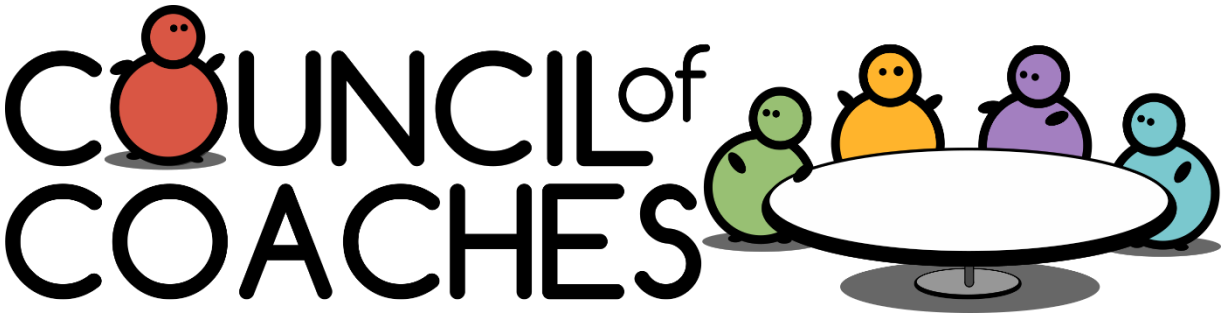

D4.3: Short-term behaviour analysis prototype

Dissemination level:	Public
Document type:	Demonstrator
Version:	1.0.1
Date:	November 29, 2018 (original)
	March 5, 2019 (this version)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement #769553. This result only reflects the author's view and the EU is not responsible for any use that may be made of the information it contains.

Document Details

Project Number	769553
Project title	Council of Coaches
Title of deliverable	Short-term behaviour analysis prototype
Due date of deliverable	November 30, 2018 (M12)
Work package	WP4
Author(s)	Oresti Banos (CMC), Kostas Konsolakis (CMC), Reshmashree Bangalore Kantharaju (SU), Catherine Pelachaud (SU), Harm op den Akker (RRD)
Reviewer(s)	Alvaro Fides Valero (UPV), María Martinez (UPV)
Approved by	Coordinator
Dissemination level	Public
Document type	Demonstrator
Total number of pages	25

Partners

- University of Twente – Centre for Monitoring and Coaching (CMC)
- Roessingh Research and Development (RRD)
- Danish Board of Technology Foundation (DBT)
- Sorbonne University (SU)
- University of Dundee (UDun)
- Universitat Politècnica de València, Grupo SABIEN (UPV)
- Innovation Sprint (iSPRINT)

Abstract

This deliverable (D4.3) describes the software implementation of the methods developed in D4.2 for the inference of short-term behaviours from physical and virtual sensor data. This deliverable is further accompanied with two demonstrators that illustrate the operation of the Holistic Behaviour Analysis Framework (HBAF) for the recognition of some exemplary short-term behaviours in both outdoor and indoor scenarios. The outdoor scenario demonstrates the operation and capabilities of the system while automatically detecting physical activity and social behaviours from smartphone data ("on-body sensors") for various daily life situations. The indoor scenario showcases the functioning of the system for the detection of emotional and cognitive behaviours from video and audio data captured via Kinect and a microphone ("off-body sensors") for a user-council interaction situation.

Corrections

v1.0.1 Correctly applied EU logo on header page.

Table of Contents

List of figures

List of tables

Symbols, abbreviations and acronyms

ACC	Accelerometer
AU	Action Unit
CMC	Centre for Monitoring and Coaching
COUCH	Council of Coaches
D	Deliverable
EC	European Commission
FFT	Fast Fourier Transform
GPS	Global Positioning System
GSR	Galvanic Skin Response
HBAF	Holistic Behaviour Analysis Framework
HCI	Human-Computer Interaction
KNN	k-Nearest Neighbour
M	Month
Max	Maximum
Min	Minimum
MS	Milestone
PPG	Photoplethysmography
RF	Random Forest
RMS	Root Mean Square
RMSE	Root Mean Square Error
RRD	Roessingh Research and Development
STD	Standard Deviation
SU	Sorbonne University
SVM	Support Vector Machine
WP	Work Package
UDI	Unique Device Identifier
UUI	Unique User Identifier
UPV	Universitat Politècnica de València
UT	University of Twente

1 Introduction

After presenting the methods used for measuring and modelling short-term behaviours (a.k.a. primitives or momentary behaviours), by explaining the selected types of sensor data and the techniques for behaviour analysis in D4.2, this deliverable deepens more on the software components. In particular, the current D4.3 document aims to further elaborate the developed short-term behaviour inference methods regarding the automatic processing of on-body and off-body sensor data.

Initially, the technical setup including the hardware components and the system's architecture is presented in section 3. Then, the developed models for short-term physical, social, emotional and cognitive behaviours are identified and explained in section 4. Finally, section 5 shows the developed demos for various scenarios.

2 Objectives

The main objective of this deliverable (D4.3) is to describe the software components developed for the inference of short-term behaviours based on multimodal sensor data (D4.2). This deliverable also includes two demonstrators to showcase the main features implemented for the COUCH 2nd Functional Prototype (D7.3).

3 Technical Setup

3.1 Hardware

The 'on-body sensors' hardware setup mainly consists of the smartphone sensors and the data collection platform (AWARE), which confer monitoring capabilities to the Holistic Behaviour Analysis Framework (HBAF). In order to collect the smartphone sensor data only Android devices are used, where a unique smartphone device is referred to a single user. Initially, the user behaviour is unobtrusively measured via a client application (AWARE Client), which monitors and collects the data, and then, the acquired data are uploaded to the HBAF server through a secured connection. The smartphone application is presented in Figure 1. Different types of sensor data are acquired, including accelerometer and GPS signals for tracking physical activity, while Bluetooth, ambient noise, location and phone usage metrics (e.g., number of incoming/outcoming calls, number of received/sent text messages) are recorded to detect the user's level social activeness.

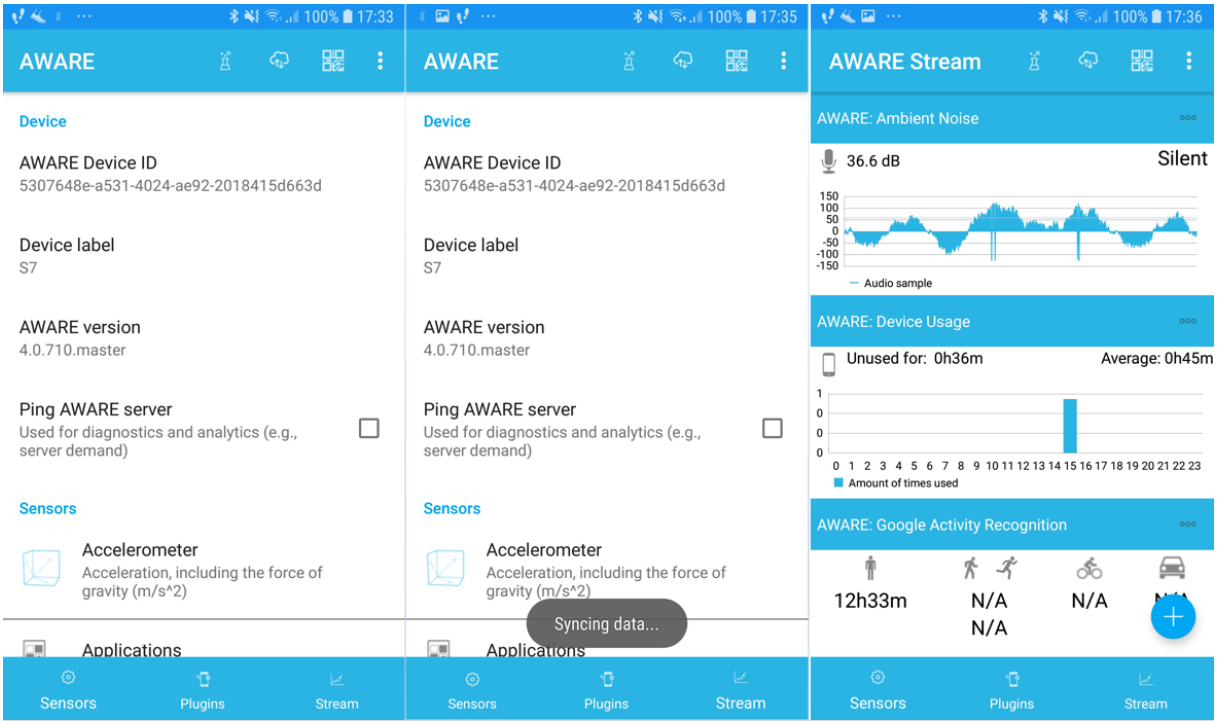

Figure 1: Snapshots from the AWARE client application (smartphone).

The 'off-body sensors' hardware setup includes a display screen, a Kinect sensor, close-to-mouth microphone and a PC. We made use of a full-length LCD display screen to display the Greta agent, Alice in human scale (see Figure 2). The screen is connected to a PC, which runs the Greta platform. A Kinect sensor is mounted on top of the screen. The Kinect, released by Microsoft, uses actively emitted structured infrared (IR) light to estimate depth at each pixel using a single IR sensitive camera. The Kinect also contains a standard RGB (colour) camera which provides the visual input. The participant wears a microphone released by TASCAM that is used to provide the audio input.

Figure 2: Greta agent, Alice.

3.2 Software

The HBAF is responsible for the smartphone authentication and sensor configuration, the data acquisition and the data access interface. Additionally, the HBAF analyses raw sensor data and extracts useful information regarding the user's behaviour. The detected behaviours are constantly pushed to the knowledge base platform through a secured REST API connection for further analysis. Finally, the provided information can be used in order to feed the dialogues among coaches. The complete operation flow of the HBAF is presented in the following picture (see Figure 3).

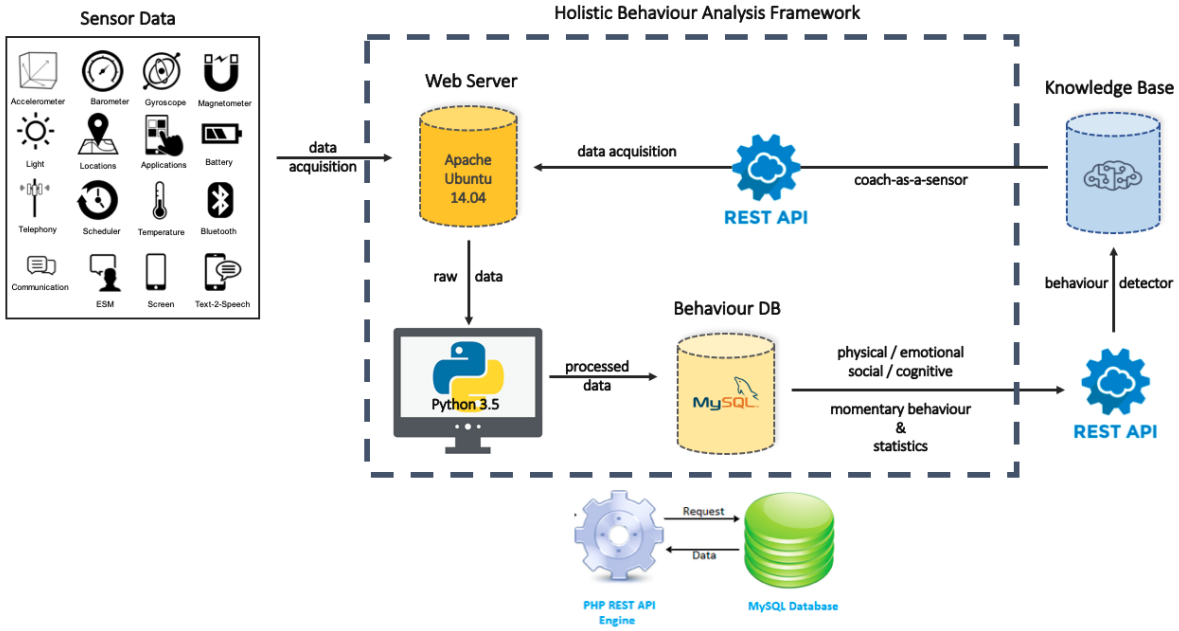

Figure 3: Operation flow of the HBAF for "on-body data" (from raw sensor data to short-term behaviours).

Specifically, smartphone raw data such as accelerometer, GPS, Bluetooth and ambient noise are recorded and stored to the Apache Web server on a Linux system (Ubuntu 14.04.5 LTS Trusty Tahr, 2018). Then,

scripts are developed in Python 3.5 (Python, 2015) in order to process the raw data and detect momentary behaviours. These models detect physical, social, emotional and cognitive behaviours, which are stored into a MySQL 5.7 database (MySQL, 2018). Additionally, the detected momentary behaviours are further analysed in order to extract relevant statistics regarding user's behaviour (e.g., the average number of steps per hour or per day, the most performed activities, etc.). The shared knowledge base gets permanent access to this database via a secured rest API connection (Slim 3.9.0 released, 2017), based on JSON web token authentication. The data are distributed through JSON format according to RFC 8259 (Datatracker, 2017) and ECMA-404 (ecma-international, 2017). It is worth mentioning that the knowledge base component can get access to the HBAF and post data extracted from the dialogs between user and coach (coach as a sensor) for further analysis.

The Figure 4 shows the overall 'off-body' system's software architecture implemented for this demonstrator.

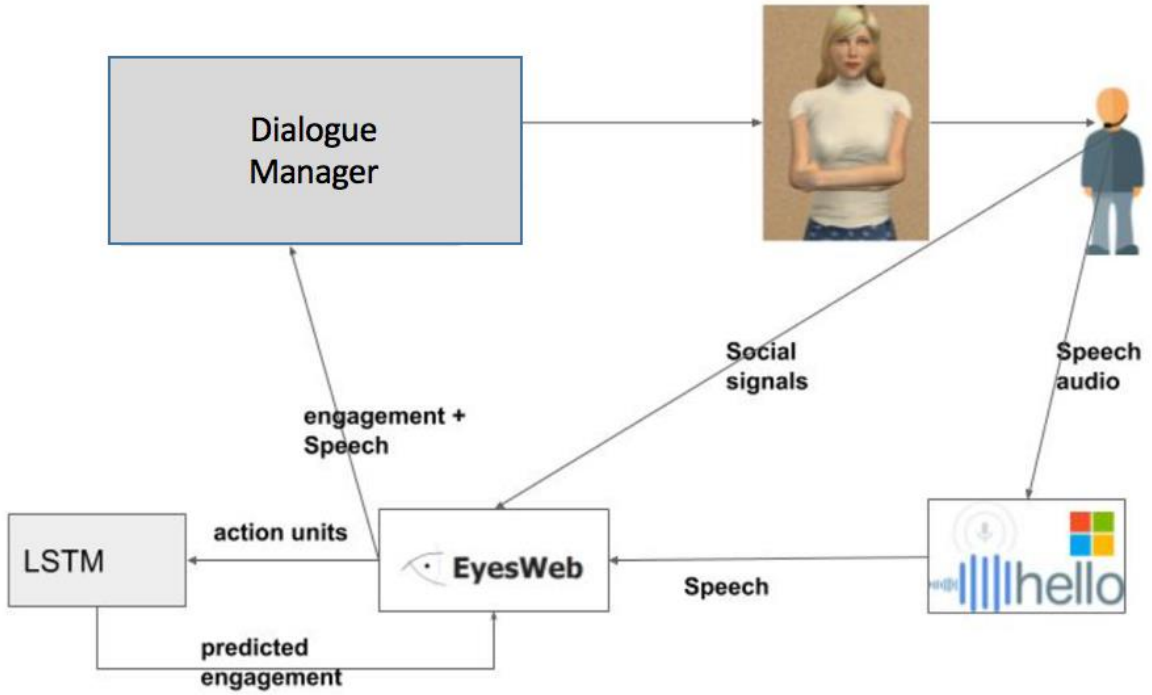

Figure 4: Operation flow of the HBAF for "off-body data" (from raw sensor data to short-term behaviours).

We make use of Microsoft speech SDK 11 platform (Microsoft, 2012) to recognise the audio data. Speech recognition requires a speech recognition engine, speech input, and a grammar. A speech recognition grammar consists of a structured list of rules that identify words or phrases that the speech recognition engine should attempt to identify in the spoken input. The system uses EyesWeb XMI 5.7 (Camurri, 2004) to analyse and record the user's non-verbal signals in real-time (e.g., torso movements, head gestures). The system manages the data coming from the sensors (i.e., microphone, Kinect v2), extracting low-level signals (i.e., audio intensity, facial Action Units - AUs -, torso and head orientation) and computing mid-level features (i.e., body and head attention over time). Internally, it utilises OpenFace 2.0 (Baltrusaitis, 2015) to extract face AUs from Kinect's RGB information. Furthermore, it transmits the data to the Automatic Speech Recognition (ASR) and the Long-Short Term Memory LSTM module (LSTM). Face engagement is extracted by the LSTM module, based on the NoXi corpus. The module uses a Recurrent Neural Network (RNN) to predict user's engagement from the face AUs. We built a set of neural networks to model the engagement level of the user in human-agent interaction using Keras v2.2.3 toolkit (Keras,

2015) with TensorFlow 1.11 (Google Brain, 2015). A basic dialogue manager helps in deciding what the agent has to say in response to an utterance made by the user. For this demonstrator, we check if there is a new speech of the user, then select the move of the agent according to the different possible moves.

4 Scripts for Short-term Behaviour Identification

The following script encapsulates the logic developed for the physical behaviour model in order to count steps and detect the user activity based on smartphone accelerometer data (please refer to deliverable D4.2). More specifically, we first estimate the magnitude value of the 3-axial accelerometer raw data to make the model orientation independent. Then, we apply a lowpass filter with a cutoff frequency equal to 3.667 Hz and we counted the peaks of the magnitude signal above the threshold 12 m/s^2, where each peak equals to one step. For detecting the other physical activities, we used the Google API for activity recognition, which is based on accelerometer and GPS data (Activity Recognition API, 2018).

```
# ================================================================================
#  Physical Behaviour Model - Counting Steps
# ================================================================================
import pandas as pd
import numpy as np
import math
from scipy.signal import butter, lfilter, argrelmax

# Change column names, parse dates and create magnitude column
df_acc['x']=df_acc["double_values_0"]
df_acc['y']=df_acc["double_values_1"]
df_acc['z']=df_acc["double_values_2"]
df_acc['magnitude'] = df_acc.apply(lambda row: math.sqrt((row.x)**2 + (row.y)**2 + (row.z)**2),axis=1)
df_acc['time'] = pd.to_datetime(df_acc['timestamp'].astype(np.int64) ,unit='ms')
df_acc = df_acc.set_index(df_acc['time'])

# Convert Timezone
df_acc.index = df_acc.index.tz_localize('UTC').tz_convert('Europe/Amsterdam')
df_acc.index= df_acc.index.tz_localize(None)

# Filter the data
order = 3
fs = 50.0  # sample rate 50Hz
cutoff = 3.667  # desired cutoff frequency of the filter, Hz
b, a = butter_lowpass(cutoff, fs, order)
r = lfilter(b, a, df_acc['magnitude'])
crossings = peak_accel_threshold(r, df_acc['time'], 12)
print("Peak Acceleration Threshold Steps:", len(crossings) / 2)

def butter_lowpass(cutoff, fs, order):  # function for lowpass filtering
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    return b, a

def peak_accel_threshold(data, timestamps, threshold):  # function for calculating peak acceleration
    last_state = 'below'
    crest_troughs = 0
    crossings = []
    for i, datum in enumerate(data):

        current_state = last_state
        if datum < threshold:
            current_state = 'below'  # below - less than threshold
        elif datum > threshold:
            current_state = 'above'  # above - above the threshold

        if current_state is not last_state:
            if current_state is 'above':
                crossing = [timestamps[i], threshold]
```

```
                crossings.append(crossing)
            else:
                crossing = [timestamps[i], threshold]
                crossings.append(crossing)

            crest_troughs += 1
        last_state = current_state

    return np.array(crossings)
```

The following script is for the social behaviour model, which is intended to detect if the user is socially isolated. For this model we use data such as Bluetooth, ambient noise, location and phone usage metrics (please refer to deliverable D4.2). To do so, we estimate the level social activeness based on the signal strength of Bluetooth visible devices (the model estimates that if a user is surrounded closely by Bluetooth devices there might be individuals to interact with), the number of performed incoming and outcoming phone calls, the number of received and sent SMS messages, the ambient noise (detects if a user has a conversation with others or if the user is in a noisy environment), and the location of the user through the Google API activity recognition (e.g., while a user is taking the bus, there might be an interaction with the bus driver or with other passengers). For any other case, the model estimates that the user is socially isolated.

```
# ================================================================================
#  Social Behaviour Model
# ================================================================================
import pandas as pd
import numpy as np
import datetime

# Bluetooth
df_bluetooth['time'] = pd.to_datetime(df_bluetooth['timestamp'].astype(np.int64) ,unit='ms')
df_bluetooth = df_bluetooth.set_index(df_bluetooth['time'])
df_bluetooth.index = df_bluetooth.index.tz_localize('UTC').tz_convert('Europe/Brussels')
df_bluetooth.index= df_bluetooth.index.tz_localize(None)
df_bluetooth = df_bluetooth.loc[df_bluetooth['device_id'] == 'f4e55e15-49ce-411c-ae41-d47a76497fa5']
df_bluetooth1 = df_bluetooth.resample('1T').mean()

df_bluetooth1["Social"] = 0
for i in range(len(df_bluetooth1)):
    if (df_bluetooth1.bt_rssi[i] >-70) and (df_bluetooth1.bt_rssi[i] <0): #condition for 0<threshold<-70
        df_bluetooth1["Social"].iloc[[i]] = 1


# Calls
df_calls['time'] = pd.to_datetime(df_calls['timestamp'].astype(np.int64) ,unit='ms')

df_calls = df_calls.set_index(df_calls['time'])
df_calls.index = df_calls.index.tz_localize('UTC').tz_convert('Europe/Brussels')
df_calls.index= df_calls.index.tz_localize(None)
df_calls = df_calls.loc[df_calls['device_id'] == 'f4e55e15-49ce-411c-ae41-d47a76497fa5']
df_calls1 = df_calls.resample('1T').sum()

df_calls1["Social"] = 0
for i in range(len(df_calls1)):
    if (df_calls1.call_duration[i] >1) and (df_calls1.call_duration[i]) <60:
        df_calls1["Social"].iloc[[i]] = 1
```

```
    if (df_calls1.call_duration[i] >=60) and (df_calls1.call_duration[i]) <120:
        df_calls1["Social"].iloc[[i,i+1]] = 1
    if (df_calls1.call_duration[i] >=120) and (df_calls1.call_duration[i]) <180:
        df_calls1["Social"].iloc[[i,i+1,i+2]] = 1
    if (df_calls1.call_duration[i] >=180) and (df_calls1.call_duration[i]) <240:
        df_calls1["Social"].iloc[[i,i+1,i+2, i+3]] = 1
    if (df_calls1.call_duration[i] >=240) and (df_calls1.call_duration[i]) <300:
        df_calls1["Social"].iloc[[i,i+1,i+2, i+3, i+4]] = 1
    if (df_calls1.call_duration[i] >=300) and (df_calls1.call_duration[i]) <360:
        df_calls1["Social"].iloc[[i,i+1,i+2, i+3, i+4, i+5]] = 1
    if (df_calls1.call_duration[i] >=360) and (df_calls1.call_duration[i]) <420:
        df_calls1["Social"].iloc[[i,i+1,i+2, i+3, i+4, i+5, i+6]] = 1
    if (df_calls1.call_duration[i] >=420) and (df_calls1.call_duration[i]) <480:
        df_calls1["Social"].iloc[[i,i+1,i+2, i+3, i+4, i+5, i+6, i+7]] = 1

# Messages
df_messages['time'] = pd.to_datetime(df_messages['timestamp'].astype(np.int64) ,unit
='ms')
df_messages = df_messages.set_index(df_messages['time'])
df_messages.index = df_messages.index.tz_localize('UTC').tz_convert('Europe/Brussels
')
df_messages.index= df_messages.index.tz_localize(None)
df_messages = df_messages.loc[df_messages['device_id'] == 'f4e55e15-49ce-411c-ae41-
d47a76497fa5']
df_messages1 = df_messages.resample('1T').sum()

df_messages1["Social"] = 0
for i in range(len(df_messages1)):
    if (df_messages1.message_type[i] >2): #condition for both received and sent mess
age
        df_messages1["Social"].iloc[[i]] = 1

#Ambient noise
df_ambient_noise['time'] = pd.to_datetime(df_ambient_noise['timestamp'].astype(np.in
t64) ,unit='ms')
df_ambient_noise = df_ambient_noise.set_index(df_ambient_noise['time'])
df_ambient_noise.index = df_ambient_noise.index.tz_localize('UTC').tz_convert('Europ
e/Brussels')
df_ambient_noise.index= df_ambient_noise.index.tz_localize(None)
df_ambient_noise = df_ambient_noise.loc[df_ambient_noise['device_id'] == 'f4e55e15-
49ce-411c-ae41-d47a76497fa5']
df_ambient_noise1 = df_ambient_noise.resample('1T').sum()

df_ambient_noise1["Social"] = 0
for i in range(len(df_ambient_noise1)):
    if (df_ambient_noise1.double_decibels[i] >30) and (df_ambient_noise1.double_rms[
i] >150): #condition for decibel threshol
        df_ambient_noise1["Social"].iloc[[i-4,i-3,i-2,i-1,i]] = 1
    if (df_ambient_noise1.double_rms[i] >200):
        df_ambient_noise1["Social"].iloc[[i-4,i-3,i-2,i-1,i]] = 1

#Google Location
df_GOOGLE['time'] = pd.to_datetime(df_GOOGLE['timestamp'].astype(np.int64) ,unit='ms
')
df_GOOGLE = df_GOOGLE.set_index(df_GOOGLE['time'])
df_GOOGLE.index = df_GOOGLE.index.tz_localize('UTC').tz_convert('Europe/Brussels')
df_GOOGLE.index= df_GOOGLE.index.tz_localize(None)
df_GOOGLE = df_GOOGLE.loc[df_GOOGLE['device_id'] == 'f4e55e15-49ce-411c-ae41-
d47a76497fa5']
df_GOOGLE1 = df_GOOGLE.copy()
df_GOOGLE1.index = df_GOOGLE.index.round("T")

df_GOOGLE1["Social"] = 0
for i in range(len(df_GOOGLE1)):
    if (df_GOOGLE1.activity_type[i] ==0): #condition for the activity taking the Bus
```

```
89.         df_GOOGLE1["Social"].iloc[[i]] = 1
90.
91.
92. Social_Activity = pd.concat([Social_Activity, df_bluetooth1["Bluetooth"]], axis=1, j
    oin_axes=[Social_Activity.index])
93. Social_Activity = pd.concat([Social_Activity, df_calls1["Calls"]], axis=1, join_axes
    =[Social_Activity.index])
94. Social_Activity = pd.concat([Social_Activity, df_messages1["SMS"]], axis=1, join_axe
    s=[Social_Activity.index])
95. Social_Activity = pd.concat([Social_Activity, df_ambient_noise1["Noise"]], axis=1, j
    oin_axes=[Social_Activity.index])
96. Social_Activity = Social_Activity.join(df_GOOGLE1["Google"])
97.
98. Social_Activity = Social_Activity.fillna(0)
99. for i in range(len(Social_Activity)):
100.           Social_Activity.Detected_Label[i] = [Social_Activity.Bluetooth[i] + Socia
    l_Activity.Calls[i] + Social_Activity.SMS[i]+
101.                                               Social_Activity.Noise[i] + Social_Ac
    tivity.Google[i]]
102.           if Social_Activity.Detected_Label[i] >0:
103.               Social_Activity.Detected_Label[i] = 1
```

The following is the algorithm for the overall engagement applied in Eyesweb. This algorithm collects the face engagement values from the LSTM model (script provided below), and computes the overall engagement.

```
at the begin of speaking turn set(attention_head = 0, attention_body = 0);
while(speaking turn is active){
       if (head_angle<angle_threshold) increase(attention_head, 1);
       if (body_angle<angle_threshold) increase(attention_body, 1);
       collect face_engagement from LSTM;
}
compute FE=mean(face_engagement)
compute HE=attention_head/speaking_turn_length;
compute BE=attention_body/speaking_turn_length;
max_HBE=max(HE,BE);
overall_engagement=αFE+βmax_HBE
```

The following is the script for the face engagement prediction where the input is the facial expression (in terms of AUs) of the participant and the output is the predicted level of engagement.

```
import numpy as np
import time
import csv
import sys
import keras
from keras.utils import to_categorical
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential
from sklearn.metrics import confusion_matrix, recall_score, precision_score, f1_score
np.random.seed(1234)
from numpy import argmax
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import KFold # import KFold
from keras.models import load_model
import matplotlib.pyplot as plt
from keras.layers.wrappers import TimeDistributed
from imblearn.under_sampling import RandomUnderSampler
from sklearn.utils import class_weight


def dataAuLoad(filename,  indice_start,  indice_end):

    # reading csv file
    with open( filename,'r') as csvfile:
        # creating a csv reader object
```

```
        csvreader = csv.reader(csvfile, quoting=csv.QUOTE_NONNUMERIC)
        # extracting each data row one by one
        rows = []
        cpt=0
        # extracting each data row one by one
        for row in csvreader:
            rows.append(row)

        features=[]
        for row in rows:
            featureAu = row[indice_start:indice_end]
            #featureAu = [float(i) for i in featureAu]
            features.append(featureAu)
            cpt=cpt+1

            #print (featureAu, 'featureAu', type(featureAu[0]), ' cpt ',cpt)
        features = np.array(features)
        print ('End loading file. features shape = ', features.shape)
        return features

def dataAuLoadsave(filename,  indice_start,  indice_end, feature_Size):

    # reading csv file
    with open( filename,'r') as csvfile:
        # creating a csv reader object
        csvreader = csv.reader(csvfile, delimiter=',', quoting=csv.QUOTE_NONNUMERIC)
        # extracting each data row one by one
        cpt=1
        rows = []
        # extracting each data row one by one
        for row in csvreader:
            rows.append(row)

        features=[]
        for row in rows:
            featureAu = row[indice_start:indice_end]
            featureAu = [float(i) for i in featureAu]
            cpt=cpt+1
            features.append(featureAu)
        features = [y for x in features for y in x] # aplatir la liste de listes pour pouoir la reshapper
        features = np.array(features)
        features = features.reshape( round(len(features)/feature_Size), feature_Size)
        print ('End loading file. features shape = ', features.shape)
        return features
def data_load(filename):

    # reading csv file
    with open( filename,'r') as csvfile:
        # creating a csv reader object
        csvreader = csv.reader(csvfile)

        rows = []
        # extracting each data row one by one
        for row in csvreader:
            rows.append(row[0])
        result = np.array(rows)
        return result


def data_load_smote(sequence_length=120, smoting=True, filenameAU="",filenameEngagement="",
filenameConState=""):
    AuActivations=dataAuLoad(filenameAU,36,54)
    print ("AuActivations  : ", AuActivations.shape)
    print ("AuActivations  : ", AuActivations)

    AuIntensites=dataAuLoad(filenameAU,19,36)
    print ("AuIntensites  : ", AuIntensites.shape)
    print (AuIntensites)
    sys.stdout.flush()
    mean = AuIntensites.mean(axis=0)
    std = AuIntensites.std(axis=0)
    AuIntensites -= mean
    AuIntensites /= std
    print ("mean  : ", mean," std ", std)
    print ("AuIntensites  : ", AuIntensites.shape)
    print (AuIntensites)
    HeadRotation=dataAuLoad(filenameAU,16,19)
    print ("HeadRotation  : ", HeadRotation.shape)
    mean = HeadRotation.mean(axis=0)
```

```
    std = HeadRotation.std(axis=0)
    HeadRotation -= mean
    HeadRotation /= std
    print ("mean  : ", mean," std ", std)

    resultEng=data_load(filenameEngagement)
    print ("resultEng  : ", resultEng.shape)
    print (resultEng)
    class_weights = class_weight.compute_class_weight('balanced',  np.unique(resultEng),  resultEng)
    print ('%%%%%%%%%%%%%', class_weights)
    TurnTaking=data_load(filenameConState)
    TurnTaking=to_categorical(TurnTaking);
    print ("TurnTaking  : ", TurnTaking.shape)
    print (TurnTaking)
    InputAu=np.c_[AuIntensites,AuActivations]
    #InputAu=AuIntensites
    InputFace=np.c_[InputAu,HeadRotation]
    #InputFace=InputAu
    Input=np.c_[InputFace,TurnTaking]
    result=[]
    if smoting:
        '''smote = SMOTE(ratio='auto')
        X_resampled, y_resampled = smote.fit_sample(Input,resultEng)'''
        rus = RandomUnderSampler(return_indices=True)
        X_resampled, y_resampled, idx_resampled = rus.fit_sample(Input, resultEng)
        #X_resampled, y_resampled= resultEng, resultAU
        Engagment=to_categorical(y_resampled)
        print ("Engagment  : ", Engagment.shape)
        print (Engagment)
        result=np.c_[X_resampled,Engagment]
    else:
        Engagment=to_categorical(resultEng)
        print ("Engagment  : ", Engagment.shape)
        print (Engagment)
        result=np.c_[Input,Engagment]


    ResX=[]
    for index in range(len(result) - sequence_length):
        ResX.append(result[index: index + sequence_length])
    ResX = np.array(ResX)
    result=ResX

    featureXnumber=43;
    featureYindice=48;
    train = result
    print ("train  : ", train.shape)
    print (train)
    np.random.shuffle(train)
    X_train1 = train[:, :-1]
    X_train = X_train1[:, :,0:featureXnumber]
    print ("X_train  : ", X_train.shape)
    print (X_train1)
    y_train = train[:, -1]
    y_train1= y_train[:,featureXnumber:featureYindice]
    print ("y_train1  : ", y_train1.shape)
    print (y_train1)

#
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], featureXnumber))

#
    return [X_train, y_train1]

def build_model(X_train,y_train):
    model = Sequential()
    hidden_nodes=20
    model = Sequential()
    model.add(LSTM(hidden_nodes, input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Dropout(0.2))
    print ('Dense .........;... ', y_train.shape[1] )
    model.add(Dense(10,activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(y_train.shape[1]))
    model.add(Activation("softmax"))

    start = time.time()
    model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
    print ("Compilation Time : ", time.time() - start)
```

```
    return model

def run_network(model=None, data=None):
    global_start_time = time.time()
    epochs = 5
    sequence_length = 10

    print ('Loading train data... ')
    X_train, y_train =
data_load_smote(sequence_length,False,'data/train/AuAllExpert.csv','data/train/EngagementAllExpert.csv','data/
train/All_conversation_states.csv')

    print ('\nData Loaded. Compiling...\n')
    if model is None:
        model = build_model(X_train,y_train)

    try:
        '''kf = KFold(n_splits=3)

        for train_index, test_index in kf.split(X_train):
             X_t, X_test = X_train[train_index], X_train[test_index]
             y_t, y_test = y_train[train_index],y_train[test_index]
             history=model.fit(X_t, y_t, batch_size=512, nb_epoch=epochs, validation_split=0.05)
             # save model to single file
             model.save('my_model.h5')'''


             # load model from single file
             #model1 = load_model('lstm_model.h5')
        #class_weights={0: 20, 1: 10, 2: 5, 3: 1, 4: 10} class_weight=class_weights
        #class_weights ={0:14.2422043,   1:2.40603996,  2:1.00247871,  3:0.31640837,  4:2.80768415}
        class_weights ={0:16.00501355,  1:3.24319055, 2:1.44689165,  3:0.27664976,  4:3.09247284}
        callbacks_list =
[keras.callbacks.EarlyStopping(monitor='acc',patience=1,),keras.callbacks.ModelCheckpoint(filepath='my_model.h
5',monitor='val_loss',save_best_only=True,)]
        history=model.fit(X_train, y_train, batch_size=100, nb_epoch=epochs, validation_split=0.1)

        # save model to single file
        #model.save('lstm_model.h5')
        # load model from single file
        model1 = load_model('my_model.h5')
        #Metrics
        print ('Loading test data... ')
        #class_weights = {0:1.76, 1:0.49, 2: 2.48}
        #lass_weights = class_weight.compute_class_weight('balanced',  np.unique(y_train),  y_train)
#class_weight=class_weights
        X_test, y_test =
data_load_smote(sequence_length,False,'data/test/AUExpert1.csv','data/test/EngagementExpert1.csv','data/test/1
_conversation_states.csv')
        results = model1.evaluate(X_test, y_test)
        print(results)
        y_test_Cat=[]
        for row in y_test:
            y_test_Cat.append(argmax(row))
        #print ('y_test_Cat : ', y_test_Cat)
        test_labels_copy = copy.copy(y_test_Cat)
        np.random.shuffle(test_labels_copy)
        hits_array = np.array(y_test_Cat) == np.array(test_labels_copy)
        #print ('hits_array ', len(hits_array),'np.sum(hits_array)) ', np.sum(hits_array),' len(y_test)
',len(y_test), ' test_labels_copy ', len(test_labels_copy))
        print ('res' ,float(np.sum(hits_array) / len(y_test_Cat)))

        loss = history.history['loss']
        val_loss = history.history['val_loss']
        epochs = range(1, len(loss) + 1)
        plt.plot(epochs, loss, 'bo', label='Training loss')
        plt.plot(epochs, val_loss, 'b', label='Validation loss')
        plt.title('Training and validation loss')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.legend()
        plt.show()

        acc = history.history['acc']
        val_acc = history.history['val_acc']
        epochs = range(1, len(acc) + 1)
        plt.plot(epochs, acc, 'bo', label='Training acc')
        plt.plot(epochs, val_acc, 'b', label='Validation acc')
        plt.title('Training and validation accuracy')
```

```
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.legend()
        plt.show()
        predicted = model1.predict(X_test)
        print ('predicted : ', predicted.shape)
        print (predicted)
        predictedCat=[]
        for row in predicted:
            predictedCat.append(argmax(row))
        print ('predictedCat : ', predictedCat)
        print ('Recall : ', recall_score(y_test_Cat,predictedCat,average='weighted'))
        print ('Recall : ', recall_score(y_test_Cat,predictedCat,average=None))
        print ('Precision : ',  precision_score(y_test_Cat,predictedCat,average='weighted'))
        print ('Precision : ',  precision_score(y_test_Cat,predictedCat,average=None))
        print ('f1_score : ',  f1_score(y_test_Cat,predictedCat,average='weighted'))
         print ('f1_score : ',  f1_score(y_test_Cat,predictedCat,average=None))
        print ('confusion_matrix : ')
        print (confusion_matrix(y_test_Cat,predictedCat))

    except KeyboardInterrupt:
        print ('Training duration (s) : ', time.time() - global_start_time)
        return model, y_test, 0
print ('Training duration (s) : ', time.time() - global_start_time)
    return model


def Newprediction(model=None,X_test=None):
    return model.predict(X_test)
model=run_network(None, None)
```

5 Demonstrator

In the first demonstrator (see Table 1 for link) we show the monitoring of momentary physical and social behaviours using smartphone data. For the physical behaviour, we record accelerometer data in order to detect the walking activity and count the number of steps based on the user´s smartphone which is placed in the trouser pocket. In Figure 5: Demonstrator for the physical behaviour model., we demonstrate the scenario of a user walking and performing 5 steps. For the social behaviour, we record ambient noise, Bluetooth and phone usage metrics (number of incoming/outcoming calls and number of received/sent text messages) in order to detect the level of user's interaction. In addition to the aforementioned smartphone data for the social behaviour model, we use the Google API for activity recognition, assuming that the user is interacting with other people inside a vehicle (such as taking the bus). In Figure 6, we show a screenshot of the aforementioned scenario where a user walks alone, and we demonstrate the case where the subject is socially isolated.

Table 1: Link to the "Holistic Behaviour Analysis Framework" Demonstrator on YouTube.

https://www.youtube.com/watch?v=Mr0afhSKrVk

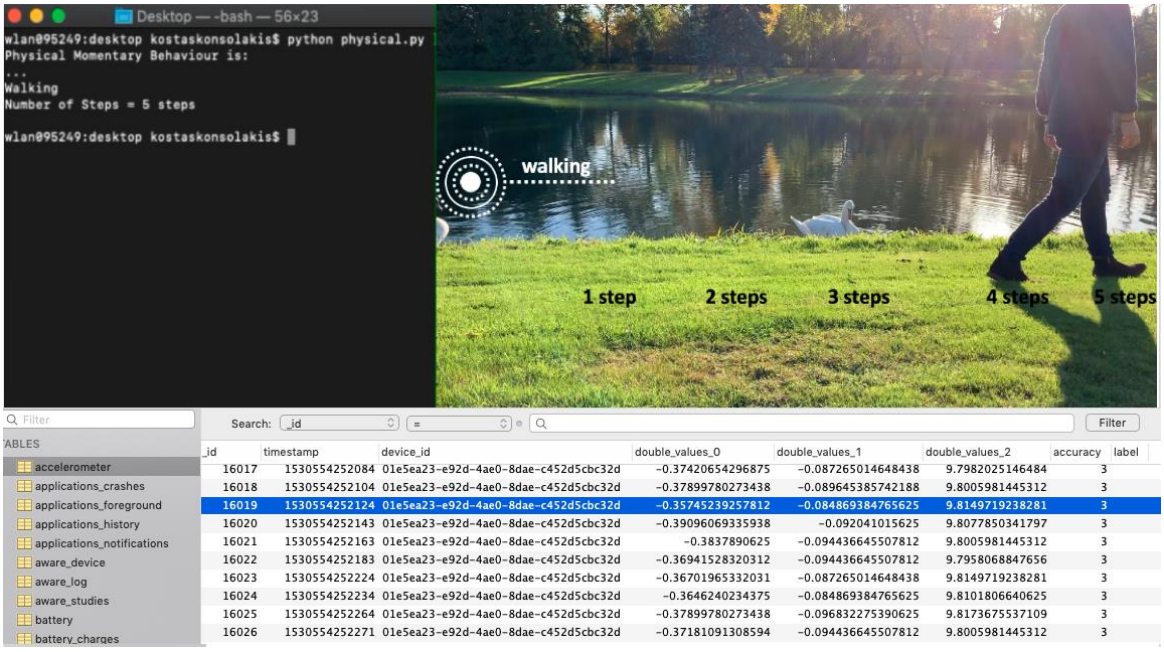

Figure 5: Demonstrator for the physical behaviour model.

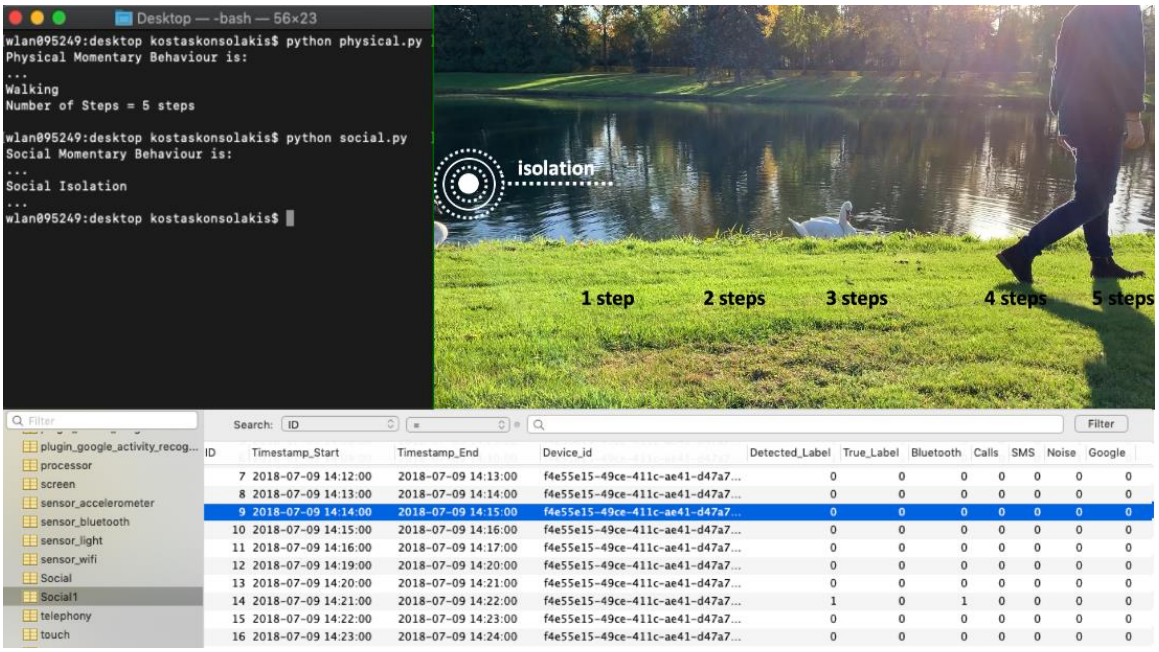

Figure 6: Demonstrator for the social behaviour model.

In the second demonstrator (published on YouTube, see Table 2), we show the participants interacting with a Greta agent in real-time in a laboratory setting. Such scenario aims at picturing the situation where a user would interact with the council of coaches at home. We have designed a simple interaction scenario to showcase the capabilities of our model to detect the user's engagement level (short-term emotional and cognitive behaviours) and adapt it accordingly. The engagement levels of the participant will be detected through visual cues, i.e. head along with facial expressions. The participants wear a close-to-mouth microphone and the body movements are detected along with facial expressions through a Kinect sensor mounted on the screen. The demonstrator scenario begins with the Greta agent greeting the participant. Further the agent asks some questions to the participant to engage in a short conversation and says goodbye. Since the demonstrator was recorded in Paris, France where most of the participants are native French speakers, and the dialogues are recorded in French.

Table 2: Link to the "Engagement Detection" Demonstrator on YouTube.

https://www.youtube.com/watch?v=wEVsWIGrlFg

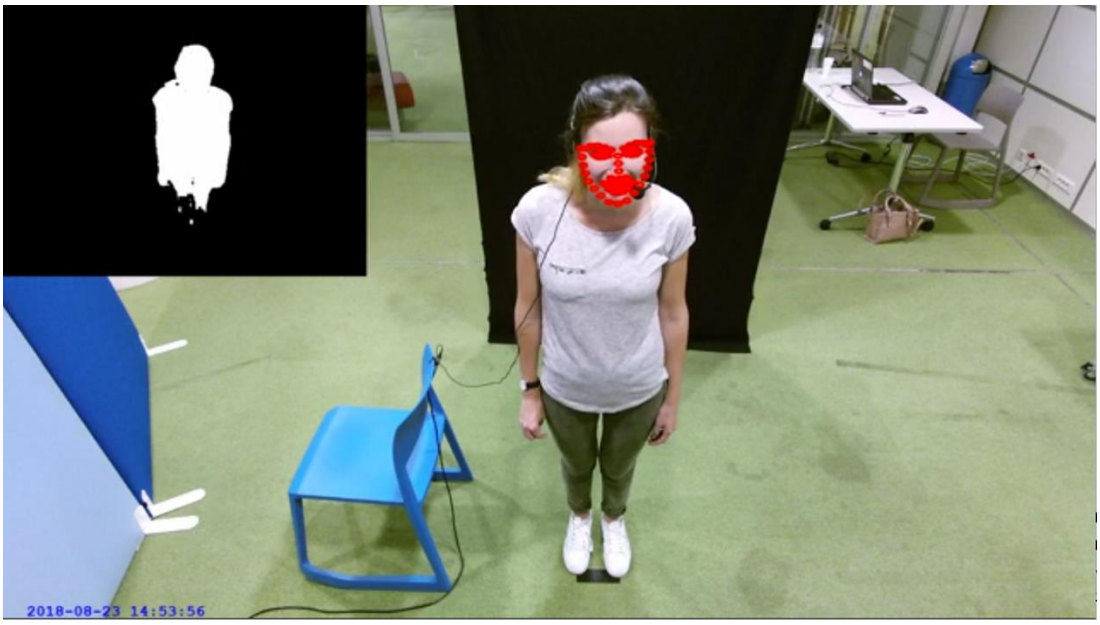

Figure 7: Snapshot of participant 1 interacting with the Greta agent.

The engagement level of participant is measured from a scale of 1 to 5 (1 = least engaged to 5= most engaged) based on facial expressions and head movement (ref. D4.2 for details), and following is the plot of a participant's engagement level over a given period of time. At certain time intervals, the detected engagement level is zero due to occlusions from hand or when participants turned back.

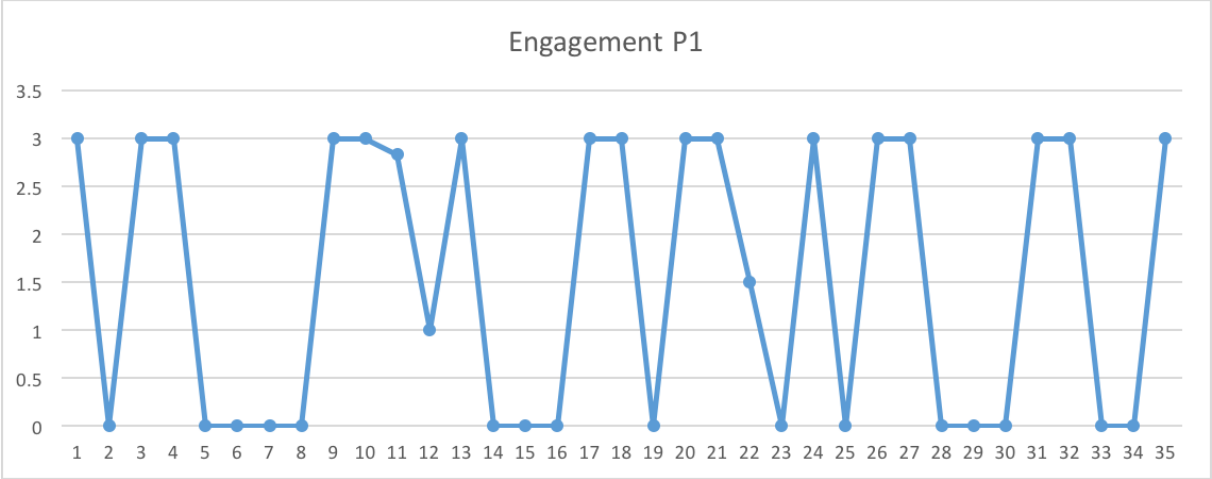

Figure 8: Detected engagement level of participant 1.

6 Bibliography

(2015). Retrieved from Python: https://www.python.org/downloads/release/python-350/

(2017). Retrieved from Datatracker: https://datatracker.ietf.org/doc/rfc8259/?include_text=1

(2017). Retrieved from ecma-international: https://www.ecma-international.org/publications/standards/Ecma-404.htm

(2018). Retrieved from MySQL: https://dev.mysql.com/doc/relnotes/mysql/5.7/en/

Activity Recognition API. (2018). Retrieved from Google: https://developers.google.com/location-context/activity-recognition/

Baltrusaitis, T. (2015). Retrieved from https://github.com/TadasBaltrusaitis/OpenFace

Camurri. (2004). Retrieved from http://www.infomus.org/eyesweb_eng.php

Google Brain. (2015). Retrieved from https://github.com/tensorflow

Keras. (2015). Retrieved from https://keras.io/

Microsoft. (2012). Retrieved from https://docs.microsoft.com/en-us/previous-versions/office/developer/speech-technologies/dd266409(v%3doffice.14)

Slim 3.9.0 released. (2017). Retrieved from Slim Framework: https://www.slimframework.com/2017/11/04/slim-3.9.0.html

Ubuntu 14.04.5 LTS Trusty Tahr. (2018). Retrieved from Ubuntu: http://releases.ubuntu.com/trusty/

YouTube - Demonstrator for 'On-Body' Sensors. (2018). Retrieved from YouTube: https://youtu.be/Mr0afhSKrVk