Konzept

Für eine Interaktive Medieninstallation
der Fakultät Digitale Medien an der Hochschule Furtwangen

Arbeitstitel:

Mirror Me

vorgelegt von
Johann Schulenburg
Laura Moser

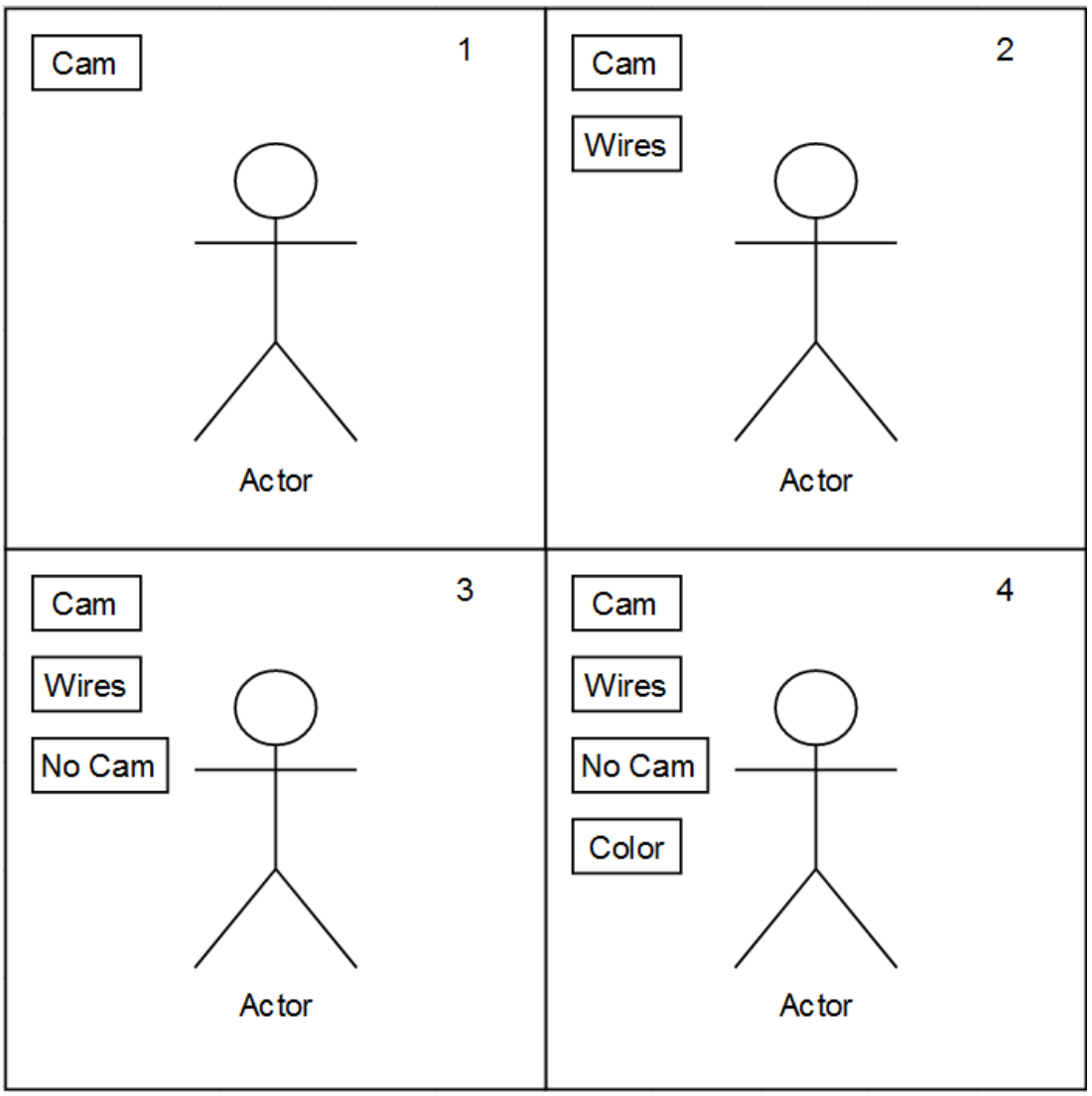
1. Einführung

Für unsere Interaktive Medieninstallation haben wir uns von bereits existierenden Interaktionen inspirieren lassen.
Die WE ARE BODY™ G+B GmbH und Lang AG haben als Werbemittel eine Interaktive Motion-Capture-Installation erstellt.
Hier konnten die Gäste ein Bewegungserkennungs- Verfahren zur digitalen Erfassung von Silhouetten testen und so mit ihrem digitalen Körper interagieren, der aus Tausenden von Partikeln in Ihren Farben besteht.
An dieses Projekt haben wir unsere Interaktive Medieninstallation entlang konzipiert.
Daher haben wir, Johann Schulenburg und Laura Moser, uns Im Rahmen der Interaktiven Medieninstallation Vorlesung für das Projekt “Mirror Me” entschieden.

WE ARE BODY™

2. Konzept

2.1. Skizze



2.2. Hintergrund

Im Rahmen der Wahlpflichtveranstaltung “Interaktive Medieninstallationen” haben Laura Moser und Johann Schulenburg ein Thema zu einer Installation gesucht, die sie umsetzen konnten. Eine dieser Ideen war eine Installation, in der der Rezipient wie vor einem Spiegel steht und mit diesem interagiert. Das erste Konzept dazu war, den Nutzer denken zu lassen, er kommuniziere mit einer künstlichen Intelligenz, indem eine zweite Person ohne des Wissens der ersten die Bewegungen dieser nachahmt und sich gelegentlich anders verhält. Diese Idee haben wir verworfen, da wir bemerkten, dass sich eine nachahmende Person niemals schnell genug bewegen könnte.

2.3. Standort und Form

Die Installation soll auf einem frei zugänglichen Platz stehen und vorbeilaufende Personen sollen durch die unten aufgeführten Mittel auf die

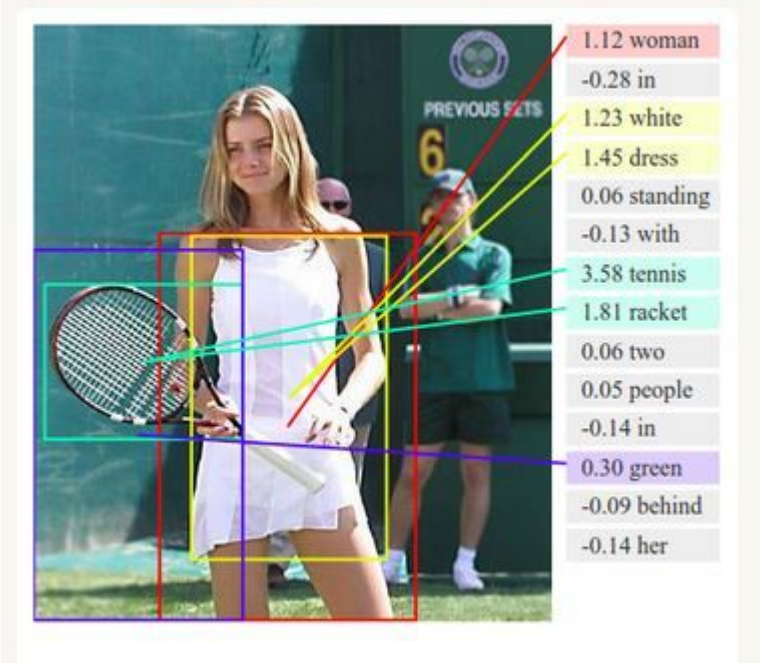
Installation aufmerksam gemacht werden. Realisiert soll die Installation mithilfe eines Beamers, der eine Leinwand von hinten bestrahlt. Auf der Leinwand wird dann das “Spiegelbild” der Person der vor Ihr steht abgebildet. Dieses Spiegelbild wird mit Hilfe einer Kamera und eines Computers produziert.

2.3.1. Aufmerksamkeits-Akquirierung

Bei dem vorbeilaufen soll der Rezipient bemerken, dass er gefilmt und auf der Leinwand abgebildet wird. Kommt er nun näher und bleibt dort für etwa 5 Sekunden stehen wird die “erste Stufe”, also die Körpererkennung gestartet.

2.4. Konzeptbilder

Konzeptidee:



3. Umsetzung

Nachdem wir uns für das Projekt “Mirror Me” entschieden haben, konnten wir direkt mit der Recherche und der Planung beginnen.

Mirror Me			Johann	Laura	21.04.22 - 30.06.22									
					W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
1.	Konzept Erarbeitung													
1.1	Code für den Kamera Input													
1.2	Recherche über Kompatibilität													
1.3	Präsentation erstellen und Information zusammentragen													
1.4	# Zwischenpräsentation 12.05.22													
	# Recherche steht und praktischer Teil beginnt													
2.	Datenverarbeitung für Animation													
2.1	Anfangsscreen erstellen													
2.2	Hardware beschaffen uns zusammenstellen													
2.3	Erster Testlauf													
2.4	Verbesserungen													
	# Installation kann aufgestellt werden													
3.	....													
3.1	Präsentation erstellen + Abgabe													
3.2	# Abschlusspräsentation 30.06.22													

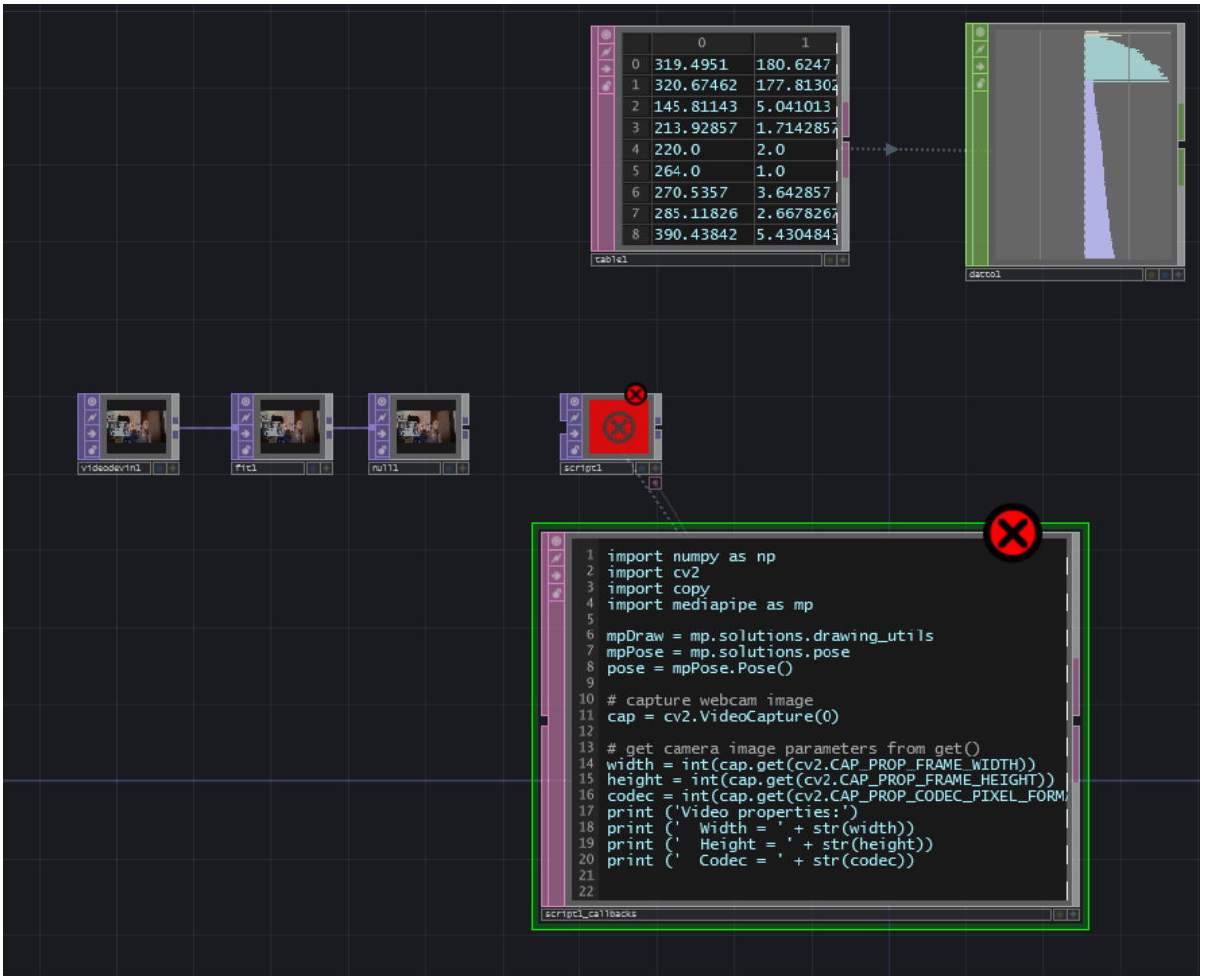
Mithilfe von verwandten Projekten konnten wir eine 3D-Posenschätzung programmieren, die den Kamera Input in Echtzeit verarbeitet und die genauen Standorte der Landmarks ausgibt.
Hierfür haben wir die Bibliothek OpenCV und Mediapipe genutzt.[1]

Da wir nun den Input der Kamera bzw. die Landmarks weiterverarbeiten und visuell umgestalten möchten, benötigen wir ein passendes Tool.
Nach unserer Recherche haben wir uns für die Tools Unity und Touchdesigner entschieden.
Die Tools haben wir jeweils aufgeteilt, sodass die kompatibilität mit unserem Projekt getestet werden kann.

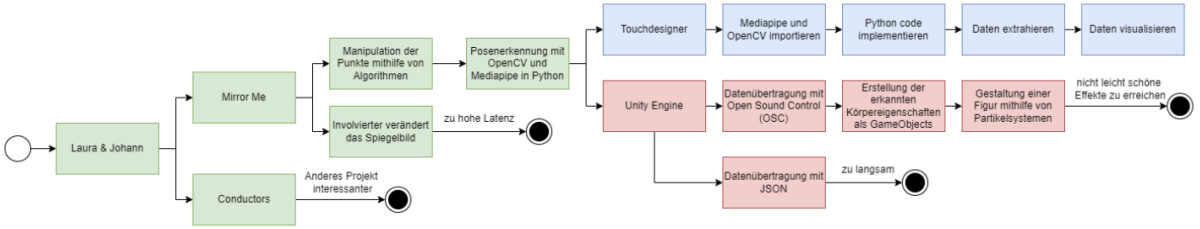
Da bereits Erfahrung in Unity in das Projekt mitgebracht wurde, konnte hier direkt mit der Datenübertragung gestartet werden.
Hierfür wurde die erste Version des Codes umgewandelt, sodass die Landmarks in einem passenden Format ausgegeben werden konnten.

Gleichzeitig wurde das Tool Touchdesigner installiert und die nötigen Einstellungen eingerichtet.
Da Touchdesigner knotenbasiert ist, kann direkt auf den Kamera Input zugegriffen werden. Mithilfe des Python Codes, der ebenso direkt in Touchdesigner integriert werden kann, kann der Input in Echtzeit verändert werden.

[1] 3D-Posenschätzung (siehe Code)



Trotz der mangelnden Erfahrung und Startschwierigkeiten in Touchdesigner, haben wir uns für dieses Tool entschieden.
Bei Unity sind wir an dem Ende der technischen Möglichkeiten gelangt, sodass die weiterführung des Projekts schwieriger wurde.
Mit Touchdesigner haben wir allerdings deutlich mehr Möglichkeiten das Projekt nach unseren Wünschen zu gestalten.



4. Hürden

Während der Umsetzung des Projektes sind wir auf einige Hürden gestoßen, die wir bewältigen mussten.

Da Unity selbst keinen Python Code verarbeiten kann, mussten die erhaltenen Landmarks entsprechend umgewandelt werden, um sie so in Unity verwenden zu können.
Wie oben bereits beschrieben, haben wir hierfür das Open Sound Control Netzwerkprotokoll verwendet, um die Schnittstelle zwischen Unity und Visual Studio Code herzustellen.
Da wir zuvor noch nicht mit diesem Protokoll gearbeitet und bis jetzt keine Schnittstelle zwischen zwei Tools erstellt haben, hatte wir hierbei Schwierigkeit, die wir allerdings bewältigen konnten.

Eine zweite Hürde war das Tool Touchdesigner. Bis zu diesem Projekt hatten wir beide noch keine Kontaktpunkte mit der knotenbasierten visuellen Programmiersprache. Aufgrund dessen mussten wir uns erst einmal in dem Tool zurecht finden und Installationseinstellungen bewältigen.

Zusätzlich war es schwierig konzeptionelle und auch technische Entscheidungen zu treffen. Grundsätzlich fiel uns schwer den Hintergrund und die Intention der Installation zu definieren.

5. Code

1. 3D-Posenschätzung

```
import numpy as np
import cv2
import copy
import mediapipe as mp

mpDraw = mp.solutions.drawing_utils
mpPose = mp.solutions.pose
pose = mpPose.Pose()

# capture webcam image
cap = cv2.VideoCapture(0)

# get camera image parameters from get()
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
codec = int(cap.get(cv2.CAP_PROP_CODEC_PIXEL_FORMAT))
print ('Video properties:')
print ('  Width = ' + str(width))
print ('  Height = ' + str(height))
print ('  Codec = ' + str(codec))


while True:
    # get video frame (always BGR format!)
    ret, frame = cap.read()
    if (ret):
        # copy image to draw on
        img = copy.copy(frame)

        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        results = pose.process(imgRGB)
        if results.pose_landmarks:
            mpDraw.draw_landmarks(img,
results.pose_landmarks,mpPose.POSE_CONNECTIONS)
            for id, lm in
enumerate(results.pose_landmarks.landmark):
                h, w, c = img.shape
                print(id, lm)
                cx, cy = int(lm.x * w), int(lm.y * h)
```

```
        # show the original image with drawings in one window
        cv2.imshow('Original image', frame)
        cv2.imshow('masked image',img)
        # deal with keyboard input
        key = cv2.waitKey(10)
        if key == ord('q'):
            break
    else:
        print('Could not start video camera')
        break

cap.release()
cv2.destroyAllWindows()
```

2. Unity Code

```
from re import X
import numpy as np
import cv2
import copy
import mediapipe as mp
import argparse
import random
import time
from pythonosc import udp_client

mpDraw = mp.solutions.drawing_utils
mpPose = mp.solutions.pose
pose = mpPose.Pose()

# capture webcam image
cap = cv2.VideoCapture(0)
#cap = cv2.VideoCapture('WorkItWillis.mp4')
# get camera image parameters from get()
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
codec = int(cap.get(cv2.CAP_PROP_CODEC_PIXEL_FORMAT))
print ('Video properties:')
print ('  Width = ' + str(width))
print ('  Height = ' + str(height))
print ('  Codec = ' + str(codec))

parser = argparse.ArgumentParser()
parser.add_argument("--ip", default="127.0.0.1", help="The ip of
the OSC server")
parser.add_argument("--port", type=int, default=7001, help="The
port the OSC server is listening on")
args = parser.parse_args()

client = udp_client.SimpleUDPClient(args.ip, args.port)


while True:
    # get video frame (always BGR format!)
    ret, frame = cap.read()
    if (ret):
        # copy image to draw on
        img = copy.copy(frame)
```

```
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        results = pose.process(imgRGB)
        if results.pose_landmarks:
            mpDraw.draw_landmarks(img,
results.pose_landmarks,mpPose.POSE_CONNECTIONS)
            for id, lm in
enumerate(results.pose_landmarks.landmark):
                h, w, c = img.shape
                if lm.visibility>0.7:
                    client.send_message(f"/message/landmark{id}",
f"{id};{lm.x};{lm.y};{lm.z}")
                cx, cy = int(lm.x * w), int(lm.y * h)


        # show the original image with drawings in one window
        cv2.imshow('Original image', img)

        # deal with keyboard input
        key = cv2.waitKey(10)
        if key == ord('q'):
            break
    else:
        print('Could not start video camera')
        break

cap.release()
cv2.destroyAllWindows()
```