity_ + 0.4f * (draw_point -
     last_drawn_point_ - last_velocity_);
108
109     if (drawing_) {
110       last_shape_ = MakeShape(draw_point);
111       AddLine(mesh_, last_shape_);
112       drawn_points_[point_index_] = draw_point;
113     }
114     else {
115       last_shape_ = null;
```

```
116         current_line_width_ = 0.0f;
117       }
118       last_drawn_point_ = draw_point;
119     }
120   }
121
122   //external source
123   void IncrementIndices() {
124     point_index_ = (point_index_ + 1) % MAX_POINTS;
125   }
126
127   //esternal source
128   private Vector3 InterpolatedPoint(float t) {
129     Vector3 naut = last_points_[0] * (t + 2) * (t + 1) * t / 6.0f;
130     Vector3 one = -last_points_[1] * (t + 2) * (t + 1) * (t - 1) / 2.0f;
131     Vector3 two = last_points_[2] * (t + 2) * t * (t - 1) / 2.0f;
132     Vector3 three = -last_points_[3] * (t + 1) * t * (t - 1) / 6.0f;
133     return naut + one + two + three;
134   }
135
136   //used to generate the mesh that will be drawn
137   Vector3[] MakeShape(Vector3 next) {
138     Vector3[] q = new Vector3[VERTEX_NUM];
139     Vector3 radius = new Vector3();
140
141     //alternative variable to TUBE_FACES (constant), for the amount of
      sides of the tube
142     float t = TUBE_FACES;
143
144     //change radius according to speed, size is proportional to velocity
      with which width is computed
145     if (!FIST) {
146       float compute_width = (1.0f / (size * last_velocity_.magnitude +
      1.0f)) * line_width_ * multiplier_size;
147       current_line_width_ = current_line_width_ + 0.1f * (compute_width -
      current_line_width_);
148       radius = new Vector3 (0.0f, current_line_width_ / 2.0f, 0.0f);
149     } else {
150       //the width resize with the fist is affected by 1/5 of the velocity
      instead of 1/1 of normal drawing
151       float compute_width = (1.0f / (size * last_velocity_.magnitude +
      1.0f)) * line_width_ * multiplier_size;
152       current_line_width_ = current_line_width_ + 0.1f * (compute_width -
      current_line_width_);
153       radius = new Vector3 (0.0f, current_line_width_ / 2.0f, 0.0f);
154       t = 4;
155     }
156
157     //depending on the status of colorEnable, either color the mesh with
      different colors, or use the standard black material
158     if (colorEnable) {
159       //change color according to speed
160       float d = last_velocity_.magnitude * 5000;
161       print (d);
162       if (d <= 200) {
163         material_ = mat1;
164       } else if (d > 200 && d <= 500) {
165         material_ = mat2;
166       } else if (d > 500 && d <= 800) {
167         material_ = mat4;
```

```
168       } else if (d > 800 && d <= 1200) {
169         material_ = mat5;
170       } else {
171         material_ = mat6;
172       }
173     } else {
174       if (FIST) {
175         material_ = mat6;
176       } else{
177         material_ = mat0;
178       }
179     }
180 
181 
182     Quaternion rotation = Quaternion.AngleAxis(360.0f / t, new Vector3(0,
     0, 1));
183 
184     if (last_shape_ == null) {
185       for (int i = 0; i < TUBE_FACES; ++i)
186         q[i] = transform.InverseTransformPoint(last_drawn_point_);
187     }
188     else {
189       for (int i = 0; i < TUBE_FACES; ++i)
190         q[i] = transform.InverseTransformPoint(last_shape_[TUBE_FACES +
     i]);
191     }
192 
193     for (int i = 0; i < TUBE_FACES; ++i) {
194       q[TUBE_FACES + i] = transform.InverseTransformPoint(next + radius);
195       radius = rotation * radius;
196     }
197 
198     return q;
199   }
200 
201   //used to add lines in the mesh - external source
202   void AddLine(Mesh m, Vector3[] shape) {
203     int vertex_index = point_index_ * VERTEX_NUM;
204     int triangle_index = point_index_ * TRIANGLE_NUM;
205     // Update Vertices.
206     Vector3[] vertices = m.vertices;
207     if (vertex_index >= m.vertices.Length) {
208       Array.Resize(ref vertices, MAX_POINTS * VERTEX_NUM);
209     }
210 
211     for (int i = 0; i < VERTEX_NUM; ++i) {
212       vertices[vertex_index + i] = shape[i];
213       float t = (1.0f * vertex_index + i) / (MAX_POINTS * VERTEX_NUM);
214     }
215 
216     // Update triangles.
217     int[] triangles = m.triangles;
218     if (triangle_index >= m.triangles.Length)
219       Array.Resize(ref triangles, MAX_POINTS * TRIANGLE_NUM);
220 
221     for (int i = 0; i < TUBE_FACES; ++i) {
222       int t_index = triangle_index + i * 6;
223       triangles[t_index] = vertex_index + i;
224       triangles[t_index + 1] = vertex_index + i + TUBE_FACES;
225       triangles[t_index + 2] = vertex_index + (i + 1) % TUBE_FACES;
```

```
      triangles[t_index + 3] = vertex_index + TUBE_FACES + (i + 1) %
   TUBE_FACES;
      triangles[t_index + 4] = vertex_index + (i + 1) % TUBE_FACES;
      triangles[t_index + 5] = vertex_index + i + TUBE_FACES;
    }

    m.vertices = vertices;
    m.triangles = triangles;
    m.RecalculateBounds();
  }

  public Vector3 GetLastPoint() {
    return drawn_points_[point_index_];
  }

  //used to toggle between colored and black versions of drawing
  public void ToggleColor(){
    if (colorEnable)
      colorEnable = false;
    else
      colorEnable = true;
  }
}
```

Drawer.cs

```
using UnityEngine;
using System.Collections;
using Leap;
using System;

public class Drawer : MonoBehaviour {

  public CanvasAutomation canvas;

  private const float THUMB_TRIGGER_DISTANCE = 0.04f;
  private const float MIN_CONFIDENCE = 0.2f;


  private float FIST_TRIGGER_DISTANCE = 0.145f;
  private float temp_DIST = 0f;

  void Update () {
    HandModel hand_model = GetComponent<HandModel>();
    Hand leap_hand = hand_model.GetLeapHand();

    if (leap_hand == null)
      return;

    bool draw_trigger = false;
    bool draw_fist = false;

    Vector3 thumb_tip = leap_hand.Fingers[0].TipPosition.ToUnityScaled();

    //draw if one finger is close to the thumb
    for (int i = 1; i < 5 && !draw_trigger; ++i) {
      for (int j = 0; j < 4 && !draw_trigger; ++j) {
        Finger.FingerJoint joint = (Finger.FingerJoint)(j);
        Vector3 difference =
   leap_hand.Fingers[i].JointPosition(joint).ToUnityScaled() - thumb_tip;
```

```
34          if (difference.magnitude < THUMB_TRIGGER_DISTANCE &&
    leap_hand.Confidence > MIN_CONFIDENCE) {
35            draw_trigger = true;
36          }
37        }
38      }
39
40      //draw if you have a fist
41      for (int k = 1; k < 5; k++) {
42        Finger.FingerJoint joint = (Finger.FingerJoint)(k);
43        Vector3 difference =
    leap_hand.Fingers[k].JointPosition(joint).ToUnityScaled() - thumb_tip;
44        temp_DIST += difference.magnitude;
45      }
46
47      if (temp_DIST <= FIST_TRIGGER_DISTANCE) {
48        draw_fist = true;
49      }
50
51      temp_DIST = 0f;
52
53      //it checks if the distance between the thumb and other fingers is
    less than a threshold and if yes
54      //then it allows to draw
55      if (draw_trigger)
56
    canvas.AddNextPoint(hand_model.fingers[1].GetJointPosition(FingerModel.NUM_JOINTS
    - 1), draw_fist);
57    }
58  }
```

MouseOrbit.js

```
1  //external source used to move the explore the scene with the mouse
       (rotate/move camera)
2  var target : Transform;
3  var distance = 10.0;
4
5  var xSpeed = 250.0;
6  var ySpeed = 120.0;
7
8  var yMinLimit = -20;
9  var yMaxLimit = 80;
10
11 private var x = 0.0;
12 private var y = 0.0;
13
14 @script AddComponentMenu("Camera-Control/Mouse Orbit")
15
16 function Start () {
17   var angles = transform.eulerAngles;
18   x = angles.y;
19   y = angles.x;
20
21   if (GetComponent.<Rigidbody>())
22     GetComponent.<Rigidbody>().freezeRotation = true;
23 }
24
25 function LateUpdate () {
26   if (target) {
```

```
    x += Input.GetAxis("Mouse X") * xSpeed * 0.02;
    y -= Input.GetAxis("Mouse Y") * ySpeed * 0.02;

    y = ClampAngle(y, yMinLimit, yMaxLimit);

    var rotation = Quaternion.Euler(y, x, 0);
    var position = rotation * Vector3(0.0, 0.0, -distance) +
   target.position;

    transform.rotation = rotation;
    transform.position = position;
  }
}

static function ClampAngle (angle : float, min : float, max : float) {
  if (angle < -360)
    angle += 360;
  if (angle > 360)
    angle -= 360;
  return Mathf.Clamp (angle, min, max);
}
```

PointFollow.cs

```
using UnityEngine;
using System.Collections;

public class PointFollow : MonoBehaviour {

  void Update () {
    transform.position =
   GameObject.Find("Canvas").GetComponent<CanvasAutomation>().GetLastPoint();
  }
}
```