

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

➞ Mounted at /content/drive

```
1 !ls /content/drive/MyDrive/action/data/raw
```

➞ 0.pkl 11.pkl 13.pkl 15.pkl 17.pkl 1.pkl 3.pkl 5.pkl 7.pkl 9.pkl
10.pkl 12.pkl 14.pkl 16.pkl 18.pkl 2.pkl 4.pkl 6.pkl 8.pkl action_

```
1 !pip install filterpy
```

➞ Collecting filterpy

Downloading filterpy-1.4.5.zip (177 kB)

178.0/178.0 kB 4.5 MB/s eta 0

Preparing metadata (setup.py) ... done

Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-pack

Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-pack

Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist

Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.1

Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/di

Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.

Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.

Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10

Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/d

Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.1

Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/pytho

Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-p

Building wheels for collected packages: filterpy

Building wheel for filterpy (setup.py) ... done

Created wheel for filterpy: filename=filterpy-1.4.5-py3-none-any.whl size

Stored in directory: /root/.cache/pip/wheels/0f/0c/ea/218f266af4ad6268975

Successfully built filterpy

Installing collected packages: filterpy

Successfully installed filterpy-1.4.5

```
1 !pip install transforms3d
```

➞ Collecting transforms3d

Downloading transforms3d-0.4.2-py3-none-any.whl.metadata (2.8 kB)

Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.10/dis

Downloading transforms3d-0.4.2-py3-none-any.whl (1.4 MB)

1.4/1.4 MB 13.8 MB/s eta 0:00:0

Installing collected packages: transforms3d

Successfully installed transforms3d-0.4.2

```
1 !pip install torch-geometric # install the missing package
```


```
⇒ Collecting torch-geometric
  Downloading torch_geometric-2.6.1-py3-none-any.whl.metadata (63 kB)
    63.1/63.1 kB 1.8 MB/s eta 0:0
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: psutil>=5.8.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/py
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.1
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python
Requirement already satisfied: yarl<2.0,>=1.12.0 in /usr/local/lib/python3.
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/py
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/p
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/di
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3
Requirement already satisfied: typing-extensions>=4.1.0 in /usr/local/lib/p
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.1
Downloading torch_geometric-2.6.1-py3-none-any.whl (1.1 MB)
    1.1/1.1 MB 19.2 MB/s eta 0:00:0
Installing collected packages: torch-geometric
Successfully installed torch-geometric-2.6.1
```

```
1 import sys
2 sys.path.append('/content/drive/MyDrive/action')
```

```

1 import os
2 import pickle
3
4 # Step 3: Specify the path to your folder in Google Drive
5 folder_path = '/content/drive/MyDrive/action/data/raw' # Replace with your a
6
7 # Step 4: List files in the folder
8 extracted_files = os.listdir(folder_path)
9 print("Files in the folder:", extracted_files)
10
11 # Step 5: Load each `.pkl` file
12 pkl_files = [f for f in os.listdir(folder_path) if f.endswith('.pkl')]
13 data = []
14
15 for pkl_file in pkl_files:
16     with open(os.path.join(folder_path, pkl_file), 'rb') as file:
17         data.append(pickle.load(file))
18
19 # Step 6: Check if data is loaded
20 print(f"Loaded {len(data)} .pkl files.")
21

```

 Files in the folder: ['7.pkl', '3.pkl', 'id.json', '10.pkl', '15.pkl', '13.
Loaded 19 .pkl files.

1. original code without clustering

```

1 import os
2 import torch
3 import numpy as np
4 import pickle
5 import logging
6 import random
7 from tqdm import tqdm
8 from torch_geometric.data import Dataset
9
10 class MMRKeypointData(Dataset):
11     raw_data_path = '/content/drive/MyDrive/action/data/raw'
12     processed_data = '/content/drive/MyDrive/action/data/processed/mmr_kp/ds'
13     max_points = 22
14     seed = 42
15     partitions = (0.8, 0.1, 0.1)
16     stacks = None
17     zero_padding = 'per_data_point'
18     zero_padding_styles = ['per_data_point', 'per_stack', 'data_point', 'stack']
19     num_keypoints = 9
20     forced_rewrite = False
21
22     def _parse_config(self, c):
23         c = {k: v for k, v in c.items() if v is not None}
24         self.seed = c.get('seed', self.seed)
25         self.processed_data = c.get('processed_data', self.processed_data)
26         self.max_points = c.get('max_points', self.max_points)
27         self.partitions = (
28             c.get('train_split', self.partitions[0]),
29             c.get('val_split', self.partitions[1]),
30             c.get('test_split', self.partitions[2]))
31         self.stacks = c.get('stacks', self.stacks)
32         self.zero_padding = c.get('zero_padding', self.zero_padding)
33         self.num_keypoints = c.get('num_keypoints', self.num_keypoints)
34         if self.zero_padding not in self.zero_padding_styles:
35             raise ValueError(
36                 f'Zero padding style {self.zero_padding} not supported.')
37         self.forced_rewrite = c.get('forced_rewrite', self.forced_rewrite)
38
39
40     def __init__(
41         self, root, partition,
42         transform=None, pre_transform=None, pre_filter=None,
43         mmr_dataset_config = None):
44         super(MMRKeypointData, self).__init__(
45             root, transform, pre_transform, pre_filter)
46         self._parse_config(mmr_dataset_config)
47         # check if processed_data exists
48         if (not os.path.isfile(self.processed_data)) or self.forced_rewrite:
49             self.data, _ = self._process()

```

```

11         os.makedirs(os.path.dirname(self.processed_data), exist_ok=True)
12         with open(self.processed_data, 'wb') as f:
13             pickle.dump(self.data, f)
14     else:
15         with open(self.processed_data, 'rb') as f:
16             self.data = pickle.load(f)
17     total_samples = len(self.data['train']) + len(self.data['val']) + len
18     self.data = self.data[partition]
19     self.num_samples = len(self.data)
20     self.target_dtype = torch.float
21     self.info = {
22         'num_samples': self.num_samples,
23         'num_keypoints': self.num_keypoints,
24         'num_classes': None,
25         'max_points': self.max_points,
26         'stacks': self.stacks,
27         'partition': partition,
28     }
29     logging.info(
30         f'Loaded {partition} data with {self.num_samples} samples, '
31         f' where the total number of samples is {total_samples}')
32
33     def len(self):
34         return self.num_samples
35
36     def get(self, idx):
37         data_point = self.data[idx]
38         x = data_point['new_x']
39         x = torch.tensor(x, dtype=torch.float32)
40         y = torch.tensor(data_point['y'], dtype=self.target_dtype)
41         return x, y
42
43     @property
44     def raw_file_names(self):
45         file_names = [i for i in range(19)]
46         return [f'{self.raw_data_path}/{i}.pkl' for i in file_names]
47
48     def _process(self):
49         data_list = []
50         for fn in self.raw_file_names:
51             logging.info(f'Loading {fn}')
52             with open(fn, 'rb') as f:
53                 data_slice = pickle.load(f)
54                 data_list = data_list + data_slice
55         num_samples = len(data_list)
56         logging.info(f'Loaded {num_samples} data points')
57
58         # stack and pad frames based on config
59         data_list = self.transform_keypoints(data_list)
60         data_list = self.stack_and_padd_frames(data_list)
61
62         #random shuffle train and val data

```

```

62         # random shuffle train and val data
63         random.seed(self.seed)
64         random.shuffle(data_list)
65
66         # get partitions
67         train_end = int(self.partitions[0] * num_samples)
68         val_end = train_end + int(self.partitions[1] * num_samples)
69         train_data = data_list[:train_end]
70         val_data = data_list[train_end:val_end]
71         test_data = data_list[val_end:]
72
73         data_map = {
74             'train': train_data,
75             'val': val_data,
76             'test': test_data,
77         }
78         return data_map, num_samples
79
80     def stack_and_padd_frames(self, data_list):
81         if self.stacks is None:
82             return data_list
83         # take multiple frames for each x
84         xs = [d['x'] for d in data_list]
85         stacked_xs = []
86         padded_xs = []
87         print("Stacking and padding frames...")
88         pbar = tqdm(total=len(xs))
89
90         if self.zero_padding in ['per_data_point', 'data_point']:
91             for i in range(len(xs)):
92                 data_point = []
93                 for j in range(self.stacks):
94                     if i - j >= 0:
95                         mydata_slice = xs[i - j]
96                         diff = self.max_points - mydata_slice.shape[0]
97                         mydata_slice = np.pad(mydata_slice, ((0, max(diff, 0)), (0, 0)), 'constant')
98                         mydata_slice = mydata_slice[np.random.choice(len(mydata_slice), self.max_points)]]
99                         data_point.append(mydata_slice)
100                     else:
101                         data_point.append(np.zeros((self.max_points, 3)))
102                 padded_xs.append(np.concatenate(data_point, axis=0))
103                 pbar.update(1)
104         elif self.zero_padding in ['per_stack', 'stack']:
105             for i in range(len(xs)):
106                 start = max(0, i - self.stacks)
107                 stacked_xs.append(np.concatenate(xs[start:i+1], axis=0))
108                 pbar.update(0.5)
109             for x in stacked_xs:
110                 diff = self.max_points * self.stacks - x.shape[0]
111                 x = np.pad(x, ((0, max(diff, 0)), (0, 0)), 'constant')
112                 x = x[np.random.choice(len(x), self.max_points * self.stacks,
113

```

```

114         pbar.update(0.5)
115     else:
116         raise NotImplementedError()
117     pbar.close()
118     print("Stacking and padding frames done")
119     # remap padded_xs to data_list
120     new_data_list = [{**d, 'new_x': x} for d, x in zip(data_list, padded_
121     return new_data_list
122
123     kp18_names = ['NOSE', 'NECK', 'RIGHT_SHOULDER', 'RIGHT_ELBOW',
124                  'RIGHT_WRIST', 'LEFT_SHOULDER', 'LEFT_ELBOW',
125                  'LEFT_WRIST', 'RIGHT_HIP', 'RIGHT_KNEE',
126                  'RIGHT_ANKLE', 'LEFT_HIP', 'LEFT_KNEE',
127                  'LEFT_ANKLE', 'RIGHT_EYE', 'LEFT_EYE',
128                  'RIGHT_EAR', 'LEFT_EAR']
129     kp9_names = ['RIGHT_SHOULDER', 'RIGHT_ELBOW',
130                 'LEFT_SHOULDER', 'LEFT_ELBOW',
131                 'RIGHT_HIP', 'RIGHT_KNEE',
132                 'LEFT_HIP', 'LEFT_KNEE', 'HEAD']
133     head_names = ['NOSE', 'RIGHT_EYE', 'LEFT_EYE', 'RIGHT_EAR', 'LEFT_EAR']
134     def transform_keypoints(self, data_list):
135         if self.num_keypoints == 18:
136             return data_list
137
138         print("Transforming keypoints ...")
139         self.kp9_idx = [self.kp18_names.index(n) for n in self.kp9_names[:-1]]
140         self.head_idx = [self.kp18_names.index(n) for n in self.head_names]
141         for data in tqdm(data_list, total=len(data_list)):
142             kpts = data['y']
143             kpts_new = kpts[self.kp9_idx]
144             head = np.mean(kpts[self.head_idx], axis=0)
145             kpts_new = np.concatenate((kpts_new, head[None]))
146             assert kpts_new.shape == (9, 3)
147             data['y'] = kpts_new
148         print("Transforming keypoints done")
149         return data_list
150
151     class MMRAActionData(MMRKeypointData):
152         processed_data = 'content/drive/MyDrive/action/data/processed/mmr_action/'
153         def __init__(self, *args, **kwargs):
154             self.action_label = np.load('/content/drive/MyDrive/action/data/raw/a
155             super().__init__(*args, **kwargs)
156             self.info['num_classes'] = len(np.unique(self.action_label))-1 # exce
157             self.target_dtype = torch.int64
158
159         def _process(self):
160             data_list = []
161             for fn in self.raw_file_names:
162                 logging.info(f'Loading {fn}')
163                 with open(fn, 'rb') as f:
164                     data_slice = pickle.load(f)
165                 data_list = data_list + data_slice

```

```

165         data_list = data_list + data_slice
166
167     for i, data in enumerate(data_list):
168         data['y'] = self.action_label[i]
169     data_list = [d for d in data_list if d['y']!=-1]
170
171     data_list = self.stack_and_padd_frames(data_list)
172     num_samples = len(data_list)
173     logging.info(f'Loaded {num_samples} data points')
174
175     # get partitions
176     train_end = int(self.partitions[0] * num_samples)
177     val_end = train_end + int(self.partitions[1] * num_samples)
178     train_data = data_list[:train_end]
179     val_data = data_list[train_end:val_end]
180     test_data = data_list[val_end:]
181
182     # #random shuffle train and val data
183     random.seed(self.seed)
184     random.shuffle(train_data)
185     random.shuffle(val_data)
186
187     data_map = {
188         'train': train_data,
189         'val': val_data,
190         'test': test_data,
191     }
192     return data_map, num_samples
193
194 def stack_and_padd_frames(self, data_list):
195     if self.stacks is None:
196         return data_list
197     # take multiple frames for each x
198     xs = [d['x'] for d in data_list]
199     stacked_xs = []
200     padded_xs = []
201     print("Stacking and padding frames...")
202     pbar = tqdm(total=len(xs))
203
204     if self.zero_padding in ['per_data_point', 'data_point']:
205         for i in range(len(xs)):
206             data_point = []
207             for j in range(self.stacks):
208                 if i - j >= 0 and self.action_label[i] == self.action_lab
209                     mydata_slice = xs[i - j]
210                     diff = self.max_points - mydata_slice.shape[0]
211                     mydata_slice = np.pad(mydata_slice, ((0, max(diff, 0)
212                     mydata_slice = mydata_slice[np.random.choice(len(myda
213                     data_point.append(mydata_slice)
214             else:
215                 data_point.append(np.zeros((self.max_points, 3)))
216             padded_xs.append(np.concatenate(data_point, axis=0))

```



```

217         pbar.update(1)
218     elif self.zero_padding in ['per_stack', 'stack']:
219         for i in range(len(xs)):
220             start = max(0, i - self.stacks)
221             while self.action_label[i] != self.action_label[start]:
222                 start = start + 1
223             stacked_xs.append(np.concatenate(xs[start:i+1], axis=0))
224             pbar.update(0.5)
225         for x in stacked_xs:
226             diff = self.max_points * self.stacks - x.shape[0]
227             x = np.pad(x, ((0, max(diff, 0)), (0, 0)), 'constant')
228             x = x[np.random.choice(len(x), self.max_points * self.stacks,
229                                   padded_xs.append(x)
230             pbar.update(0.5)
231     else:
232         raise NotImplementedError()
233     pbar.close()
234     print("Stacking and padding frames done")
235     # remap padded_xs to data_list
236     new_data_list = [{**d, 'new_x': x} for d, x in zip(data_list, padded_
237     return new_data_list
238
239
240 # Testing the MMRAActionData class
241 if __name__ == "__main__":
242     # Define root directory and configuration
243     root_dir = '' # Root directory is the current directory
244     mmr_dataset_config = {
245         'processed_data': '/content/drive/MyDrive/action/data/processed/mmr_a
246         'stacks': 5, # example config, adjust according to needs
247         'max_points': 22,
248         'num_keypoints': 9,
249         'zero_padding': 'per_data_point',
250         'seed': 42,
251         'forced_rewrite': True # Added line
252     }
253
254     # Load train data
255     train_dataset = MMRAActionData(root=root_dir, partition='train', mmr_datas
256     # Load validation data
257     val_dataset = MMRAActionData(root=root_dir, partition='val', mmr_dataset_c
258     # Load test data
259     test_dataset = MMRAActionData(root=root_dir, partition='test', mmr_dataset
260
261     # Print out the shapes of the train, val, and test data
262     print(f"Train data shape: {len(train_dataset)} samples")
263     print(f"Validation data shape: {len(val_dataset)} samples")
264     print(f"Test data shape: {len(test_dataset)} samples")
265
266     # Optional: inspect a specific sample (e.g., the first one) in the datase
267     x_train, y_train = train_dataset.get(0)

```

```

268     x_val, y_val = val_dataset.get(0)
269     x_test, y_test = test_dataset.get(0)
270
271     for data, label in train_dataset:
272         print(f"Train data shape: {data.shape}")
273         print(f"Train label shape: {label.shape}")
274
275     for data, label in val_dataset:
276         print(f"Val data shape: {data.shape}")
277         print(f"Val label shape: {label.shape}")
278
279     # print(f"First train sample shape: x={x_train.shape}, y={y_train.shape}")
280     # print("First train sample values:")
281     # print(f"x_train: {x_train}")
282     # print(f"y_train: {y_train}")
283
284     # print(f"First val sample shape: x={x_val.shape}, y={y_val.shape}")
285     # print("First val sample values:")
286     # print(f"x_val: {x_val}")
287     # print(f"y_val: {y_val}")
288
289     # print(f"First test sample shape: x={x_test.shape}, y={y_test.shape}")
290     # print("First test sample values:")
291     # print(f"x_test: {x_test}")
292     # print(f"y_test: {y_test}")
293
294     print(f"First train sample shape: x={x_train.shape}, y={y_train.shape}")
295     print(f"First val sample shape: x={x_val.shape}, y={y_val.shape}")
296     print(f"First test sample shape: x={x_test.shape}, y={y_test.shape}")
297

```

```

1 # import os
2 # import torch
3 # import numpy as np
4 # import pickle
5 # import logging
6 # import random
7 # from tqdm import tqdm
8 # from torch_geometric.data import [
9 # import pandas as pd
10
11 # def Normalize(x, x_min, x_max):
12 #     """Normalize a value x to a range [0, 1]"""
13 #     return (x - x_min) / (x_max - x_min)
14
15 # class MMRKeypointData(Dataset):
16 #     raw_data_path = '/content/drive/MyDrive/raw_data'
17 #     processed_data = '/content/drive/MyDrive/processed_data'
18 #     max_points = 22
19 #     seed = 42
20 #     partitions = (0.8, 0.1, 0.1)

```

"@property" is not an allowed annotation
 - allowed values include [@param, @title, @markdown].



```

21 #         stacks = None
22 #         zero_padding = 'per_data_point'
23 #         zero_padding_styles = ['per_data_point']
24 #         num_keypoints = 9
25 #         forced_rewrite = False
26
27 #     def _parse_config(self, c):
28 #         c = {k: v for k, v in c.items()}
29 #         self.seed = c.get('seed', 0)
30 #         self.processed_data = c.get('processed_data', None)
31 #         self.max_points = c.get('max_points', 1000)
32 #         self.partitions = (
33 #             c.get('train_split', 0.8),
34 #             c.get('val_split', 0.1),
35 #             c.get('test_split', 0.1)
36 #         )
37 #         self.stacks = c.get('stacks', None)
38 #         self.zero_padding = c.get('zero_padding', None)
39 #         self.num_keypoints = c.get('num_keypoints', 9)
40 #         if self.zero_padding not in ['per_data_point', 'per_keypoint']:
41 #             raise ValueError(
42 #                 f'Zero padding style {self.zero_padding} is not supported'
43 #             )
44 #         self.forced_rewrite = c.get('forced_rewrite', False)
45
46 #     def __init__(self, root, partition):
47 #         super(MMRKeypointData, self).__init__(root)
48 #         self._parse_config(mmr_data_config)
49 #         # Check if processed_data exists
50 #         if (not os.path.isfile(self.processed_data) and
51 #             self.data, _ = self._load_data()):
52 #             os.makedirs(os.path.dirname(self.processed_data))
53 #             with open(self.processed_data, 'wb') as f:
54 #                 pickle.dump(self.data, f)
55 #         else:
56 #             with open(self.processed_data, 'rb') as f:
57 #                 self.data = pickle.load(f)
58
59 #         total_samples = len(self.data)
60 #         self.data = self.data[partition['train']]
61 #         self.num_samples = len(self.data)
62 #         self.target_dtype = torch.float32
63 #         self.info = {
64 #             'num_samples': self.num_samples,
65 #             'num_keypoints': self.num_keypoints,
66 #             'num_classes': None,
67 #             'max_points': self.max_points,
68 #             'stacks': self.stacks,
69 #             'partition': partition
70 #         }
71 #         logging.info(f'Loaded {partition} partition')
72
73 #     def __len__(self):

```

```

72 #         return self.num_samples
73
74 #     def __getitem__(self, idx):
75 #         data_point = self.data[idx]
76 #         x = data_point['new_x']
77 #         x = torch.tensor(x, dtype=
78 #         y = torch.tensor(data_point
79 #         return x, y
80
81 #     @property
82 #     def raw_file_names(self):
83 #         file_names = [i for i in
84 #         return [f'{self.raw_data_p
85
86 #     def _process(self):
87 #         data_list = []
88 #         for fn in self.raw_file_na
89 #             logging.info(f'Loading
90 #             with open(fn, 'rb') as
91 #                 data_slice = pickl
92 #                 data_list += data_slic
93 #             num_samples = len(data_list
94 #             logging.info(f'Loaded {num
95
96 #         # Stack and pad frames bas
97 #         data_list = self.transform
98 #         data_list = self.stack_and
99
100 #         # Random shuffle train and
101 #         random.seed(self.seed)
102 #         random.shuffle(data_list)
103
104 #         # Get partitions
105 #         train_end = int(self.parti
106 #         val_end = train_end + int(
107 #         train_data = data_list[:tr
108 #         val_data = data_list[train
109 #         test_data = data_list[val_
110
111 #         data_map = {
112 #             'train': train_data,
113 #             'val': val_data,
114 #             'test': test_data,
115 #         }
116 #         return data_map, num_samp
117
118 #     def stack_and_padd_frames(self
119 #         if self.stacks is None:
120 #             return data_list
121 #         # Take multiple frames for
122 #         xs = [d['x'] for d in data

```

```

123 #         stacked_xs = []
124 #         padded_xs = []
125 #         print("Stacking and padding")
126 #         pbar = tqdm(total=len(xs))
127
128 #         if self.zero_padding in ['none', 'right']:
129 #             for i in range(len(xs)):
130 #                 data_point = []
131 #                 for j in range(self.window_size):
132 #                     if i - j >= 0:
133 #                         mydata_slice = xs[i - j]
134 #                         diff = self.window_size - (i - j)
135 #                         mydata_slice = np.zeros(diff) + mydata_slice
136 #                         data_point.append(mydata_slice)
137 #                     else:
138 #                         data_point.append(0)
139 #                 padded_xs.append(data_point)
140 #                 pbar.update(1)
141 #         else:
142 #             raise NotImplementedError
143 #         pbar.close()
144 #         print("Stacking and padding complete")
145 #         # Remap padded_xs to data_list
146 #         new_data_list = [{**d, 'new_data_list': padded_xs[i]} for i, d in enumerate(data_list)]
147 #         return new_data_list
148
149 #         kp18_names = ['NOSE', 'NECK', 'RIGHT_SHOULDER', 'LEFT_SHOULDER', 'RIGHT_HIP', 'LEFT_HIP',
150 #                       'RIGHT_WRIST', 'LEFT_WRIST', 'RIGHT_ANKLE', 'LEFT_ANKLE', 'RIGHT_EAR', 'LEFT_EAR']
151 #         kp9_names = ['RIGHT_SHOULDER', 'LEFT_SHOULDER', 'RIGHT_HIP', 'LEFT_HIP', 'RIGHT_KNEE', 'LEFT_KNEE',
152 #                      'RIGHT_ANKLE', 'LEFT_ANKLE', 'RIGHT_FOOT', 'LEFT_FOOT']
153 #
154 #         def transform_keypoints(self, data_list):
155 #             if self.num_keypoints == 18:
156 #                 return data_list
157 #
158 #             print("Transforming keypoints")
159 #             self.kp9_idx = [self.kp18_names.index(kp) for kp in self.kp9_names]
160 #             for data in tqdm(data_list):
161 #                 kpts = data['y']
162 #                 kpts_new = kpts[self.kp9_idx]
163 #                 head = np.mean(kpts[self.kp9_idx], axis=0)
164 #                 kpts_new = np.concatenate([kpts_new, head], axis=-1)
165 #                 assert kpts_new.shape == (self.num_keypoints, self.num_channels)
166 #                 data['y'] = kpts_new
167 #             print("Transforming keypoints complete")

```

```

174 #         return data_list
175
176
177 # class MMRActionData(MMRKeypointData):
178 #     processed_data = '/content/drive/MyDrive/ActionData/'
179
180 #     def __init__(self, *args, **kwargs):
181 #         self.action_label = np.zeros((len(args),))
182 #         super().__init__(*args, **kwargs)
183 #         self.info['num_classes'] = self.action_label.shape[0]
184 #         self.target_dtype = torch.long
185
186 #         # Verify labels: Check shape
187 #         print(f"Action labels shape: {self.action_label.shape}")
188 #         print(f"Unique action labels: {np.unique(self.action_label)}")
189
190 #     def _process(self):
191 #         data_list = []
192 #         for fn in self.raw_file_names:
193 #             logging.info(f'Loading {fn}')
194 #             with open(fn, 'rb') as f:
195 #                 data_slice = pickle.load(f)
196 #                 data_list += data_slice
197
198 #         for i, data in enumerate(data_list):
199 #             data['y'] = self.action_label[i]
200 #             # Verify label assignment
201 #             if data['y'] == -1:
202 #                 print(f"Warning: Label -1 for {fn}")
203
204 #         data_list = [d for d in data_list if d['y'] != -1]
205
206 #         # Normalization step (before stacking)
207 #         self.normalize_features(data_list)
208
209 #         num_samples = len(data_list)
210 #         logging.info(f'Loaded {num_samples} samples')
211
212 #         data_list = self.stack_and_normalize(data_list)
213
214 #         # Get partitions
215 #         train_end = int(self.partition_ratio * num_samples)
216 #         val_end = train_end + int(self.partition_ratio * num_samples)
217 #         train_data = data_list[:train_end]
218 #         val_data = data_list[train_end:val_end]
219 #         test_data = data_list[val_end:]
220
221 #         # Random shuffle train and validation sets
222 #         random.seed(self.seed)
223 #         random.shuffle(train_data)
224 #         random.shuffle(val_data)

```

```

225
226 #         data_map = {
227 #             'train': train_data,
228 #             'val': val_data,
229 #             'test': test_data,
230 #         }
231 #         return data_map, num_samples
232
233 #     def normalize_features(self, data_list):
234 #         """Normalize intensity features"""
235 #         intensity_values = np.array([data['intensity'] for data in data_list])
236 #         intensity_min = intensity_values.min()
237 #         intensity_max = intensity_values.max()
238
239 #         for data in data_list:
240 #             data['normalized_intensity'] = (data['intensity'] - intensity_min) / (intensity_max - intensity_min)
241
242 #         print(f"Normalized intensity range: {intensity_min} to {intensity_max}")
243
244 # # Testing the MMRAActionData class
245 # if __name__ == "__main__":
246 #     # Define root directory and configuration
247 #     root_dir = '' # Root directory
248 #     mmr_dataset_config = {
249 #         'processed_data': '/content',
250 #         'stacks': 5, # Example configuration
251 #         'max_points': 22,
252 #         'num_keypoints': 9,
253 #         'zero_padding': 'per_data_point',
254 #         'seed': 42,
255 #         'forced_rewrite': True
256 #     }
257
258 #     # Load train data
259 #     train_dataset = MMRAActionData(root_dir, mmr_dataset_config)
260 #     # Load validation data
261 #     val_dataset = MMRAActionData(root_dir, mmr_dataset_config)
262 #     # Load test data
263 #     test_dataset = MMRAActionData(root_dir, mmr_dataset_config)
264
265 #     # Print out the shapes of the datasets
266 #     print(f"Train data shape: {len(train_dataset)}")
267 #     print(f"Validation data shape: {len(val_dataset)}")
268 #     print(f"Test data shape: {len(test_dataset)}")
269

```

```

1 import os
2 import torch
3 import numpy as np
4 import pickle

```

```

5 import logging
6 import random
7 from tqdm import tqdm
8 from torch_geometric.data import Dataset
9 from sklearn.cluster import DBSCAN # Import DBSCAN
10 import matplotlib.pyplot as plt
11 from mpl_toolkits.mplot3d import Axes3D
12
13 class MMRKeypointData(Dataset):
14     raw_data_path = '/content/drive/MyDrive/action/data/raw' # Updated path
15     processed_data = '/content/drive/MyDrive/action/data/processed/mmr_kp/d'
16     max_points = 22
17     seed = 42
18     partitions = (0.8, 0.1, 0.1)
19     stacks = None
20     zero_padding = 'per_data_point'
21     zero_padding_styles = ['per_data_point', 'per_stack', 'data_point', 'sta
22     num_keypoints = 9
23     forced_rewrite = False
24
25     def _parse_config(self, c):
26         c = {k: v for k, v in c.items() if v is not None}
27         self.seed = c.get('seed', self.seed)
28         self.processed_data = c.get('processed_data', self.processed_data)
29         self.max_points = c.get('max_points', self.max_points)
30         self.partitions = (
31             c.get('train_split', self.partitions[0]),
32             c.get('val_split', self.partitions[1]),
33             c.get('test_split', self.partitions[2]))
34         self.stacks = c.get('stacks', self.stacks)
35         self.zero_padding = c.get('zero_padding', self.zero_padding)
36         self.num_keypoints = c.get('num_keypoints', self.num_keypoints)
37         if self.zero_padding not in self.zero_padding_styles:
38             raise ValueError(
39                 f'Zero padding style {self.zero_padding} not supported.')
40         self.forced_rewrite = c.get('forced_rewrite', self.forced_rewrite)
41
42     def __init__(
43         self, root, partition,
44         transform=None, pre_transform=None, pre_filter=None,
45         mmr_dataset_config = None):
46         super(MMRKeypointData, self).__init__(
47             root, transform, pre_transform, pre_filter)
48         self._parse_config(mmr_dataset_config)
49         # Check if processed_data exists
50         if (not os.path.isfile(self.processed_data)) or self.forced_rewrite:
51             self.data, _ = self._process()
52             # Create directory if it doesn't exist
53             os.makedirs(os.path.dirname(self.processed_data), exist_ok=True)
54             with open(self.processed_data, 'wb') as f:
55                 pickle.dump(self.data, f)

```



```

56         else:
57             with open(self.processed_data, 'rb') as f:
58                 self.data = pickle.load(f)
59             total_samples = len(self.data['train']) + len(self.data['val']) + len(self.data['test'])
60             self.data = self.data[partition]
61             self.num_samples = len(self.data)
62             self.target_dtype = torch.float
63             self.info = {
64                 'num_samples': self.num_samples,
65                 'num_keypoints': self.num_keypoints,
66                 'num_classes': None,
67                 'max_points': self.max_points,
68                 'stacks': self.stacks,
69                 'partition': partition,
70             }
71             logging.info(
72                 f'Loaded {partition} data with {self.num_samples} samples, '
73                 f'where the total number of samples is {total_samples}')
74
75     def len(self):
76         return self.num_samples
77
78     def get(self, idx):
79         data_point = self.data[idx]
80         x = data_point['new_x']
81         x = torch.tensor(x, dtype=torch.float32)
82         y = torch.tensor(data_point['y'], dtype=self.target_dtype)
83         return x, y
84
85     @property
86     def raw_file_names(self):
87         file_names = [i for i in range(19)]
88         return [f'{self.raw_data_path}/{i}.pkl' for i in file_names]
89
90     def _process(self):
91         data_list = []
92         for fn in self.raw_file_names:
93             logging.info(f'Loading {fn}')
94             with open(fn, 'rb') as f:
95                 data_slice = pickle.load(f)
96                 data_list = data_list + data_slice
97         num_samples = len(data_list)
98         logging.info(f'Loaded {num_samples} data points')
99
100         # Transform keypoints based on config
101         data_list = self.transform_keypoints(data_list)
102
103         # Stack and pad frames
104         data_list = self.stack_and_padd_frames(data_list)
105
106         # Apply DBSCAN clustering

```

```

107     data_list = self.apply_dbscan(data_list)
108
109     # Random shuffle train and val data
110     random.seed(self.seed)
111     random.shuffle(data_list)
112
113     # Get partitions
114     train_end = int(self.partitions[0] * num_samples)
115     val_end = train_end + int(self.partitions[1] * num_samples)
116     train_data = data_list[:train_end]
117     val_data = data_list[train_end:val_end]
118     test_data = data_list[val_end:]
119
120     data_map = {
121         'train': train_data,
122         'val': val_data,
123         'test': test_data,
124     }
125     return data_map, num_samples
126
127 def stack_and_padd_frames(self, data_list):
128     if self.stacks is None:
129         return data_list
130     # Take multiple frames for each x
131     xs = [d['x'] for d in data_list]
132     padded_xs = []
133     print("Stacking and padding frames...")
134     pbar = tqdm(total=len(xs))
135
136     if self.zero_padding in ['per_data_point', 'data_point']:
137         for i in range(len(xs)):
138             data_point = []
139             for j in range(self.stacks):
140                 if i - j >= 0:
141                     mydata_slice = xs[i - j]
142                     diff = self.max_points - mydata_slice.shape[0]
143                     mydata_slice = np.pad(mydata_slice, ((0, max(diff, 0)),))
144                     if mydata_slice.shape[0] > self.max_points:
145                         idx = np.random.choice(mydata_slice.shape[0], size=1)
146                         mydata_slice = mydata_slice[idx:]
147                     data_point.append(mydata_slice)
148                 else:
149                     data_point.append(np.zeros((self.max_points, 3)))
150             padded_xs.append(np.concatenate(data_point, axis=0))
151             pbar.update(1)
152     elif self.zero_padding in ['per_stack', 'stack']:
153         stacked_xs = []
154         for i in range(len(xs)):
155             start = max(0, i - self.stacks + 1)
156             stacked_xs.append(np.concatenate(xs[start:i+1], axis=0))
157         pbar.update(0.5)

```

```

158         for x in stacked_xs:
159             diff = self.max_points * self.stacks - x.shape[0]
160             x = np.pad(x, ((0, max(diff, 0)), (0, 0)), 'constant')
161             if x.shape[0] > self.max_points * self.stacks:
162                 idx = np.random.choice(x.shape[0], self.max_points * sel
163                 x = x[idx]
164             padded_xs.append(x)
165             pbar.update(0.5)
166         else:
167             raise NotImplementedError()
168         pbar.close()
169         print("Stacking and padding frames done")
170         # Remap padded_xs to data_list
171         new_data_list = [{**d, 'new_x': x} for d, x in zip(data_list, padded
172         return new_data_list
173
174     # Modified apply_dbscan method
175     def apply_dbscan(self, data_list):
176         print("Applying DBSCAN clustering...")
177         desired_num_points = self.max_points * (self.stacks if self.stacks <
178         for data in tqdm(data_list, total=len(data_list)):
179             x = data['new_x'] # Shape: [num_points, num_features], e.g., [1
180             # Store the data before clustering
181             data['new_x_before_dbscan'] = x.copy()
182             # Apply DBSCAN clustering
183             clustering = DBSCAN(eps=0.5, min_samples=3).fit(x) # Updated pa
184             labels = clustering.labels_
185             # Keep only the points that are in clusters (labels != -1)
186             mask = labels != -1
187             x_filtered = x[mask]
188             # Handle cases where x_filtered is empty or has too few/many poi
189             num_points = x_filtered.shape[0]
190             if num_points == 0:
191                 # All points are noise; pad with zeros
192                 x_filtered = np.zeros((desired_num_points, x.shape[1]))
193             elif num_points < desired_num_points:
194                 # Pad with zeros
195                 diff = desired_num_points - num_points
196                 x_filtered = np.pad(x_filtered, ((0, diff), (0, 0)), 'consta
197             elif num_points > desired_num_points:
198                 # Randomly sample desired_num_points
199                 idx = np.random.choice(num_points, desired_num_points, repla
200                 x_filtered = x_filtered[idx]
201             # Else, num_points == desired_num_points; no change needed
202             data['new_x'] = x_filtered
203         print("DBSCAN clustering applied.")
204         return data_list
205
206     kp18_names = ['NOSE', 'NECK', 'RIGHT_SHOULDER', 'RIGHT_ELBOW',
207                  'RIGHT_WRIST', 'LEFT_SHOULDER', 'LEFT_ELBOW',
208                  'LEFT_WRIST', 'RIGHT_HIP', 'RIGHT_KNEE',

```

```

209         'RIGHT_ANKLE', 'LEFT_HIP', 'LEFT_KNEE',
210         'LEFT_ANKLE', 'RIGHT_EYE', 'LEFT_EYE',
211         'RIGHT_EAR', 'LEFT_EAR']
212     kp9_names = ['RIGHT_SHOULDER', 'RIGHT_ELBOW',
213                 'LEFT_SHOULDER', 'LEFT_ELBOW',
214                 'RIGHT_HIP', 'RIGHT_KNEE',
215                 'LEFT_HIP', 'LEFT_KNEE', 'HEAD']
216     head_names = ['NOSE', 'RIGHT_EYE', 'LEFT_EYE', 'RIGHT_EAR', 'LEFT_EAR']
217
218     def transform_keypoints(self, data_list):
219         if self.num_keypoints == 18:
220             return data_list
221
222         print("Transforming keypoints ...")
223         self.kp9_idx = [self.kp18_names.index(n) for n in self.kp9_names[:-1]]
224         self.head_idx = [self.kp18_names.index(n) for n in self.head_names]
225         for data in tqdm(data_list, total=len(data_list)):
226             kpts = data['y']
227             kpts_new = kpts[self.kp9_idx]
228             head = np.mean(kpts[self.head_idx], axis=0)
229             kpts_new = np.concatenate((kpts_new, head[None]))
230             assert kpts_new.shape == (9, 3)
231             data['y'] = kpts_new
232         print("Transforming keypoints done")
233         return data_list
234
235     class MMRActionData(MMRKeypointData):
236         processed_data = '/content/drive/MyDrive/action/data/processed/mmr_action_data.pkl'
237         def __init__(self, *args, **kwargs):
238             self.action_label = np.load('/content/drive/MyDrive/action/data/raw/action_label.npy')
239             super().__init__(*args, **kwargs)
240             self.info['num_classes'] = len(np.unique(self.action_label))-1 # exclude -1
241             self.target_dtype = torch.int64
242
243         def _process(self):
244             data_list = []
245             for fn in self.raw_file_names:
246                 logging.info(f'Loading {fn}')
247                 with open(fn, 'rb') as f:
248                     data_slice = pickle.load(f)
249                     data_list = data_list + data_slice
250
251             for i, data in enumerate(data_list):
252                 data['y'] = self.action_label[i]
253             data_list = [d for d in data_list if d['y'] != -1]
254
255             data_list = self.stack_and_padd_frames(data_list)
256
257             # Apply DBSCAN clustering
258             data_list = self.apply_dbscan(data_list)
259

```

```

260     num_samples = len(data_list)
261     logging.info(f'Loaded {num_samples} data points')
262
263     # Get partitions
264     train_end = int(self.partitions[0] * num_samples)
265     val_end = train_end + int(self.partitions[1] * num_samples)
266     train_data = data_list[:train_end]
267     val_data = data_list[train_end:val_end]
268     test_data = data_list[val_end:]
269
270     # Random shuffle train and val data
271     random.seed(self.seed)
272     random.shuffle(train_data)
273     random.shuffle(val_data)
274
275     data_map = {
276         'train': train_data,
277         'val': val_data,
278         'test': test_data,
279     }
280     return data_map, num_samples
281
282 # Testing the MMRAActionData class with DBSCAN clustering and visualization
283 if __name__ == "__main__":
284     # Define root directory and configuration
285     root_dir = '' # Root directory is the current directory
286     mmr_dataset_config = {
287         'processed_data': '/content/drive/MyDrive/action/data/processed/mmr_
288         'stacks': 5, # Example config, adjust according to needs
289         'max_points': 22,
290         'num_keypoints': 9,
291         'zero_padding': 'per_data_point',
292         'seed': 42,
293         'forced_rewrite': True # Set to True to process data again
294     }
295
296     # Load train data
297     train_dataset = MMRAActionData(root=root_dir, partition='train', mmr_data
298     # Load validation data
299     val_dataset = MMRAActionData(root=root_dir, partition='val', mmr_dataset_
300     # Load test data
301     test_dataset = MMRAActionData(root=root_dir, partition='test', mmr_datase
302
303     # Print out the shapes of the train, val, and test data
304     print(f"Train data shape: {len(train_dataset)} samples")
305     print(f"Validation data shape: {len(val_dataset)} samples")
306     print(f"Test data shape: {len(test_dataset)} samples")
307
308     # Visualization of 5 random sequences
309     # Combine datasets for selection (you can choose from any partition)
310     combined_data = train_dataset.data + val_dataset.data + test_dataset.dat

```

```

311     selected_sequences = random.sample(combined_data, 5)
312
313     for idx, data_point in enumerate(selected_sequences):
314         x_before = data_point['new_x_before_dbscan']
315         x_after = data_point['new_x']
316
317         fig = plt.figure(figsize=(12, 6))
318
319         # Plot before DBSCAN
320         ax1 = fig.add_subplot(121, projection='3d')
321         ax1.scatter(x_before[:, 0], x_before[:, 1], x_before[:, 2], c='b', n
322         ax1.set_title(f'Sequence {idx+1} Before DBSCAN')
323         ax1.set_xlabel('X')
324         ax1.set_ylabel('Y')
325         ax1.set_zlabel('Z')
326         ax1.view_init(elev=20., azimuth=-35)
327
328         # Plot after DBSCAN
329         ax2 = fig.add_subplot(122, projection='3d')
330         ax2.scatter(x_after[:, 0], x_after[:, 1], x_after[:, 2], c='r', mark
331         ax2.set_title(f'Sequence {idx+1} After DBSCAN')
332         ax2.set_xlabel('X')
333         ax2.set_ylabel('Y')
334         ax2.set_zlabel('Z')
335         ax2.view_init(elev=20., azimuth=-35)
336
337         plt.tight_layout()
338         plt.show()
339
340

```



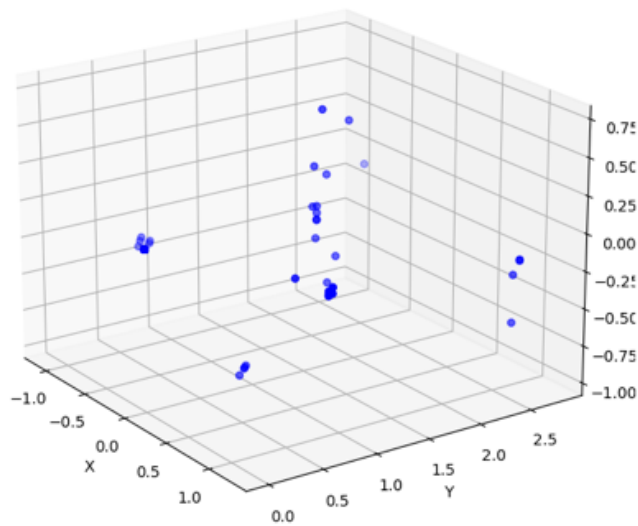
```

Stacking and padding frames...
100%|██████████| 212920/212920 [00:51<00:00, 4161.67it/s]
Stacking and padding frames done
Applying DBSCAN clustering...
100%|██████████| 212920/212920 [05:44<00:00, 617.68it/s]
DBSCAN clustering applied.
Stacking and padding frames...
100%|██████████| 212920/212920 [00:55<00:00, 3841.03it/s]
Stacking and padding frames done
Applying DBSCAN clustering...
100%|██████████| 212920/212920 [05:49<00:00, 609.07it/s]
DBSCAN clustering applied.
Stacking and padding frames...
100%|██████████| 212920/212920 [00:56<00:00, 3759.31it/s]
Stacking and padding frames done
Applying DBSCAN clustering...
100%|██████████| 212920/212920 [05:46<00:00, 615.13it/s]
DBSCAN clustering applied.
Train data shape: 170336 samples
Validation data shape: 21292 samples
Test data shape: 21292 samples

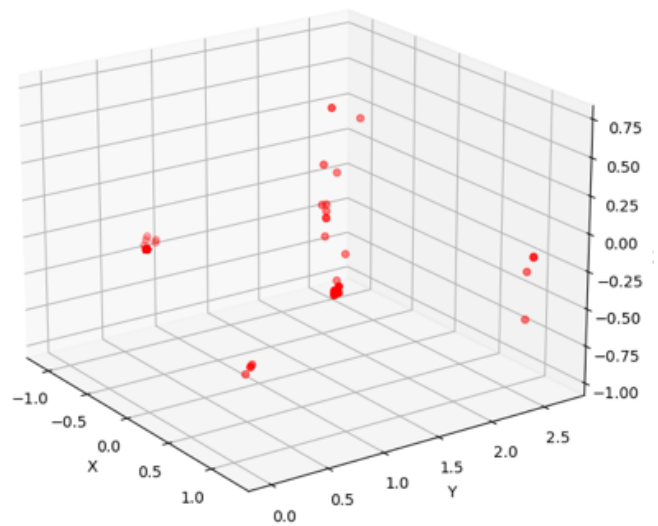
```

Sequence 1 Before DBSCAN

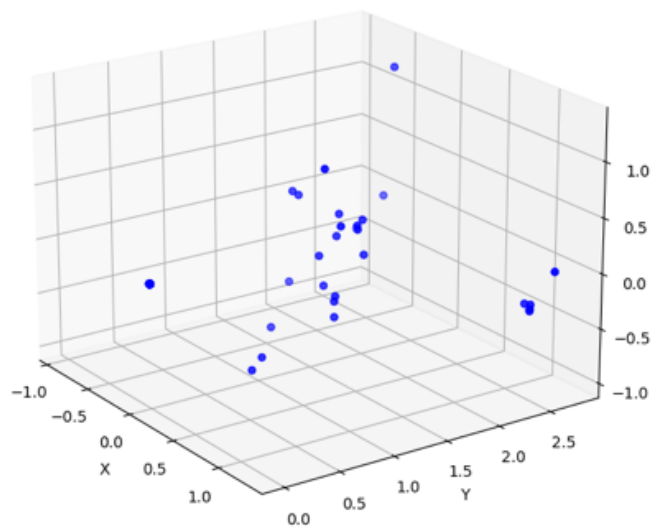
Sequence 1 After DBSCAN



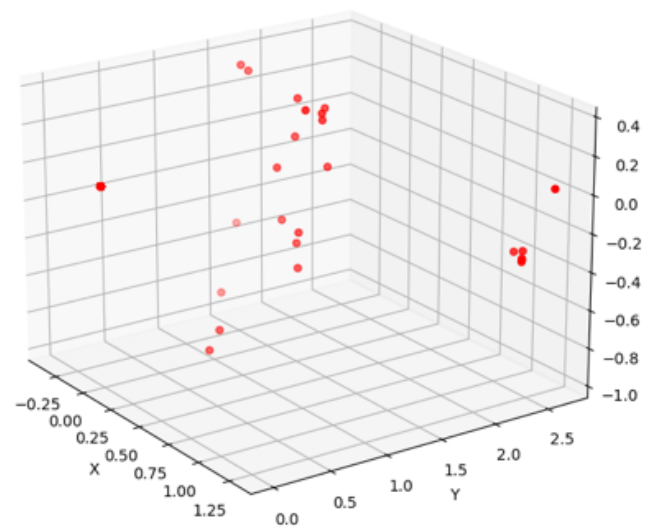
Sequence 2 Before DBSCAN



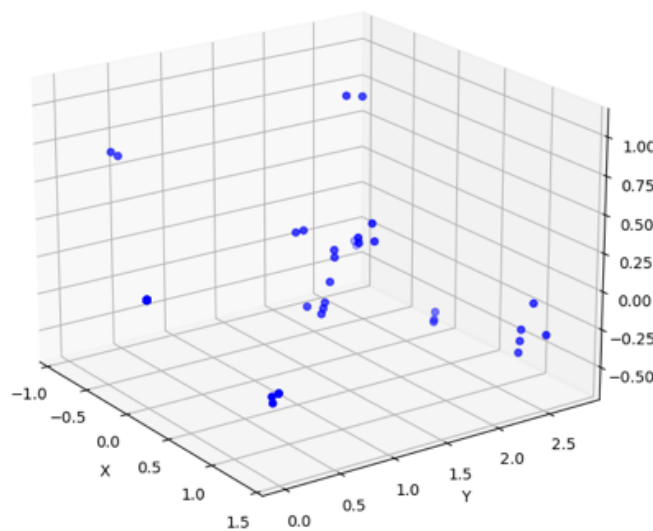
Sequence 2 After DBSCAN



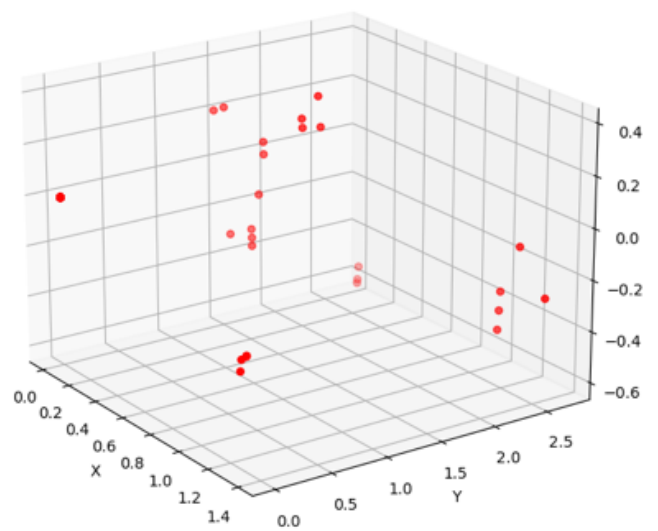
Sequence 3 Before DBSCAN



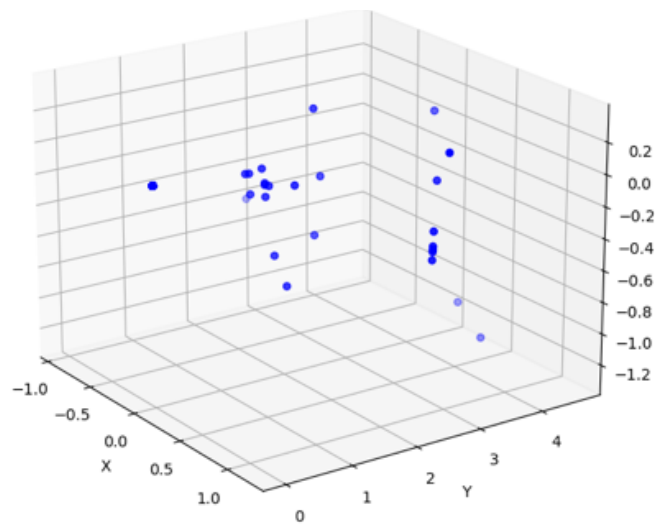
Sequence 3 After DBSCAN



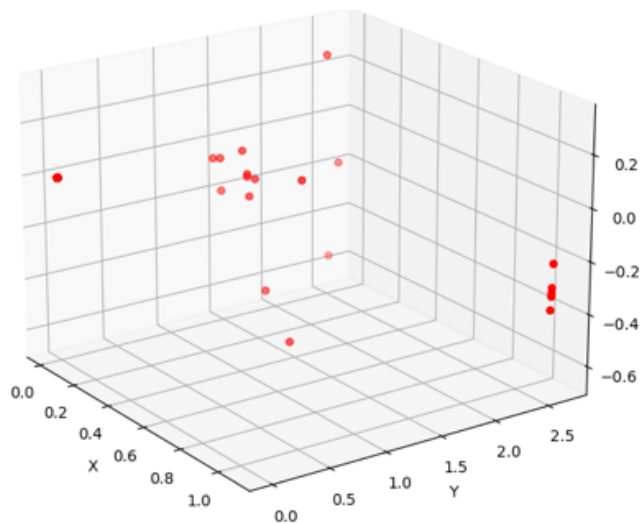
Sequence 4 Before DBSCAN



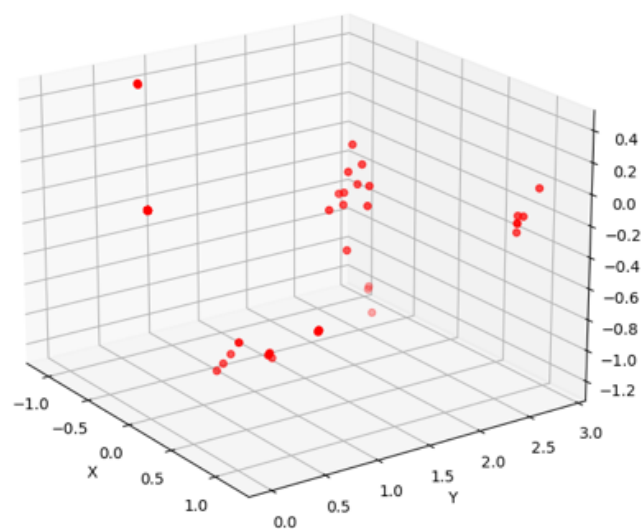
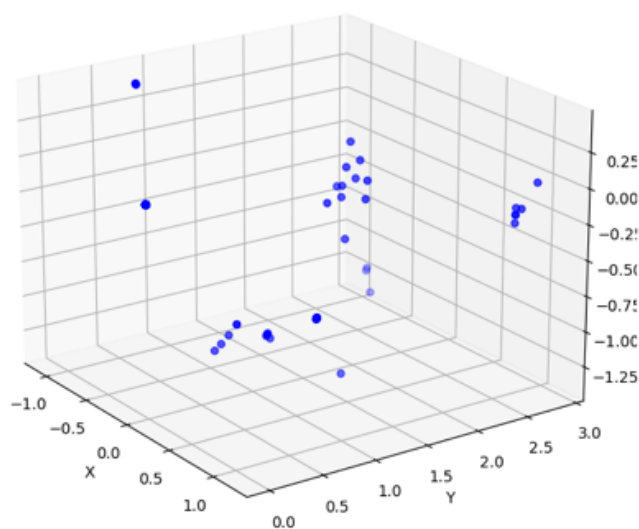
Sequence 4 After DBSCAN



Sequence 5 Before DBSCAN



Sequence 5 After DBSCAN




```

1 # import os
2 # import torch
3 # import numpy as np
4 # import pickle
5 # import logging
6 # import random
7 # from tqdm import tqdm
8 # from torch_geometric.data import [
9 # from sklearn.cluster import DBSCAN
10 # import matplotlib.pyplot as plt
11 # from mpl_toolkits.mplot3d import /
12
13 # class MMRKeypointData(Dataset):
14 #     raw_data_path = '/content/driv
15 #     processed_data = '/content/dr
16 #     max_points = 22
17 #     seed = 42
18 #     partitions = (0.8, 0.1, 0.1)
19 #     stacks = None
20 #     zero_padding = 'per_data_point
21 #     zero_padding_styles = ['per_da
22 #     num_keypoints = 9
23 #     forced_rewrite = False
24
25 #     def _parse_config(self, c):
26 #         c = {k: v for k, v in c.it
27 #         self.seed = c.get('seed',
28 #         self.processed_data = c.ge
29 #         self.max_points = c.get('n
30 #         self.partitions = (
31 #             c.get('train_split', s
32 #             c.get('val_split', sel
33 #             c.get('test_split', se
34 #         self.stacks = c.get('stack
35 #         self.zero_padding = c.get(
36 #         self.num_keypoints = c.get
37 #         if self.zero_padding not :
38 #             raise ValueError(f'Zer
39 #         self.forced_rewrite = c.ge
40
41 #     def __init__(self, root, parti
42 #         transform=None, p
43 #         super(MMRKeypointData, sel
44 #         if mmr_dataset_config is r
45 #             self._parse_config(mmr

```

"@property" is not an allowed annotation
 - allowed values include [@param, @title, @markdown].



```

46
47 #         if (not os.path.isfile(self
48 #             self.data, _ = self._p
49 #             os.makedirs(os.path.di
50 #             with open(self.process
51 #                 pickle.dump(self.c
52 #         else:
53 #             with open(self.process
54 #                 self.data = pickle
55
56 #         total_samples = len(self.c
57 #         self.data = self.data[part
58 #         self.num_samples = len(sel
59 #         self.target_dtype = torch.
60 #         self.info = {
61 #             'num_samples': self.nu
62 #             'num_keypoints': self.
63 #             'num_classes': None,
64 #             'max_points': self.ma
65 #             'stacks': self.stacks,
66 #             'partition': partition
67 #         }
68 #         logging.info(
69 #             f'Loaded {partition} c
70 #             f' where the total num
71
72 #     def len(self):
73 #         return self.num_samples
74
75 #     def get(self, idx):
76 #         data_point = self.data[idx]
77 #         x = data_point['new_x']
78 #         x = torch.tensor(x, dtype=
79 #         y = torch.tensor(data_poi
80 #         return x, y
81
82 #     @property
83 #     def raw_file_names(self):
84 #         file_names = [i for i in r
85 #         return [f'{self.raw_data_p
86
87 #     def _process(self):
88 #         data_list = []
89 #         for fn in self.raw_file_na
90 #             logging.info(f'Loading
91 #             with open(fn, 'rb') as
92 #                 data_slice = pickl
93 #                 data_list = data_list
94 #             num_samples = len(data_lis
95 #             logging.info(f'Loaded {num
96

```

```

97 #         # Transform keypoints base
98 #         data_list = self.transform
99
100 #         # Stack and pad frames
101 #         data_list = self.stack_and_padd
102
103 #         # Apply DBSCAN clustering
104 #         data_list = self.apply_dbSCAN
105
106 #         # Random shuffle train and val
107 #         random.seed(self.seed)
108 #         random.shuffle(data_list)
109
110 #         # Get partitions
111 #         train_end = int(self.partition)
112 #         val_end = train_end + int(self.val_ratio)
113 #         train_data = data_list[:train_end]
114 #         val_data = data_list[train_end:val_end]
115 #         test_data = data_list[val_end:]
116
117 #         data_map = {
118 #             'train': train_data,
119 #             'val': val_data,
120 #             'test': test_data,
121 #         }
122 #         return data_map, num_samples
123
124 #     def stack_and_padd_frames(self, data_list):
125 #         if self.stacks is None:
126 #             return data_list
127 #         xs = [d['x'] for d in data_list]
128 #         padded_xs = []
129 #         print("Stacking and padding frames")
130 #         pbar = tqdm(total=len(xs))
131
132 #         if self.zero_padding in ['none', 'right']:
133 #             for i in range(len(xs)):
134 #                 data_point = []
135 #                 for j in range(self.stacks):
136 #                     if i - j >= 0:
137 #                         mydata_slice = xs[i - j]
138 #                         diff = self.stacks - i + j
139 #                         mydata_slice = mydata_slice[-diff:]
140 #                         if mydata_slice:
141 #                             idx = len(mydata_slice) - 1
142 #                             mydata_slice[idx] = 0
143 #                         data_point.append(mydata_slice)
144 #                     else:
145 #                         data_point.append([0] * self.channels)
146 #                 padded_xs.append(np.array(data_point))
147 #                 pbar.update(1)

```

```

148 #         elif self.zero_padding in
149 #             stacked_xs = []
150 #             for i in range(len(xs)):
151 #                 start = max(0, i -
152 #                     stacked_xs.append(
153 #                         pbar.update(0.5)
154 #             for x in stacked_xs:
155 #                 diff = self.max_po
156 #                 x = np.pad(x, ((0,
157 #                     if x.shape[0] > se
158 #                     idx = np.randc
159 #                     x = x[idx]
160 #                 padded_xs.append(>
161 #                 pbar.update(0.5)
162 #         else:
163 #             raise NotImplementedError
164 #         pbar.close()
165 #         print("Stacking and paddin
166 #         new_data_list = [{**d, 'ne
167 #         return new_data_list
168
169 #     def apply_dbscan(self, data_li
170 #         print("Applying DBSCAN cl
171 #         desired_num_points = self.
172 #         for data in tqdm(data_list
173 #             x = data['new_x'] # 5
174 #             data['new_x_before_db
175 #             clustering = DBSCAN(eps
176 #             labels = clustering.la
177 #             mask = labels != -1
178 #             x_filtered = x[mask]
179 #             num_points = x_filtere
180 #             if num_points == 0:
181 #                 x_filtered = np.zer
182 #             elif num_points < des:
183 #                 diff = desired_num
184 #                 x_filtered = np.pa
185 #             elif num_points > des:
186 #                 idx = np.random.ch
187 #                 x_filtered = x_fil
188 #             data['new_x'] = x_filt
189 #             print("DBSCAN clustering a
190 #             return data_list
191
192 #     kp18_names = ['NOSE', 'NECK',
193 #                   'RIGHT_WRIST', '
194 #                   'LEFT_WRIST', 'F
195 #                   'RIGHT_ANKLE', '
196 #                   'LEFT_ANKLE', 'F
197 #                   'RIGHT_EAR', 'LE
198 #     kp9_names = ['RIGHT_SHOULDER',

```

```

199 #             'LEFT_SHOULDER',
200 #             'RIGHT_HIP', 'RIGHT_SHOULDER',
201 #             'LEFT_HIP', 'LEFT_SHOULDER']
202 #     head_names = ['NOSE', 'RIGHT_EYE', 'RIGHT_EAR', 'RIGHT_SHOULDER',
203 #                   'RIGHT_HIP', 'RIGHT_LEG', 'LEFT_EYE', 'LEFT_EAR',
204 #                   'LEFT_SHOULDER', 'LEFT_HIP', 'LEFT_LEG']
205 #
206 #     def transform_keypoints(self, data_list):
207 #         if self.num_keypoints == 1:
208 #             return data_list
209 #
210 #         print("Transforming keypoints")
211 #         self.kp9_idx = [self.kp18_idx, self.kp19_idx, self.kp20_idx,
212 #                         self.kp21_idx, self.kp22_idx, self.kp23_idx,
213 #                         self.kp24_idx, self.kp25_idx, self.kp26_idx]
214 #         self.head_idx = [self.kp18_idx, self.kp19_idx, self.kp20_idx,
215 #                          self.kp21_idx, self.kp22_idx, self.kp23_idx,
216 #                          self.kp24_idx, self.kp25_idx, self.kp26_idx]
217 #         for data in tqdm(data_list):
218 #             kpts = data['y']
219 #             kpts_new = kpts[self.kp9_idx + self.head_idx]
220 #             head = np.mean(kpts[self.head_idx])
221 #             kpts_new = np.concatenate([kpts_new, head])
222 #             assert kpts_new.shape == (self.num_keypoints + 1, 2)
223 #             data['y'] = kpts_new
224 #         print("Transforming keypoints done")
225 #         return data_list
226 #
227 # class MMRAActionData(MMRKeypointData):
228 #     processed_data = '/content/drive/My Drive/processed_data'
229 #
230 #     def __init__(self, *args, **kwargs):
231 #         self.action_label = np.zeros(self.num_classes)
232 #         super().__init__(*args, **kwargs)
233 #         self.info['num_classes'] = self.num_classes
234 #         self.target_dtype = torch.float32
235 #
236 #     def _process(self):
237 #         data_list = []
238 #         for fn in self.raw_file_names:
239 #             logging.info(f'Loading {fn}')
240 #             with open(fn, 'rb') as f:
241 #                 data_slice = pickle.load(f)
242 #                 data_list.append(data_slice)
243 #
244 #         for i, data in enumerate(data_list):
245 #             data['y'] = self.action_label[i]
246 #         data_list = [d for d in data_list]
247 #
248 #         data_list = self.stack_and_shuffle(data_list)
249 #         data_list = self.apply_dropout(data_list)
250 #
251 #         num_samples = len(data_list)
252 #         logging.info(f'Loaded {num_samples} samples')
253 #
254 #         train_end = int(self.partition * num_samples)
255 #         val_end = train_end + int(self.validation * num_samples)

```

```

250 #         train_data = data_list[:train_data_size]
251 #         val_data = data_list[train_data_size:train_data_size+val_data_size]
252 #         test_data = data_list[val_data_size:]
253
254 #         random.seed(self.seed)
255 #         random.shuffle(train_data)
256 #         random.shuffle(val_data)
257
258 #         data_map = {
259 #             'train': train_data,
260 #             'val': val_data,
261 #             'test': test_data,
262 #         }
263 #         return data_map, num_samples
264
265 # Cluster analysis method
266 # def cluster_analysis(self):
267 #     labels = [data['new_x'] for data in self.data]
268 #     total_points = len(labels)
269 #     unique_clusters = np.unique(labels)
270 #     num_clusters = len(unique_clusters)
271
272 #     print('Total:', total_points)
273 #     for i in range(num_clusters):
274 #         print('Cluster', i, ':')
275 #         print('Noise:', np.sum(labels[i] != unique_clusters[i]))
276
277 # # Testing the MMRAActionData class
278 # if __name__ == "__main__":
279 #     # Define root directory and config
280 #     root_dir = ''
281 #     mmr_dataset_config = {
282 #         'processed_data': '/content',
283 #         'stacks': 5,
284 #         'max_points': 22,
285 #         'num_keypoints': 9,
286 #         'zero_padding': 'per_data',
287 #         'seed': 42,
288 #         'forced_rewrite': True
289 #     }
290 #     if __name__ == '__main__':
291 #         X,y = make_moons(100)
292 #         model = DBSCAN()
293 #         preds = model.fit_predict(X)
294 #         # Either low or high values are predicted
295 #         print(f"Accuracy: {round((sum(preds == y) / len(preds)), 2)}")
296
297
298 # Load train, validation, and test data
299 # train_dataset = MMRAActionData(root_dir, mmr_dataset_config)
300 # val_dataset = MMRAActionData(root_dir, mmr_dataset_config)

```

```

301 #     test_dataset = MMRAActionData(
302
303 #     # Print out the shapes of the
304 #     print(f"Train data shape: {len(
305 #     print(f"Validation data shape:
306 #     print(f"Test data shape: {len(
307
308 #     # Perform cluster analysis aft
309 #     train_dataset.cluster_analysis
310 #     val_dataset.cluster_analysis()
311 #     test_dataset.cluster_analysis(
312
313 #     # Visualization of 5 random se
314 #     combined_data = train_dataset.
315 #     selected_sequences = random.sa
316
317 #     for idx, data_point in enumera
318 #         x_before = data_point['nev
319 #         x_after = data_point['new_
320
321 #         fig = plt.figure(figsize=(
322
323 #         # Plot before DBSCAN
324 #         ax1 = fig.add_subplot(121,
325 #         ax1.scatter(x_before[:, 0]
326 #         ax1.set_title(f'Sequence +
327 #         ax1.set_xlabel('X')
328 #         ax1.set_ylabel('Y')
329 #         ax1.set_zlabel('Z')
330 #         ax1.view_init(elev=20., az
331
332 #         # Plot after DBSCAN
333 #         ax2 = fig.add_subplot(122,
334 #         ax2.scatter(x_after[:, 0],
335 #         ax2.set_title(f'Sequence +
336 #         ax2.set_xlabel('X')
337 #         ax2.set_ylabel('Y')
338 #         ax2.set_zlabel('Z')
339 #         ax2.view_init(elev=20., az
340
341 #         plt.tight_layout()
342 #         plt.show()
343

```

```

1 import os
2 import torch
3 import numpy as np
4 import pickle
5 import logging
6 import random

```

```

7 from tqdm import tqdm
8 from torch_geometric.data import Dataset
9 from sklearn.cluster import DBSCAN
10 from sklearn.metrics import silhouette_score, calinski_harabasz_score
11 import matplotlib.pyplot as plt
12 from mpl_toolkits.mplot3d import Axes3D
13 import seaborn as sns
14 import pandas as pd
15
16 class MMRActionData(Dataset):
17     raw_data_path = '/content/drive/MyDrive/action/data/raw'
18     processed_data = '/content/drive/MyDrive/action/data/processed/mmr_actic
19     max_points = 22
20     seed = 42
21     partitions = (0.8, 0.1, 0.1)
22     stacks = None
23     zero_padding = 'per_data_point'
24     zero_padding_styles = ['per_data_point', 'per_stack', 'data_point', 'sta
25     num_keypoints = 9
26     forced_rewrite = False
27
28     def _parse_config(self, c):
29         c = {k: v for k, v in c.items() if v is not None}
30         self.seed = c.get('seed', self.seed)
31         self.processed_data = c.get('processed_data', self.processed_data)
32         self.max_points = c.get('max_points', self.max_points)
33         self.partitions = (
34             c.get('train_split', self.partitions[0]),
35             c.get('val_split', self.partitions[1]),
36             c.get('test_split', self.partitions[2]))
37         self.stacks = c.get('stacks', self.stacks)
38         self.zero_padding = c.get('zero_padding', self.zero_padding)
39         self.num_keypoints = c.get('num_keypoints', self.num_keypoints)
40         if self.zero_padding not in self.zero_padding_styles:
41             raise ValueError(f'Zero padding style {self.zero_padding} not su
42         self.forced_rewrite = c.get('forced_rewrite', self.forced_rewrite)
43
44     def __init__(self, root, partition, mmr_dataset_config=None,
45                 transform=None, pre_transform=None, pre_filter=None):
46         self.partition = partition
47         self.metrics = {} # Store DBSCAN metrics
48
49         # Load action labels before super().__init__
50         try:
51             self.action_label = np.load(f'{self.raw_data_path}/action_label.
52         except FileNotFoundError:
53             print(f"Warning: Could not find action_label.npy in {self.raw_da
54             self.action_label = None
55
56         if mmr_dataset_config is not None:
57             self._parse_config(mmr_dataset_config)

```



```

58
59     super(MMRActionData, self).__init__(root, transform, pre_transform,
60
61     if (not os.path.isfile(self.processed_data)) or self.forced_rewrite:
62         self.data, _ = self._process()
63         os.makedirs(os.path.dirname(self.processed_data), exist_ok=True)
64         with open(self.processed_data, 'wb') as f:
65             pickle.dump(self.data, f)
66     else:
67         with open(self.processed_data, 'rb') as f:
68             self.data = pickle.load(f)
69
70     total_samples = len(self.data['train']) + len(self.data['val']) + len(self.data['test'])
71     self.data = self.data[partition]
72     self.num_samples = len(self.data)
73     self.target_dtype = torch.int64
74
75     self.info = {
76         'num_samples': self.num_samples,
77         'num_keypoints': self.num_keypoints,
78         'num_classes': len(np.unique(self.action_label)) - 1 if self.action_label is not None else 0,
79         'max_points': self.max_points,
80         'stacks': self.stacks,
81         'partition': partition,
82     }
83
84     logging.info(
85         f'Loaded {partition} data with {self.num_samples} samples, '
86         f'where the total number of samples is {total_samples}')
87
88     @property
89     def raw_file_names(self):
90         file_names = [i for i in range(19)]
91         return [f'{self.raw_data_path}/{i}.pkl' for i in file_names]
92
93     @property
94     def processed_file_names(self):
95         return [os.path.basename(self.processed_data)]
96
97     def process(self):
98         pass
99
100    def len(self):
101        return self.num_samples
102
103    def get(self, idx):
104        data_point = self.data[idx]
105        x = data_point['new_x']
106        x = torch.tensor(x, dtype=torch.float32)
107        y = torch.tensor(data_point['y'], dtype=self.target_dtype)
108        return x, y

```

```
109
110 # [Previous code remains the same until _process method]
111
112 def _process(self):
113     data_list = []
114     for fn in self.raw_file_names:
115         logging.info(f'Loading {fn}')
116         try:
117             with open(fn, 'rb') as f:
118                 data_slice = pickle.load(f)
119                 data_list = data_list + data_slice
120         except FileNotFoundError:
121             print(f"Warning: Could not find {fn}")
122             continue
123
124     num_samples = len(data_list)
125     logging.info(f'Loaded {num_samples} data points')
126
127     # First transform keypoints
128     data_list = self.transform_keypoints(data_list)
129
130     # Then assign action labels
131     if self.action_label is not None:
132         for i, data in enumerate(data_list):
133             if i < len(self.action_label):
134                 data['y'] = self.action_label[i]
135             else:
136                 print(f"Warning: No action label for data point {i}")
137                 data['y'] = -1
138         data_list = [d for d in data_list if d['y'] != -1]
139
140     # Stack and pad frames
141     data_list = self.stack_and_padd_frames(data_list)
142
143     # Apply DBSCAN clustering
144     data_list = self.apply_dbscan(data_list)
145
146     num_samples = len(data_list)
147     logging.info(f'Processed {num_samples} data points')
148
149     # Get partitions
150     train_end = int(self.partitions[0] * num_samples)
151     val_end = train_end + int(self.partitions[1] * num_samples)
152
153     random.seed(self.seed)
154     random.shuffle(data_list)
155
156     train_data = data_list[:train_end]
157     val_data = data_list[train_end:val_end]
158     test_data = data_list[val_end:]
159
```

```

160     data_map = {
161         'train': train_data,
162         'val': val_data,
163         'test': test_data,
164     }
165     return data_map, num_samples
166
167 def transform_keypoints(self, data_list):
168     if self.num_keypoints == 18:
169         return data_list
170
171     print("Transforming keypoints ...")
172     self.kp9_idx = [self.kp18_names.index(n) for n in self.kp9_names[:-1]]
173     self.head_idx = [self.kp18_names.index(n) for n in self.head_names]
174
175     transformed_list = []
176     for data in tqdm(data_list, total=len(data_list)):
177         try:
178             if isinstance(data['y'], (np.ndarray, list)) and len(data['y
179                 kpts = np.array(data['y'])
180                 kpts_new = kpts[self.kp9_idx]
181                 head = np.mean(kpts[self.head_idx], axis=0)
182                 kpts_new = np.concatenate((kpts_new, head[None]))
183
184                 if kpts_new.shape == (9, 3): # Verify correct shape
185                     data['y'] = kpts_new
186                     transformed_list.append(data)
187                 else:
188                     print(f"Warning: Skipping data point with incorrect
189             else:
190                 print(f"Warning: Skipping data point with invalid keypoi
191         except Exception as e:
192             print(f"Warning: Error transforming keypoints: {str(e)}")
193             continue
194
195     print(f"Transformed {len(transformed_list)} keypoints out of {len(da
196     return transformed_list
197
198 # [Rest of the code remains the same]
199 def stack_and_padd_frames(self, data_list):
200     if self.stacks is None:
201         return data_list
202     xs = [d['x'] for d in data_list]
203     padded_xs = []
204     print("Stacking and padding frames...")
205     pbar = tqdm(total=len(xs))
206
207     if self.zero_padding in ['per_data_point', 'data_point']:
208         for i in range(len(xs)):
209             data_point = []
210             for j in range(self.stacks):

```

```

211         if i - j >= 0:
212             mydata_slice = xs[i - j]
213             diff = self.max_points - mydata_slice.shape[0]
214             mydata_slice = np.pad(mydata_slice, ((0, max(diff, 0)), (0, 0)), 'constant')
215             if mydata_slice.shape[0] > self.max_points:
216                 idx = np.random.choice(mydata_slice.shape[0], self.max_points)
217                 mydata_slice = mydata_slice[idx]
218             data_point.append(mydata_slice)
219         else:
220             data_point.append(np.zeros((self.max_points, 3)))
221             padded_xs.append(np.concatenate(data_point, axis=0))
222             pbar.update(1)
223     elif self.zero_padding in ['per_stack', 'stack']:
224         stacked_xs = []
225         for i in range(len(xs)):
226             start = max(0, i - self.stacks + 1)
227             stacked_xs.append(np.concatenate(xs[start:i+1], axis=0))
228             pbar.update(0.5)
229         for x in stacked_xs:
230             diff = self.max_points * self.stacks - x.shape[0]
231             x = np.pad(x, ((0, max(diff, 0)), (0, 0)), 'constant')
232             if x.shape[0] > self.max_points * self.stacks:
233                 idx = np.random.choice(x.shape[0], self.max_points * self.stacks)
234                 x = x[idx]
235             padded_xs.append(x)
236             pbar.update(0.5)
237     else:
238         raise NotImplementedError()
239
240     pbar.close()
241     print("Stacking and padding frames done")
242     new_data_list = [{**d, 'new_x': x} for d, x in zip(data_list, padded_xs)]
243     return new_data_list
244
245     def apply_dbscan(self, data_list):
246         print("Applying DBSCAN clustering...")
247         desired_num_points = self.max_points * (self.stacks if self.stacks > 1 else 1)
248         all_metrics = []
249
250         for data in tqdm(data_list, total=len(data_list)):
251             x = data['new_x']
252             data['new_x_before_dbscan'] = x.copy()
253
254             clustering = DBSCAN(eps=0.5, min_samples=3)
255             labels = clustering.fit_predict(x)
256
257             # Calculate metrics
258             sequence_metrics = self._calculate_sequence_metrics(x, labels)
259             all_metrics.append(sequence_metrics)
260
261             # Filter points

```

```

262     mask = labels != -1
263     x_filtered = x[mask]
264     num_points = x_filtered.shape[0]
265
266     if num_points == 0:
267         x_filtered = np.zeros((desired_num_points, x.shape[1]))
268     elif num_points < desired_num_points:
269         diff = desired_num_points - num_points
270         x_filtered = np.pad(x_filtered, ((0, diff), (0, 0)), 'constant')
271     elif num_points > desired_num_points:
272         idx = np.random.choice(num_points, desired_num_points, replace=True)
273         x_filtered = x_filtered[idx]
274
275     data['new_x'] = x_filtered
276     data['dbscan_labels'] = labels
277     data['metrics'] = sequence_metrics
278
279     self.metrics = self._calculate_overall_metrics(all_metrics)
280     print("DBSCAN clustering applied.")
281     return data_list
282
283 def _calculate_sequence_metrics(self, x, labels):
284     n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
285     n_noise = list(labels).count(-1)
286
287     valid_points = labels != -1
288     if sum(valid_points) > 1 and len(set(labels[valid_points])) > 1:
289         try:
290             silhouette = silhouette_score(x[valid_points], labels[valid_points])
291             calinski = calinski_harabasz_score(x[valid_points], labels[valid_points])
292         except:
293             silhouette = calinski = 0
294     else:
295         silhouette = calinski = 0
296
297     return {
298         'n_clusters': n_clusters,
299         'n_noise': n_noise,
300         'noise_ratio': n_noise / len(x),
301         'silhouette_score': silhouette,
302         'calinski_score': calinski,
303         'total_points': len(x),
304         'valid_points': sum(valid_points)
305     }
306
307 def _calculate_overall_metrics(self, all_metrics):
308     overall = {}
309     for key in all_metrics[0].keys():
310         if key in ['n_clusters', 'n_noise', 'total_points', 'valid_points']:
311             overall[key] = sum(m[key] for m in all_metrics)
312         else:

```

```

313         overall[key] = np.mean([m[key] for m in all_metrics])
314
315     overall['sequences_analyzed'] = len(all_metrics)
316     return overall
317
318     kp18_names = ['NOSE', 'NECK', 'RIGHT_SHOULDER', 'RIGHT_ELBOW',
319                  'RIGHT_WRIST', 'LEFT_SHOULDER', 'LEFT_ELBOW',
320                  'LEFT_WRIST', 'RIGHT_HIP', 'RIGHT_KNEE',
321                  'RIGHT_ANKLE', 'LEFT_HIP', 'LEFT_KNEE',
322                  'LEFT_ANKLE', 'RIGHT_EYE', 'LEFT_EYE',
323                  'RIGHT_EAR', 'LEFT_EAR']
324     kp9_names = ['RIGHT_SHOULDER', 'RIGHT_ELBOW',
325                  'LEFT_SHOULDER', 'LEFT_ELBOW',
326                  'RIGHT_HIP', 'RIGHT_KNEE',
327                  'LEFT_HIP', 'LEFT_KNEE', 'HEAD']
328     head_names = ['NOSE', 'RIGHT_EYE', 'LEFT_EYE', 'RIGHT_EAR', 'LEFT_EAR']
329
330     def transform_keypoints(self, data_list):
331         if self.num_keypoints == 18:
332             return data_list
333
334         print("Transforming keypoints ...")
335         self.kp9_idx = [self.kp18_names.index(n) for n in self.kp9_names[:-1]]
336         self.head_idx = [self.kp18_names.index(n) for n in self.head_names]
337
338         for data in tqdm(data_list, total=len(data_list)):
339             kpts = data['y']
340             kpts_new = kpts[self.kp9_idx]
341             head = np.mean(kpts[self.head_idx], axis=0)
342             kpts_new = np.concatenate((kpts_new, head[None]))
343             assert kpts_new.shape == (9, 3)
344             data['y'] = kpts_new
345
346         print("Transforming keypoints done")
347         return data_list
348
349     def cluster_analysis(self):
350         if not hasattr(self, 'metrics') or not self.metrics:
351             print("No clustering metrics available. Run DBSCAN first.")
352             return
353
354         print("\n=== DBSCAN Analysis Results ===")
355         print(f"\nAnalyzed {self.metrics['sequences_analyzed']} sequences")
356
357         print("\nOverall Statistics:")
358         print(f"Total points processed: {self.metrics['total_points']}")
359         print(f"Total valid points: {self.metrics['valid_points']}")
360         print(f"Total noise points: {self.metrics['n_noise']}")
361         print(f"Average noise ratio: {self.metrics['noise_ratio']:.2%}")
362
363         print("\nClustering Quality:")

```

```

364     print(f"Average clusters per sequence: {self.metrics['n_clusters']}/s
365     print(f"Average silhouette score: {self.metrics['silhouette_score']}:
366     print(f"Average Calinski-Harabasz score: {self.metrics['calinski_scc
367
368     def visualize_sequence(self, data_point):
369         x_before = data_point['new_x_before_dbscan']
370         x_after = data_point['new_x']
371
372         fig = plt.figure(figsize=(12, 6))
373
374         # Plot before DBSCAN
375         ax1 = fig.add_subplot(121, projection='3d')
376         ax1.scatter(x_before[:, 0], x_before[:, 1], x_before[:, 2], c='b', n
377         ax1.set_title('Before DBSCAN')
378         ax1.set_xlabel('X')
379         ax1.set_ylabel('Y')
380         ax1.set_zlabel('Z')
381         ax1.view_init(elev=20., azimuth=-35)
382
383         # Plot after DBSCAN
384         ax2 = fig.add_subplot(122, projection='3d')
385         ax2.scatter(x_after[:, 0], x_after[:, 1], x_after[:, 2], c='r', mark
386         ax2.set_title('After DBSCAN')
387         ax2.set_xlabel('X')
388         ax2.set_ylabel('Y')
389         ax2.set_zlabel('Z')
390         ax2.view_init(elev=20., azimuth=-35)
391
392         plt.tight_layout()
393         plt.show()
394
395         # Print sequence metrics if available
396         if 'metrics' in data_point:
397             print("\nSequence Metrics:")
398             for k, v in data_point['metrics'].items():
399                 print(f"{k}: {v:.3f}" if isinstance(v, float) else f"{k}: {\v
400
401 if __name__ == "__main__":
402     # Define root directory and configuration
403     root_dir = ''
404     mmr_dataset_config = {
405         'processed_data': '/content/drive/MyDrive/action/data/processed/mmr_
406         'stacks': 5,
407         'max_points': 22,
408         'num_keypoints': 9,
409         'zero_padding': 'per_data_point',
410         'seed': 42,
411         'forced_rewrite': True
412     }
413
414     try:

```

```

415     # Create processed directory if it doesn't exist
416     os.makedirs(os.path.dirname(mmr_dataset_config['processed_data']), exist_ok=True)
417
418     print("Loading datasets...")
419     # Load datasets
420     train_dataset = MMRAActionData(root=root_dir, partition='train',
421                                     mmr_dataset_config=mmr_dataset_config)
422     val_dataset = MMRAActionData(root=root_dir, partition='val',
423                                   mmr_dataset_config=mmr_dataset_config)
424     test_dataset = MMRAActionData(root=root_dir, partition='test',
425                                    mmr_dataset_config=mmr_dataset_config)
426
427     # Print dataset sizes
428     print(f"\nDataset sizes:")
429     print(f"Train data: {len(train_dataset)} samples")
430     print(f"Validation data: {len(val_dataset)} samples")
431     print(f"Test data: {len(test_dataset)} samples")
432
433     # Perform cluster analysis
434     print("\nPerforming cluster analysis...")
435     train_dataset.cluster_analysis()
436     val_dataset.cluster_analysis()
437     test_dataset.cluster_analysis()
438
439     # Visualize random sequences
440     print("\nVisualizing random sequences...")
441     combined_data = train_dataset.data + val_dataset.data + test_dataset.data
442     selected_sequences = random.sample(combined_data, 5)
443
444     for idx, data_point in enumerate(selected_sequences):
445         print(f"\nVisualizing Sequence {idx+1}")
446         train_dataset.visualize_sequence(data_point)
447
448     except Exception as e:
449         print(f"An error occurred: {str(e)}")
450         raise

```



```

Loading datasets...
Transforming keypoints ...
100%|██████████| 545059/545059 [00:21<00:00, 25633.52it/s]
Transforming keypoints done
Stacking and padding frames...
100%|██████████| 212920/212920 [00:50<00:00, 4251.16it/s]
Stacking and padding frames done
Applying DBSCAN clustering...
100%|██████████| 212920/212920 [12:37<00:00, 280.91it/s]
DBSCAN clustering applied.
Transforming keypoints ...
100%|██████████| 545059/545059 [00:13<00:00, 39402.62it/s]
Transforming keypoints done
Stacking and padding frames...
100%|██████████| 212920/212920 [00:49<00:00, 4260.10it/s]
Stacking and padding frames done

```



```
Applying DBSCAN clustering...
100%|██████████| 212920/212920 [12:32<00:00, 282.83it/s]
DBSCAN clustering applied.
Transforming keypoints ...
100%|██████████| 545059/545059 [00:22<00:00, 24101.17it/s]
Transforming keypoints done
Stacking and padding frames...
100%|██████████| 212920/212920 [00:50<00:00, 4249.32it/s]
Stacking and padding frames done
Applying DBSCAN clustering...
100%|██████████| 212920/212920 [12:39<00:00, 280.52it/s]
DBSCAN clustering applied.
Transforming keypoints ...
100%|██████████| 545059/545059 [00:14<00:00, 38779.07it/s]
Transforming keypoints done
Stacking and padding frames...
100%|██████████| 212920/212920 [00:51<00:00, 4171.42it/s]
Stacking and padding frames done
Applying DBSCAN clustering...
100%|██████████| 212920/212920 [12:37<00:00, 280.94it/s]
DBSCAN clustering applied.
Transforming keypoints ...
100%|██████████| 545059/545059 [00:23<00:00, 23145.42it/s]
Transforming keypoints done
Stacking and padding frames...
100%|██████████| 212920/212920 [00:50<00:00, 4250.64it/s]
Stacking and padding frames done
Applying DBSCAN clustering...
100%|██████████| 212920/212920 [12:56<00:00, 274.11it/s]
DBSCAN clustering applied.
Transforming keypoints ...
100%|██████████| 545059/545059 [00:14<00:00, 38660.96it/s]
Transforming keypoints done
Stacking and padding frames...
100%|██████████| 212920/212920 [00:50<00:00, 4202.22it/s]
Stacking and padding frames done
Applying DBSCAN clustering...
100%|██████████| 212920/212920 [13:07<00:00, 270.54it/s]
DBSCAN clustering applied.
```

Dataset sizes:

Train data: 170336 samples

Validation data: 21292 samples

Test data: 21292 samples

Performing cluster analysis...

=== DBSCAN Analysis Results ===

Analyzed 212920 sequences

Overall Statistics:

Total points processed: 23421200

Total valid points: 22295010

Total noise points: 1126190

Average noise ratio: 4.81%

Clustering Quality:

Clustering Quality:
Average clusters per sequence: 4.54
Average silhouette score: 0.933
Average Calinski-Harabasz score: 1762.260

=== DBSCAN Analysis Results ===

Analyzed 212920 sequences

Overall Statistics:

Total points processed: 23421200
Total valid points: 22295010
Total noise points: 1126190
Average noise ratio: 4.81%

Clustering Quality:

Average clusters per sequence: 4.54
Average silhouette score: 0.933
Average Calinski-Harabasz score: 1762.260

=== DBSCAN Analysis Results ===

Analyzed 212920 sequences

Overall Statistics:

Total points processed: 23421200
Total valid points: 22295010
Total noise points: 1126190
Average noise ratio: 4.81%

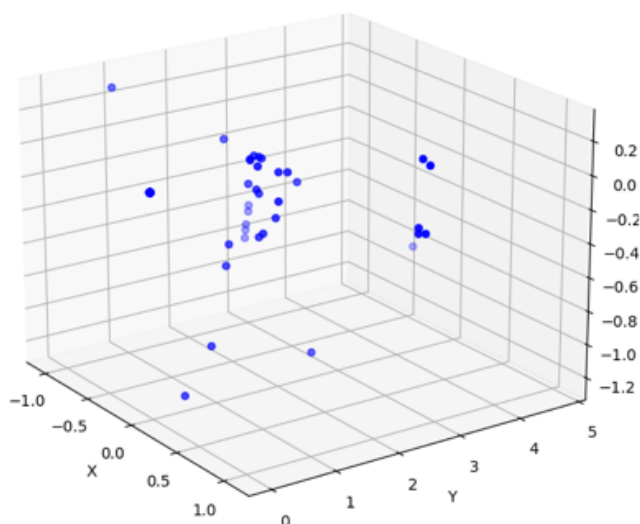
Clustering Quality:

Average clusters per sequence: 4.54
Average silhouette score: 0.933
Average Calinski-Harabasz score: 1762.260

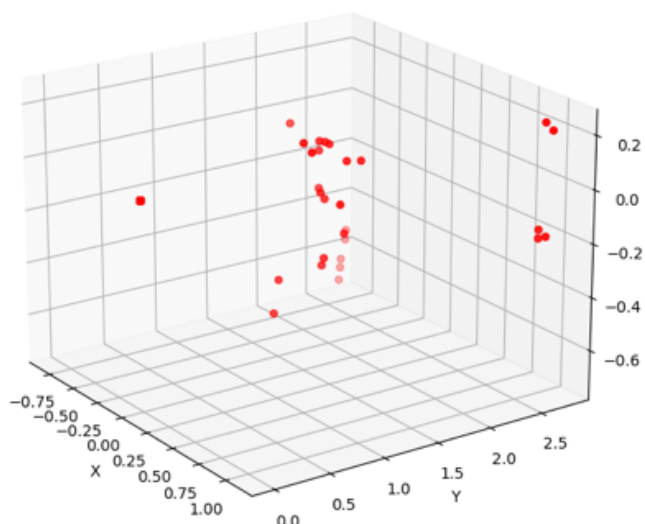
Visualizing random sequences...

Visualizing Sequence 1

Before DBSCAN

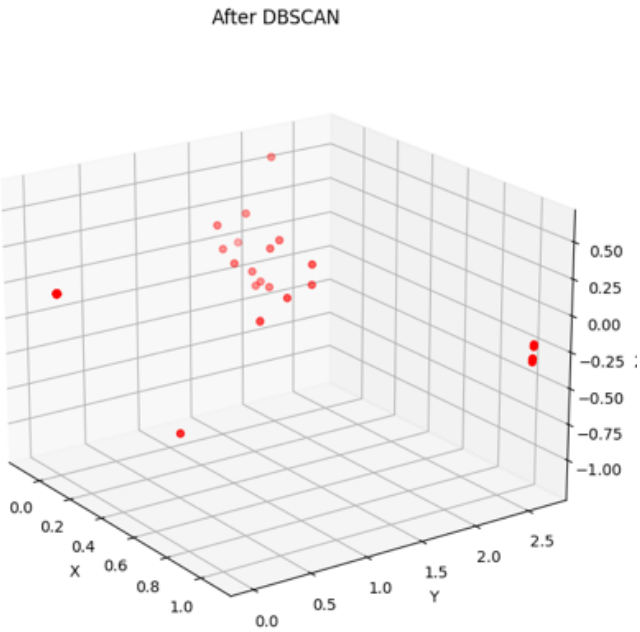
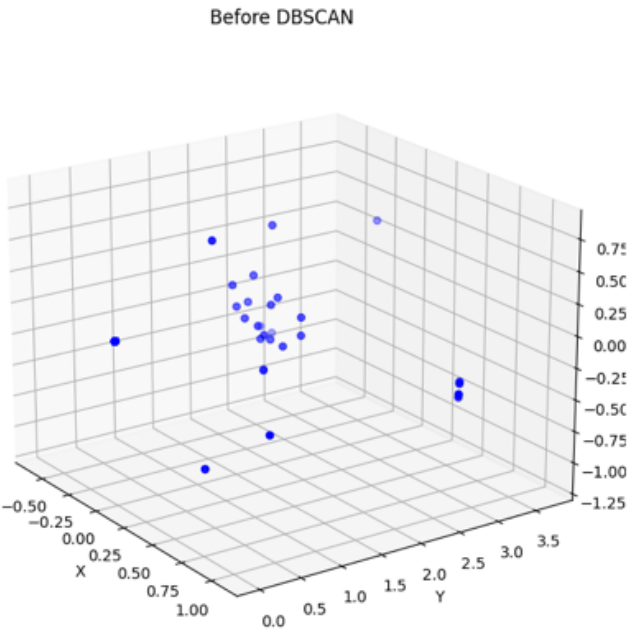


After DBSCAN



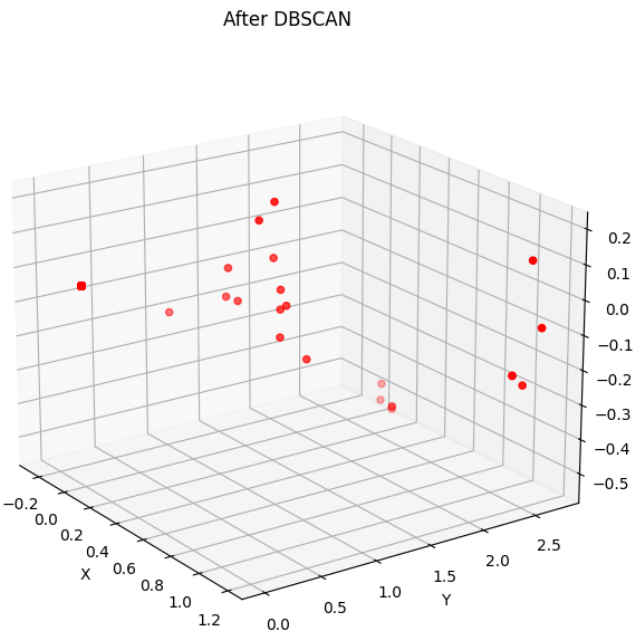
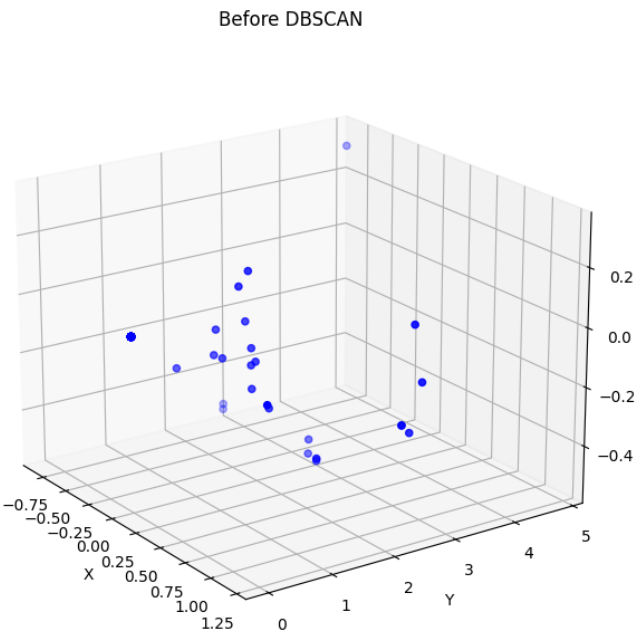
Sequence Metrics:
n_clusters: 4
n_noise: 6
noise_ratio: 0.055
silhouette_score: 0.931
calinski_score: 1546.800
total_points: 110
valid_points: 104

Visualizing Sequence 2



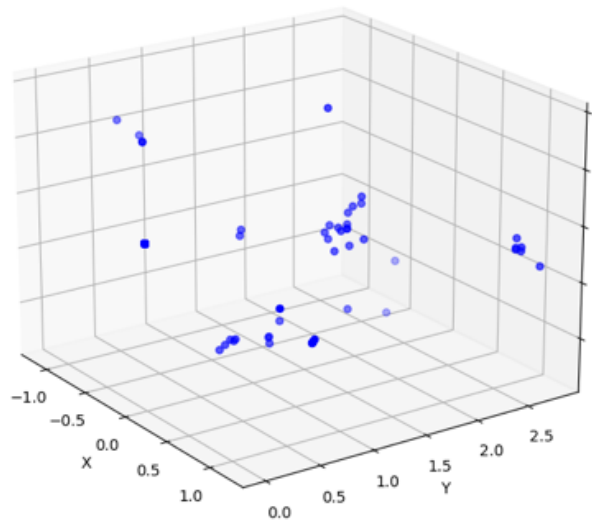
Sequence Metrics:
n_clusters: 4
n_noise: 5
noise_ratio: 0.045
silhouette_score: 0.947
calinski_score: 1729.891
total_points: 110
valid_points: 105

Visualizing Sequence 3

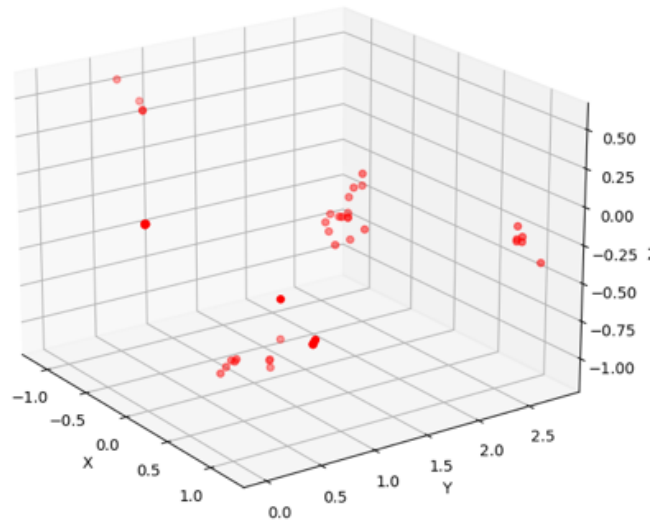


Sequence Metrics:
n_clusters: 4
n_noise: 4
noise_ratio: 0.036
silhouette_score: 0.957
calinski_score: 3174.033
total_points: 110
valid_points: 106

Visualizing Sequence 4
Before DBSCAN

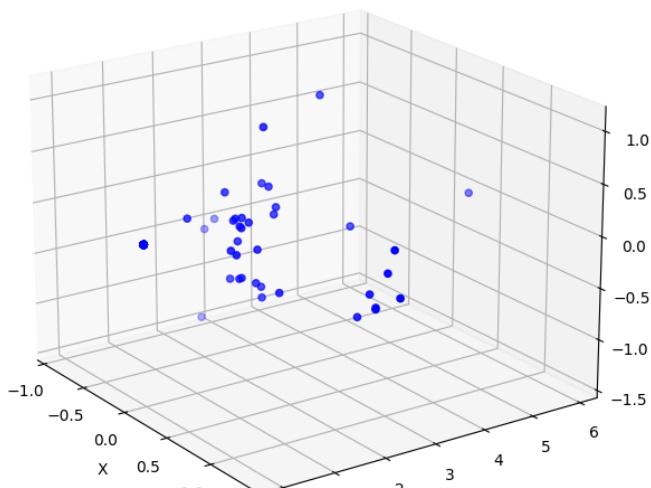


After DBSCAN

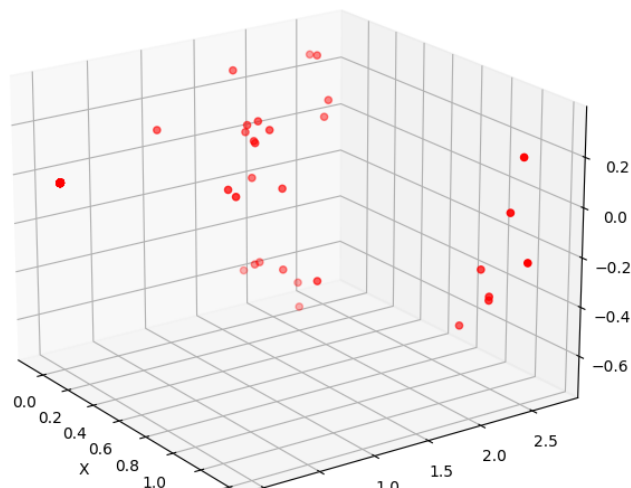


Sequence Metrics:
n_clusters: 6
n_noise: 6
noise_ratio: 0.055
silhouette_score: 0.935
calinski_score: 1811.647
total_points: 110
valid_points: 104

Visualizing Sequence 5
Before DBSCAN



After DBSCAN



1.0 0 1 4 Y

1.2 1.4 0.0 0.5 Y

Sequence Metrics:
n_clusters: 4
n_noise: 7
noise_ratio: 0.064
silhouette_score: 0.855
calinski_score: 795.872
total_points: 110
valid_points: 103

```
1 !pip install filterpy
```

```
➦ Collecting filterpy
  Downloading filterpy-1.4.5.zip (177 kB)
178.0/178.0 kB 5.4 MB/s eta 0
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.1
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/di
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.1
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/pytho
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-p
Building wheels for collected packages: filterpy
  Building wheel for filterpy (setup.py) ... done
  Created wheel for filterpy: filename=filterpy-1.4.5-py3-none-any.whl size
  Stored in directory: /root/.cache/pip/wheels/0f/0c/ea/218f266af4ad6268975
Successfully built filterpy
Installing collected packages: filterpy
Successfully installed filterpy-1.4.5
```

```
1 import numpy as np
2 from filterpy.kalman import KalmanFilter as FPKalmanFilter
3 import matplotlib.pyplot as plt
4 from scipy.optimize import linear_sum_assignment
5 from sklearn.metrics import silhouette_score, calinski_harabasz_score
6 from tqdm import tqdm
7 import seaborn as sns
8
9 class ComprehensiveKalmanFilter:
10     def __init__(self, dim_z=3, dt=1.0, max_cost=10.0):
11         """
12         Initialize Kalman Filter with Hungarian algorithm
13
14         Parameters:
15         -----
16         dim_z : int
17             Dimension of measurements (default 3 for x,y,z coordinates)
18         dt : float
19             Time step between measurements
20         max_cost : float
21             Maximum cost for point association
22         """
23         # Previous initialization code remains the same
24         self.dim_z = dim_z
25         dim_x = dim_z * 3
```

```

26
27     self.s_hat = np.zeros(dim_x)
28     self.P_hat = np.eye(dim_x) * 100
29     self.max_cost = max_cost
30
31     # State transition matrix
32     self.F = np.zeros((dim_x, dim_x))
33     for i in range(dim_z):
34         idx = i * 3
35         self.F[idx:idx+3, idx:idx+3] = np.array([
36             [1, dt, 0.5*dt**2],
37             [0, 1, dt],
38             [0, 0, 1]
39         ])
40
41     # Measurement matrix
42     self.H = np.zeros((dim_z, dim_x))
43     for i in range(dim_z):
44         self.H[i, i*3] = 1
45
46     # Noise matrices
47     self.Q = np.eye(dim_x) * 0.1
48     self.Q[0::3, 0::3] *= 0.1
49     self.Q[1::3, 1::3] *= 0.2
50     self.Q[2::3, 2::3] *= 0.3
51
52     self.R = np.eye(dim_z) * 0.1
53
54     # Analysis storage
55     self.K_gain = []
56     self.innovation = []
57     self.predictions = []
58     self.corrections = []
59     self.associations = []
60
61 def compute_association_cost(self, predictions, measurements):
62     """
63     Compute cost matrix for Hungarian algorithm
64
65     Parameters:
66     -----
67     predictions : array-like
68         Predicted positions (n_points, dim_z)
69     measurements : array-like
70         Measured positions (n_measurements, dim_z)
71
72     Returns:
73     -----
74     cost_matrix : array-like
75         Matrix of association costs
76     """

```

```

77     n_pred = len(predictions)
78     n_meas = len(measurements)
79
80     cost_matrix = np.zeros((n_pred, n_meas))
81
82     for i in range(n_pred):
83         for j in range(n_meas):
84             cost_matrix[i, j] = np.linalg.norm(predictions[i] - measurements[j])
85
86     return cost_matrix
87
88 def associate_points(self, predictions, measurements):
89     """
90     Associate predictions with measurements using Hungarian algorithm
91
92     Parameters:
93     -----
94     predictions : array-like
95         Predicted positions
96     measurements : array-like
97         Measured positions
98
99     Returns:
100    -----
101    associations : list of tuples
102        List of (prediction_idx, measurement_idx) pairs
103    unmatched_predictions : list
104        Indices of unmatched predictions
105    unmatched_measurements : list
106        Indices of unmatched measurements
107    """
108    cost_matrix = self.compute_association_cost(predictions, measurements)
109
110    # Apply Hungarian algorithm
111    pred_idx, meas_idx = linear_sum_assignment(cost_matrix)
112
113    # Filter associations based on maximum cost
114    valid_associations = []
115    unmatched_predictions = set(range(len(predictions)))
116    unmatched_measurements = set(range(len(measurements)))
117
118    for p, m in zip(pred_idx, meas_idx):
119        if cost_matrix[p, m] <= self.max_cost:
120            valid_associations.append((p, m))
121            unmatched_predictions.remove(p)
122            unmatched_measurements.remove(m)
123
124    return valid_associations, list(unmatched_predictions), list(unmatched_measurements)
125
126 def smooth_sequence_with_association(self, points):
127     """

```



```

128     Smooth sequence with point association
129     """
130     n_frames = len(points)
131     n_points = points.shape[1] if len(points.shape) > 2 else 1
132     smoothed = np.zeros_like(points)
133     metrics = {
134         'innovations': [],
135         'kalman_gains': [],
136         'uncertainties': [],
137         'associations': [],
138         'unmatched_ratio': []
139     }
140
141     # Initialize states for all points
142     states = [np.zeros(self.dim_z * 3) for _ in range(n_points)]
143     covs = [np.eye(self.dim_z * 3) * 100 for _ in range(n_points)]
144
145     for i in range(n_frames):
146         current_points = points[i]
147         predicted_points = np.array([state[0:3] for state in states])
148
149         # Associate points
150         associations, unmatched_pred, unmatched_meas = self.associate_points(
151             predicted_points, current_points)
152
153         metrics['associations'].append(len(associations))
154         metrics['unmatched_ratio'].append(
155             (len(unmatched_pred) + len(unmatched_meas)) / n_points)
156
157         # Update matched points
158         for pred_idx, meas_idx in associations:
159             # Prediction
160             states[pred_idx] = self.F @ states[pred_idx]
161             covs[pred_idx] = self.F @ covs[pred_idx] @ self.F.T + self.Q
162
163             # Kalman gain
164             K = covs[pred_idx] @ self.H.T @ np.linalg.inv(
165                 self.H @ covs[pred_idx] @ self.H.T + self.R)
166
167             # Update
168             innovation = current_points[meas_idx] - self.H @ states[pred_idx]
169             states[pred_idx] += K @ innovation
170             covs[pred_idx] = (np.eye(len(states[pred_idx])) -
171                             K @ self.H) @ covs[pred_idx]
172
173             smoothed[i, pred_idx] = states[pred_idx][0:3]
174
175         # Initialize new tracks for unmatched measurements
176         for meas_idx in unmatched_meas:
177             if len(unmatched_pred) > 0:
178                 pred_idx = unmatched_pred.pop(0)

```

```

179         states[pred_idx][0::3] = current_points[meas_idx]
180         states[pred_idx][3:] = 0 # Reset velocity and accelerat
181         covs[pred_idx] = np.eye(self.dim_z * 3) * 100
182         smoothed[i, pred_idx] = current_points[meas_idx]
183
184         metrics['innovations'].append(np.mean([np.linalg.norm(s[0::3] -
185                                             current_points[j]) for j, s in enumerate
186         metrics['kalman_gains'].append(np.mean([np.trace(c) for c in cov
187         metrics['uncertainties'].append(np.mean([np.trace(c) for c in cc
188
189     return smoothed, metrics
190
191 def analyze_association_performance(self, metrics):
192     """
193     Analyze point association performance
194     """
195     analysis = {
196         'average_associations': np.mean(metrics['associations']),
197         'association_stability': np.std(metrics['associations']),
198         'average_unmatched_ratio': np.mean(metrics['unmatched_ratio']),
199         'max_unmatched_ratio': np.max(metrics['unmatched_ratio'])
200     }
201     return analysis
202
203 def visualize_associations(self, original, smoothed, metrics):
204     """
205     Visualize point associations
206     """
207     fig = plt.figure(figsize=(15, 10))
208
209     # Association count
210     ax1 = plt.subplot(221)
211     ax1.plot(metrics['associations'])
212     ax1.set_title('Number of Associations per Frame')
213     ax1.set_xlabel('Frame')
214     ax1.set_ylabel('Associations')
215
216     # Unmatched ratio
217     ax2 = plt.subplot(222)
218     ax2.plot(metrics['unmatched_ratio'])
219     ax2.set_title('Unmatched Points Ratio')
220     ax2.set_xlabel('Frame')
221     ax2.set_ylabel('Ratio')
222
223     # Point trajectories
224     ax3 = plt.subplot(223, projection='3d')
225     for i in range(min(5, original.shape[1])): # Plot first 5 points
226         ax3.plot(original[:, i, 0], original[:, i, 1], original[:, i, 2]
227                 'b-', alpha=0.5, label='Original' if i == 0 else '')
228         ax3.plot(smoothed[:, i, 0], smoothed[:, i, 1], smoothed[:, i, 2]
229                 'r-', alpha=0.5, label='Smoothed' if i == 0 else '')

```

```

230     ax3.set_title('Point Trajectories')
231     ax3.legend()
232
233     # Association matrix heatmap
234     ax4 = plt.subplot(224)
235     cost_matrix = self.compute_association_cost(
236         original[-1], smoothed[-1])
237     sns.heatmap(cost_matrix, ax=ax4)
238     ax4.set_title('Final Frame Association Costs')
239
240     plt.tight_layout()
241     plt.show()
242
243 # Example usage
244 if __name__ == "__main__":
245     # Generate sample data with multiple points
246     n_frames = 100
247     n_points = 5
248     t = np.linspace(0, 2*np.pi, n_frames)
249
250     # Create trajectories for multiple points
251     original = np.zeros((n_frames, n_points, 3))
252     for i in range(n_points):
253         original[:, i] = np.column_stack([
254             np.cos(t + i*2*np.pi/n_points),
255             np.sin(t + i*2*np.pi/n_points),
256             0.1*t
257         ]) + np.random.normal(0, 0.1, (n_frames, 3))
258
259     # Initialize and apply filter
260     kf = ComprehensiveKalmanFilter(dim_z=3, max_cost=1.0)
261     smoothed, metrics = kf.smooth_sequence_with_association(original)
262
263     # Analyze performance
264     association_analysis = kf.analyze_association_performance(metrics)
265     print("\nAssociation Analysis:")
266     for k, v in association_analysis.items():
267         print(f"{k}: {v:.6f}")
268
269     # Visualize results
270     kf.visualize_associations(original, smoothed, metrics)

```



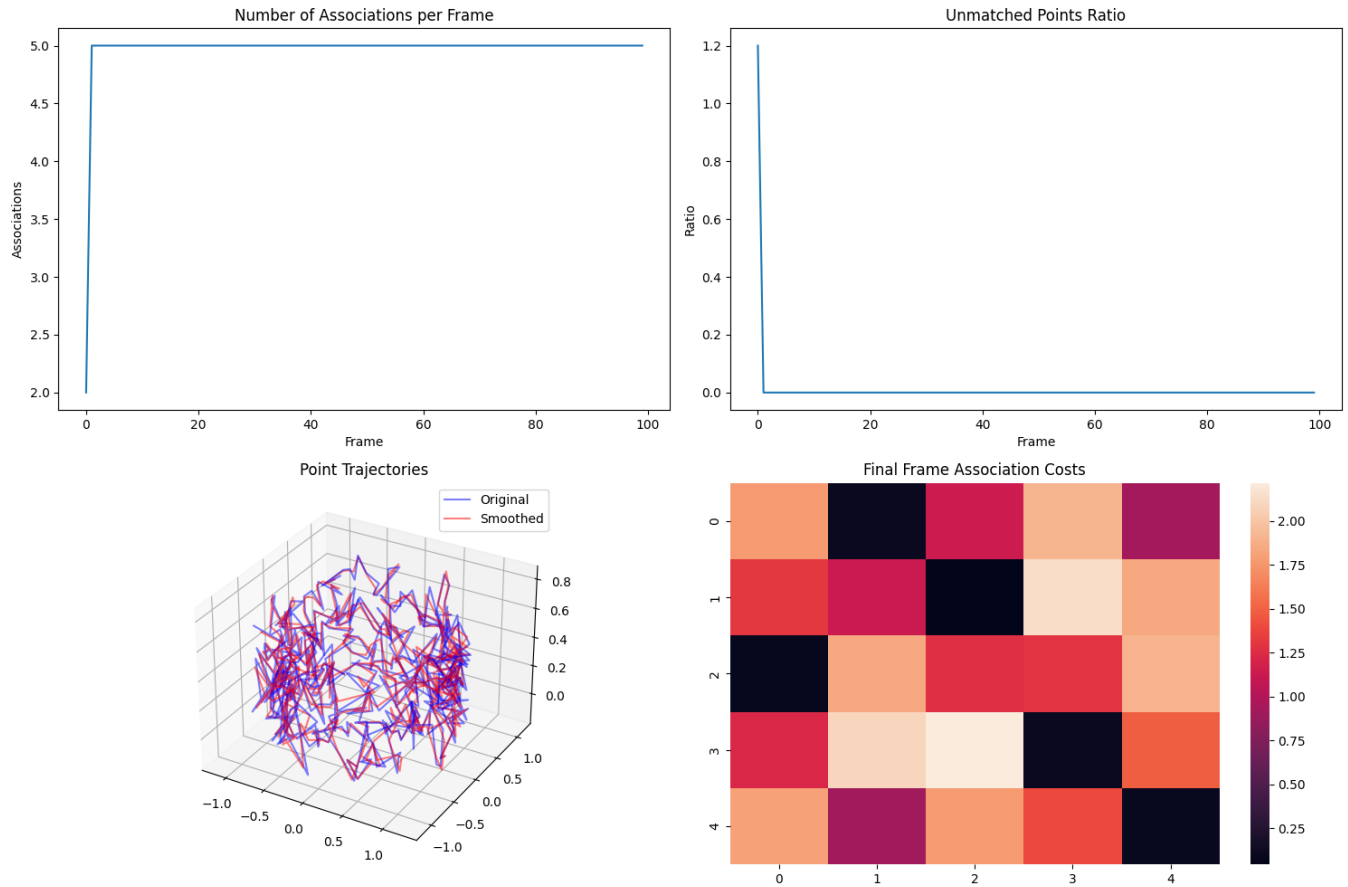
Association Analysis:

average_associations: 4.970000

association_stability: 0.298496

average_unmatched_ratio: 0.012000

max_unmatched_ratio: 1.200000



```
1 class MMRAActionDataEnhanced(MMRAActionData):
2     def __init__(self, *args, **kwargs):
3         super().__init__(*args, **kwargs)
4         self.kf = ComprehensiveKalmanFilter(dim_z=3, max_cost=1.0)
5         self.enhanced_data = None
6
7     def enhance_sequence(self, data_point):
8         """Enhance a single sequence using DBSCAN results and Kalman filterin
9         x_dbscan = data_point['new_x'] # Shape: [110, 3] (22 points * 5 stac
10
11         # Reshape to sequence format: [frames, keypoints, dims]
12         n_frames = self.stacks if self.stacks else 1
13         n_points = self.max_points
14         x_resaped = x_dbscan.reshape(n_frames, n_points, 3)
15
```

```

16         # Apply Kalman filtering with Hungarian association
17         x_enhanced, metrics = self.kf.smooth_sequence_with_association(x_res
18
19         # Store original and enhanced data
20         enhanced_point = {
21             'original': data_point['new_x'].copy(),
22             'dbscan': x_dbscan,
23             'enhanced': x_enhanced.reshape(-1, 3), # Reshape back to origina
24             'kalman_metrics': metrics
25         }
26
27         return enhanced_point
28
29     def enhance_dataset(self):
30         """Enhance entire dataset"""
31         print("Enhancing dataset with Kalman filtering...")
32         enhanced_data = []
33
34         for data_point in tqdm(self.data):
35             enhanced_point = self.enhance_sequence(data_point)
36             enhanced_data.append(enhanced_point)
37
38         self.enhanced_data = enhanced_data
39         return enhanced_data
40
41     def analyze_enhancement(self):
42         """Analyze enhancement results"""
43         if self.enhanced_data is None:
44             print("No enhanced data available. Run enhance_dataset first.")
45             return
46
47         metrics = {
48             'dbscan_metrics': [],
49             'kalman_metrics': [],
50             'combined_metrics': []
51         }
52
53         for data in self.enhanced_data:
54             # Calculate improvement metrics
55             original = data['original']
56             dbscan = data['dbscan']
57             enhanced = data['enhanced']
58
59             # DBSCAN improvement
60             dbscan_diff = np.linalg.norm(dbscan - original, axis=1)
61
62             # Kalman improvement
63             kalman_diff = np.linalg.norm(enhanced - dbscan, axis=1)
64
65             # Overall improvement
66             total_diff = np.linalg.norm(enhanced - original, axis=1)

```

```

67
68         metrics['dbscan_metrics'].append({
69             'mean_improvement': np.mean(dbscan_diff),
70             'max_improvement': np.max(dbscan_diff),
71             'std_improvement': np.std(dbscan_diff)
72         })
73
74         metrics['kalman_metrics'].append({
75             'mean_improvement': np.mean(kalman_diff),
76             'max_improvement': np.max(kalman_diff),
77             'std_improvement': np.std(kalman_diff),
78             'association_stats': data['kalman_metrics']
79         })
80
81         metrics['combined_metrics'].append({
82             'mean_improvement': np.mean(total_diff),
83             'max_improvement': np.max(total_diff),
84             'std_improvement': np.std(total_diff)
85         })
86
87     return metrics
88
89 def visualize_enhancement(self, sequence_idx=0):
90     """Visualize enhancement results for a sequence"""
91     if self.enhanced_data is None or sequence_idx >= len(self.enhanced_data):
92         print("Invalid sequence index or no enhanced data available")
93         return
94
95     data = self.enhanced_data[sequence_idx]
96
97     fig = plt.figure(figsize=(20, 10))
98
99     # 3D trajectories
100    ax1 = fig.add_subplot(231, projection='3d')
101    ax1.scatter(data['original'][:, 0],
102                data['original'][:, 1],
103                data['original'][:, 2],
104                c='b', marker='o', label='Original', alpha=0.3)
105    ax1.scatter(data['dbscan'][:, 0],
106                data['dbscan'][:, 1],
107                data['dbscan'][:, 2],
108                c='r', marker='o', label='DBSCAN', alpha=0.3)
109    ax1.scatter(data['enhanced'][:, 0],
110                data['enhanced'][:, 1],
111                data['enhanced'][:, 2],
112                c='g', marker='o', label='Enhanced', alpha=0.3)
113    ax1.set_title('3D Point Trajectories')
114    ax1.legend()
115
116    # Point associations
117    ax2 = fig.add_subplot(232)
118    associations = data['kalman_metrics']['associations']

```

```

118     associations = data['kalman_metrics']['associations']
119     ax2.plot(associations, label='Associations')
120     ax2.set_title('Point Associations per Frame')
121     ax2.set_xlabel('Frame')
122     ax2.set_ylabel('Number of Associations')
123
124     # Improvement metrics
125     ax3 = fig.add_subplot(233)
126     improvements = [
127         np.linalg.norm(data['dbscan'] - data['original'], axis=1).mean(),
128         np.linalg.norm(data['enhanced'] - data['dbscan'], axis=1).mean(),
129         np.linalg.norm(data['enhanced'] - data['original'], axis=1).mean(
130     ]
131     ax3.bar(['DBSCAN', 'Kalman', 'Combined'], improvements)
132     ax3.set_title('Average Improvements')
133
134     # Velocity profiles
135     ax4 = fig.add_subplot(234)
136     vel_orig = np.linalg.norm(np.diff(data['original'].reshape(self.stack
137     vel_enh = np.linalg.norm(np.diff(data['enhanced'].reshape(self.stacks
138     ax4.plot(vel_orig.mean(axis=0), label='Original', alpha=0.5)
139     ax4.plot(vel_enh.mean(axis=0), label='Enhanced', alpha=0.5)
140     ax4.set_title('Average Velocity Profiles')
141     ax4.legend()
142
143     # Uncertainty evolution
144     ax5 = fig.add_subplot(235)
145     ax5.plot(data['kalman_metrics']['uncertainties'])
146     ax5.set_title('State Uncertainty Evolution')
147     ax5.set_xlabel('Frame')
148     ax5.set_ylabel('Average Uncertainty')
149
150     # Innovation history
151     ax6 = fig.add_subplot(236)
152     ax6.plot(data['kalman_metrics']['innovations'])
153     ax6.set_title('Innovation History')
154     ax6.set_xlabel('Frame')
155     ax6.set_ylabel('Average Innovation')
156
157     plt.tight_layout()
158     plt.show()
159
160 # Example usage
161 if __name__ == "__main__":
162     # Initialize dataset with enhancement capabilities
163     root_dir = ''
164     mmr_dataset_config = {
165         'processed_data': '/content/drive/MyDrive/action/data/processed/mmr_a
166         'stacks': 5,
167         'max_points': 22,
168         'num_keypoints': 9,
169         'zero_padding': 'per_data_point',

```

```
170         'seed': 42,
171         'forced_rewrite': True
172     }
173
174     # Load enhanced dataset
175     dataset = MMRAActionDataEnhanced(root=root_dir, partition='train',
176                                     mmr_dataset_config=mmr_dataset_config)
177
178     # Apply enhancement pipeline
179     print("Starting enhancement pipeline...")
180     enhanced_data = dataset.enhance_dataset()
181
182     # Analyze results
183     print("\nAnalyzing enhancement results...")
184     metrics = dataset.analyze_enhancement()
185
186     # Print summary statistics
187     print("\nEnhancement Summary:")
188     print("\nDBSCAN Improvements:")
189     dbscan_means = np.mean([m['mean_improvement'] for m in metrics['dbscan_me
190     print(f"Average DBSCAN improvement: {dbscan_means:.3f}")
191
192     print("\nKalman Improvements:")
193     kalman_means = np.mean([m['mean_improvement'] for m in metrics['kalman_me
194     print(f"Average Kalman improvement: {kalman_means:.3f}")
195
196     print("\nCombined Improvements:")
197     combined_means = np.mean([m['mean_improvement'] for m in metrics['combine
198     print(f"Average total improvement: {combined_means:.3f}")
199
200     # Visualize results for first few sequences
201     print("\nVisualizing enhancement results...")
202     for i in range(5):
203         print(f"\nSequence {i+1}:")
204         dataset.visualize_enhancement(i)
```



```

➡ Transforming keypoints ...
100%|██████████| 545059/545059 [00:13<00:00, 40790.65it/s]
Transforming keypoints done
Stacking and padding frames...
100%|██████████| 212920/212920 [00:48<00:00, 4380.96it/s]
Stacking and padding frames done
Applying DBSCAN clustering...
100%|██████████| 212920/212920 [12:58<00:00, 273.58it/s]
DBSCAN clustering applied.
Transforming keypoints ...
100%|██████████| 545059/545059 [00:14<00:00, 38497.20it/s]
Transforming keypoints done
Stacking and padding frames...
100%|██████████| 212920/212920 [00:48<00:00, 4413.69it/s]
Stacking and padding frames done
Applying DBSCAN clustering...
100%|██████████| 212920/212920 [12:50<00:00, 276.44it/s]
DBSCAN clustering applied.
Starting enhancement pipeline...
Enhancing dataset with Kalman filtering...
1%|          | 1650/170336 [01:00<1:49:27, 25.68it/s]

```

```

1 # def analyze_parameter_sensitivity(self, data_point, parameter_ranges):
2 #     """Analyze sensitivity to different parameters"""
3 #     results = {
4 #         'dbscan_eps': [],
5 #         'dbscan_min_samples': [],
6 #         'kalman_q': [],
7 #         'kalman_r': []
8 #     }
9
10 #     # Test DBSCAN parameters
11 #     for eps in parameter_ranges['eps']:
12 #         for min_samples in parameter_ranges['min_samples']:
13 #             dbscan = DBSCAN(eps=eps, min_samples=min_samples)
14 #             labels = dbscan.fit_predict(data_point['new_x'])
15
16 #             metrics = {
17 #                 'n_clusters': len(set(labels)) - (1 if -1 in labels else 0),
18 #                 'noise_ratio': list(labels).count(-1) / len(labels),
19 #                 'silhouette': silhouette_score(data_point['new_x'], labels)
20 #                 if len(set(labels)) > 1 else 0
21 #             }
22
23 #             results['dbscan_eps'].append({
24 #                 'eps': eps,
25 #                 'min_samples': min_samples,
26 #                 'metrics': metrics
27 #             })
28
29 #     # Test Kalman parameters
30 #     for q in parameter_ranges['q']:

```

```

31 #         for r in parameter_ranges['r']:
32 #             kf = self.init_kalman(q=q, r=r)
33 #             smoothed = self.smooth_sequence(data_point['new_x'], kf)
34
35 #             metrics = {
36 #                 'smoothness': np.mean(np.abs(np.diff(smoothed, axis=0))),
37 #                 'tracking_error': np.mean(np.linalg.norm(
38 #                     smoothed - data_point['new_x'], axis=1))
39 #             }
40
41 #             results['kalman_q'].append({
42 #                 'q': q,
43 #                 'r': r,
44 #                 'metrics': metrics
45 #             })
46
47 #     return results
48
49 # def plot_parameter_sensitivity(self, results):
50 #     """Plot parameter sensitivity analysis"""
51 #     fig = plt.figure(figsize=(20, 10))
52
53 #     # DBSCAN parameters
54 #     ax1 = fig.add_subplot(221)
55 #     eps_values = [r['eps'] for r in results['dbscan_eps']]
56 #     noise_ratios = [r['metrics']['noise_ratio'] for r in results['dbscan_eps']]
57 #     ax1.plot(eps_values, noise_ratios)
58 #     ax1.set_title('DBSCAN eps vs Noise Ratio')
59 #     ax1.set_xlabel('eps')
60 #     ax1.set_ylabel('Noise Ratio')
61
62 #     ax2 = fig.add_subplot(222)
63 #     min_samples = [r['min_samples'] for r in results['dbscan_eps']]
64 #     n_clusters = [r['metrics']['n_clusters'] for r in results['dbscan_eps']]
65 #     ax2.plot(min_samples, n_clusters)
66 #     ax2.set_title('min_samples vs Number of Clusters')
67 #     ax2.set_xlabel('min_samples')
68 #     ax2.set_ylabel('Number of Clusters')
69
70 #     # Kalman parameters
71 #     ax3 = fig.add_subplot(223)
72 #     q_values = [r['q'] for r in results['kalman_q']]
73 #     smoothness = [r['metrics']['smoothness'] for r in results['kalman_q']]
74 #     ax3.plot(q_values, smoothness)
75 #     ax3.set_title('Process Noise (Q) vs Smoothness')
76 #     ax3.set_xlabel('Q')
77 #     ax3.set_ylabel('Smoothness')
78
79 #     ax4 = fig.add_subplot(224)
80 #     r_values = [r['r'] for r in results['kalman_q']]
81 #     tracking_error = [r['metrics']['tracking_error'] for r in results['kalman_q']]

```

```
82 # ax4.plot(r_values, tracking_error)
83 # ax4.set_title('Measurement Noise (R) vs Tracking Error')
84 # ax4.set_xlabel('R')
85 # ax4.set_ylabel('Tracking Error')
86
87 # plt.tight_layout()
```

1 Start coding or generate with AI.