get_measures

May 7, 2023

[ ]:

```
import torch as tc
import pandas as pd
import numpy as np
import open3d as otd
from tqdm import tqdm
import matplotlib.pyplot as plt
otd_vector3d = otd.utility.Vector3dVector
from src.star.star import STAR
from src.curve_utils import CurveUtils
from src.curve_generator import CurveGenerator
from src.mesh_manipulation import save_obj
device = tc.device("cuda" if tc.cuda.is_available() else "cpu")
genders = ['female', 'male']
male = ['male']
female = ['female']
```

[ ]:

```
their_semantics = [
    'Bust girth',
    'Waist girth',
    'Hip girth',
    'Thigh girth R',
    'Upper arm girth R',
    'Neck girth',
    'Height (m)',
]
our_semantic = [
    'bust_chest_girth', # 5.3.4
    'waist_girth', # 5.3.10
    'hip_girth', # 5.3.13
    'thigh_girth', # 5.3.20
    'upper_arm_girth', # 5.3.16
    'neck_girth', # 5.3.2
    'stature', # 5.1.1
]

curve_index = {
    'neck_girth':4, # 5.3.2
    'bust_chest_girth': 0, # 5.3.4
    'waist_girth': 1, # 5.3.10
    'hip_girth': 1, # 5.3.13
    'upper_arm_girth': 3, # 5.3.16
    'thigh_girth': 2, # 5.3.20
}
```

[ ]:

```
subdivided_bodies = tc.load('data/subdivided_bodies.pt')
measures = pd.read_pickle(f'data/cleaned_measures.zip')
measures.index = measures['Subject']
measures = measures[measures['Measuring station'] == "MOVE4D"]

mfd_gender_measures = dict()
for gender in genders:
    mfd_gender_measures[gender] = measures[measures['Sex'] == gender]
    mfd_gender_measures[gender] = mfd_gender_measures[gender][their_semantics]
    mfd_gender_measures[gender].columns = our_semantic
    mfd_gender_measures[gender]['stature'] *= 1000
```

[ ]:

```
selected_subjects = dict()
selected_measures = dict()

for gender in genders:
    gender_measures = measures[measures['Sex'] == gender]
    temp_measures = gender_measures[their_semantics].iloc[::2]
    selected_subjects[gender] = 'IEEEP2_07' if gender == 'female' else␣
  ↪'IEEEP2_04'
    selected_measures[gender] = temp_measures.loc[selected_subjects[gender]]
    selected_subjects[gender] = temp_measures.index.
  ↪get_loc(selected_subjects[gender])*2
    selected_measures[gender].index = our_semantic
    selected_measures[gender]['stature'] *= 1000
```

[ ]:

```
gender_curves = dict()
for gender in genders:
    print(f'SEGMENTING {gender.upper()} BODIES', end='')
    faces = subdivided_bodies['faces'][gender].to(device)
    bodies = subdivided_bodies['vertices'][gender]
    body = bodies[selected_subjects[gender]].to(device)
    measures = selected_measures[gender]
    result = CurveGenerator.get_curves(body, measures, faces, device, gender)
    gender_curves[gender] = result[0]
tc.save(gender_curves, "data/gender_curves.zip")
```

```
SEGMENTING FEMALE BODIES

processing body: 100%|          | 44/44 [06:54<00:00,  9.41s/it]
```

```
SEGMENTING MALE BODIES

processing body: 100%|          | 44/44 [07:22<00:00, 10.05s/it]
```

[ ]:

```
gender_measures = dict()
for gender in genders:
    print(f'MEASURING {gender.upper()} BODIES', end='')
    gender_measures[gender] = [[]]*5
    for segments_index, curves_segments in enumerate(gender_curves[gender]):
        gender_measures[gender][segments_index] = []
        for curves_index, curves in enumerate(tqdm(curves_segments)): # bust,␣
↪torso, leg, arm, neck

            gender_measures[gender][segments_index].append([])
            for body in subdivided_bodies['vertices'][gender]:
                position = CurveUtils.generate_positions(curves, body.
↪to(device))
                calculated_measures = CurveUtils.calculate_distances(position)
                gender_measures[gender][segments_index][curves_index].
↪append(calculated_measures)

            calculated_measures =␣
↪gender_measures[gender][segments_index][curves_index]
            gender_measures[gender][segments_index][curves_index] = tc.
↪FloatTensor(calculated_measures)
        gender_measures[gender][segments_index] = tc.
↪row_stack(gender_measures[gender][segments_index])

tc.save(gender_measures, "data/calculated_measures.zip")
```

```
MEASURING FEMALE BODIES

100%|          | 302/302 [00:27<00:00, 10.97it/s]
100%|          | 353/353 [00:22<00:00, 15.72it/s]
100%|          | 604/604 [00:38<00:00, 15.72it/s]
100%|          | 402/402 [00:25<00:00, 15.72it/s]
100%|          | 6040/6040 [06:30<00:00, 15.47it/s]

MEASURING MALE BODIES

100%|          | 338/338 [00:30<00:00, 11.14it/s]
100%|          | 395/395 [00:24<00:00, 15.81it/s]
100%|          | 676/676 [00:45<00:00, 14.96it/s]
100%|          | 450/450 [00:31<00:00, 14.22it/s]
100%|          | 6760/6760 [08:04<00:00, 13.95it/s]
```

[ ]:

```
gender_calculated_measures = tc.load("data/calculated_measures.zip")
```

```
[ ]: gender_measures_wc = dict()
     for gender in genders:
         gender_measures_wc[gender] = []
         for index in range(0,5):
             gender_measures_wc[gender].append(gender_measures[gender][index])
             if index == 1:
                 gender_measures_wc[gender].append(gender_measures[gender][index])
     gender_measures = gender_measures_wc
```

```
[ ]: best_gender_measures = dict() ## caso minimo
     +
     for idx, gender in enumerate(genders):
         best_gender_measures[gender] = []
         for index, curve in enumerate(our_semantic[:-1]):
             measured = gender_measures[gender][index].T[::2].T
             ground_truth = mfd_gender_measures[gender][curve][::2]
             result = (measured - tc.FloatTensor(ground_truth/10).unsqueeze(0)).abs()
             min_rows_values, min_rows_indices = result.min(0)
             min_columns_values, min_columns_indices = min_rows_values.min(0)
             best = min_rows_indices[min_columns_indices]
             best_gender_measures[gender].append((
                 best.numpy(),
                 result[best].min().numpy(),
                 result[best].max().numpy(),
                 result[best].mean().numpy(),
                 result[best].std().numpy()
             ))
     gender_results = {
         'male': pd.DataFrame(best_gender_measures['male'], columns=['best', 'min',␣
      ↪'max', 'mean', 'std']),
         'female': pd.DataFrame(best_gender_measures['female'], columns=['best',␣
      ↪'min', 'max', 'mean', 'std'])
     }
     print("male errors:")
     print(gender_results['male'])
     print("\nfemale errors:")
     print(gender_results['female'])
```

```
male errors:
   best             min        max        mean         std
0   185    0.0058059692  4.8767014   1.2775197   1.1429859
1   143    0.0001449585  22.587074   10.640367    6.196411
2   152    0.0005264282  2.7030334    1.201998  0.65851164
3   544    0.0002822876  2.0477638  0.68975365  0.51716274
4   384   0.00093078613  1.1062737  0.48048836  0.32412186
5  1090   7.6293945e-06  4.3489075   1.2007937   1.0351614
```

```
female errors:
   best             min        max        mean        std
0   156  0.00089263916   5.222603   2.2750773    1.49956
1   273    0.003112793  7.7436066   3.7575183  2.2311535
2   113  0.00047302246  4.9770966   1.3233047  1.1549083
3   473  0.00037765503  3.8846436  0.84255683  0.8231378
4   352  0.00018692017  2.0258427   0.6941188  0.4883672
5  3641  3.8146973e-05  1.7938309  0.63073695 0.49782026
```

[ ]:

```
best_gender_measures = dict() ## caso médio
for idx, gender in enumerate(genders):
    best_gender_measures[gender] = []
    for index, curve in enumerate(our_semantic[:-1]):
        measured = gender_measures[gender][index].T[::2].T
        ground_truth = mfd_gender_measures[gender][curve][::2]
        result = (measured - tc.FloatTensor(ground_truth/10).unsqueeze(0)).abs()
        min_rows_values = result.mean(1)
        min_columns_values, min_columns_indices = min_rows_values.min(0)
        best = min_columns_indices
        best_gender_measures[gender].append((
            result[best].min().numpy(),
            result[best].max().numpy(),
            result[best].mean().numpy(),
            result[best].std().numpy()
        ))
gender_results = {
    'male': pd.DataFrame(best_gender_measures['male'], columns=['min', 'max',␣
 ↪'mean', 'std'], index=our_semantic[:-1]),
    'female': pd.DataFrame(best_gender_measures['female'], columns=['min',␣
 ↪'max', 'mean', 'std'], index=our_semantic[:-1])
}
print("male errors:")
print(gender_results['male'])
print("\nfemale errors:")
print(gender_results['female'])
```

```
male errors:
                          min         max        mean         std
bust_chest_girth  0.050933838   4.6888275   1.1817387   1.0458738
waist_girth       0.048301697   1.6495056   0.5694864   0.4000256
hip_girth         0.007156372   1.2815323   0.4521351   0.3367785
thigh_girth       0.037849426   1.9917145  0.68962806   0.5091672
upper_arm_girth  0.0028152466  0.90377045    0.312722  0.20489208
neck_girth         0.01726532   2.3889008   0.6928124   0.5737223

female errors:
                          min        max        mean         std
bust_chest_girth  0.020835876   4.166733   1.7483453   1.2649621
```

```
waist_girth       0.0072021484  3.5882034   1.0795076    0.954025
hip_girth          0.065208435  4.9380493   1.1920289   1.0881943
thigh_girth         0.05659485   3.587326  0.82730645   0.7422466
upper_arm_girth    0.014976501  1.3496666  0.54173285  0.35838273
neck_girth        0.0028457642  1.2863541  0.49442312  0.34019846
```

[ ]:
```
(gender_results['male']*10).astype(float).round(decimals=2)
```

[ ]:
```
                   min    max   mean    std
bust_chest_girth  0.51  46.89  11.82  10.46
waist_girth       0.48  16.50   5.69   4.00
hip_girth         0.07  12.82   4.52   3.37
thigh_girth       0.38  19.92   6.90   5.09
upper_arm_girth   0.03   9.04   3.13   2.05
neck_girth        0.17  23.89   6.93   5.74
```

[ ]:
```
(gender_results['female']*10).astype(float).round(decimals=2)
```

[ ]:
```
                   min    max   mean    std
bust_chest_girth  0.21  41.67  17.48  12.65
waist_girth       0.07  35.88  10.80   9.54
hip_girth         0.65  49.38  11.92  10.88
thigh_girth       0.57  35.87   8.27   7.42
upper_arm_girth   0.15  13.50   5.42   3.58
neck_girth        0.03  12.86   4.94   3.40
```

[ ]:
```
best_gender_curves = dict()
for gender in genders:
    all_positions = []
    best_gender_curves[gender] = []
    for index, curve in enumerate(our_semantic[:-1]):
        best = gender_results[gender].loc[index]['best']
        coordinates = gender_curves[gender][curve_index[curve]][best]
        best_gender_curves[gender].append(coordinates)
        faces = subdivided_bodies['faces'][gender].to(device)
        bodies = subdivided_bodies['vertices'][gender]
        body = bodies[selected_subjects[gender]].to(device)
        position = CurveUtils.generate_positions(coordinates, body.to(device))
        all_positions.append(position)
    save_obj(f'output/{gender}_points.obj', tc.row_stack(all_positions))
tc.save(best_gender_curves, 'data/selected_gender_curves.zip')
```

[ ]:
```
best_gender_measures = dict()
for idx, gender in enumerate(genders):
    best_gender_measures[gender] = []
    for index, curve in enumerate(our_semantic[:-1]):
        best = gender_results[gender].loc[index]['best']
        measured = gender_measures[gender][index][best].numpy()
        best_gender_measures[gender].append(measured*10)
    best_gender_measures[gender].append(tc.arange(2).repeat(36).numpy()+1)
    best_gender_measures[gender].append([gender]*72)
    best_gender_measures[gender].append(["our"]*72)
    best_gender_measures[gender].append(mfd_gender_measures[gender].index)
```

[ ]:

```
aditional_semantic = ['repetition', "gender", "measures_station", 'subject']
our_measures = pd.concat([
    pd.DataFrame(best_gender_measures['female'], index=our_semantic[:
 ↪-1]+aditional_semantic).T,
    pd.DataFrame(best_gender_measures['male'], index=our_semantic[:
 ↪-1]+aditional_semantic).T
])
our_measures.to_pickle("data/our_measures.zip")
```

[ ]:

```

```